Data Science Utils

Release 1.7.1

Feb 28, 2022

Contents:

Data Science Utils extends the Scikit-Learn API and Matplotlib API to provide simple methods that simplify task and visualization over data.

CHAPTER 1

Contributing

Interested in contributing to Data Science Utils? Great! You're welcome, and we would love to have you. We follow the Python Software Foundation Code of Conduct and Matplotlib Usage Guide.

No matter your level of technical skill, you can be helpful. We appreciate bug reports, user testing, feature requests, bug fixes, product enhancements, and documentation improvements.

Thank you for your contributions!

CHAPTER 2

Find a Bug?

Check if there's already an open issue on the topic. If needed, file an issue.

CHAPTER 3

Open Source

Data Science Utils license is MIT License.

3.1 Installation

Data Science Utils is compatible with Python 3.8 or later. The simplest way to install Data Science Utils and its dependencies is from PyPI with pip, Python's preferred package installer:

```
pip install data-science-utils
```

Note that this package is an active project and routinely publishes new releases with more methods. In order to upgrade Data Science Utils to the latest version, use pip as follows:

```
pip install -U data-science-utils
```

Alternatively you can install from source by cloning the repo and running:

```
git clone https://github.com/idanmoradarthas/DataScienceUtils.git
cd DataScienceUtils
python setup.py install
```

Or install using pip from source:

```
pip install git+https://github.com/idanmoradarthas/DataScienceUtils.git
```

If you're using Anaconda, you can install using conda:

```
conda install -c idanmorad data-science-utils
```

3.2 Metrics

The module of metrics contains methods that help to calculate and/or visualize evaluation performance of an algorithm.

3.2.1 Plot Confusion Matrix

metrics.**plot_confusion_matrix**(*y_test: numpy.ndarray, y_pred: numpy.ndarray, labels: List[Union[str, int]], sample_weight: Optional[List[float]] = None, annot_kws=None, cbar=True, cbar_kws=None, **kwargs*) → matplotlib.axes._axes.Axes

Computes and plot confusion matrix, False Positive Rate, False Negative Rate, Accuracy and F1 score of a classification.

Parameters

- **y_test** – array, shape = [n_samples]. Ground truth (correct) target values.
- **y_pred** – array, shape = [n_samples]. Estimated targets as returned by a classifier.
- **labels** – array, shape = [n_classes]. List of labels to index the matrix. This may be used to reorder or select a subset of labels.
- **sample_weight** – array-like of shape = [n_samples], optional

  Sample weights.
- **annot_kws** – dict of key, value mappings, optional

  Keyword arguments for `ax.text`.
- **cbar** – boolean, optional

  Whether to draw a colorbar.
- **cbar_kws** – dict of key, value mappings, optional

  Keyword arguments for `figure.colorbar`
- **kwargs** – other keyword arguments

  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the matrix drawn onto it.

Code Examples

In following examples we are going to use the iris dataset from scikit-learn. so firstly let's import it:

```
import numpy
from sklearn import datasets

IRIS = datasets.load_iris()
RANDOM_STATE = numpy.random.RandomState(0)
```

Next we'll add a small function to add noise:

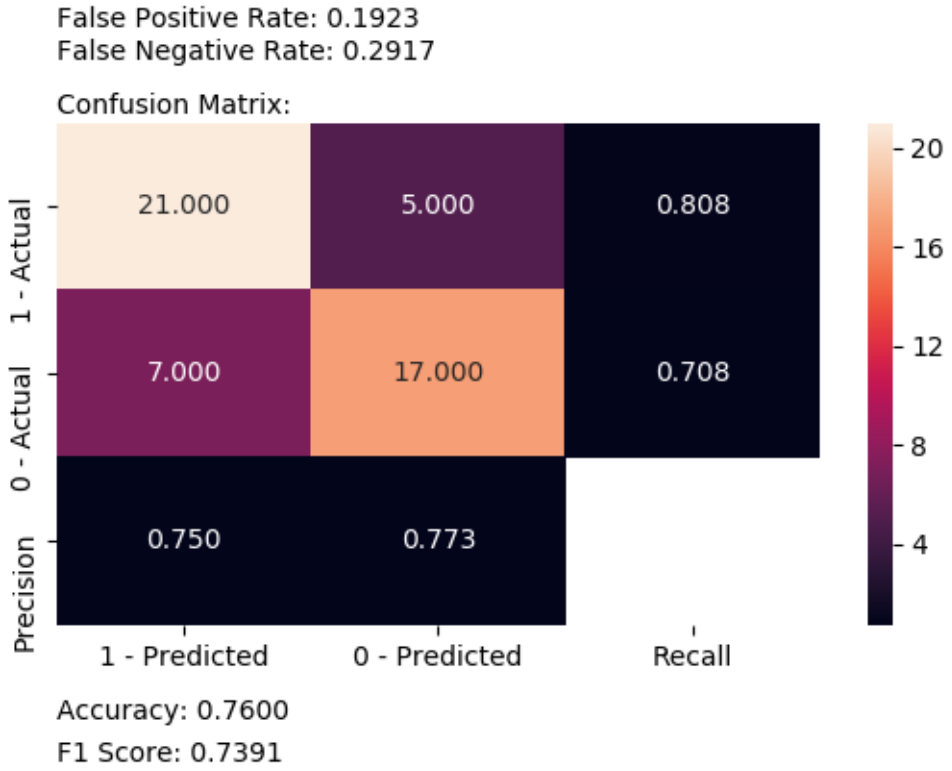
```
def _add_noisy_features(x, random_state):
    n_samples, n_features = x.shape
    return numpy.c_[x, random_state.randn(n_samples, 200 * n_features)]
```

Binary Classification

So We'll use the only first two classes in the iris dataset, build a SVM classifier and evaluate it:

```
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn import svm

from ds_utils.metrics import plot_confusion_matrix


x = IRIS.data
y = IRIS.target

# Add noisy features
x = _add_noisy_features(x, RANDOM_STATE)

# Limit to the two first classes, and split into training and test
x_train, x_test, y_train, y_test = train_test_split(x[y < 2], y[y < 2], test_size=.5,
                                                    random_state=RANDOM_STATE)

# Create a simple classifier
classifier = svm.LinearSVC(random_state=RANDOM_STATE)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)

plot_confusion_matrix(y_test, y_pred, [1, 0])

pyplot.show()
```

And the following image will be shown:



Multi-Label Classification

This time we'll train on all the classes and plot an evaluation:

```
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsRestClassifier
from sklearn import svm

from ds_utils.metrics import plot_confusion_matrix


x = IRIS.data
y = IRIS.target

# Add noisy features
x = _add_noisy_features(x, RANDOM_STATE)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.5, random_
↪state=RANDOM_STATE)

# Create a simple classifier
classifier = OneVsRestClassifier(svm.LinearSVC(random_state=RANDOM_STATE))
classifier.fit(x_train, y_train)
```

(continues on next page)

(continued from previous page)

```
y_pred = classifier.predict(x_test)

plot_confusion_matrix(y_test, y_pred, [0, 1, 2])
pyplot.show()
```

And the following image will be shown:

3.2.2 Plot Metric Growth per Labeled Instances

metrics.**plot_metric_growth_per_labeled_instances**(*X_train: numpy.ndarray, y_train: numpy.ndarray, X_test: numpy.ndarray, y_test: numpy.ndarray, classifiers_dict: Dict[str, sklearn.base.ClassifierMixin], n_samples: Optional[List[int]] = None, quantiles: Optional[List[float]] = [0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.39999999999999997, 0.44999999999999996, 0.49999999999999994, 0.5499999999999999, 0.6, 0.65, 0.7, 0.75, 0.7999999999999999, 0.85, 0.9, 0.95, 1.0], metric: Callable[[numpy.ndarray, numpy.ndarray], float] = <function accuracy_score>, random_state: Union[int, numpy.random.mtrand.RandomState, None] = None, n_jobs: Optional[int] = None, verbose: int = 0, pre_dispatch: Union[int, str, None] = '2*n_jobs', *, ax: Optional[matplotlib.axes._axes.Axes] = None, **kwargs*) → matplotlib.axes._axes.Axes

Receives a train and test sets, and plots given metric change in increasing amount of trained instances.

Parameters

- **X_train** – {array-like or sparse matrix} of shape (n_samples, n_features) The training input samples.
- **y_train** – 1d array-like, or label indicator array / sparse matrix The target values (class labels) as integers or strings.
- **X_test** – {array-like or sparse matrix} of shape (n_samples, n_features) The test or evaluation input samples.
- **y_test** – 1d array-like, or label indicator array / sparse matrix Predicted labels, as returned by a classifier.
- **classifiers_dict** – mapping from classifier name to classifier object.
- **n_samples** – List of numbers of samples for training batches, optional (default=None).
- **quantiles** – List of percentages of samples for training batches, optional (default=[0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1]. Used when n_samples=None.
- **metric** – sklearn.metrics api function which receives y_true and y_pred and returns float.
- **random_state** – int, RandomState instance or None, optional (default=None)

  The seed of the pseudo random number generator to use when shuffling the data.

  - If int, random_state is the seed used by the random number generator;
  - If RandomState instance, random_state is the random number generator;
  - If None, the random number generator is the RandomState instance initiated with seed zero.
- **n_jobs** – int or None, optional (default=None)

  Number of jobs to run in parallel.
  - `None` means 1 unless in a `joblib.parallel_backend` context.
  - `-1` means using all processors.
- **verbose** – integer. Controls the verbosity: the higher, the more messages.
- **pre_dispatch** – int, or string, optional

  Controls the number of jobs that get dispatched during parallel execution. Reducing this number can be useful to avoid an explosion of memory consumption when more jobs get dispatched than CPUs can process. This parameter can be:
  - None, in which case all the jobs are immediately created and spawned. Use this for lightweight and fast-running jobs, to avoid delays due to on-demand spawning of the jobs
  - An int, giving the exact number of total jobs that are spawned
  - A string, giving an expression as a function of n_jobs, as in '2*n_jobs'
- **ax** – Axes object to draw the plot onto, otherwise uses the current Axes.
- **kwargs** – other keyword arguments

  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

Code Example

In this example we'll divide the data into train and test sets, decide on which classifiers we want to measure and plot the results:

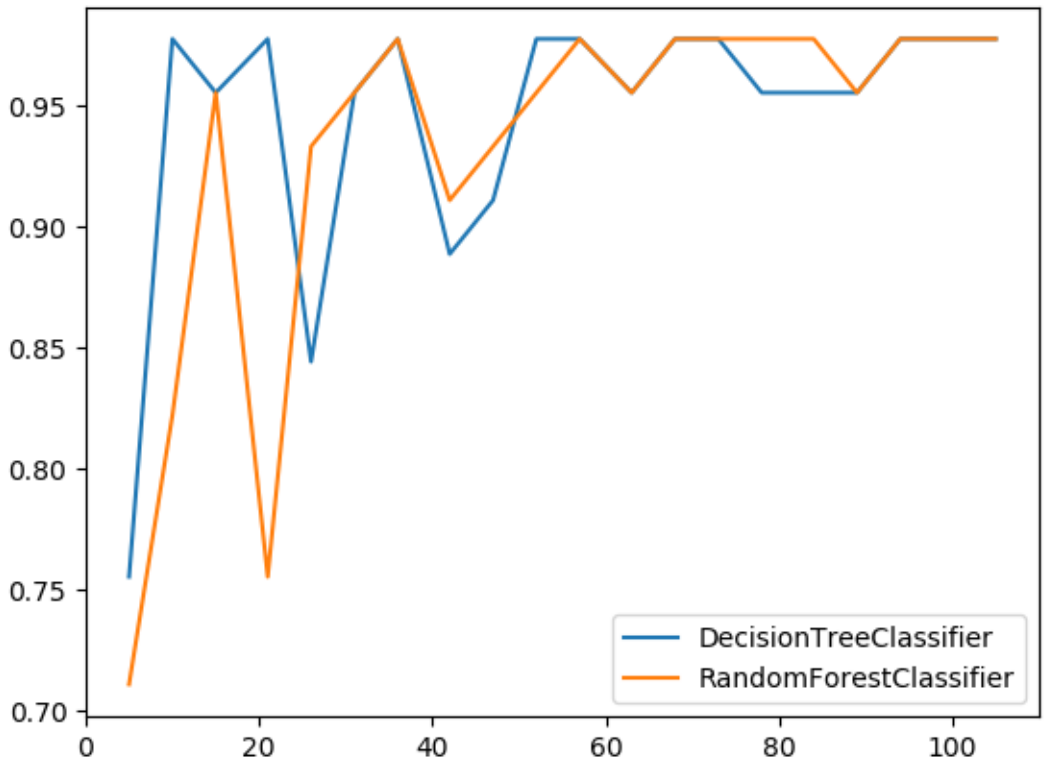
```
from matplotlib import pyplot
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

from ds_utils.metrics import plot_metric_growth_per_labeled_instances


x = IRIS.data
y = IRIS.target

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.3, random_
↪state=0)
plot_metric_growth_per_labeled_instances(x_train, y_train, x_test, y_test,
                                         {"DecisionTreeClassifier":
                                            DecisionTreeClassifier(random_state=0),
                                          "RandomForestClassifier":
```

(continues on next page)

(continued from previous page)

```
                                                    RandomForestClassifier(random_state=0, n_
↪estimators=5)})
pyplot.show()
```

And the following image will be shown:



3.2.3 Visualize Accuracy Grouped by Probability

This method was created due the lack of maintenance of the package EthicalML / xai.

`metrics.`**`visualize_accuracy_grouped_by_probability`**(*y_test: numpy.ndarray, labeled_class: Union[str, int], probabilities: numpy.ndarray, threshold: float = 0.5, display_breakdown: bool = False, bins: Union[int, Sequence[float], pandas.core.indexes.interval.IntervalIndex, None] = None, *, ax: Optional[matplotlib.axes._axes.Axes] = None, **kwargs*) → matplotlib.axes._axes.Axes

Receives test true labels and classifier probabilities predictions, divide and classify the results and finally plots a stacked bar chart with the results.

Parameters

- **y_test** – array, shape = [n_samples]. Ground truth (correct) target values.
- **labeled_class** – the class to enquire for.
- **probabilities** – array, shape = [n_samples]. classifier probabilities for the labeled class.
- **threshold** – the probability threshold for classifying the labeled class.
- **display_breakdown** – if True the results will be displayed as "correct" and "incorrect"; otherwise as "true-positives", "true-negative", "false-positives" and "false-negative"
- **bins** – int, sequence of scalars, or IntervalIndex

  The criteria to bin by.

  – int : Defines the number of equal-width bins in the range of x. The range of x is extended by .1% on each side to include the minimum and maximum values of x.

  – sequence of scalars : Defines the bin edges allowing for non-uniform width. No extension of the range of x is done.

  – IntervalIndex : Defines the exact bins to be used. Note that IntervalIndex for bins must be non-overlapping.

  default: [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
- **ax** – Axes object to draw the plot onto, otherwise uses the current Axes.
- **kwargs** – other keyword arguments

  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

Code Example

The example uses a small sample from of a dataset from kaggle, which a dummy bank provides loans.

Let's see how to use the code:

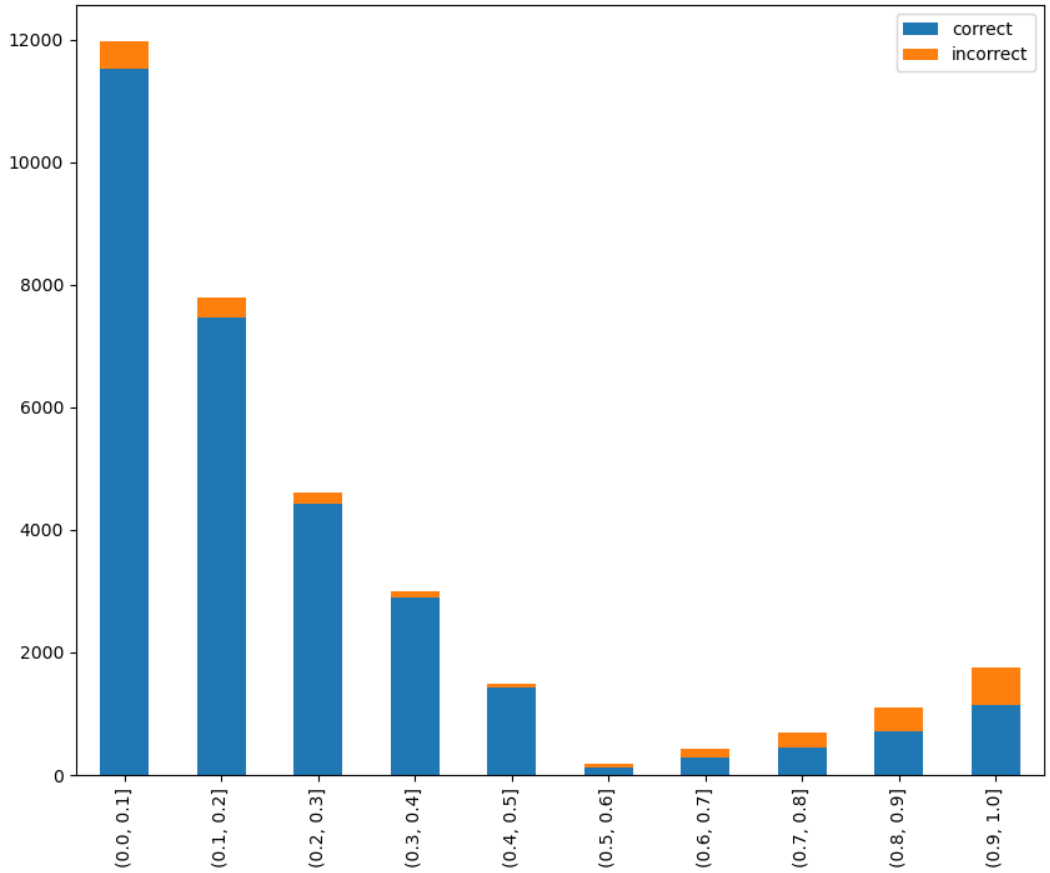
```
from matplotlib import pyplot
from sklearn.ensemble import RandomForestClassifier


from ds_utils.metrics import visualize_accuracy_grouped_by_probability


loan_data = pandas.read_csv(path/to/dataset, encoding="latin1", nrows=11000,
                            parse_dates=["issue_d"])
                            .drop("id", axis=1)
loan_data = loan_data.drop("application_type", axis=1)
loan_data = loan_data.sort_values("issue_d")
loan_data = pandas.get_dummies(loan_data)
train = loan_data.head(int(loan_data.shape[0] * 0.7)).sample(frac=1)
        .reset_index(drop=True).drop("issue_d", axis=1)
test = loan_data.tail(int(loan_data.shape[0] * 0.3)).drop("issue_d", axis=1)

selected_features = ['emp_length_int', 'home_ownership_MORTGAGE', 'home_ownership_RENT
↪',
```

(continues on next page)

(continued from previous page)

```
                    'income_category_Low', 'term_ 36 months', 'purpose_debt_
↪consolidation',
                    'purpose_small_business', 'interest_payments_High']
classifier = RandomForestClassifier(min_samples_leaf=int(train.shape[0] * 0.01),
                                    class_weight="balanced",
                                    n_estimators=1000, random_state=0)
classifier.fit(train[selected_features], train["loan_condition_cat"])

probabilities = classifier.predict_proba(test[selected_features])
visualize_accuracy_grouped_by_probability(test["loan_condition_cat"], 1,␣
↪probabilities[:, 1],
                                          display_breakdown=False)

pyplot.show()
```

And the following image will be shown:



If we chose to display the breakdown:

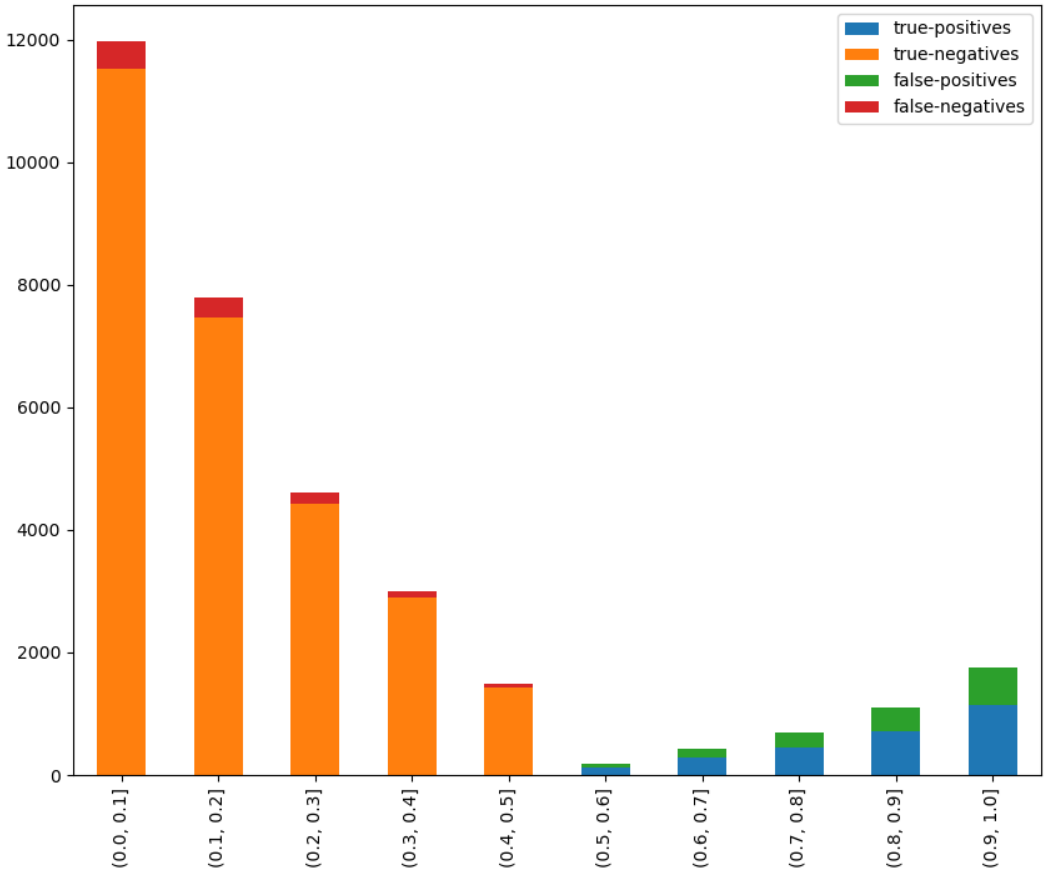
```
visualize_accuracy_grouped_by_probability(test["loan_condition_cat"], 1,␣
↪probabilities[:, 1],
                                          display_breakdown=True)
```

(continues on next page)

(continued from previous page)

```
pyplot.show()
```

And the following image will be shown:



3.3 Preprocess

The module of preprocess contains methods that are processes that could be made to data before training.

3.3.1 Visualize Feature

This method was created due a quick solution to long time calculation of Pandas Profiling. This method give a quick visualization with small latency time.

`preprocess.`**`visualize_feature`**(*series: pandas.core.series.Series*, *remove_na: bool = False*, ***, *ax: Optional[matplotlib.axes._axes.Axes] = None*, ***kwargs*) → matplotlib.axes._axes.Axes

Visualize a feature series:

- If the feature is float then the method plots the distribution plot.

- If the feature is datetime then the method plots a line plot of progression of amount thought time.
- If the feature is object, categorical, boolean or integer then the method plots count plot (histogram).

Parameters

- **series** – the data series.
- **remove_na** – True to ignore NA values when plotting; False otherwise.
- **ax** – Axes in which to draw the plot, otherwise use the currently-active Axes.
- **kwargs** – other keyword arguments

  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

Code Example

The example uses a small sample from of a dataset from kaggle, which a dummy bank provides loans.

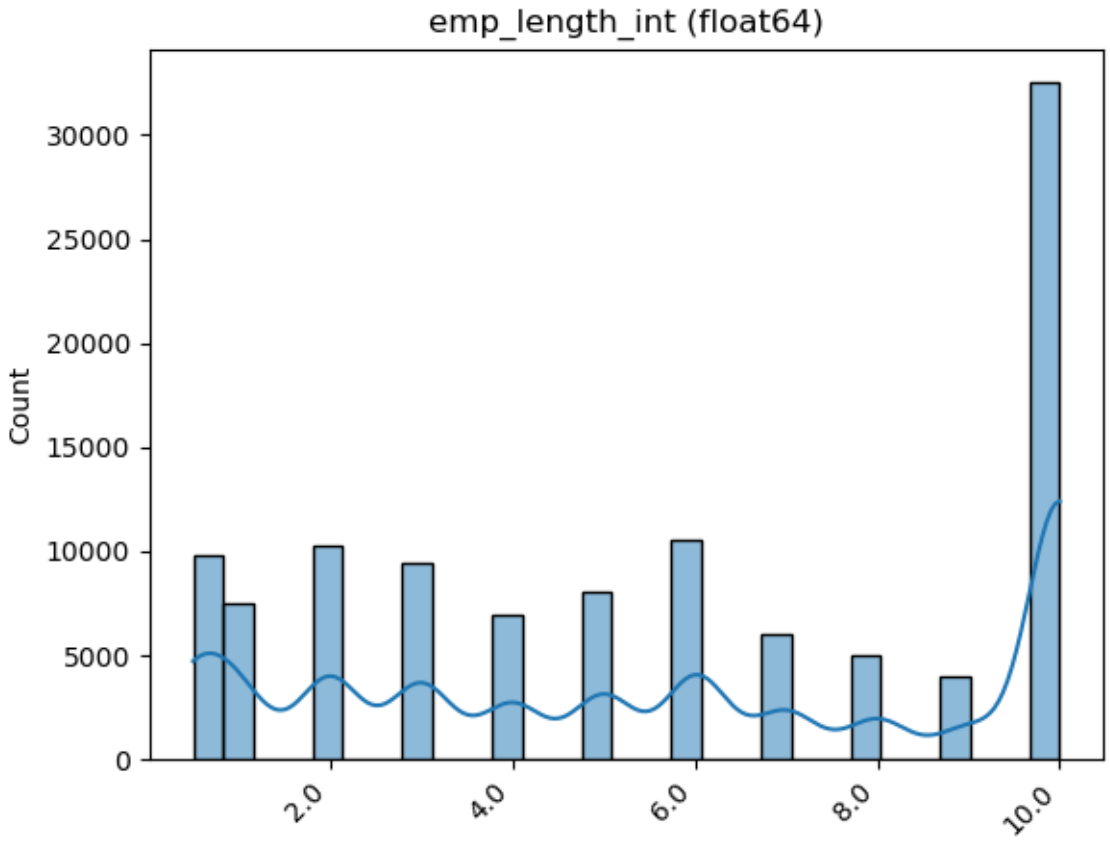
Let's see how to use the code:

```
import pandas

from matplotlib import pyplot

from ds_utils.preprocess import visualize_feature

loan_frame = pandas.read_csv(path/to/dataset, encoding="latin1", nrows=11000,
                             parse_dates=["issue_d"])
loan_frame = loan_frame.drop("id", axis=1)


visualize_features(loan_frame["some feature"])

pyplot.show()
```

For ech different type of feature a different graph will be generated:

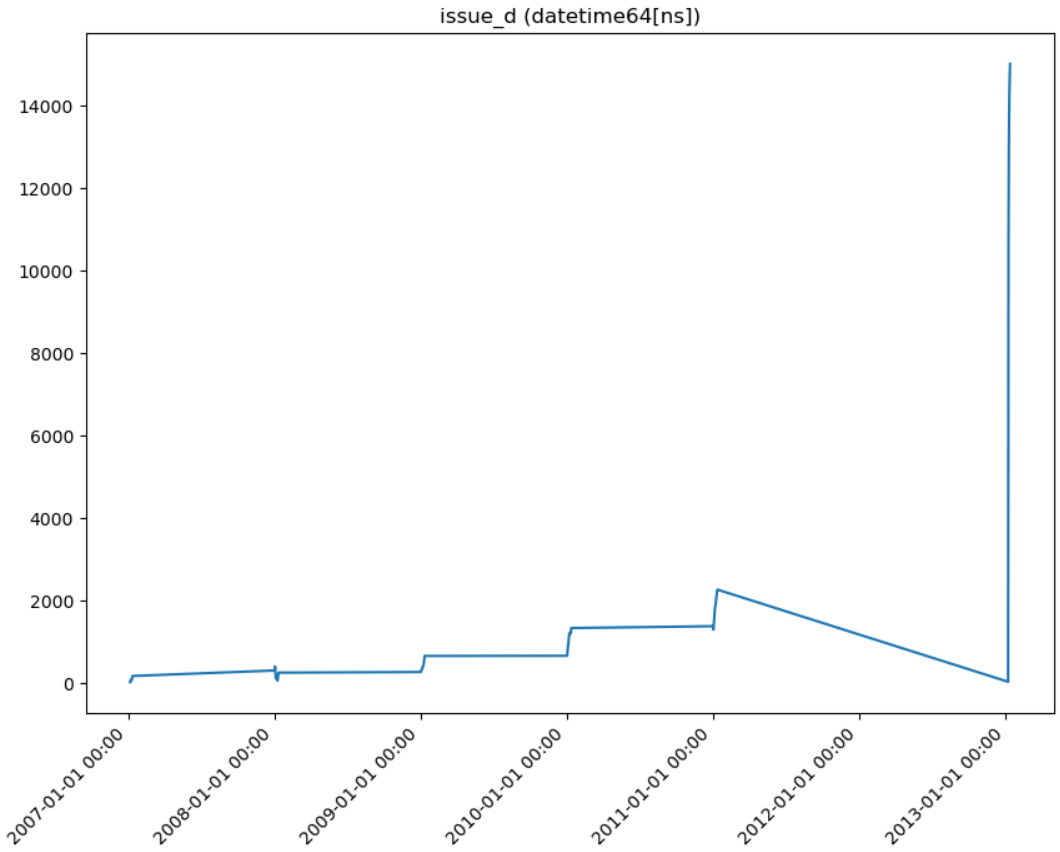
Float

A distribution plot is shown:



Datetime Series

A line plot is shown:

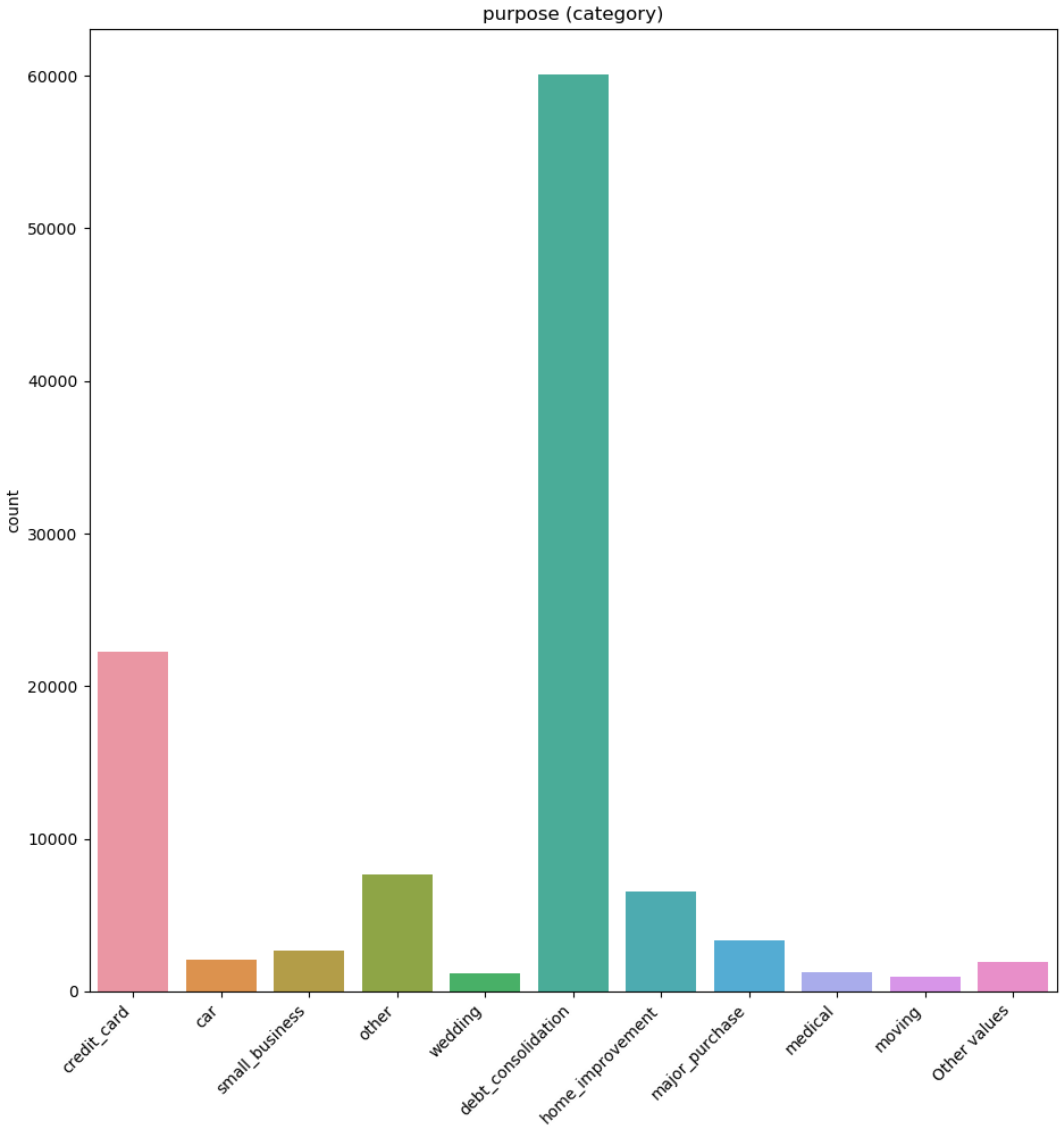


Object, Categorical, Boolean or Integer

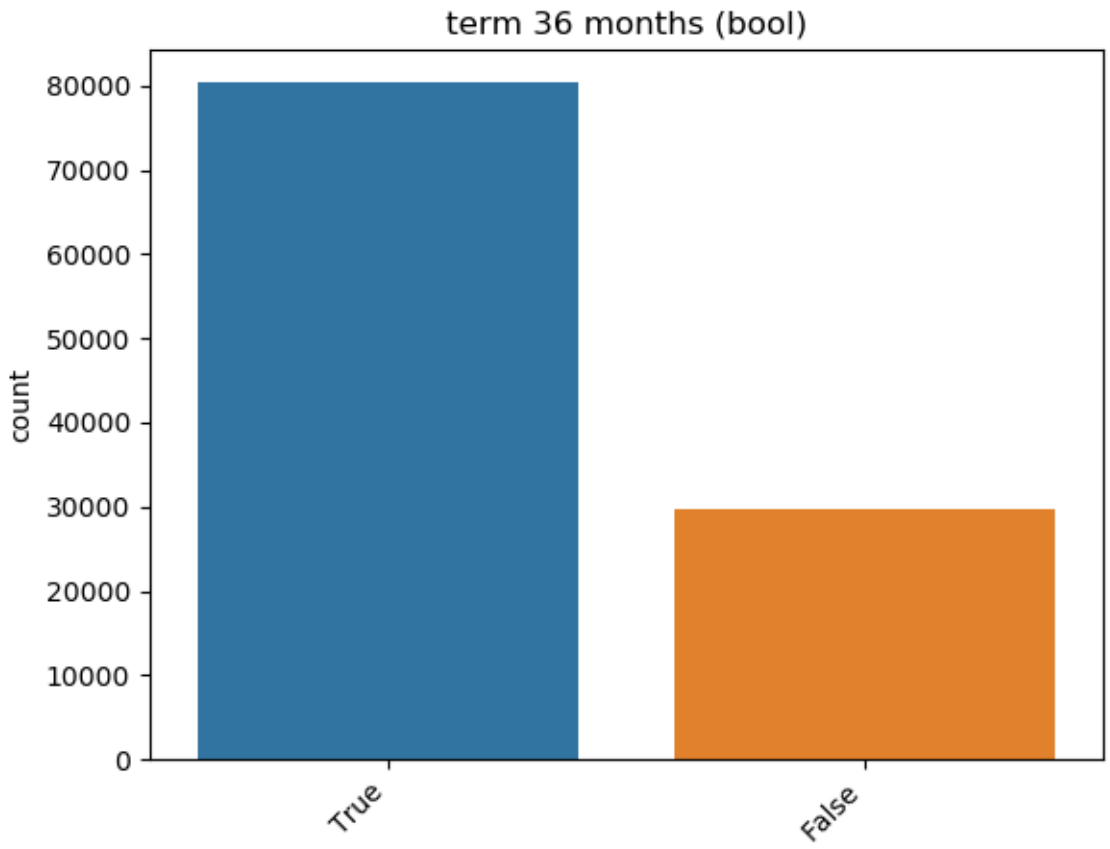
A count plot is shown.

Categorical / Object:

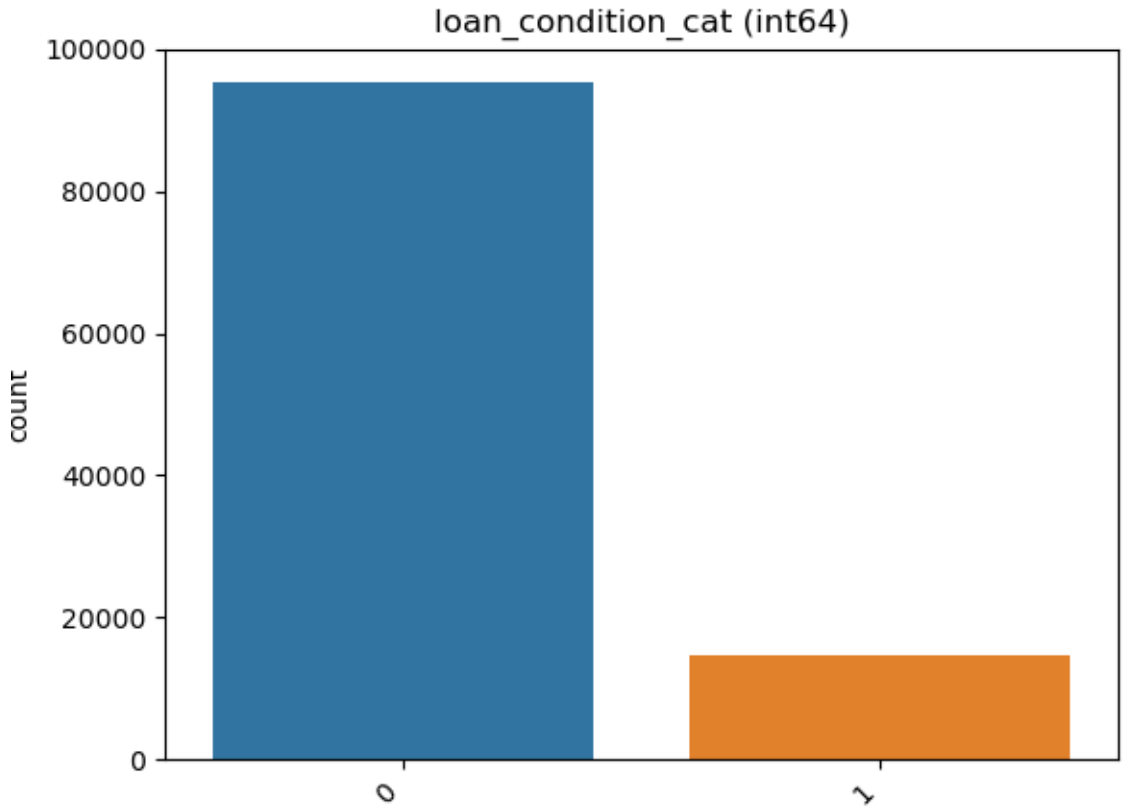
If the categorical / object feature has more than 10 unique values, then the 10 most common values are shown and the other are labeled “Other Values”.


purpose (category)
count
60000
50000
40000
30000
20000
10000
0
credit_card
car
small_business
other
wedding
debt_consolidation
home_improvement
major_purchase
medical
moving
Other values

Boolean:


term 36 months (bool)
count
80000
70000
60000
50000
40000
30000
20000
10000
0
True
False

Integer:



Looping Over All the Features

This code example shows how a loop can be constructed in order to show all of features:

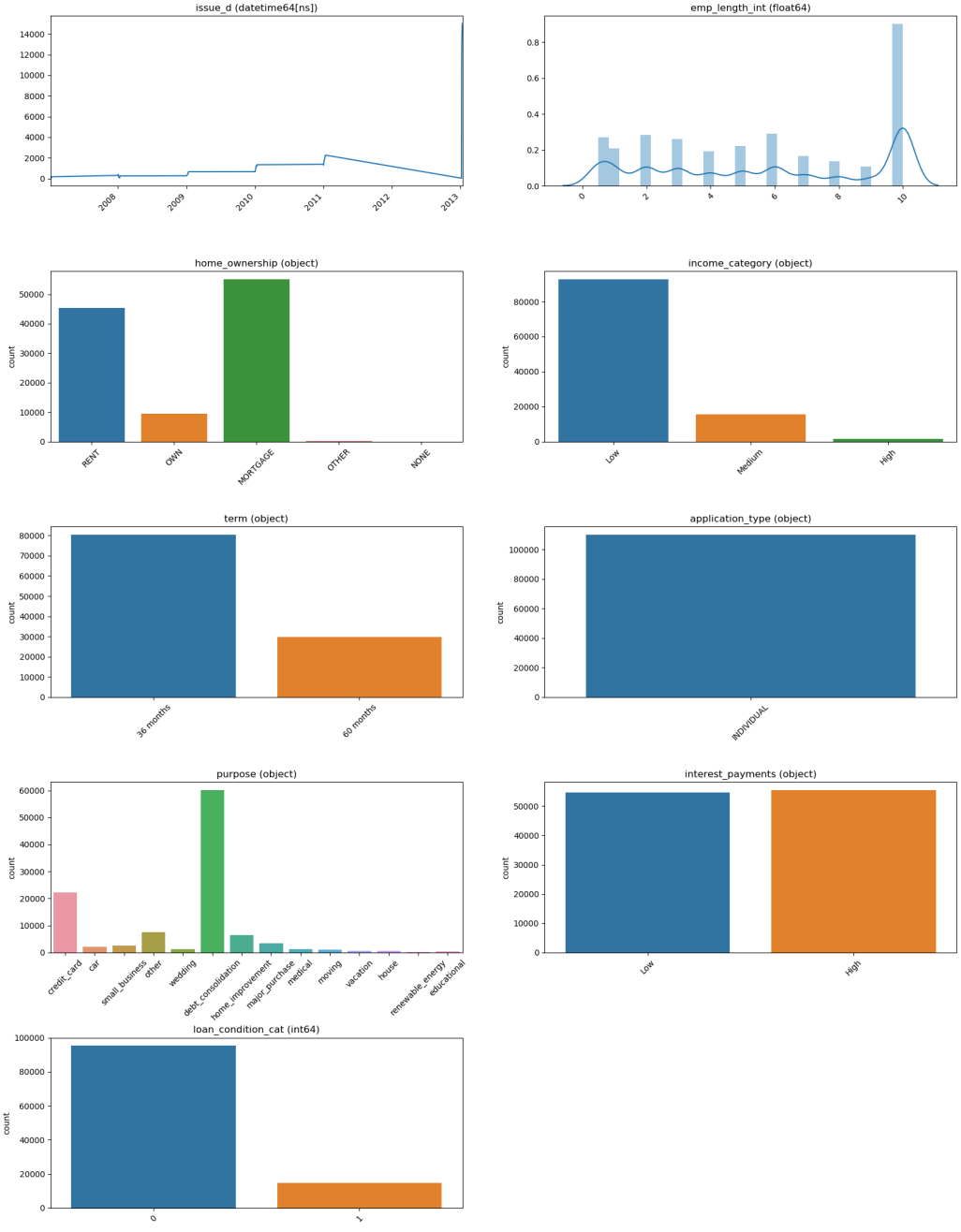
```
import pandas

from matplotlib import pyplot

from ds_utils.preprocess import visualize_feature


loan_frame = pandas.read_csv(path/to/dataset, encoding="latin1", nrows=11000,
                             parse_dates=["issue_d"])
loan_frame = loan_frame.drop("id", axis=1)

figure, axes = pyplot.subplots(5, 2)
axes = axes.flatten()
figure.set_size_inches(18, 30)

features = loan_frame.columns
i = 0

for feature in features:
    visualize_feature(loan_frame[feature], ax=axes[i])
```

(continues on next page)

(continued from previous page)

```
    i += 1

figure.delaxes(axes[9])
pyplot.subplots_adjust(hspace=0.5)
pyplot.show()
```

And the following image will be shown:


issue_d (datetime64[ns])
emp_length_int (float64)
home_ownership (object)
RENT
OWN
MORTGAGE
OTHER
NONE
income_category (object)
Low
Medium
High
term (object)
36 months
60 months
application_type (object)
INDIVIDUAL
purpose (object)
credit_card
car
small_business
other
wedding
debt_consolidation
home_improvement
major_purchase
medical
moving
vacation
house
renewable_energy
educational
interest_payments (object)
Low
High
loan_condition_cat (int64)
count

3.3.2 Get Correlated Features

preprocess.**get_correlated_features**(*data_frame: pandas.core.frame.DataFrame, features: List[str], target_feature: str, threshold: float = 0.95, method: Union[str, Callable] = 'pearson', min_periods: Optional[int] = 1*) → pandas.core.frame.DataFrame

Calculate which features correlated above a threshold and extract a data frame with the correlations and correlation to the target feature.

Parameters

- **data_frame** – the data frame.
- **features** – list of features names.
- **target_feature** – name of target feature.
- **threshold** – the threshold (default 0.95).
- **method** – {'pearson', 'kendall', 'spearman'} or callable

  Method of correlation:

  – pearson : standard correlation coefficient

  – kendall : Kendall Tau correlation coefficient

  – spearman : Spearman rank correlation

  – callable: callable with input two 1d ndarrays and returning a float. Note that the returned matrix from corr will have 1 along the diagonals and will be symmetric regardless of the callable's behavior.
- **min_periods** – Minimum number of observations required per pair of columns to have a valid result. Currently only available for Pearson and Spearman correlation.

Returns data frame with the correlations and correlation to the target feature.

Code Example

The example uses a small sample from of a dataset from kaggle, which a dummy bank provides loans.

Let's see how to use the code:

```
import pandas
from ds_utils.preprocess import get_correlated_features


loan_frame = pandas.read_csv(path/to/dataset, encoding="latin1", nrows=30)
target = "loan_condition_cat"
features = train.columns.drop("loan_condition_cat", "issue_d", "application_type").
↪tolist()
correlations = get_correlated_features(pandas.get_dummies(loan_frame), features,␣
↪target)
print(correlations)
```

The following table will be the output:

level_0	level_1	level_0_level_1_corr	level_0_target_corr	level_1_target_corr
income_category_Low	income_category_Medium	1.0	0.11821656093586504	0.11821656093586504
term_ 36 months	term_ 60 months	1.0	0.11821656093586504	0.11821656093586504
interest_payments_High	interest_payments_Low	1.0	0.11821656093586504	0.11821656093586504

3.3.3 Visualize Correlations

This method was created due a quick solution to long time calculation of Pandas Profiling. This method give a quick visualization with small latency time.

`preprocess.`**`visualize_correlations`**(*data: pandas.core.frame.DataFrame*, *method: Union[str, Callable] = 'pearson'*, *min_periods: Optional[int] = 1*, ***, *ax: Optional[matplotlib.axes._axes.Axes] = None*, ***kwargs*) → matplotlib.axes._axes.Axes

Compute pairwise correlation of columns, excluding NA/null values, and visualize it with heat map. Original code

Parameters

- **`data`** – the data frame, were each feature is a column.
- **`method`** – {'pearson', 'kendall', 'spearman'} or callable

  Method of correlation:

  - pearson : standard correlation coefficient
  - kendall : Kendall Tau correlation coefficient
  - spearman : Spearman rank correlation
  - callable: callable with input two 1d ndarrays and returning a float. Note that the returned matrix from corr will have 1 along the diagonals and will be symmetric regardless of the callable's behavior.
- **`min_periods`** – Minimum number of observations required per pair of columns to have a valid result. Currently only available for Pearson and Spearman correlation.
- **`ax`** – Axes in which to draw the plot, otherwise use the currently-active Axes.
- **`kwargs`** – other keyword arguments

  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

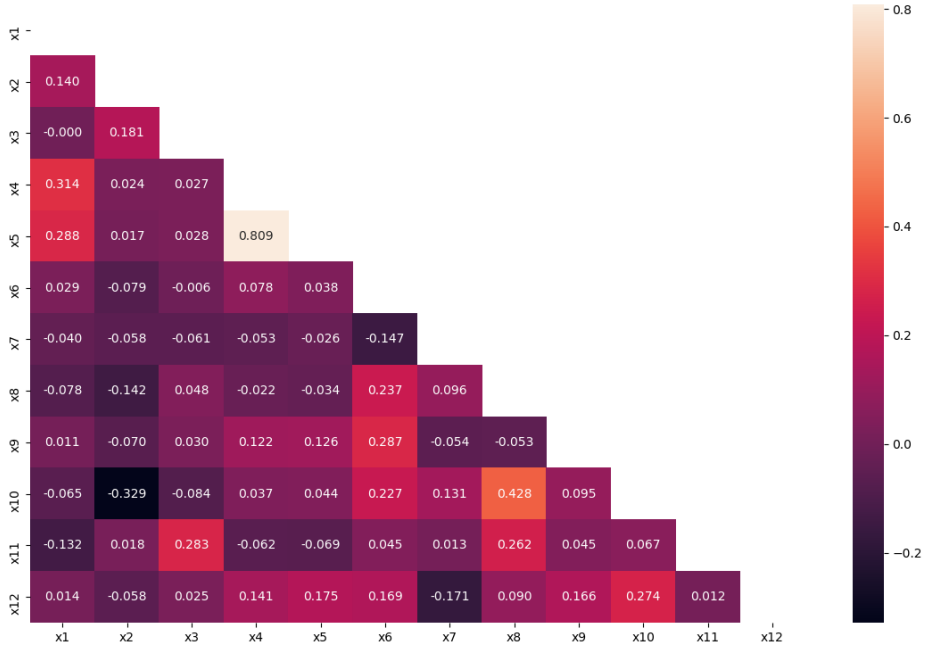
Code Example

For this example I created a dummy data set. You can find the data at the resources directory in the packages tests folder.

Let's see how to use the code:

```
import pandas

from matplotlib import pyplot

from ds_utils.preprocess import visualize_correlations


data_1M = pandas.read_csv(path/to/dataset)
visualize_correlations(data_1M)
pyplot.show()
```

And the following image will be shown:



3.3.4 Plot Correlation Dendrogram

This method was created due the lack of maintenance of the package EthicalML / xai.

preprocess.**plot_correlation_dendrogram**(*data: pandas.core.frame.DataFrame, correlation_method: Union[str, Callable] = 'pearson', min_periods: Optional[int] = 1, cluster_distance_method: Union[str, Callable] = 'average', *, ax: Optional[matplotlib.axes._axes.Axes] = None, **kwargs*) → matplotlib.axes._axes.Axes

Plot dendrogram of a correlation matrix. This consists of a chart that that shows hierarchically the variables that are most correlated by the connecting trees. The closer to the right that the connection is, the more correlated the features are.

Parameters

- **data** – the data frame, were each feature is a column.
- **correlation_method** – {'pearson', 'kendall', 'spearman'} or callable

  Method of correlation:

  – pearson : standard correlation coefficient

  – kendall : Kendall Tau correlation coefficient

  – spearman : Spearman rank correlation

  – callable: callable with input two 1d ndarrays and returning a float. Note that the returned matrix from corr will have 1 along the diagonals and will be symmetric regardless of the callable's behavior.
- **min_periods** – Minimum number of observations required per pair of columns to have a valid result. Currently only available for Pearson and Spearman correlation.
- **cluster_distance_method** – The following are methods for calculating the distance between the newly formed cluster.

  Methods of linkage:

  – single: This is also known as the Nearest Point Algorithm.

  – complete: This is also known by the Farthest Point Algorithm or Voor Hees Algorithm.

  – average:

$$d(u, v) = \sum_{ij} \frac{d(u[i], v[j])}{(|u| * |v|)}$$

  This is also called the UPGMA algorithm.

  – weighted:

$$d(u, v) = (dist(s, v) + dist(t, v))/2$$

  where cluster u was formed with cluster s and t and v is a remaining cluster in the forest. (also called WPGMA)

  – centroid: Euclidean distance between the centroids

  – median: This is also known as the WPGMC algorithm.

  – ward: uses the Ward variance minimization algorithm.

  see scipy.cluster.hierarchy.linkage for more information.
- **ax** – Axes in which to draw the plot, otherwise use the currently-active Axes.
- **kwargs** – other keyword arguments

  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

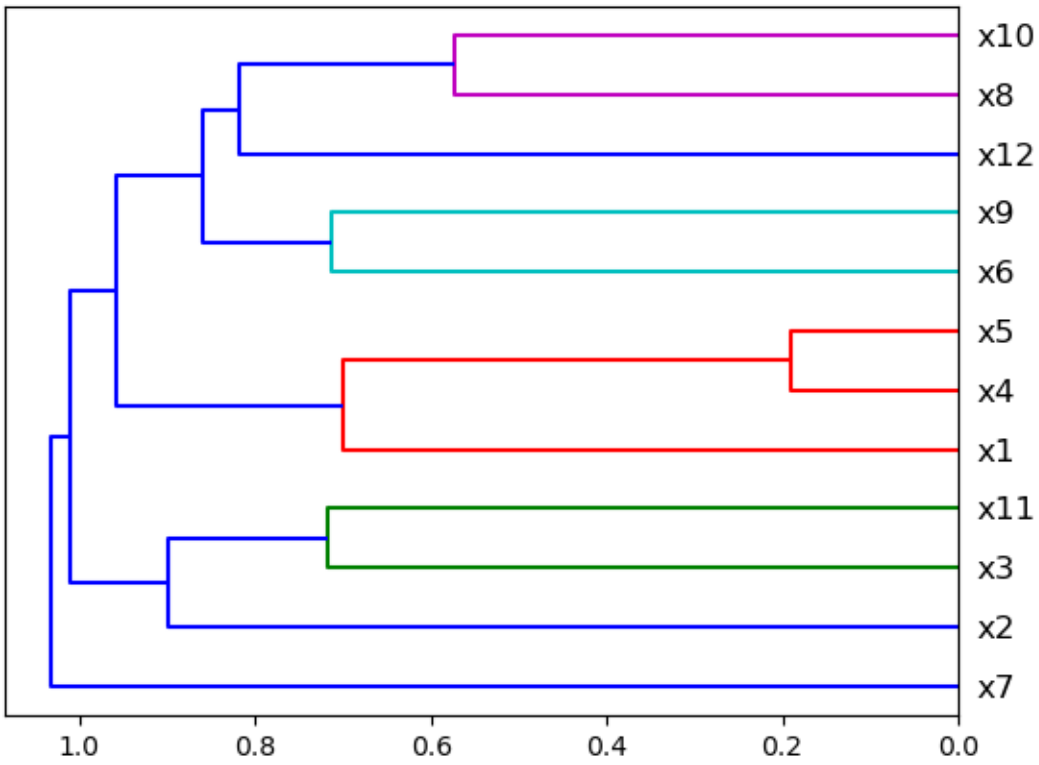
Code Example

For this example I created a dummy data set. You can find the data at the resources directory in the packages tests folder.

Let's see how to use the code:

```
import pandas

from matplotlib import pyplot

from ds_utils.preprocess import plot_correlation_dendrogram


data_1M = pandas.read_csv(path/to/dataset)
plot_correlation_dendrogram(data_1M)
pyplot.show()
```

And the following image will be shown:



3.3.5 Plot Features' Interaction

This method was created due a quick solution to long time calculation of Pandas Profiling. This method give a quick visualization with small latency time.

preprocess.**plot_features_interaction**(*feature_1: str, feature_2: str, data: pandas.core.frame.DataFrame, *, ax: Optional[matplotlib.axes._axes.Axes] = None, **kwargs*) → matplotlib.axes._axes.Axes

Plots the joint distribution between two features:

- If both features are either categorical, boolean or object then the method plots the shared histogram.

- If one feature is either categorical, boolean or object and the other is numeric then the method plots a boxplot chart.
- If one feature is datetime and the other is numeric or datetime then the method plots a line plot graph.
- If one feature is datetime and the other is either categorical, boolean or object the method plots a violin plot (combination of boxplot and kernel density estimate).
- If both features are numeric then the method plots scatter graph.

Parameters

- **feature_1** – the name of the first feature.
- **feature_2** – the name of the second feature.
- **data** – the data frame, were each feature is a column.
- **ax** – Axes in which to draw the plot, otherwise use the currently-active Axes.
- **kwargs** – other keyword arguments

  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

Code Example

For this example I created a dummy data set. You can find the data at the resources directory in the packages tests folder.

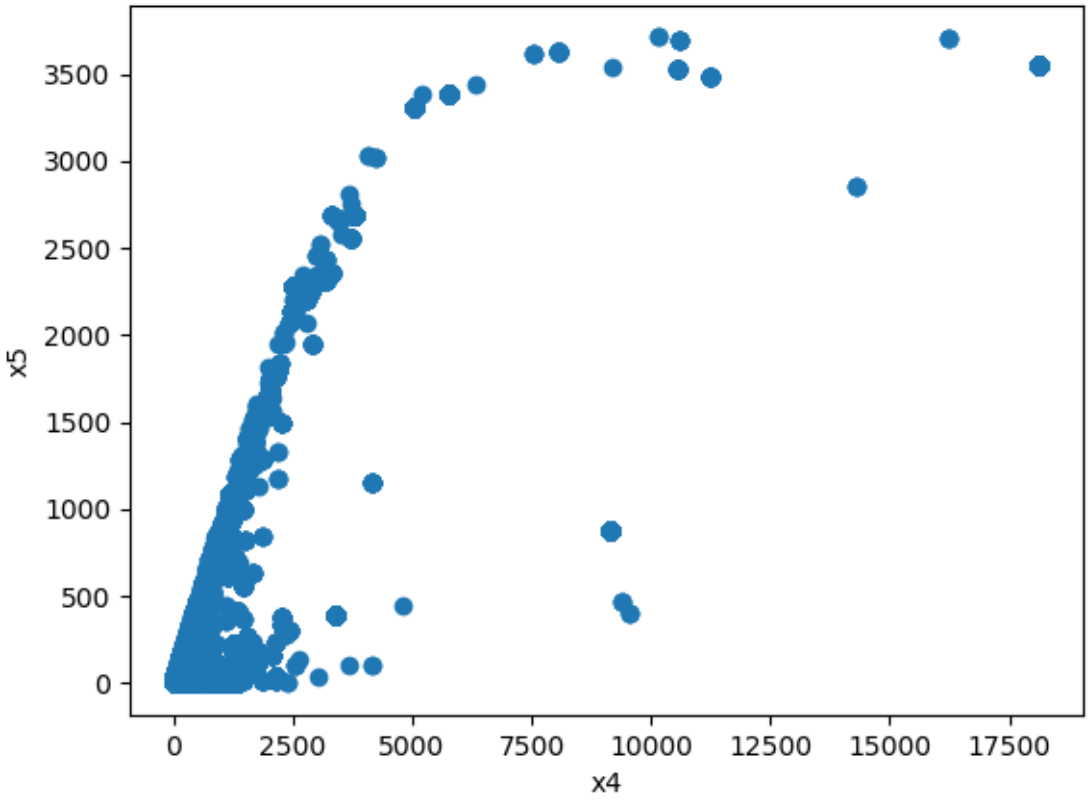
Let's see how to use the code:

```
import pandas

from matplotlib import pyplot

from ds_utils.preprocess import plot_features_interaction


data_1M = pandas.read_csv(path/to/dataset)
plot_features_interaction("x7", "x10", data_1M)
pyplot.show()
```

For each different combination of features types a different plot will be shown:

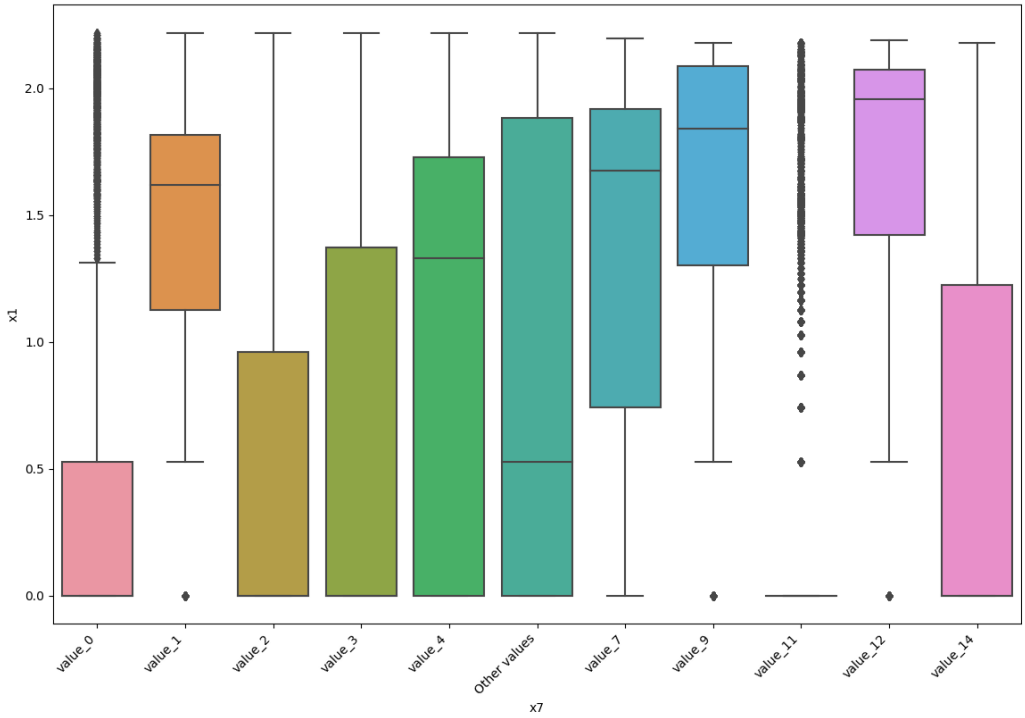
Both Features are Numeric

A scatter plot of the shared distribution is shown:

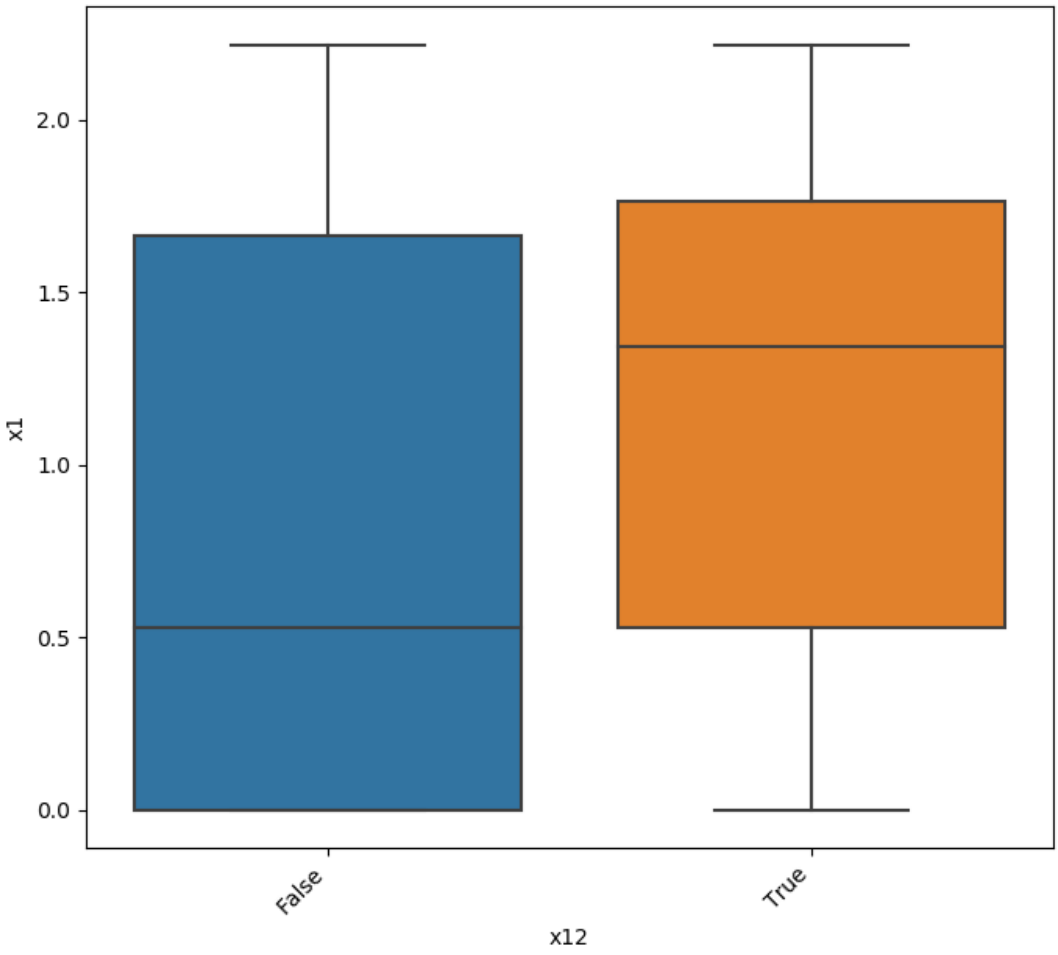


One Feature is Numeric and The Other is Categorical

If one feature is numeric, but the the other is either an `object`, a `category` or a `bool`, then a box plot is shown. In the plot it can be seen for each unique value of the category feature what is the distribution of the numeric feature. If the categorical feature has more than 10 unique values, then the 10 most common values are shown and the other are labeled "Other Values".

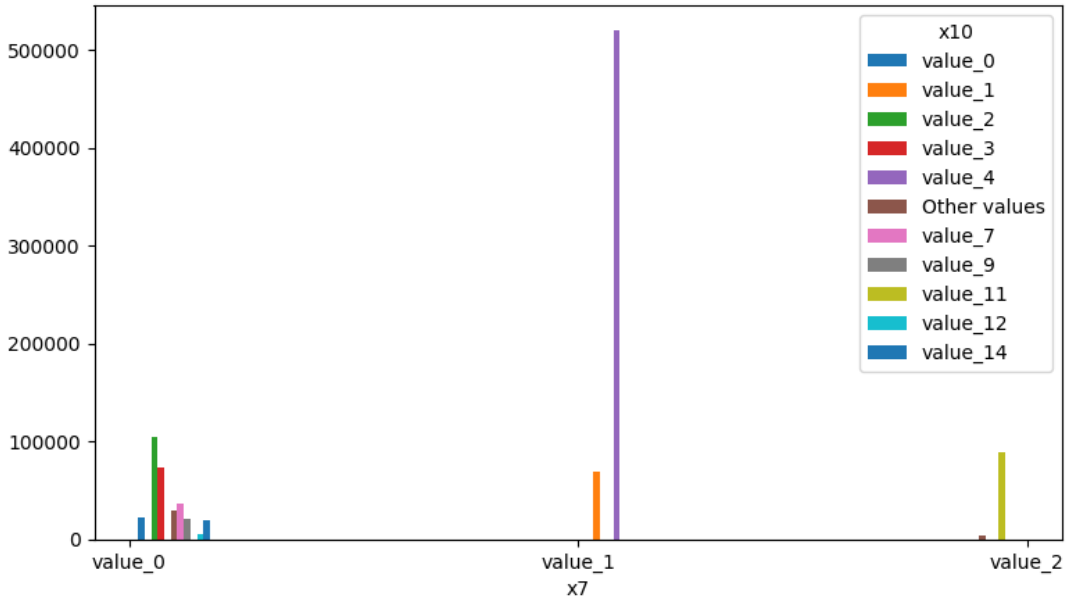


Here is an example for boolean feature plot:



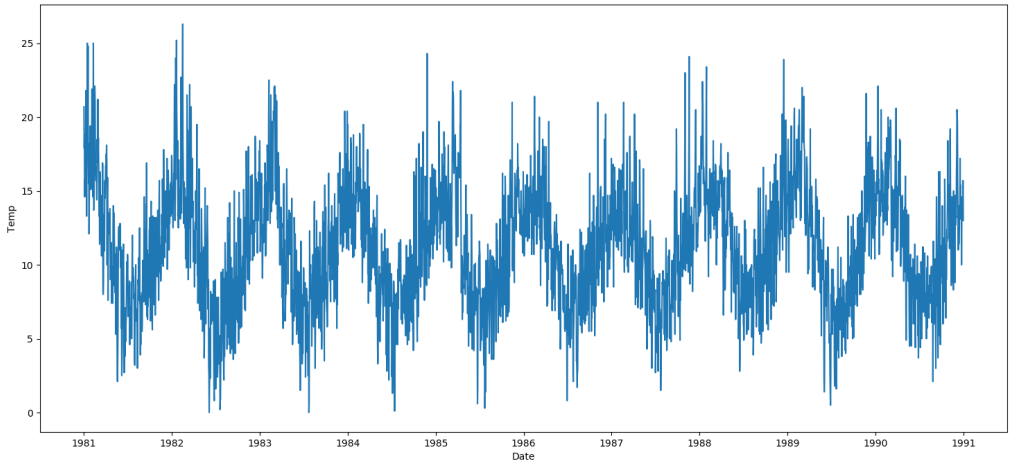
Both Features are Categorical

A shared histogram will be shown. If one or both features have more than 10 unique values, then the 10 most common values are shown and the other are labeled "Other Values".



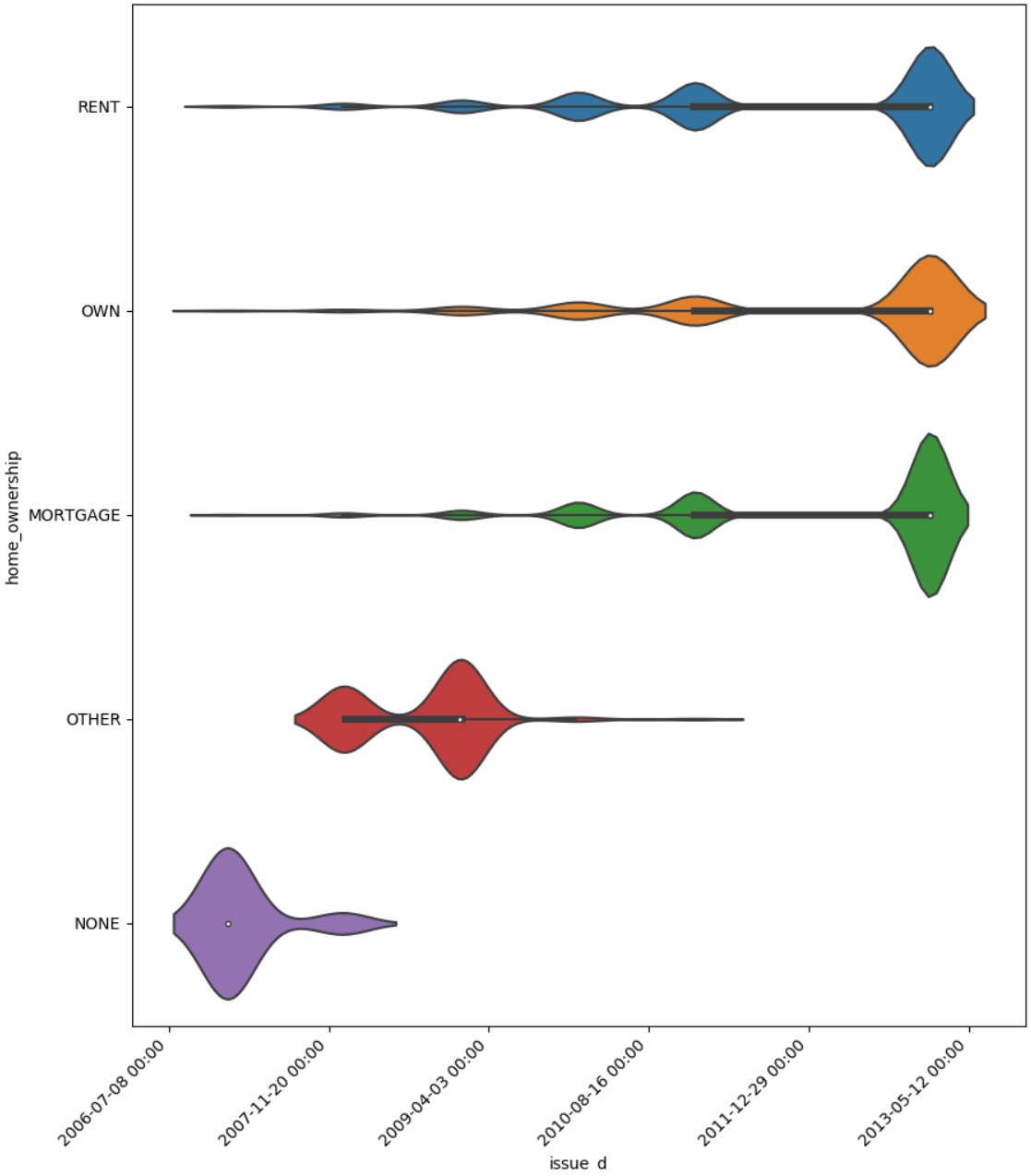
One Feature is Datetime Series and the Other is Numeric or Datetime Series

A line plot where the datetime series is at x axis is shown:

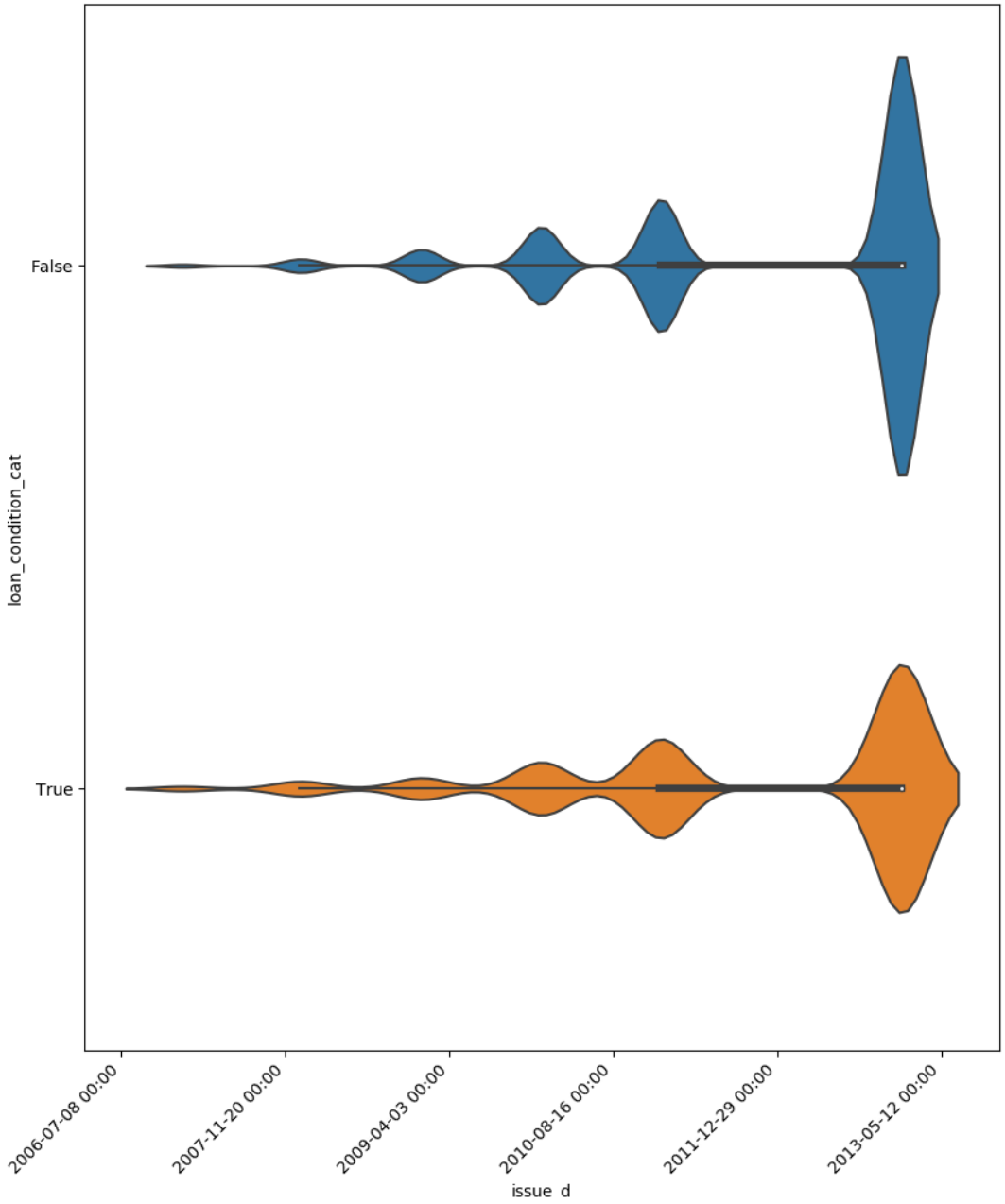


One Feature is Datetime Series and the Other is Categorical

If one feature is datetime series, but the the other is either an `object`, a `category` or a `bool`, then a violin plot is shown. Violin plot is a combination of boxplot and kernel density estimate. If the categorical feature has more than 10 unique values, then the 10 most common values are shown and the other are labeled "Other Values". The datetime series will be at x axis:



Here is an example for boolean feature plot:

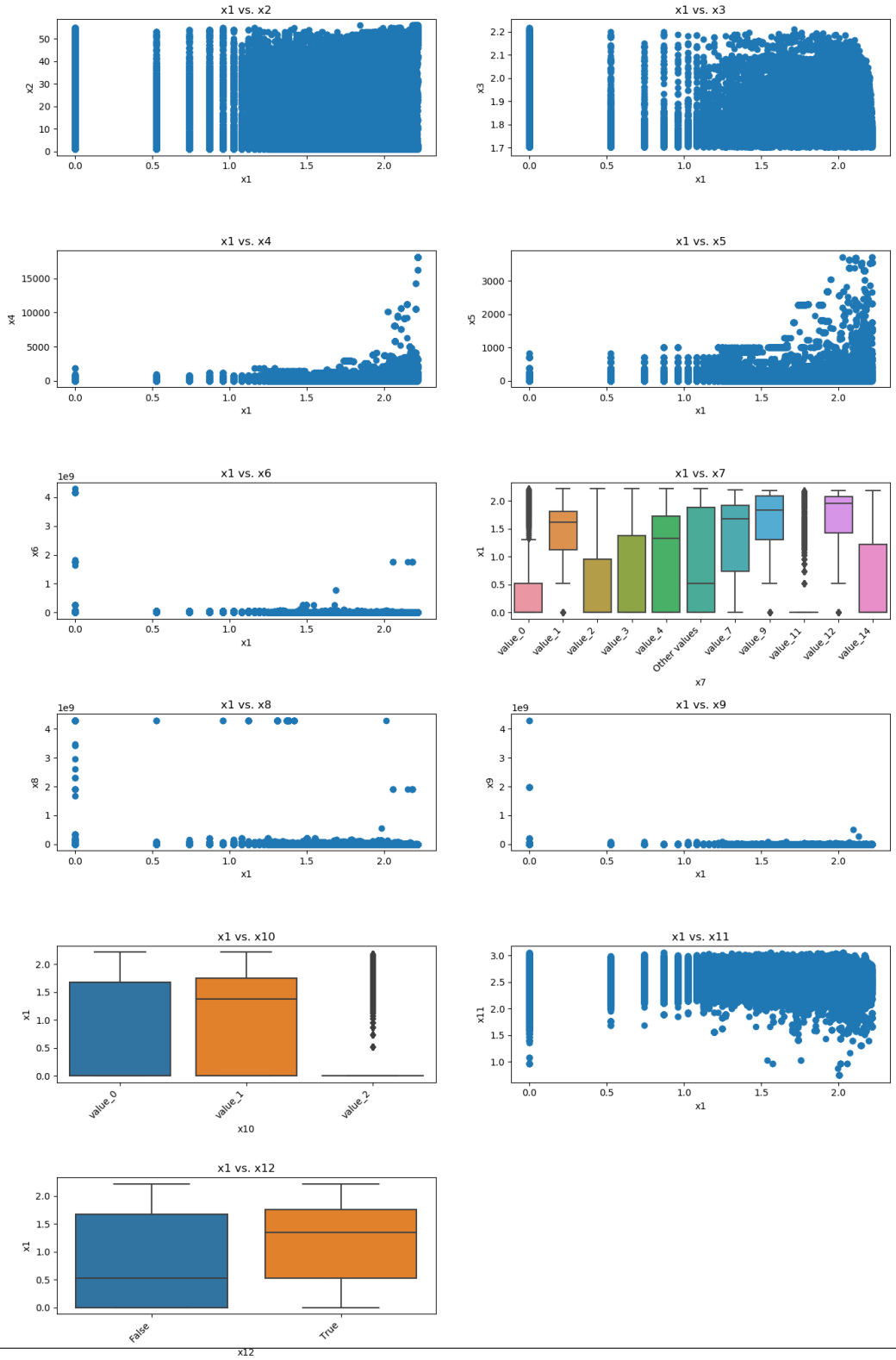


Looping One Feature over The Others

This code example shows how a loop can be constructed in order to show all of one feature relationship with all the others:

```
import pandas

from matplotlib import pyplot

from ds_utils.preprocess import plot_features_interaction


data_1M = pandas.read_csv(path/to/dataset)

figure, axes = pyplot.subplots(6, 2)
axes = axes.flatten()
figure.set_size_inches(16, 25)

feature_1 = "x1"
other_features = ["x2", "x3", "x4", "x5", "x6", "x7", "x8", "x9", "x10", "x11", "x12"]

for i in range(0, len(other_features)):
    axes[i].set_title(f"{feature_1} vs. {other_features[i]}")
    plot_features_interaction(feature_1, other_features[i], data_1M, ax=axes[i])

figure.delaxes(axes[11])
figure.subplots_adjust(hspace=0.7)
pyplot.show()
```

And the following image will be shown:


x1 vs. x2
x1 vs. x3
x1 vs. x4
x1 vs. x5
x1 vs. x6
x1 vs. x7
x1 vs. x8
x1 vs. x9
x1 vs. x10
x1 vs. x11
x1 vs. x12
value_0
value_1
value_2
value_3
value_4
Other values
value_7
value_9
value_11
value_12
value_14
False
True

3.4 Strings

The module of strings contains methods that help manipulate and process strings in a dataframe.

3.4.1 Append Tags to Frame

strings.**append_tags_to_frame**(*X_train: pandas.core.frame.DataFrame, X_test: pandas.core.frame.DataFrame, field_name: str, prefix: Optional[str] = '', max_features: Optional[int] = 500, min_df: Union[int, float] = 1, lowercase=False, tokenizer: Optional[Callable[[str], List[str]]] = <function _tokenize>*) → Tuple[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]

Extracts tags from a given field and append them as dataframe.

Parameters

- **X_train** – Pandas' dataframe with the train features.
- **X_test** – Pandas' dataframe with the test features.
- **field_name** – the feature to parse.
- **prefix** – the given prefix for new tag feature.
- **max_features** – int or None, default=500. max tags names to consider.
- **min_df** – float in range [0.0, 1.0] or int, default=1. When building the tag name set ignore tags that have a document frequency strictly higher than the given threshold (corpus-specific stop words). If float, the parameter represents a proportion of documents, integer absolute counts.
- **lowercase** – boolean, default=False. Convert all characters to lowercase before tokenizing the tag names.
- **tokenizer** – callable or None. Override the string tokenization step while preserving the preprocessing and n-grams generation steps. Default splits by ",", and retain alphanumeric characters with special characters "_", "$" and "-".

Returns the train and test with tags appended.

Code Example

In this example we'll create our own simple dataset that looks like that:

x_train:

article_name	article_tags
1	ds,ml,dl
2	ds,ml

x_test:

article_name	article_tags
3	ds,ml,py

and parse it:

```
import pandas

from ds_utils.strings import append_tags_to_frame


x_train = pandas.DataFrame([{"article_name": "1", "article_tags": "ds,ml,dl"},
                            {"article_name": "2", "article_tags": "ds,ml"}])
x_test = pandas.DataFrame([{"article_name": "3", "article_tags": "ds,ml,py"}])

x_train_with_tags, x_test_with_tags = append_tags_to_frame(x_train, x_test, "article_
↪tags", "tag_")
```

And the following table will be the output for `x_train_with_tags`:

article_name	tag_ds	tag_ml	tag_dl
1	1	1	1
2	1	1	0

And the following table will be the output for `x_test_with_tags`:

article_name	tag_ds	tag_ml	tag_dl
3	1	1	0

3.4.2 Significant Terms

strings.**extract_significant_terms_from_subset**(*data_frame: pandas.core.frame.DataFrame*, *subset_data_frame: pandas.core.frame.DataFrame*, *field_name: str*, *vectorizer: sklearn.feature_extraction.text.CountVectorizer = CountVectorizer(encoding='latin1', max_features=500)*) → pandas.core.series.Series

Returns interesting or unusual occurrences of terms in a subset.

Based on the elasticsearch significant_text aggregation

Parameters

- **data_frame** – the full data set.
- **subset_data_frame** – the subset partition data, with over it the scoring will be calculated. Can a filter by feature or other boolean criteria.
- **field_name** – the feature to parse.
- **vectorizer** – text count vectorizer which converts collection of text to a matrix of token counts. See more info here .

Returns Series of terms with scoring over the subset.

Author Eran Hirsch

Code Example

This method will help extract the significant terms that will differentiate between subset of documents from the full corpus. Let's create a simple corpus and extract significant terms from it:

```
import pandas

from ds_utils.strings import extract_significant_terms_from_subset

corpus = ['This is the first document.', 'This document is the second document.',
          'And this is the third one.', 'Is this the first document?']
data_frame = pandas.DataFrame(corpus, columns=["content"])
# Let's differentiate between the last two documents from the full corpus
subset_data_frame = data_frame[data_frame.index > 1]
terms = extract_significant_terms_from_subset(data_frame, subset_data_frame,
                                              "content")
```

And the following table will be the output for `terms`:

third	one	and	this	the	is	first	document	second
1.0	1.0	1.0	0.67	0.67	0.67	0.5	0.25	0.0

3.5 Unsupervised

The module of unsupervised contains methods that calculate and/or visualize evaluation performance of an unsupervised model.

Mostly inspired by the Interpet Results of Cluster in Google's Machine Learning Crash Course. See more information here

3.5.1 Plot Cluster Cardinality

unsupervised.**plot_cluster_cardinality**(*labels: numpy.ndarray*, ***, *ax: Optional[matplotlib.axes._axes.Axes] = None*, ***kwargs*) → matplotlib.axes._axes.Axes

Cluster cardinality is the number of examples per cluster. This method plots the number of points per cluster as a bar chart.

Parameters

- **labels** – Labels of each point.
- **ax** – Axes object to draw the plot onto, otherwise uses the current Axes.
- **kwargs** – other keyword arguments

  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

In following examples we are going to use the iris dataset from scikit-learn. so firstly let's import it:

```
from sklearn import datasets
```

(continues on next page)

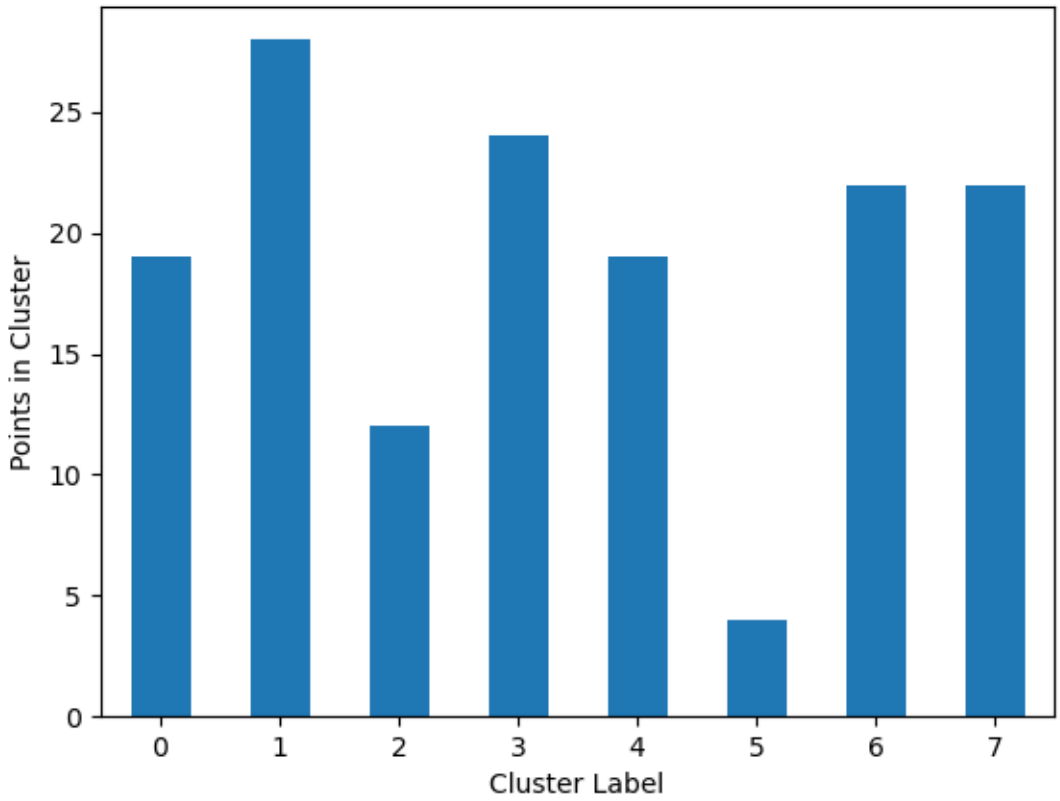
(continued from previous page)

```
iris = datasets.load_iris()
x = iris.data
```

We'll create a simple K-Means algorithm with k=8 and plot how many point goes to each cluster:

```
from matplotlib import pyplot
from sklearn.cluster import KMeans

from ds_utils.unsupervised import plot_cluster_cardinality


estimator = KMeans(n_clusters=8, random_state=42)
estimator.fit(x)

plot_cluster_cardinality(estimator.labels_)

pyplot.show()
```

And the following image will be shown:



3.5.2 Plot Cluster Magnitude

unsupervised.**plot_cluster_magnitude**(*X: numpy.ndarray, labels: numpy.ndarray, cluster_centers: numpy.ndarray, distance_function: Callable[[numpy.ndarray, numpy.ndarray], float], *, ax: Optional[matplotlib.axes._axes.Axes] = None, **kwargs*) → matplotlib.axes._axes.Axes

Cluster magnitude is the sum of distances from all examples to the centroid of the cluster. This method plots the Total Point-to-Centroid Distance per cluster as a bar chart.

Parameters

- **X** – Training instances.
- **labels** – Labels of each point.
- **cluster_centers** – Coordinates of cluster centers.
- **distance_function** – The function used to calculate the distance between an instance to its cluster center. The function receives two ndarrays, one the instance and the second is the center and return a float number representing the distance between them.
- **ax** – Axes object to draw the plot onto, otherwise uses the current Axes.
- **kwargs** – other keyword arguments

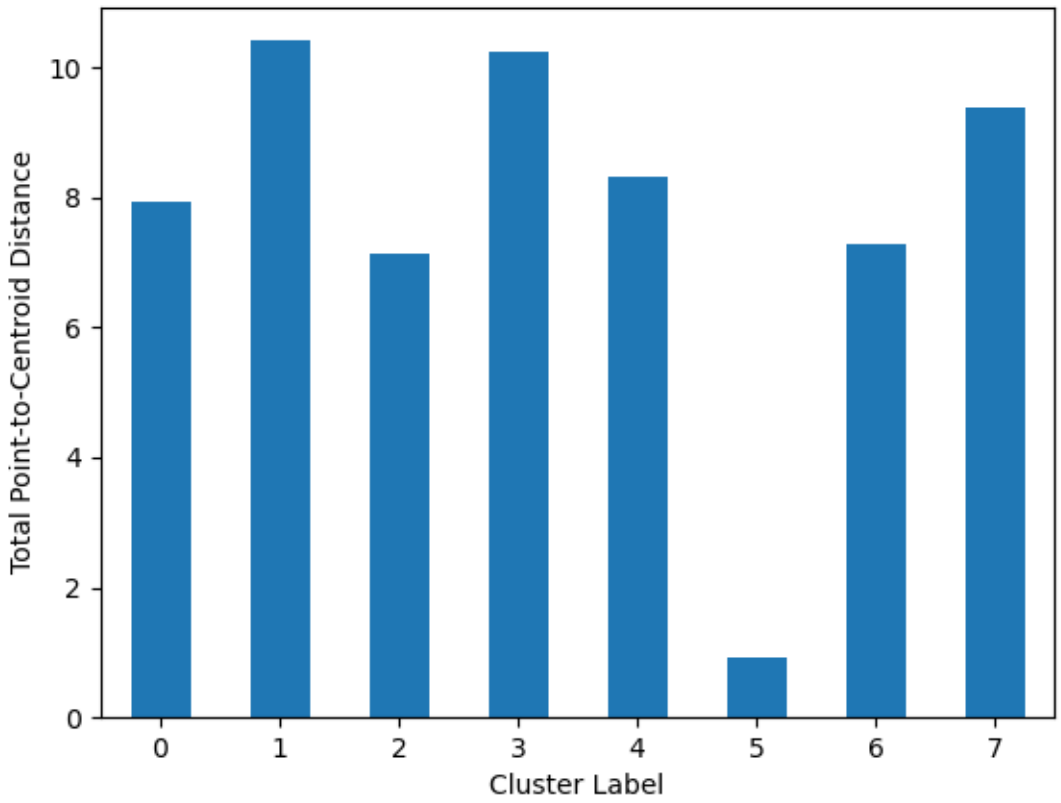
  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

Again we'll create a simple K-Means algorithm with k=8. This time we'll plot the sum of distances from points to their centroid:

```
from matplotlib import pyplot
from sklearn.cluster import KMeans
from scipy.spatial.distance import euclidean

from ds_utils.unsupervised import plot_cluster_magnitude



estimator = KMeans(n_clusters=8, random_state=42)
estimator.fit(x)

plot_cluster_magnitude(x, estimator.labels_, estimator.cluster_centers_, euclidean)

pyplot.show()
```

And the following image will be shown:



3.5.3 Magnitude vs. Cardinality

unsupervised.**plot_magnitude_vs_cardinality**(*X: numpy.ndarray, labels: numpy.ndarray, cluster_centers: numpy.ndarray, distance_function: Callable[[numpy.ndarray, numpy.ndarray], float], *, ax: Optional[matplotlib.axes._axes.Axes] = None, **kwargs*) → matplotlib.axes._axes.Axes

Higher cluster cardinality tends to result in a higher cluster magnitude, which intuitively makes sense. Clusters are anomalous when cardinality doesn't correlate with magnitude relative to the other clusters. Find anomalous clusters by plotting magnitude against cardinality as a scatter plot.

Parameters

- **X** – Training instances.
- **labels** – Labels of each point.
- **cluster_centers** – Coordinates of cluster centers.
- **distance_function** – The function used to calculate the distance between an instance to its cluster center. The function receives two ndarrays, one the instance and the second is the center and return a float number representing the distance between them.
- **ax** – Axes object to draw the plot onto, otherwise uses the current Axes.
- **kwargs** – other keyword arguments

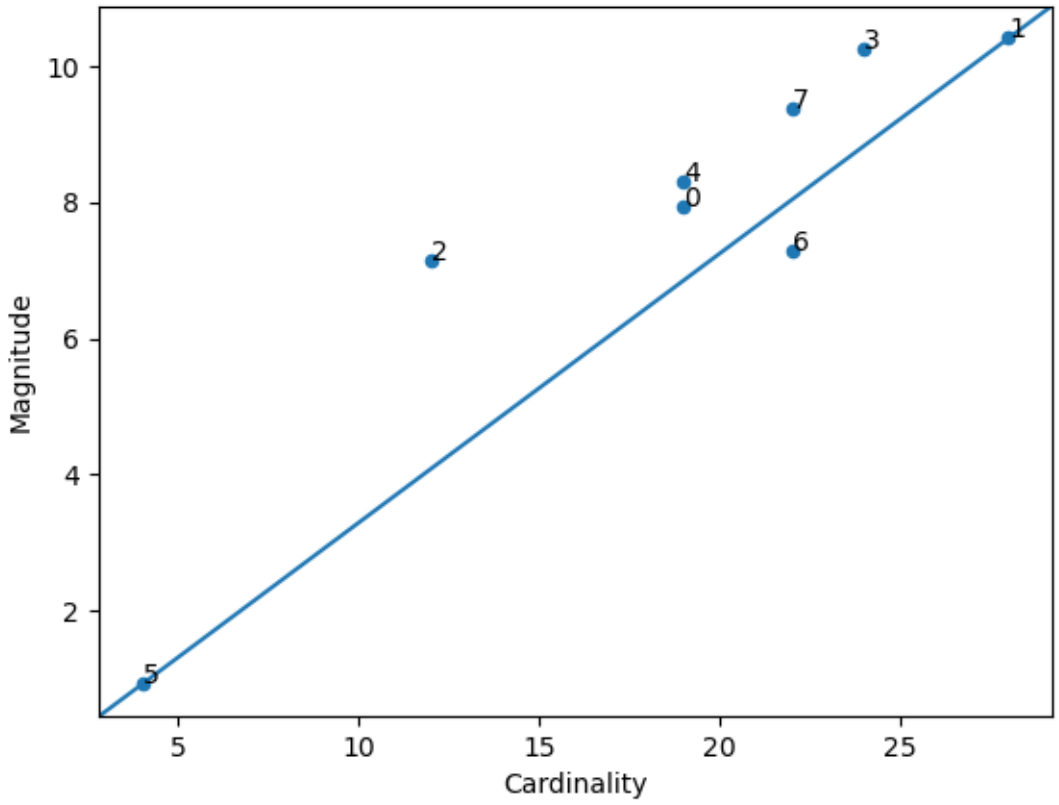
All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

Now let's plot the Cardinality vs. the Magnitude:

```
from matplotlib import pyplot
from sklearn.cluster import KMeans
from scipy.spatial.distance import euclidean

from ds_utils.unsupervised import plot_magnitude_vs_cardinality



estimator = KMeans(n_clusters=8, random_state=42)
estimator.fit(x)

plot_magnitude_vs_cardinality(x, estimator.labels_, estimator.cluster_centers_,␣
↪euclidean)

pyplot.show()
```

And the following image will be shown:



3.5.4 Optimum Number of Clusters

unsupervised.**plot_loss_vs_cluster_number**(*X: numpy.ndarray, k_min: int, k_max: int, distance_function: Callable[[numpy.ndarray, numpy.ndarray], float], *, algorithm_parameters: Dict[str, Any] = None, ax: Optional[matplotlib.axes._axes.Axes] = None, **kwargs*) → matplotlib.axes._axes.Axes

k-means requires you to decide the number of clusters `k` beforehand. This method runs the KMean algorithm and increases the cluster number at each try. The Total magnitude or sum of distance is used as loss.

Right now the method only works with `sklearn.cluster.KMeans`.

Parameters

- **X** – Training instances.
- **k_min** – The minimum cluster number.
- **k_max** – The maximum cluster number.
- **distance_function** – The function used to calculate the distance between an instance to its cluster center. The function receives two ndarrays, one the instance and the second is the center and return a float number representing the distance between them.
- **algorithm_parameters** – parameters to use for the algorithm. If None, deafult parameters of `KMeans` will be used.
- **ax** – Axes object to draw the plot onto, otherwise uses the current Axes.
- **kwargs** – other keyword arguments

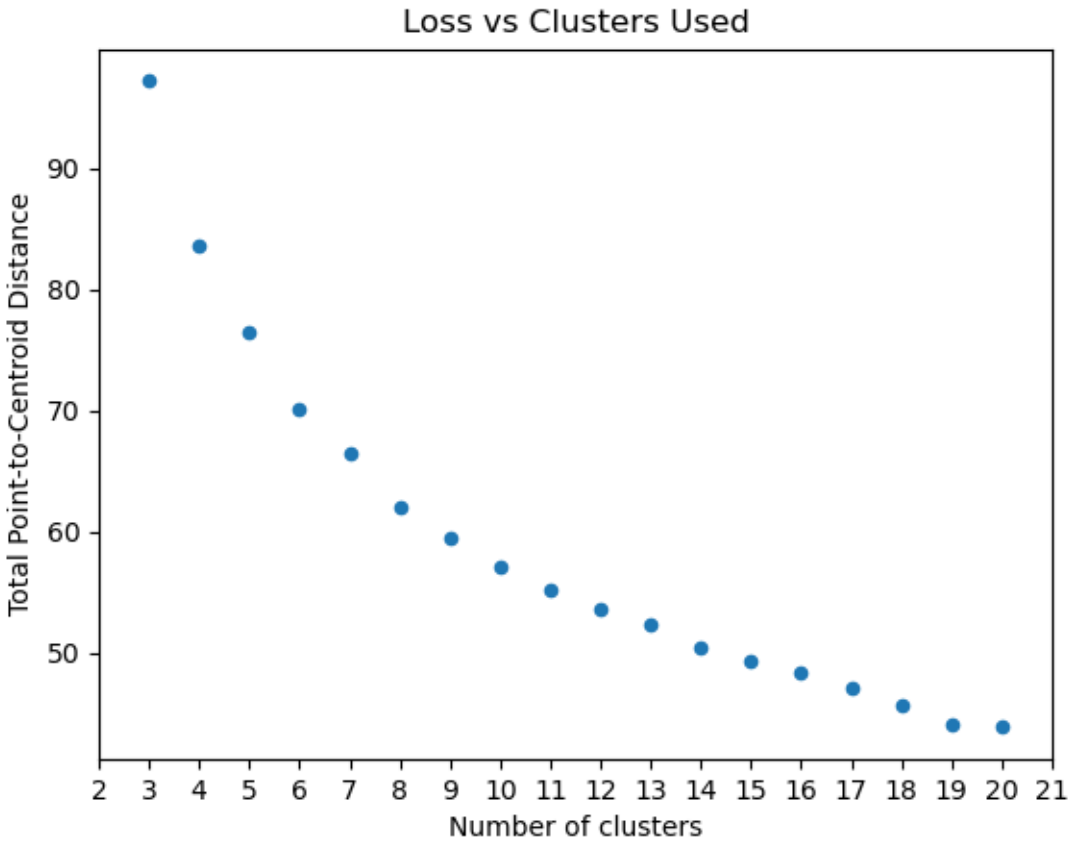
  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

Final plot we ca use is Loss vs Cluster Number:

```
from matplotlib import pyplot
from scipy.spatial.distance import euclidean

from ds_utils.unsupervised import plot_loss_vs_cluster_number



plot_loss_vs_cluster_number(x, 3, 20, euclidean)

pyplot.show()
```

And the following image will be shown:



3.6 XAI

The module of xai contains methods that help explain a model decisions.

In order for this module to work properly, Graphiz must be installed. In linux based operating systems use:

```
sudo apt-get install graphviz
```

Or using conda:

```
conda install graphviz
```

For more information see here.

3.6.1 Draw Tree

Deprecated since version 1.6.4: Use sklearn.tree.plot_tree instead

xai.**draw_tree**(*tree: sklearn.tree._classes.BaseDecisionTree*, *feature_names: Optional[List[str]] = None*, *class_names: Optional[List[str]] = None*, ***, *ax: Optional[matplotlib.axes._axes.Axes] = None*, ***kwargs*) → matplotlib.axes._axes.Axes

Receives a decision tree and return a plot graph of the tree for easy interpretation.

Parameters

- **tree** – decision tree.
- **feature_names** – the features names.
- **class_names** – the classes names or labels.
- **ax** – Axes object to draw the plot onto, otherwise uses the current Axes.
- **kwargs** – other keyword arguments

  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

Code Example

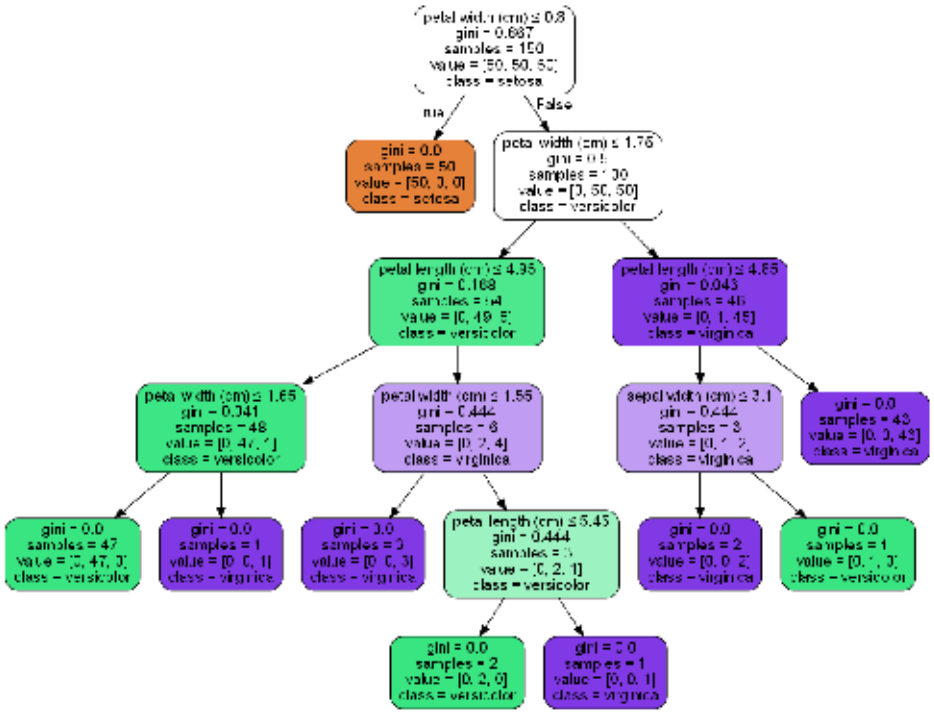
In following examples we are going to use the iris dataset from scikit-learn. so firstly let's import it:

```
from sklearn import datasets


iris = datasets.load_iris()
x = iris.data
y = iris.target
```

We'll create a simple decision tree classifier and plot it:

```
from matplotlib import pyplot
from sklearn.tree import DecisionTreeClassifier

from ds_utils.xai import draw_tree


# Create decision tree classifier object
clf = DecisionTreeClassifier(random_state=0)

# Train model
clf.fit(x, y)

draw_tree(clf, iris.feature_names, iris.target_names)
pyplot.show()
```

And the following image will be shown:



3.6.2 Draw Dot Data

`xai.draw_dot_data`(*dot_data: str*, *, *ax: Optional[matplotlib.axes._axes.Axes] = None*, ***kwargs*) → matplotlib.axes._axes.Axes

Receives a Graphiz's Dot language string and return a plot graph of the data.

Parameters

- **`dot_data`** – Graphiz's Dot language string.
- **`ax`** – Axes object to draw the plot onto, otherwise uses the current Axes.
- **`kwargs`** – other keyword arguments

  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

Code Example

We'll create a simple diagram and plot it:

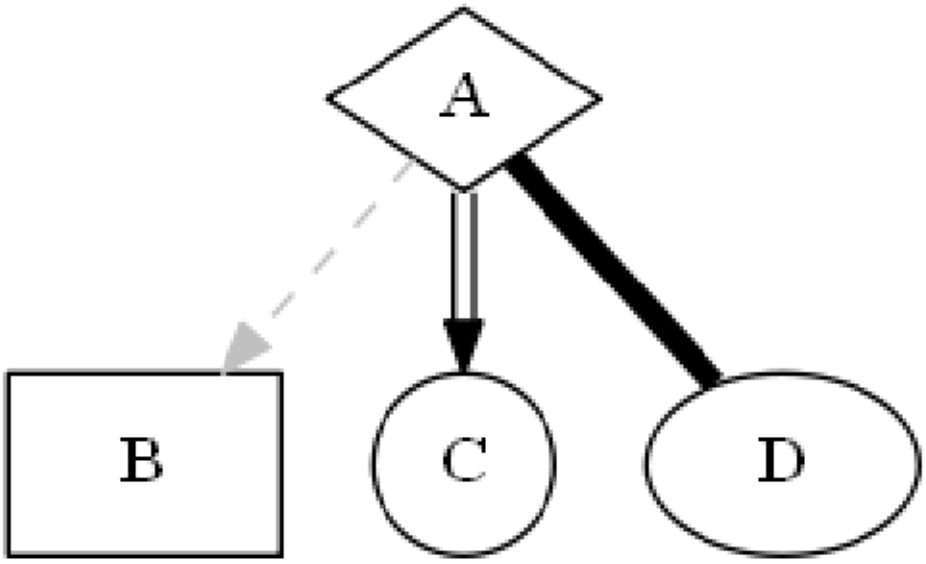
```
from matplotlib import pyplot
```

(continues on next page)

(continued from previous page)

```
from ds_utils.xai import draw_dot_data


dot_data = "digraph D{\n" \
           "\tA [shape=diamond]\n" \
           "\tB [shape=box]\n" \
           "\tC [shape=circle]\n" \
           "\n" \
           "\tA -> B [style=dashed, color=grey]\n" \
           "\tA -> C [color=\"black:invis:black\"]\n" \
           "\tA -> D [penwidth=5, arrowhead=none]\n" \
           "\n" \
           "}"

draw_dot_data(dot_data)
pyplot.show()
```

And the following image will be shown:



3.6.3 Generate Decision Paths

xai.**generate_decision_paths**(*classifier: sklearn.tree._classes.BaseDecisionTree*, *feature_names: Optional[List[str]] = None*, *class_names: Optional[List[str]] = None*, *tree_name: Optional[str] = None*, *indent_char: str = '\t'*) → str

Receives a decision tree and return the underlying decision-rules (or 'decision paths') as text (valid python syntax). Original code

Parameters

- **classifier** – decision tree.
- **feature_names** – the features names.
- **class_names** – the classes names or labels.
- **tree_name** – the name of the tree (function signature).
- **indent_char** – the character used for indentation.

Returns textual representation of the decision paths of the tree.

Code Example

We'll create a simple decision tree classifier and print it:

```
from sklearn.tree import DecisionTreeClassifier

from ds_utils.xai import generate_decision_paths


# Create decision tree classifier object
clf = DecisionTreeClassifier(random_state=0, max_depth=3)

# Train model
clf.fit(x, y)
print(generate_decision_paths(clf, iris.feature_names, iris.target_names.tolist(),
                      "iris_tree"))
```

The following text will be printed:

```
def iris_tree(petal width (cm), petal length (cm)):
    if petal width (cm) <= 0.8000:
        # return class setosa with probability 0.9804
        return ("setosa", 0.9804)
    else:  # if petal width (cm) > 0.8000
        if petal width (cm) <= 1.7500:
            if petal length (cm) <= 4.9500:
                # return class versicolor with probability 0.9792
                return ("versicolor", 0.9792)
            else:  # if petal length (cm) > 4.9500
                # return class virginica with probability 0.6667
                return ("virginica", 0.6667)
        else:  # if petal width (cm) > 1.7500
            if petal length (cm) <= 4.8500:
                # return class virginica with probability 0.6667
                return ("virginica", 0.6667)
            else:  # if petal length (cm) > 4.8500
```

(continues on next page)

(continued from previous page)

```
                # return class virginica with probability 0.9773
                return ("virginica", 0.9773)
```

3.6.4 Plot Features' Importance

xai.**plot_features_importance**(*feature_names: List[str], feature_importance: List[float], *, ax: Optional[matplotlib.axes._axes.Axes] = None, **kwargs*) → matplotlib.axes._axes.Axes

plot feature importance as a bar chart.

Parameters

- **feature_names** – strings list of feature names
- **feature_importance** – float list of feature importance
- **ax** – Axes object to draw the plot onto, otherwise uses the current Axes.
- **kwargs** – other keyword arguments

  All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns Returns the Axes object with the plot drawn onto it.

Code Example

For this example I created a dummy data set. You can find the data at the resources directory in the packages tests folder.

Let's see how to use the code:

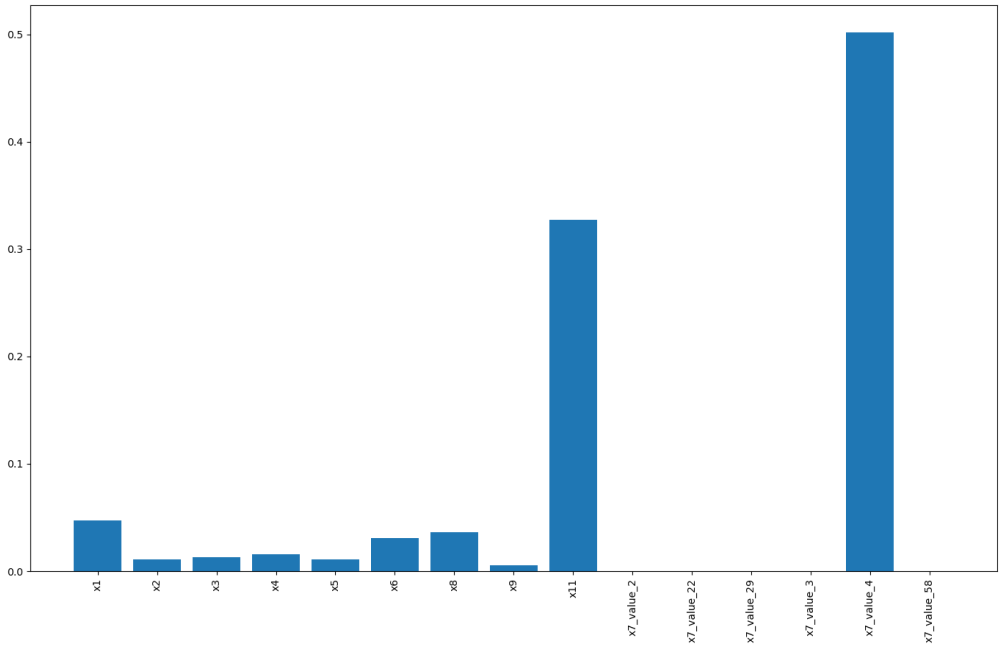
```
import pandas

from matplotlib import pyplot
from sklearn.preprocessing import OneHotEncoder
from sklearn.tree import DecisionTreeClassifier

from ds_utils.xai import plot_features_importance


data_1M = pandas.read_csv(path/to/dataset)
target = data_1M["x12"]
categorical_features = ["x7", "x10"]
for i in range(0, len(categorical_features)):
    enc = OneHotEncoder(sparse=False, handle_unknown="ignore")
    enc_out = enc.fit_transform(data_1M[[categorical_features[i]]])
    for j in range(0, len(enc.categories_[0])):
        data_1M[categorical_features[i] + "_" + enc.categories_[0][j]] = enc_out[:, j]
features = data_1M.columns.to_list()
features.remove("x12")
features.remove("x7")
features.remove("x10")

clf = DecisionTreeClassifier(random_state=42)
clf.fit(data_1M[features], target)
```

(continues on next page)

(continued from previous page)

```
plot_features_importance(features, clf.feature_importances_)

pyplot.show()
```

And the following image will be shown:



CHAPTER 4

Indices and tables

- genindex
- search

Index