Titanic

March 5, 2019

In [1]:
```
from numpy import *
from sklearn import model_selection
import csv as csv
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
import re
```

In [2]:
```
train = pd.read_csv('train.csv')
```

In [3]:
```
test = pd.read_csv('test.csv')
```

In [4]:
```
train.head()
```

Out[4]:

	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 Th...	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
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [5]:
```
train.describe()
```

```
Out[5]:        PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

```
In [6]: train.describe(include=['O'])
```

```
Out[6]:                     Name   Sex    Ticket        Cabin Embarked
count                    891   891       891          204      889
unique                   891     2       681          147        3
top     Calic, Mr. Petar  male  CA. 2343  C23 C25 C27        S
freq                       1   577         7            4      644
```

```
In [7]: train.isna().sum()
```

```
Out[7]: PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
In [8]: test.isna().sum()
```

```
Out[8]: PassengerId      0
Pclass           0
Name             0
```

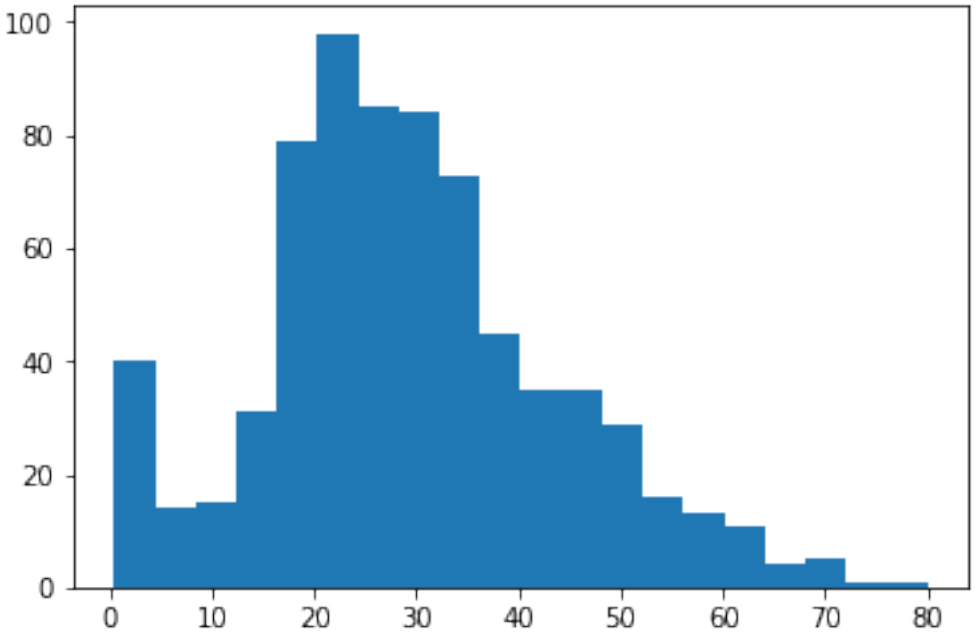
```
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

```
In [9]: train.dtypes
```

```
Out[9]: PassengerId      int64
        Survived         int64
        Pclass           int64
        Name            object
        Sex             object
        Age            float64
        SibSp            int64
        Parch            int64
        Ticket          object
        Fare           float64
        Cabin           object
        Embarked        object
        dtype: object
```

```
In [10]: plt.hist(train['Age'].dropna(), bins=20 )
```

```
Out[10]: (array([40., 14., 15., 31., 79., 98., 85., 84., 73., 45., 35., 35., 29.,
                 16., 13., 11.,  4.,  5.,  1.,  1.]),
          array([ 0.42 ,  4.399,  8.378, 12.357, 16.336, 20.315, 24.294, 28.273,
                 32.252, 36.231, 40.21 , 44.189, 48.168, 52.147, 56.126, 60.105,
                 64.084, 68.063, 72.042, 76.021, 80.   ]),
          <a list of 20 Patch objects>)
```

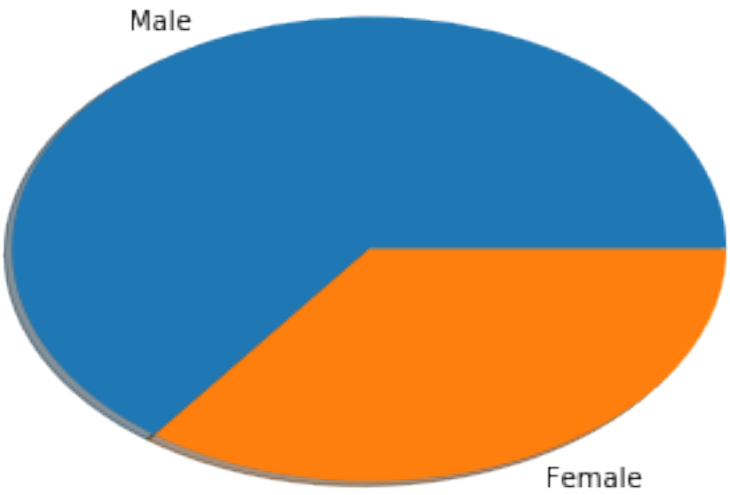



```
In [11]: train['Sex'].value_counts()

Out[11]: male      577
         female    314
         Name: Sex, dtype: int64

In [12]: plt.pie(
             x=train['Sex'].value_counts(),
             shadow=True,
             labels=['Male', 'Female']
         )

Out[12]: ([<matplotlib.patches.Wedge at 0x10d659da0>,
           <matplotlib.patches.Wedge at 0x10d664518>],
          [Text(-0.491945,0.983865,'Male'), Text(0.491946,-0.983865,'Female')])
```




```
In [13]: train['Cabin'].isna().value_counts()
```

```
Out[13]: True     687
         False    204
         Name: Cabin, dtype: int64
```

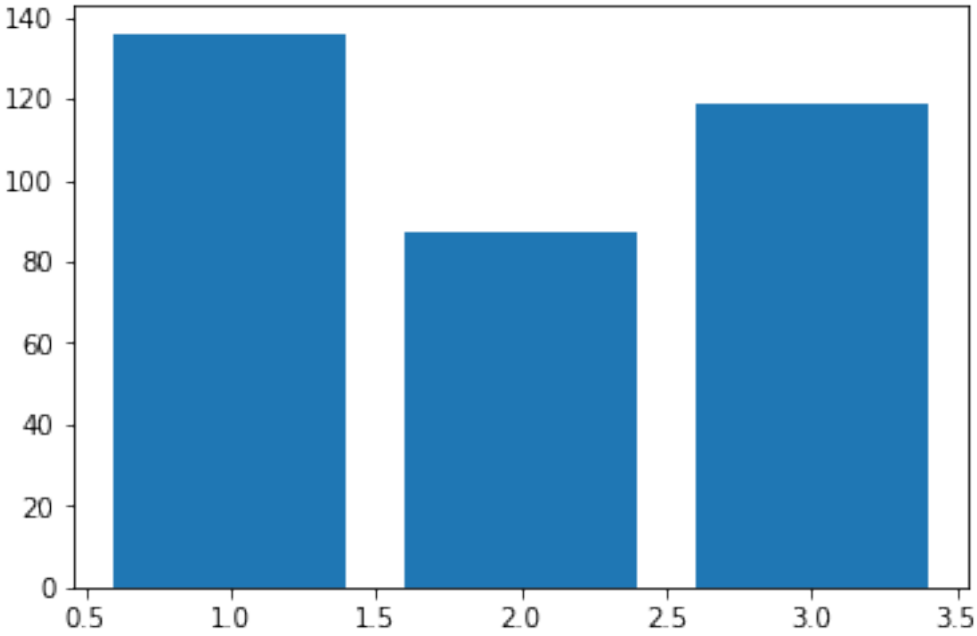
```
In [14]: train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).sum()
         classdf = train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).sum()
         classdf['Dead'] = train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).co

         classdf
```

```
Out[14]:    Pclass  Survived  Dead
         0       1       136    80
         1       2        87    97
         2       3       119   372
```

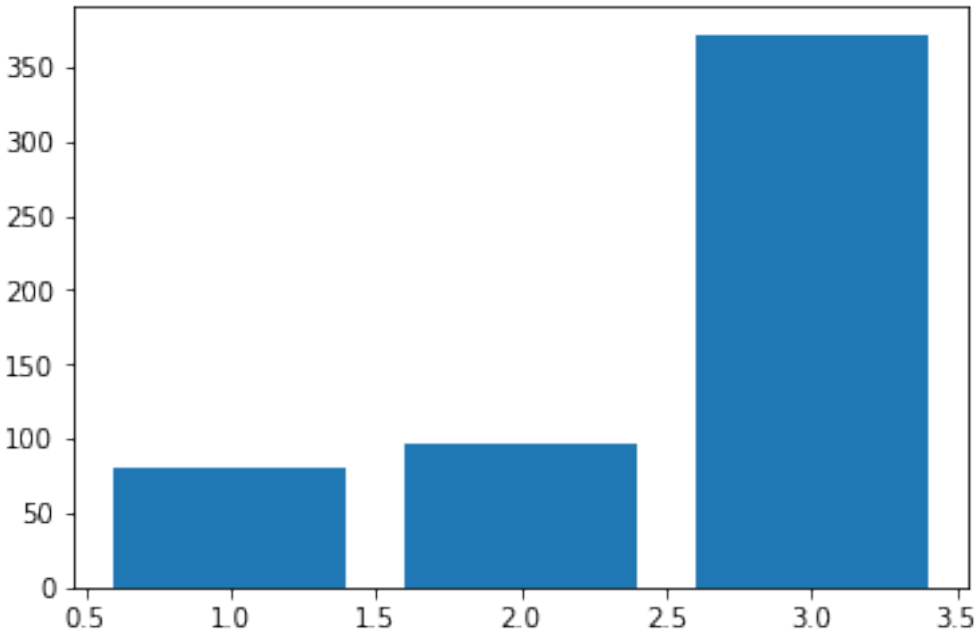
```
In [15]: plt.bar(classdf['Pclass'], classdf['Survived'])
```

```
Out[15]: <BarContainer object of 3 artists>
```




```
In [16]: plt.bar(classdf['Pclass'], classdf['Dead'])
```

```
Out[16]: <BarContainer object of 3 artists>
```



In [17]:
```
def get_title(name):
    title_search = re.search(' ([A-Za-z]+)\.', name)
    # If the title exists, extract and return it.
    if title_search:
        return title_search.group(1)
    return ""

# Create a new feature Title, containing the titles of passenger names
train['Title'] = train['Name'].apply(get_title)
test['Title'] = test['Name'].apply(get_title)
# Group all non-common titles into one single grouping "Rare"

train['Title'] = train['Title'].replace(['Lady', 'Countess','Capt', 'Col','Don', 'Dr'

train['Title'] = train['Title'].replace('Mlle', 'Miss')
train['Title'] = train['Title'].replace('Ms', 'Miss')
train['Title'] = train['Title'].replace('Mme', 'Mrs')


test['Title'] = test['Title'].replace(['Lady', 'Countess','Capt', 'Col','Don', 'Dr',

test['Title'] = test['Title'].replace('Mlle', 'Miss')
test['Title'] = test['Title'].replace('Ms', 'Miss')
test['Title'] = test['Title'].replace('Mme', 'Mrs')
```

In [18]:
```
# get average, std, and number of NaN values in titanic_df
average_age_titanic   = train["Age"].mean()
std_age_titanic       = train["Age"].std()
count_nan_age_titanic = train["Age"].isnull().sum()

# get average, std, and number of NaN values in test_df
average_age_test   = test["Age"].mean()
std_age_test       = test["Age"].std()
count_nan_age_test = test["Age"].isnull().sum()

rand_1 = np.random.randint(average_age_titanic - std_age_titanic, average_age_titanic
rand_2 = np.random.randint(average_age_test - std_age_test, average_age_test + std_ag

# fill NaN values in Age column with random values generated
train["Age"][(train["Age"]).isna()] = rand_1
test["Age"][(test["Age"]).isna()] = rand_2
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  from ipykernel import kernelapp as app
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:16: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  app.launch_new_instance()
```

```
In [19]: title_mapping = {"Miss": 1, "Mrs": 2, "SMiss": 3, "SMrs": 4 , "Mr": 5 , "SMr": 6}
         train['Title'] = train['Title'].map(title_mapping)
         train['Title'] = train['Title'].fillna(0)
```

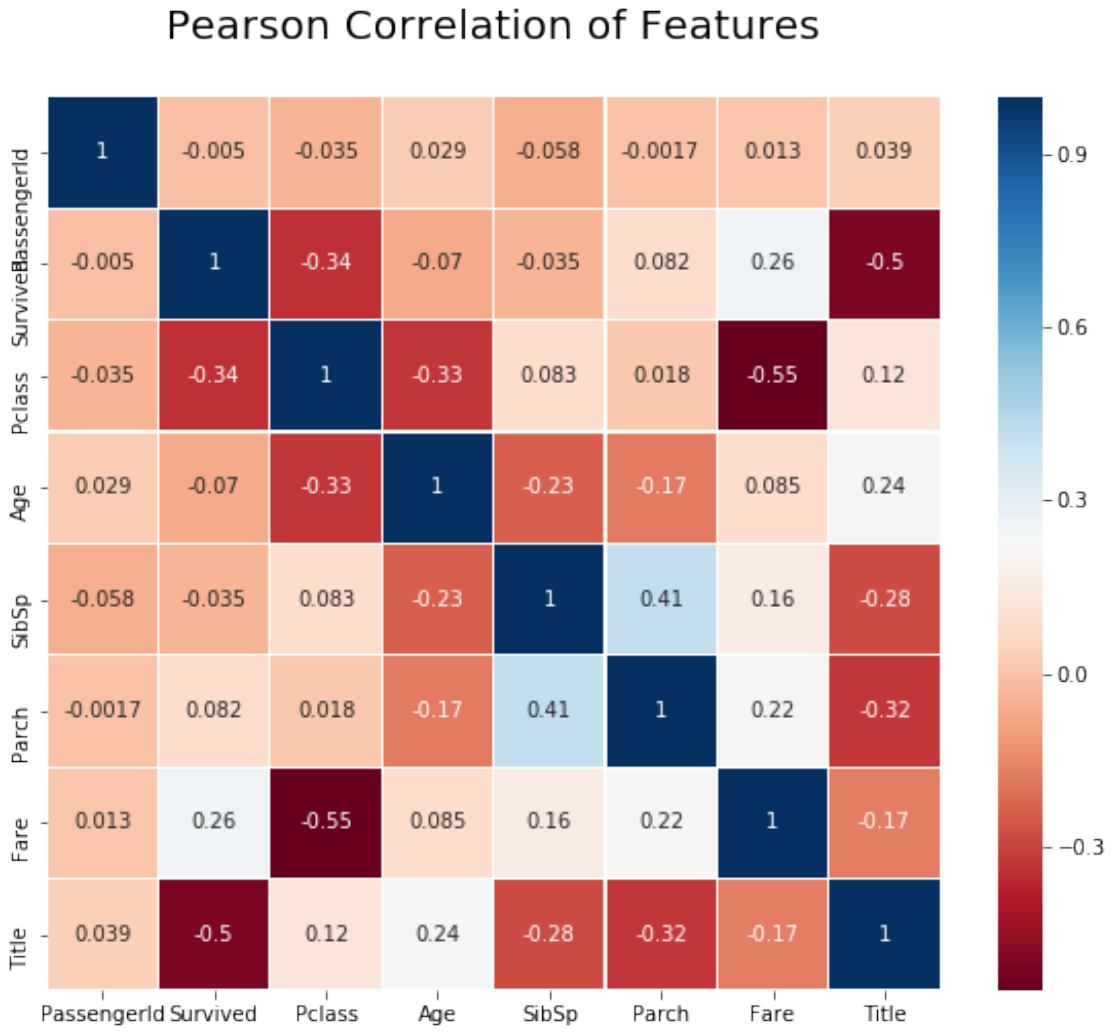
```
In [20]: test['Title'] = test['Title'].map(title_mapping)
         test['Title'] = test['Title'].fillna(0)
```

```
In [21]: numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']

         newdf = train.select_dtypes(include=numerics)
```

```
In [22]: colormap = plt.cm.RdBu
         plt.figure(figsize=(10,8))
         plt.title('Pearson Correlation of Features', y=1.05, size=20)
         sns.heatmap(newdf.astype(float).corr(),linewidths=0.1,vmax=1.0,
                     square=True, cmap=colormap, linecolor='white', annot=True)
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x10c4c0a20>
```




```
In [23]: test.isna().sum()

Out[23]: PassengerId      0
         Pclass           0
         Name             0
         Sex              0
         Age              0
         SibSp            0
         Parch            0
         Ticket           0
         Fare             1
         Cabin          327
         Embarked         0
         Title            0
         dtype: int64
```

```
In [24]: train["Cabin_y_n"] = np.where(train['Cabin'].isna(), 0, 1)
         test["Cabin_y_n"] = np.where(test['Cabin'].isna(), 0, 1)

In [25]: train.head()

Out[25]:    PassengerId  Survived  Pclass  \
         0            1         0       3
         1            2         1       1
         2            3         1       3
         3            4         1       1
         4            5         0       3

                                                        Name     Sex   Age  SibSp  \
         0                            Braund, Mr. Owen Harris    male  22.0      1
         1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
         2                             Heikkinen, Miss. Laina  female  26.0      0
         3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
         4                           Allen, Mr. William Henry    male  35.0      0

            Parch            Ticket     Fare Cabin Embarked  Title  Cabin_y_n
         0      0         A/5 21171   7.2500   NaN        S    5.0          0
         1      0          PC 17599  71.2833   C85        C    2.0          1
         2      0  STON/O2. 3101282   7.9250   NaN        S    1.0          0
         3      0            113803  53.1000  C123        S    2.0          1
         4      0            373450   8.0500   NaN        S    5.0          0

In [26]: train["sex_0_1"] = np.where(train['Sex'] == "male", 0, 1)
         test["sex_0_1"] = np.where(test['Sex'] == "male", 0, 1)

In [27]: train['Embarked'].isna().value_counts()

Out[27]: False    889
         True       2
         Name: Embarked, dtype: int64

In [28]: train['Embarked'] = np.where(train['Embarked'].isna(), "S", train['Embarked'])
         test['Embarked'] = np.where(test['Embarked'].isna(), "S", test['Embarked'])

In [29]: embark_map = { "S": 1, "C": 2, "Q": 3}
         train["Embarked"] = train["Embarked"].map(embark_map)
         test["Embarked"] = test["Embarked"].map(embark_map)

In [30]: test.head()

Out[30]:    PassengerId  Pclass                                          Name     Sex  \
         0          892       3                              Kelly, Mr. James    male
         1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
         2          894       2                     Myles, Mr. Thomas Francis    male
         3          895       3                              Wirz, Mr. Albert    male
```

```
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

    Age  SibSp  Parch   Ticket     Fare Cabin  Embarked  Title  Cabin_y_n  \
0  34.5      0      0   330911   7.8292   NaN         3    5.0          0
1  47.0      1      0   363272   7.0000   NaN         1    2.0          0
2  62.0      0      0   240276   9.6875   NaN         3    5.0          0
3  27.0      0      0   315154   8.6625   NaN         1    5.0          0
4  22.0      1      1  3101298  12.2875   NaN         1    2.0          0

   sex_0_1
0        0
1        1
2        0
3        0
4        1
```

```
In [31]: test["Fare"] = np.where(test['Fare'].isna(), test['Fare'].mean(), test['Fare'])
```

0.1 Random Forest

```
In [32]: rf = RandomForestClassifier()

In [33]: train_x = train[["Pclass", "Age", "SibSp", "Parch", "Fare", "Embarked", "Title", "Cabi

In [34]: train_y = train["Survived"]

In [35]: test_x = test[["Pclass", "Age", "SibSp", "Parch", "Fare", "Embarked", "Title", "Cabin_

In [44]: test_y = test["Survived"]

In [36]: rf.fit(train_x, train_y)

Out[36]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', 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, n_estimators=10, n_jobs=1,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)

In [37]: survival = rf.predict(test_x)

In [38]: rf.score(train_x, train_y)

Out[38]: 0.9719416386083053

In [39]: test["Survived"] = survival

In [40]: test[["PassengerId", "Survived"]].to_csv('output_1.csv', index = False)
```

```
In [42]: from sklearn.model_selection import cross_val_score
         # Use cross_val_score function
         # We are passing the entirety of X and y, not X_train or y_train, it takes care of sp
         # cv=10 for 10 folds
         # scoring='accuracy' for evaluation metric - althought they are many
         scores_rf = cross_val_score(rf, train_x, train_y, cv=10)
         print (scores_rf)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_rf.mean(), scores_rf.std() * 2))
```

```
[0.81111111 0.8        0.75280899 0.82022472 0.82022472 0.86516854
 0.78651685 0.76404494 0.87640449 0.80681818]
Accuracy: 0.81 (+/- 0.07)
```

```
In [48]: scores_rf_t = cross_val_score(rf, test_x, test_y, cv=10)
         print (scores_rf_t)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_rf_t.mean(), scores_rf_t.std() * 2))
```

```
[0.88372093 0.97619048 0.9047619  0.85714286 0.88095238 0.88095238
 0.9047619  0.92682927 0.90243902 0.85365854]
Accuracy: 0.90 (+/- 0.07)
```

```
In [54]: from sklearn.linear_model import LogisticRegression
         from sklearn.svm import SVC, LinearSVC
         from sklearn.neighbors import KNeighborsClassifier

In [55]: from sklearn.naive_bayes import GaussianNB
```

0.2 Logistic Regression

```
In [56]: lr = LogisticRegression()

In [57]: lr.fit(train_x, train_y)

Out[57]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [58]: y_lr = lr.predict(test_x)

In [59]: lr.score(train_x, train_y)

Out[59]: 0.8159371492704826

In [60]: scores_lr = cross_val_score(LogisticRegression(), train_x, train_y, cv=5)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_lr.mean(), scores_lr.std() * 2))
         # with 5 fold we recognize a better mean accuracy
```

```
Accuracy: 0.81 (+/- 0.04)
```

```
In [61]: test["Survived"] = y_lr
         test[["PassengerId", "Survived"]].to_csv('output_lr.csv', index = False)
```

```
In [62]: scores = cross_val_score(rf, train_x, train_y, cv=10)
         print(scores)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.76666667 0.78888889 0.73033708 0.82022472 0.82022472 0.84269663
 0.7752809  0.79775281 0.85393258 0.81818182]
Accuracy: 0.80 (+/- 0.07)
```

```
In [63]: scores_lr_t = cross_val_score(lr, test_x, test_y, cv=10)
         print (scores_lr_t)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_lr_t.mean(), scores_lr_t.std() * 2))
```

```
[0.97674419 0.95348837 0.95348837 0.95238095 0.95238095 1.
 0.97560976 0.97560976 1.         0.97560976]
Accuracy: 0.97 (+/- 0.04)
```

0.3 KNN

```
In [64]: knn = KNeighborsClassifier(n_neighbors = 3)
```

```
In [65]: knn.fit(train_x, train_y)
```

```
Out[65]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=3, p=2,
                    weights='uniform')
```

```
In [66]: y_knn = knn.predict(test_x)
```

```
In [67]: knn.score(train_x, train_y)
```

```
Out[67]: 0.8473625140291807
```

```
In [68]: test["Survived"] = y_knn
         test[["PassengerId", "Survived"]].to_csv('output_knn.csv', index = False)
```

```
In [69]: scores = cross_val_score(KNeighborsClassifier(n_neighbors = 3), train_x, train_y, cv=
         print(scores)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
[0.7        0.72222222 0.71910112 0.73033708 0.75280899 0.78651685
 0.84269663 0.75280899 0.75280899 0.77272727]
Accuracy: 0.75 (+/- 0.08)
```

```
In [87]: scores_knn_t = cross_val_score(knn, test_x, test_y, cv=10)
         print (scores_knn_t)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_lr_t.mean(), scores_lr_t.std() * 2))

[0.74418605 0.69767442 0.88372093 0.66666667 0.69047619 0.65853659
 0.68292683 0.7804878  0.75609756 0.56097561]
Accuracy: 0.71 (+/- 0.16)
```

0.4 Naives Bayes

```
In [71]: nb = GaussianNB()

In [72]: nb.fit(train_x, train_y)

Out[72]: GaussianNB(priors=None)

In [73]: y_nb = nb.predict(test_x)

In [74]: nb.score(train_x, train_y)

Out[74]: 0.7856341189674523

In [75]: test["Survived"] = y_nb
         test[["PassengerId", "Survived"]].to_csv('output_nb.csv', index = False)

In [157]: scores_bayes = cross_val_score(nb, train_x, train_y, cv=10)
          print("Accuracy: %0.2f (+/- %0.2f)" % (scores_bayes.mean(), scores_bayes.std() * 2))

Accuracy: 0.78 (+/- 0.05)


In [76]: scores_bayes_t = cross_val_score(nb, test_x, test_y, cv=10)
         print (scores_bayes_t)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_bayes_t.mean(), scores_bayes_t.std() *

[0.95238095 0.97619048 0.95238095 0.97619048 0.95238095 1.
 0.97619048 0.9047619  0.87804878 0.97560976]
Accuracy: 0.95 (+/- 0.07)
```

0.5 Decision Tree

```
In [77]: from sklearn.tree import DecisionTreeClassifier

In [78]: dtree = DecisionTreeClassifier()
         dtree.fit(train_x,train_y)
```

```
Out[78]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                     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')

In [79]: y_dtree = dtree.predict(test_x)

In [80]: dtree.score(train_x, train_y)

Out[80]: 0.9921436588103255

In [81]: test["Survived"] = y_dtree
         test[["PassengerId", "Survived"]].to_csv('output_dtree.csv', index = False)

In [82]: dtree.fit(train_x.head(600),train_y.head(600))

Out[82]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                     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')

In [83]: train_x.shape

Out[83]: (891, 9)

In [84]: y_dtree_sample = dtree.predict(train_x.tail(291))

In [85]: scores_dtree = cross_val_score(dtree, train_x, train_y, cv=10)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_dtree.mean(), scores_dtree.std() * 2))

Accuracy: 0.77 (+/- 0.06)


In [86]: scores_dtree_t = cross_val_score(dtree, test_x, test_y, cv=10)
         print (scores_bayes_t)
         print("Accuracy: %0.2f (+/- %0.2f)" % (scores_dtree_t.mean(), scores_dtree_t.std() * 2

[0.95238095 0.97619048 0.95238095 0.97619048 0.95238095 1.
 0.97619048 0.9047619  0.87804878 0.97560976]
Accuracy: 0.80 (+/- 0.11)
```