eland

Release 7.6.0a3

Feb 15, 2020

Contents

Date: Feb 15, 2020 **Version**: 7.6.0a3

Useful links: Source Repository | Issues & Ideas | Q&A Support |

eland is an open source, Apache2-licensed elasticsearch Python client to analyse, explore and manipulate data that resides in elasticsearch. Where possible the package uses existing Python APIs and data structures to make it easy to switch between Numpy, Pandas, Scikit-learn to their elasticsearch powered equivalents. In general, the data resides in elasticsearch and not in memory, which allows eland to access large datasets stored in elasticsearch.

CHAPTER 1

API reference

This page gives an overview of all public eland objects, functions and methods. All classes and functions exposed in `eland.*` namespace are public.

1.1 Input/Output

1.1.1 Flat File

read_csv(filepath_or_buffer, es_client, ...)	Read a comma-separated values (csv) file into eland.DataFrame (i.e.

eland.read_csv

eland.**read_csv**(*filepath_or_buffer*, *es_client*, *es_dest_index*, *es_if_exists='fail'*, *es_refresh=False*, *es_dropna=False*, *es_geo_points=None*, *sep=','*, *delimiter=None*, *header='infer'*, *names=None*, *index_col=None*, *usecols=None*, *squeeze=False*, *prefix=None*, *mangle_dupe_cols=True*, *dtype=None*, *engine=None*, *converters=None*, *true_values=None*, *false_values=None*, *skipinitialspace=False*, *skiprows=None*, *skipfooter=0*, *nrows=None*, *chunksize=None*, *na_values=None*, *keep_default_na=True*, *na_filter=True*, *verbose=False*, *skip_blank_lines=True*, *parse_dates=False*, *infer_datetime_format=False*, *keep_date_col=False*, *date_parser=None*, *dayfirst=False*, *cache_dates=True*, *compression='infer'*, *thousands=None*, *decimal='.'*, *lineterminator=None*, *quotechar='"'*, *quoting=0*, *doublequote=True*, *escapechar=None*, *comment=None*, *encoding=None*, *dialect=None*, *error_bad_lines=True*, *warn_bad_lines=True*, *delim_whitespace=False*, *low_memory=True*, *memory_map=False*, *float_precision=None*)

Read a comma-separated values (csv) file into eland.DataFrame (i.e. an Elasticsearch index).

Modifies an Elasticsearch index

Note pandas iteration options not supported

Parameters

es_client: Elasticsearch client argument(s)

- elasticsearch-py parameters or
- elasticsearch-py instance or
- eland.Client instance

es_dest_index: str Name of Elasticsearch index to be appended to

es_if_exists [{'fail', 'replace', 'append'}, default 'fail'] How to behave if the index already exists.

- fail: Raise a ValueError.
- replace: Delete the index before inserting new values.
- append: Insert new values to the existing index. Create if does not exist.

es_dropna: bool, default 'False'

- True: Remove missing values (see pandas.Series.dropna)
- False: Include missing values - may cause bulk to fail

es_geo_points: list, default None List of columns to map to geo_point data type

chunksize number of csv rows to read before bulk index into Elasticsearch

Other Parameters

Parameters derived from :pandas_api_docs:'pandas.read_csv'.

See also:

pandas.read_csv

Notes

iterator not supported

Examples

See if 'churn' index exists in Elasticsearch

```
>>> from elasticsearch import Elasticsearch # doctest: +SKIP
>>> es = Elasticsearch() # doctest: +SKIP
>>> es.indices.exists(index="churn") # doctest: +SKIP
False
```

Read 'churn.csv' and use first column as _id (and eland.DataFrame index)

```
# churn.csv
,state,account length,area code,phone number,international plan,voice mail plan,
↪number vmail messages,total day minutes,total day calls,total day charge,total␣
↪eve minutes,total eve calls,total eve charge,total night minutes,total night␣
↪calls,total night charge,total intl minutes,total intl calls,total intl charge,
↪customer service calls,churn
0,KS,128,415,382-4657,no,yes,25,265.1,110,45.07,197.4,99,16.78,244.7,91,11.01,10.
↪0,3,2.7,1,0
```

(continues on next page)

(continued from previous page)

```
1,OH,107,415,371-7191,no,yes,26,161.6,123,27.47,195.5,103,16.62,254.4,103,11.45,
↪13.7,3,3.7,1,0
...
```

```
>>>  ed.read_csv("churn.csv",
...              es_client='localhost',
...              es_dest_index='churn',
...              es_refresh=True,
...              index_col=0) # doctest: +SKIP
          account length  area code  churn  customer service calls  ... total␣
↪night calls  total night charge total night minutes voice mail plan
0                 128        415      0                       1  ...              ␣
↪ 91               11.01               244.7             yes
1                 107        415      0                       1  ...              ␣
↪103               11.45               254.4             yes
2                 137        415      0                       0  ...              ␣
↪104                7.32               162.6              no
3                  84        408      0                       2  ...              ␣
↪ 89               8.86               196.9              no
4                  75        415      0                       3  ...              ␣
↪121                8.41               186.9              no
...               ...        ...    ...                     ...  ...              ␣
↪...                ...                 ...             ...
3328              192        415      0                       2  ...              ␣
↪ 83               12.56               279.1             yes
3329               68        415      0                       3  ...              ␣
↪123                8.61               191.3              no
3330               28        510      0                       2  ...              ␣
↪ 91               8.64               191.9              no
3331              184        510      0                       2  ...              ␣
↪137                6.26               139.2              no
3332               74        415      0                       0  ...              ␣
↪ 77               10.86               241.4             yes
<BLANKLINE>
[3333 rows x 21 columns]
```

Validate data now exists in 'churn' index:

```
>>> es.search(index="churn", size=1) # doctest: +SKIP
{'took': 1, 'timed_out': False, '_shards': {'total': 1, 'successful': 1, 'skipped
↪': 0, 'failed': 0}, 'hits': {'total': {'value': 3333, 'relation': 'eq'}, 'max_
↪score': 1.0, 'hits': [{'_index': 'churn', '_id': '0', '_score': 1.0, '_source':
↪{'state': 'KS', 'account length': 128, 'area code': 415, 'phone number': '382-
↪4657', 'international plan': 'no', 'voice mail plan': 'yes', 'number vmail␣
↪messages': 25, 'total day minutes': 265.1, 'total day calls': 110, 'total day␣
↪charge': 45.07, 'total eve minutes': 197.4, 'total eve calls': 99, 'total eve␣
↪charge': 16.78, 'total night minutes': 244.7, 'total night calls': 91, 'total␣
↪night charge': 11.01, 'total intl minutes': 10.0, 'total intl calls': 3, 'total␣
↪intl charge': 2.7, 'customer service calls': 1, 'churn': 0}}]}}
```

TODO - currently the eland.DataFrame may not retain the order of the data in the csv.

1.2 General utility functions

1.2.1 Elasticsearch access

read_es(es_client, es_index_pattern)	Utility method to create an eland.Dataframe from an Elasticsearch index_pattern.

eland.read_es

eland.**read_es**(*es_client*, *es_index_pattern*)

Utility method to create an eland.Dataframe from an Elasticsearch index_pattern. (Similar to pandas.read_csv, but source data is an Elasticsearch index rather than a csv file)

Parameters

es_client: Elasticsearch client argument(s)

- elasticsearch-py parameters or
- elasticsearch-py instance or
- eland.Client instance

es_index_pattern: str Elasticsearch index pattern

Returns

eland.DataFrame

See also:

eland.pandas_to_eland Create an eland.Dataframe from pandas.DataFrame

eland.eland_to_pandas Create a pandas.Dataframe from eland.DataFrame

1.2.2 Pandas and Eland

pandas_to_eland(pd_df, es_client, es_dest_index)	Append a pandas DataFrame to an Elasticsearch index.
eland_to_pandas(ed_df[, show_progress])	Convert an eland.Dataframe to a pandas.DataFrame

eland.pandas_to_eland

eland.**pandas_to_eland**(*pd_df*, *es_client*, *es_dest_index*, *es_if_exists='fail'*, *es_refresh=False*, *es_dropna=False*, *es_geo_points=None*, *chunksize=None*)

Append a pandas DataFrame to an Elasticsearch index. Mainly used in testing. Modifies the elasticsearch destination index

Parameters

es_client: Elasticsearch client argument(s)

- elasticsearch-py parameters or
- elasticsearch-py instance or
- eland.Client instance

es_dest_index: str Name of Elasticsearch index to be appended to

es_if_exists [{'fail', 'replace', 'append'}, default 'fail'] How to behave if the index already exists.

- fail: Raise a ValueError.
- replace: Delete the index before inserting new values.
- append: Insert new values to the existing index. Create if does not exist.

es_refresh: bool, default 'False' Refresh es_dest_index after bulk index

es_dropna: bool, default 'False'

- True: Remove missing values (see pandas.Series.dropna)
- False: Include missing values - may cause bulk to fail

es_geo_points: list, default None List of columns to map to geo_point data type

chunksize: int, default None number of pandas.DataFrame rows to read before bulk index into Elasticsearch

Returns

eland.Dataframe eland.DataFrame referencing data in destination_index

See also:

eland.read_es Create an eland.Dataframe from an Elasticsearch index

eland.eland_to_pandas Create a pandas.Dataframe from eland.DataFrame

Examples

```
>>> pd_df = pd.DataFrame(data={'A': 3.141,
...                            'B': 1,
...                            'C': 'foo',
...                            'D': pd.Timestamp('20190102'),
...                            'E': [1.0, 2.0, 3.0],
...                            'F': False,
...                            'G': [1, 2, 3]},
...                      index=['0', '1', '2'])
>>> type(pd_df)
<class 'pandas.core.frame.DataFrame'>
>>> pd_df
       A  B  ...      F  G
0  3.141  1  ...  False  1
1  3.141  1  ...  False  2
2  3.141  1  ...  False  3
<BLANKLINE>
[3 rows x 7 columns]
>>> pd_df.dtypes
A           float64
B             int64
C            object
D    datetime64[ns]
E           float64
F              bool
G             int64
dtype: object
```

Convert *pandas.DataFrame* to *eland.DataFrame* - this creates an Elasticsearch index called *pandas_to_eland*. Overwrite existing Elasticsearch index if it exists *if_exists="replace"*, and sync index so it is readable on return *refresh=True*

```
>>> ed_df = ed.pandas_to_eland(pd_df,
...                            'localhost',
...                            'pandas_to_eland',
...                            es_if_exists="replace",
...                            es_refresh=True)
>>> type(ed_df)
<class 'eland.dataframe.DataFrame'>
>>> ed_df
       A  B  ...      F  G
0  3.141  1  ...  False  1
1  3.141  1  ...  False  2
2  3.141  1  ...  False  3
<BLANKLINE>
[3 rows x 7 columns]
>>> ed_df.dtypes
A           float64
B             int64
C            object
D    datetime64[ns]
E           float64
F              bool
G             int64
dtype: object
```

eland.eland_to_pandas

eland.**eland_to_pandas**(*ed_df*, *show_progress=False*)

Convert an eland.Dataframe to a pandas.DataFrame

Note: this loads the entire Elasticsearch index into in core pandas.DataFrame structures. For large indices this can create significant load on the Elasticsearch cluster and require signficant memory

Parameters

ed_df: eland.DataFrame The source eland.Dataframe referencing the Elasticsearch index

show_progress: bool Output progress of option to stdout? By default False.

Returns

pandas.Dataframe pandas.DataFrame contains all rows and columns in eland.DataFrame

See also:

eland.read_es Create an eland.Dataframe from an Elasticsearch index

eland.pandas_to_eland Create an eland.Dataframe from pandas.DataFrame

Examples

```
>>> ed_df = ed.DataFrame('localhost', 'flights').head()
>>> type(ed_df)
<class 'eland.dataframe.DataFrame'>
```

(continues on next page)

(continued from previous page)

```
>>> ed_df
   AvgTicketPrice  Cancelled  ... dayOfWeek           timestamp
0      841.265642      False  ...         0 2018-01-01 00:00:00
1      882.982662      False  ...         0 2018-01-01 18:27:00
2      190.636904      False  ...         0 2018-01-01 17:11:14
3      181.694216       True  ...         0 2018-01-01 10:33:28
4      730.041778      False  ...         0 2018-01-01 05:13:00
<BLANKLINE>
[5 rows x 27 columns]
```

Convert *eland.DataFrame* to *pandas.DataFrame* (Note: this loads entire Elasticsearch index into core memory)

```
>>> pd_df = ed.eland_to_pandas(ed_df)
>>> type(pd_df)
<class 'pandas.core.frame.DataFrame'>
>>> pd_df
   AvgTicketPrice  Cancelled  ... dayOfWeek           timestamp
0      841.265642      False  ...         0 2018-01-01 00:00:00
1      882.982662      False  ...         0 2018-01-01 18:27:00
2      190.636904      False  ...         0 2018-01-01 17:11:14
3      181.694216       True  ...         0 2018-01-01 10:33:28
4      730.041778      False  ...         0 2018-01-01 05:13:00
<BLANKLINE>
[5 rows x 27 columns]
```

Convert *eland.DataFrame* to *pandas.DataFrame* and show progress every 10000 rows

```
>>> pd_df = ed.eland_to_pandas(ed.DataFrame('localhost', 'flights'), show_
↪progress=True) # doctest: +SKIP
2020-01-29 12:43:36.572395: read 10000 rows
2020-01-29 12:43:37.309031: read 13059 rows
```

1.3 DataFrame

1.3.1 Constructor

DataFrame([client, index_pattern, columns, ...])	Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns) referencing data stored in Elasticsearch indices.

eland.DataFrame

class eland.**DataFrame**(*client=None*, *index_pattern=None*, *columns=None*, *index_field=None*, *query_compiler=None*)

Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns) referencing data stored in Elasticsearch indices. Where possible APIs mirror pandas.DataFrame APIs. The underlying data is stored in Elasticsearch rather than core memory.

Parameters

client: Elasticsearch client argument(s) (e.g. 'localhost:9200')

- elasticsearch-py parameters or
- elasticsearch-py instance or
- eland.Client instance

index_pattern: str Elasticsearch index pattern. This can contain wildcards. (e.g. 'flights')

columns: list of str, optional List of DataFrame columns. A subset of the Elasticsearch index's fields.

index_field: str, optional The Elasticsearch index field to use as the DataFrame index. Defaults to _id if None is used.

See also:

pandas.DataFrame

Examples

Constructing DataFrame from an Elasticsearch configuration arguments and an Elasticsearch index

```
>>> df = ed.DataFrame('localhost:9200', 'flights')
>>> df.head()
   AvgTicketPrice  Cancelled  ... dayOfWeek           timestamp
0      841.265642      False  ...         0 2018-01-01 00:00:00
1      882.982662      False  ...         0 2018-01-01 18:27:00
2      190.636904      False  ...         0 2018-01-01 17:11:14
3      181.694216       True  ...         0 2018-01-01 10:33:28
4      730.041778      False  ...         0 2018-01-01 05:13:00
<BLANKLINE>
[5 rows x 27 columns]
```

Constructing DataFrame from an Elasticsearch client and an Elasticsearch index

```
>>> from elasticsearch import Elasticsearch
>>> es = Elasticsearch("localhost:9200")
>>> df = ed.DataFrame(client=es, index_pattern='flights', columns=['AvgTicketPrice
↪', 'Cancelled'])
>>> df.head()
   AvgTicketPrice  Cancelled
0      841.265642      False
1      882.982662      False
2      190.636904      False
3      181.694216       True
4      730.041778      False
<BLANKLINE>
[5 rows x 2 columns]
```

Constructing DataFrame from an Elasticsearch client and an Elasticsearch index, with 'timestamp' as the DataFrame index field (TODO - currently index_field must also be a field if not _id)

```
>>> df = ed.DataFrame(client='localhost', index_pattern='flights', columns=[
↪'AvgTicketPrice', 'timestamp'],
...                   index_field='timestamp')
>>> df.head()
                     AvgTicketPrice           timestamp
2018-01-01T00:00:00      841.265642 2018-01-01 00:00:00
```

(continues on next page)

(continued from previous page)

```
2018-01-01T00:02:06      772.100846 2018-01-01 00:02:06
2018-01-01T00:06:27      159.990962 2018-01-01 00:06:27
2018-01-01T00:33:31      800.217104 2018-01-01 00:33:31
2018-01-01T00:36:51      803.015200 2018-01-01 00:36:51
<BLANKLINE>
[5 rows x 2 columns]
```

1.3.2 Attributes and underlying data

Axes

DataFrame.index	Return eland index referencing Elasticsearch field to index a DataFrame/Series
DataFrame.columns	The column labels of the DataFrame.
DataFrame.dtypes	Return the pandas dtypes in the DataFrame.
DataFrame.select_dtypes(self[, include, exclude])	Return a subset of the DataFrame's columns based on the column dtypes.
DataFrame.values	Not implemented.
DataFrame.empty	Determines if the DataFrame is empty.
DataFrame.shape	Return a tuple representing the dimensionality of the DataFrame.

eland.DataFrame.index

DataFrame.**index**

Return eland index referencing Elasticsearch field to index a DataFrame/Series

Returns

eland.Index: Note eland.Index has a very limited API compared to pandas.Index

See also:

pandas.DataFrame.index

pandas.Series.index

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> assert isinstance(df.index, ed.Index)
>>> df.index.index_field
'_id'
>>> s = df['Carrier']
>>> assert isinstance(s.index, ed.Index)
>>> s.index.index_field
'_id'
```

eland.DataFrame.columns

DataFrame.**columns**

The column labels of the DataFrame.

Returns

pandas.Index Elasticsearch field names as pandas.Index

See also:

pandas.DataFrame.columns

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> assert isinstance(df.columns, pd.Index)
>>> df.columns
Index(['AvgTicketPrice', 'Cancelled', 'Carrier', 'Dest', 'DestAirportID',
↪'DestCityName',
...    'DestCountry', 'DestLocation', 'DestRegion', 'DestWeather',
↪'DistanceKilometers',
...    'DistanceMiles', 'FlightDelay', 'FlightDelayMin', 'FlightDelayType',
↪'FlightNum',
...    'FlightTimeHour', 'FlightTimeMin', 'Origin', 'OriginAirportID',
↪'OriginCityName',
...    'OriginCountry', 'OriginLocation', 'OriginRegion', 'OriginWeather',
↪'dayOfWeek',
...    'timestamp'],
...    dtype='object')
```

eland.DataFrame.dtypes

DataFrame.**dtypes**

Return the pandas dtypes in the DataFrame. Elasticsearch types are mapped to pandas dtypes via Mappings._es_dtype_to_pd_dtype.__doc__

Returns

pandas.Series The data type of each column.

See also:

pandas.DataFrame.dtypes

Examples

```
>>> df = ed.DataFrame('localhost', 'flights', columns=['Origin', 'AvgTicketPrice',
↪ 'timestamp', 'dayOfWeek'])
>>> df.dtypes
Origin                    object
AvgTicketPrice           float64
timestamp         datetime64[ns]
dayOfWeek                  int64
dtype: object
```

eland.DataFrame.select_dtypes

DataFrame.**select_dtypes**(*self*, *include=None*, *exclude=None*)

Return a subset of the DataFrame's columns based on the column dtypes.

Compatible with pandas.DataFrame.select_dtypes

Returns

eland.DataFrame DataFrame contains only columns of selected dtypes

Examples

```
>>> df = ed.DataFrame('localhost', 'flights',
... columns=['AvgTicketPrice', 'Dest', 'Cancelled', 'timestamp', 'dayOfWeek'])
>>> df.dtypes
AvgTicketPrice           float64
Dest                      object
Cancelled                   bool
timestamp         datetime64[ns]
dayOfWeek                  int64
dtype: object
>>> df = df.select_dtypes(include=[np.number, 'datetime'])
>>> df.dtypes
AvgTicketPrice           float64
timestamp         datetime64[ns]
dayOfWeek                  int64
dtype: object
```

eland.DataFrame.values

DataFrame.**values**

Not implemented.

In pandas this returns a Numpy representation of the DataFrame. This would involve scan/scrolling the entire index.

If this is required, call `ed.eland_to_pandas(ed_df).values`, *but beware this will scan/scroll the entire Elasticsearch index(s) into memory.*

See also:

pandas.DataFrame.values

eland_to_pandas

to_numpy

eland.DataFrame.empty

DataFrame.**empty**

Determines if the DataFrame is empty.

Returns

bool If DataFrame is empty, return True, if not return False.

See also:

pandas.DataFrame.empty

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> df.empty
False
```

eland.DataFrame.shape

`DataFrame.`**`shape`**

Return a tuple representing the dimensionality of the DataFrame.

Returns

shape: tuple

0. **number of rows**
1. **number of columns**

Notes

- number of rows `len(df)` queries Elasticsearch
- number of columns `len(df.columns)` is cached. If mappings are updated, DataFrame must be updated.

Examples

```
>>> df = ed.read_es('localhost', 'ecommerce')
>>> df.shape
(4675, 45)
```

1.3.3 Indexing, iteration

DataFrame.head(self[, n])	Return the first n rows.
DataFrame.keys(self)	Return columns
DataFrame.tail(self[, n])	Return the last n rows.
DataFrame.get(self, key[, default])	Get item from object for given key (ex: DataFrame column).
DataFrame.query(self, expr)	Query the columns of a DataFrame with a boolean expression.

eland.DataFrame.head

`DataFrame.`**`head`**(*self*, *n=5*)

Return the first n rows.

This function returns the first n rows for the object based on position. The row order is sorted by index field. It

is useful for quickly testing if your object has the right type of data in it.

Parameters

n: int, default 5 Number of rows to select.

Returns

eland.DataFrame eland DataFrame filtered on first n rows sorted by index field

See also:

pandas.DataFrame.head

Examples

```
>>> df = ed.DataFrame('localhost', 'flights', columns=['Origin', 'Dest'])
>>> df.head(3)
                           Origin                                          Dest
0        Frankfurt am Main Airport  Sydney Kingsford Smith International Airport
1  Cape Town International Airport                    Venice Marco Polo Airport
2        Venice Marco Polo Airport                    Venice Marco Polo Airport
<BLANKLINE>
[3 rows x 2 columns]
```

eland.DataFrame.keys

`DataFrame.`**`keys`**`(`*`self`*`)`

Return columns

See pandas.DataFrame.keys

Returns

pandas.Index Elasticsearch field names as pandas.Index

eland.DataFrame.tail

`DataFrame.`**`tail`**`(`*`self`*`, `*`n=5`*`)`

Return the last n rows.

This function returns the last n rows for the object based on position. The row order is sorted by index field. It is useful for quickly testing if your object has the right type of data in it.

Parameters

n: int, default 5 Number of rows to select.

Returns

eland.DataFrame: eland DataFrame filtered on last n rows sorted by index field

See also:

pandas.DataFrame.tail

Examples

```
>>> df = ed.DataFrame('localhost', 'flights', columns=['Origin', 'Dest'])
>>> df.tail()
                                                 Origin                                                 ␣
↪            Dest
13054                          Pisa International Airport      Xi'an Xianyang␣
↪International Airport
13055  Winnipeg / James Armstrong Richardson Internat...                                                 ␣
↪  Zurich Airport
13056     Licenciado Benito Juarez International Airport                                                 ␣
↪Ukrainka Air Base
13057                                       Itami Airport  Ministro Pistarini␣
↪International Airport
13058                      Adelaide International Airport   Washington Dulles␣
↪International Airport
<BLANKLINE>
[5 rows x 2 columns]
```

eland.DataFrame.get

DataFrame.**get**(*self*, *key*, *default=None*)

Get item from object for given key (ex: DataFrame column). Returns default value if not found.

Parameters

key: object

default: default value if not found

Returns

value: same type as items contained in object

See also:

pandas.DataFrame.get

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> df.get('Carrier')
0         Kibana Airlines
1        Logstash Airways
2        Logstash Airways
3         Kibana Airlines
4         Kibana Airlines
              ...
13054    Logstash Airways
13055    Logstash Airways
13056    Logstash Airways
13057            JetBeats
13058            JetBeats
Name: Carrier, Length: 13059, dtype: object
```

eland.DataFrame.query

DataFrame.**query**(*self*, *expr*)

Query the columns of a DataFrame with a boolean expression.

TODO - add additional pandas arguments

Parameters

expr: str A boolean expression

Returns

eland.DataFrame: DataFrame populated by results of the query

TODO - add link to eland user guide

See also:

pandas.DataFrame.query

Pandas User Guide/indexing

Examples

```
>>> df = ed.read_es('localhost', 'flights')
>>> df.shape
(13059, 27)
>>> df.query('FlightDelayMin > 60').shape
(2730, 27)
```

1.3.4 Function application, GroupBy & window

DataFrame.agg(self, func[, axis])	Aggregate using one or more operations over the specified axis.
DataFrame.aggregate(self, func[, axis])	Aggregate using one or more operations over the specified axis.

eland.DataFrame.agg

DataFrame.**agg**(*self*, *func*, *axis=0*, **args*, ***kwargs*)

Aggregate using one or more operations over the specified axis.

Parameters

func: function, str, list or dict Function to use for aggregating the data. If a function, must either work when passed a %(klass)s or when passed to %(klass)s.apply.

Accepted combinations are:

- function
- string function name
- list of functions and/or function names, e.g. `[np.sum, 'mean']`
- dict of axis labels -> functions, function names or list of such.

Currently, we only support `['count', 'mad', 'max', 'mean', 'median', 'min', 'mode', 'quantile', 'rank', 'sem', 'skew', 'sum', 'std', 'var']`

axis Currently, we only support axis=0 (index)

***args** Positional arguments to pass to *func*

****kwargs** Keyword arguments to pass to *func*

Returns

DataFrame, Series or scalar if DataFrame.agg is called with a single function, returns a Series if DataFrame.agg is called with several functions, returns a DataFrame if Series.agg is called with single function, returns a scalar if Series.agg is called with several functions, returns a Series

See also:

pandas.DataFrame.aggregate

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> df[['DistanceKilometers', 'AvgTicketPrice']].aggregate(['sum', 'min', 'std'])
     DistanceKilometers  AvgTicketPrice
sum        9.261629e+07    8.204365e+06
min        0.000000e+00    1.000205e+02
std        4.578263e+03    2.663867e+02
```

eland.DataFrame.aggregate

DataFrame.**aggregate**(*self*, *func*, *axis=0*, **args*, ***kwargs*)

Aggregate using one or more operations over the specified axis.

Parameters

func: function, str, list or dict Function to use for aggregating the data. If a function, must either work when passed a %(klass)s or when passed to %(klass)s.apply.

Accepted combinations are:

- function
- string function name
- list of functions and/or function names, e.g. `[np.sum, 'mean']`
- dict of axis labels -> functions, function names or list of such.

Currently, we only support `['count', 'mad', 'max', 'mean', 'median', 'min', 'mode', 'quantile', 'rank', 'sem', 'skew', 'sum', 'std', 'var']`

axis Currently, we only support axis=0 (index)

***args** Positional arguments to pass to *func*

****kwargs** Keyword arguments to pass to *func*

Returns

DataFrame, Series or scalar if DataFrame.agg is called with a single function, returns a Series if DataFrame.agg is called with several functions, returns a DataFrame if Series.agg is called with single function, returns a scalar if Series.agg is called with several functions, returns a Series

See also:

pandas.DataFrame.aggregate

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> df[['DistanceKilometers', 'AvgTicketPrice']].aggregate(['sum', 'min', 'std'])
     DistanceKilometers  AvgTicketPrice
sum        9.261629e+07    8.204365e+06
min        0.000000e+00    1.000205e+02
std        4.578263e+03    2.663867e+02
```

1.3.5 Computations / descriptive stats

DataFrame.count(self)	Count non-NA cells for each column.
DataFrame.describe(self)	Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.
DataFrame.info(self[, verbose, buf, ...])	Print a concise summary of a DataFrame.
DataFrame.max(self[, numeric_only])	Return the maximum value for each numeric column
DataFrame.mean(self[, numeric_only])	Return mean value for each numeric column
DataFrame.min(self[, numeric_only])	Return the minimum value for each numeric column
DataFrame.sum(self[, numeric_only])	Return sum for each numeric column
DataFrame.nunique(self)	Return cardinality of each field.

eland.DataFrame.count

DataFrame.**count**(*self*)

Count non-NA cells for each column.

Counts are based on exists queries against ES.

This is inefficient, as it creates N queries (N is number of fields). An alternative approach is to use value_count aggregations. However, they have issues in that:

- They can only be used with aggregatable fields (e.g. keyword not text)
- For list fields they return multiple counts. E.g. tags=['elastic', 'ml'] returns value_count=2 for a single document.

TODO - add additional pandas.DataFrame.count features

Returns

pandas.Series: Summary of column counts

See also:

pandas.DataFrame.count

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce', columns=['customer_first_name',
↪'geoip.city_name'])
>>> df.count()
customer_first_name    4675
geoip.city_name        4094
dtype: int64
```

eland.DataFrame.describe

DataFrame.**describe**(*self*)

Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.

Analyzes both numeric and object series, as well as DataFrame column sets of mixed data types. The output will vary depending on what is provided. Refer to the notes below for more detail.

TODO - add additional arguments (current only numeric values supported)

Returns

pandas.Dataframe: Summary information

See also:

pandas.DataFrame.describe

Examples

```
>>> df = ed.DataFrame('localhost', 'flights', columns=['AvgTicketPrice',
↪'FlightDelayMin'])
>>> df.describe() # ignoring percentiles as they don't generate consistent results
       AvgTicketPrice  FlightDelayMin
count    13059.000000    13059.000000
mean       628.253689       47.335171
std        266.386661       96.743006
min        100.020531        0.000000
...
...
...
max       1199.729004      360.000000
```

eland.DataFrame.info

DataFrame.**info**(*self*, *verbose=None*, *buf=None*, *max_cols=None*, *memory_usage=None*, *null_counts=None*)

Print a concise summary of a DataFrame.

This method prints information about a DataFrame including the index dtype and column dtypes, non-null values and memory usage.

See pandas.DataFrame.info for full details.

Notes

This copies a lot of code from pandas.DataFrame.info as it is difficult to split out the appropriate code or creating a SparseDataFrame gives incorrect results on types and counts.

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce', columns=['customer_first_name',
↪'geoip.city_name'])
>>> df.info()
<class 'eland.dataframe.DataFrame'>
Index: 4675 entries, 0 to 4674
Data columns (total 2 columns):
customer_first_name    4675 non-null object
geoip.city_name        4094 non-null object
dtypes: object(2)
memory usage: ...
```

eland.DataFrame.max

`DataFrame.`**`max`**(*self*, *numeric_only=True*)

Return the maximum value for each numeric column

TODO - implement remainder of pandas arguments, currently non-numerics are not supported

Returns

pandas.Series max value for each numeric column

See also:

pandas.DataFrame.max

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> df.max()
AvgTicketPrice         1199.729004
Cancelled                 1.000000
DistanceKilometers    19881.482422
DistanceMiles         12353.780273
FlightDelay               1.000000
FlightDelayMin          360.000000
FlightTimeHour           31.715034
FlightTimeMin          1902.901978
dayOfWeek                 6.000000
dtype: float64
```

eland.DataFrame.mean

`DataFrame.`**`mean`**(*self*, *numeric_only=True*)

Return mean value for each numeric column

TODO - implement remainder of pandas arguments, currently non-numerics are not supported

Returns

pandas.Series mean value for each numeric column

See also:

pandas.DataFrame.mean

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> df.mean()
AvgTicketPrice         628.253689
Cancelled                0.128494
DistanceKilometers    7092.142457
DistanceMiles         4406.853010
FlightDelay              0.251168
FlightDelayMin          47.335171
FlightTimeHour           8.518797
FlightTimeMin          511.127842
dayOfWeek                2.835975
dtype: float64
```

eland.DataFrame.min

DataFrame.**min**(*self*, *numeric_only=True*)

Return the minimum value for each numeric column

TODO - implement remainder of pandas arguments, currently non-numerics are not supported

Returns

pandas.Series min value for each numeric column

See also:

pandas.DataFrame.min

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> df.min()
AvgTicketPrice        100.020531
Cancelled               0.000000
DistanceKilometers      0.000000
DistanceMiles           0.000000
FlightDelay             0.000000
FlightDelayMin          0.000000
FlightTimeHour          0.000000
FlightTimeMin           0.000000
dayOfWeek               0.000000
dtype: float64
```

eland.DataFrame.sum

DataFrame.**sum**(*self*, *numeric_only=True*)

Return sum for each numeric column

TODO - implement remainder of pandas arguments, currently non-numerics are not supported

Returns

pandas.Series sum for each numeric column

See also:

pandas.DataFrame.sum

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> df.sum()
AvgTicketPrice        8.204365e+06
Cancelled             1.678000e+03
DistanceKilometers    9.261629e+07
DistanceMiles         5.754909e+07
FlightDelay           3.280000e+03
FlightDelayMin        6.181500e+05
FlightTimeHour        1.112470e+05
FlightTimeMin         6.674818e+06
dayOfWeek             3.703500e+04
dtype: float64
```

eland.DataFrame.nunique

DataFrame.**nunique**(*self*)

Return cardinality of each field.

Note we can only do this for aggregatable Elasticsearch fields - (in general) numeric and keyword rather than text fields

This method will try and field aggregatable fields if possible if mapping has:

```
"customer_first_name" : {
  "type" : "text",
  "fields" : {
    "keyword" : {
      "type" : "keyword",
      "ignore_above" : 256
    }
  }
}
```

we will aggregate `customer_first_name` columns using `customer_first_name.keyword`.

TODO - implement remainder of pandas arguments

Returns

pandas.Series cardinality of each column

See also:

pandas.DataFrame.nunique

Examples

```
>>> columns = ['category', 'currency', 'customer_birth_date', 'customer_first_name
↪', 'user']
>>> df = ed.DataFrame('localhost', 'ecommerce', columns=columns)
>>> df.nunique()
category                6
currency                1
customer_birth_date     0
customer_first_name    46
user                   46
dtype: int64
```

1.3.6 Reindexing / selection / label manipulation

DataFrame.drop(self[, labels, axis, index, ...])	Return new object with labels in requested axis removed.

eland.DataFrame.drop

DataFrame.**drop**(*self*, *labels=None*, *axis=0*, *index=None*, *columns=None*, *level=None*, *inplace=False*, *errors='raise'*)

Return new object with labels in requested axis removed.

Parameters

labels: Index or column labels to drop.

axis: Whether to drop labels from the index (0 / 'index') or columns (1 / 'columns').

index, columns: Alternative to specifying axis (labels, axis=1 is equivalent to columns=labels).

level: For MultiIndex - not supported

inplace: If True, do operation inplace and return None.

errors: If 'ignore', suppress error and existing labels are dropped.

Returns

dropped: type of caller

See also:

pandas.DataFrame.drop

Examples

Drop a column

```
>>> df = ed.DataFrame('localhost', 'ecommerce', columns=['customer_first_name',
↪'email', 'user'])
>>> df.drop(columns=['user'])
     customer_first_name                        email
```

(continues on next page)

(continued from previous page)

```
0                  Eddie  eddie@underwood-family.zzz
1                   Mary      mary@bailey-family.zzz
2                   Gwen      gwen@butler-family.zzz
3                  Diane   diane@chandler-family.zzz
4                  Eddie      eddie@weber-family.zzz
...                  ...                         ...
4670                Mary     mary@lambert-family.zzz
4671                 Jim      jim@gilbert-family.zzz
4672               Yahya     yahya@rivera-family.zzz
4673                Mary     mary@hampton-family.zzz
4674             Jackson  jackson@hopkins-family.zzz
<BLANKLINE>
[4675 rows x 2 columns]
```

Drop rows by index value (axis=0)

```
>>> df.drop(['1', '2'])
     customer_first_name                         email     user
0                  Eddie  eddie@underwood-family.zzz    eddie
3                  Diane   diane@chandler-family.zzz    diane
4                  Eddie      eddie@weber-family.zzz    eddie
5                  Diane    diane@goodwin-family.zzz    diane
6                 Oliver      oliver@rios-family.zzz   oliver
...                  ...                         ...      ...
4670                Mary     mary@lambert-family.zzz     mary
4671                 Jim      jim@gilbert-family.zzz      jim
4672               Yahya     yahya@rivera-family.zzz    yahya
4673                Mary     mary@hampton-family.zzz     mary
4674             Jackson  jackson@hopkins-family.zzz  jackson
<BLANKLINE>
[4673 rows x 3 columns]
```

1.3.7 Plotting

DataFrame.hist(data[, column, by, grid, ...])	Make a histogram of the DataFrame's.

eland.DataFrame.hist

DataFrame.**hist**(*data*, *column=None*, *by=None*, *grid=True*, *xlabelsize=None*, *xrot=None*, *ylabelsize=None*, *yrot=None*, *ax=None*, *sharex=False*, *sharey=False*, *figsize=None*, *layout=None*, *bins=10*, ***kwds*)

Make a histogram of the DataFrame's.

See pandas.DataFrame.hist for usage.

Notes

Derived from `pandas.plotting._core.hist_frame 0.25.3`

Ideally, we'd call the pandas method *hist_frame* directly with histogram data, but weights are applied to ALL series. For example, we can plot a histogram of pre-binned data via:

```
counts, bins = np.histogram(data)
plt.hist(bins[:-1], bins, weights=counts)
```

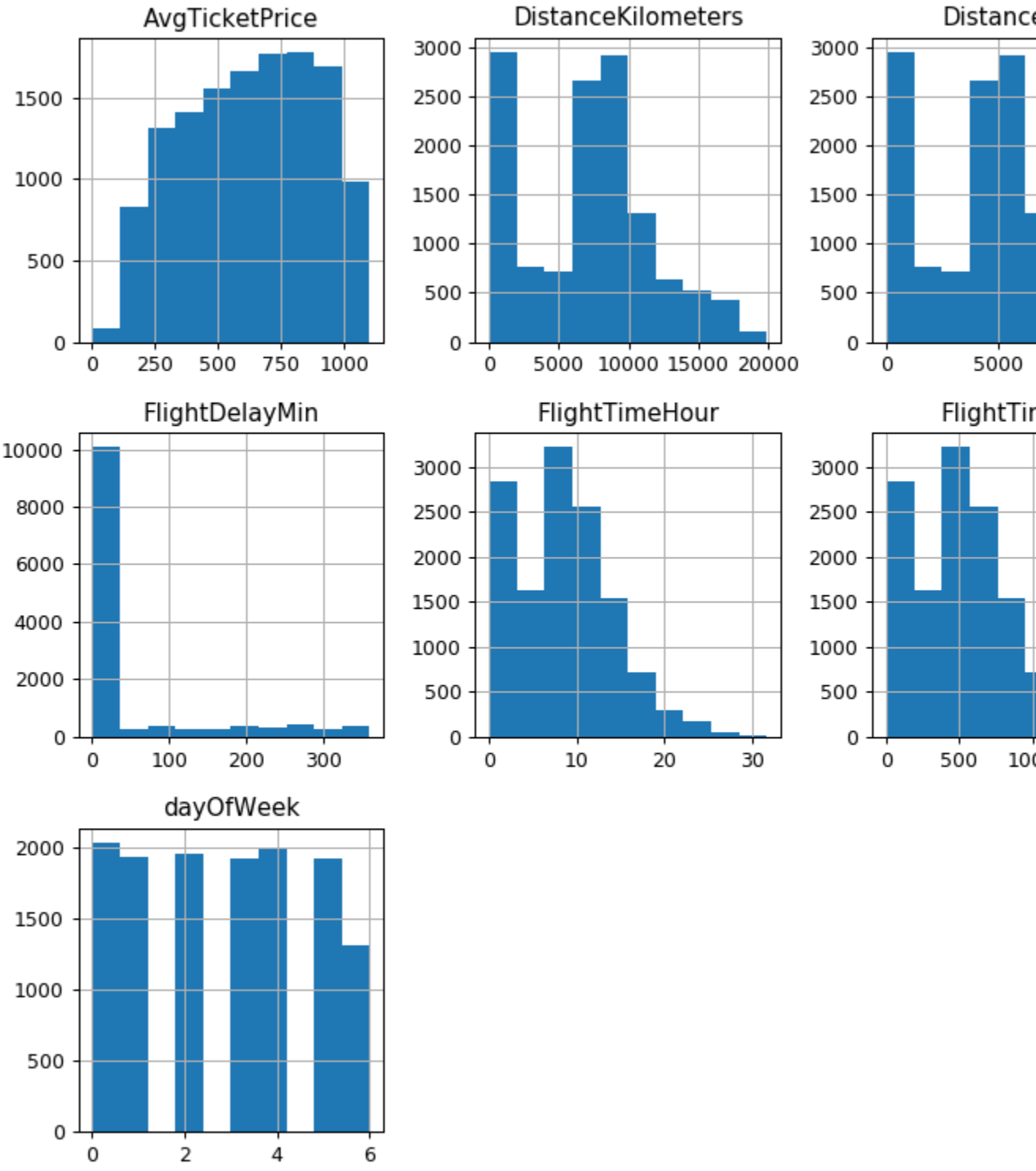
However,

```
ax.hist(data[col].dropna().values, bins=bins, **kwds)
```

is for `[col]` and weights are a single array.

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> hist = df.select_dtypes(include=[np.number]).hist(figsize=[10,10]) # doctest:␣
↪+SKIP
```


AvgTicketPrice
DistanceKilometers
Distance
FlightDelayMin
FlightTimeHour
FlightTi
dayOfWeek

1.3.8 Serialization / IO / conversion

DataFrame.info(self[, verbose, buf, ...])	Print a concise summary of a DataFrame.
DataFrame.to_numpy(self)	Not implemented.
DataFrame.to_csv(self[, path_or_buf, sep, ...])	Write Elasticsearch data to a comma-separated values (csv) file.
DataFrame.to_html(self[, buf, columns, ...])	Render a Elasticsearch data as an HTML table.
DataFrame.to_string(self[, buf, columns, ...])	Render a DataFrame to a console-friendly tabular output.

eland.DataFrame.to_numpy

DataFrame.**to_numpy**(*self*)

Not implemented.

In pandas this returns a Numpy representation of the DataFrame. This would involve scan/scrolling the entire index.

If this is required, call `ed.eland_to_pandas(ed_df).values`, *but beware this will scan/scroll the entire Elasticsearch index(s) into memory.*

See also:

pandas.DataFrame.to_numpy

eland_to_pandas

Examples

```
>>> ed_df = ed.DataFrame('localhost', 'flights', columns=['AvgTicketPrice',
↪'Carrier']).head(5)
>>> pd_df = ed.eland_to_pandas(ed_df)
>>> print("type(ed_df)={0}\ntype(pd_df)={1}".format(type(ed_df), type(pd_df)))
type(ed_df)=<class 'eland.dataframe.DataFrame'>
type(pd_df)=<class 'pandas.core.frame.DataFrame'>
>>> ed_df
   AvgTicketPrice           Carrier
0      841.265642   Kibana Airlines
1      882.982662  Logstash Airways
2      190.636904  Logstash Airways
3      181.694216   Kibana Airlines
4      730.041778   Kibana Airlines
<BLANKLINE>
[5 rows x 2 columns]
>>> pd_df.values
array([[841.2656419677076, 'Kibana Airlines'],
       [882.9826615595518, 'Logstash Airways'],
       [190.6369038508356, 'Logstash Airways'],
       [181.69421554118, 'Kibana Airlines'],
       [730.041778346198, 'Kibana Airlines']], dtype=object)
```

eland.DataFrame.to_csv

DataFrame.**to_csv**(*self*, *path_or_buf=None*, *sep=','*, *na_rep=''*, *float_format=None*, *columns=None*, *header=True*, *index=True*, *index_label=None*, *mode='w'*, *encoding=None*, *compression='infer'*, *quoting=None*, *quotechar='"'*, *line_terminator=None*, *chunksize=None*, *tupleize_cols=None*, *date_format=None*, *doublequote=True*, *escapechar=None*, *decimal='.'*)

Write Elasticsearch data to a comma-separated values (csv) file.

See also:

pandas.DataFrame.to_csv

eland.DataFrame.to_html

DataFrame.**to_html**(*self*, *buf=None*, *columns=None*, *col_space=None*, *header=True*, *index=True*, *na_rep='NaN'*, *formatters=None*, *float_format=None*, *sparsify=None*, *index_names=True*, *justify=None*, *max_rows=None*, *max_cols=None*, *show_dimensions=False*, *decimal='.'*, *bold_rows=True*, *classes=None*, *escape=True*, *notebook=False*, *border=None*, *table_id=None*, *render_links=False*)

Render a Elasticsearch data as an HTML table.

Follows pandas implementation except when `max_rows=None`. In this scenario, we set `max_rows=60` to avoid accidentally dumping an entire index. This can be overridden by explicitly setting `max_rows`.

See also:

pandas.DataFrame.to_html

eland.DataFrame.to_string

DataFrame.**to_string**(*self*, *buf=None*, *columns=None*, *col_space=None*, *header=True*, *index=True*, *na_rep='NaN'*, *formatters=None*, *float_format=None*, *sparsify=None*, *index_names=True*, *justify=None*, *max_rows=None*, *max_cols=None*, *show_dimensions=False*, *decimal='.'*, *line_width=None*)

Render a DataFrame to a console-friendly tabular output.

Follows pandas implementation except when `max_rows=None`. In this scenario, we set `max_rows=60` to avoid accidentally dumping an entire index. This can be overridden by explicitly setting `max_rows`.

See also:

pandas.DataFrame.to_string

1.3.9 Elasticsearch utilities

DataFrame.info_es(self)	A debug summary of an eland DataFrame internals.

eland.DataFrame.info_es

DataFrame.**info_es**(*self*)

A debug summary of an eland DataFrame internals.

This includes the Elasticsearch search queries and query compiler task list.

Returns

str A debug summary of an eland DataFrame internals.

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> df = df[(df.OriginAirportID == 'AMS') & (df.FlightDelayMin > 60)]
>>> df = df[['timestamp', 'OriginAirportID', 'DestAirportID', 'FlightDelayMin']]
>>> df = df.tail()
>>> df
                timestamp OriginAirportID DestAirportID  FlightDelayMin
12608 2018-02-10 01:20:52             AMS          CYEG             120
12720 2018-02-10 14:09:40             AMS           BHM             255
12725 2018-02-10 00:53:01             AMS           ATL             360
12823 2018-02-10 15:41:20             AMS           NGO             120
12907 2018-02-11 20:08:25             AMS           LIM             225
<BLANKLINE>
[5 rows x 4 columns]
>>> print(df.info_es())
index_pattern: flights
Index:
 index_field: _id
 is_source_field: False
Mappings:
 capabilities:
                  es_field_name  is_source es_dtype                  es_date_
↪format        pd_dtype  is_searchable  is_aggregatable  is_scripted␣
↪aggregatable_es_field_name
timestamp              timestamp       True     date  strict_date_hour_minute_
↪second  datetime64[ns]           True             True        False␣
↪␣
↪    timestamp
OriginAirportID  OriginAirportID       True  keyword
↪None          object           True             True        False␣
↪␣
↪OriginAirportID
DestAirportID      DestAirportID       True  keyword
↪None          object           True             True        False␣
↪␣
↪DestAirportID
FlightDelayMin    FlightDelayMin       True  integer
↪None           int64           True             True        False␣
↪␣
↪FlightDelayMin
Operations:
 tasks: [('boolean_filter': ('boolean_filter': {'bool': {'must': [{'term': {
↪'OriginAirportID': 'AMS'}}, {'range': {'FlightDelayMin': {'gt': 60}}}]}})), (
↪'tail': ('sort_field': '_doc', 'count': 5))]
 size: 5
 sort_params: _doc:desc
 _source: ['timestamp', 'OriginAirportID', 'DestAirportID', 'FlightDelayMin']
 body: {'query': {'bool': {'must': [{'term': {'OriginAirportID': 'AMS'}}, {'range
↪': {'FlightDelayMin': {'gt': 60}}}]}}}
 post_processing: [('sort_index')]
<BLANKLINE>
```

1.4 Series

1.4.1 Constructor

Series([client, index_pattern, name, ...])	pandas.Series like API that proxies into Elasticsearch index(es).

eland.Series

class eland.**Series**(*client=None*, *index_pattern=None*, *name=None*, *index_field=None*, *query_compiler=None*)

pandas.Series like API that proxies into Elasticsearch index(es).

Parameters

client [eland.Client] A reference to a Elasticsearch python client

index_pattern [str] An Elasticsearch index pattern. This can contain wildcards.

index_field [str] The field to base the series on

See also:

pandas.Series

Notes

If the Elasticsearch index is deleted or index mappings are changed after this object is created, the object is not rebuilt and so inconsistencies can occur.

Examples

```
>>> ed.Series(client='localhost', index_pattern='flights', name='Carrier')
0         Kibana Airlines
1        Logstash Airways
2        Logstash Airways
3         Kibana Airlines
4         Kibana Airlines
              ...
13054    Logstash Airways
13055    Logstash Airways
13056    Logstash Airways
13057            JetBeats
13058            JetBeats
Name: Carrier, Length: 13059, dtype: object
```

1.4.2 Attributes and underlying data

Axes

Series.index	Return eland index referencing Elasticsearch field to index a DataFrame/Series
Series.shape	Return a tuple representing the dimensionality of the Series.
Series.name	
Series.empty	Determines if the Series is empty.

eland.Series.index

Series.**index**

Return eland index referencing Elasticsearch field to index a DataFrame/Series

Returns

eland.Index: Note eland.Index has a very limited API compared to pandas.Index

See also:

pandas.DataFrame.index

pandas.Series.index

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> assert isinstance(df.index, ed.Index)
>>> df.index.index_field
'_id'
>>> s = df['Carrier']
>>> assert isinstance(s.index, ed.Index)
>>> s.index.index_field
'_id'
```

eland.Series.shape

Series.**shape**

Return a tuple representing the dimensionality of the Series.

Returns

shape: tuple

0. **number of rows**
1. **number of columns**

Notes

- number of rows `len(series)` queries Elasticsearch
- number of columns == 1

Examples

```
>>> df = ed.Series('localhost', 'ecommerce', name='total_quantity')
>>> df.shape
(4675, 1)
```

eland.Series.name

Series.**name**

eland.Series.empty

Series.**empty**

Determines if the Series is empty.

Returns: True if the Series is empty. False otherwise.

1.4.3 Indexing, iteration

Series.head(self[, n])	
Series.tail(self[, n])	

eland.Series.head

Series.**head**(*self*, *n=5*)

eland.Series.tail

Series.**tail**(*self*, *n=5*)

1.4.4 Binary operator functions

Series.add(self, right)	Return addition of series and right, element-wise (binary operator add).
Series.sub(self, right)	Return subtraction of series and right, element-wise (binary operator sub).
Series.mul(self, right)	Return multiplication of series and right, element-wise (binary operator mul).
Series.div(self, right)	Return floating division of series and right, element-wise (binary operator truediv).
Series.truediv(self, right)	Return floating division of series and right, element-wise (binary operator truediv).
Series.floordiv(self, right)	Return integer division of series and right, element-wise (binary operator floordiv //).
Series.mod(self, right)	Return modulo of series and right, element-wise (binary operator mod %).

Continued on next page

Table 16 – continued from previous page

Series.pow(self, right)	Return exponential power of series and right, element-wise (binary operator pow).
Series.radd(self, left)	Return addition of series and left, element-wise (binary operator add).
Series.rsub(self, left)	Return subtraction of series and left, element-wise (binary operator sub).
Series.rmul(self, left)	Return multiplication of series and left, element-wise (binary operator mul).
Series.rdiv(self, left)	Return division of series and left, element-wise (binary operator div).
Series.rtruediv(self, left)	Return division of series and left, element-wise (binary operator div).
Series.rfloordiv(self, left)	Return integer division of series and left, element-wise (binary operator floordiv //).
Series.rmod(self, left)	Return modulo of series and left, element-wise (binary operator mod %).
Series.rpow(self, left)	Return exponential power of series and left, element-wise (binary operator pow).

eland.Series.add

Series.**add**(*self*, *right*)

Return addition of series and right, element-wise (binary operator add).

Parameters

right: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> df.taxful_total_price + 1
0     37.980000
1     54.980000
2    200.979996
3    175.979996
4     81.980003
Name: taxful_total_price, dtype: float64
>>> df.total_quantity
0    2
1    2
2    2
3    2
```

(continues on next page)

(continued from previous page)

```
4    2
Name: total_quantity, dtype: int64
>>> df.taxful_total_price + df.total_quantity
0     38.980000
1     55.980000
2    201.979996
3    176.979996
4     82.980003
dtype: float64
>>> df.customer_first_name + df.customer_last_name
0    EddieUnderwood
1        MaryBailey
2        GwenButler
3     DianeChandler
4        EddieWeber
dtype: object
>>> "First name: " + df.customer_first_name
0    First name: Eddie
1     First name: Mary
2     First name: Gwen
3    First name: Diane
4    First name: Eddie
Name: customer_first_name, dtype: object
```

eland.Series.sub

`Series.`**`sub`**(*self*, *right*)

Return subtraction of series and right, element-wise (binary operator sub).

Parameters

right: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> df.total_quantity
0    2
1    2
2    2
3    2
4    2
Name: total_quantity, dtype: int64
>>> df.taxful_total_price - df.total_quantity
```

(continues on next page)

(continued from previous page)

```
0     34.980000
1     51.980000
2    197.979996
3    172.979996
4     78.980003
dtype: float64
```

eland.Series.mul

`Series.`**`mul`**(*self*, *right*)

Return multiplication of series and right, element-wise (binary operator mul).

Parameters

right: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> df.total_quantity
0    2
1    2
2    2
3    2
4    2
Name: total_quantity, dtype: int64
>>> df.taxful_total_price * df.total_quantity
0     73.959999
1    107.959999
2    399.959991
3    349.959991
4    161.960007
dtype: float64
```

eland.Series.div

`Series.`**`div`**(*self*, *right*)

Return floating division of series and right, element-wise (binary operator truediv).

Parameters

right: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> df.total_quantity
0    2
1    2
2    2
3    2
4    2
Name: total_quantity, dtype: int64
>>> df.taxful_total_price / df.total_quantity
0    18.490000
1    26.990000
2    99.989998
3    87.489998
4    40.490002
dtype: float64
```

eland.Series.truediv

`Series.`**`truediv`**(*self*, *right*)

Return floating division of series and right, element-wise (binary operator truediv).

Parameters

right: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> df.total_quantity
0    2
1    2
2    2
3    2
```

(continues on next page)

(continued from previous page)

```
4    2
Name: total_quantity, dtype: int64
>>> df.taxful_total_price / df.total_quantity
0    18.490000
1    26.990000
2    99.989998
3    87.489998
4    40.490002
dtype: float64
```

eland.Series.floordiv

`Series.`**`floordiv`**(*self*, *right*)

Return integer division of series and right, element-wise (binary operator floordiv //).

Parameters

right: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> df.total_quantity
0    2
1    2
2    2
3    2
4    2
Name: total_quantity, dtype: int64
>>> df.taxful_total_price // df.total_quantity
0    18.0
1    26.0
2    99.0
3    87.0
4    40.0
dtype: float64
```

eland.Series.mod

`Series.`**`mod`**(*self*, *right*)

Return modulo of series and right, element-wise (binary operator mod %).

Parameters

right: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> df.total_quantity
0    2
1    2
2    2
3    2
4    2
Name: total_quantity, dtype: int64
>>> df.taxful_total_price % df.total_quantity
0    0.980000
1    1.980000
2    1.979996
3    0.979996
4    0.980003
dtype: float64
```

eland.Series.pow

`Series.`**`pow`**(*self*, *right*)

Return exponential power of series and right, element-wise (binary operator pow).

Parameters

right: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> df.total_quantity
0    2
```

(continues on next page)

(continued from previous page)

```
1    2
2    2
3    2
4    2
Name: total_quantity, dtype: int64
>>> df.taxful_total_price ** df.total_quantity
0     1367.520366
1     2913.840351
2    39991.998691
3    30617.998905
4     6557.760944
dtype: float64
```

eland.Series.radd

`Series.`**`radd`**(*self*, *left*)

Return addition of series and left, element-wise (binary operator add).

Parameters

left: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> 1 + df.taxful_total_price
0     37.980000
1     54.980000
2    200.979996
3    175.979996
4     81.980003
Name: taxful_total_price, dtype: float64
```

eland.Series.rsub

`Series.`**`rsub`**(*self*, *left*)

Return subtraction of series and left, element-wise (binary operator sub).

Parameters

left: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> 1.0 - df.taxful_total_price
0    -35.980000
1    -52.980000
2   -198.979996
3   -173.979996
4    -79.980003
Name: taxful_total_price, dtype: float64
```

eland.Series.rmul

`Series.`**`rmul`**(*self*, *left*)

Return multiplication of series and left, element-wise (binary operator mul).

Parameters

left: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> 10.0 * df.taxful_total_price
0     369.799995
1     539.799995
2    1999.799957
3    1749.799957
4     809.800034
Name: taxful_total_price, dtype: float64
```

eland.Series.rdiv

`Series.`**`rdiv`**(*self*, *left*)

Return division of series and left, element-wise (binary operator div).

Parameters

left: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> 1.0 / df.taxful_total_price
0    0.027042
1    0.018525
2    0.005001
3    0.005715
4    0.012349
Name: taxful_total_price, dtype: float64
```

eland.Series.rtruediv

`Series.`**`rtruediv`**(*self*, *left*)

Return division of series and left, element-wise (binary operator div).

Parameters

left: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> 1.0 / df.taxful_total_price
0    0.027042
1    0.018525
2    0.005001
3    0.005715
4    0.012349
Name: taxful_total_price, dtype: float64
```

eland.Series.rfloordiv

`Series.`**`rfloordiv`**(*self*, *left*)

Return integer division of series and left, element-wise (binary operator floordiv //).

Parameters

left: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> 500.0 // df.taxful_total_price
0    13.0
1     9.0
2     2.0
3     2.0
4     6.0
Name: taxful_total_price, dtype: float64
```

eland.Series.rmod

`Series.`**`rmod`**(*self*, *left*)

Return modulo of series and left, element-wise (binary operator mod %).

Parameters

left: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.taxful_total_price
0     36.98
1     53.98
2    199.98
3    174.98
4     80.98
Name: taxful_total_price, dtype: float64
>>> 500.0 % df.taxful_total_price
0     19.260006
```

(continues on next page)

(continued from previous page)

```
1     14.180004
2    100.040009
3    150.040009
4     14.119980
Name: taxful_total_price, dtype: float64
```

eland.Series.rpow

`Series.`**`rpow`**(*self*, *left*)

Return exponential power of series and left, element-wise (binary operator pow).

Parameters

left: eland.Series

Returns

eland.Series

Examples

```
>>> df = ed.DataFrame('localhost', 'ecommerce').head(5)
>>> df.total_quantity
0    2
1    2
2    2
3    2
4    2
Name: total_quantity, dtype: int64
>>> np.int(2) ** df.total_quantity
0    4.0
1    4.0
2    4.0
3    4.0
4    4.0
Name: total_quantity, dtype: float64
```

1.4.5 Computations / descriptive stats

Series.describe(self)	Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.
Series.max(self[, numeric_only])	Return the maximum of the Series values
Series.mean(self[, numeric_only])	Return the mean of the Series values
Series.min(self[, numeric_only])	Return the minimum of the Series values
Series.sum(self[, numeric_only])	Return the sum of the Series values
Series.nunique(self)	Return the sum of the Series values
Series.value_counts(self[, es_size])	Return the value counts for the specified field.

eland.Series.describe

Series.**describe**(*self*)

Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.

Analyzes both numeric and object series, as well as DataFrame column sets of mixed data types. The output will vary depending on what is provided. Refer to the notes below for more detail.

TODO - add additional arguments (current only numeric values supported)

Returns

pandas.Dataframe: Summary information

See also:

pandas.DataFrame.describe

Examples

```
>>> df = ed.DataFrame('localhost', 'flights', columns=['AvgTicketPrice',
↪'FlightDelayMin'])
>>> df.describe() # ignoring percentiles as they don't generate consistent results
       AvgTicketPrice  FlightDelayMin
count    13059.000000    13059.000000
mean       628.253689       47.335171
std        266.386661       96.743006
min        100.020531        0.000000
...
...
...
max       1199.729004      360.000000
```

eland.Series.max

Series.**max**(*self*, *numeric_only=None*)

Return the maximum of the Series values

TODO - implement remainder of pandas arguments, currently non-numerics are not supported

Returns

float max value

See also:

pandas.Series.max

Examples

```
>>> s = ed.Series('localhost', 'flights', name='AvgTicketPrice')
>>> int(s.max())
1199
```

eland.Series.mean

`Series.`**`mean`**(*self*, *numeric_only=None*)

Return the mean of the Series values

TODO - implement remainder of pandas arguments, currently non-numerics are not supported

Returns

float max value

See also:

pandas.Series.mean

Examples

```
>>> s = ed.Series('localhost', 'flights', name='AvgTicketPrice')
>>> int(s.mean())
628
```

eland.Series.min

`Series.`**`min`**(*self*, *numeric_only=None*)

Return the minimum of the Series values

TODO - implement remainder of pandas arguments, currently non-numerics are not supported

Returns

float max value

See also:

pandas.Series.min

Examples

```
>>> s = ed.Series('localhost', 'flights', name='AvgTicketPrice')
>>> int(s.min())
100
```

eland.Series.sum

`Series.`**`sum`**(*self*, *numeric_only=None*)

Return the sum of the Series values

TODO - implement remainder of pandas arguments, currently non-numerics are not supported

Returns

float max value

See also:

pandas.Series.sum

Examples

```
>>> s = ed.Series('localhost', 'flights', name='AvgTicketPrice')
>>> int(s.sum())
8204364
```

eland.Series.nunique

Series.**nunique**(*self*)

Return the sum of the Series values

Returns

float max value

See also:

pandas.Series.sum

Examples

```
>>> s = ed.Series('localhost', 'flights', name='Carrier')
>>> s.nunique()
4
```

eland.Series.value_counts

Series.**value_counts**(*self*, *es_size=10*)

Return the value counts for the specified field.

Note we can only do this for aggregatable Elasticsearch fields - (in general) numeric and keyword rather than text fields

TODO - implement remainder of pandas arguments

Parameters

es_size: int, default 10 Number of buckets to return counts for, automatically sorts by count descending. This parameter is specific to *eland*, and determines how many term buckets elasticsearch should return out of the overall terms list.

Returns

pandas.Series number of occurrences of each value in the column

See also:

pandas.Series.value_counts

search-aggregations-bucket-terms-aggregation

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> df['Carrier'].value_counts()
Logstash Airways    3331
JetBeats            3274
Kibana Airlines     3234
ES-Air              3220
Name: Carrier, dtype: int64
```

1.4.6 Reindexing / selection / label manipulation

Series.rename(self, new_name)	Rename name of series.

eland.Series.rename

Series.**rename**(*self*, *new_name*)

Rename name of series. Only column rename is supported. This does not change the underlying Elasticsearch index, but adds a symbolic link from the new name (column) to the Elasticsearch field name.

For instance, if a field was called 'total_quantity' it could be renamed 'Total Quantity'.

Parameters

new_name: str

Returns

eland.Series eland.Series with new name.

See also:

pandas.Series.rename

Examples

```
>>> df = ed.DataFrame('localhost', 'flights')
>>> df.Carrier
0         Kibana Airlines
1        Logstash Airways
2        Logstash Airways
3         Kibana Airlines
4         Kibana Airlines
              ...
13054    Logstash Airways
13055    Logstash Airways
13056    Logstash Airways
13057            JetBeats
13058            JetBeats
Name: Carrier, Length: 13059, dtype: object
>>> df.Carrier.rename('Airline')
0         Kibana Airlines
1        Logstash Airways
2        Logstash Airways
3         Kibana Airlines
4         Kibana Airlines
```

(continues on next page)

(continued from previous page)

```
              ...
13054    Logstash Airways
13055    Logstash Airways
13056    Logstash Airways
13057            JetBeats
13058            JetBeats
Name: Airline, Length: 13059, dtype: object
```

1.4.7 Plotting

Series.hist(self[, by, ax, grid, ...])	Draw histogram of the input series using matplotlib.

eland.Series.hist

Series.**hist**(*self*, *by=None*, *ax=None*, *grid=True*, *xlabelsize=None*, *xrot=None*, *ylabelsize=None*, *yrot=None*, *figsize=None*, *bins=10*, ***kwds*)

Draw histogram of the input series using matplotlib.

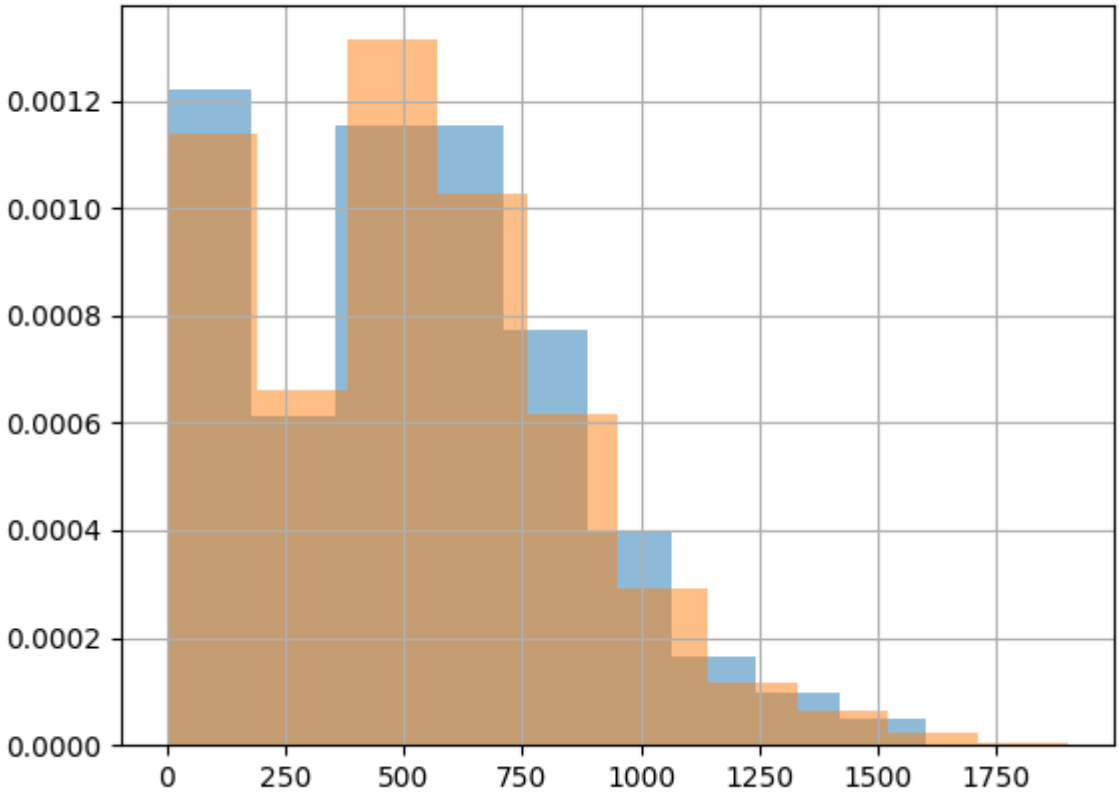
See pandas.Series.hist for usage.

Notes

Derived from `pandas.plotting._core.hist_frame 0.25.3`

Examples

```
>>> import matplotlib.pyplot as plt
>>> df = ed.DataFrame('localhost', 'flights')
>>> df[df.OriginWeather == 'Sunny']['FlightTimeMin'].hist(alpha=0.5,␣
↪density=True) # doctest: +SKIP
>>> df[df.OriginWeather != 'Sunny']['FlightTimeMin'].hist(alpha=0.5,␣
↪density=True) # doctest: +SKIP
>>> plt.show() # doctest: +SKIP
```



1.4.8 Serialization / IO / conversion

Series.to_string(self[, buf, na_rep, ...])	Render a string representation of the Series.
Series.to_numpy(self)	Not implemented.

eland.Series.to_string

Series.**to_string**(*self*, *buf=None*, *na_rep='NaN'*, *float_format=None*, *header=True*, *index=True*, *length=False*, *dtype=False*, *name=False*, *max_rows=None*, *min_rows=None*)

Render a string representation of the Series.

Follows pandas implementation except when `max_rows=None`. In this scenario, we set `max_rows=60` to avoid accidentally dumping an entire index. This can be overridden by explicitly setting `max_rows`.

See also:

pandas.Series.to_string for argument details.

eland.Series.to_numpy

Series.**to_numpy**(*self*)

Not implemented.

In pandas this returns a Numpy representation of the Series. This would involve scan/scrolling the entire index.

If this is required, call `ed.eland_to_pandas(ed_series).values`, *but beware this will scan/scroll the entire Elasticsearch index(s) into memory.*

See also:

pandas.DataFrame.to_numpy

eland_to_pandas

Examples

```
>>> ed_s = ed.Series('localhost', 'flights', name='Carrier').head(5)
>>> pd_s = ed.eland_to_pandas(ed_s)
>>> print("type(ed_s)={0}\ntype(pd_s)={1}".format(type(ed_s), type(pd_s)))
type(ed_s)=<class 'eland.series.Series'>
type(pd_s)=<class 'pandas.core.series.Series'>
>>> ed_s
0     Kibana Airlines
1    Logstash Airways
2    Logstash Airways
3     Kibana Airlines
4     Kibana Airlines
Name: Carrier, dtype: object
>>> pd_s.to_numpy()
array(['Kibana Airlines', 'Logstash Airways', 'Logstash Airways',
       'Kibana Airlines', 'Kibana Airlines'], dtype=object)
```

1.4.9 Elasticsearch utilities

Series.info_es(self)

eland.Series.info_es

Series.**info_es**(*self*)

1.5 Index

Many of these methods or variants thereof are available on the objects that contain an index (Series/DataFrame) and those should most likely be used before calling these methods directly.

1.5.1 Constructor

Index(query_compiler[, index_field])	The index for an eland.DataFrame.

eland.Index

class eland.**Index**(*query_compiler*, *index_field=None*)

The index for an eland.DataFrame.

TODO - This currently has very different behaviour than pandas.Index

Currently, the index is a field that exists in every document in an Elasticsearch index. For slicing and sorting operations it must be a docvalues field. By default _id is used, which can't be used for range queries and is inefficient for sorting:

https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-id-field.html (The value of the _id field is also accessible in aggregations or for sorting, but doing so is discouraged as it requires to load a lot of data in memory. In case sorting or aggregating on the _id field is required, it is advised to duplicate the content of the _id field in another field that has doc_values enabled.)

1.6 Machine Learning

Machine learning is built into the Elastic Stack and enables users to gain insights into their Elasticsearch data. There are a wide range of capabilities from identifying in anomalies in your data, to training and deploying regression or classification models based on Elasticsearch data.

To use the Elastic Stack machine learning features, you must have the appropriate license and at least one machine learning node in your Elasticsearch cluster. If Elastic Stack security features are enabled, you must also ensure your users have the necessary privileges.

The fastest way to get started with machine learning features is to start a free 14-day trial of Elasticsearch Service in the cloud.

See https://www.elastic.co/guide/en/machine-learning/current/setup.html and other documentation for more detail.

1.6.1 ImportedMLModel

Constructor

ImportedMLModel(es_client, model_id, model, ...)	Transform and serialize a trained 3rd party model into Elasticsearch.

eland.ml.ImportedMLModel

class eland.ml.**ImportedMLModel**(*es_client*, *model_id: str*, *model: Union[sklearn.tree._classes.DecisionTreeClassifier, sklearn.tree._classes.DecisionTreeRegressor, sklearn.ensemble._forest.RandomForestRegressor, sklearn.ensemble._forest.RandomForestClassifier, xgboost.sklearn.XGBClassifier, xgboost.sklearn.XGBRegressor]*, *feature_names: List[str]*, *classification_labels: List[str] = None*, *classification_weights: List[float] = None*, *overwrite=False*)

Transform and serialize a trained 3rd party model into Elasticsearch. This model can then be used for inference in the Elastic Stack.

Parameters

es_client: Elasticsearch client argument(s)

- elasticsearch-py parameters or
- elasticsearch-py instance or
- eland.Client instance

model_id: str The unique identifier of the trained inference model in Elasticsearch.

model: An instance of a supported python model. We support the following model types:

- sklearn.tree.DecisionTreeClassifier
- sklearn.tree.DecisionTreeRegressor
- sklearn.ensemble.RandomForestRegressor
- sklearn.ensemble.RandomForestClassifier
- xgboost.XGBClassifier
- xgboost.XGBRegressor

feature_names: List[str] Names of the features (required)

classification_labels: List[str] Labels of the classification targets

classification_weights: List[str] Weights of the classification targets

overwrite: bool Delete and overwrite existing model (if exists)

Examples

```
>>> from sklearn import datasets
>>> from sklearn.tree import DecisionTreeClassifier
>>> from eland.ml import ImportedMLModel
```

```
>>> # Train model
>>> training_data = datasets.make_classification(n_features=5, random_state=0)
>>> test_data = [[-50.1, 0.2, 0.3, -0.5, 1.0], [1.6, 2.1, -10, 50, -1.0]]
>>> classifier = DecisionTreeClassifier()
>>> classifier = classifier.fit(training_data[0], training_data[1])
```

```
>>> # Get some test results
>>> classifier.predict(test_data)
array([0, 1])
```

```
>>> # Serialise the model to Elasticsearch
>>> feature_names = ["f0", "f1", "f2", "f3", "f4"]
>>> model_id = "test_decision_tree_classifier"
>>> es_model = ImportedMLModel('localhost', model_id, classifier, feature_names,␣
↪overwrite=True)
```

```
>>> # Get some test results from Elasticsearch model
>>> es_model.predict(test_data)
array([0, 1])
```

```
>>> # Delete model from Elasticsearch
>>> es_model.delete_model()
```

Learning API

ImportedMLModel.predict(self, X)	Make a prediction using a trained model stored in Elasticsearch.

eland.ml.ImportedMLModel.predict

ImportedMLModel.**predict**(*self*, *X*)

Make a prediction using a trained model stored in Elasticsearch.

Parameters for this method are not yet fully compatible with standard sklearn.predict.

Parameters

X: list or list of lists of type float Input feature vector - TODO support DataFrame and other formats

Returns

y: np.ndarray of dtype float for regressors or int for classifiers

Examples

```
>>> from sklearn import datasets
>>> from xgboost import XGBRegressor
>>> from eland.ml import ImportedMLModel
```

```
>>> # Train model
>>> training_data = datasets.make_classification(n_features=6, random_state=0)
>>> test_data = [[-1, -2, -3, -4, -5, -6], [10, 20, 30, 40, 50, 60]]
>>> regressor = XGBRegressor(objective='reg:squarederror')
>>> regressor = regressor.fit(training_data[0], training_data[1])
```

```
>>> # Get some test results
>>> regressor.predict(np.array(test_data))
array([0.23733574, 1.1897984 ], dtype=float32)
```

```
>>> # Serialise the model to Elasticsearch
>>> feature_names = ["f0", "f1", "f2", "f3", "f4", "f5"]
>>> model_id = "test_xgb_regressor"
>>> es_model = ImportedMLModel('localhost', model_id, regressor, feature_names,␣
↪overwrite=True)
```

```
>>> # Get some test results from Elasticsearch model
>>> es_model.predict(test_data)
array([0.2373357, 1.1897984], dtype=float32)
```

```
>>> # Delete model from Elasticsearch
>>> es_model.delete_model()
```

CHAPTER 2

Implementation Notes

2.1 Implementation Details

The goal of an `eland.DataFrame` is to enable users who are familiar with `pandas.DataFrame` to access, explore and manipulate data that resides in Elasticsearch.

Ideally, all data should reside in Elasticsearch and not to reside in memory. This restricts the API, but allows access to huge data sets that do not fit into memory, and allows use of powerful Elasticsearch features such as aggregations.

2.1.1 Pandas and 3rd Party Storage Systems

Generally, integrations with [3rd party storage systems](https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html) (SQL, Google Big Query etc.) involve accessing these systems and reading all external data into an in-core pandas data structure. This also applies to [Apache Arrow](https://arrow.apache.org/docs/python/pandas.html) structures.

Whilst this provides access to data in these systems, for large datasets this can require significant in-core memory, and for systems such as Elasticsearch, bulk export of data can be an inefficient way of exploring the data.

An alternative option is to create an API that proxies `pandas.DataFrame`-like calls to Elasticsearch queries and operations. This could allow the Elasticsearch cluster to perform operations such as aggregations rather than exporting all the data and performing this operation in-core.

2.1.2 Implementation Options

An option would be to replace the `pandas.DataFrame` backend in-core memory structures with Elasticsearch accessors. This would allow full access to the `pandas.DataFrame` APIs. However, this has issues:

- If a `pandas.DataFrame` instance maps to an index, typical manipulation of a `pandas.DataFrame` may involve creating many derived `pandas.DataFrame` instances. Constructing an index per `pandas.DataFrame` may result in many Elasticsearch indexes and a significant load on Elasticsearch. For example, `df_a = df['a']` should not require Elasticsearch indices `df` and `df_a`

- Not all `pandas.DataFrame` APIs map to things we may want to do in Elasticsearch. In particular, API calls that involve exporting all data from Elasticsearch into memory e.g. `df.to_dict()`.
- The backend `pandas.DataFrame` structures are not easily abstractable and are deeply embedded in the implementation.

Another option is to create a `eland.DataFrame` API that mimics appropriate aspects of the `pandas.DataFrame` API. This resolves some of the issues above as:

- `df_a = df['a']` could be implemented as a change to the Elasticsearch query used, rather than a new index
- Instead of supporting the enitre `pandas.DataFrame` API we can support a subset appropriate for Elasticsearch. If addition calls are required, we could to create a `eland.DataFrame._to_pandas()` method which would explicitly export all data to a `pandas.DataFrame`
- Creating a new `eland.DataFrame` API gives us full flexibility in terms of implementation. However, it does create a large amount of work which may duplicate a lot of the `pandas` code - for example, printing objects etc. - this creates maintenance issues etc.

2.2 pandas.DataFrame supported APIs

The following table lists both implemented and not implemented methods. If you have need of an operation that is listed as not implemented, feel free to open an issue on the http://github.com/elastic/eland, or give a thumbs up to already created issues. Contributions are also welcome!

The following table is structured as follows: The first column contains the method name. The second column is a flag for whether or not there is an implementation in Modin for the method in the left column. `Y` stands for yes, `N` stands for no.

https://github.com/adgirish/kaggleScape/blob/master/results/annotResults.csv represents a prioritised list.

Method	Count	Notes
pd.read_csv	1422	y
pd.DataFrame	886	y
df.append	792	n
df.mean	783	y
df.head	783	y
df.drop	761	y
df.sum	755	y
df.to_csv	693	y
df.get	669	y
df.mode	653	n
df.astype	649	n
df.sub	637	n
pd.concat	582	n
df.apply	577	n
df.groupby	557	n
df.join	544	n
df.fillna	543	n
df.max	508	y
df.reset_index	434	n
pd.unique	433	n
df.le	405	n
df.count	399	y

Continued on next page

Table 1 – continued from previous page

pd.value_counts	397	y
df.sort_values	390	n
df.transform	387	n
df.merge	376	n
df.add	346	n
df.isnull	338	n
df.min	321	y
df.copy	314	n
df.replace	300	n
df.std	261	n
df.hist	246	y
df.filter	234	n
df.describe	220	y
df.ne	218	n
df.corr	217	n
df.median	217	n
df.items	212	n
pd.to_datetime	204	n
df.isin	203	n
df.dropna	195	n
pd.get_dummies	190	n
df.rename	185	n
df.info	180	y
df.set_index	166	n
df.keys	159	y
df.sample	155	n
df.agg	140	y
df.where	138	n
df.boxplot	134	n
df.clip	116	n
df.round	116	n
df.abs	101	n
df.stack	97	n
df.tail	94	y
df.update	92	n
df.iterrows	90	n
df.transpose	87	n
df.any	85	n
df.pipe	80	n
pd.eval	73	n
df.eval	73	n
pd.read_json	72	n
df.nunique	70	y
df.pivot	70	n
df.select	68	n
df.as_matrix	67	n
df.notnull	66	n
df.cumsum	66	n
df.prod	64	n
df.unstack	64	n

Continued on next page

Table 1 – continued from previous page

df.drop_duplicates	63	n
df.div	63	n
pd.crosstab	59	n
df.select_dtypes	57	y
df.pow	56	n
df.sort_index	56	n
df.product	52	n
df.isna	51	n
df.dot	46	n
pd.cut	45	n
df.bool	44	n
df.to_dict	44	n
df.diff	44	n
df.insert	44	n
df.pop	44	n
df.query	43	y
df.var	43	n
df.__init__	41	y
pd.to_numeric	39	n
df.squeeze	39	n
df.ge	37	n
df.quantile	37	n
df.reindex	37	n
df.rolling	35	n
pd.factorize	32	n
pd.melt	31	n
df.melt	31	n
df.rank	31	n
pd.read_table	30	n
pd.pivot_table	30	n
df.idxmax	30	n
pd.test	29	n
df.iteritems	29	n
df.shift	28	n
df.mul	28	n
pd.qcut	25	n
df.set_value	25	n
df.all	24	n
df.skew	24	n
df.aggregate	23	y
pd.match	22	n
df.nlargest	22	n
df.multiply	21	n
df.set_axis	19	n
df.eq	18	n
df.resample	18	n
pd.read_sql	17	n
df.duplicated	16	n
pd.date_range	16	n
df.interpolate	15	n

Continued on next page

Table 1 – continued from previous page

df.memory_usage	15	n
df.divide	14	n
df.cov	13	n
df.assign	12	n
df.subtract	12	n
pd.read_pickle	11	n
df.applymap	11	n
df.first	11	n
df.kurt	10	n
df.truncate	10	n
df.get_value	9	n
pd.read_hdf	9	n
df.to_html	9	y
pd.read_sql_query	9	n
df.take	8	n
df.to_pickle	7	n
df.itertuples	7	n
df.to_string	7	y
df.last	7	n
df.sem	7	n
pd.to_pickle	7	n
df.to_json	7	n
df.idxmin	7	n
df.xs	6	n
df.combine	6	n
pd.rolling_mean	6	n
df.to_period	6	n
df.convert_objects	5	n
df.mask	4	n
df.pct_change	4	n
df.add_prefix	4	n
pd.read_excel	4	n
pd.rolling_std	3	n
df.to_records	3	n
df.corrwith	3	n
df.swapaxes	3	n
df.__iter__	3	n
df.to_sql	3	n
pd.read_feather	3	n
df.to_feather	3	n
df.__len__	3	n
df.kurtosis	3	n
df.mod	2	n
df.to_sparse	2	n
df.get_values	2	n
df.__eq__	2	n
pd.bdate_range	2	n
df.get_dtype_counts	2	n
df.combine_first	2	n
df._get_numeric_data	2	n

Continued on next page

Table 1 – continued from previous page

df.nsmallest	2	n
pd.scatter_matrix	2	n
df.rename_axis	2	n
df.__setstate__	2	n
df.cumprod	2	n
df.__getstate__	2	n
df.equals	2	n
df.__getitem__	2	y
df.clip_upper	2	n
df.floordiv	2	n
df.to_excel	2	n
df.reindex_axis	1	n
pd.to_timedelta	1	n
df.ewm	1	n
df.tz_localize	1	n
df.tz_convert	1	n
df.to_hdf	1	n
df.lookup	1	n
pd.merge_ordered	1	n
df.swaplevel	1	n
df.first_valid_index	1	n
df.lt	1	n
df.add_suffix	1	n
pd.rolling_median	1	n
df.to_dense	1	n
df.mad	1	n
df.align	1	n
df.__copy__	1	n
pd.set_eng_float_format	1	n
df.add_suffix	1	n
pd.rolling_median	1	n
df.to_dense	1	n
df.mad	1	n
df.align	1	n
df.__copy__	1	n
pd.set_eng_float_format	1	n

DataFrame method	Eland Implementation? (Y/N/P/D)	Notes for Current implementation
`T`	N	
`abs`	N	
`add`	N	
`add_prefix`	N	
`add_suffix`	N	
`agg aggregate`	Y	
`align`	N	
`all`	N	
`any`	N	
`append`	N	
`apply`	N	See `agg`

Continued on next page

Table 2 – continued from previous page

applymap	N	
as_blocks	N	
as_matrix	N	
asfreq	N	
asof	N	
assign	N	
astype	N	
at	N	
at_time	N	
axes	N	
between_time	N	
bfill	N	
blocks	N	
bool	N	
boxplot	N	
clip	N	
clip_lower	N	
clip_upper	N	
combine	N	
combine_first	N	
compound	N	
consolidate	N	
convert_objects	N	
copy	N	
corr	N	
corrwith	N	
count	Y	
cov	N	
cummax	N	
cummin	N	
cumprod	N	
cumsum	N	
describe	Y	
diff	N	
div	N	
divide	N	
dot	N	
drop	Y	
drop_duplicates	N	
dropna	N	
dtypes	Y	
duplicated	N	
empty	Y	
eq	N	
equals	N	
eval	N	
ewm	N	
expanding	N	
ffill	N	
fillna	N	

Continued on next page

Table 2 – continued from previous page

filter	N	
first	N	
first_valid_index	N	
floordiv	N	
from_csv	N	
from_dict	N	
from_items	N	
from_records	N	
ftypes	N	
ge	N	
get	Y	
get_dtype_counts	N	
get_ftype_counts	N	
get_value	N	
get_values	N	
groupby	N	
gt	N	
head	Y	
hist	Y	
iat	N	
idxmax	N	
idxmin	N	
iloc	N	
infer_objects	N	
info	Y	
insert	N	
interpolate	N	
is_copy	N	
isin	N	
isna	N	
isnull	N	
items	N	
iteritems	N	
iterrows	N	
itertuples	N	
ix	N	
join	N	
keys	Y	
kurt	N	
kurtosis	N	
last	N	
last_valid_index	N	
le	N	
loc	N	
lookup	N	
lt	N	
mad	N	
mask	N	
max	Y	
mean	Y	

Continued on next page

Table 2 – continued from previous page

median	N	
melt	N	
memory_usage	N	
merge	N	
min	Y	
mod	N	
mode	N	
mul	N	
multiply	N	
ndim	N	
ne	N	
nlargest	N	
notna	N	
notnull	N	
nsmallest	N	
nunique	Y	
pct_change	N	
pipe	N	
pivot	N	
pivot_table	N	
plot	N	
pop	N	
pow	N	
prod	N	
product	N	
quantile	N	
query	Y	
radd	N	
rank	N	
rdiv	N	
reindex	N	
reindex_axis	N	
reindex_like	N	
rename	N	
rename_axis	N	
reorder_levels	N	
replace	N	
resample	N	
reset_index	N	
rfloordiv	N	
rmod	N	
rmul	N	
rolling	N	
round	N	
rpow	N	
rsub	N	
rtruediv	N	
sample	N	
select	N	
select_dtypes	Y	

Continued on next page

Table 2 – continued from previous page

sem	N	
set_axis	N	
set_index	N	
set_value	N	
shape	Y	
shift	N	
size	N	
skew	N	
slice_shift	N	
sort_index	N	
sort_values	N	
sortlevel	N	
squeeze	N	
stack	N	
std	N	
style	N	
sub	N	
subtract	N	
sum	Y	
swapaxes	N	
swaplevel	N	
tail	Y	
take	N	
to_clipboard	N	
to_csv	Y	
to_dense	N	
to_dict	N	
to_excel	N	
to_feather	N	
to_gbq	N	
to_hdf	N	
to_html	Y	
to_json	N	
to_latex	N	
to_msgpack	N	
to_panel	N	
to_parquet	N	
to_period	N	
to_pickle	N	
to_records	N	
to_sparse	N	
to_sql	N	
to_stata	N	
to_string	Y	Default sets *max_rows=60*
to_timestamp	N	
to_xarray	N	
transform	N	
transpose	N	
truediv	N	
truncate	N	

Continued on next page

Table 2 – continued from previous page

tshift	N	
tz_convert	N	
tz_localize	N	
unstack	N	
update	N	
values	N	
var	N	
where	N	
xs	N	Deprecated in pandas

CHAPTER 3

Development

3.1 Contributing to eland

Eland is an open source project and we love to receive contributions from our community — you! There are many ways to contribute, from writing tutorials or blog posts, improving the documentation, submitting bug reports and feature requests or writing code which can be incorporated into eland itself.

3.1.1 Bug reports

If you think you have found a bug in eland, first make sure that you are testing against the latest version of eland - your issue may already have been fixed. If not, search our issues list on GitHub in case a similar issue has already been opened.

It is very helpful if you can prepare a reproduction of the bug. In other words, provide a small test case which we can run to confirm your bug. It makes it easier to find the problem and to fix it. Test cases should be provided as python scripts, ideally with some details of your Elasticsearch environment and index mappings, and (where appropriate) a pandas example.

Provide as much information as you can. You may think that the problem lies with your query, when actually it depends on how your data is indexed. The easier it is for us to recreate your problem, the faster it is likely to be fixed.

3.1.2 Feature requests

If you find yourself wishing for a feature that doesn't exist in eland, you are probably not alone. There are bound to be others out there with similar needs. Many of the features that eland has today have been added because our users saw the need. Open an issue on our issues list on GitHub which describes the feature you would like to see, why you need it, and how it should work.

3.1.3 Contributing code and documentation changes

If you have a bugfix or new feature that you would like to contribute to eland, please find or open an issue about it first. Talk about what you would like to do. It may be that somebody is already working on it, or that there are particular issues that you should know about before implementing the change.

We enjoy working with contributors to get their code accepted. There are many approaches to fixing a problem and it is important to find the best approach before writing too much code.

Note that it is unlikely the project will merge refactors for the sake of refactoring. These types of pull requests have a high cost to maintainers in reviewing and testing with little to no tangible benefit. This especially includes changes generated by tools.

The process for contributing to any of the Elastic repositories is similar. Details for individual projects can be found below.

Fork and clone the repository

You will need to fork the main eland code or documentation repository and clone it to your local machine. See github help page for help.

Further instructions for specific projects are given below.

Submitting your changes

Once your changes and tests are ready to submit for review:

1. Test your changes

   Run the test suite to make sure that nothing is broken (TODO add link to testing doc).

2. Sign the Contributor License Agreement

   Please make sure you have signed our Contributor License Agreement. We are not asking you to assign copyright to us, but to give us the right to distribute your code without restriction. We ask this of all contributors in order to assure our users of the origin and continuing existence of the code. You only need to sign the CLA once.

3. Rebase your changes

   Update your local repository with the most recent code from the main eland repository, and rebase your branch on top of the latest master branch. We prefer your initial changes to be squashed into a single commit. Later, if we ask you to make changes, add them as separate commits. This makes them easier to review. As a final step before merging we will either ask you to squash all commits yourself or we'll do it for you.

4. Submit a pull request

   Push your local changes to your forked copy of the repository and submit a pull request. In the pull request, choose a title which sums up the changes that you have made, and in the body provide more details about what your changes do. Also mention the number of the issue where discussion has taken place, eg "Closes #123".

Then sit back and wait. There will probably be discussion about the pull request and, if any changes are needed, we would love to work with you to get your pull request merged into eland.

Please adhere to the general guideline that you should never force push to a publicly shared branch. Once you have opened your pull request, you should consider your branch publicly shared. Instead of force pushing you can just add incremental commits; this is generally easier on your reviewers. If you need to pick up changes from master, you can merge master into your branch. A reviewer might ask you to rebase a long-running pull request in which case force pushing is okay for that request. Note that squashing at the end of the review process should also not be done, that can be done when the pull request is integrated via GitHub.

3.1.4 Contributing to the eland codebase

Repository: https://github.com/elastic/eland

We internally develop using the PyCharm IDE. For PyCharm, we are currently using a minimum version of PyCharm 2019.2.4.

Configuring PyCharm And Running Tests

(All commands should be run from module root)

- Create a new project via 'Check out from Version Control'->'Git' on the "Welcome to PyCharm" page (or other)
- Enter the URL to your fork of eland (e.g. `git@github.com:stevedodson/eland.git`)
- Click 'Yes' for 'Checkout from Version Control'
- Configure PyCharm environment:
- In 'Preferences' configure a 'Project: eland'->'Project Interpreter'. Generally, we recommend creating a virtual environment (TODO link to installing for python version support).
- In 'Preferences' set 'Tools'->'Python Integrated Tools'->'Default test runner' to `pytest`
- In 'Preferences' set 'Tools'->'Python Integrated Tools'->'Docstring format' to `numpy`
- Install development requirements. Open terminal in virtual environment and run `pip install -r requirements-dev.txt`
- Setup Elasticsearch instance (assumes `localhost:9200`), and run `python -m eland.tests.setup_tests` to setup test environment - *note this modifies Elasticsearch indices*
- Run `pytest --doctest-modules` to validate install

Documentation

- Install documentation requirements. Open terminal in virtual environment and run `pip install -r requirements-dev.txt`

CHAPTER 4

Examples

4.1 Eland Demo Notebook

[1]:
```
import eland as ed
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from elasticsearch import Elasticsearch

# Import standard test settings for consistent results
from eland.conftest import *
```

4.1.1 Compare eland DataFrame vs pandas DataFrame

Create an eland.DataFrame from a `flights` index

[2]:
```
ed_flights = ed.read_es('localhost', 'flights')
```

[3]:
```
type(ed_flights)
```

[3]:
```
eland.dataframe.DataFrame
```

Compare to pandas DataFrame (created from the same data)

[4]:
```
pd_flights = ed.eland_to_pandas(ed_flights)
```

[5]:
```
type(pd_flights)
```

[5]:
```
pandas.core.frame.DataFrame
```

4.1.2 Attributes and underlying data

DataFrame.columns

```
[6]: pd_flights.columns
```

```
[6]: Index(['AvgTicketPrice', 'Cancelled', 'Carrier', 'Dest', 'DestAirportID',
     ↪'DestCityName',
           'DestCountry', 'DestLocation', 'DestRegion', 'DestWeather', 'DistanceKilometers
     ↪',
           'DistanceMiles', 'FlightDelay', 'FlightDelayMin', 'FlightDelayType', 'FlightNum
     ↪',
           'FlightTimeHour', 'FlightTimeMin', 'Origin', 'OriginAirportID', 'OriginCityName
     ↪',
           'OriginCountry', 'OriginLocation', 'OriginRegion', 'OriginWeather', 'dayOfWeek
     ↪',
           'timestamp'],
          dtype='object')
```

```
[7]: ed_flights.columns
```

```
[7]: Index(['AvgTicketPrice', 'Cancelled', 'Carrier', 'Dest', 'DestAirportID',
     ↪'DestCityName',
           'DestCountry', 'DestLocation', 'DestRegion', 'DestWeather', 'DistanceKilometers
     ↪',
           'DistanceMiles', 'FlightDelay', 'FlightDelayMin', 'FlightDelayType', 'FlightNum
     ↪',
           'FlightTimeHour', 'FlightTimeMin', 'Origin', 'OriginAirportID', 'OriginCityName
     ↪',
           'OriginCountry', 'OriginLocation', 'OriginRegion', 'OriginWeather', 'dayOfWeek
     ↪',
           'timestamp'],
          dtype='object')
```

DataFrame.dtypes

```
[8]: pd_flights.dtypes
```

```
[8]: AvgTicketPrice           float64
     Cancelled                  bool
     Carrier                  object
     Dest                     object
     DestAirportID            object
                              ...
     OriginLocation           object
     OriginRegion             object
     OriginWeather            object
     dayOfWeek                 int64
     timestamp         datetime64[ns]
     Length: 27, dtype: object
```

```
[9]: ed_flights.dtypes
```

```
[9]: AvgTicketPrice           float64
     Cancelled                  bool
     Carrier                  object
```

(continues on next page)

(continued from previous page)

```
Dest                      object
DestAirportID             object
                         ...
OriginLocation            object
OriginRegion              object
OriginWeather             object
dayOfWeek                  int64
timestamp         datetime64[ns]
Length: 27, dtype: object
```

DataFrame.select_dtypes

[10]:
```
pd_flights.select_dtypes(include=np.number)
```

[10]:
```
       AvgTicketPrice  DistanceKilometers  ...  FlightTimeMin  dayOfWeek
0           841.265642        16492.326654  ...    1030.770416          0
1           882.982662         8823.400140  ...     464.389481          0
2           190.636904            0.000000  ...       0.000000          0
3           181.694216          555.737767  ...     222.749059          0
4           730.041778        13358.244200  ...     785.779071          0
...                ...                 ...  ...            ...        ...
13054      1080.446279         8058.581753  ...     402.929088          6
13055       646.612941         7088.598322  ...     644.418029          6
13056       997.751876        10920.652972  ...     937.540811          6
13057      1102.814465        18748.859647  ...    1697.404971          6
13058       858.144337        16809.141923  ...    1610.761827          6

[13059 rows x 7 columns]
```

[11]:
```
ed_flights.select_dtypes(include=np.number)
```

[11]:
```
       AvgTicketPrice  DistanceKilometers  ...  FlightTimeMin  dayOfWeek
0           841.265642        16492.326654  ...    1030.770416          0
1           882.982662         8823.400140  ...     464.389481          0
2           190.636904            0.000000  ...       0.000000          0
3           181.694216          555.737767  ...     222.749059          0
4           730.041778        13358.244200  ...     785.779071          0
...                ...                 ...  ...            ...        ...
13054      1080.446279         8058.581753  ...     402.929088          6
13055       646.612941         7088.598322  ...     644.418029          6
13056       997.751876        10920.652972  ...     937.540811          6
13057      1102.814465        18748.859647  ...    1697.404971          6
13058       858.144337        16809.141923  ...    1610.761827          6

[13059 rows x 7 columns]
```

DataFrame.empty

[12]:
```
pd_flights.empty
```

[12]:
```
False
```

[13]:
```
ed_flights.empty
```

```
[13]: False
```

DataFrame.shape

```
[14]: pd_flights.shape
[14]: (13059, 27)
```

```
[15]: ed_flights.shape
[15]: (13059, 27)
```

DataFrame.index

Note, `eland.DataFrame.index` does not mirror `pandas.DataFrame.index`.

```
[16]: pd_flights.index
[16]: Index(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
             ...
             '13049', '13050', '13051', '13052', '13053', '13054', '13055', '13056', '13057
      ↪', '13058'],
            dtype='object', length=13059)
```

```
[17]: # NBVAL_IGNORE_OUTPUT
      ed_flights.index
[17]: <eland.index.Index at 0x116b3efd0>
```

```
[18]: ed_flights.index.index_field
[18]: '_id'
```

DataFrame.values

Note, `eland.DataFrame.values` is not supported.

```
[19]: pd_flights.values
[19]: array([[841.2656419677076, False, 'Kibana Airlines', ..., 'Sunny', 0,
              Timestamp('2018-01-01 00:00:00')],
             [882.9826615595518, False, 'Logstash Airways', ..., 'Clear', 0,
              Timestamp('2018-01-01 18:27:00')],
             [190.6369038508356, False, 'Logstash Airways', ..., 'Rain', 0,
              Timestamp('2018-01-01 17:11:14')],
             ...,
             [997.7518761454494, False, 'Logstash Airways', ..., 'Sunny', 6,
              Timestamp('2018-02-11 04:09:27')],
             [1102.8144645388556, False, 'JetBeats', ..., 'Hail', 6,
              Timestamp('2018-02-11 08:28:21')],
             [858.1443369038839, False, 'JetBeats', ..., 'Rain', 6,
              Timestamp('2018-02-11 14:54:34')]], dtype=object)
```

[20]:
```
try:
    ed_flights.values
except AttributeError as e:
    print(e)
```

```
This method would scan/scroll the entire Elasticsearch index(s) into memory. If this␣
↪is explicitly required, and there is sufficient memory, call `ed.eland_to_pandas(ed_
↪df).values`
```

4.1.3 Indexing, iteration

DataFrame.head

[21]:
```
pd_flights.head()
```

[21]:
```
   AvgTicketPrice  Cancelled  ... dayOfWeek           timestamp
0      841.265642      False  ...         0 2018-01-01 00:00:00
1      882.982662      False  ...         0 2018-01-01 18:27:00
2      190.636904      False  ...         0 2018-01-01 17:11:14
3      181.694216       True  ...         0 2018-01-01 10:33:28
4      730.041778      False  ...         0 2018-01-01 05:13:00

[5 rows x 27 columns]
```

[22]:
```
ed_flights.head()
```

[22]:
```
   AvgTicketPrice  Cancelled  ... dayOfWeek           timestamp
0      841.265642      False  ...         0 2018-01-01 00:00:00
1      882.982662      False  ...         0 2018-01-01 18:27:00
2      190.636904      False  ...         0 2018-01-01 17:11:14
3      181.694216       True  ...         0 2018-01-01 10:33:28
4      730.041778      False  ...         0 2018-01-01 05:13:00

[5 rows x 27 columns]
```

DataFrame.tail

[23]:
```
pd_flights.tail()
```

[23]:
```
       AvgTicketPrice  Cancelled  ... dayOfWeek           timestamp
13054     1080.446279      False  ...         6 2018-02-11 20:42:25
13055      646.612941      False  ...         6 2018-02-11 01:41:57
13056      997.751876      False  ...         6 2018-02-11 04:09:27
13057     1102.814465      False  ...         6 2018-02-11 08:28:21
13058      858.144337      False  ...         6 2018-02-11 14:54:34

[5 rows x 27 columns]
```

[24]:
```
ed_flights.tail()
```

[24]:
```
       AvgTicketPrice  Cancelled  ... dayOfWeek           timestamp
13054     1080.446279      False  ...         6 2018-02-11 20:42:25
13055      646.612941      False  ...         6 2018-02-11 01:41:57
13056      997.751876      False  ...         6 2018-02-11 04:09:27
```

(continues on next page)

(continued from previous page)

```
13057     1102.814465      False  ...          6 2018-02-11 08:28:21
13058      858.144337      False  ...          6 2018-02-11 14:54:34

[5 rows x 27 columns]
```

DataFrame.keys

[25]:
```
pd_flights.keys()
```

[25]:
```
Index(['AvgTicketPrice', 'Cancelled', 'Carrier', 'Dest', 'DestAirportID',
↪'DestCityName',
       'DestCountry', 'DestLocation', 'DestRegion', 'DestWeather', 'DistanceKilometers
↪',
       'DistanceMiles', 'FlightDelay', 'FlightDelayMin', 'FlightDelayType', 'FlightNum
↪',
       'FlightTimeHour', 'FlightTimeMin', 'Origin', 'OriginAirportID', 'OriginCityName
↪',
       'OriginCountry', 'OriginLocation', 'OriginRegion', 'OriginWeather', 'dayOfWeek
↪',
       'timestamp'],
      dtype='object')
```

[26]:
```
ed_flights.keys()
```

[26]:
```
Index(['AvgTicketPrice', 'Cancelled', 'Carrier', 'Dest', 'DestAirportID',
↪'DestCityName',
       'DestCountry', 'DestLocation', 'DestRegion', 'DestWeather', 'DistanceKilometers
↪',
       'DistanceMiles', 'FlightDelay', 'FlightDelayMin', 'FlightDelayType', 'FlightNum
↪',
       'FlightTimeHour', 'FlightTimeMin', 'Origin', 'OriginAirportID', 'OriginCityName
↪',
       'OriginCountry', 'OriginLocation', 'OriginRegion', 'OriginWeather', 'dayOfWeek
↪',
       'timestamp'],
      dtype='object')
```

DataFrame.get

[27]:
```
pd_flights.get('Carrier')
```

[27]:
```
0          Kibana Airlines
1         Logstash Airways
2         Logstash Airways
3          Kibana Airlines
4          Kibana Airlines
                ...
13054     Logstash Airways
13055     Logstash Airways
13056     Logstash Airways
13057             JetBeats
13058             JetBeats
Name: Carrier, Length: 13059, dtype: object
```

[28]:
```
ed_flights.get('Carrier')
```

[28]:
```
0         Kibana Airlines
1        Logstash Airways
2        Logstash Airways
3         Kibana Airlines
4         Kibana Airlines
              ...
13054    Logstash Airways
13055    Logstash Airways
13056    Logstash Airways
13057            JetBeats
13058            JetBeats
Name: Carrier, Length: 13059, dtype: object
```

[29]:
```
pd_flights.get(['Carrier', 'Origin'])
```

[29]:
```
                Carrier                                              Origin
0       Kibana Airlines                             Frankfurt am Main Airport
1      Logstash Airways                       Cape Town International Airport
2      Logstash Airways                            Venice Marco Polo Airport
3       Kibana Airlines                          Naples International Airport
4       Kibana Airlines       Licenciado Benito Juarez International Airport
...                 ...                                                  ...
13054  Logstash Airways                           Pisa International Airport
13055  Logstash Airways  Winnipeg / James Armstrong Richardson Internat...
13056  Logstash Airways       Licenciado Benito Juarez International Airport
13057          JetBeats                                        Itami Airport
13058          JetBeats                       Adelaide International Airport

[13059 rows x 2 columns]
```

List input not currently supported by `eland.DataFrame.get`

[30]:
```
try:
    ed_flights.get(['Carrier', 'Origin'])
except TypeError as e:
    print(e)
```

```
unhashable type: 'list'
```

DataFrame.query

[31]:
```
pd_flights.query('Carrier == "Kibana Airlines" & AvgTicketPrice > 900.0 & Cancelled␣
↪== True')
```

[31]:
```
       AvgTicketPrice  Cancelled  ... dayOfWeek           timestamp
8           960.869736       True  ...         0 2018-01-01 12:09:35
26          975.812632       True  ...         0 2018-01-01 15:38:32
311         946.358410       True  ...         0 2018-01-01 11:51:12
651         975.383864       True  ...         2 2018-01-03 21:13:17
950         907.836523       True  ...         2 2018-01-03 05:14:51
...                ...        ...  ...       ...                 ...
12820       909.973606       True  ...         5 2018-02-10 05:11:35
12906       983.429244       True  ...         6 2018-02-11 06:19:58
12918      1136.678150       True  ...         6 2018-02-11 16:03:10
12919      1105.211803       True  ...         6 2018-02-11 05:36:05
```

(continues on next page)

(continued from previous page)

```
13013     1055.350213       True  ...         6 2018-02-11 13:20:16

[68 rows x 27 columns]
```

`eland.DataFrame.query` requires qualifier on bool i.e.

`ed_flights.query('Carrier == "Kibana Airlines" & AvgTicketPrice > 900.0 & Cancelled')` fails

[32]:
```
ed_flights.query('Carrier == "Kibana Airlines" & AvgTicketPrice > 900.0 & Cancelled
↪== True')
```

[32]:
```
       AvgTicketPrice  Cancelled  ... dayOfWeek           timestamp
8           960.869736       True  ...         0 2018-01-01 12:09:35
26          975.812632       True  ...         0 2018-01-01 15:38:32
311         946.358410       True  ...         0 2018-01-01 11:51:12
651         975.383864       True  ...         2 2018-01-03 21:13:17
950         907.836523       True  ...         2 2018-01-03 05:14:51
...                ...        ...  ...       ...                 ...
12820       909.973606       True  ...         5 2018-02-10 05:11:35
12906       983.429244       True  ...         6 2018-02-11 06:19:58
12918      1136.678150       True  ...         6 2018-02-11 16:03:10
12919      1105.211803       True  ...         6 2018-02-11 05:36:05
13013      1055.350213       True  ...         6 2018-02-11 13:20:16

[68 rows x 27 columns]
```

Boolean indexing query

[33]:
```
pd_flights[(pd_flights.Carrier=="Kibana Airlines") &
           (pd_flights.AvgTicketPrice > 900.0) &
           (pd_flights.Cancelled == True)]
```

[33]:
```
       AvgTicketPrice  Cancelled  ... dayOfWeek           timestamp
8           960.869736       True  ...         0 2018-01-01 12:09:35
26          975.812632       True  ...         0 2018-01-01 15:38:32
311         946.358410       True  ...         0 2018-01-01 11:51:12
651         975.383864       True  ...         2 2018-01-03 21:13:17
950         907.836523       True  ...         2 2018-01-03 05:14:51
...                ...        ...  ...       ...                 ...
12820       909.973606       True  ...         5 2018-02-10 05:11:35
12906       983.429244       True  ...         6 2018-02-11 06:19:58
12918      1136.678150       True  ...         6 2018-02-11 16:03:10
12919      1105.211803       True  ...         6 2018-02-11 05:36:05
13013      1055.350213       True  ...         6 2018-02-11 13:20:16

[68 rows x 27 columns]
```

[34]:
```
ed_flights[(ed_flights.Carrier=="Kibana Airlines") &
           (ed_flights.AvgTicketPrice > 900.0) &
           (ed_flights.Cancelled == True)]
```

[34]:
```
       AvgTicketPrice  Cancelled  ... dayOfWeek           timestamp
8           960.869736       True  ...         0 2018-01-01 12:09:35
26          975.812632       True  ...         0 2018-01-01 15:38:32
311         946.358410       True  ...         0 2018-01-01 11:51:12
```

(continues on next page)

(continued from previous page)

```
651         975.383864       True  ...          2 2018-01-03 21:13:17
950         907.836523       True  ...          2 2018-01-03 05:14:51
...                ...        ...  ...        ...                 ...
12820       909.973606       True  ...          5 2018-02-10 05:11:35
12906       983.429244       True  ...          6 2018-02-11 06:19:58
12918      1136.678150       True  ...          6 2018-02-11 16:03:10
12919      1105.211803       True  ...          6 2018-02-11 05:36:05
13013      1055.350213       True  ...          6 2018-02-11 13:20:16

[68 rows x 27 columns]
```

4.1.4 Function application, GroupBy & window

DataFrame.aggs

[35]:
```
pd_flights[['DistanceKilometers', 'AvgTicketPrice']].aggregate(['sum', 'min', 'std'])
```

[35]:
```
     DistanceKilometers  AvgTicketPrice
sum        9.261629e+07    8.204365e+06
min        0.000000e+00    1.000205e+02
std        4.578438e+03    2.663969e+02
```

`eland.DataFrame.aggregate` currently only supported numeric columns

[36]:
```
ed_flights[['DistanceKilometers', 'AvgTicketPrice']].aggregate(['sum', 'min', 'std'])
```

[36]:
```
     DistanceKilometers  AvgTicketPrice
sum        9.261629e+07    8.204365e+06
min        0.000000e+00    1.000205e+02
std        4.578263e+03    2.663867e+02
```

4.1.5 Computations / descriptive stats

DataFrame.count

[37]:
```
pd_flights.count()
```

[37]:
```
AvgTicketPrice    13059
Cancelled         13059
Carrier           13059
Dest              13059
DestAirportID     13059
                  ...
OriginLocation    13059
OriginRegion      13059
OriginWeather     13059
dayOfWeek         13059
timestamp         13059
Length: 27, dtype: int64
```

[38]:
```
ed_flights.count()
```

[38]:
```
AvgTicketPrice    13059
Cancelled         13059
Carrier           13059
Dest              13059
DestAirportID     13059
                  ...
OriginLocation    13059
OriginRegion      13059
OriginWeather     13059
dayOfWeek         13059
timestamp         13059
Length: 27, dtype: int64
```

DataFrame.describe

[39]:
```
pd_flights.describe()
```

[39]:
```
       AvgTicketPrice  DistanceKilometers  ...  FlightTimeMin     dayOfWeek
count    13059.000000        13059.000000  ...   13059.000000  13059.000000
mean       628.253689         7092.142455  ...     511.127842      2.835975
std        266.396861         4578.438497  ...     334.753952      1.939439
min        100.020528            0.000000  ...       0.000000      0.000000
25%        409.893816         2459.705673  ...     252.333192      1.000000
50%        640.556668         7610.330866  ...     503.045170      3.000000
75%        842.185470         9736.637600  ...     720.416036      4.000000
max       1199.729053        19881.482315  ...    1902.902032      6.000000

[8 rows x 7 columns]
```

Values returned from `eland.DataFrame.describe` may vary due to results of Elasticsearch aggregations.

[40]:
```
# NBVAL_IGNORE_OUTPUT
ed_flights.describe()
```

[40]:
```
       AvgTicketPrice  DistanceKilometers  ...  FlightTimeMin     dayOfWeek
count    13059.000000        13059.000000  ...   13059.000000  13059.000000
mean       628.253689         7092.142457  ...     511.127842      2.835975
std        266.386661         4578.263193  ...     334.741135      1.939365
min        100.020531            0.000000  ...       0.000000      0.000000
25%        410.008918         2470.545974  ...     251.938710      1.000000
50%        640.362667         7612.072403  ...     503.148975      3.000000
75%        840.617448         9738.206675  ...     720.026320      4.160448
max       1199.729004        19881.482422  ...    1902.901978      6.000000

[8 rows x 7 columns]
```

DataFrame.info

[41]:
```
pd_flights.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 13059 entries, 0 to 13058
Data columns (total 27 columns):
AvgTicketPrice        13059 non-null float64
Cancelled             13059 non-null bool
```

(continues on next page)

(continued from previous page)

```
Carrier                13059 non-null object
Dest                   13059 non-null object
DestAirportID          13059 non-null object
DestCityName           13059 non-null object
DestCountry            13059 non-null object
DestLocation           13059 non-null object
DestRegion             13059 non-null object
DestWeather            13059 non-null object
DistanceKilometers     13059 non-null float64
DistanceMiles          13059 non-null float64
FlightDelay            13059 non-null bool
FlightDelayMin         13059 non-null int64
FlightDelayType        13059 non-null object
FlightNum              13059 non-null object
FlightTimeHour         13059 non-null float64
FlightTimeMin          13059 non-null float64
Origin                 13059 non-null object
OriginAirportID        13059 non-null object
OriginCityName         13059 non-null object
OriginCountry          13059 non-null object
OriginLocation         13059 non-null object
OriginRegion           13059 non-null object
OriginWeather          13059 non-null object
dayOfWeek              13059 non-null int64
timestamp              13059 non-null datetime64[ns]
dtypes: bool(2), datetime64[ns](1), float64(5), int64(2), object(17)
memory usage: 3.2+ MB
```

[42]:
```
ed_flights.info()
```

```
<class 'eland.dataframe.DataFrame'>
Index: 13059 entries, 0 to 13058
Data columns (total 27 columns):
AvgTicketPrice         13059 non-null float64
Cancelled              13059 non-null bool
Carrier                13059 non-null object
Dest                   13059 non-null object
DestAirportID          13059 non-null object
DestCityName           13059 non-null object
DestCountry            13059 non-null object
DestLocation           13059 non-null object
DestRegion             13059 non-null object
DestWeather            13059 non-null object
DistanceKilometers     13059 non-null float64
DistanceMiles          13059 non-null float64
FlightDelay            13059 non-null bool
FlightDelayMin         13059 non-null int64
FlightDelayType        13059 non-null object
FlightNum              13059 non-null object
FlightTimeHour         13059 non-null float64
FlightTimeMin          13059 non-null float64
Origin                 13059 non-null object
OriginAirportID        13059 non-null object
OriginCityName         13059 non-null object
OriginCountry          13059 non-null object
OriginLocation         13059 non-null object
OriginRegion           13059 non-null object
```

(continues on next page)

(continued from previous page)

```
OriginWeather          13059 non-null object
dayOfWeek              13059 non-null int64
timestamp              13059 non-null datetime64[ns]
dtypes: bool(2), datetime64[ns](1), float64(5), int64(2), object(17)
memory usage: 96.0 bytes
```

DataFrame.max, DataFrame.min, DataFrame.mean, DataFrame.sum

max

[43]:
```
pd_flights.max(numeric_only=True)
```

[43]:
```
AvgTicketPrice         1199.729053
Cancelled                 1.000000
DistanceKilometers    19881.482315
DistanceMiles         12353.780369
FlightDelay               1.000000
FlightDelayMin          360.000000
FlightTimeHour           31.715034
FlightTimeMin          1902.902032
dayOfWeek                 6.000000
dtype: float64
```

`eland.DataFrame.max,min,mean,sum` only aggregate numeric columns

[44]:
```
ed_flights.max(numeric_only=True)
```

[44]:
```
AvgTicketPrice         1199.729004
Cancelled                 1.000000
DistanceKilometers    19881.482422
DistanceMiles         12353.780273
FlightDelay               1.000000
FlightDelayMin          360.000000
FlightTimeHour           31.715034
FlightTimeMin          1902.901978
dayOfWeek                 6.000000
dtype: float64
```

min

[45]:
```
pd_flights.min(numeric_only=True)
```

[45]:
```
AvgTicketPrice        100.020528
Cancelled               0.000000
DistanceKilometers      0.000000
DistanceMiles           0.000000
FlightDelay             0.000000
FlightDelayMin          0.000000
FlightTimeHour          0.000000
FlightTimeMin           0.000000
dayOfWeek               0.000000
dtype: float64
```

[46]:
```
ed_flights.min(numeric_only=True)
```

[46]:
```
AvgTicketPrice        100.020531
Cancelled               0.000000
DistanceKilometers      0.000000
DistanceMiles           0.000000
FlightDelay             0.000000
FlightDelayMin          0.000000
FlightTimeHour          0.000000
FlightTimeMin           0.000000
dayOfWeek               0.000000
dtype: float64
```

mean

[47]:
```
pd_flights.mean(numeric_only=True)
```

[47]:
```
AvgTicketPrice         628.253689
Cancelled                0.128494
DistanceKilometers    7092.142455
DistanceMiles         4406.853013
FlightDelay              0.251168
FlightDelayMin          47.335171
FlightTimeHour           8.518797
FlightTimeMin          511.127842
dayOfWeek                2.835975
dtype: float64
```

[48]:
```
ed_flights.mean(numeric_only=True)
```

[48]:
```
AvgTicketPrice         628.253689
Cancelled                0.128494
DistanceKilometers    7092.142457
DistanceMiles         4406.853010
FlightDelay              0.251168
FlightDelayMin          47.335171
FlightTimeHour           8.518797
FlightTimeMin          511.127842
dayOfWeek                2.835975
dtype: float64
```

sum

[49]:
```
pd_flights.sum(numeric_only=True)
```

[49]:
```
AvgTicketPrice        8.204365e+06
Cancelled             1.678000e+03
DistanceKilometers    9.261629e+07
DistanceMiles         5.754909e+07
FlightDelay           3.280000e+03
FlightDelayMin        6.181500e+05
FlightTimeHour        1.112470e+05
FlightTimeMin         6.674818e+06
dayOfWeek             3.703500e+04
dtype: float64
```

[50]:
```
ed_flights.sum(numeric_only=True)
```

[50]:
```
AvgTicketPrice        8.204365e+06
Cancelled             1.678000e+03
DistanceKilometers    9.261629e+07
DistanceMiles         5.754909e+07
FlightDelay           3.280000e+03
FlightDelayMin        6.181500e+05
FlightTimeHour        1.112470e+05
FlightTimeMin         6.674818e+06
dayOfWeek             3.703500e+04
dtype: float64
```

DataFrame.nunique

[51]:
```
pd_flights[['Carrier', 'Origin', 'Dest']].nunique()
```

[51]:
```
Carrier      4
Origin     156
Dest       156
dtype: int64
```

[52]:
```
ed_flights[['Carrier', 'Origin', 'Dest']].nunique()
```

[52]:
```
Carrier      4
Origin     156
Dest       156
dtype: int64
```

DataFrame.drop

[53]:
```
pd_flights.drop(columns=['AvgTicketPrice',
                         'Cancelled',
                         'DestLocation',
                         'Dest',
                         'DestAirportID',
                         'DestCityName',
                         'DestCountry'])
```

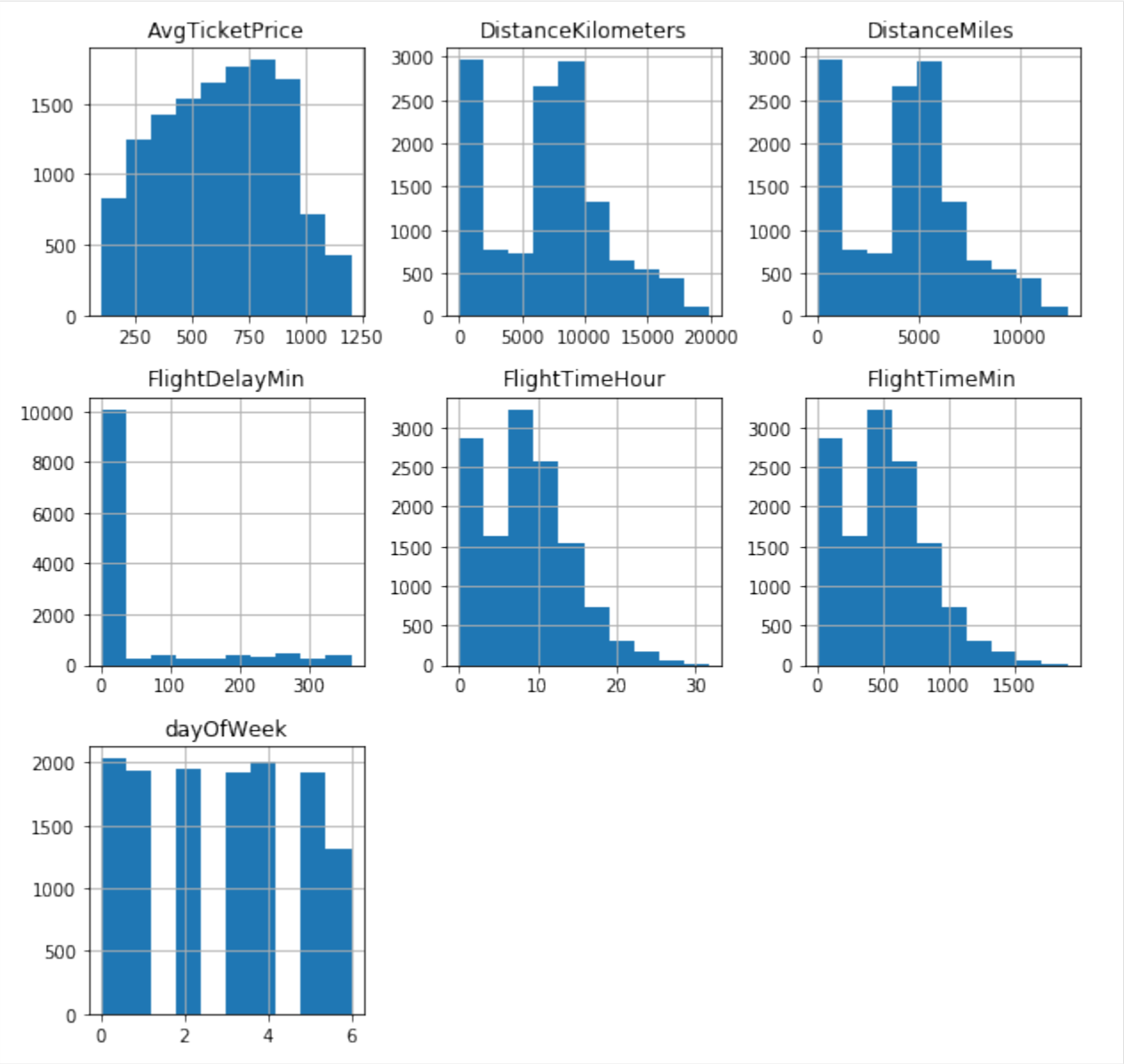
[53]:
```
                Carrier DestRegion  ... dayOfWeek           timestamp
0       Kibana Airlines      SE-BD  ...         0 2018-01-01 00:00:00
1      Logstash Airways      IT-34  ...         0 2018-01-01 18:27:00
2      Logstash Airways      IT-34  ...         0 2018-01-01 17:11:14
3       Kibana Airlines      IT-34  ...         0 2018-01-01 10:33:28
4       Kibana Airlines      SE-BD  ...         0 2018-01-01 05:13:00
...                 ...        ...  ...       ...                 ...
13054  Logstash Airways      SE-BD  ...         6 2018-02-11 20:42:25
13055  Logstash Airways      CH-ZH  ...         6 2018-02-11 01:41:57
13056  Logstash Airways     RU-AMU  ...         6 2018-02-11 04:09:27
13057          JetBeats      SE-BD  ...         6 2018-02-11 08:28:21
13058          JetBeats      US-DC  ...         6 2018-02-11 14:54:34

[13059 rows x 20 columns]
```

[54]:
```
ed_flights.drop(columns=['AvgTicketPrice',
                         'Cancelled',
                         'DestLocation',
                         'Dest',
                         'DestAirportID',
                         'DestCityName',
                         'DestCountry'])
```

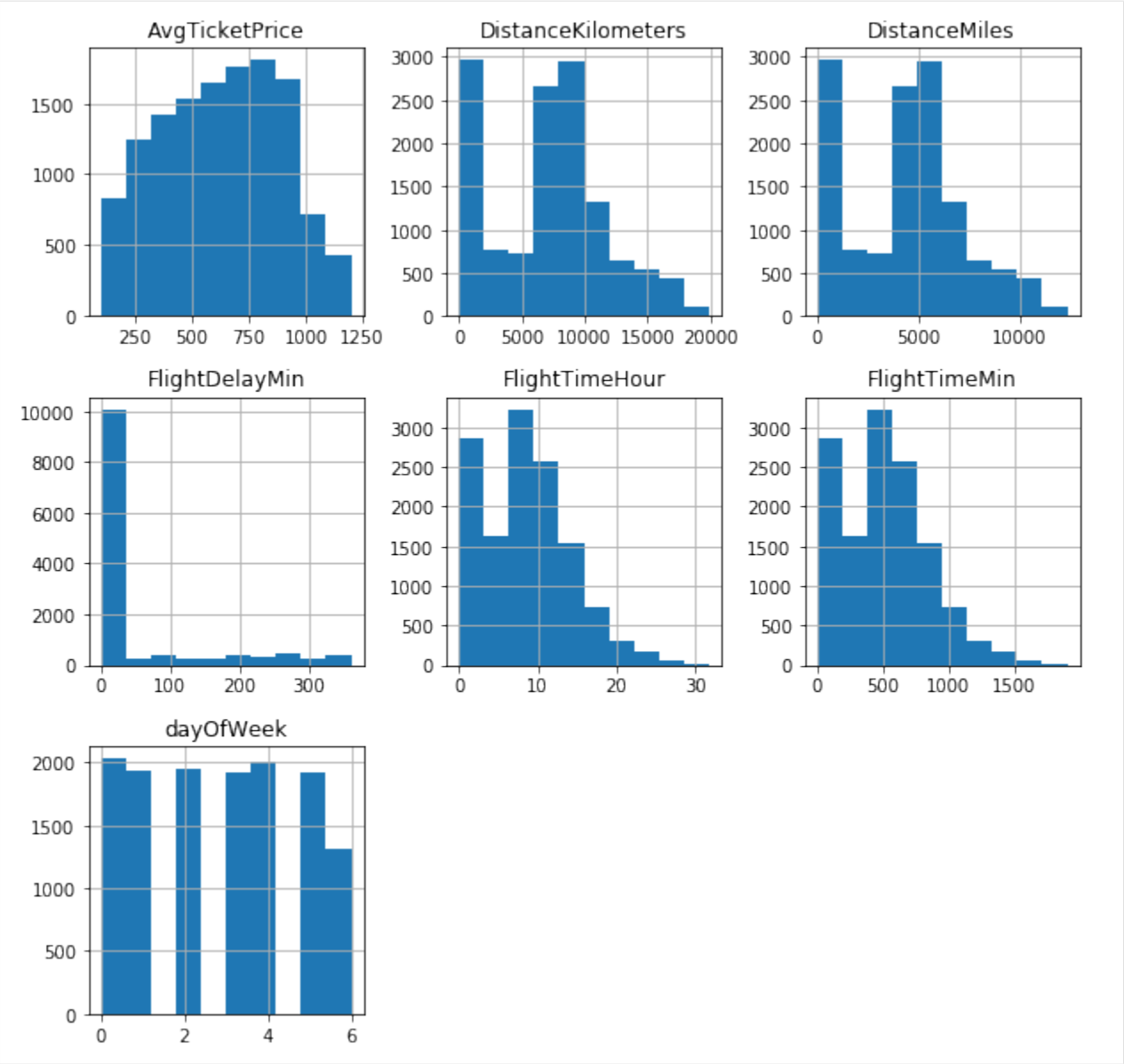
[54]:
```
                Carrier DestRegion  ... dayOfWeek           timestamp
0       Kibana Airlines      SE-BD  ...         0 2018-01-01 00:00:00
1      Logstash Airways      IT-34  ...         0 2018-01-01 18:27:00
2      Logstash Airways      IT-34  ...         0 2018-01-01 17:11:14
3       Kibana Airlines      IT-34  ...         0 2018-01-01 10:33:28
4       Kibana Airlines      SE-BD  ...         0 2018-01-01 05:13:00
...                 ...        ...  ...       ...                 ...
13054  Logstash Airways      SE-BD  ...         6 2018-02-11 20:42:25
13055  Logstash Airways      CH-ZH  ...         6 2018-02-11 01:41:57
13056  Logstash Airways     RU-AMU  ...         6 2018-02-11 04:09:27
13057          JetBeats      SE-BD  ...         6 2018-02-11 08:28:21
13058          JetBeats      US-DC  ...         6 2018-02-11 14:54:34

[13059 rows x 20 columns]
```

Plotting

[55]:
```
pd_flights.select_dtypes(include=np.number).hist(figsize=[10,10])
plt.show()
```



[56]:
```
ed_flights.select_dtypes(include=np.number).hist(figsize=[10,10])
plt.show()
```



Elasticsearch utilities

[57]:
```
ed_flights2 = ed_flights[(ed_flights.OriginAirportID == 'AMS') & (ed_flights.
↪FlightDelayMin > 60)]
ed_flights2 = ed_flights2[['timestamp', 'OriginAirportID', 'DestAirportID',
↪'FlightDelayMin']]
ed_flights2 = ed_flights2.tail()
```

[58]:
```
print(ed_flights2.info_es())
```

```
index_pattern: flights
Index:
 index_field: _id
 is_source_field: False
```

(continues on next page)

(continued from previous page)

```
Mappings:
 capabilities:
                   es_field_name  is_source es_dtype                    es_date_format ␣
↪      pd_dtype  is_searchable  is_aggregatable  is_scripted aggregatable_es_field_
↪name
timestamp              timestamp       True     date  strict_date_hour_minute_second ␣
↪datetime64[ns]           True             True        False                    ␣
↪timestamp
OriginAirportID  OriginAirportID       True  keyword                              None ␣
↪        object           True             True        False              ␣
↪OriginAirportID
DestAirportID      DestAirportID       True  keyword                              None ␣
↪        object           True             True        False                ␣
↪DestAirportID
FlightDelayMin    FlightDelayMin       True  integer                              None ␣
↪         int64           True             True        False              ␣
↪FlightDelayMin
Operations:
 tasks: [('boolean_filter': ('boolean_filter': {'bool': {'must': [{'term': {
↪'OriginAirportID': 'AMS'}}, {'range': {'FlightDelayMin': {'gt': 60}}}]}})), ('tail':
↪ ('sort_field': '_doc', 'count': 5))]
 size: 5
 sort_params: _doc:desc
 _source: ['timestamp', 'OriginAirportID', 'DestAirportID', 'FlightDelayMin']
 body: {'query': {'bool': {'must': [{'term': {'OriginAirportID': 'AMS'}}, {'range': {
↪'FlightDelayMin': {'gt': 60}}}]}}}
 post_processing: [('sort_index')]
```

[1]:
```
import eland as ed
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Fix console size for consistent test results
from eland.conftest import *
```

4.2 Online Retail Analysis

4.2.1 Getting Started

To get started, let's create an `eland.DataFrame` by reading a csv file. This creates and populates the `online-retail` index in the local Elasticsearch cluster.

[2]:
```
df = ed.read_csv("data/online-retail.csv.gz",
                 es_client='localhost',
                 es_dest_index='online-retail',
                 es_if_exists='replace',
                 es_dropna=True,
                 es_refresh=True,
                 compression='gzip',
                 index_col=0)
```

Here we see that the `"_id"` field was used to index our data frame.

[3]:
```
df.index.index_field
```

[3]:
```
'_id'
```

Next, we can check which field from elasticsearch are available to our eland data frame. `columns` is available as a parameter when instantiating the data frame which allows one to choose only a subset of fields from your index to be included in the data frame. Since we didn't set this parameter, we have access to all fields.

[4]:
```
df.columns
```

[4]:
```
Index(['Country', 'CustomerID', 'Description', 'InvoiceDate', 'InvoiceNo', 'Quantity',
 ↪ 'StockCode',
       'UnitPrice'],
      dtype='object')
```

Now, let's see the data types of our fields. Running `df.dtypes`, we can see that elasticsearch field types are mapped to pandas field types.

[5]:
```
df.dtypes
```

[5]:
```
Country          object
CustomerID      float64
Description      object
InvoiceDate      object
InvoiceNo        object
Quantity          int64
StockCode        object
UnitPrice       float64
dtype: object
```

We also offer a `.info_es()` data frame method that shows all info about the underlying index. It also contains information about operations being passed from data frame methods to elasticsearch. More on this later.

[6]:
```
print(df.info_es())
```

```
index_pattern: online-retail
Index:
 index_field: _id
 is_source_field: False
Mappings:
 capabilities:
             es_field_name  is_source es_dtype es_date_format pd_dtype  is_searchable ␣
↪is_aggregatable  is_scripted aggregatable_es_field_name
Country           Country       True  keyword           None   object           True ␣
↪           True        False                    Country
CustomerID     CustomerID       True   double           None  float64           True ␣
↪           True        False                 CustomerID
Description   Description       True  keyword           None   object           True ␣
↪           True        False                Description
InvoiceDate   InvoiceDate       True  keyword           None   object           True ␣
↪           True        False                InvoiceDate
InvoiceNo       InvoiceNo       True  keyword           None   object           True ␣
↪           True        False                  InvoiceNo
Quantity         Quantity       True     long           None    int64           True ␣
↪           True        False                   Quantity
StockCode       StockCode       True  keyword           None   object           True ␣
↪           True        False                  StockCode
UnitPrice       UnitPrice       True   double           None  float64           True ␣
↪           True        False                  UnitPrice
```

(continues on next page)

(continued from previous page)

```
Operations:
 tasks: []
 size: None
 sort_params: None
 _source: ['Country', 'CustomerID', 'Description', 'InvoiceDate', 'InvoiceNo',
↪'Quantity', 'StockCode', 'UnitPrice']
 body: {}
 post_processing: []
```

4.2.2 Selecting and Indexing Data

Now that we understand how to create a data frame and get access to it's underlying attributes, let's see how we can select subsets of our data.

head and tail

much like pandas, eland data frames offer `.head(n)` and `.tail(n)` methods that return the first and last n rows, respectively.

[7]:
```
df.head(2)
```

[7]:
```
             Country  CustomerID  ... StockCode UnitPrice
1000  United Kingdom     14729.0  ...     21123      1.25
1001  United Kingdom     14729.0  ...     21124      1.25

[2 rows x 8 columns]
```

[8]:
```
print(df.tail(2).head(2).tail(2).info_es())
```

```
index_pattern: online-retail
Index:
 index_field: _id
 is_source_field: False
Mappings:
 capabilities:
            es_field_name  is_source es_dtype es_date_format pd_dtype  is_searchable ␣
↪is_aggregatable  is_scripted aggregatable_es_field_name
Country           Country       True  keyword           None   object           True ␣
↪           True        False                    Country
CustomerID     CustomerID       True   double           None  float64           True ␣
↪           True        False                 CustomerID
Description   Description       True  keyword           None   object           True ␣
↪           True        False                Description
InvoiceDate   InvoiceDate       True  keyword           None   object           True ␣
↪           True        False                InvoiceDate
InvoiceNo       InvoiceNo       True  keyword           None   object           True ␣
↪           True        False                  InvoiceNo
Quantity         Quantity       True     long           None    int64           True ␣
↪           True        False                   Quantity
StockCode       StockCode       True  keyword           None   object           True ␣
↪           True        False                  StockCode
UnitPrice       UnitPrice       True   double           None  float64           True ␣
↪           True        False                  UnitPrice
```

(continues on next page)

(continued from previous page)

```
Operations:
 tasks: [('tail': ('sort_field': '_doc', 'count': 2)), ('head': ('sort_field': '_doc',
↪ 'count': 2)), ('tail': ('sort_field': '_doc', 'count': 2))]
 size: 2
 sort_params: _doc:desc
 _source: ['Country', 'CustomerID', 'Description', 'InvoiceDate', 'InvoiceNo',
↪'Quantity', 'StockCode', 'UnitPrice']
 body: {}
 post_processing: [('sort_index'), ('head': ('count': 2)), ('tail': ('count': 2))]
```

[9]:
```
df.tail(2)
```

[9]:
```
              Country  CustomerID  ... StockCode UnitPrice
14998  United Kingdom     17419.0  ...     21773      1.25
14999  United Kingdom     17419.0  ...     22149      2.10

[2 rows x 8 columns]
```

selecting columns

you can also pass a list of columns to select columns from the data frame in a specified order.

[10]:
```
df[['Country', 'InvoiceDate']].head(5)
```

[10]:
```
             Country          InvoiceDate
1000  United Kingdom  2010-12-01 12:43:00
1001  United Kingdom  2010-12-01 12:43:00
1002  United Kingdom  2010-12-01 12:43:00
1003  United Kingdom  2010-12-01 12:43:00
1004  United Kingdom  2010-12-01 12:43:00

[5 rows x 2 columns]
```

Boolean Indexing

we also allow you to filter the data frame using boolean indexing. Under the hood, a boolean index maps to a `terms` query that is then passed to elasticsearch to filter the index.

[11]:
```
# the construction of a boolean vector maps directly to an elasticsearch query
print(df['Country']=='Germany')
df[(df['Country']=='Germany')].head(5)
```

```
{'term': {'Country': 'Germany'}}
```

[11]:
```
      Country  CustomerID  ... StockCode UnitPrice
1109  Germany     12662.0  ...     22809      2.95
1110  Germany     12662.0  ...     84347      2.55
1111  Germany     12662.0  ...     84945      0.85
1112  Germany     12662.0  ...     22242      1.65
1113  Germany     12662.0  ...     22244      1.95

[5 rows x 8 columns]
```

we can also filter the data frame using a list of values.

[12]:
```
print(df['Country'].isin(['Germany', 'United States']))
df[df['Country'].isin(['Germany', 'United Kingdom'])].head(5)
```

```
{'terms': {'Country': ['Germany', 'United States']}}
```

[12]:
```
             Country  CustomerID  ... StockCode UnitPrice
1000  United Kingdom     14729.0  ...     21123      1.25
1001  United Kingdom     14729.0  ...     21124      1.25
1002  United Kingdom     14729.0  ...     21122      1.25
1003  United Kingdom     14729.0  ...     84378      1.25
1004  United Kingdom     14729.0  ...     21985      0.29

[5 rows x 8 columns]
```

We can also combine boolean vectors to further filter the data frame.

[13]:
```
df[(df['Country']=='Germany') & (df['Quantity']>90)]
```

[13]:
```
Empty DataFrame
Columns: [Country, CustomerID, Description, InvoiceDate, InvoiceNo, Quantity,␣
↪StockCode, UnitPrice]
Index: []

[0 rows x 8 columns]
```

Using this example, let see how eland translates this boolean filter to an elasticsearch `bool` query.

[14]:
```
print(df[(df['Country']=='Germany') & (df['Quantity']>90)].info_es())
```

```
index_pattern: online-retail
Index:
 index_field: _id
 is_source_field: False
Mappings:
 capabilities:
            es_field_name  is_source es_dtype es_date_format pd_dtype  is_searchable ␣
↪is_aggregatable  is_scripted aggregatable_es_field_name
Country           Country       True  keyword           None   object           True ␣
↪           True        False                   Country
CustomerID     CustomerID       True   double           None  float64           True ␣
↪           True        False                CustomerID
Description   Description       True  keyword           None   object           True ␣
↪           True        False               Description
InvoiceDate   InvoiceDate       True  keyword           None   object           True ␣
↪           True        False               InvoiceDate
InvoiceNo       InvoiceNo       True  keyword           None   object           True ␣
↪           True        False                 InvoiceNo
Quantity         Quantity       True     long           None    int64           True ␣
↪           True        False                  Quantity
StockCode       StockCode       True  keyword           None   object           True ␣
↪           True        False                 StockCode
UnitPrice       UnitPrice       True   double           None  float64           True ␣
↪           True        False                 UnitPrice
Operations:
 tasks: [('boolean_filter': ('boolean_filter': {'bool': {'must': [{'term': {'Country':
↪ 'Germany'}}, {'range': {'Quantity': {'gt': 90}}}]}}))]
 size: None
 sort_params: None
 _source: ['Country', 'CustomerID', 'Description', 'InvoiceDate', 'InvoiceNo',
↪'Quantity', 'StockCode', 'UnitPrice']
```

(continues on next page)

(continued from previous page)

```
body: {'query': {'bool': {'must': [{'term': {'Country': 'Germany'}}, {'range': {
↪'Quantity': {'gt': 90}}}]}}}
post_processing: []
```

4.2.3 Aggregation and Descriptive Statistics

Let's begin to ask some questions of our data and use eland to get the answers.

How many different countries are there?

[15]:
```
df['Country'].nunique()
```

[15]:
```
16
```

What is the total sum of products ordered?

[16]:
```
df['Quantity'].sum()
```

[16]:
```
111960.0
```

Show me the sum, mean, min, and max of the qunatity and unit_price fields

[17]:
```
df[['Quantity','UnitPrice']].agg(['sum', 'mean', 'max', 'min'])
```

[17]:
```
        Quantity     UnitPrice
sum   111960.000  61548.490000
mean       7.464      4.103233
max     2880.000    950.990000
min    -9360.000      0.000000
```

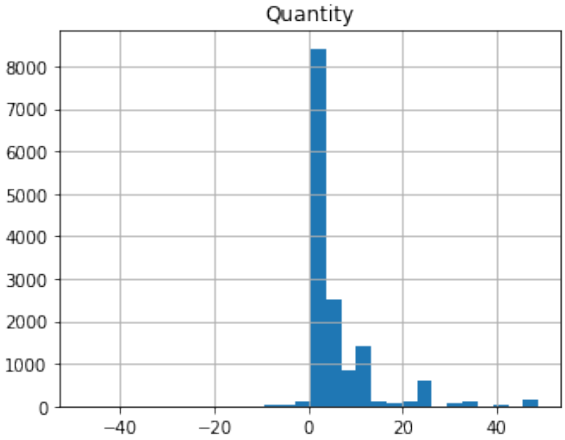
Give me descriptive statistics for the entire data frame

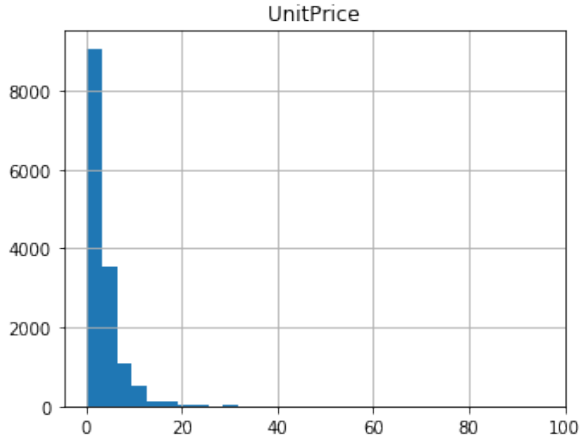
[18]:
```
# NBVAL_IGNORE_OUTPUT
df.describe()
```

[18]:
```
         CustomerID      Quantity     UnitPrice
count  10729.000000  15000.000000  15000.000000
mean   15590.776680      7.464000      4.103233
std     1764.025160     85.924387     20.104873
min    12347.000000  -9360.000000      0.000000
25%    14227.934845      1.000000      1.250000
50%    15669.138235      2.000000      2.510000
75%    17212.690092      6.610262      4.211297
max    18239.000000   2880.000000    950.990000
```

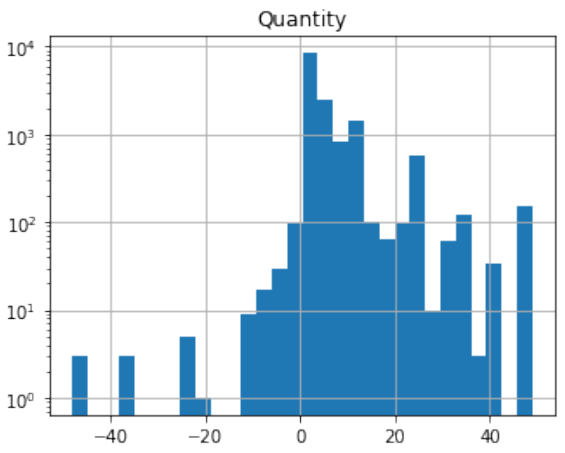
Show me a histogram of numeric columns

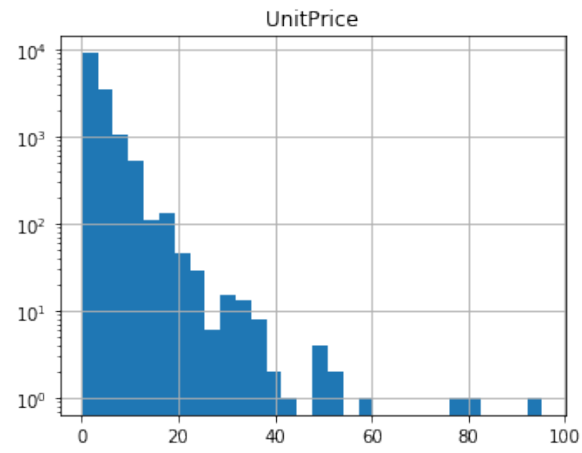
[19]:
```
df[(df['Quantity']>-50) &
   (df['Quantity']<50) &
   (df['UnitPrice']>0) &
   (df['UnitPrice']<100)][['Quantity', 'UnitPrice']].hist(figsize=[12,4], bins=30)
plt.show()
```





[20]:
```
df[(df['Quantity']>-50) &
   (df['Quantity']<50) &
   (df['UnitPrice']>0) &
   (df['UnitPrice']<100)][['Quantity', 'UnitPrice']].hist(figsize=[12,4], bins=30,␣
↪log=True)
plt.show()
```





[21]:
```
df.query('Quantity>50 & UnitPrice<100')
```

[21]:

	Country	CustomerID	...	StockCode	UnitPrice
1228	United Kingdom	15485.0	...	22086	2.55
1237	Norway	12433.0	...	22444	1.06
1286	Norway	12433.0	...	84050	1.25
1293	Norway	12433.0	...	22197	0.85
1333	United Kingdom	18144.0	...	84879	1.69
...	...	...	...	...	...
14784	United Kingdom	15061.0	...	22423	10.95
14785	United Kingdom	15061.0	...	22075	1.45
14788	United Kingdom	15061.0	...	17038	0.07
14974	United Kingdom	14739.0	...	21704	0.72
14980	United Kingdom	14739.0	...	22178	1.06

[258 rows x 8 columns]

4.2.4 Arithmetic Operations

Numeric values

[22]:
```
df['Quantity'].head()
```

[22]:
```
1000     1
1001     1
1002     1
1003     1
1004    12
Name: Quantity, dtype: int64
```

[23]:
```
df['UnitPrice'].head()
```

[23]:
```
1000    1.25
1001    1.25
1002    1.25
1003    1.25
1004    0.29
Name: UnitPrice, dtype: float64
```

[24]:
```
product = df['Quantity'] * df['UnitPrice']
```

[25]:
```
product.head()
```

[25]:
```
1000    1.25
1001    1.25
1002    1.25
1003    1.25
1004    3.48
dtype: float64
```

String concatenation

[26]:
```
df['Country'] + df['StockCode']
```

[26]:
```
1000      United Kingdom21123
1001      United Kingdom21124
1002      United Kingdom21122
1003      United Kingdom84378
1004      United Kingdom21985
                  ...
14995    United Kingdom72349B
14996     United Kingdom72741
14997     United Kingdom22762
14998     United Kingdom21773
14999     United Kingdom22149
Length: 15000, dtype: object
```

- *API reference*
  - *Input/Output*
  - *General utility functions*
  - *DataFrame*
  - *Series*
  - *Index*

Python Module Index

e

Index

S

T

V