TITANIC:DATA ANALYSIS AND PREDICTION | FINAL PROJECT ICT ACADEMY

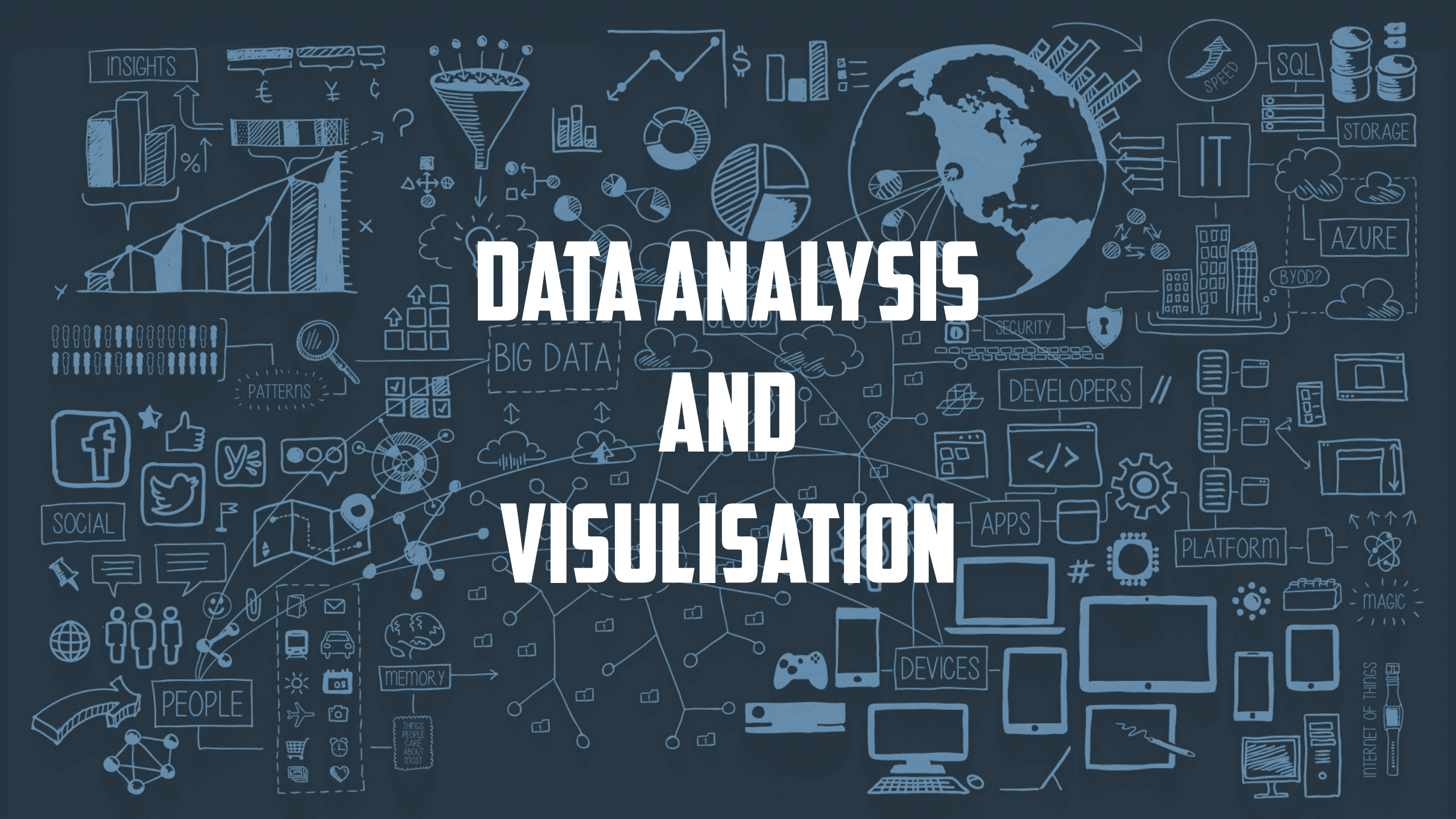
SUBMITTED BY: SHREYASH SHARMA
amans9985@gmail.com

INTRODUCTION

- RMS Titanic was a British passenger liner that started its journey with 2200 passengers and four days later sank in the North Atlantic Ocean in the early morning of 15th April 1912. Around 1500 people died and 700 survived the tragedy • According to Encyclopedia Titanica, of the 712 survivors 500 were passengers (369-women & children ,131-men) and 212 were crew (20- women, 192-men)
- The sinking of the RMS Titanic is one of the most infamous shipwrecks in history.
- In this challenge, we ask you to complete the analysis of what sorts of people were likely to survive.
- In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

GOAL AND OBJECTIVE

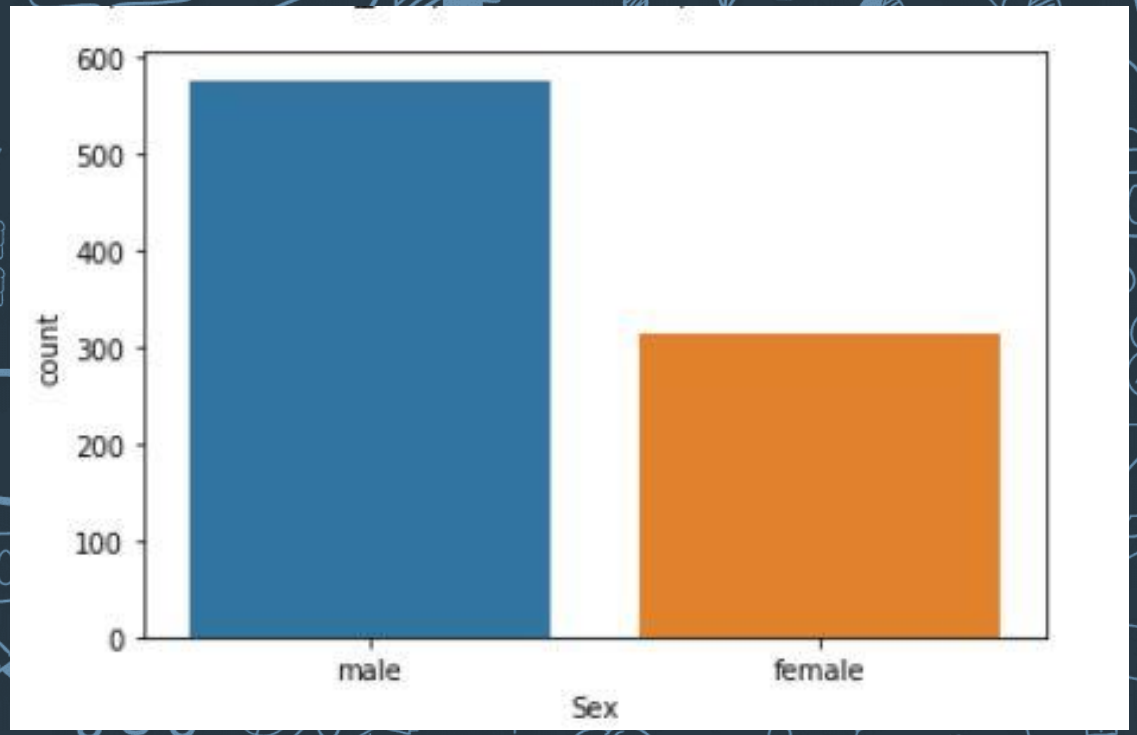
- The objective of this project was to build models that could successfully determine whether a Titanic passenger lived or died and to compare which model is able to predict the closest to the original data.



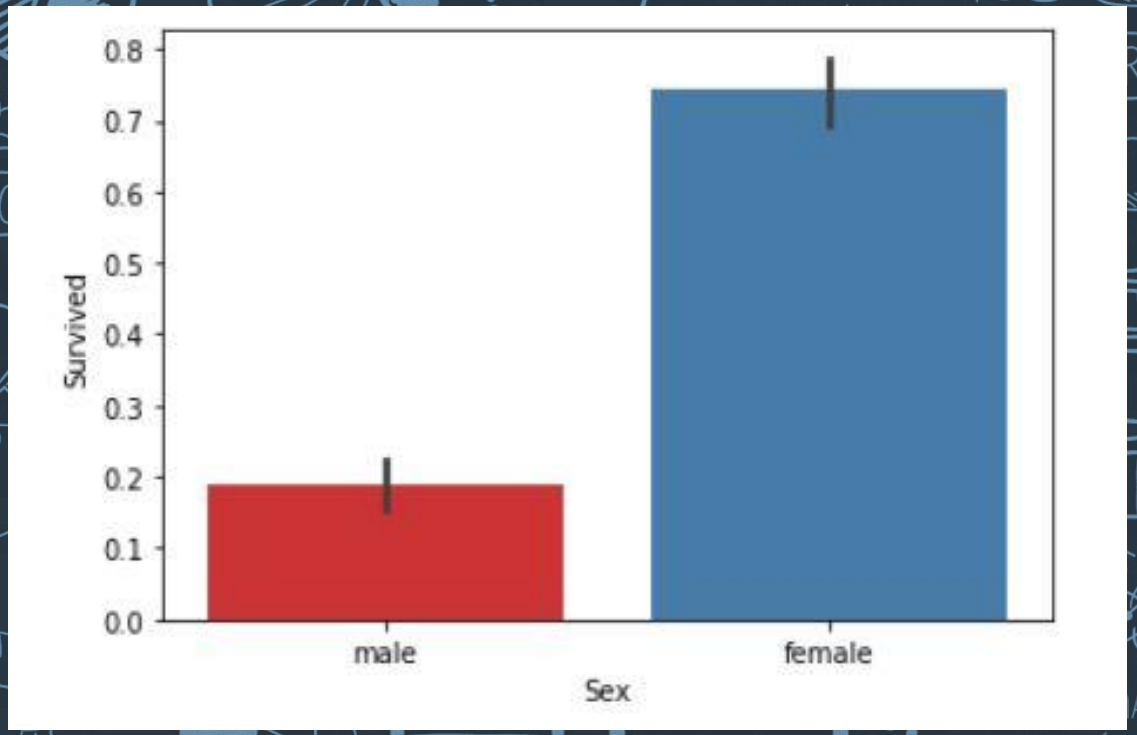
DATA ANALYSIS AND VISULISATION

	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
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

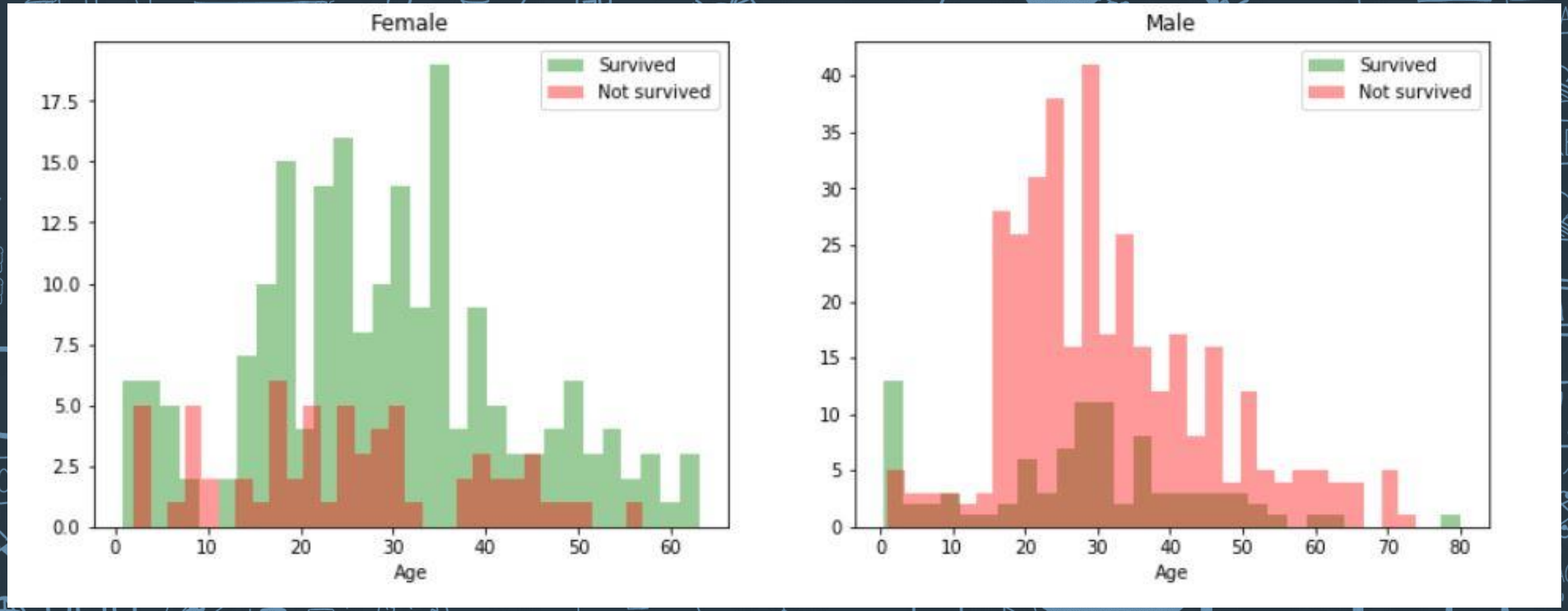
ORIGINAL DATA



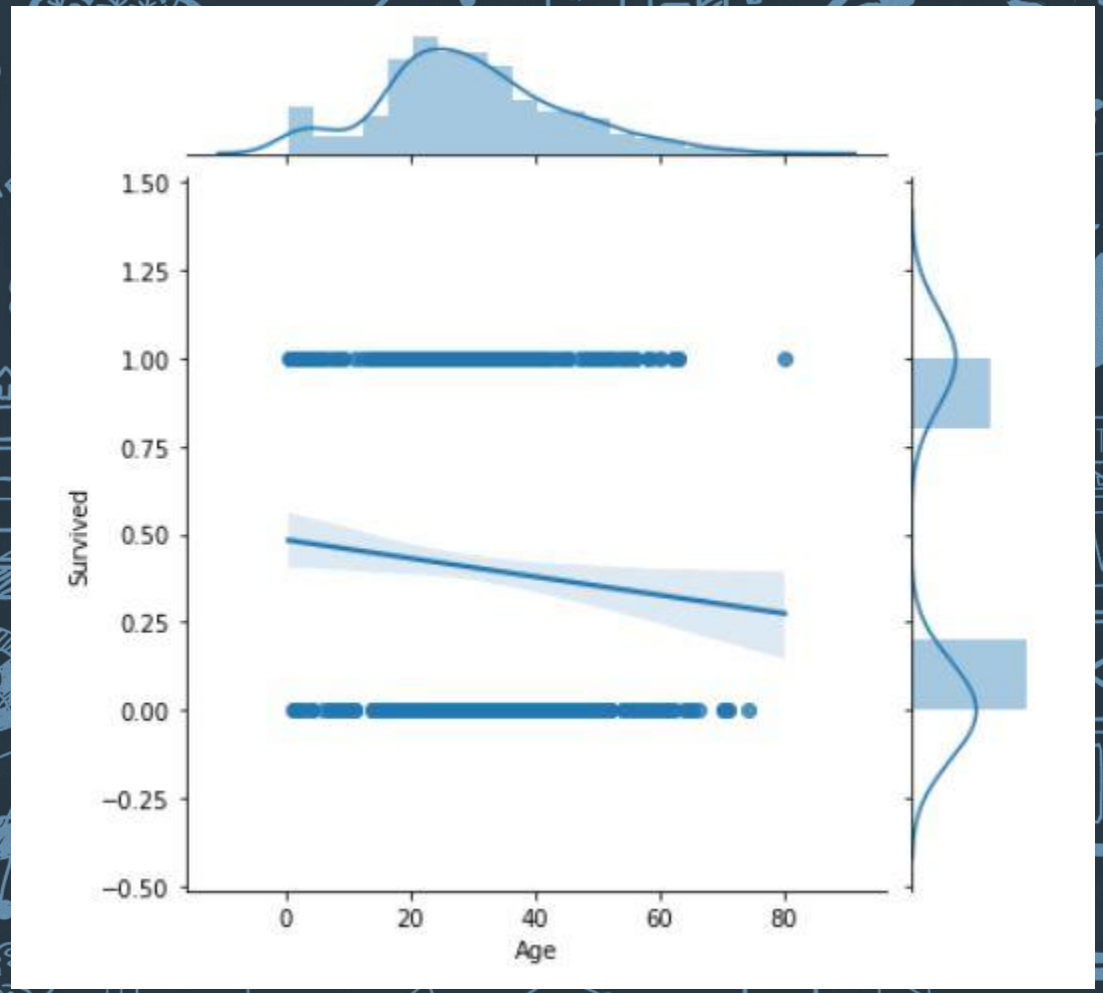
SEX FEATURE



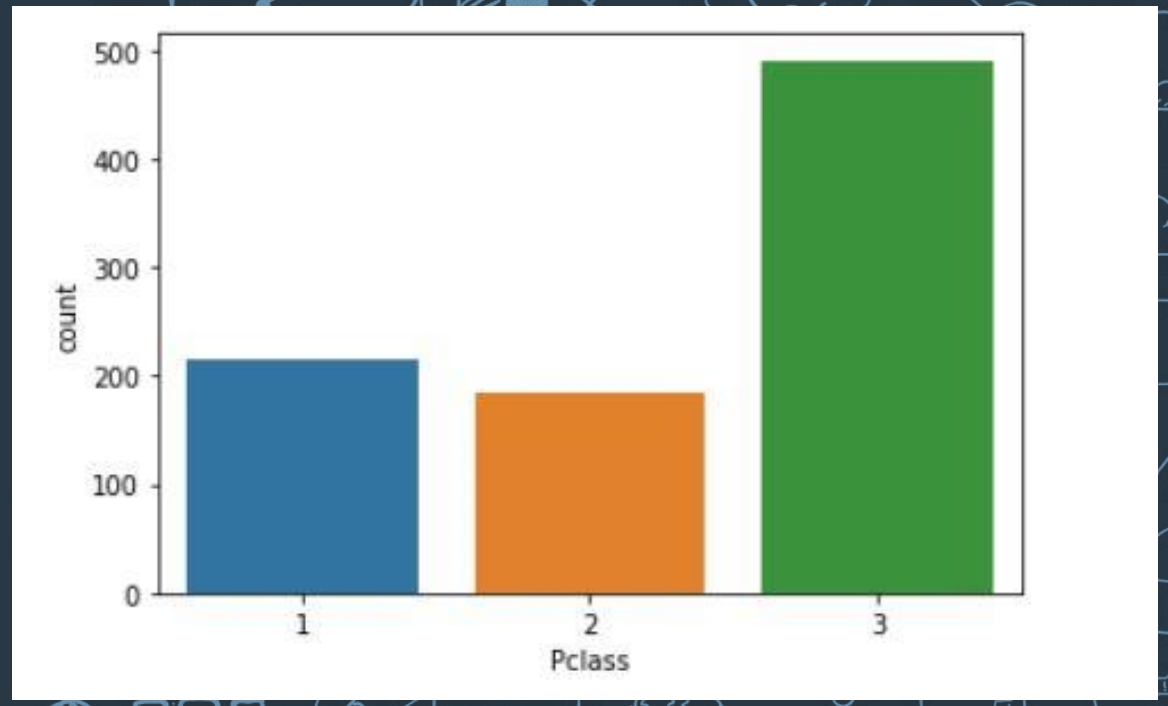
SURVIVED VS SEX


Female
Survived
Not survived
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
10
20
30
40
50
60
Age
Male
Survived
Not survived
40
35
30
25
20
15
10
5
0
0
10
20
30
40
50
60
70
80
Age

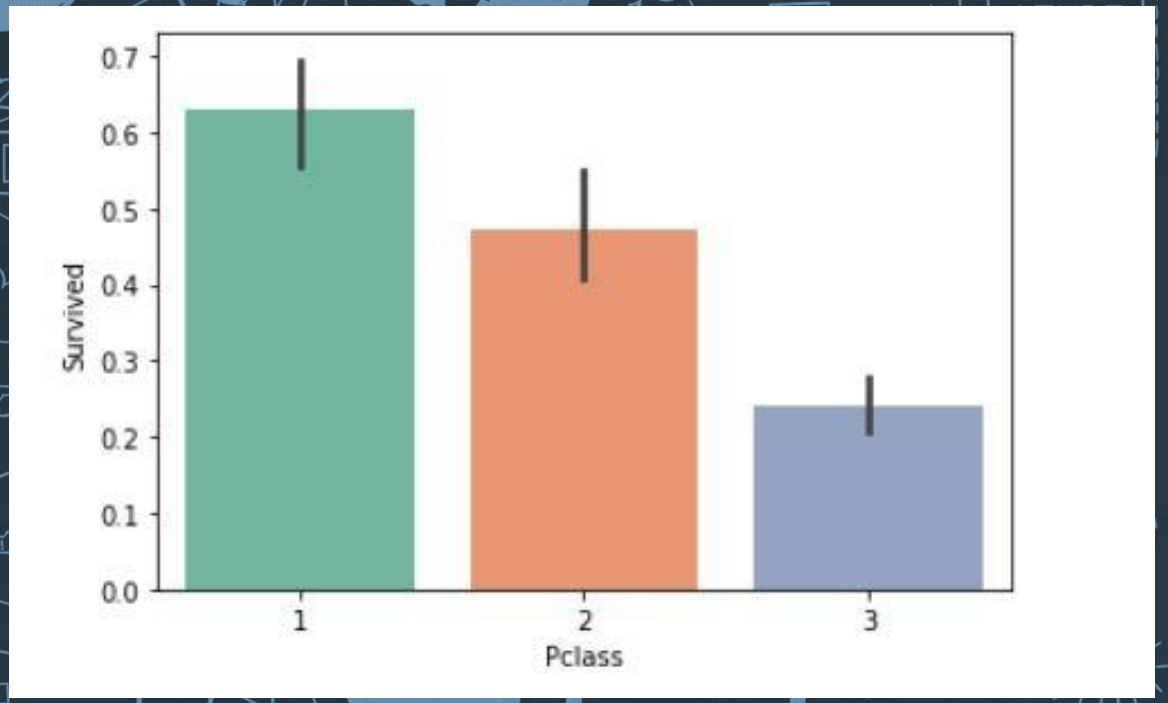
SEX VS AGE



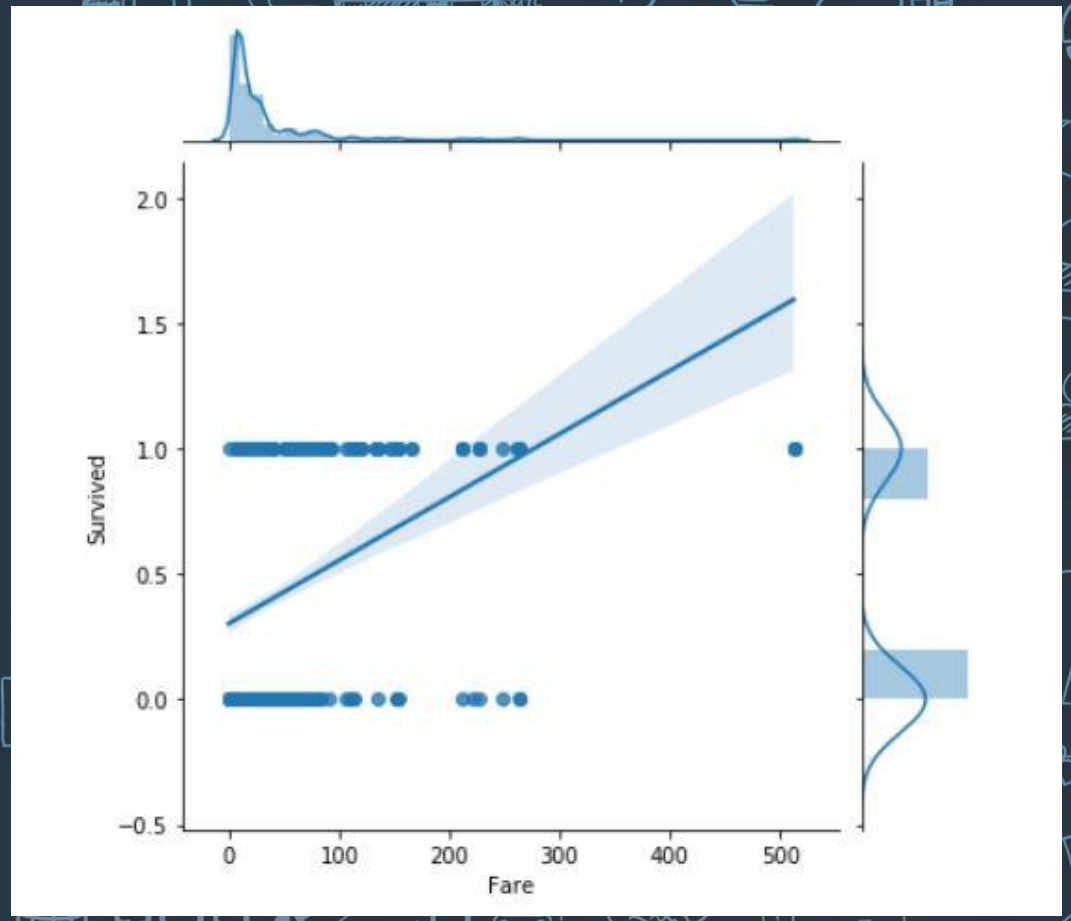
SURVIVED BY AGE (JOINT PLOT)



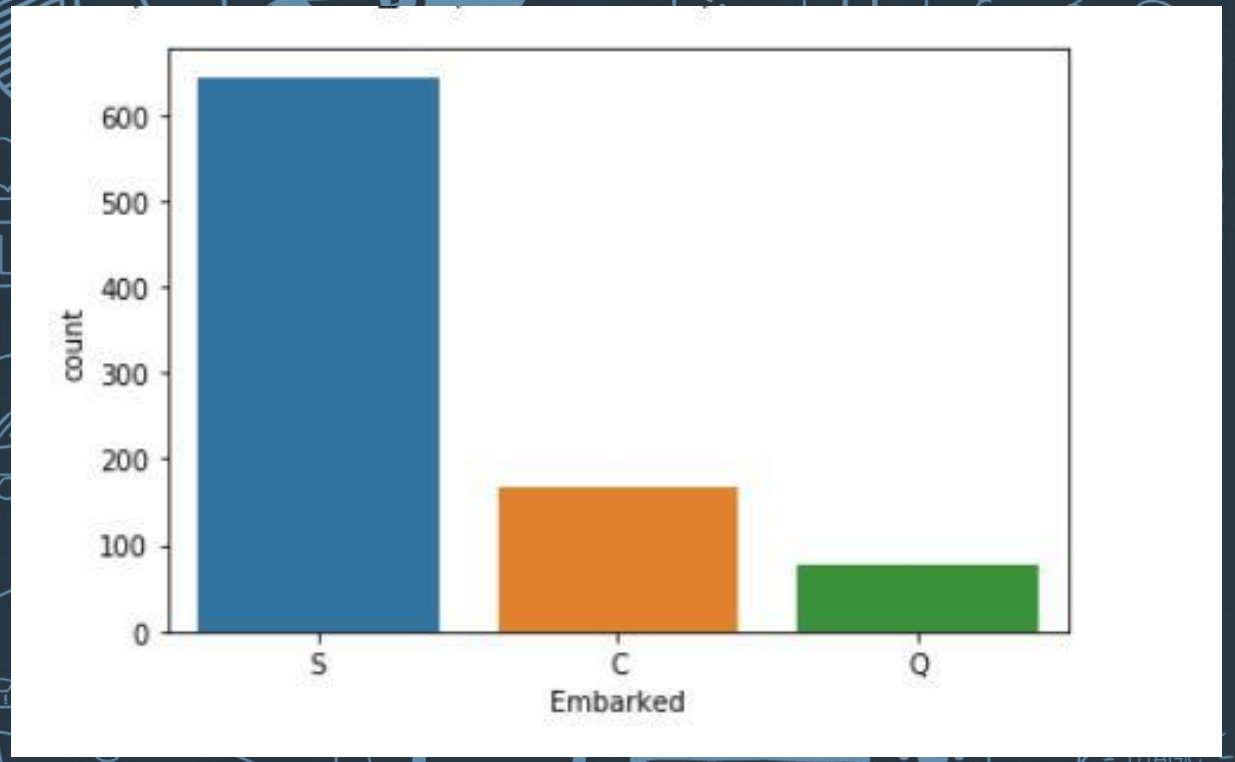
PCLASS FEATURE



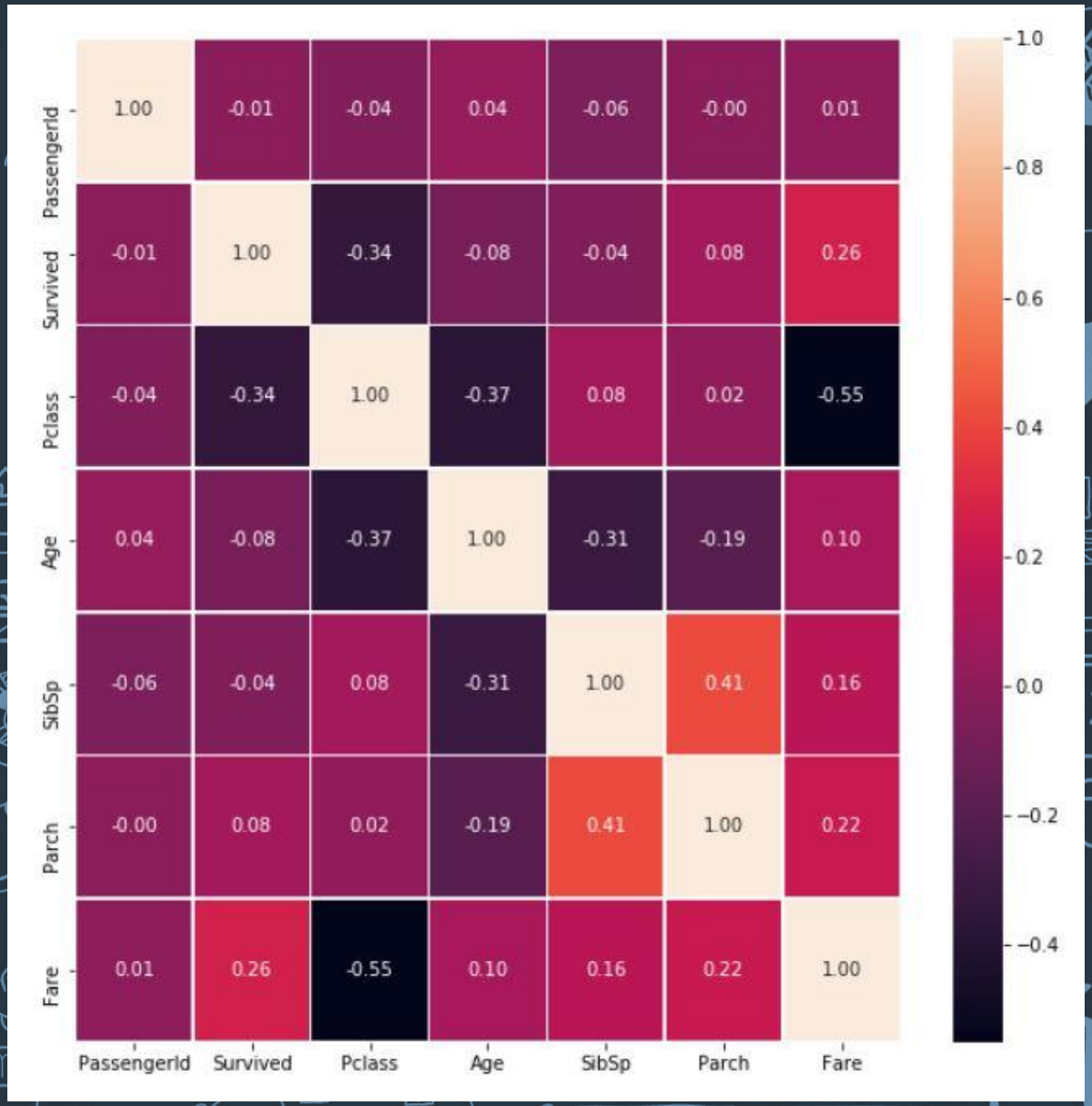
SURVIVED VS PCLASS



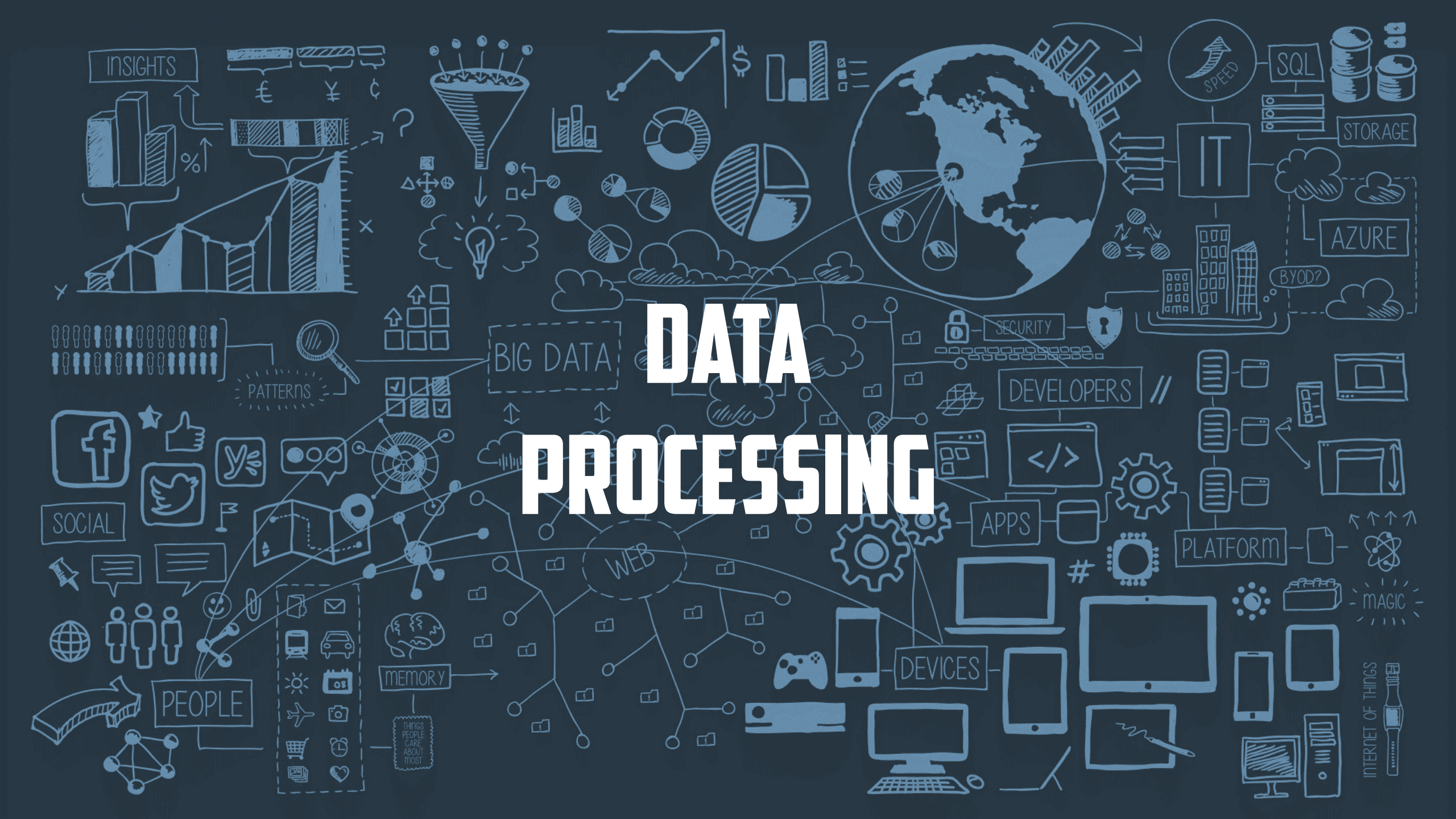
SURVIVED VS FARE



EMBARKED FEATURE



CORREALATION HEATMAP OF FEATURES



DATA PROCESSING

```
dataset1 = dataset[['Pclass','Sex','Age','Fare']]
```

```
dataset1.head()
```

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

Data for model building

```
dataset1.isnull().sum()
```

```
Pclass      0
Sex         0
Age       177
Fare        0
dtype: int64
```

```
x =pd.get_dummies(dataset1)
x.head()
```

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

Checking null values
And
Selection of
Independent feature
Using which model
Shall be made

```
y = dataset.iloc[:,1]
print(y)
```

```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

Selection of dependent Feature

ONE HOT ENCODING

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[("encoder",OneHotEncoder(),[0])],remainder="passthrough")
x= np.array(ct.fit_transform(x))
print(x)
```

```
[[ 0.      0.      1.      ...  7.25    0.      1.    ]
 [ 1.      0.      0.      ... 71.2833  1.      0.    ]
 [ 0.      0.      1.      ...  7.925   1.      0.    ]
 ...
 [ 0.      0.      1.      ... 23.45    1.      0.    ]
 [ 1.      0.      0.      ... 30.      0.      1.    ]
 [ 0.      0.      1.      ...  7.75    0.      1.    ]]
```

TAKING CARE OF NaN VALUES

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy='mean')
imputer.fit(x)
x= imputer.transform(x)
print(x)
```

```
[[ 0.      0.      1.      ...  7.25    0.      1.    ]
 [ 1.      0.      0.      ... 71.2833  1.      0.    ]
 [ 0.      0.      1.      ...  7.925   1.      0.    ]
 ...
 [ 0.      0.      1.      ... 23.45    1.      0.    ]
 [ 1.      0.      0.      ... 30.      0.      1.    ]
 [ 0.      0.      1.      ...  7.75    0.      1.    ]]
```

SCALING THE VALUES

```
from sklearn.preprocessing import MinMaxScaler
norm = MinMaxScaler()
x = norm.fit_transform(x)
print(x)
```

```
[[0.         0.         1.         ... 0.01415106 0.         1.        ]
 [1.         0.         0.         ... 0.13913574 1.         0.        ]
 [0.         0.         1.         ... 0.01546857 1.         0.        ]
 ...
 [0.         0.         1.         ... 0.04577135 1.         0.        ]
 [1.         0.         0.         ... 0.0585561  0.         1.        ]
 [0.         0.         1.         ... 0.01512699 0.         1.        ]]
```

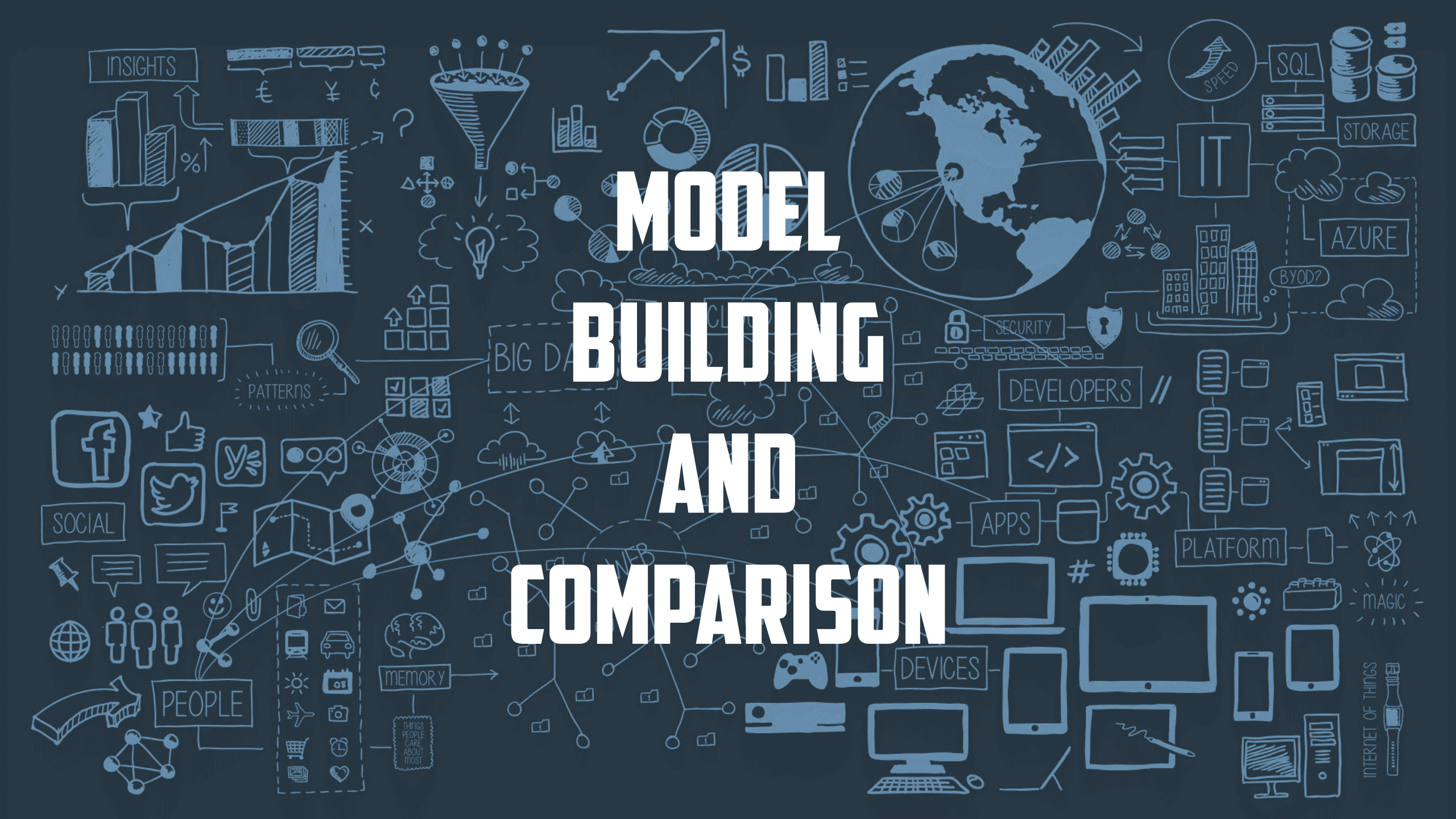
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=0)
print(x_train)
```

```
[[0.         0.         1.         ... 0.02975782 1.         0.        ]
 [0.         1.         0.         ... 0.02049464 0.         1.        ]
 [0.         1.         0.         ... 0.07222739 0.         1.        ]
 ...
 [0.         0.         1.         ... 0.0150944  0.         1.        ]
 [0.         0.         1.         ... 0.03396254 1.         0.        ]
 [0.         1.         0.         ... 0.07612293 0.         1.        ]]
```

```
print(x_test)
```

```
[[0.         0.         1.         ... 0.02822072 0.         1.        ]
 [0.         0.         1.         ... 0.01473662 0.         1.        ]
 [0.         0.         1.         ... 0.05684821 0.         1.        ]
 ...
 [1.         0.         0.         ... 0.22109808 1.         0.        ]
 [0.         0.         1.         ... 0.01533038 0.         1.        ]
 [0.         0.         1.         ... 0.01571255 0.         1.        ]]
```

SPLITTING THE DATA INTO TEST AND TRAIN SET



MODEL BUILDING AND COMPARISON

LINEAR REGRESSION

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

```
regressor.fit(x_train,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
regressor.score(x_test,y_test)
```

```
0.42080113507699 28
```

- As the score suggest it is not the ideal model for this prediction as this model tries to fit the model on a line equation.

POLYNOMIAL REGRESSION

```
from sklearn.preprocessing import PolynomialFeatures
poly_reg= PolynomialFeatures(degree=4)
x_poly= poly_reg.fit_transform(x)
regressorPoly= LinearRegression()
regressorPoly.fit(x_poly, y)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
regressorPoly.score(x_poly, y)
```

```
0.4988636649611491
```

- The result score is similar to the linear model and it suggests that regression models are unable to predict when the result must be in 0 or 1 i.e. true or false.

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
regressor1 = LogisticRegression()
regressor1.fit(x_train,y_train)
```

```
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)
```

```
regressor1.score(x_test,y_test)
```

```
0.776536312849162
```

- It is a special type of regression model which can be used for Boolean results. It has a pretty high score as compared to previous regression models which suggests that it as a good model this prediction.

DECISION TREE CLASSIFIER

```
from sklearn.tree import DecisionTreeClassifier
dtree= DecisionTreeClassifier()
dtree.fit(x_train,y_train)
y_pred=dtree.predict(x_test)
```

```
dtree.score(x_test,y_test)
```

```
0.7877094972067039
```

- It is a classification model and can predict very well as the score also suggests

RANDOM FOREST CLASSIFIER

```
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(x_train, y_train)
y_pred = random_forest.predict(x_test)
```

```
random_forest.score(x_test,y_test)
```

```
0.8324022346368715
```

- It is also a classification model and it also have the best score so far.

CONCLUSION

- With all the models we built, we calculated the prediction score for the test set and compared the score of the models and now we can conclude that RANDOM FOREST CLASSIFIER is the best model with a score of 83.24 %.
- We can also conclude that classification models are better for predicting Boolean results as we saw in this case.
- Logistic regression was an exception, it had a decent score.

THANK YOU VERY MUCH