Title - Classification Algorithms on Titanic Dataset

Group No - 23

Participants Name:

1. AlWasi Khan
2. Vikas Dutta
3. Sandipta Subir Khare

About Dataset-

The RMS Titanic was a British passenger liner that sank in the North Atlantic Ocean in the early morning hours of 15 April 1912, after it collided with an iceberg during its maiden voyage from Southampton to New York City. There were an estimated 2,224 passengers and crew aboard the ship, and more than 1,500 died, making it one of the deadliest commercial peacetime maritime disasters in modern history. The RMS Titanic was the largest ship afloat at the time it entered service and was the second of three Olympic-class ocean liners operated by the White Star Line. The Titanic was built by the Harland and Wolff shipyard in Belfast. Thomas Andrews, her architect, died in the disaster.

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

Out[5]:

	pclass	survived	age	sibsp	parch	fare	body
count	1309.000000	1309.000000	1046.000000	1309.000000	1309.000000	1308.000000	121.000000
mean	2.294882	0.381971	29.881135	0.498854	0.385027	33.295479	160.809917
std	0.837836	0.486055	14.413500	1.041658	0.865560	51.758668	97.696922
min	1.000000	0.000000	0.166700	0.000000	0.000000	0.000000	1.000000
25%	2.000000	0.000000	21.000000	0.000000	0.000000	7.895800	72.000000
50%	3.000000	0.000000	28.000000	0.000000	0.000000	14.454200	155.000000
75%	3.000000	1.000000	39.000000	1.000000	0.000000	31.275000	256.000000
max	3.000000	1.000000	80.000000	8.000000	9.000000	512.329200	328.000000

Above we can see that 38% out of the training-set survived the Titanic. We can also see that the passenger ages range from 0.16 to 80. On top of that we can already detect some features, that contain missing values, like the ‘Age’ feature.

We also get information about the mean age of passengers on titanic ship, average survival among the total passengers on the ship.

The columns in the dataset can be classified in the following manner:

Categorical: Survived, Sex, and Embarked.

Ordinal: Pclass.

Continous/Numerical: Age, Fare. Discrete: SibSp, Parch.

Ticket is a mix of numeric and alphanumeric data types.

Cabin is alphanumeric.

Name feature may contain errors or typos as there are several ways used to describe a name including titles, round brackets, and quotes used for alternative or short names.

DATA TYPES:

- Seven features are integer or floating type.
- Five features are string type.

Assumptions based on data analysis

We arrive at following assumptions based on data analysis done so far. We may validate these assumptions further before taking appropriate actions.

Correlating:

We want to know how well does each feature correlate with Survival. We want to do this early in our project and match these quick correlations with modelled correlations later in the project.

Completing:

1. We may want to complete Age feature as it is definitely correlated to survival.
2. We may want to complete the Embarked feature as it may also correlate with survival or another important feature.

Classifying:

We may also add to our assumptions based on the problem description noted earlier.

1. Women (Sex=female) were more likely to have survived.
2. Children (Age<?) were more likely to have survived.
3. The upper-class passengers (Pclass=1) were more likely to have survived.

```
In [2]: df = pd.read_csv("titanic.csv")
        df.head()
```

Out[2]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1.0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1.0	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2	1.0	0.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
3	1.0	0.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
4	1.0	0.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON

From the table above, we can note a few things. First of all, that we **need to convert a lot of features into numeric** ones later on, so that the machine learning algorithms can process them. Furthermore, we can see that the **features have widely different ranges**, that we will need to convert into roughly the same scale. We can also spot some more features, that contain missing values (NaN = not a number), that we need to deal with.

Analyze by pivoting features

To confirm some of our observations and assumptions, we can quickly analyze our feature correlations by pivoting features against each other. We can only do so at this stage for features which do not have any empty values. It also makes sense doing so only for features which are categorical (Sex), ordinal (Pclass) or discrete (SibSp, Parch) type.

- **Pclass** We observe significant correlation (>0.5) among Pclass=1 and Survived. We decide to include this feature in our model.
- **Sex** We confirm the observation during problem definition that Sex=female had very high survival rate at 72%.
- **SibSp and Parch** These features have zero correlation for certain values. It may be best to derive a feature or a set of features from these individual features

```
In [8]: df[['pclass', 'survived']].groupby(['pclass'], as_index=False).mean().sort_values(by='survived', ascending=False)
```

Out[8]:

	pclass	survived
0	1.0	0.619195
1	2.0	0.429603
2	3.0	0.255289

```
In [6]: df[["sex", "survived"]].groupby(['sex'], as_index=False).mean().sort_values(by='survived', ascending=False)
```

Out[6]:

	sex	survived
0	female	0.727468
1	male	0.190985

```
In [7]: df[["parch", "survived"]].groupby(['parch'], as_index=False).mean().sort_values(by='survived', ascending=False)
```

Out[7]:

	parch	survived
3	3.0	0.625000
1	1.0	0.588235
2	2.0	0.504425
0	0.0	0.335329
4	4.0	0.166667
5	5.0	0.166667
6	6.0	0.000000
7	9.0	0.000000

Here we get the percent of null values in a column so we can decide whether to include or exclude the column or feature.

```
In [8]: total = df.isnull().sum().sort_values(ascending=False)
        percent_1 = df.isnull().sum()/df.isnull().count()*100
        percent_2 = (round(percent_1, 1)).sort_values(ascending=False)
        missing_data = pd.concat([total, percent_2], axis=1, keys=['Total', '%'])
        missing_data.head(8)
```

Out[8]:

	Total	%
body	1189	90.8
cabin	1015	77.5
boat	824	62.9
home.dest	565	43.1
age	264	20.2
embarked	3	0.2
fare	2	0.2
ticket	1	0.1

The Embarked feature has only 3 missing values, which can easily be filled.

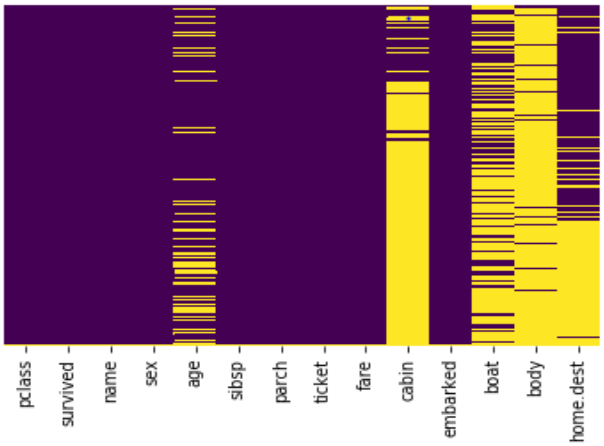
The Fare feature has only 2 missing values, which can easily be filled.

It will be much trickier, to deal with the ‘age’ feature, which has 264 missing values respectively so we might do further investigation.

```
In [123]: #Dropping the not useful columns
          df.drop('body',axis=1, inplace=True)
          df.drop('boat', axis=1, inplace=True)
          df.drop('cabin', axis=1, inplace=True)
          df.drop('home.dest', axis=1, inplace = True)
```

```
In [4]: #All the null values have been filled out
        sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```

```
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x243ec7c1d30>
```



The ‘Cabin’ feature along with ‘Home.dest’ needs further investigation, but it looks like that we might want to drop it from the dataset, since 77.5% and 43%of it are missing.

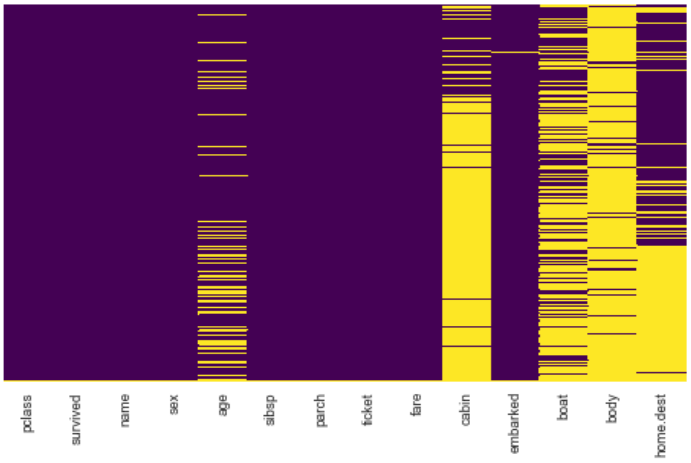
Also, the ‘Boat’ feature needs further investigation, but it looks like that we might want to drop it from the dataset, since 62.9 % of it are missing.

Also, since the body feature has 90.9% missing values we should straight away drop it from table since it doesn’t gives significance to our data.

EDA:

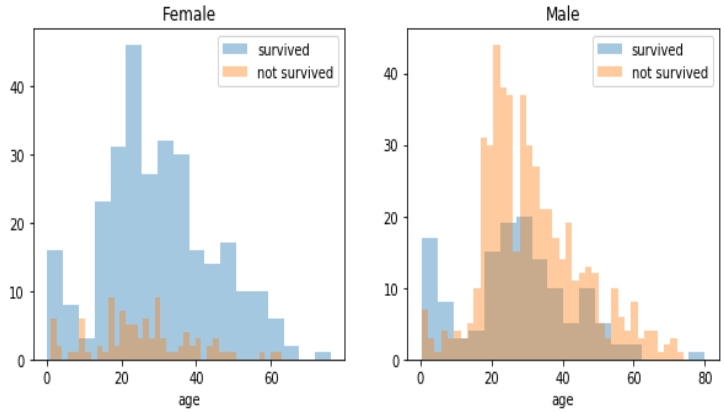
In [31]:
```
#We can see the columns that have missing values and will deal with it in the next section
plt.figure(figsize=(8,5))
sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1efb7e36358>



We can show the distribution of “NAN” values using heatmap also,which shows us to discard columns with high no. of “NAN” values.

In [13]:
```
survived = 'survived'
not_survived = 'not survived'
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(10, 4))
women = df[df['sex']=='female']
men = df[df['sex']=='male']
ax = sns.distplot(women[women['survived']==1].age.dropna(), bins=18, label = survived, ax = axes[0], kde =False)
ax = sns.distplot(women[women['survived']==0].age.dropna(), bins=40, label = not_survived, ax = axes[0], kde =False)
ax.legend()
ax.set_title('Female')
ax = sns.distplot(men[men['survived']==1].age.dropna(), bins=18, label = survived, ax = axes[1], kde = False)
ax = sns.distplot(men[men['survived']==0].age.dropna(), bins=40, label = not_survived, ax = axes[1], kde = False)
ax.legend()
_ = ax.set_title('Male')
```

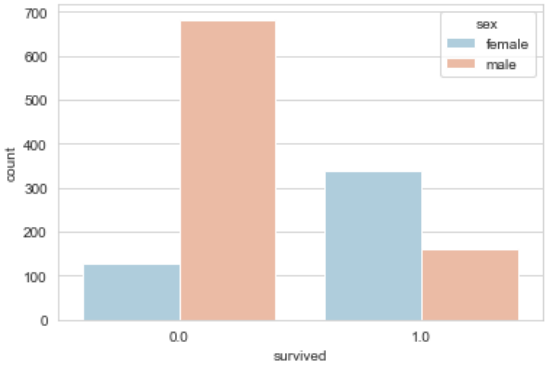


We can see that men have a high probability of survival when they are between 18 and 30 years old, which is also a little bit true for women but not fully. For women the survival chances are higher between 14 and 40.

For men the probability of survival is very low between the age of 5 and 18, but that isn't true for women. Another thing to note is that infants also have a little bit higher probability of survival.

```
In [21]: #It shows that from the people who dint survive maximum were males and almost twice females survi
         sns.countplot(x='survived',data=df, hue='sex', palette='RdBu_r')
```

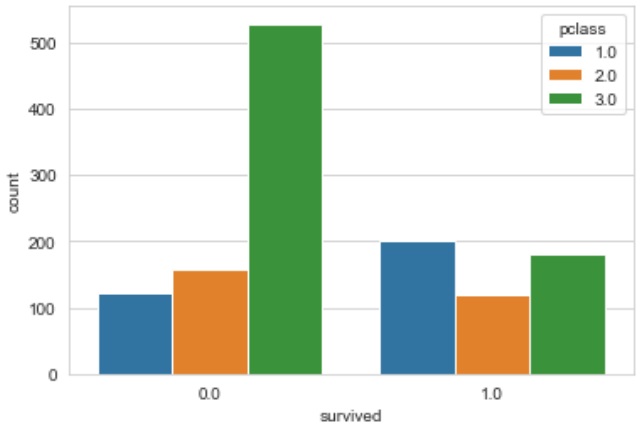
```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1efb7689390>
```



It shows that from the people who didn't survive maximum were males and almost twice the number of females survived.

```
In [22]: #The third class passangers were amazingly most among Non-Survivors
         sns.countplot(x='survived',data=df, hue='pclass')
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1efb7739400>
```

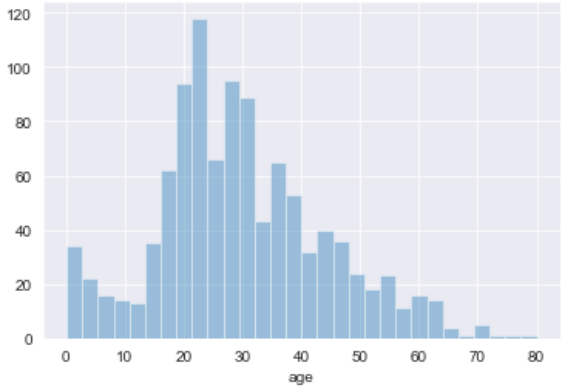


We can infer that, if you were a female and you were in class 1, probably you would survive.
On the other hand, if you were a man and you were in class 3, you didn't have good chances to live and least chances of survival.

```
In [16]: #There were few senior citizens and most people belonged to the younger age group
         sns.set_style('darkgrid')

         sns.distplot(df['age'].dropna(),kde=False, bins=30)
```

```
Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x1efb5d33630>
```

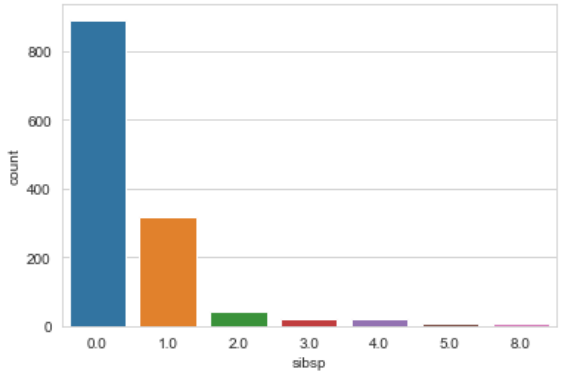


This plot shows the distribution of age of the passengers on the boat from which we can infer that there were very few Senior citizens on the ship. Since we can see the highest frequency at the age 23, we know the maximum passengers were of 23 years old.

Also since we can see that there is high frequency of passengers between the age of 20-40 which tells us that the maximum no. of passengers belong to younger age group.

```
In [20]: #Category 0 were single's with no spouse or children, Category 1 are probably married couples.
         sns.set_style('whitegrid')
         sns.countplot(x='sibsp', data=df)
```

```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1efb5d2cac8>
```



Here we can see category of different people who survived the disaster where

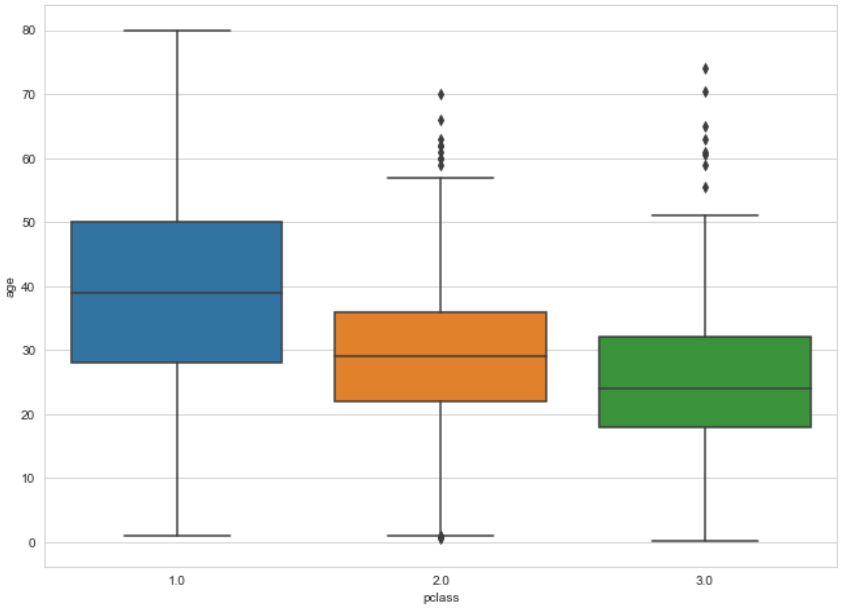
Category 0 might be those who were single without any family

While Category 1 are probably married couples.

```
In [47]: #We can see the wealthier passangers from 1st and 2nd class tend to be older from the passangers in 3rd class
         sns.set_style('whitegrid')
         plt.figure(figsize=(11,8))

         sns.boxplot(x='pclass', y='age', data=df)
```

```
Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1efc16880f0>
```



Here we show different classes of passengers and their age and we see if their age and class is corelated. Here, we can see the wealthier passengers from 1st and 2nd class tend to be older from the passengers in 3rd class.

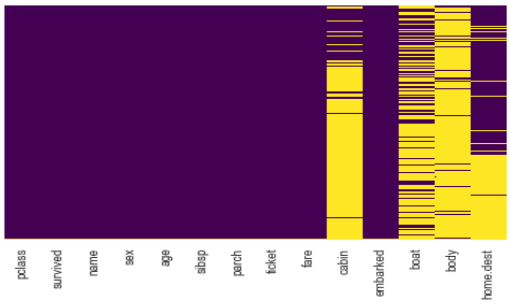
Hence, we deduce that oldest passengers are in 1st class as mean age of 1st class is 40 years.

Mean age of 2nd class is 30 years. Mean age of 3rd class is 25 years.

```
In [125]: df['age'] = df[['age','pclass']].apply(fill_age, axis=1) #filling null values with mean ages
```

```
In [51]: #We have filled all the null values for the age column
         sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```

```
Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x1efbd3fe6d8>
```



Here we fill the null values with appropriate values, like for e.g. We can use these average age values depending upon the classes to fill on the missing data i.e “NAN” values.

Feature Importance

Another great quality of random forest is that they make it very easy to measure the relative importance of each feature. Sklearn measure a features importance by looking at how much the treee nodes, that use that feature, reduce impurity on average (across all trees in the forest). It computes this score automatically for each feature after training and scales the results so that the sum of all importance is equal to 1.

ADDING COLUMNS USING DUMMIES

Here we are done with cleaning the data. We are going to convert some categorical data into numeric. For example, the sex column.
Let's use the "GET_DUMMIES"function of Pandas. It will create two columns, one for male, one for female.
What we can do is to remove the first column because one column indicates the value of the other column.
For example, if the male is 1, then the female will be 0 and vice versa.

```
In [21]: #We drop the first column because then these two columns will become perfect predictors of each other
         sex = pd.get_dummies(df['sex'], drop_first=True)
         sex.head()
```

Out[21]:

	male
0	0
1	1
2	0
3	1
4	0

Let's do the same for Embarked and PClass:

```
In [28]: # C column is dropped and its value can be obtained when both are 0
         embark = pd.get_dummies(df['embarked'],drop_first=True)
         embark.head()
```

Out[28]:

	Q	S
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1

```
In [26]: pclass = pd.get_dummies(df['pclass'],drop_first=True)
         pclass.head()
```

Out[26]:

	2.0	3.0
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

We add these variables to the dataset:

```
In [29]: #Adding these columns to the dataframe
         df = pd.concat([df,sex,embark,pclass],axis=1)
```

```
In [30]: df.head()
```

Out[30]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest	male	Q	S	2.0	3.0
0	1.0	1.0	Allen, Miss. Elisabeth Walton	female	29.0000	0.0	0.0	24160	211.3375	B5	S	2	NaN	St Louis, MO	0	0	1	0	0
1	1.0	1.0	Allison, Master. Hudson Trevor	male	0.9167	1.0	2.0	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON	1	0	1	0	0
2	1.0	0.0	Allison, Miss. Helen Loraine	female	2.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON	0	0	1	0	0
3	1.0	0.0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON	1	0	1	0	0
4	1.0	0.0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1.0	2.0	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON	0	0	1	0	0

After Adding the Columns “Sex”, “Embark” and “Pclass” in the dataset,We need to drop them along with “name” and “ticket” as we have encoded them to other columns.

```
In [89]: #Dropping pclass,embarked and sex because we have encoded it to other columns
```

```
In [90]: df.drop(['pclass','sex','embarked','name','ticket'], axis=1, inplace = True)
```

```
In [91]: df.dropna(inplace=True)
```

```
In [92]: #Finally we have out data where all columns are numberical perfect for the algorithm
         df.head()
```

Out[92]:

	survived	age	sibsp	parch	fare	male	Q	S	2.0	3.0
0	1.0	29.0000	0.0	0.0	211.3375	0	0	1	0	0
1	1.0	0.9167	1.0	2.0	151.5500	1	0	1	0	0
2	0.0	2.0000	1.0	2.0	151.5500	0	0	1	0	0
3	0.0	30.0000	1.0	2.0	151.5500	1	0	1	0	0
4	0.0	25.0000	1.0	2.0	151.5500	0	0	1	0	0

After performing above code, we finally have data where all columns are numerical which is perfect for our algorithm and further exploration for machine learning.

LOGISTIC REGRESSION ACCURACY

```
In [157]: from sklearn.metrics import classification_report

In [169]: print(classification_report(y_test,predictions))
```

	precision	recall	f1-score	support
0.0	0.82	0.85	0.84	240
1.0	0.75	0.70	0.73	153
accuracy			0.79	393
macro avg	0.79	0.78	0.78	393
weighted avg	0.79	0.79	0.79	393

It gives us Accuracy of 79% .

```
n [159]: from sklearn.metrics import confusion_matrix

n [167]: cm = confusion_matrix(y_test, predictions)
         cmatrix = pd.DataFrame(cm, columns=['Predicted:\n NO', 'Predicted:\n YES'], index=['Actual: NO', 'Actual: YES'])
         cmatrix
```

ut[167]:

	Predicted: NO	Predicted: YES
Actual: NO	205	35
Actual: YES	46	107

True positive: 205 (We predicted a positive result and it was positive)
True negative: 107(We predicted a negative result and it was negative)
False positive: 35 (We predicted a positive result and it was negative)
False negative: 46 (We predicted a negative result and it was positive)

Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

LOGISTIC REGRESSION PREDICTION ERROR

```
In [25]: print("MAE is:\t",metrics.mean_absolute_error(y_test,y_pred))
         print("MSE is:\t",metrics.mean_squared_error(y_test, y_pred))
         print("RMSE is:",np.sqrt(metrics.mean_squared_error(y_test, y_pred))

MAE is:  0.01090101412486229
MSE is:  0.01177316900270165
RMSE is: 0.1085042349528425
```

KNN ALGORITHM CLASSIFICATION REPORT AND CONFUSION MATRIX

```
In [51]: from sklearn.metrics import classification_report,confusion_matrix
         print(confusion_matrix(y_test,pred))
         print(classification_report(y_test,pred))

[[196  43]
 [ 60  94]]
              precision    recall  f1-score   support

         0.0       0.77      0.82      0.79       239
         1.0       0.69      0.61      0.65       154

    accuracy                           0.74       393
   macro avg       0.73      0.72      0.72       393
weighted avg       0.73      0.74      0.73       393
```

The K-Nearest Neighbours (K-NN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. The K-NN algorithm assumes that similar things exist close. K-NN makes predictions using the training dataset directly. Predictions are made for a new instance (x) by searching through the entire training set for the K most similar instances (the neighbours) and summarizing the output variable for those K instances. For regression this might be the mean output variable, in classification, this might be the mode (or most common) class value. To determine which of the K instances in the training dataset are most similar to a new input a distance measure is used. For real-valued input variables, the most popular distance measure is Euclidean distance.

KNN ALGORITHM ACCURACY

```
In [186]:
# NOW WITH K=5
knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=5')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=5


[[218  35]
 [ 46  94]]


              precision    recall  f1-score   support

         0.0       0.83      0.86      0.84       253
         1.0       0.73      0.67      0.70       140

    accuracy                           0.79       393
   macro avg       0.78      0.77      0.77       393
weighted avg       0.79      0.79      0.79       393
```

SVM ACCURACY

```
In [31]:
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
accuracy = float(cm.diagonal().sum())/len(y_test)
print("\nAccuracy Of SVM For The Given Dataset : ", accuracy)
```

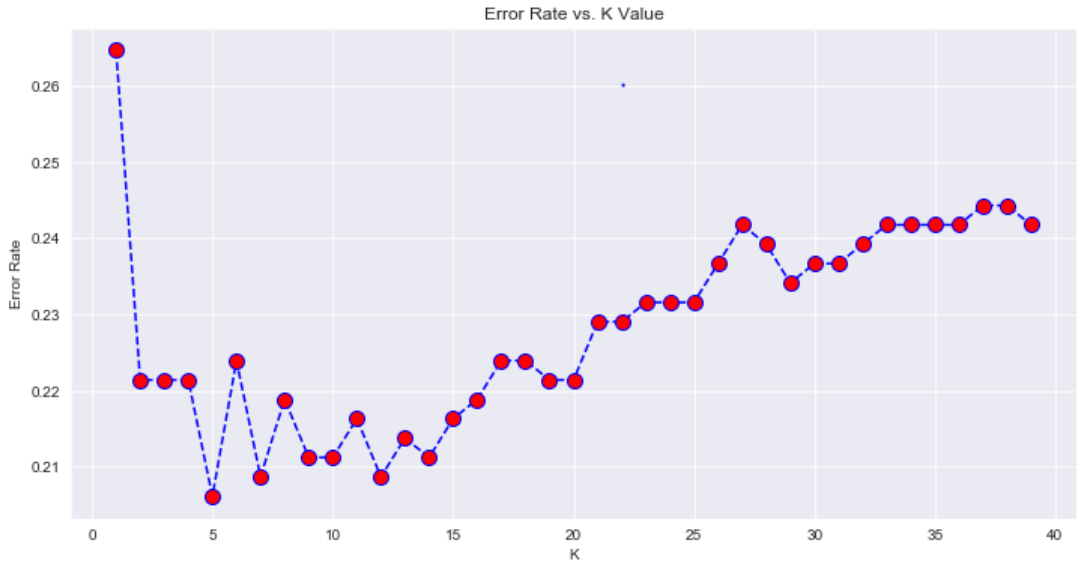
```
Accuracy Of SVM For The Given Dataset :  0.6844783715012722
```

Support Vector Machines are perhaps one of the most popular and talked about machine learning algorithms. The model produced by support-vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

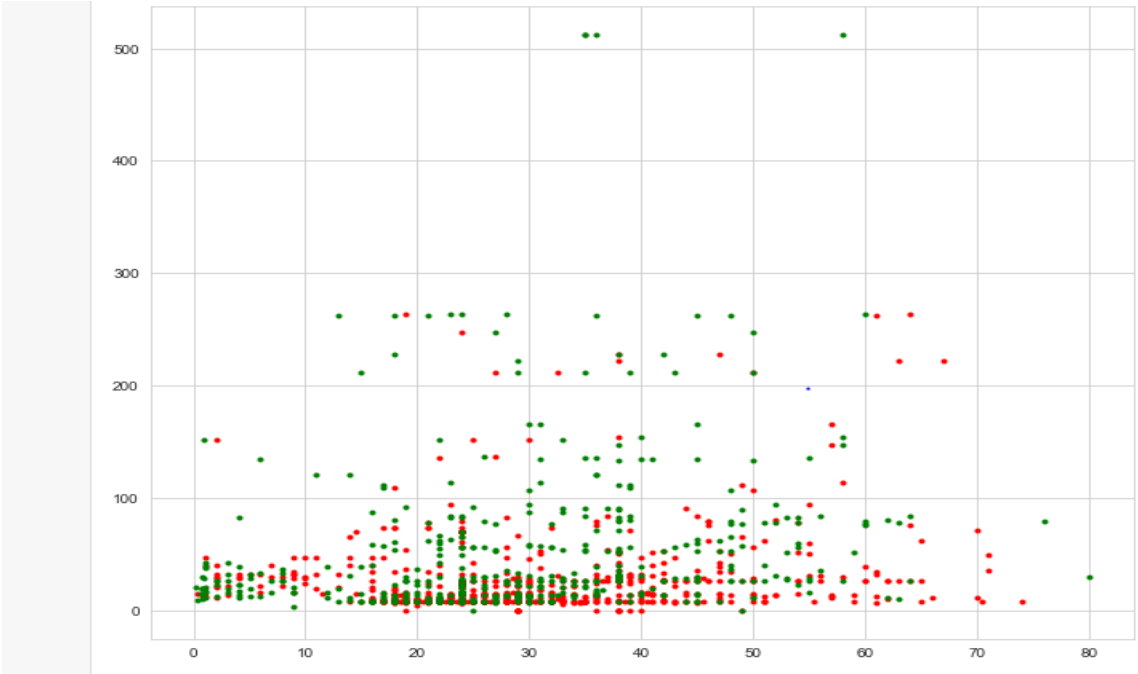
METHOD	ACCURACY
LOGISTIC REGRESSION	79%
KNN ALGORITHM	74%
SUPPORT VECTOR MACHINE	68.44%

MODEL EVALUATION

KNN – ALGORITHM:



SUPPORT VECTOR MACHINE:



Conclusion:

We started with the data exploration where we got a feeling for the dataset, checked about missing data and learned which features are important. During this process we used seaborn and matplotlib to do the visualizations. During the data pre-processing part, we computed the missing null values, converted features into numeric ones, grouped values into categories and created a few new features. Afterwards we started training 8 different machine learning models, picked one of them (random forest) and applied cross validation on it. Then we discussed how random forest works, took a look at the importance it assigns to the different features and tuned its performance through optimizing its parameter values.

Using different methods and analysing the derived metrics, as well as finding and plotting the AUC and ROC plots, we found that the KNN model with neighbour=5 has outperformed all the other models. Thereby stating that KNN model to be the most accurate model for the titanic dataset.