TASK P - 1.2

ML Development using Python

Survival prediction on
Titanic dataset

Saurabh Dharmadhikari
Saurabh.dharma01@gmail.com

Table of Content:

1. **What is the selected dataset and what is related problem for this dataset? You need to provide details of datasets, dataset description, what are the features, output (class label) and discuss the problem needs to be solved by machine learning model. (Minimum 200 words)**

The Titanic dataset is one of the most famous datasets used in machine learning. It is based on the passengers who were aboard the RMS Titanic during its maiden voyage that sank on April 15, 1912. The dataset contains information on 891 of the 2,224 passengers and crew members.

The dataset includes various attributes such as age, gender, class, fare, cabin, ticket number, and embarkation port, among others. The target variable in this dataset is survival, which indicates whether a passenger survived the disaster or not. This makes it a binary classification problem where the aim is to predict whether a passenger survived or not based on the given features.

Some of the features in the dataset contain missing values, making data pre-processing an essential part of this task. We have used the dataset to build decision tree and random forest to predict survivors and attributes contributing to it.

Sample dataset:

	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

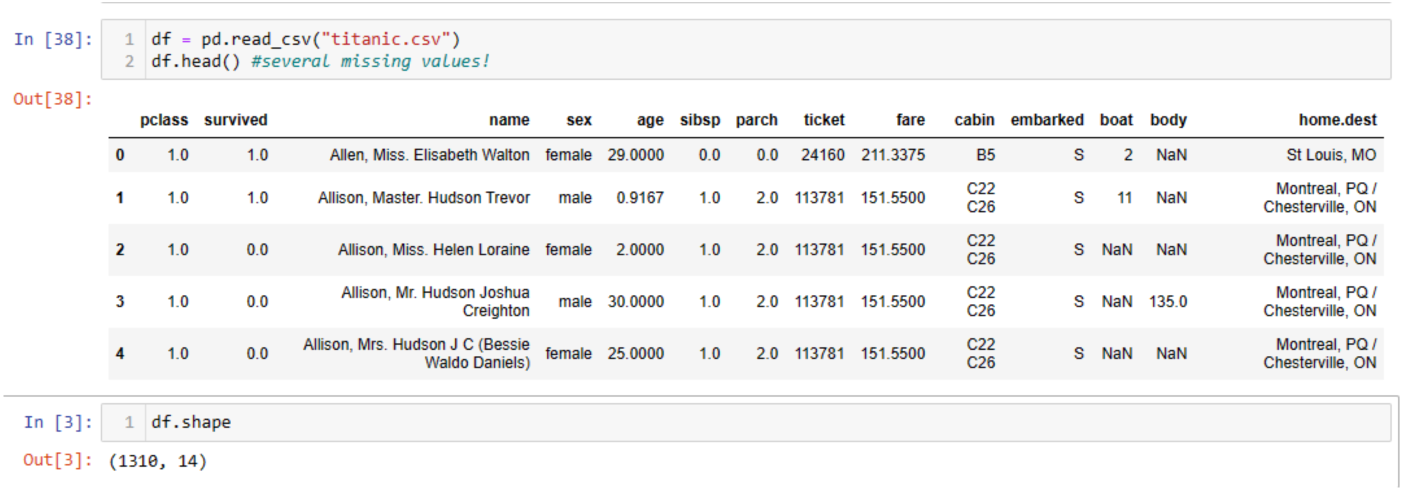
Data Description:

	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

We can easily see from the above table different statistical value of features in the dataset. Average age of people on board is 29. Average fare of the voyage is 33 with maximum being 512. Other such helpful statistical values can be seen above.

2. **You need to provide the screenshot of the built ML pipeline (Data ingestion, Data preparation, model training and evaluating the model). You need to provide explanation for cell by cell of the code.**

Data Ingestion:



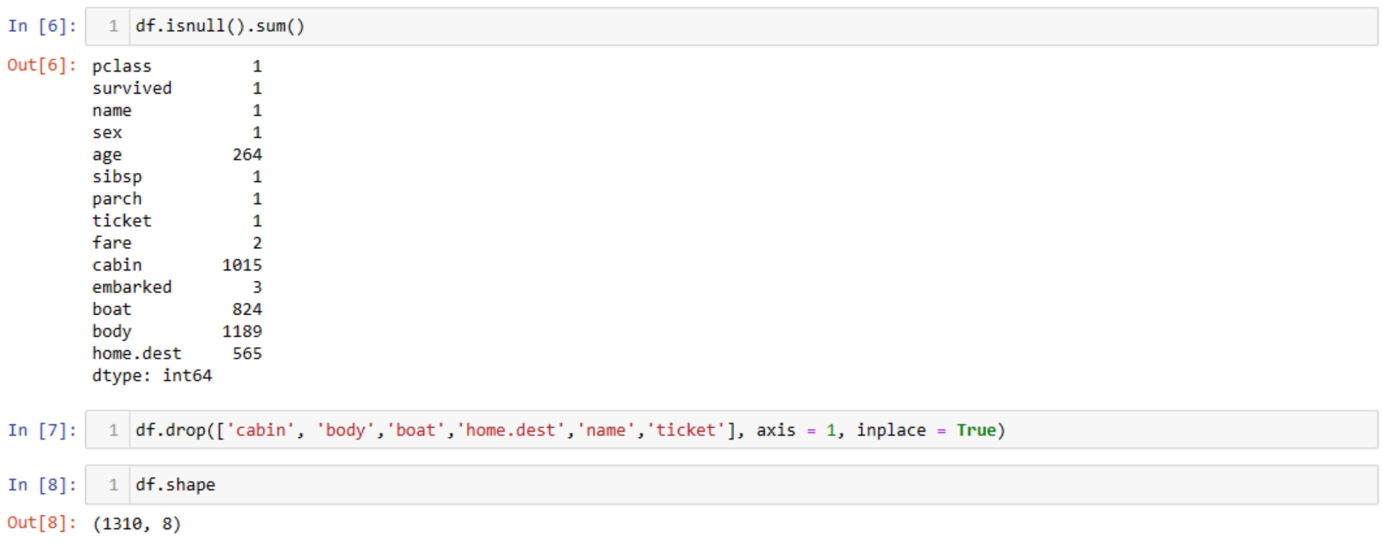
```
In [38]: df = pd.read_csv("titanic.csv")
         df.head() #several missing values!
Out[38]:
```

	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

```
In [3]: df.shape
Out[3]: (1310, 14)
```

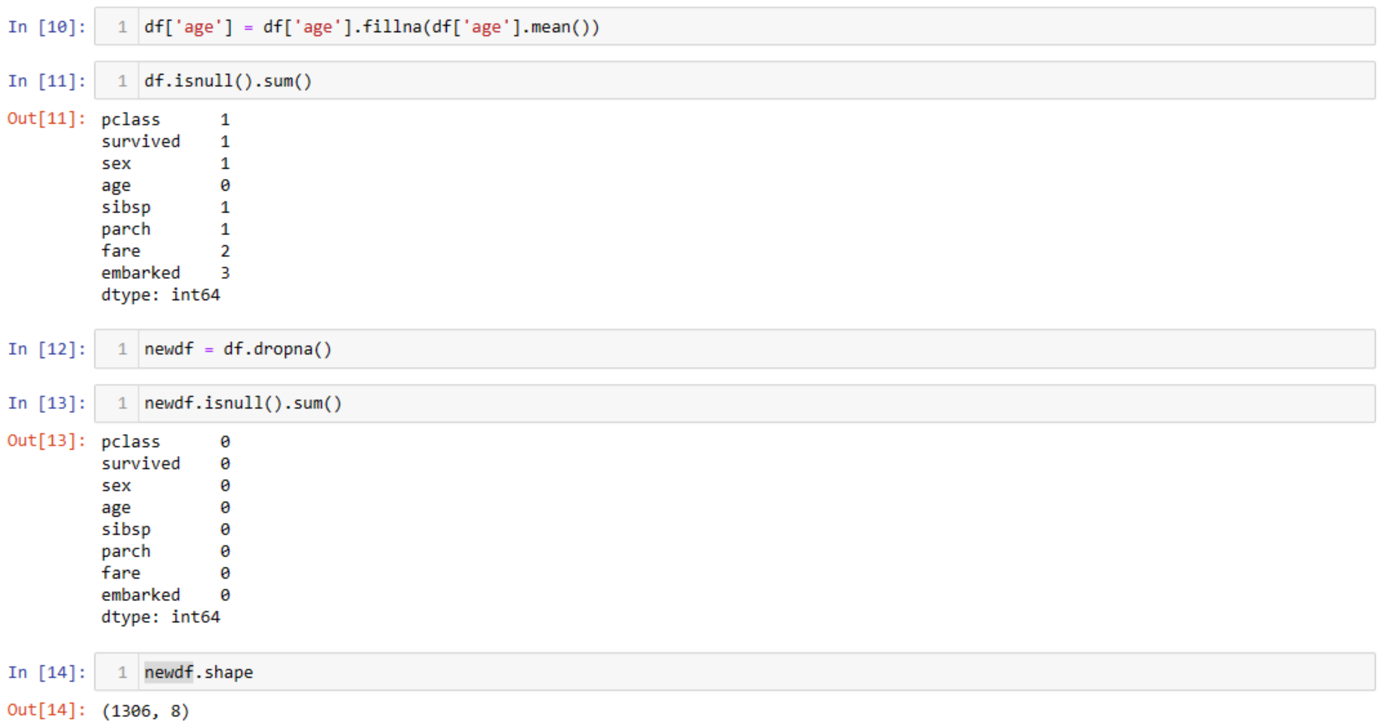
We uploaded the dataset provided.

Data preparation:



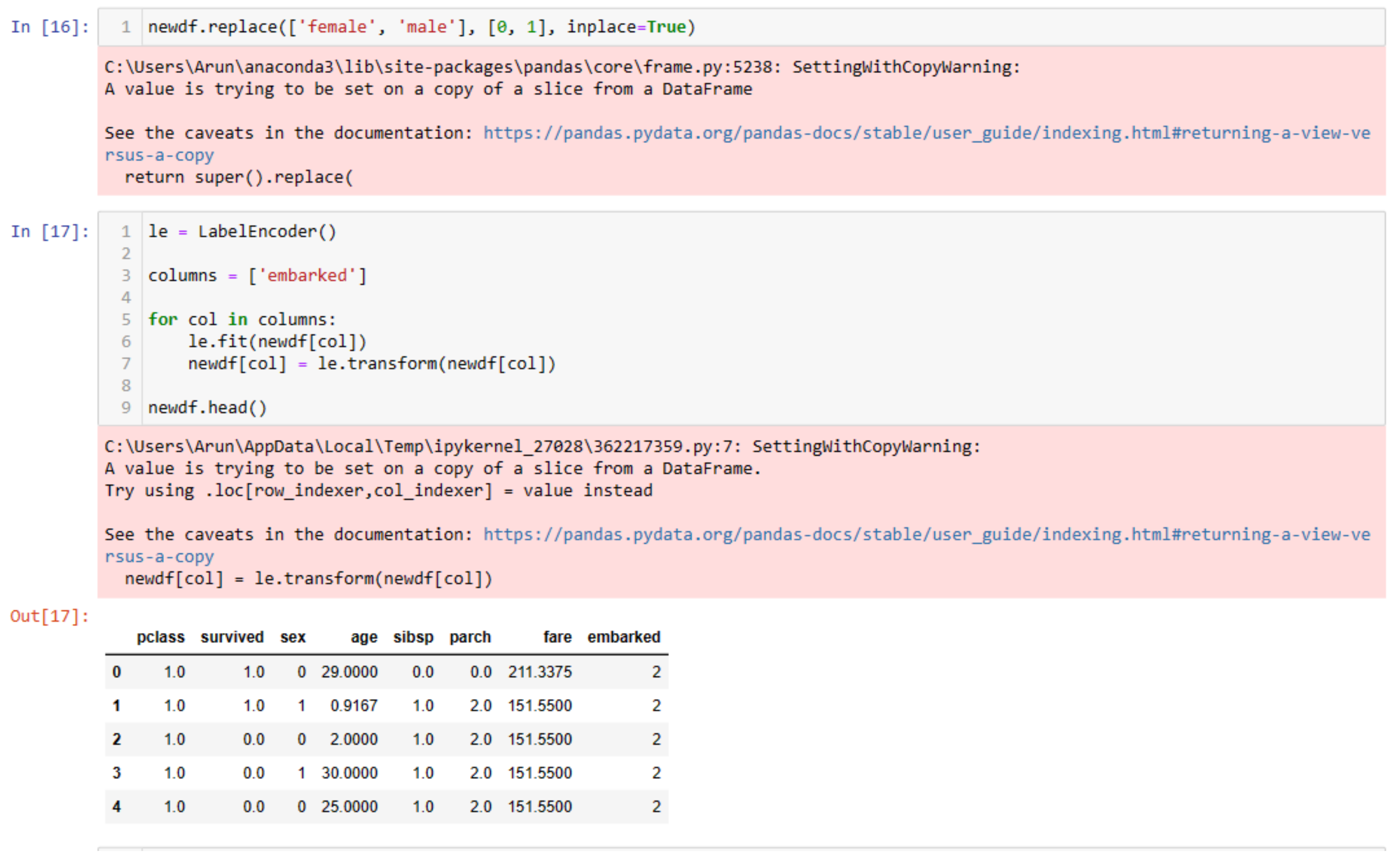
```
In [6]:  df.isnull().sum()

Out[6]:  pclass          1
         survived        1
         name            1
         sex             1
         age           264
         sibsp           1
         parch           1
         ticket          1
         fare            2
         cabin        1015
         embarked        3
         boat          824
         body         1189
         home.dest     565
         dtype: int64

In [7]:  df.drop(['cabin', 'body','boat','home.dest','name','ticket'], axis = 1, inplace = True)

In [8]:  df.shape

Out[8]:  (1310, 8)
```

Here we can see that there are a lot of missing values, especially in features like cabin, boat, body, home.dest and age. So we have dropped cabin, body, home.dest, name and ticket. So we are left with 8 features.



```
In [10]:  df['age'] = df['age'].fillna(df['age'].mean())

In [11]:  df.isnull().sum()

Out[11]:  pclass      1
          survived    1
          sex         1
          age         0
          sibsp       1
          parch       1
          fare        2
          embarked    3
          dtype: int64

In [12]:  newdf = df.dropna()

In [13]:  newdf.isnull().sum()

Out[13]:  pclass      0
          survived    0
          sex         0
          age         0
          sibsp       0
          parch       0
          fare        0
          embarked    0
          dtype: int64

In [14]:  newdf.shape

Out[14]:  (1306, 8)
```

For feature age we filled the null values with mean value. We also removed a few rows which had null values in other features of this dataset. Now we are left with 1306 rows and 8 features.



```
In [16]: newdf.replace(['female', 'male'], [0, 1], inplace=True)
```

```
C:\Users\Arun\anaconda3\lib\site-packages\pandas\core\frame.py:5238: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  return super().replace(
```

```
In [17]: le = LabelEncoder()

columns = ['embarked']

for col in columns:
    le.fit(newdf[col])
    newdf[col] = le.transform(newdf[col])

newdf.head()
```

```
C:\Users\Arun\AppData\Local\Temp\ipykernel_27028\362217359.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  newdf[col] = le.transform(newdf[col])
```

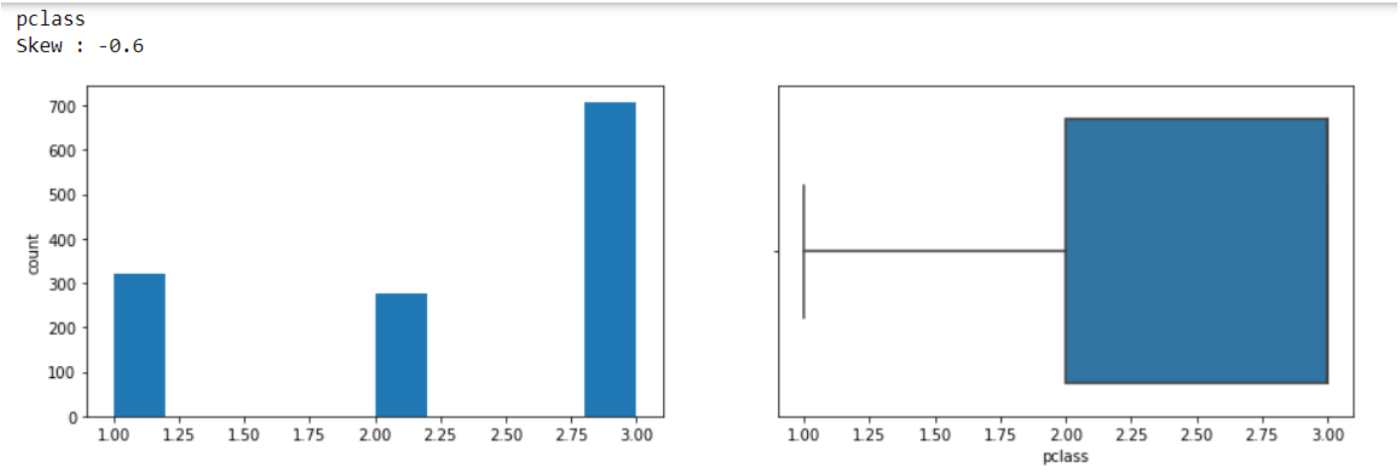
Out[17]:

	pclass	survived	sex	age	sibsp	parch	fare	embarked
0	1.0	1.0	0	29.0000	0.0	0.0	211.3375	2
1	1.0	1.0	1	0.9167	1.0	2.0	151.5500	2
2	1.0	0.0	0	2.0000	1.0	2.0	151.5500	2
3	1.0	0.0	1	30.0000	1.0	2.0	151.5500	2
4	1.0	0.0	0	25.0000	1.0	2.0	151.5500	2

We replaced female with 0 and male with 1 in the sex column and used label encoding for embark feature. Thus, we have all the features in numeric form and is ready to be used in decision tree.

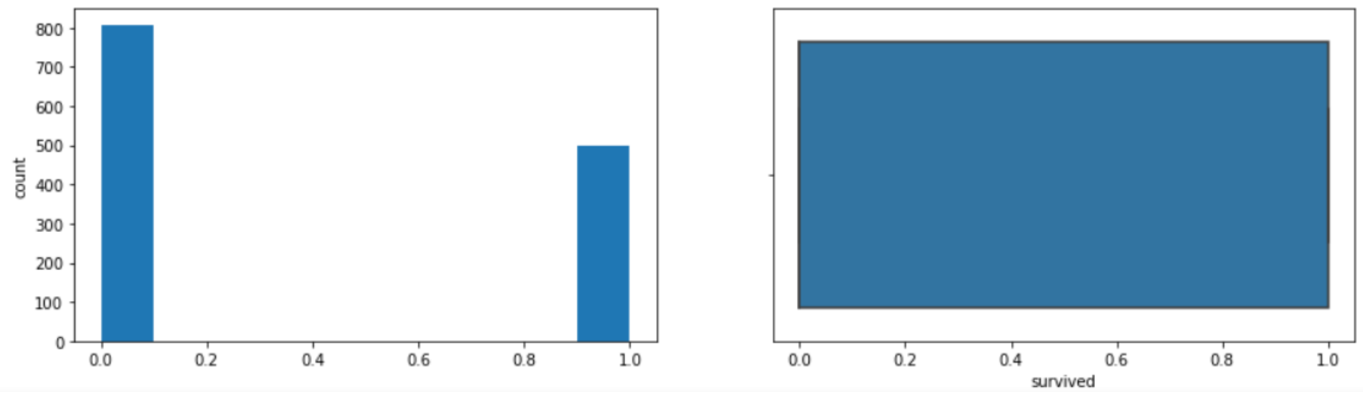
Let us first do some exploratory data analysis just before that as now we have data prepared.

There are no outliers in feature pclass:



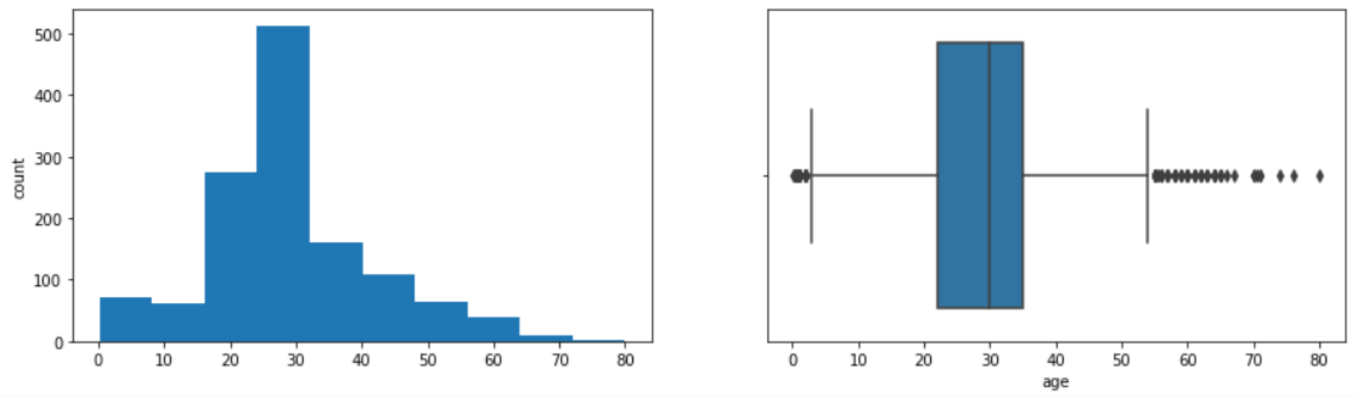
```
pclass
Skew : -0.6
```

```
survived
Skew : 0.49
```



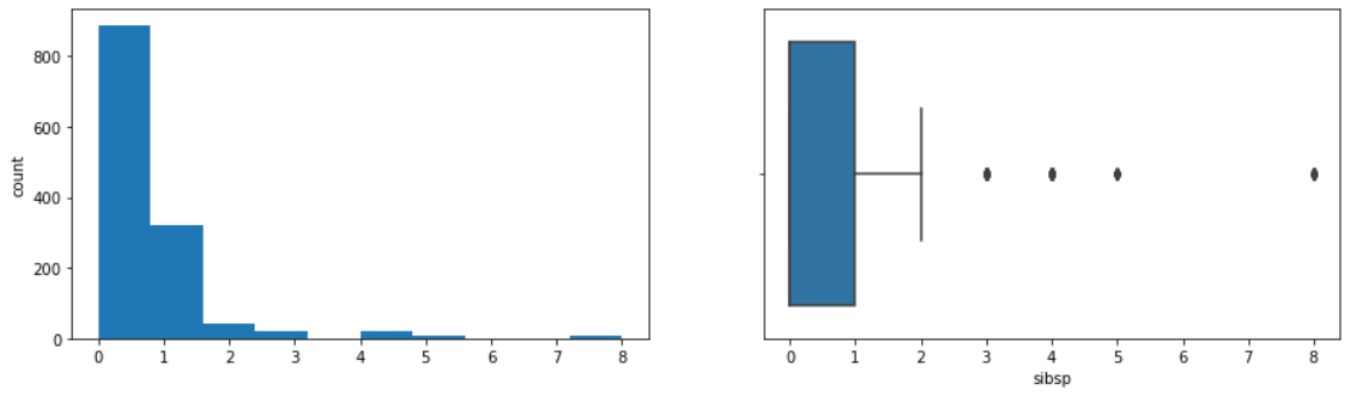
From the above bar chart, we can see that almost 800 people could not survive from the sample set that we have taken and close to 450 people survived the titanic disaster.

```
age
Skew : 0.45
```



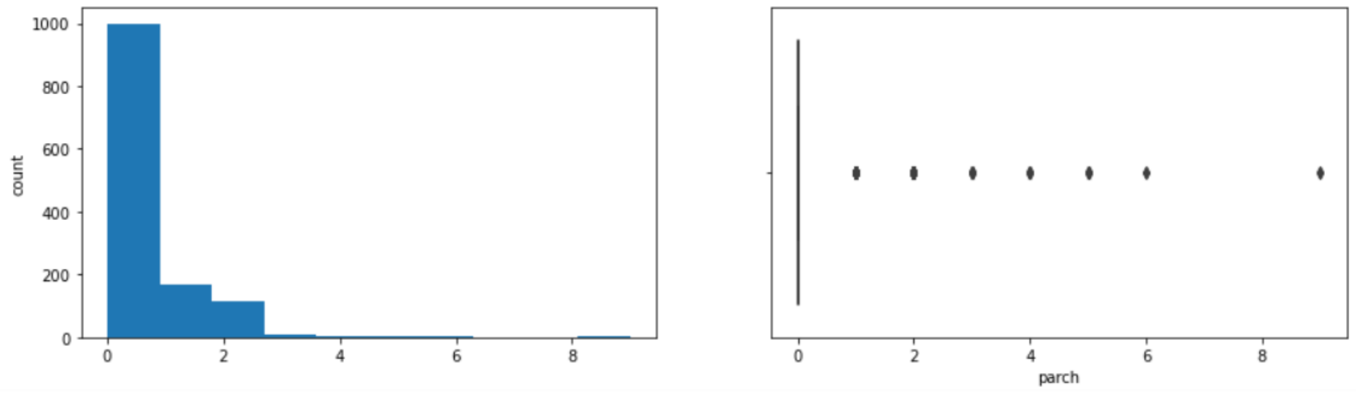
In the above box plot we see that there are numerous outliers in feature age but none of them are uncommon so we will presume that the information collected is correct and good to go with. Through bar chart we observe that most people boarding the ship were from the age group of 25 to 30.

```
sibsp
Skew : 3.84
```



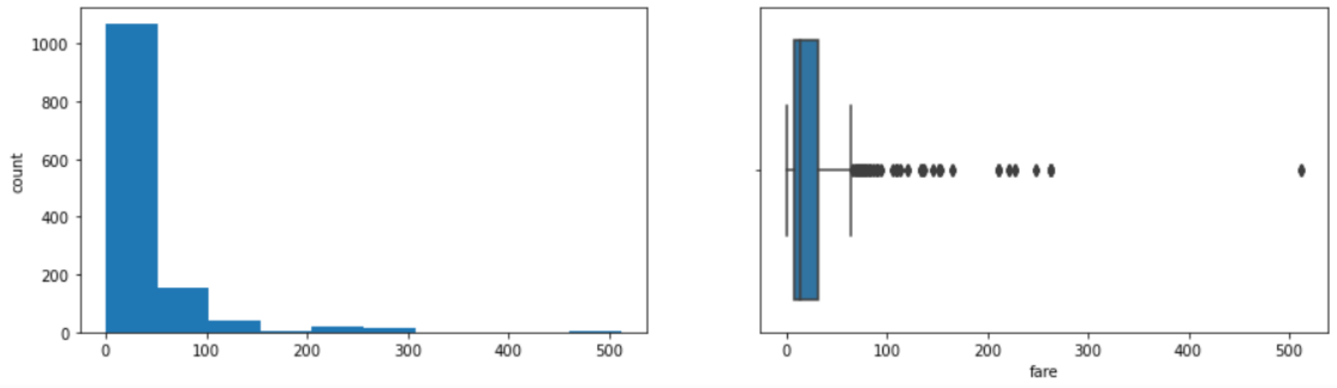
Sibling or spouse aboard feature shows most passengers did not have any siblings or spouse travelling with them. There are also a few outliers with maximum number being 8.

```
parch
Skew : 3.66
```

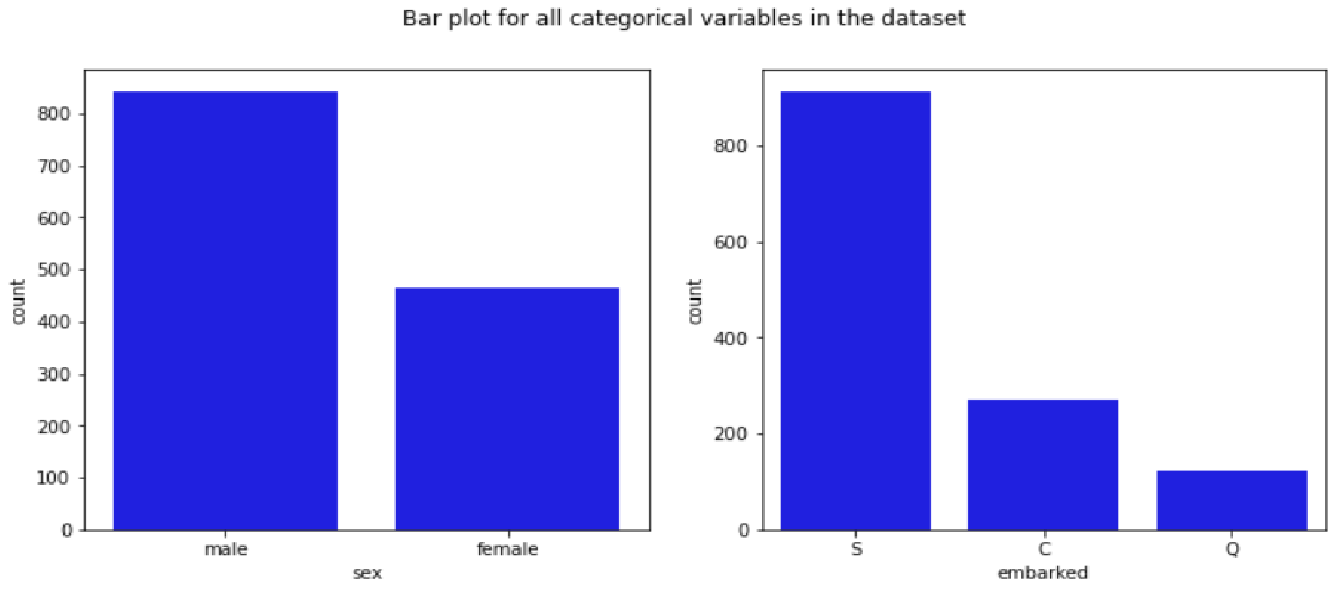


Above feature shows us that chances of parents or children travelling with a passenger is very low. Again, there are a few outliers with maximum as 9.

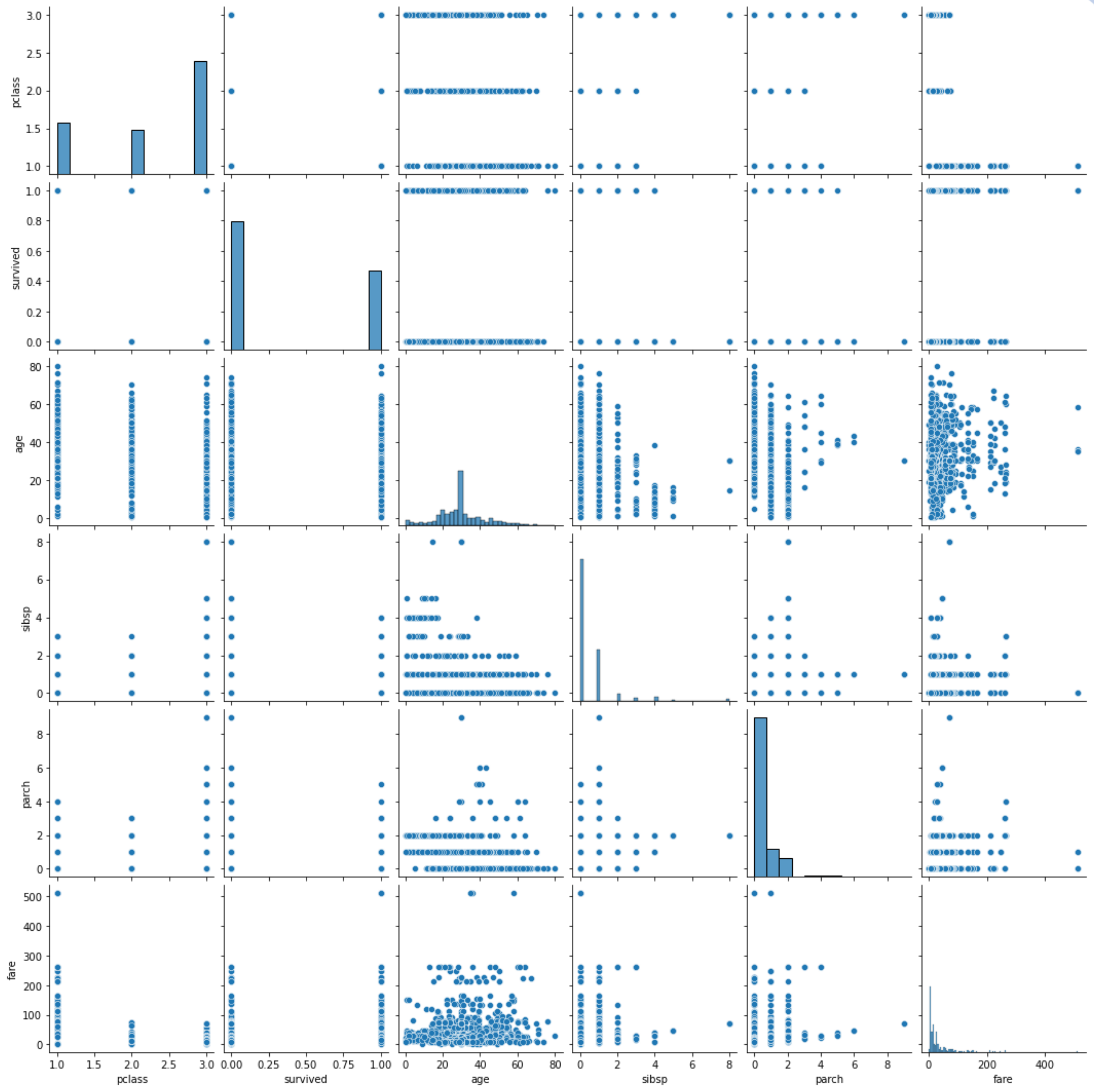
```
fare
Skew : 4.38
```



Fare on an average is 33 for one passenger with a minimum of 0 going up to around 500 in British currency, which is an outlier.



We can now see here that number of males in our sample dataset is more than female with about 850 whereas females count about 450. Also on the right in the above bar chart we find most people started their journey from Southampton(850), about 240 from Cherbourg and about 100 from Queenstown.

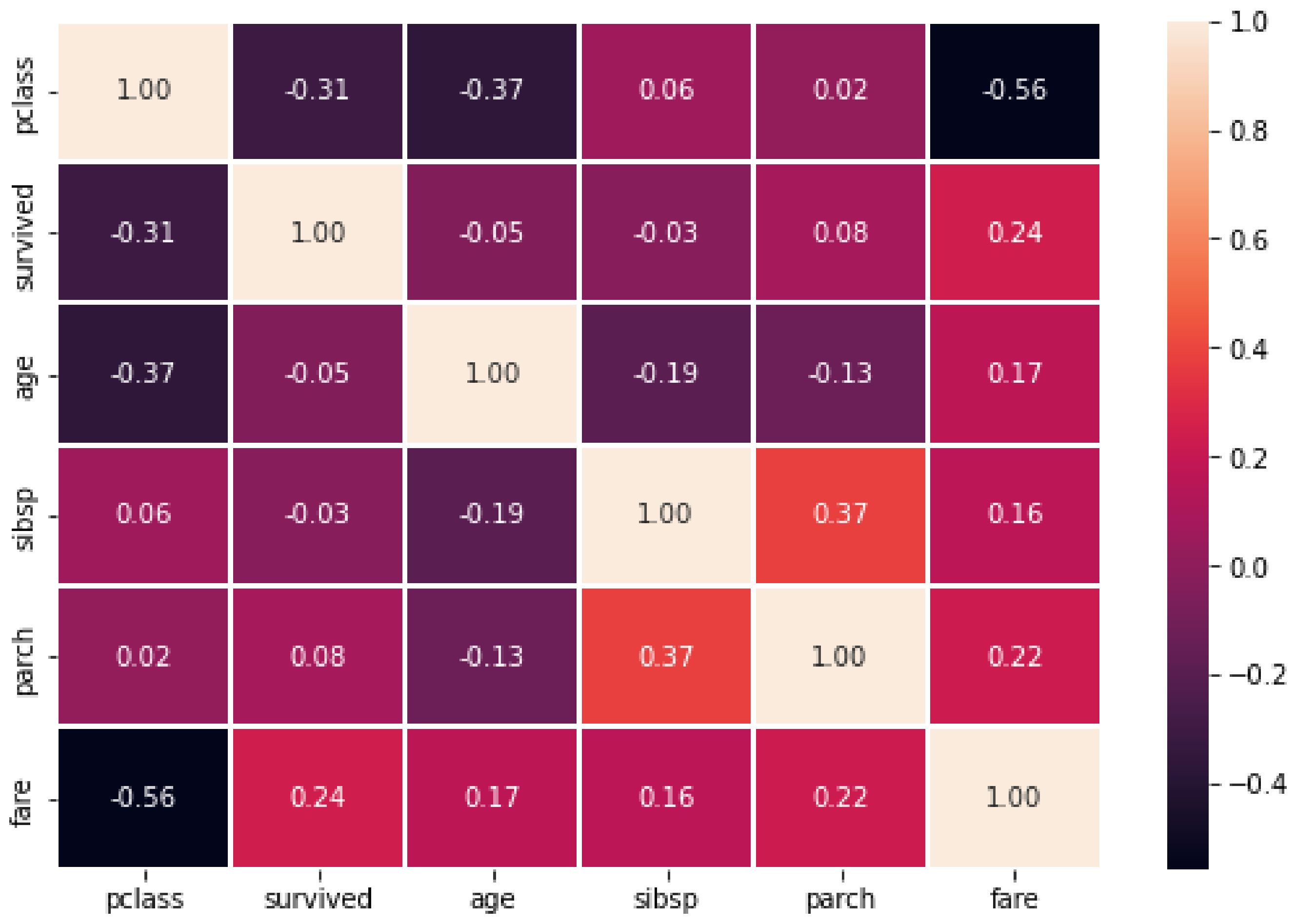


Above is a scatter plot for the prepared dataset.

Age and survived scatter plot equally distributed.

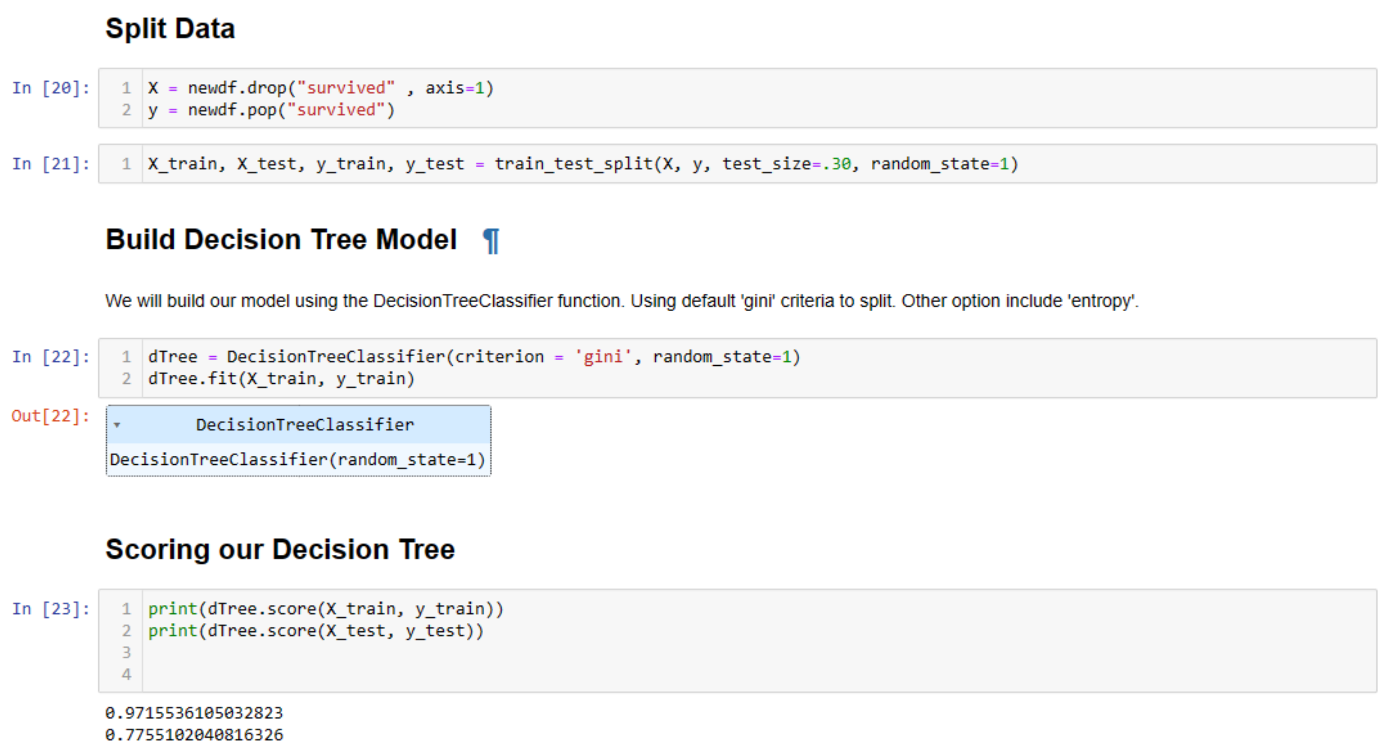
We do not find much linear pattern in the scatter plots we created.

Let us see the correlation heatmap for better understanding:



Here we are able to see that pclass and fare has a good negative correlation. Otherwise we do not have a very good correlation between features.

Let us now move ahead and split data to train and test to build and test our models:



Split Data

```
In [20]:  X = newdf.drop("survived" , axis=1)
          y = newdf.pop("survived")
```

```
In [21]:  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.30, random_state=1)
```

Build Decision Tree Model ¶

We will build our model using the DecisionTreeClassifier function. Using default 'gini' criteria to split. Other option include 'entropy'.

```
In [22]:  dTree = DecisionTreeClassifier(criterion = 'gini', random_state=1)
          dTree.fit(X_train, y_train)
```

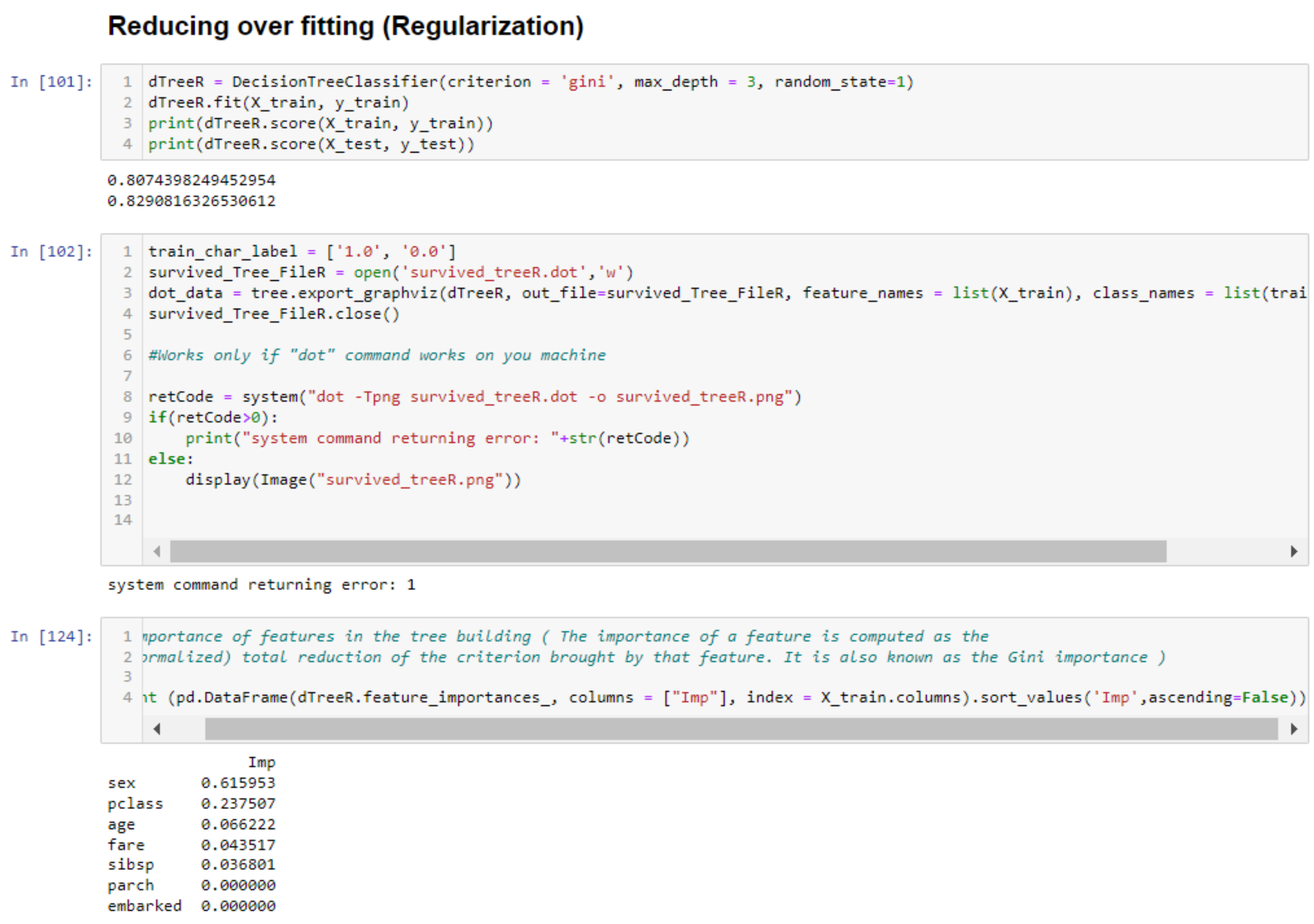
```
Out[22]:  ▾       DecisionTreeClassifier
          DecisionTreeClassifier(random_state=1)
```

Scoring our Decision Tree

```
In [23]:  print(dTree.score(X_train, y_train))
          print(dTree.score(X_test, y_test))
```

```
0.9715536105032823
0.7755102040816326
```

We split our data in 70:30 ratio. Used gini criterion and built our decision tree. We got a score of 97.1% on train set and 77.5% score on our test set. Thus, this model looks to be over fit.



Reducing over fitting (Regularization)

```
In [101]:  dTreeR = DecisionTreeClassifier(criterion = 'gini', max_depth = 3, random_state=1)
           dTreeR.fit(X_train, y_train)
           print(dTreeR.score(X_train, y_train))
           print(dTreeR.score(X_test, y_test))
```

```
0.8074398249452954
0.8290816326530612
```

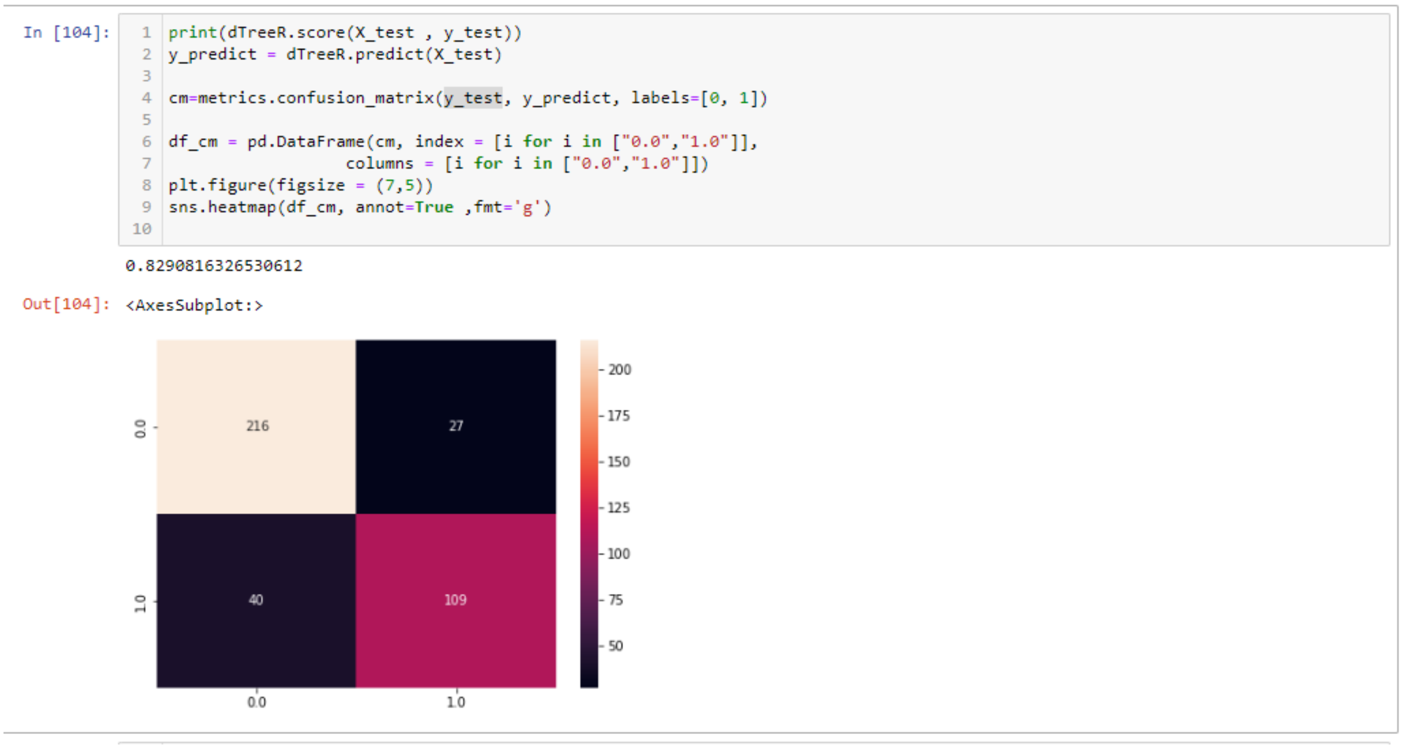
```
In [102]:  train_char_label = ['1.0', '0.0']
           survived_Tree_FileR = open('survived_treeR.dot','w')
           dot_data = tree.export_graphviz(dTreeR, out_file=survived_Tree_FileR, feature_names = list(X_train), class_names = list(trai
           survived_Tree_FileR.close()

           #Works only if "dot" command works on you machine

           retCode = system("dot -Tpng survived_treeR.dot -o survived_treeR.png")
           if(retCode>0):
               print("system command returning error: "+str(retCode))
           else:
               display(Image("survived_treeR.png"))
```

```
system command returning error: 1
```

```
In [124]:  nportance of features in the tree building ( The importance of a feature is computed as the
           ormalized) total reduction of the criterion brought by that feature. It is also known as the Gini importance )

           nt (pd.DataFrame(dTreeR.feature_importances_, columns = ["Imp"], index = X_train.columns).sort_values('Imp',ascending=False))
```

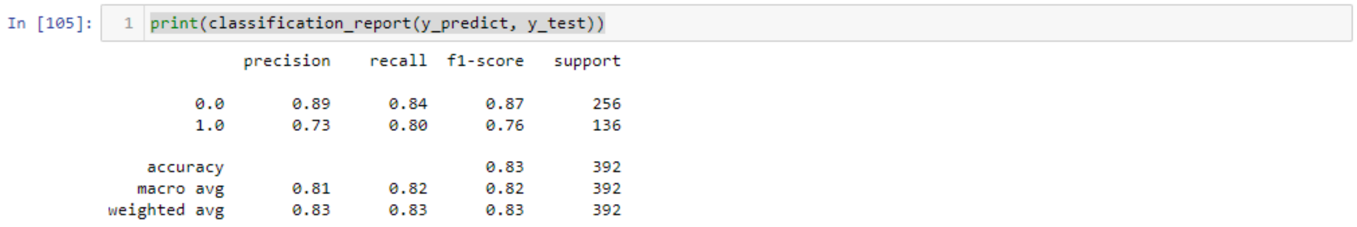
```
               Imp
sex       0.615953
pclass    0.237507
age       0.066222
fare      0.043517
sibsp     0.036801
parch     0.000000
embarked  0.000000
```

We then regularised the data by pruning giving a max depth of 3. We now see that scores have improved for the test set with 82.9%.



```
In [104]:
print(dTreeR.score(X_test , y_test))
y_predict = dTreeR.predict(X_test)

cm=metrics.confusion_matrix(y_test, y_predict, labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in ["0.0","1.0"]],
                  columns = [i for i in ["0.0","1.0"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True ,fmt='g')

0.8290816326530612
Out[104]: <AxesSubplot:>
```

Above is the confusion matrix for the model built.



```
In [105]: print(classification_report(y_predict, y_test))

              precision    recall  f1-score   support

         0.0       0.89      0.84      0.87       256
         1.0       0.73      0.80      0.76       136

    accuracy                           0.83       392
   macro avg       0.81      0.82      0.82       392
weighted avg       0.83      0.83      0.83       392
```

Above we see a precision for survived is 73 and recall is 80, with an accuracy of 83.

The model looks good.

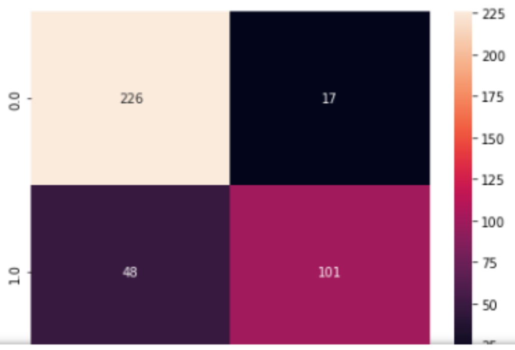
Let us try to build more models and see.

Ensemble Learning - GradientBoost

```
In [112]:
from sklearn.ensemble import GradientBoostingClassifier
gbcl = GradientBoostingClassifier(n_estimators = 40,random_state=1)
gbcl = gbcl.fit(X_train, y_train)

```

```
In [113]:
y_predict = gbcl.predict(X_test)
print(gbcl.score(X_test, y_test))
cm=metrics.confusion_matrix(y_test, y_predict,labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in ["0.0","1.0"]],
                  columns = [i for i in ["0.0","1.0"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True ,fmt='g')
```

```
0.8341836734693877
Out[113]: <AxesSubplot:>
```



We used ensemble learning with gradient boost to give us better results with a score of 83.4%.

```
In [114]:
print(classification_report(y_predict, y_test))
```

```
              precision    recall  f1-score   support

         0.0       0.93      0.82      0.87       274
         1.0       0.68      0.86      0.76       118

    accuracy                           0.83       392
   macro avg       0.80      0.84      0.82       392
weighted avg       0.85      0.83      0.84       392
```

In gradient boost we get a precision of survived of only 63 and recall is 86. But we need a much-balanced model.

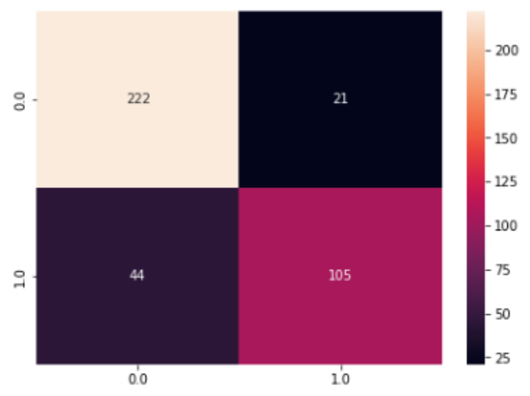
Ensemble RandomForest Classifier

In [115]:
```python
from sklearn.ensemble import RandomForestClassifier
rfcl = RandomForestClassifier(n_estimators = 100, random_state=1,max_features=4, max_depth= 10)
rfcl = rfcl.fit(X_train, y_train)

```

In [116]:
```python
y_predict = rfcl.predict(X_test)
print(rfcl.score(X_test, y_test))
cm=metrics.confusion_matrix(y_test, y_predict,labels=[0, 1])

df_cm = pd.DataFrame(cm, index = [i for i in ["0.0","1.0"]],
                  columns = [i for i in ["0.0","1.0"]])
plt.figure(figsize = (7,5))
sns.heatmap(df_cm, annot=True ,fmt='g')
```

```
0.8341836734693877
```

Out[116]:
```
<AxesSubplot:>
```



We deployed random forest on the test set and we got a score of 83.4%.

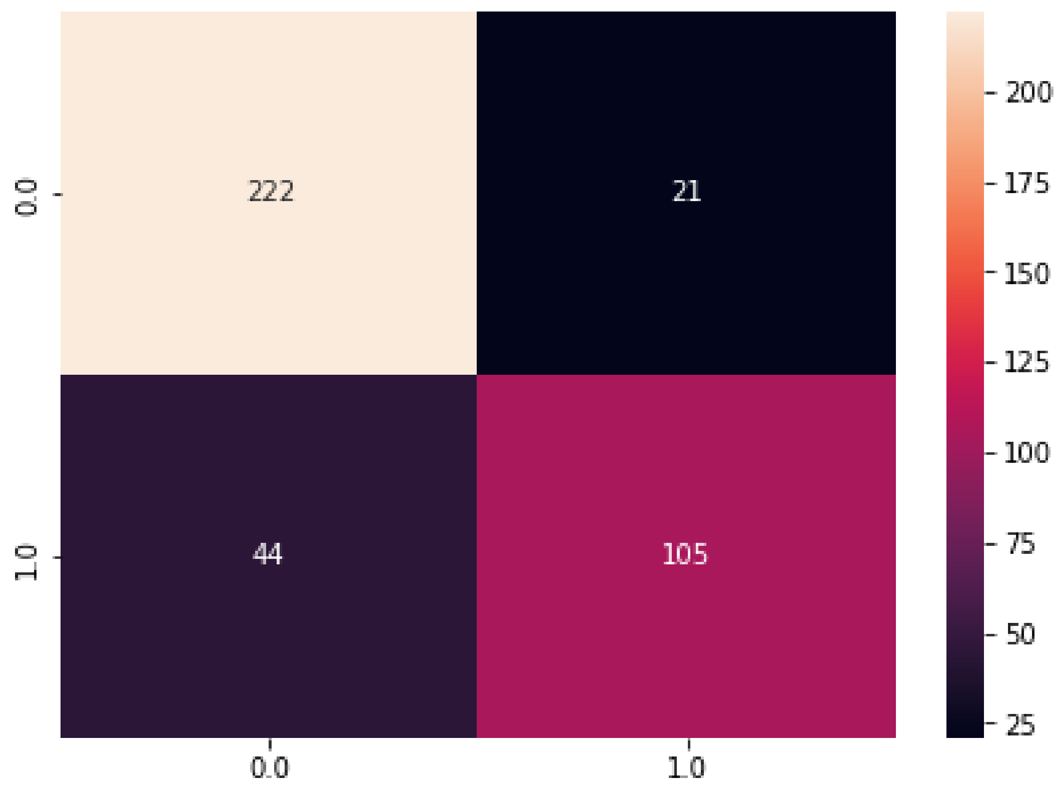
In [118]:
```python
print(classification_report(y_predict, y_test))
```

```
              precision    recall  f1-score   support

         0.0       0.91      0.83      0.87       266
         1.0       0.70      0.83      0.76       126

    accuracy                           0.83       392
   macro avg       0.81      0.83      0.82       392
weighted avg       0.85      0.83      0.84       392
```

Here we find a precision of 70 and recall of 83.

3. **What is the performance of the build model/ models? You need to provide discussion and justification of how the model is performing (discuss different matrices, accuracy confusion matrix) based on the selected dataset.**

   We may say that the performance of random forest the best compared to other models.



With an accuracy score of 83.4. It has predicted correctly that 222 passengers did not survive, who actually could not survive and predicted 105 passengers survived, who actually survived. But it also has predicted incorrectly 21 people survive who actually did not survived and 44 people did not survive but actually survived.

4. **You need to compare the performance of the models and provide justifications which mode is performing better and why.**

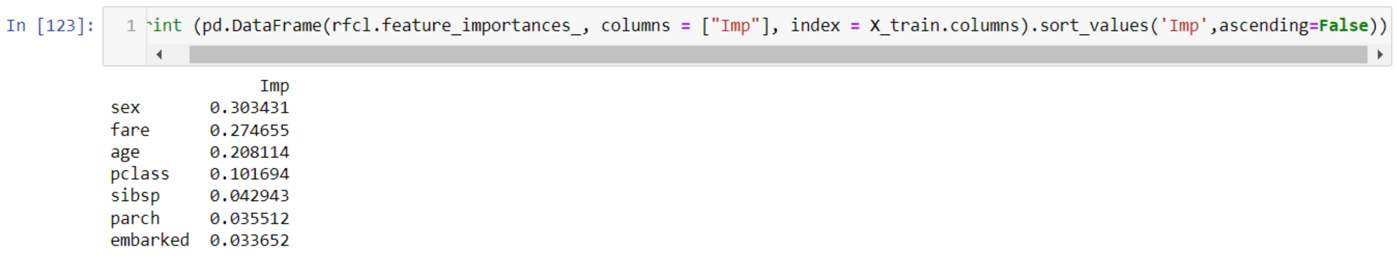
Decision tree using gradient boost is the best model that we were able to form as it gives us the best accuracy of 83.1.

We will also be more interested in recall to be higher as in a real case scenario of a disaster we would want to know how many predicted survived have actually survived and efforts will be to save as many as we can.

```
In [118]:  print(classification_report(y_predict, y_test))
```

```
              precision    recall  f1-score   support

         0.0       0.91      0.83      0.87       266
         1.0       0.70      0.83      0.76       126

    accuracy                           0.83       392
   macro avg       0.81      0.83      0.82       392
weighted avg       0.85      0.83      0.84       392
```

This model again gives us the best recall when compared to other models that we have developed.



```
In [123]: int (pd.DataFrame(rfcl.feature_importances_, columns = ["Imp"], index = X_train.columns).sort_values('Imp',ascending=False))
```

```
               Imp
sex       0.303431
fare      0.274655
age       0.208114
pclass    0.101694
sibsp     0.042943
parch     0.035512
embarked  0.033652
```

The most important features for prediction according to random forest are in above table in descending order. The most important feature is sex, followed by fare and then age. These three features collectively are almost 77% important for predictions.