Teachnok Intership

MAJOR PROJECT REPORT

FEBRUARY BATCH 2023

Take any Dataset of your choice ,perform EDA(Exploratory Data Analysis) and apply a suitable Classifier,Regressor or Clusterer and calculate the accuracy of the model.

BY MUSTKEEM AHMAD

Exploratory Data Analysis (EDA)

Code snippets and excerpts from the tutorial. Python 3. From DataCamp.

Exploratory Data Analysis (EDA) prior to Machine Learning

Supervised learning models with the help of exploratory data analysis (EDA) on the Titanic data.

How to Start with Supervised Learning (Take 1)

Approach supervised learning is the following:

- Perform an Exploratory Data Analysis (EDA) on a dataset;
- Build a quick and dirty model, or a baseline model, which can serve as a comparison against later models that we will build;
- Iterate this process. We will do more EDA and build another model;
- Engineer features: take the features that we already have and combine them or extract more information from them to eventually come to the last point, which is
- Get a model that performs better.

Import the Data and Explore it

```
# Import modules
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import tree
from sklearn.metrics import accuracy_score

# Figures inline and set visualization style
%matplotlib inline
sns.set()
```

```
# Import test and train datasets
df_train = pd.read_csv('data/train.csv')
df_test = pd.read_csv('data/test.csv')

# View first lines of training data
df_train.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thhttps://ugoproto.github.io/ugo_py_doc.	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

- The target variable is the variable we are trying to predict;
- Other variables are known as “features” (or “predictor variables”, the features that we are using to predict the target variable).

Note that the `df_test` DataFrame doesn’t have the `Survived` column because this is what we will try to predict!

```
# View first lines of test data
df_test.head(3)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

```
df_train.describe()
```

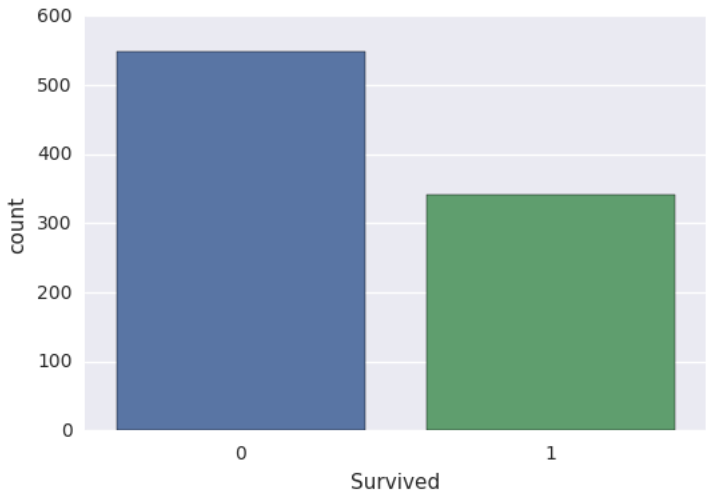
	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Visual Exploratory Data Analysis (EDA) and a First Model

With seaborn.

```
sns.countplot(x='Survived', data=df_train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc65fa0e668>
```



Take-away: in the training set, less people survived than didn't. Let's then build a first model that predicts that nobody survived.

This is a bad model as we know that people survived. But it gives us a **baseline**: any model that we build later needs to do better than this one.

- Create a column `Survived` for `df_test` that encodes 'did not survive' for all rows;
- Save `PassengerId` and `Survived` columns of `df_test` to a .csv and submit to Kaggle.

```
df_test['Survived'] = 0
df_test[['PassengerId', 'Survived']].to_csv('results/no_survivors.csv', index=False)
```

Submit to Kaggle (1st)

- Go to Kaggle, log in, and search for *Titanic: Machine Learning from Disaster*.
- Join the competition and submit the .csv file.
- Add a description and submit.
- Kaggle returns a ranking.
- At the time of the first submission: score 0.63679, rank 9387.

9381	new	**Aditya Ramesh**	0.62678	1	2d
9382	new	**Alok Jha**	0.62678	1	2d
9383	new	**Eman Khan**	0.62678	1	2d
9384	new	**Adrian Stein**	0.62678	1	2d
9385	new	**Kcvizer**	0.62678	1	2d
9386	new	**Chihiro**	0.62678	1	2d
9387	new	**Ugo Sparks**	0.62678	1	now

Your Best Entry ↑

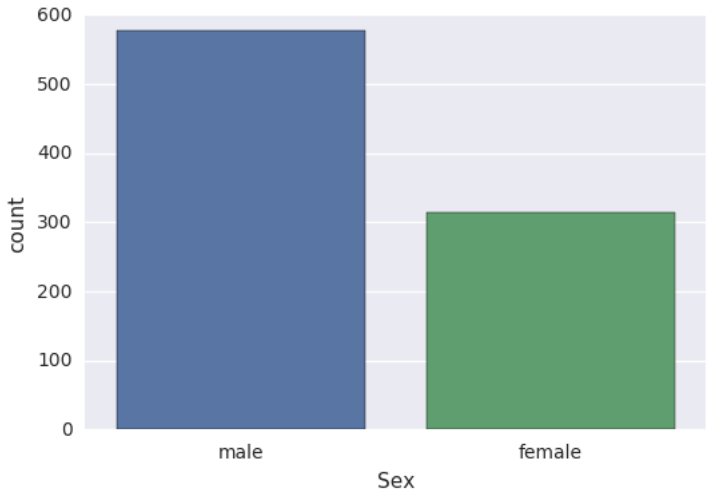
Your submission scored 0.62679, which is not an improvement of your best score. Keep trying!

9388	▼982	**Lokesh Batra**	0.62200	1	2mo
9389	▼982	**yurunyang**	0.62200	2	2mo
9390	▼982	**riya kondo**	0.62200	1	2mo
9391	▼982	**Karan Deshmukh**	0.62200	4	2mo
9392	▼982	**mcesar2**	0.62200	1	2mo
9393	▼982	**BE_KURO**	0.62200	1	2mo

EDA on Feature Variables

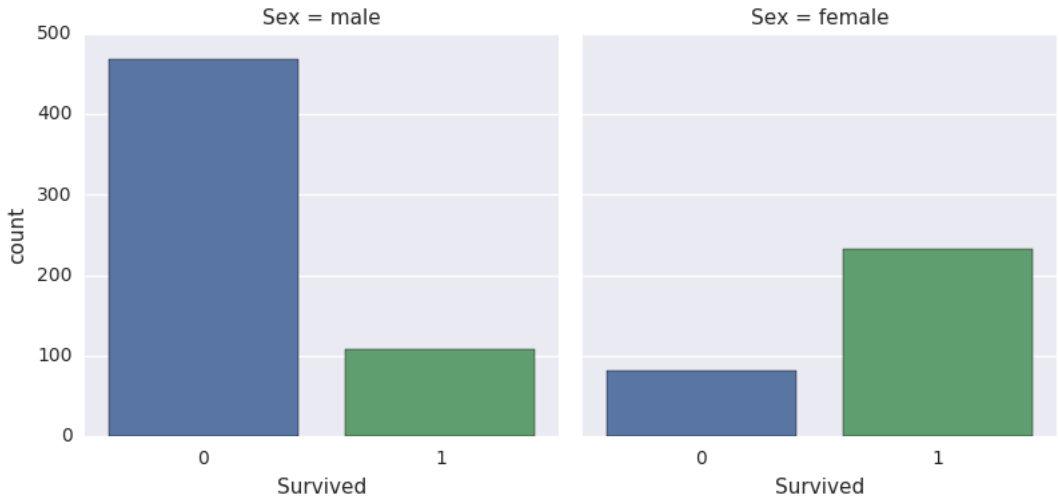
Do some more Exploratory Data Analysis and build another model!

```
sns.countplot(x='Sex', data=df_train);
```



```
# kind is the facets
sns.factorplot(x='Survived', col='Sex', kind='count', data=df_train)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fc65fa35a20>
```



Take-away: Women were more likely to survive than men.

With this take-away, we can use pandas to figure out how many women and how many men survived:

```
df_train.head(1)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25	NaN	S

```
# Chain a group by Sex, sum Survived
df_train.groupby(['Sex']).Survived.sum()
```

```
Sex
female    233
male      109
Name: Survived, dtype: int64
```

```
# Chain calculations
print(df_train[df_train.Sex == 'female'].Survived.sum() /
df_train[df_train.Sex == 'female'].Survived.count())

print(df_train[df_train.Sex == 'male'].Survived.sum() /
df_train[df_train.Sex == 'male'].Survived.count())
```

```
0.742038216561
0.188908145581
```

74% of women survived, while 19% of men survived.

Build a second model and predict that all women survived and all men didn't.

- Create a column `Survived` for `df_test` that encodes the above prediction.
- Save `PassengerId` and `Survived` columns of `df_test` to a .csv and submit to Kaggle.

```
df_test['Survived'] = df_test.Sex == 'female'
df_test['Survived'] = df_test.Survived.apply(lambda x: int(x))
df_test.head(3)
```

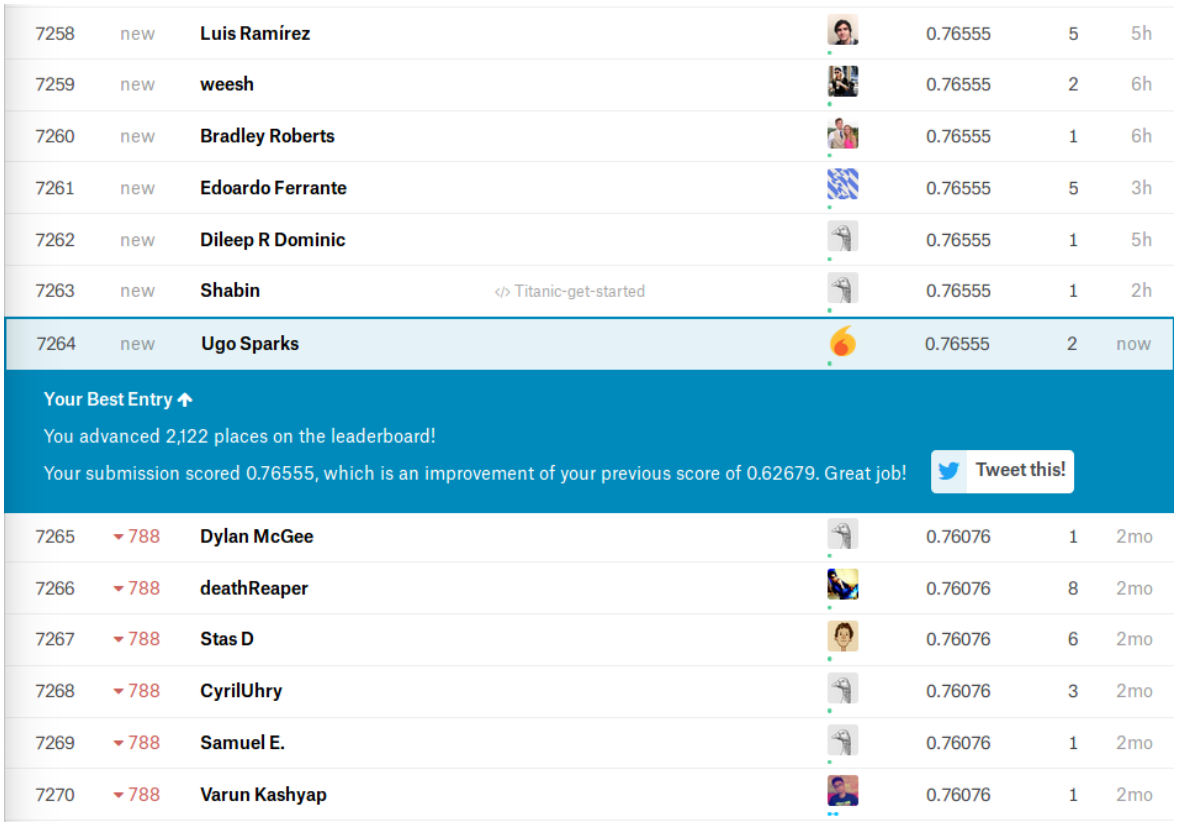
	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q	0

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S	1
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q	0

```
df_test[['PassengerId', 'Survived']].to_csv('results/women_survived.csv', index=False)
```

Submit to Kaggle (2nd)

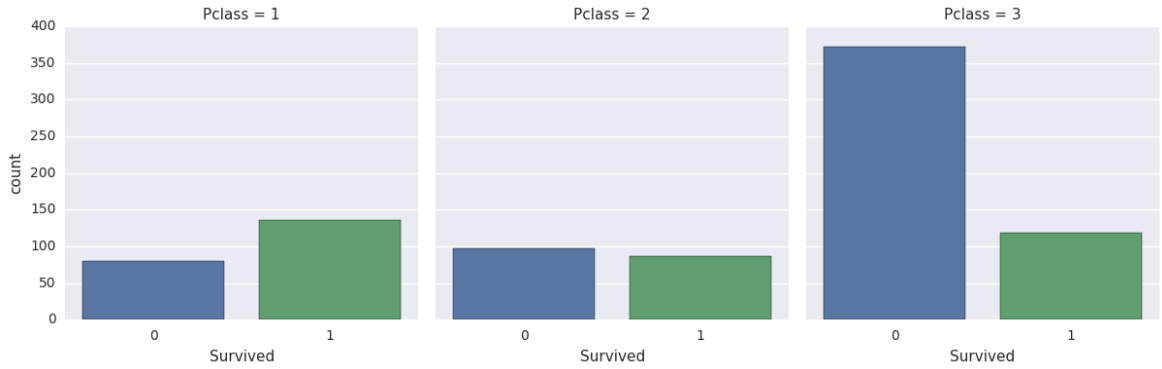
- Go to Kaggle, log in, and search for *Titanic: Machine Learning from Disaster*.
- Join the competition and submit the .csv file.
- Add a description and submit.
- Kaggle returns a ranking.
- At the time of the first submission: score 0.76555 (from 0.62679), rank 7274 (a jump of 2122 places).



Explore the Data More!

```
# kind is the facets
sns.factorplot(x='Survived', col='Pclass', kind='count', data=df_train)
```

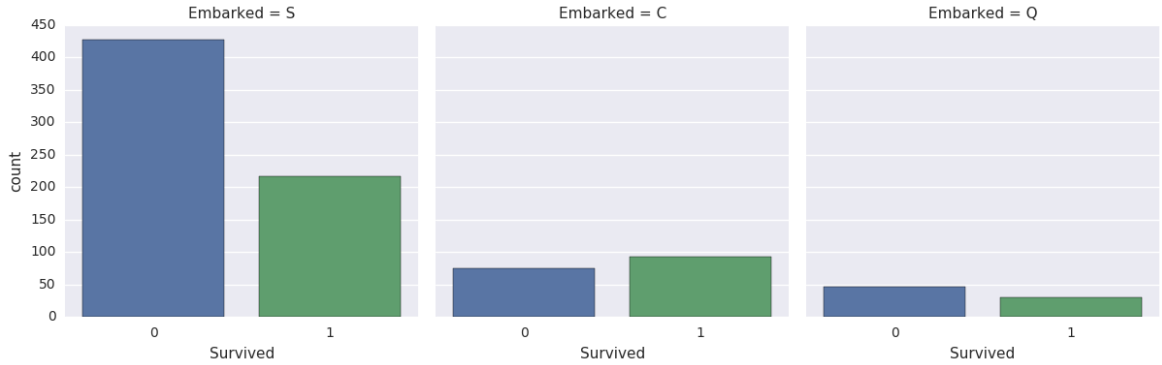
```
1 <seaborn.axisgrid.FacetGrid at 0x7fc65f8dcf98>
```



Take-away: Passengers that travelled in first class were more likely to survive. On the other hand, passengers travelling in third class were more unlikely to survive.

```
# kind is the facets
sns.factorplot(x='Survived', col='Embarked', kind='count', data=df_train)
```

```
1 <seaborn.axisgrid.FacetGrid at 0x7fc65f937c50>
```

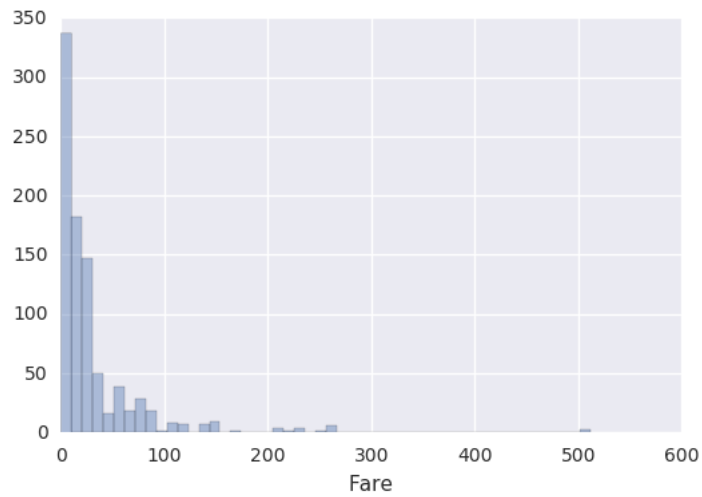


Take-away: Passengers that embarked in Southampton were less likely to survive.

EDA with Numeric Variables

```
sns.distplot(df_train.Fare, kde=False)
```

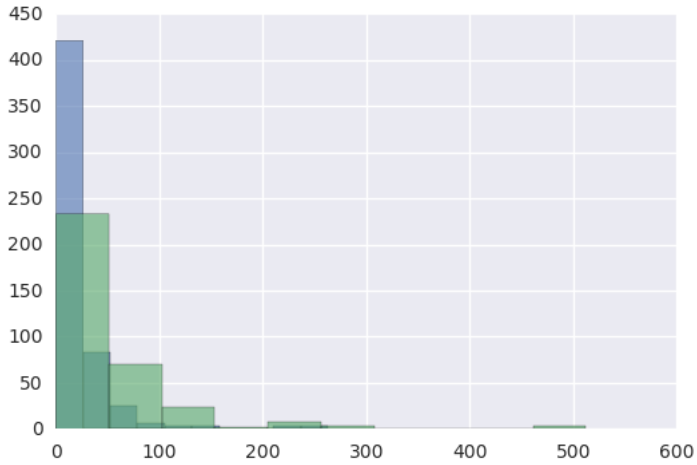
```
1 <matplotlib.axes._subplots.AxesSubplot at 0x7fc6633fb9e8>
```



Take-away: Most passengers paid less than 100 for travelling with the Titanic.

```
# Group by Survived, trace histograms of Fare with alpha color 0.6
df_train.groupby('Survived').Fare.hist(alpha=0.6)
```

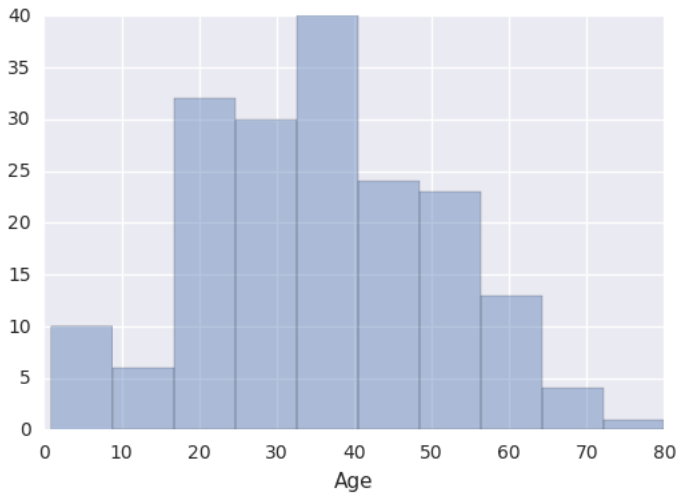
```
Survived
0    Axes(0.125,0.125;0.775x0.775)
1    Axes(0.125,0.125;0.775x0.775)
Name: Fare, dtype: object
```



Take-away: It looks as though those that paid more had a higher chance of surviving.

```
# Remove NaN
df_train_drop = df_train.dropna()

sns.distplot(df_train_drop.Age, kde=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc65f64be80>
```

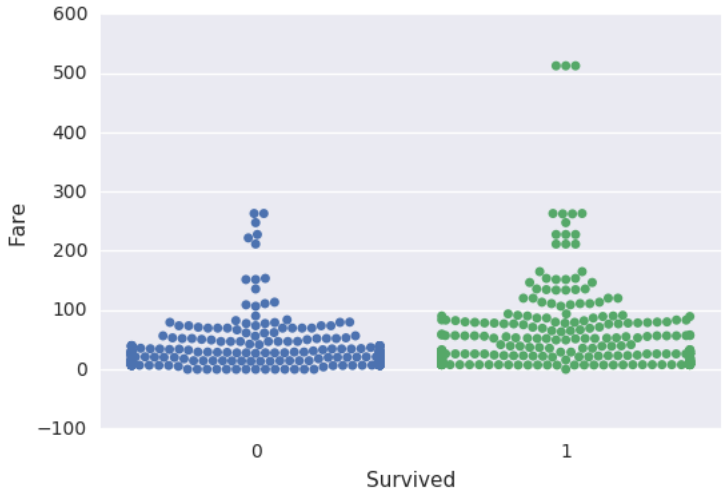



```
# Alternative to bars or scatter
sns.stripplot(x='Survived',
    y='Fare',
    data=df_train,
    alpha=0.3, jitter=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc65f5cca58>
```


```
# Alternative to bars or scatter
sns.swarmplot(x='Survived',
    y='Fare',
    data=df_train)
```

```
1 <matplotlib.axes._subplots.AxesSubplot at 0x7fc65f2e4ef0>
```



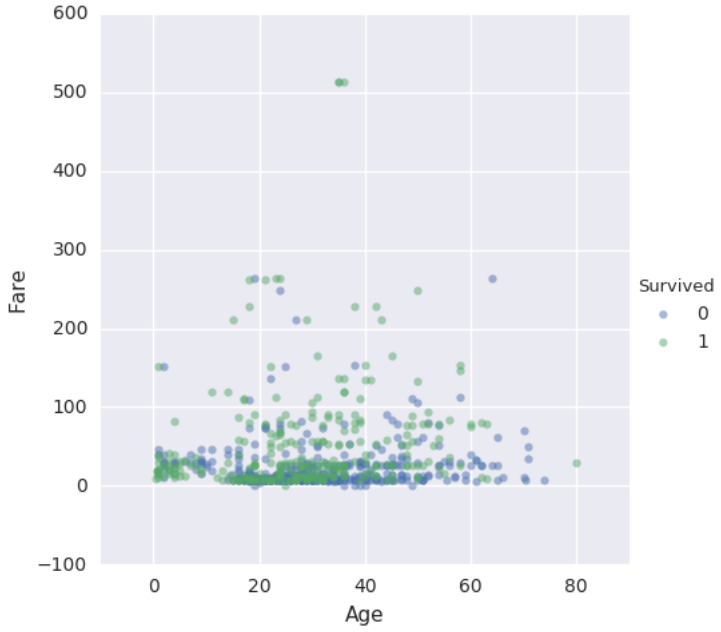
Take-away: Fare definitely seems to be correlated with survival aboard the Titanic.

```
# Group by Survived, describe Fare (descriptive statistics)
df_train.groupby('Survived').Fare.describe()
```

	count	mean	std	min	25%	50%	75%	max
Survived								
0	549.0	22.117887	31.388207	0.0	7.8542	10.5	26.0	263.0000
1	342.0	48.395408	66.596998	0.0	12.4750	26.0	57.0	512.3292

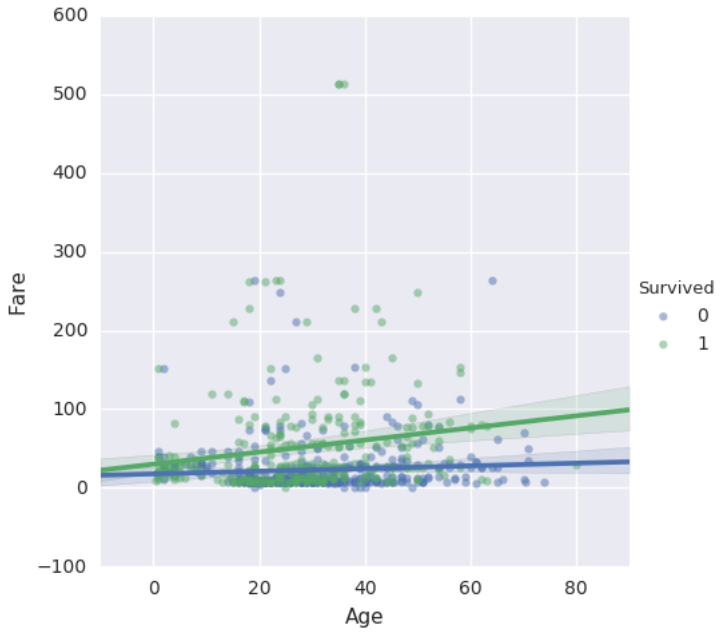
```
sns.lmplot(x='Age',
 y='Fare',
 hue='Survived',
 data=df_train,
 fit_reg=False, scatter_kws={'alpha':0.5})
```

```
1 <seaborn.axisgrid.FacetGrid at 0x7fc65f5ccba8>
```




```
sns.lmplot(x='Age',
 y='Fare',
 hue='Survived',
 data=df_train,
 fit_reg=True, scatter_kws={'alpha':0.5})
```

```
1 <seaborn.axisgrid.FacetGrid at 0x7fc65f22d710>
```



Take-away: It looks like those who survived either paid quite a bit for their ticket or they were young.

```
sns.pairplot(df_train_drop, hue='Survived')
```

```
1 <seaborn.axisgrid.PairGrid at 0x7fc65f8826d8>
```

A First Machine Learning Model

A decision tree classifier, with the Python scikit-learn.

How to Start with Supervised Learning (Take 2)

Now that we have done our homeworks with EDA…

```
# Import modules
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import numpy as np
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

# Figures inline and set visualization style
%matplotlib inline
sns.set()
```

```
# Import data
df_train = pd.read_csv('data/train.csv')
df_test = pd.read_csv('data/test.csv')
```

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
PassengerId    418 non-null int64
Pclass         418 non-null int64
Name           418 non-null object
Sex            418 non-null object
Age            332 non-null float64
SibSp          418 non-null int64
Parch          418 non-null int64
Ticket         418 non-null object
Fare           417 non-null float64
Cabin          91 non-null object
Embarked       418 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

```
# Store target variable of training data in a safe place
survived_train = df_train.Survived

# Concatenate (along the index or axis=1) training and test sets
# to preprocess the data a little bit
# and make sure that any operations that
# we perform on the training set are also
# being done on the test data set
data = pd.concat([df_train.drop(['Survived'], axis=1), df_test])
```

```
# The combined datasets (891+418 entries)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 0 to 417
Data columns (total 11 columns):
PassengerId    1309 non-null int64
Pclass         1309 non-null int64
Name           1309 non-null object
Sex            1309 non-null object
Age            1046 non-null float64
SibSp          1309 non-null int64
Parch          1309 non-null int64
Ticket         1309 non-null object
Fare           1308 non-null float64
Cabin          295 non-null object
Embarked       1307 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 122.7+ KB
```

Missing values for the `Age` and `Fare` columns! Also notice that `Cabin` and `Embarked` are also missing values and we will need to deal with that also at some point. However, now we will focus on fixing the numerical variables `Age` and `Fare`, using the median of the of these variables where we know them. It's perfect for dealing with outliers. In other words, the median is useful to use when the distribution of data is skewed. Other ways to impute the missing values would be to use the mean or the mode.

```
# Impute missing numerical variables where NaN
data['Age'] = data.Age.fillna(data.Age.median())
data['Fare'] = data.Fare.fillna(data.Fare.median())

# Check out info of data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 0 to 417
Data columns (total 11 columns):
PassengerId    1309 non-null int64
Pclass         1309 non-null int64
Name           1309 non-null object
Sex            1309 non-null object
Age            1309 non-null float64
SibSp          1309 non-null int64
Parch          1309 non-null int64
Ticket         1309 non-null object
Fare           1309 non-null float64
Cabin          295 non-null object
Embarked       1307 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 122.7+ KB
```

Encode the data with numbers with `.get_dummies()`.

It creates a new column for female, called `Sex_female`, and then a new column for `Sex_male`, which encodes whether that row was male or female (1 if that row is a male - and a 0 if that row is female). Because of `drop_first` argument, we dropped `Sex_female` because, essentially, these new columns, `Sex_female` and `Sex_male`, encode the same information.

```
data = pd.get_dummies(data, columns=['Sex'], drop_first=True)
data.head(3)
```

	PassengerId	Pclass	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Sex_male
0	1	3	Braund, Mr. Owen Harris	22.0	1	0	A/5 21171	7.2500	NaN	S	1
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Thhttps://ugoproto.github.io/ugo_py_doc.	38.0	1	0	PC 17599	71.2833	C85	C	0
2	3	3	Heikkinen, Miss. Laina	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	0

```
# Select columns and view head
data = data[['Sex_male', 'Fare', 'Age','Pclass', 'SibSp']]
data.head(3)
```

	Sex_male	Fare	Age	Pclass	SibSp
0	1	7.2500	22.0	3	1
1	0	71.2833	38.0	1	1
2	0	7.9250	26.0	3	0

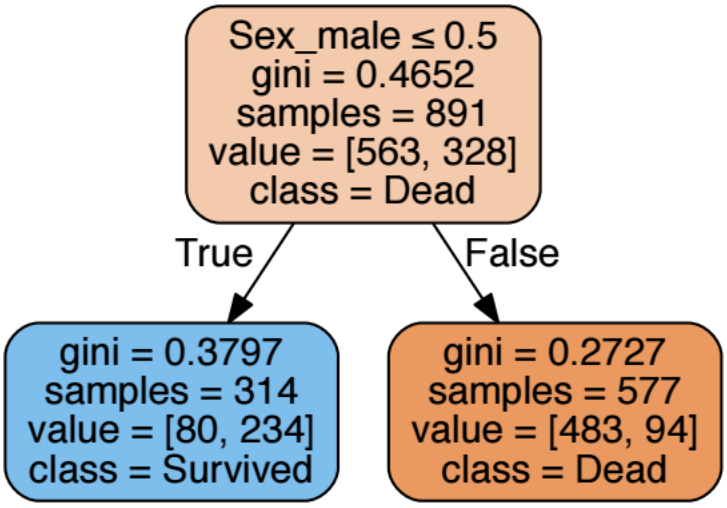
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 0 to 417
Data columns (total 5 columns):
Sex_male    1309 non-null uint8
Fare        1309 non-null float64
Age         1309 non-null float64
Pclass      1309 non-null int64
SibSp       1309 non-null int64
dtypes: float64(2), int64(2), uint8(1)
memory usage: 52.4 KB
```

All the entries are non-null now.

Build a Decision Tree Classifier

"Was `Sex_male`" less than 0.5? In other words, was the data point a female. If the answer to this question is `True`, we can go down to the left and we get `Survived`. If `False`, we go down the right and we get `Dead`.



That the first branch is on `Male` or not and that `Male` results in a prediction of `Dead`. The gini coefficient is used to make these decisions.

Before fitting a model to the data, split it back into training and test sets:

```
data_train = data.iloc[:891]
data_test = data.iloc[891:]
```

scikit-learn requires the data as arrays, not DataFrames. Transform them.

```
X = data_train.values
test = data_test.values

# and from above: survived_train = df_train.Survived
y = survived_train.values
```

```
X
```

```
array([[  1.    ,   7.25  ,  22.    ,   3.    ,   1.    ],
[  0.    ,  71.2833,  38.    ,   1.    ,   1.    ],
[  0.    ,   7.925 ,  26.    ,   3.    ,   0.    ],
https://ugoproto.github.io/ugo_py_doc.,
[  0.    ,  23.45  ,  28.    ,   3.    ,   1.    ],
[  1.    ,  30.    ,  26.    ,   1.    ,   0.    ],
[  1.    ,   7.75  ,  32.    ,   3.    ,   0.    ]])
```

Build a decision tree classifier! First create such a model with `max_depth=3` and then fit it the data. Name the model `clf`, which is short for "Classifier".

```
# Instantiate model and fit to data
# The max depth is set at 3
clf = tree.DecisionTreeClassifier(max_depth=3)

# X is the indenpendent variables, y is the dependent variable
clf.fit(X, y)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
  max_features=None, max_leaf_nodes=None,
  min_impurity_decrease=0.0, min_impurity_split=None,
  min_samples_leaf=1, min_samples_split=2,
  min_weight_fraction_leaf=0.0, presort=False, random_state=None,
  splitter='best')
```

Make predictions on the test set.

```
# Make predictions and store in 'Survived' column of df_test
Y_pred = clf.predict(test)
df_test['Survived'] = Y_pred

# Save it
df_test[['PassengerId', 'Survived']].to_csv('results/1st_dec_tree.csv',
                                index=False)
```

Submit to Kaggle (3rd)

- Go to Kaggle, log in, and search for *Titanic: Machine Learning from Disaster*.
- Join the competition and submit the .csv file.
- Add a description and submit.
- Kaggle returns a ranking.
- At the time of the first submission: score 0.77990 (from 0.76555), rank 4828 (a jump of 2434 places).

4822	new	**Jatin Jain**		0.77990	3	7h
4823	new	**Nikita Konstantinovskiy**		0.77990	5	7h
4824	▲ 2037	**nishtha**		0.77990	2	7h
4825	new	**Shyam Patibandla**		0.77990	5	6h
4826	new	**minato**		0.77990	5	5h
4827	new	**BizDahaOlmedik**		0.77990	2	2h
4828	new	**Ugo Sparks**		0.77990	3	now

Your Best Entry ↑

You advanced 2,434 places on the leaderboard!

Your submission scored 0.77990, which is an improvement of your previous score of 0.76555. Great job! Tweet this!

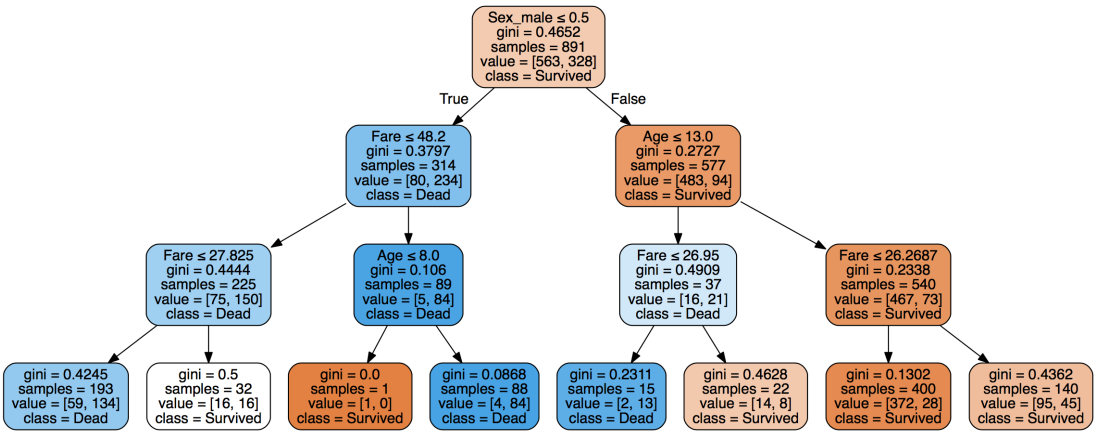
		My First Random Forest		0.77511		
4829	▼ 543	**MasaKusar**		0.77511	2	2mo
4830	▼ 543	**easy0000**		0.77511	1	2mo
4831	▼ 543	**ChaoYang**		0.77511	1	2mo
4832	▼ 543	**jrz371**		0.77511	10	1mo
4833	▼ 543	**NamanDoshi**		0.77511	1	2mo
4834	▼ 543	**henryfang**		0.77511	1	2mo

```
# Compute accuracy on the training set
train_accuracy = clf.score(X, y)
train_accuracy
```

```
0.8271604938271605
```

A Decision Tree Classifier in More Details

The Decision Tree Classifier we just built had a max_depth=3 and it looks like this:



The maximal distance between the first decision and the last is 3, so that's `max_depth=3`.

Generate images with graphviz.

```
import graphviz

tree_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(tree_data)
# Save the pdf
graph.render("img/tree_data")
```

```
1  'img/tree_data.pdf'
```

We get a tree_data test file (the code for generating the image) and a pdf file. We can generate an image.

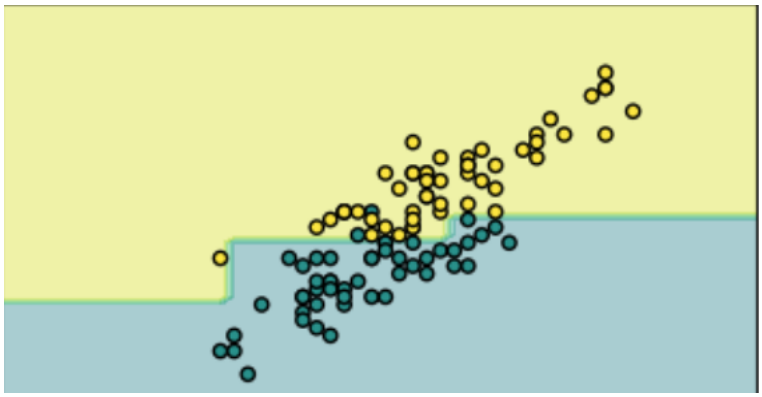
```
feature_names = list(data_train)
feature_names
```

```
1  ['Sex_male', 'Fare', 'Age', 'Pclass', 'SibSp']
```

```
#data_train
#data_test
tree_data = tree.export_graphviz(clf, out_file=None,
                     feature_names=feature_names,
                     class_names=None,
                     filled=True, rounded=True,
                     special_characters=True)
graph = graphviz.Source(tree_data)
graph
```

IN THE NOTEBOOK ONLY!

In building this model, what we are essentially doing is creating a decision boundary in the space of feature variables.



Why Choose `max_depth=3`?

The depth of the tree is known as a hyperparameter, which means a parameter we need to decide before we fit the model to the data. If we choose a larger `max_depth`, we will get a more complex decision boundary; the bias-variance trade-off.

- If the decision boundary is too complex, we can overfit to the data, which means that the model will be describing noise as well as signal.
- If the `max_depth` is too small, we might be underfitting the data, meaning that the model doesn't contain enough of the signal.

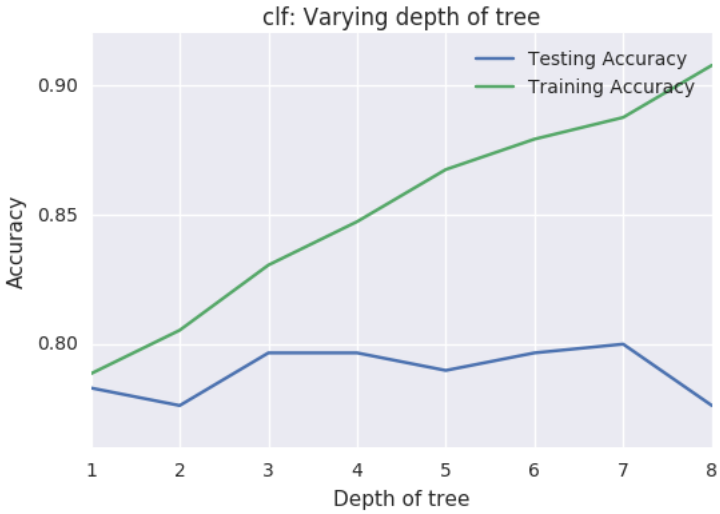
One way is to hold out a test set from the training data. We can then fit the model to the training data, make predictions on the test set and see how well the prediction does on the test set.

Split the original training data into training and test sets:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42, stratify=y)
```

Iterate over values of `max_depth` ranging from 1 to 9 and plot the accuracy of the models on training and test sets:

```
# Setup arrays to store train and test accuracies
dep = np.arange(1, 9)
train_accuracy = np.empty(len(dep))
test_accuracy = np.empty(len(dep))

# Loop over different values of k
for i, k in enumerate(dep):
# Setup a k-NN Classifier with k neighbors: knn
clf = tree.DecisionTreeClassifier(max_depth=k)

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Compute accuracy on the training set
train_accuracy[i] = clf.score(X_train, y_train)

# Compute accuracy on the testing set
test_accuracy[i] = clf.score(X_test, y_test)

# Generate plot
plt.title('clf: Varying depth of tree')
plt.plot(dep, test_accuracy, label = 'Testing Accuracy')
plt.plot(dep, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Depth of tree')
plt.ylabel('Accuracy')
plt.show()
```



At `max_depth-3`, we get the same results as with the model before (around 82%).

As we increase the max_depth, we are going to fit better and better to the training data because we will make decisions that describe the training data. The accuracy for the training data will go up and up, but we see that this doesn't happen for the test data: we are overfitting.

So that's why we chose `max_depth=3`.

Feature Engineering

https://www.datacamp.com/community/tutorials/feature-engineering-kaggle

A process where we use domain knowledge of the data to create additional relevant features (create new columns, transform variables and more) that increase the predictive power of the learning algorithm and make the machine learning models perform even better.

How to Start with Feature Engineering

```
# Imports
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import numpy as np
from sklearn import tree
from sklearn.model_selection import GridSearchCV

# Figures inline and set visualization style
%matplotlib inline
sns.set()
```

```
# Import data
df_train = pd.read_csv('data/train.csv')
df_test = pd.read_csv('data/test.csv')

# Store target variable of training data in a safe place
survived_train = df_train.Survived

# Concatenate training and test sets
data = pd.concat([df_train.drop(['Survived'], axis=1), df_test])

# View head
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 0 to 417
Data columns (total 11 columns):
PassengerId    1309 non-null int64
Pclass         1309 non-null int64
Name           1309 non-null object
Sex            1309 non-null object
Age            1046 non-null float64
SibSp          1309 non-null int64
Parch          1309 non-null int64
```

```
Ticket         1309 non-null object
Fare           1308 non-null float64
Cabin          295 non-null object
Embarked       1307 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 122.7+ KB
```

Why Feature Engineer At All?

Titanic's Passenger Titles

```
# View head of 'Name' column
data.Name.tail()
```

```
413              Spector, Mr. Woolf
414    Oliva y Ocana, Dona. Fermina
415    Saether, Mr. Simon Sivertsen
416             Ware, Mr. Frederick
417        Peter, Master. Michael J
Name: Name, dtype: object
```

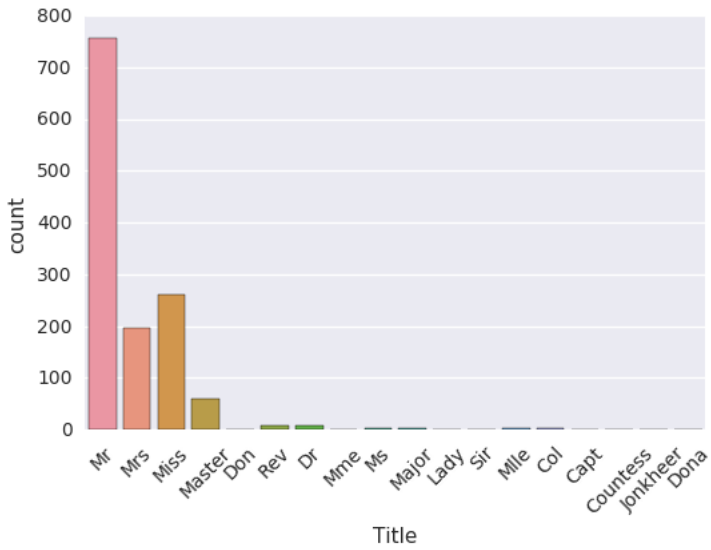
These titles of course give us information on social status, profession, etc., which in the end could tell us something more about survival. use regular expressions to extract the title and store it in a new column 'Title':

```
# Extract Title from Name, store in column and plot barplot
# One upper character, one lower character, one dot
data['Title'] = data.Name.apply(lambda x: re.search(' ([A-Z][a-z]+)\.', x).group(1))
```

```
# New column Title is a new feature of the dataset
data.Title.head(3)
```

```
0      Mr
1     Mrs
2    Miss
Name: Title, dtype: object
```

```
sns.countplot(x='Title', data=data);
plt.xticks(rotation=45);
```

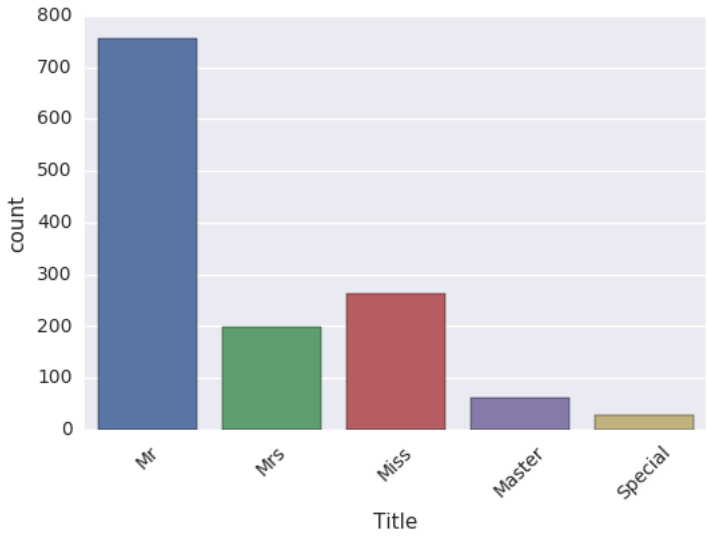



```
# Substitute some title with their English form
data['Title'] = data['Title'].replace({'Mlle':'Miss', 'Mme':'Mrs', 'Ms':'Miss'})
# Gather exceptions
data['Title'] = data['Title'].replace(['Don', 'Dona', 'Rev', 'Dr', 'Major', 'Lady', 'Sir', 'Col', 'Capt', 'Countess', 'Jonkheer'],'Special')
```

```
data.Title.head(3)
```

```
0     Mr
1    Mrs
2   Miss
Name: Title, dtype: object
```

```
sns.countplot(x='Title', data=data);
plt.xticks(rotation=45);
```




```
# View tail of data (for change)
data.tail(3)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Title
415	1307	3	Saether, Mr. Simon Sivertsen	male	38.5	0	0	SOTON/O.Q. 3101262	7.2500	NaN	S	Mr
416	1308	3	Ware, Mr. Frederick	male	NaN	0	0	359309	8.0500	NaN	S	Mr
417	1309	3	Peter, Master. Michael J	male	NaN	1	1	2668	22.3583	NaN	C	Master

Passenger's Cabins

There are several NaNs or missing values in the `Cabin` column. Those NaNs didn't have a cabin, which could tell us something about survival.

```
# View head of data
data[['Name', 'PassengerId', 'Ticket', 'Cabin']].head()
```

	Name	PassengerId	Ticket	Cabin
0	Braund, Mr. Owen Harris	1	A/5 21171	NaN
1	Cumings, Mrs. John Bradley (Florence Briggs Thhttps://ugoproto.github.io/ugo_py_doc.	2	PC 17599	C85
2	Heikkinen, Miss. Laina	3	STON/O2. 3101282	NaN
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	4	113803	C123
4	Allen, Mr. William Henry	5	373450	NaN

```
# Did they have a Cabin?
# Return True is the passenger has a cabin
data['Has_Cabin'] = ~data.Cabin.isnull()

# # View head of data
data[['Name', 'PassengerId', 'Ticket', 'Cabin', 'Has_Cabin']].head()
```

	Name	PassengerId	Ticket	Cabin	Has_Cabin
0	Braund, Mr. Owen Harris	1	A/5 21171	NaN	False
1	Cumings, Mrs. John Bradley (Florence Briggs Thhttps://ugoproto.github.io/ugo_py_doc.	2	PC 17599	C85	True
2	Heikkinen, Miss. Laina	3	STON/O2. 3101282	NaN	False

	Name	PassengerId	Ticket	Cabin	Has_Cabin
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	4	113803	C123	True
4	Allen, Mr. William Henry	5	373450	NaN	False

Drop these columns, except `Has_Cabin`, in the actual `data` DataFrame; make sure to use the `inplace` argument in the `.drop()` method and set it to `True`:

```
# Drop columns and view head
data.drop(['Cabin', 'Name', 'PassengerId', 'Ticket'], axis=1, inplace=True)
data.head()
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	Has_Cabin
0	3	male	22.0	1	0	7.2500	S	Mr	False
1	1	female	38.0	1	0	71.2833	C	Mrs	True
2	3	female	26.0	0	0	7.9250	S	Miss	False
3	1	female	35.0	1	0	53.1000	S	Mrs	True
4	3	male	35.0	0	0	8.0500	S	Mr	False

New features such as `Title` and `Has_Cabin`.

Features that don't add any more useful information for the machine learning model are now dropped from the DataFrame.

Handling Missing Values

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 0 to 417
Data columns (total 9 columns):
Pclass       1309 non-null int64
Sex          1309 non-null object
Age          1046 non-null float64
SibSp        1309 non-null int64
Parch        1309 non-null int64
Fare         1308 non-null float64
Embarked     1307 non-null object
Title        1309 non-null object
Has_Cabin    1309 non-null bool
dtypes: bool(1), float64(2), int64(3), object(3)
memory usage: 133.3+ KB
```

Missing values in `Age`, `Fare`, and `Embarked`. Impute these missing values with the help of `.fillna()` and use the median to fill in the columns (or the mean, the mode, etc.).

```
# Impute missing values for Age, Fare, Embarked
data['Age'] = data.Age.fillna(data.Age.median())
data['Fare'] = data.Fare.fillna(data.Fare.median())
data['Embarked'] = data['Embarked'].fillna('S')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1309 entries, 0 to 417
Data columns (total 9 columns):
Pclass       1309 non-null int64
Sex          1309 non-null object
Age          1309 non-null float64
SibSp        1309 non-null int64
Parch        1309 non-null int64
Fare         1309 non-null float64
Embarked     1309 non-null object
Title        1309 non-null object
Has_Cabin    1309 non-null bool
dtypes: bool(1), float64(2), int64(3), object(3)
memory usage: 133.3+ KB
```

```
data.head(3)
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	Has_Cabin
0	3	male	22.0	1	0	7.2500	S	Mr	False
1	1	female	38.0	1	0	71.2833	C	Mrs	True
2	3	female	26.0	0	0	7.9250	S	Miss	False

Binning Numerical Data

```
# Binning numerical columns
# q=4 means 4 quantiles 0, 1, 2, 3
# labels=False are numbers, not characters
data['CatAge'] = pd.qcut(data.Age, q=4, labels=False )
data['CatFare']= pd.qcut(data.Fare, q=4, labels=False)
data.head(3)
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	Has_Cabin	CatAge	CatFare
0	3	male	22.0	1	0	7.2500	S	Mr	False	0	0
1	1	female	38.0	1	0	71.2833	C	Mrs	True	3	3
2	3	female	26.0	0	0	7.9250	S	Miss	False	1	1

```
# Drop the 'Age' and 'Fare' columns
data = data.drop(['Age', 'Fare'], axis=1)
data.head(3)
```

	Pclass	Sex	SibSp	Parch	Embarked	Title	Has_Cabin	CatAge	CatFare
0	3	male	1	0	S	Mr	False	0	0
1	1	female	1	0	C	Mrs	True	3	3
2	3	female	0	0	S	Miss	False	1	1

Number of Members in Family Onboard

Create a new column, which is the number of members in families that were onboard of the Titanic.

```
# Create column of number of Family members onboard
data['Fam_Size'] = data.Parch + data.SibSp

# Drop columns
data = data.drop(['SibSp','Parch'], axis=1)
data.head(3)
```

	Pclass	Sex	Embarked	Title	Has_Cabin	CatAge	CatFare	Fam_Size
0	3	male	S	Mr	False	0	0	1
1	1	female	C	Mrs	True	3	3	1
2	3	female	S	Miss	False	1	1	0

Transforming all Variables into Numerical Variables

Transform all variables into numeric ones. We do this because machine learning models generally take numeric input.

```
# Transform into binary variables
# Has_Cabin is a boolean
# Sex becomes Sex_male=1 or 0
# Embarked becomes Embarked_Q=1 or 0, Embarked_https://ugoproto.github.io/ugo_py_doc.
# Title becomes Title_Miss=1 or 0, https://ugoproto.github.io/ugo_py_doc.
# The former variables are dropped, only the later variables remain
data_dum = pd.get_dummies(data, drop_first=True)
data_dum.head(3)
```

	Pclass	Has_Cabin	CatAge	CatFare	Fam_Size	Sex_male	Embarked_Q	Embarked_S	Title_Miss	Title_Mr	Title_Mrs	Title_Special
0	3	False	0	0	1	1	0	1	0	1	0	0
1	1	True	3	3	1	0	0	0	0	0	1	0
2	3	False	1	1	0	0	0	1	1	0	0	0

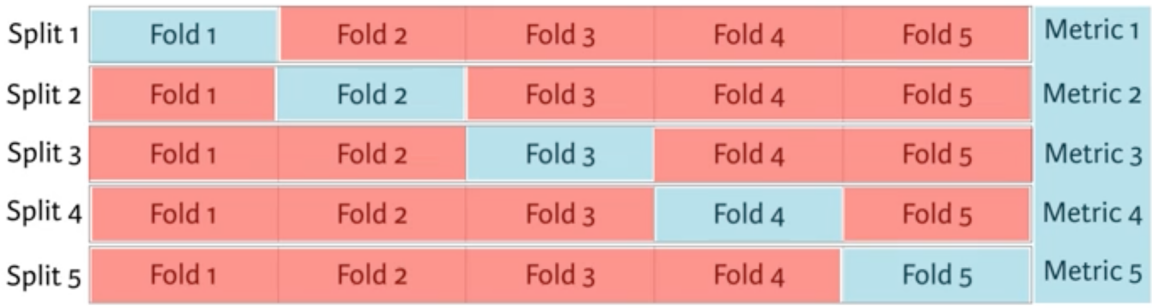
First, split the data back into training and test sets. Then, transform them into arrays:

```
# Split into test.train
data_train = data_dum.iloc[:891]
data_test = data_dum.iloc[891:]

# Transform into arrays for scikit-learn
```

```
X = data_train.values
test = data_test.values
y = survived_train.values
```

Building models with a New Dataset!

Build a decision tree on a brand new feature-engineered dataset. To choose the hyperparameter `max_depth`, we will use a variation on test train split called "cross validation".



Split the dataset into 5 groups or folds. Then we hold out the first fold as a test set, fit the model on the remaining four folds, predict on the test set and compute the metric of interest. Next, we hold out the second fold as the test set, fit on the remaining data, predict on the test set and compute the metric of interest. Then similarly with the third, fourth and fifth.

As a result, we get five values of accuracy, from which we can compute statistics of interest, such as the median and/or mean and 95% confidence intervals.

We do this for each value of each hyperparameter that we are tuning and choose the set of hyperparameters that performs the best. This is called grid search.

In the following, we will use cross validation and grid search to choose the best `max_depth` for the new feature-engineered dataset:

```
# Setup the hyperparameter grid
dep = np.arange(1,9)
param_grid = {'max_depth' : dep}

# Instantiate a decision tree classifier: clf
clf = tree.DecisionTreeClassifier()

# Instantiate the GridSearchCV object: clf_cv
clf_cv = GridSearchCV(clf, param_grid=param_grid, cv=5)

# Fit it to the data
clf_cv.fit(X, y)

# Print the tuned parameter and score
print("Tuned Decision Tree Parameters: {}".format(clf_cv.best_params_))
print("Best score is {}".format(clf_cv.best_score_))
```

```
Tuned Decision Tree Parameters: {'max_depth': 3}
Best score is 0.8294051627384961
```

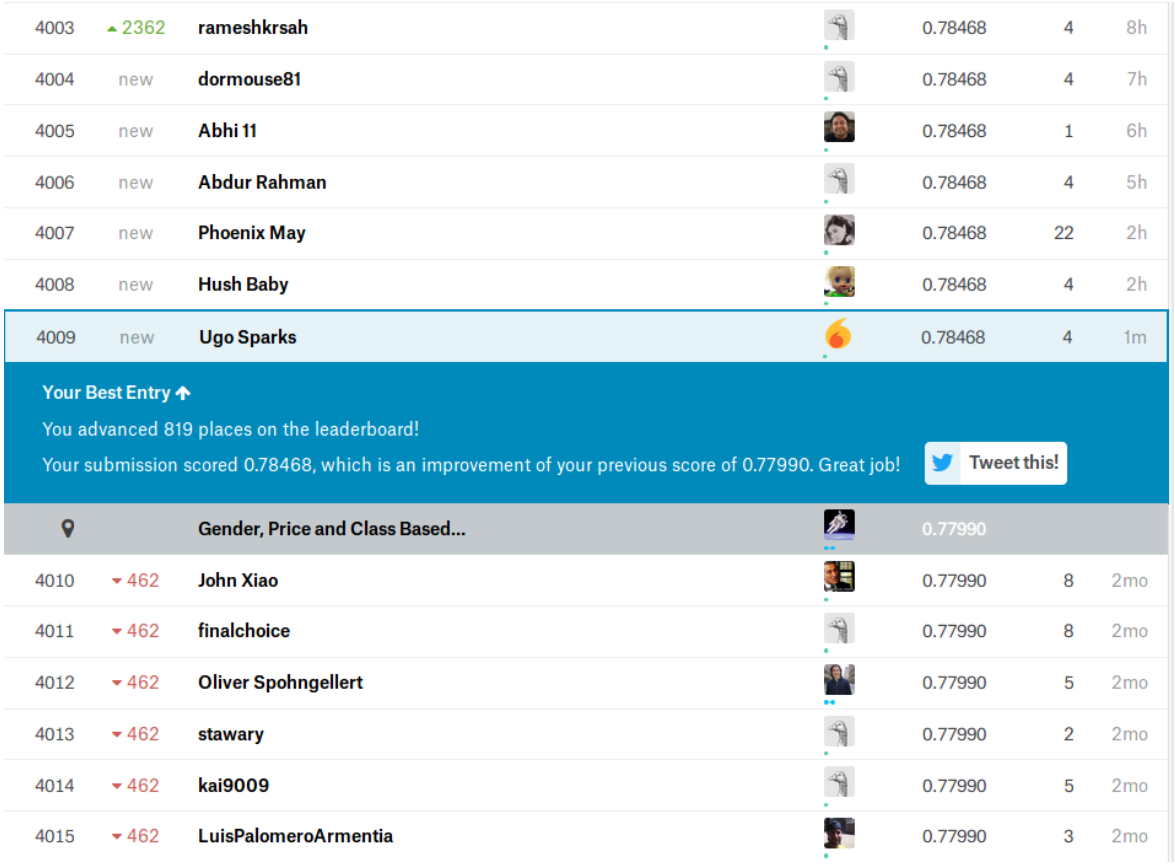
Make predictions on the test set, create a new column `Survived` and store the predictions in it.

Save the `PassengerId` and `Survived` columns of `df_test` to a .csv and submit it to Kaggle.

```
Y_pred = clf_cv.predict(test)
df_test['Survived'] = Y_pred
df_test[['PassengerId', 'Survived']].to_csv('results/dec_tree_feat_eng.csv', index=False)
```

Submit to Kaggle (4th)

- Go to Kaggle, log in, and search for *Titanic: Machine Learning from Disaster*.
- Join the competition and submit the .csv file.
- Add a description and submit.
- Kaggle returns a ranking.
- At the time of the first submission: score 0.78468 (from 0.77980), rank 4009 (a jump of 819 places).



4003	▲2362	rameshkrsah		0.78468	4	8h
4004	new	dormouse81		0.78468	4	7h
4005	new	Abhi 11		0.78468	1	6h
4006	new	Abdur Rahman		0.78468	4	5h
4007	new	Phoenix May		0.78468	22	2h
4008	new	Hush Baby		0.78468	4	2h
4009	new	Ugo Sparks		0.78468	4	1m

		Gender, Price and Class Based...		0.77990		
4010	▼462	John Xiao		0.77990	8	2mo
4011	▼462	finalchoice		0.77990	8	2mo
4012	▼462	Oliver Spohngellert		0.77990	5	2mo
4013	▼462	stawary		0.77990	2	2mo
4014	▼462	kai9009		0.77990	5	2mo
4015	▼462	LuisPalomeroArmentia		0.77990	3	2mo

Exploratory Data Analysis (EDA)

Code snippets and excerpts from the tutorial. Python 3. From DataCamp.

Import the Data

- `sep`, delimiter.
- `delimiter`, delimiter.
- `names`, column names to use.
- `index_col`, column to use as the row labels.
- `read_table()`, general delimited files.
- `read_excel()`, Excel files.
- `read_fwf()`, Fixed-Width Formatted data.
- `read_clipboard`, data copied to the clipboard.
- `read_sql()`, SQL query.

Input-output documentation.

```
%pylab inline
import numpy as np
import pandas as pd
```

```
1 Populating the interactive namespace from numpy and matplotlib
```

```
digits
```

```
# Load in the data with `read_csv()`
digits = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/optdigits.tra",
       header=None)

digits.head()
```

	0	1	2	3	4	5	6	7	8	9	https://ugoproto.github.io/ugo_py_doc.	55	56	57	58	59	60	61	62	63	64
0	0	1	6	15	12	1	0	0	0	7	https://ugoproto.github.io/ugo_py_doc.	0	0	0	6	14	7	1	0	0	0
1	0	0	10	16	6	0	0	0	0	7	https://ugoproto.github.io/ugo_py_doc.	0	0	0	10	16	15	3	0	0	0
2	0	0	8	15	16	13	0	0	0	1	https://ugoproto.github.io/ugo_py_doc.	0	0	0	9	14	0	0	0	0	7
3	0	0	0	3	11	16	0	0	0	0	https://ugoproto.github.io/ugo_py_doc.	0	0	0	0	1	15	2	0	0	4
4	0	0	5	14	4	0	0	0	0	0	https://ugoproto.github.io/ugo_py_doc.	0	0	0	4	12	14	7	0	0	6

5 rows × 65 columns

Find out about the dataset.

`iris`

Another classical dataset.

```
iris = pd.read_csv("http://mlr.cs.umass.edu/ml/machine-learning-databases/iris/iris.data")

iris.columns = ['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width', 'Class']

iris.head()
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

Basic Description of the Data

Describing The Data

```
iris.dtypes
```

```
Sepal_Length    float64
Sepal_Width     float64
Petal_Length    float64
Petal_Width     float64
Class            object
dtype: object
```

```
def get_var_category(series):
unique_count = series.nunique(dropna=False)
total_count = len(series)
if pd.api.types.is_numeric_dtype(series):
return 'Numerical'
elif pd.api.types.is_datetime64_dtype(series):
return 'Date'
elif unique_count==total_count:
return 'Text (Unique)'
else:
return 'Categorical'

def print_categories(df):
for column_name in df.columns:
print(column_name, ": ", get_var_category(df[column_name]))
```

```
print_categories(iris)
```

```
Sepal_Length :  Numerical
Sepal_Width :  Numerical
```

```
Petal_Length :  Numerical
Petal_Width :  Numerical
Class :  Categorical
```

```
digits.describe()
```

	0	1	2	3	4	5	6	7	8	9	https://(
count	3823.0	3823.000000	3823.000000	3823.000000	3823.000000	3823.000000	3823.000000	3823.000000	3823.000000	3823.000000	https://(
mean	0.0	0.301334	5.481821	11.805912	11.451478	5.505362	1.387392	0.142297	0.002093	1.960502	https://(
std	0.0	0.866986	4.631601	4.259811	4.537556	5.613060	3.371444	1.051598	0.088572	3.052353	https://(
min	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	https://(
25%	0.0	0.000000	1.000000	10.000000	9.000000	0.000000	0.000000	0.000000	0.000000	0.000000	https://(
50%	0.0	0.000000	5.000000	13.000000	13.000000	4.000000	0.000000	0.000000	0.000000	0.000000	https://(
75%	0.0	0.000000	9.000000	15.000000	15.000000	10.000000	0.000000	0.000000	0.000000	3.000000	https://(
max	0.0	8.000000	16.000000	16.000000	16.000000	16.000000	16.000000	16.000000	5.000000	15.000000	https://(

8 rows × 65 columns

```
iris.describe()
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
count	149.000000	149.000000	149.000000	149.000000
mean	5.848322	3.051007	3.774497	1.205369
std	0.828594	0.433499	1.759651	0.761292
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.400000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
iris[["Sepal_Length", "Sepal_Width"]].describe()
```

	Sepal_Length	Sepal_Width
count	149.000000	149.000000
mean	5.848322	3.051007
std	0.828594	0.433499
min	4.300000	2.000000
25%	5.100000	2.800000
50%	5.800000	3.000000
75%	6.400000	3.300000
max	7.900000	4.400000

```
length = len(digits)
print(length)
```

```
3823
```

```
count = digits[2].count()
print(count)
```

```
3823
```

```
number_of_missing_values = length - count
pct_of_missing_values = float(number_of_missing_values / length)
pct_of_missing_values = "{0:.1f}%".format(pct_of_missing_values*100)
print(pct_of_missing_values)
```

```
0.0%
```

```
print("Minimum value: ", iris["Sepal_Length"].min())
print("Maximum value: ", iris["Sepal_Length"].max())
```

```
Minimum value:  4.3
Maximum value:  7.9
```

```
print(iris["Sepal_Length"].mode())
```

```
0    5.0
dtype: float64
```

```
print(iris["Sepal_Length"].mean())
```

```
5.848322147651008
```

```
print(iris["Sepal_Length"].median())
```

```
5.8
```

```
print(iris["Sepal_Length"].std())
```

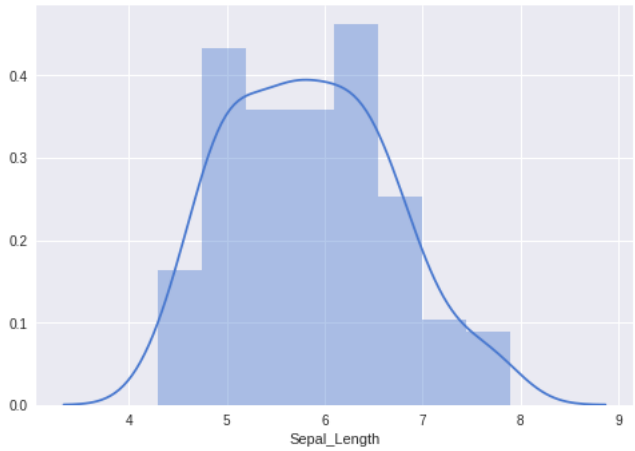
```
0.8285940572656172
```

```
print(iris["Sepal_Length"].quantile([.25, .5, .75]))
```

```
0.25    5.1
0.50    5.8
0.75    6.4
Name: Sepal_Length, dtype: float64
```

```
import seaborn as sns
sns.set(color_codes=True)

sns.set_palette(sns.color_palette("muted"))

sns.distplot(iris["Sepal_Length"].dropna())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f87b39b0320>
```




```
iris[["Sepal_Length", "Sepal_Width"]].corr()
```

	Sepal_Length	Sepal_Width
Sepal_Length	1.000000	-0.103784
Sepal_Width	-0.103784	1.000000

```
import pandas_profiling

# Print a full report
pandas_profiling.ProfileReport(iris)
```

Overview

Dataset info

Number of variables	5
Number of observations	149
Total Missing (%)	0.0%
Total size in memory	5.9 KiB
Average record size in memory	40.5 B

Variables types

Numeric	3
Categorical	1
Date	0
Text (Unique)	0
Rejected	1

Warnings

- `Petal_Width` is highly correlated with `Petal_Length` (ρ = 0.96231) Rejected
- Dataset has 3 duplicate rows Warning

Variables

Class

Categorical

Distinct count	3
Unique (%)	2.0%
Missing (%)	0.0%
Missing (n)	0

Toggle details

Value	Count	Frequency (%)	
Iris-virginica	50	33.6%	
Iris-versicolor	50	33.6%	
Iris-setosa	49	32.9%	

Petal_Length

Numeric

Distinct count	43
Unique (%)	28.9%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	3.7745
Minimum	1
Maximum	6.9
Zeros (%)	0.0%

Toggle details

- Statistics
- Histogram
- Common Values
- Extreme Values

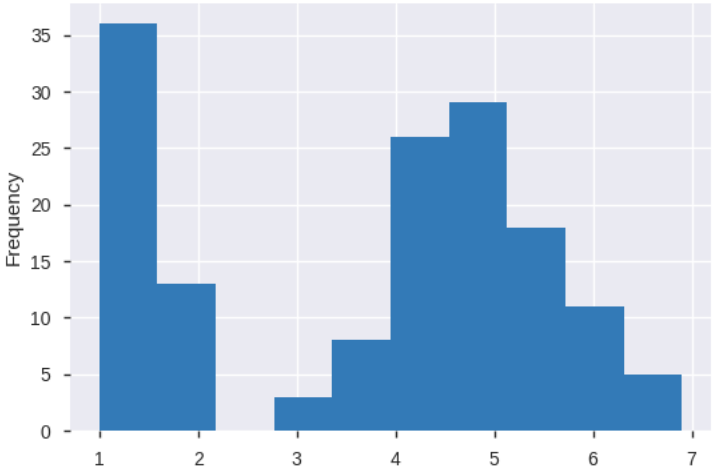
Quantile statistics

Minimum	1
5-th percentile	1.3
Q1	1.6
Median	4.4
Q3	5.1
95-th percentile	6.1
Maximum	6.9
Range	5.9
Interquartile range	3.5

Descriptive statistics

Standard deviation	1.7597
Coef of variation	0.46619
Kurtosis	-1.385
Mean	3.7745
MAD	1.5526

Skewness	-0.28946
Sum	562.4
Variance	3.0964
Memory size	1.2 KiB



Value	Count	Frequency (%)
1.5	14	9.4%
1.4	11	7.4%
5.1	8	5.4%
4.5	8	5.4%
1.3	7	4.7%
1.6	7	4.7%
5.6	6	4.0%
4.0	5	3.4%
4.9	5	3.4%
4.7	5	3.4%
Other values (33)	73	49.0%

Minimum 5 values

Value	Count	Frequency (%)
1.0	1	0.7%
1.1	1	0.7%
1.2	2	1.3%
1.3	7	4.7%
1.4	11	7.4%

Maximum 5 values

Value	Count	Frequency (%)
6.3	1	0.7%
6.4	1	0.7%
6.6	1	0.7%
6.7	2	1.3%
6.9	1	0.7%

`Petal_Width`

Highly correlated

This variable is highly correlated with `Petal_Length` *and should be ignored for analysis*

Correlation 0.96231

Sepal_Length

Numeric

Distinct count	35
Unique (%)	23.5%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	5.8483
Minimum	4.3
Maximum	7.9
Zeros (%)	0.0%

Toggle details

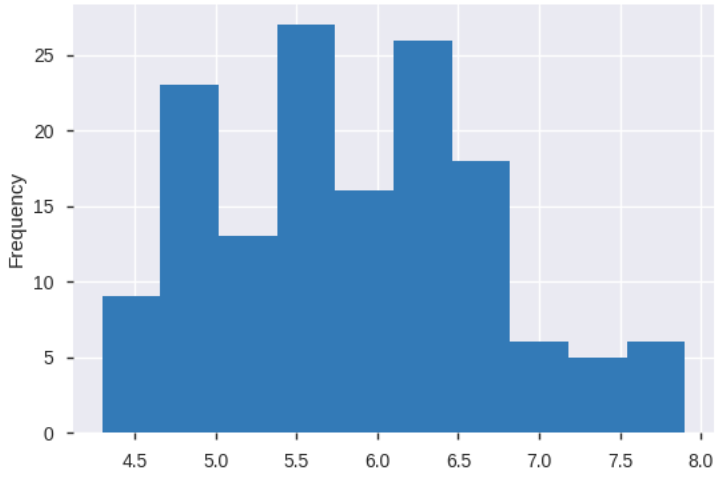
- Statistics
- Histogram
- Common Values
- Extreme Values

Quantile statistics

Minimum	4.3
5-th percentile	4.6
Q1	5.1
Median	5.8
Q3	6.4
95-th percentile	7.26
Maximum	7.9
Range	3.6
Interquartile range	1.3

Descriptive statistics

Standard deviation	0.82859
Coef of variation	0.14168
Kurtosis	-0.55356
Mean	5.8483
MAD	0.68748
Skewness	0.3031
Sum	871.4
Variance	0.68657
Memory size	1.2 KiB



Value	Count	Frequency (%)
5.0	10	6.7%
6.3	9	6.0%
5.1	8	5.4%
6.7	8	5.4%
5.7	8	5.4%
5.5	7	4.7%
5.8	7	4.7%
6.4	7	4.7%
6.0	6	4.0%
4.9	6	4.0%
Other values (25)	73	49.0%

Minimum 5 values

Value	Count	Frequency (%)
4.3	1	0.7%
4.4	3	2.0%
4.5	1	0.7%
4.6	4	2.7%
4.7	2	1.3%

Maximum 5 values

Value	Count	Frequency (%)
7.3	1	0.7%
7.4	1	0.7%
7.6	1	0.7%
7.7	4	2.7%
7.9	1	0.7%

Sepal_Width

Numeric

Distinct count	23
Unique (%)	15.4%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0
Mean	3.051
Minimum	2
Maximum	4.4

Zeros (%)	0.0%

Toggle details

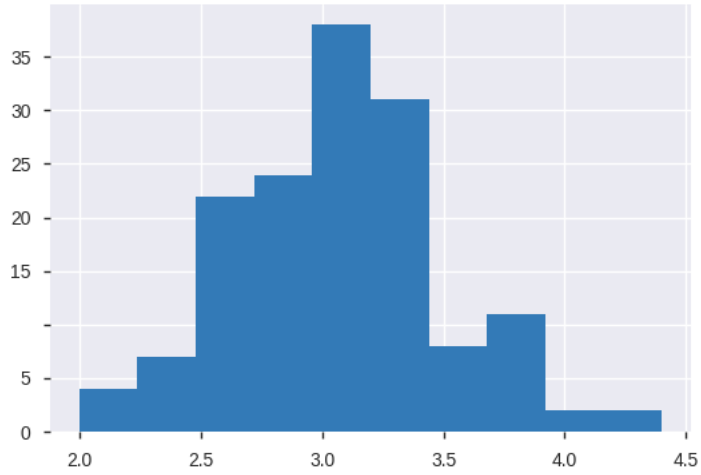
- Statistics
- Histogram
- Common Values
- Extreme Values

Quantile statistics

Minimum	2
5-th percentile	2.34
Q1	2.8
Median	3
Q3	3.3
95-th percentile	3.8
Maximum	4.4
Range	2.4
Interquartile range	0.5

Descriptive statistics

Standard deviation	0.4335
Coef of variation	0.14208
Kurtosis	0.31865
Mean	3.051
MAD	0.33199
Skewness	0.3501
Sum	454.6
Variance	0.18792
Memory size	1.2 KiB



Value	Count	Frequency (%)	
3.0	26	17.4%	
2.8	14	9.4%	
3.2	13	8.7%	
3.4	12	8.1%	
3.1	12	8.1%	
2.9	10	6.7%	
2.7	9	6.0%	
2.5	8	5.4%	
3.8	6	4.0%	

Value	Count	Frequency (%)	
3.3	6	4.0%	
Other values (13)	33	22.1%	

Minimum 5 values

Value	Count	Frequency (%)	
2.0	1	0.7%	
2.2	3	2.0%	
2.3	4	2.7%	
2.4	3	2.0%	
2.5	8	5.4%	

Maximum 5 values

Value	Count	Frequency (%)	
3.9	2	1.3%	
4.0	1	0.7%	
4.1	1	0.7%	
4.2	1	0.7%	
4.4	1	0.7%	

Sample

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa

```
# Print a full report
pandas_profiling.ProfileReport(digits)
```

>>> Full (lengthy) report here!!! <<<

First and Last DataFrame Rows

```
# Inspect the first 5 rows of `digits`
first = digits.head(5)

# Inspect the last 3 rows
last = digits.tail(3)
```

```
print(first)
```

```
   0   1   2   3   4   5   6   7   8   9  https://ugoproto.github.io/ugo_py_doc.  55  56  57  58  59  60  61  62  \
0  0   1   6  15  12   1   0   0   0   7 https://ugoproto.github.io/ugo_py_doc.   0   0   0   6  14   7   1   0
1  0   0  10  16   6   0   0   0   0   7 https://ugoproto.github.io/ugo_py_doc.   0   0   0  10  16  15   3   0
2  0   0   8  15  16  13   0   0   0   1 https://ugoproto.github.io/ugo_py_doc.   0   0   0   9  14   0   0   0
3  0   0   0   3  11  16   0   0   0   0 https://ugoproto.github.io/ugo_py_doc.   0   0   0   0   1  15   2   0
4  0   0   5  14   4   0   0   0   0   0 https://ugoproto.github.io/ugo_py_doc.   0   0   0   4  12  14   7   0

  63  64
0  0   0
```

```
1   0   0
2   0   7
3   0   4
4   0   6

[5 rows x 65 columns]
```

```
print(last)
```

```
      0   1   2   3   4   5   6   7   8   9  https://ugoproto.github.io/ugo_py_doc.  55  56  57  58  59  60  61  \
3820   0   0   3  15   0   0   0   0   0   0 https://ugoproto.github.io/ugo_py_doc.   0   0   0   4  14  16   9
3821   0   0   6  16   2   0   0   0   0   0 https://ugoproto.github.io/ugo_py_doc.   0   0   0   5  16  16  16
3822   0   0   2  15  16  13   1   0   0   0 https://ugoproto.github.io/ugo_py_doc.   0   0   0   4  14   1   0

62  63  64
3820   0   0   6
3821   5   0   6
3822   0   0   7

[3 rows x 65 columns]
```

Sample the Data

```
# Take a sample of 5
digits.sample(5)
```

	0	1	2	3	4	5	6	7	8	9	https://ugoproto.github.io/ugo_py_doc.	55	56	57	58	59	60	61	62	63	64
1249	0	0	14	14	13	15	5	0	0	0	https://ugoproto.github.io/ugo_py_doc.	0	0	0	12	16	10	2	0	0	5
3702	0	0	0	9	16	12	2	0	0	0	https://ugoproto.github.io/ugo_py_doc.	0	0	0	0	9	14	2	0	0	0
1605	0	0	7	16	13	2	0	0	0	2	https://ugoproto.github.io/ugo_py_doc.	0	0	0	5	14	11	1	0	0	0
1890	0	0	3	15	15	5	0	0	0	0	https://ugoproto.github.io/ugo_py_doc.	2	0	0	3	15	16	16	13	1	9
1295	0	0	7	15	13	3	0	0	0	0	https://ugoproto.github.io/ugo_py_doc.	0	0	0	9	13	12	3	0	0	0

5 rows × 65 columns

```
# import `sample` from `random`
from random import sample

# Create a random index
randomIndex = np.array(sample(range(len(digits)), 5))

print(randomIndex)
```

```
[ 846  569  315 2932 2328]
```

```
# Get 5 random rows
digitsSample = digits.ix[randomIndex]

# Print the sample
print(digitsSample)
```

```
      0   1   2   3   4   5   6   7   8   9  https://ugoproto.github.io/ugo_py_doc.  55  56  57  58  59  60  61  \
846    0   5  14  15   9   1   0   0   0   7 https://ugoproto.github.io/ugo_py_doc.   0   0   4  12  16  12  10
569    0   1   7  12  12   0   0   0   0   3 https://ugoproto.github.io/ugo_py_doc.   0   0   0  10  16  13   7
315    0   1   6  13  13   4   0   0   0   9 https://ugoproto.github.io/ugo_py_doc.   0   0   0   4  14  16   9
2932   0   0   4  12  10   1   0   0   0   0 https://ugoproto.github.io/ugo_py_doc.   0   0   0   4  12  11   3
2328   0   0   4  15  16  16  16  15   0   0 https://ugoproto.github.io/ugo_py_doc.   0   0   0   5  15   3   0

62  63  64
846    4   0   2
569    0   0   3
315    2   0   2
2932   0   0   0
2328   0   0   7

[5 rows x 65 columns]
```

14
15

Queries

```
iris.head(2)
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa

```
# Petal length greater than sepal length?
iris.query('Petal_Length > Sepal_Length')
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class

```
# reverse
iris.query('Sepal_Length > Petal_Length')
```

	Sepal_Length	Sepal_Width	
0	4.9	3.0	1.4
1	4.7	3.2	1.3
2	4.6	3.1	1.5
3	5.0	3.6	1.4
4	5.4	3.9	1.7
5	4.6	3.4	1.4
6	5.0	3.4	1.5
7	4.4	2.9	1.4
8	4.9	3.1	1.5
9	5.4	3.7	1.5
10	4.8	3.4	1.6
11	4.8	3.0	1.4
12	4.3	3.0	1.1
13	5.8	4.0	1.2
14	5.7	4.4	1.5
15	5.4	3.9	1.3
16	5.1	3.5	1.4
17	5.7	3.8	1.7
18	5.1	3.8	1.5
19	5.4	3.4	1.7
20	5.1	3.7	1.5
21	4.6	3.6	1.0
22	5.1	3.3	1.7
23	4.8	3.4	1.9
24	5.0	3.0	1.6
25	5.0	3.4	1.6
26	5.2	3.5	1.5
27	5.2	3.4	1.4
28	4.7	3.2	1.6
29	4.8	3.1	1.6
https://ugoproto.github.io/ugo_py_doc.	https://ugoproto.github.io/ugo_py_doc.	https://ugoproto.github.io/ugo_py_doc.	https://ugoproto.githu
119	6.9	3.2	5.7
120	5.6	2.8	4.9
121	7.7	2.8	6.7
122	6.3	2.7	4.9
123	6.7	3.3	5.7

	Sepal_Length	Sepal_Width	
124	7.2	3.2	6.0
125	6.2	2.8	4.8
126	6.1	3.0	4.9
127	6.4	2.8	5.6
128	7.2	3.0	5.8
129	7.4	2.8	6.1
130	7.9	3.8	6.4
131	6.4	2.8	5.6
132	6.3	2.8	5.1
133	6.1	2.6	5.6
134	7.7	3.0	6.1
135	6.3	3.4	5.6
136	6.4	3.1	5.5
137	6.0	3.0	4.8
138	6.9	3.1	5.4
139	6.7	3.1	5.6
140	6.9	3.1	5.1
141	5.8	2.7	5.1
142	6.8	3.2	5.9
143	6.7	3.3	5.7
144	6.7	3.0	5.2
145	6.3	2.5	5.0
146	6.5	3.0	5.2
147	6.2	3.4	5.4
148	5.9	3.0	5.1

149 rows × 5 columns

```
# alternatively
iris[iris.Sepal_Length > iris.Petal_Length]
```

	Sepal_Length	Sepal_Width	
0	4.9	3.0	1.4
1	4.7	3.2	1.3
2	4.6	3.1	1.5
3	5.0	3.6	1.4
4	5.4	3.9	1.7
5	4.6	3.4	1.4
6	5.0	3.4	1.5
7	4.4	2.9	1.4
8	4.9	3.1	1.5
9	5.4	3.7	1.5
10	4.8	3.4	1.6
11	4.8	3.0	1.4
12	4.3	3.0	1.1
13	5.8	4.0	1.2
14	5.7	4.4	1.5
15	5.4	3.9	1.3
16	5.1	3.5	1.4
17	5.7	3.8	1.7
18	5.1	3.8	1.5
19	5.4	3.4	1.7
20	5.1	3.7	1.5
21	4.6	3.6	1.0

	Sepal_Length	Sepal_Width	
22	5.1	3.3	1.7
23	4.8	3.4	1.9
24	5.0	3.0	1.6
25	5.0	3.4	1.6
26	5.2	3.5	1.5
27	5.2	3.4	1.4
28	4.7	3.2	1.6
29	4.8	3.1	1.6
https://ugoproto.github.io/ugo_py_doc.	https://ugoproto.github.io/ugo_py_doc.	https://ugoproto.github.io/ugo_py_doc.	https://ugoproto.githu
119	6.9	3.2	5.7
120	5.6	2.8	4.9
121	7.7	2.8	6.7
122	6.3	2.7	4.9
123	6.7	3.3	5.7
124	7.2	3.2	6.0
125	6.2	2.8	4.8
126	6.1	3.0	4.9
127	6.4	2.8	5.6
128	7.2	3.0	5.8
129	7.4	2.8	6.1
130	7.9	3.8	6.4
131	6.4	2.8	5.6
132	6.3	2.8	5.1
133	6.1	2.6	5.6
134	7.7	3.0	6.1
135	6.3	3.4	5.6
136	6.4	3.1	5.5
137	6.0	3.0	4.8
138	6.9	3.1	5.4
139	6.7	3.1	5.6
140	6.9	3.1	5.1
141	5.8	2.7	5.1
142	6.8	3.2	5.9
143	6.7	3.3	5.7
144	6.7	3.0	5.2
145	6.3	2.5	5.0
146	6.5	3.0	5.2
147	6.2	3.4	5.4
148	5.9	3.0	5.1

149 rows × 5 columns

The Challenges of Data

Missing Values

```
# Identifiy missing values
pd.isnull(digits)
```

	0	1	
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False

	0	1	
4	False	False	False
5	False	False	False
6	False	False	False
7	False	False	False
8	False	False	False
9	False	False	False
10	False	False	False
11	False	False	False
12	False	False	False
13	False	False	False
14	False	False	False
15	False	False	False
16	False	False	False
17	False	False	False
18	False	False	False
19	False	False	False
20	False	False	False
21	False	False	False
22	False	False	False
23	False	False	False
24	False	False	False
25	False	False	False
26	False	False	False
27	False	False	False
28	False	False	False
29	False	False	False
https://ugoproto.github.io/ugo_py_doc.	https://ugoproto.github.io/ugo_py_doc.	https://ugoproto.github.io/ugo_py_doc.	https://ugoproto.githu
3793	False	False	False
3794	False	False	False
3795	False	False	False
3796	False	False	False
3797	False	False	False
3798	False	False	False
3799	False	False	False
3800	False	False	False
3801	False	False	False
3802	False	False	False
3803	False	False	False
3804	False	False	False
3805	False	False	False
3806	False	False	False
3807	False	False	False
3808	False	False	False
3809	False	False	False
3810	False	False	False
3811	False	False	False
3812	False	False	False
3813	False	False	False
3814	False	False	False
3815	False	False	False
3816	False	False	False
3817	False	False	False
3818	False	False	False

	0	1	
3819	False	False	False
3820	False	False	False
3821	False	False	False
3822	False	False	False

3823 rows × 65 columns

Delete

```
# Drop rows with missing values
df.dropna(axis=0)

# Drop columns with missing values
df.dropna(axis=1)
```

Impute

Imputation: mean, median, another variable, estimate with regression ANOVA, logit, k-NN.

```
# Import NumPy
import numpy as np

# Calculate the mean of the DataFrame variable Salary
mean = np.mean(df.Salary)

# Replace missing values with the mean
df = df.Salary.fillna(mean)

df = df.Salary.fillna(mean, method=ffill)
```

- `ffill` and `bfill` for forward and backward fill.

```
from scipy import interpolate

# Fill the DataFrame
df.interpolate()

df.interpolate(method=cubic)
```

- `cubic`, `polynomial`.
- `limit` and `limit_direction`.

Outliers

Delete (data entry, processing errors), transform (assign weights, natural log to reduce variation) or impute them (replace extreme values with median, mean or mode values).

The Data's Features

Feature Engineering

Increase the predictive power of learning algorithms by creating features from raw data that will help the learning process.

Encode categorical variables into numerical ones

```
# Factorize the values
labels,levels = pd.factorize(iris.Class)

# Save the encoded variables in `iris.Class`
iris.Class = labels

# Print out the first rows
iris.Class.head()
```

```
0    0
1    0
2    0
3    0
```

```
4    0
Name: Class, dtype: int64
```

Bin continuous variables in groups

```
# Define the bins
mybins = range(0, df.age.max(), 10)

# Cut the data from the DataFrame with the help of the bins
df['age_bucket'] = pd.cut(df.age, bins=mybins)

# Count the number of values per bucket
df['age_bucket'].value_counts()
```

Scale features

Center the data around 0.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(X)

rescaledX = scaler.transform(X)
```

Feature Selection

Select the key subset of original data features in an attempt to reduce the dimensionality of the training problem.

PCA combines similar (correlated) attributes and creates new ones that are considered superior to the original attributes of the dataset.

Feature selection doesn't combine attributes: it evaluates the quality and predictive power and selects the best set.

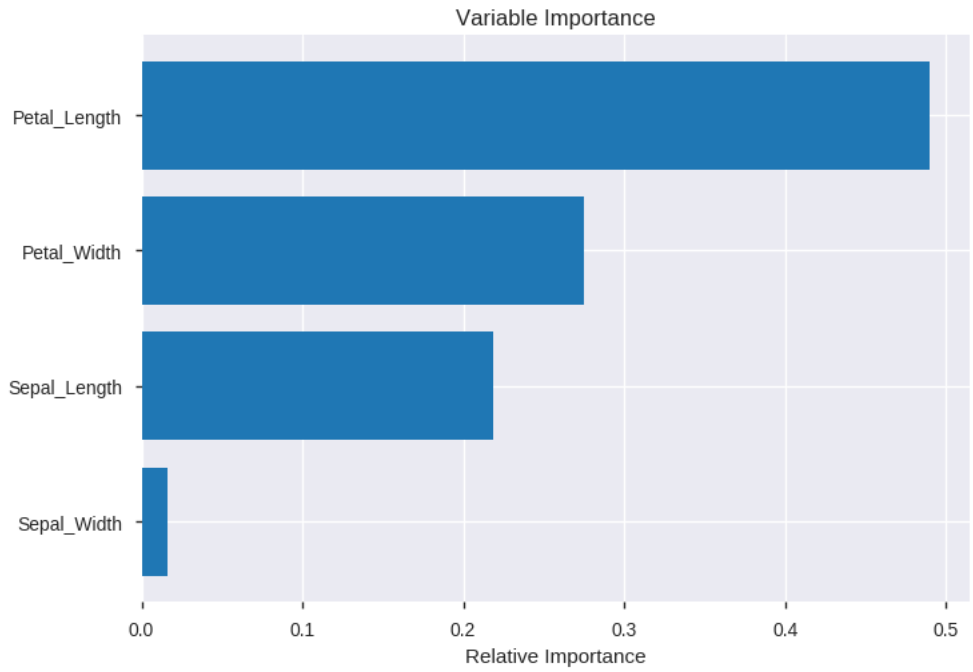
To find important features, calculate how much better or worse a model does when we leave one variable out of the equation.

```
# Import `RandomForestClassifier`
from sklearn.ensemble import RandomForestClassifier

# Isolate Data, class labels and column values
X = iris.iloc[:,0:4]
Y = iris.iloc[:,-1]
names = iris.columns.values

# Build the model
rfc = RandomForestClassifier()

# Fit the model
rfc.fit(X, Y)

# Print the results
print("Features sorted by their score:")
print(sorted(zip(map(lambda x: round(x, 4), rfc.feature_importances_), names), reverse=True))
```

```
Features sorted by their score:
[(0.4899, 'Petal_Length'), (0.2752, 'Petal_Width'), (0.2185, 'Sepal_Length'), (0.016400000000000001, 'Sepal_Width')]
```

The best feature set is one that includes the petal length and petal width data.

```
# Isolate feature importances
importance = rfc.feature_importances_

# Sort the feature importances
sorted_importances = np.argsort(importance)

# Insert padding
padding = np.arange(len(names)-1) + 0.5

# Plot the data
plt.barh(padding, importance[sorted_importances], align='center')

# Customize the plot
plt.yticks(padding, names[sorted_importances])
plt.xlabel("Relative Importance")
plt.title("Variable Importance")

# Show the plot
plt.show()
```



Patterns In the Data

Visualization of the data; static with Matplotlib or Seaborn, interactive with Bokeh or Plotly.

Correlation Identification with PCA from scikit-learn

Matplotlib

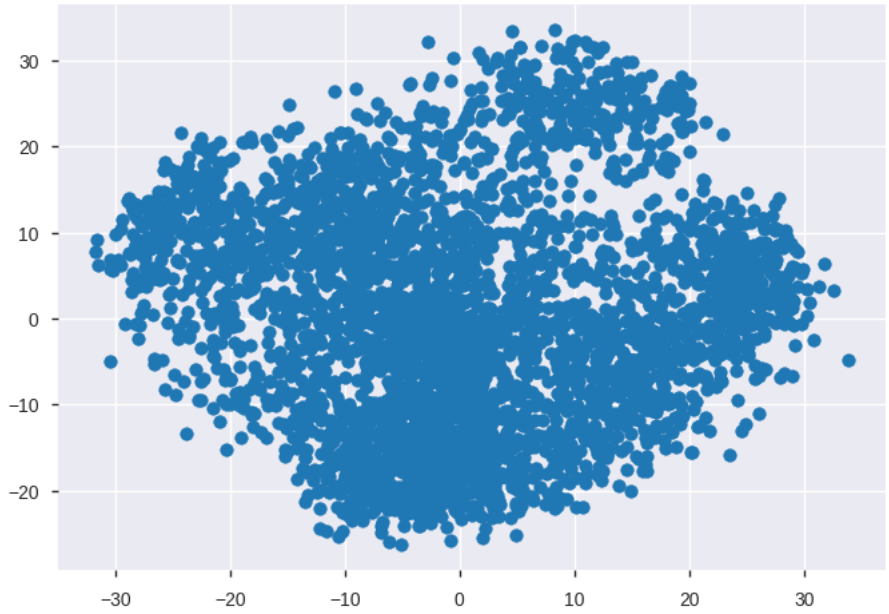
Dimensionality Reduction techniques, such as Principal Component Analysis (PCA). From 'many' to two 'principal components'.

```
# Import `PCA` from `sklearn.decomposition`
from sklearn.decomposition import PCA

# Build the model
pca = PCA(n_components=2)

# Reduce the data, output is ndarray
reduced_data = pca.fit_transform(digits)

# Inspect shape of the `reduced_data`
reduced_data.shape

# print out the reduced data
print(reduced_data)
```

```
[[ 12.65674168  -4.63610357]
 [ 16.82906354 -12.96575346]
 [-19.08072301  10.58293767]
 https://ugoproto.github.io/ugo_py_doc.,
 [ 23.90693984   6.06265415]
 [ 29.1798759   -3.06847144]
 [-25.23132536  11.60863909]]
```

```
reduced_data = pd.DataFrame(reduced_data)
```

```
import matplotlib.pyplot as plt

plt.scatter(reduced_data[0], reduced_data[1])

plt.show()
```

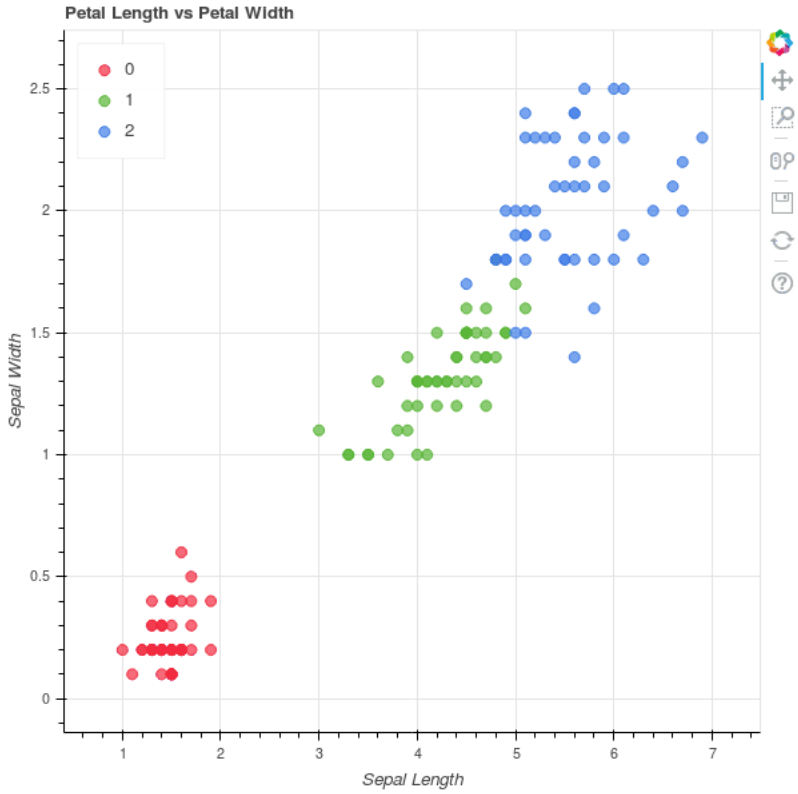


Bokeh

To be implemented in a webpage for example.

```
from bokeh.charts import Scatter, output_file, show

# Construct the scatter plot
p = Scatter(iris, x='Petal_Length', y='Petal_Width', color="Class", title="Petal Length vs Petal Width",
xlabel="Sepal Length", ylabel="Sepal Width")

# Output the file
output_file('scatter.html')

# Show the scatter plot
show(p)
```

The GIF output:



Correlation Identification with Pandas

The Pearson correlation assumes that the variables are normally distributed, that there is a straight line relationship between each of the variables and that the data is normally distributed about the regression line.

The Spearman correlation, on the other hand, assumes that we have two ordinal variables or two variables that are related in some way, but not linearly. The Spearman coefficient is the sum of deviation squared by `n` times `n` minus 1.

The Kendall Tau correlation is a coefficient that represents the degree of concordance between two columns of ranked data. We can use the Spearman correlation to measure the degree of association between two variables. The Kendal Tau coefficient is calculated by the number of concordant pairs minus the number of discordant pairs divided by the total number of pairs.

Spearman's coefficient will usually be larger than the Kendall's Tau coefficient, but this is not always the case: we'll get a smaller Spearman's coefficient when the deviations are huge among the observations of the data. The Spearman correlation is very sensitive to this and this might come in handy in some cases!

The two last correlation measures require ranking the data.

```
# Pearson correlation
iris.corr()
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
Sepal_Length	1.000000	-0.103784	0.871283	0.816971	0.781219
Sepal_Width	-0.103784	1.000000	-0.415218	-0.350733	-0.414532
Petal_Length	0.871283	-0.415218	1.000000	0.962314	0.948519
Petal_Width	0.816971	-0.350733	0.962314	1.000000	0.956014
Class	0.781219	-0.414532	0.948519	0.956014	1.000000

```
iris2 = iris.rank()
# Kendall Tau correlation
iris2.corr('kendall')
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
Sepal_Length	1.000000	-0.067636	0.718290	0.654197	0.669163
Sepal_Width	-0.067636	1.000000	-0.175665	-0.140207	-0.327228

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
Petal_Length	0.718290	-0.175665	1.000000	0.803041	0.822578
Petal_Width	0.654197	-0.140207	0.803041	1.000000	0.837934
Class	0.669163	-0.327228	0.822578	0.837934	1.000000

```
# Spearman Rank correlation
iris2.corr('spearman')
```

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
Sepal_Length	1.000000	-0.152136	0.881759	0.833586	0.796546
Sepal_Width	-0.152136	1.000000	-0.294020	-0.267686	-0.426319
Petal_Length	0.881759	-0.294020	1.000000	0.936188	0.935220
Petal_Width	0.833586	-0.267686	0.936188	1.000000	0.937409
Class	0.796546	-0.426319	0.935220	0.937409	1.000000