ANALYZING TITANIC DATASET

BY SUDHISH SUBRAMANIAM,

STUDENT OF VIT VELLORE

2ND YEAR B-TECH : ECE CORE

INTRODUCTION

- On 10 April 1912, the Titanic departed on its maiden voyage from Southampton to New York.
- On 14 April 1912, despite warnings of ice fields, the ship did not reduce speed and struck an iceberg shortly before midnight.
- The iceberg ripped a long gash in the side and the ship began to flood. Passengers were unaware and joked about the ice found on the deck.
- The Captain ordered the lifeboats to be filled and lowered, with women and children first.
- The dataset tells us the number of people who survived the tragedy.

OBJECTIVE

The main objective of the machine algorithm is create a model which could predict whether a person could survive the tragedy, by analyzing the parameters.

PARAMETERS TO DETERMINE

```
In [83]: train_df.columns

Out[83]: Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
               'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
              dtype='object')
```

FINDING THE OPTIMUM TARGET COLUMNS

In [14]:
```
train_df[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

Out[14]:

	Pclass	Survived
1	1	0.629630
2	2	0.472826
3	3	0.242363
0		NaN

In [15]:
```
train_df[["Sex", "Survived"]].groupby(['Sex'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

Out[15]:

	Sex	Survived
0	female	0.742038
1	male	0.188908

In [16]:
```
train_df[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

Out[16]:

	SibSp	Survived
1	1.0	0.535885
2	2.0	0.464286
0	0.0	0.345395
3	3.0	0.250000
4	4.0	0.166667
5	5.0	0.000000
6	8.0	0.000000

SPLITTING THE DATASET

In [35]:
```
df1.head()
```

Out[35]:

	Pclass	Sex	Age	Fare	Embarked
0	3	male	22.0	7.2500	S
1	1	female	38.0	71.2833	C
2	3	female	26.0	7.9250	S
3	1	female	35.0	53.1000	S
4	3	male	35.0	8.0500	S

In [16]:
```
df2.head()
```

Out[16]:

	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	0	1	0	0	1
1	1	0	1	0	0
2	1	0	0	0	1
3	1	0	0	0	1
4	0	1	0	0	1

In [17]:
```
df3 = df1.select_dtypes(exclude=['object'])     #
df3.head()
```

Out[17]:

	Age	Fare
0	22.0	7.2500
1	38.0	71.2833
2	26.0	7.9250
3	35.0	53.1000
4	35.0	8.0500

In [19]:
```
final_data.head()
```

Out[19]:

	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	Age	Fare
0	0	1	0	0	1	22.0	7.2500
1	1	0	1	0	0	38.0	71.2833
2	1	0	0	0	1	26.0	7.9250
3	1	0	0	0	1	35.0	53.1000
4	0	1	0	0	1	35.0	8.0500

RELATION BETWEEN COLUMNS: I

```
In [85]: train_df[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

Out[85]:

	Pclass	Survived
1	1	0.629630
2	2	0.472826
3	3	0.242363
0		NaN

```
In [86]: train_df[["Sex", "Survived"]].groupby(['Sex'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

Out[86]:

	Sex	Survived
0	female	0.742038
1	male	0.188908

RELATION BETWEEN COLUMNS: II

In [87]:
```
train_df[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

Out[87]:

	SibSp	Survived
1	1.0	0.535885
2	2.0	0.464286
0	0.0	0.345395
3	3.0	0.250000
4	4.0	0.166667
5	5.0	0.000000
6	8.0	0.000000

In [88]:
```
train_df[["Parch", "Survived"]].groupby(['Parch'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

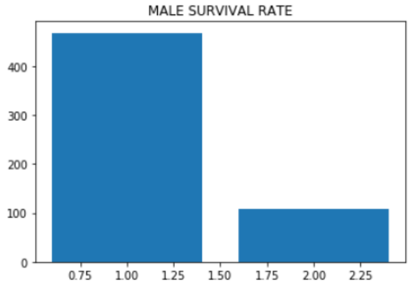
Out[88]:

	Parch	Survived
3	3.0	0.600000
1	1.0	0.550847
2	2.0	0.500000
0	0.0	0.343658
5	5.0	0.200000
4	4.0	0.000000
6	6.0	0.000000

FINDING THE OPTIMUM COLUMN

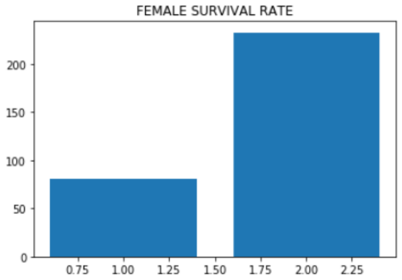
```
In [123]: a = [train_df[train_df.Sex=='male'].Survived.value_counts()[0],train_df[train_df.Sex=='male'].Survived.value_counts()[1]]
          b = [1,2]
          plt.bar(b,a)
          plt.title('MALE SURVIVAL RATE')

Out[123]: Text(0.5, 1.0, 'MALE SURVIVAL RATE')
```




```
In [124]: a = [train_df[train_df.Sex=='female'].Survived.value_counts()[0],train_df[train_df.Sex=='female'].Survived.value_counts()[1]]
          b = [1,2]
          plt.bar(b,a)
          plt.title('FEMALE SURVIVAL RATE')

Out[124]: Text(0.5, 1.0, 'FEMALE SURVIVAL RATE')
```



EXTRACTING THE REQUIRED COLUMNS

In [89]:
```
X = pd.get_dummies(train_df['Sex']).values
print(X)
```

```
[[0 1]
 [1 0]
 [1 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

In [90]:
```
y = pd.get_dummies(train_df['Survived']).values
print(y)
```

```
[[1 0]
 [0 1]
 [0 1]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

DATASET

In [8]:
```
df.head()
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1.0	0.0	3	Braund, Mr. Owen Harris	male	22.0	1.0	0.0	A/5 21171	7.2500	NaN	S
1	2.0	1.0	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1.0	0.0	PC 17599	71.2833	C85	C
2	3.0	1.0	3	Heikkinen, Miss. Laina	female	26.0	0.0	0.0	STON/O2. 3101282	7.9250	NaN	S
3	4.0	1.0	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1.0	0.0	113803	53.1000	C123	S
4	5.0	0.0	3	Allen, Mr. William Henry	male	35.0	0.0	0.0	373450	8.0500	NaN	S

LIBRARIES IMPORTED

```
In [6]: import numpy as np
        import pandas as pd
        from sklearn.svm import SVC

In [7]: df = pd.read_csv('train.csv')
```

```
In [80]: import pandas as pd
         import numpy as np


         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.linear_model import LogisticRegression
         from sklearn.svm import SVC, LinearSVC
```

LOGISTIC REGRESSION

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist.

In regression analysis, logistic regression is estimating the parameters of a logistic model (a form of binary regression).

LOGISTIC REGRESSION OUTPUT ON OUR DATASET

```
In [24]: from sklearn.linear_model import LogisticRegression
         regressor = LogisticRegression()
         regressor.fit(X_train.reshape(-1,1),y_train.ravel())

Out[24]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, l1_ratio=None, max_iter=100,
                            multi_class='auto', n_jobs=None, penalty='l2',
                            random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                            warm_start=False)

In [25]: y_pred = regressor.predict(X_test.reshape(-1,1))
         print(y_pred.reshape(-1,1)[:5])
         print(y_test.reshape(-1,1)[:5])

         [[1]
          [0]
          [1]
          [0]
          [0]]
         [[1]
          [0]
          [1]
          [0]
          [1]]

In [26]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test.reshape(-1,1),y_pred.reshape(-1,1))
         print(cm)

         [[132  48]
          [ 48 132]]

In [27]: regressor.score(X_test.reshape(-1,1),y_test.reshape(-1,1))

Out[27]: 0.7333333333333333
```

SVM AND SVC

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outlier detection.

SVC ON OUR DATASET

```
In [92]: from sklearn.model_selection import train_test_split
         X_train ,X_test, Y_train,Y_test = train_test_split(X,y, test_size=0.2)

In [93]: svc = SVC()

In [94]: svc.fit(X_train.reshape(-1,1), Y_train.ravel())

Out[94]: SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)

In [95]: Y_pred = svc.predict(X_test.reshape(-1,1))

In [96]: acc_svc = round(svc.score(X_test.reshape(-1,1), Y_test.ravel()) * 100, 2)

In [97]: print(acc_svc)

         80.0
```

KNN ALGORITHM

KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output
2. Calculation time
3. Predictive Power

KNN ON OUR DATASET

In [99]:
```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
acc_knn
```

Out[99]: 78.49

CONVERTING THE DATASET INTO TRAIN AND TEST

```
from sklearn.model_selection import train_test_split
X_train ,X_test, y_train,y_test = train_test_split(X,y, test_size=0.2)
print(X_train)
```

RESULTS OF THE PREDICTION

```
In [26]: from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test.reshape(-1,1),y_pred.reshape(-1,1))
         print(cm)

[[132  48]
 [ 48 132]]
```

Confusion matrix

```
In [25]: y_pred = regressor.predict(X_test.reshape(-1,1))
         print(y_pred.reshape(-1,1)[:5])
         print(y_test.reshape(-1,1)[:5])

[[1]
 [0]
 [1]
 [0]
 [0]]
[[1]
 [0]
 [1]
 [0]
 [1]]
```

y_test and y_pred comparision

COMPARING THE RESULTS

ALGORITHM USED	PREDICTION SCORE
LOGISTIC REGRESSION	73.333
SVC	**80.0**
KNN	78.49

CONCLUSION

- Variables such as “Passenger Id”, ”Name”, “Ticket”, “Fare”, “Cabin” are removed as they are not effecting the target variable much.
- Women, children and first class passengers as well as people with a small family had a better chance of survival.
- I am getting a maximum of accuracy of **80%** by using the SVC algorithm.