University of Massachusetts Boston
MSIS685 – Big data Analytics – Spring 2021
Programming Assignment II: Titanic Data analysis using Pandas, Numpy, Tensorflow
Submitted by - Alisha Warke

1. Use either the Google Colab environment or install relevant libraries such as numpy, pandas, matplotlib, seaborn, and tensorflow within a Notebook Python environment such as Anaconda.

When running the version check the Tensorflow version should be 2.0 or above ..for the other libraries such as pandas, matplotlib, seaborn, numpy.. etc

```
[ ] import tensorflow as tf
    print(tf.__version__)

    2.4.1
```

```
import pandas as pd
import numpy as np

# Make numpy values easier to read.
np.set_printoptions(precision=3, suppress=True)

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.layers.experimental import preprocessing

# machine learning
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
# Algorithms
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

2. Load the data for Titanic {either load the full data and then perform a train_test_split or load a fraction of data for training, and fraction for testing.

>Loaded datasets train.csv and test.csv for training the model and testing as “train” and “test” respectively.

```
BD Prog2 Alisha.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

[60] # Reading data to Dataframes

     train = pd.read_csv("train.csv")
     test = pd.read_csv("test.csv")
```

3. Do some exploratory analysis of the data with:
... info()
...describe()

>information of the attributes and descriptions of all attributes in the “train” data

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   passengerId  891 non-null    int64
 1   survived     891 non-null    int64
 2   pclass       891 non-null    int64
 3   name         891 non-null    object
 4   sex          891 non-null    object
 5   age          714 non-null    float64
 6   sibsp        891 non-null    int64
 7   parch        891 non-null    int64
 8   ticket       891 non-null    object
 9   fare         891 non-null    float64
 10  cabin        204 non-null    object
 11  embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
train.describe()
```

	passengerId	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

>information of the attributes and descriptions of all attributes in the "test" data

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   passengerId  418 non-null    int64
 1   pclass       418 non-null    int64
 2   name         418 non-null    object
 3   sex          418 non-null    object
 4   age          332 non-null    float64
 5   sibsp        418 non-null    int64
 6   parch        418 non-null    int64
 7   ticket       418 non-null    object
 8   fare         417 non-null    float64
 9   cabin        91 non-null     object
 10  embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

```
test.describe()
```

	passengerId	pclass	age	sibsp	parch	fare
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200

And a few graphs outlining the nature of the attributes in the dataset.

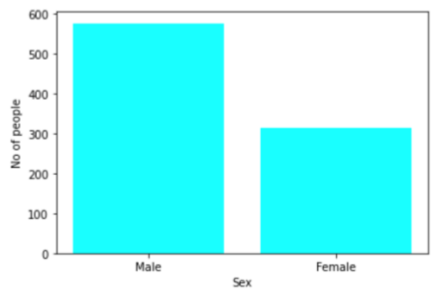
Graph (3.1): Plotting Survival on passengers' gender

>counted number of males and females who survived and plotted a bar chart

```
[238] #count number of males and females
      males = len(train[train['sex'] == 'male'])
      females = len(train[train['sex'] == 'female'])
      males, females

      (577, 314)
```

```
#Plotting survival on sex
sex = ['Male','Female']
values = [577,314]
# Change the bar colors here
plt.bar(sex, values, color=['cyan'])
plt.xlabel("Sex")
plt.ylabel("No of people")
plt.show()
```



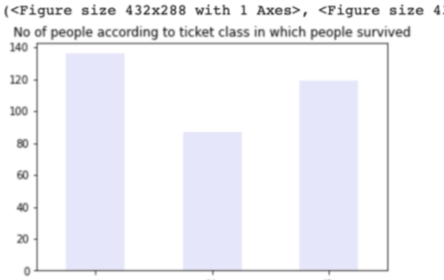
Graph (3.2): Plotting survival on pclass (ticket class)

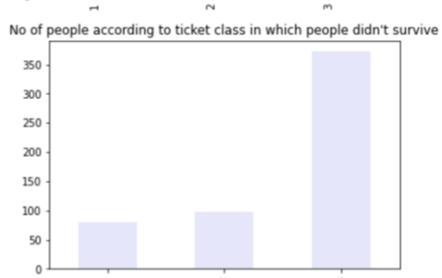
>counted number of passengers by each ticket class and plotted bar charts for those who survived and those who did not survive

```
#Plotting survival on pclass
plt.figure(1)
train.loc[train['survived'] == 1, 'pclass'].value_counts().sort_index().plot.bar(color=['lavender'])
plt.title('No of people according to ticket class in which people survived')

plt.figure(2)
train.loc[train['survived'] == 0, 'pclass'].value_counts().sort_index().plot.bar(color=['lavender'])
plt.title('No of people according to ticket class in which people didn\'t survive')

(<Figure size 432x288 with 1 Axes>, <Figure size 432x288 with 1 Axes>)
```



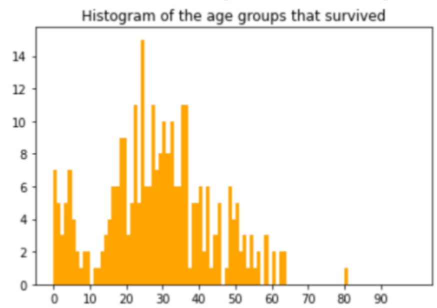


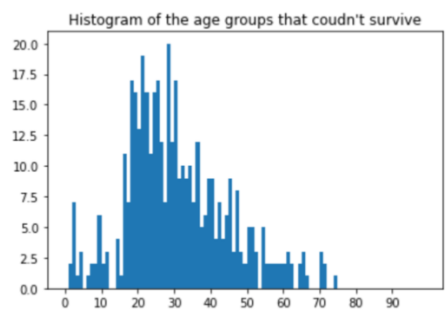
Graph (3.3): Plotting survival on age groups

>Plotted Histogram for who survived and did not survive on basis of their age groups

```
#plotting survival on age groups
plt.figure(3)
age  = train.loc[train.survived == 1, 'age']
plt.title('Histogram of the age groups that survived')
plt.hist(age, np.arange(0,100,1),color=['orange'])
plt.xticks(np.arange(0,100,10))


plt.figure(4)
age  = train.loc[train.survived == 0, 'age']
plt.title('Histogram of the age groups that coudn\'t survive')
plt.hist(age, np.arange(0,100,1))
plt.xticks(np.arange(0,100,10))
```





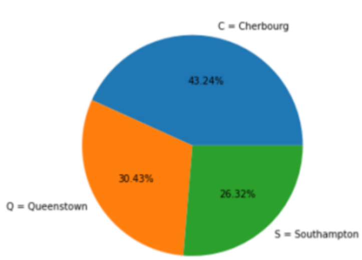
Graph (3.4): Categorizing Survival on port of embarkation

>Plotted Pie chart showing survival rate of each port of embarkation

```
[254] #count embarkation at each port
      train[["embarked", "survived"]].groupby(['embarked'], as_index=False).mean().sort_values(by='survived', ascending=False)
```

	embarked	survived
0	C	0.553571
1	Q	0.389610
2	S	0.336957

```
#pie chart for port of embarkation
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
l = ['C = Cherbourg', 'Q = Queenstown', 'S = Southampton']
s = [0.553571,0.389610,0.336957]
ax.pie(s, labels = l,autopct='%1.2f%%')
plt.show()
```



4. Do some data cleaning to eliminate redundant columns if necessary {they would highly extend computing time if included in latest models}

>dropped all those columns which are not making any difference to my dataset or they cannot be changed to numbers. Removed columns ‘name’, ’ticket’ as they are mostly unique values and would not help in training data. Also, removed ‘fare’, ‘cabin’, ‘embarked’ as they don’t really help to predict the survival rate

```
#Removing unused columns
data_train = train.drop(['name','ticket','fare','cabin','embarked'],axis =1)
data_test = test.drop(['name','ticket','fare','cabin','embarked'],axis =1)
```

```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   passengerId  891 non-null    int64
 1   survived     891 non-null    int64
 2   pclass       891 non-null    int64
 3   sex          891 non-null    object
 4   age          714 non-null    float64
 5   sibsp        891 non-null    int64
 6   parch        891 non-null    int64
dtypes: float64(1), int64(5), object(1)
memory usage: 48.9+ KB
```

```
[220] data_train= data_train.dropna()
```

```
[221] data_train.isnull().sum()
```

```
passengerId    0
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
dtype: int64
```

> Convert categorical features into numeric

```
[222] # Convert categorical  features into numeric
      data_train['sex'] = data_train['sex'].map( {'female': 1, 'male': 0} ).astype(int)
```

```
[223] # Convert categorical  features into numeric
      data_test['sex'] = data_test['sex'].map( {'female': 1, 'male': 0} ).astype(int)
```

```
[225] data_train.head()
```

	passengerId	survived	pclass	sex	age	sibsp	parch
0	1	0	3	0	22.0	1	0
1	2	1	1	1	38.0	1	0
2	3	1	3	1	26.0	0	0
3	4	1	1	1	35.0	1	0
4	5	0	3	0	35.0	0	0

```
data_test.head()
```

	passengerId	pclass	sex	age	sibsp	parch
0	892	3	0	34.5	0	0
1	893	3	1	47.0	1	0
2	894	2	0	62.0	0	0
3	895	3	0	27.0	0	0
4	896	3	1	22.0	1	1

> Arranging columns of the dataframe keeping target= survived in the end.

```
[227] # arranging data
      data_train = data_train[['passengerId','sex','age','sibsp','parch','pclass','survived']]
      data_test = data_test[['passengerId','sex','age','sibsp','parch','pclass']]
```

```
[228] data_train.head()
```

	passengerId	sex	age	sibsp	parch	pclass	survived
0	1	0	22.0	1	0	3	0
1	2	1	38.0	1	0	1	1
2	3	1	26.0	0	0	3	1
3	4	1	35.0	1	0	1	1
4	5	0	35.0	0	0	3	0

```
data_test.head()
```

	passengerId	sex	age	sibsp	parch	pclass
0	892	0	34.5	0	0	3
1	893	1	47.0	1	0	3
2	894	0	62.0	0	0	2
3	895	0	27.0	0	0	3
4	896	1	22.0	1	1	3

5. Try a few different models {e.g. Logistics, Random Forest, Decision tree, Classification} to analyze the data and run them. Which one would have best predictions

>split the data into inputs and outputs (Will be training our model with first 5 columns as input and 6th column as output) and using Standard Scalar to normalize data.

```
[230] #Importing LabelEncoder from Sklearn
      from sklearn.preprocessing import LabelEncoder
      label_encoder_sex = LabelEncoder()
```

```
[231] X_train = data_train.iloc[:,0:6]  # // Inputs
      y_train = data_train.iloc[:,6]    # //Output (Survived)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(data_test)
```

MODEL 1: Keras Classification Model

>importing Sequential and Dense from TensorFlow, created object of the Sequential class added layers to the classifier model, built a neural network with 3 layers, we have 5 inputs and 1 output. After adding layers compiled the data.

Using Keras Classifier Model

```
[125] import keras
      from keras.models import Sequential
      from keras.layers import Dense
```

```
def build_classifier():
    classifier = Sequential()
  #Input layer with 5 inputs neurons
    classifier.add(Dense(3, kernel_initializer='uniform', activation = 'relu'))
  #Hidden layer
    classifier.add(Dense(2, kernel_initializer='uniform', activation = 'relu'))
  #output layer with 1 output neuron which will predict 1 or 0
    classifier.add(Dense(1, kernel_initializer='uniform', activation = 'sigmoid'))
    #compile the model
    classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    return classifier
classifier = KerasClassifier(build_fn = build_classifier)
```

> Provided the training data to our model so that it can learn, using epoch = 200, it is perfect in this case so that the model do not overfit

```
classifier.fit(X_train, y_train, batch_size = 10, epochs = 200)
```

```
Epoch 1/200
72/72 [==============================] - 1s 766us/step - loss: 0.6921 - accuracy: 0.6019
Epoch 2/200
72/72 [==============================] - 0s 959us/step - loss: 0.6875 - accuracy: 0.5861
Epoch 3/200
72/72 [==============================] - 0s 946us/step - loss: 0.6763 - accuracy: 0.5964
Epoch 4/200
72/72 [==============================] - 0s 811us/step - loss: 0.6556 - accuracy: 0.5966
Epoch 5/200
72/72 [==============================] - 0s 904us/step - loss: 0.6351 - accuracy: 0.5821
Epoch 6/200
72/72 [==============================] - 0s 884us/step - loss: 0.6047 - accuracy: 0.5940
Epoch 7/200
72/72 [==============================] - 0s 902us/step - loss: 0.5925 - accuracy: 0.5883
Epoch 8/200
72/72 [==============================] - 0s 960us/step - loss: 0.5856 - accuracy: 0.5901
Epoch 9/200
72/72 [==============================] - 0s 939us/step - loss: 0.5689 - accuracy: 0.5950
Epoch 10/200
72/72 [==============================] - 0s 831us/step - loss: 0.5594 - accuracy: 0.6357
72/72 [==============================] - 0s 1ms/step - loss: 0.3896 - accuracy: 0.8415
Epoch 188/200
72/72 [==============================] - 0s 832us/step - loss: 0.4131 - accuracy: 0.8102
Epoch 189/200
72/72 [==============================] - 0s 931us/step - loss: 0.4235 - accuracy: 0.8279
Epoch 190/200
72/72 [==============================] - 0s 940us/step - loss: 0.3979 - accuracy: 0.8406
Epoch 191/200
72/72 [==============================] - 0s 941us/step - loss: 0.3747 - accuracy: 0.8448
Epoch 192/200
72/72 [==============================] - 0s 893us/step - loss: 0.4149 - accuracy: 0.8250
Epoch 193/200
72/72 [==============================] - 0s 877us/step - loss: 0.3638 - accuracy: 0.8548
Epoch 194/200
72/72 [==============================] - 0s 1ms/step - loss: 0.3852 - accuracy: 0.8442
Epoch 195/200
72/72 [==============================] - 0s 968us/step - loss: 0.3711 - accuracy: 0.8448
Epoch 196/200
72/72 [==============================] - 0s 867us/step - loss: 0.3803 - accuracy: 0.8421
Epoch 197/200
72/72 [==============================] - 0s 960us/step - loss: 0.4053 - accuracy: 0.8301
Epoch 198/200
72/72 [==============================] - 0s 951us/step - loss: 0.4090 - accuracy: 0.8236
Epoch 199/200
72/72 [==============================] - 0s 1ms/step - loss: 0.3945 - accuracy: 0.8320
Epoch 200/200
72/72 [==============================] - 0s 913us/step - loss: 0.4117 - accuracy: 0.8275
<tensorflow.python.keras.callbacks.History at 0x7fe748348750>
```

```
#getting predictions of test data
prediction = classifier.predict(X_test).tolist()
# list to series
se = pd.Series(prediction)
```

>added a new column ‘check’ (Which are the predictions by our model) to a new dataset test_result.csv which has columns passengerID and survived.

```
[55] # new data results with passengerid and survival
data_check = pd.read_csv("test_result.csv")
```

```
# creating new column of predictions in data_check dataframe

data_check['check'] = se
data_check['check'] = data_check['check'].str.get(0)
data_check.head(25)
```

	passengerId	survived	check
0	892	0	0
1	893	1	0
2	894	0	0
3	895	0	0
4	896	1	0
5	897	0	0
6	898	1	1
7	899	0	0
8	900	1	1
9	901	0	0
10	902	0	0
11	903	0	0
12	904	1	1
13	905	0	0
14	906	1	1
15	907	1	1

>checking the accuracy of Keras model
Accuracy = 85.16%

```
[52] #check accuracy
     match = 0
     nomatch = 0
     for val in data_check.values:
       if val[1] == val[2]:
             match = match +1
       else:
             nomatch = nomatch +1
```

```
[53] match, nomatch

     (356, 62)
```

```
[54] #calculating accuracy
     match*100/(match+nomatch)

     85.16746411483254
```

MODEL 2: Logistic Regression Model

Using Logistic Regression

```
[193] #Training Testing and Spliting the model
      from sklearn.model_selection import train_test_split
      xtrain, xtest, ytrain, ytest = train_test_split(X_train,y_train,test_size=0.3,random_state=0)
```

```
[194] #scaling the data as range if Age and Survival is different
      from sklearn.preprocessing import StandardScaler
      sc_x = StandardScaler()
      xtrain = sc_x.fit_transform(xtrain)
      xtest = sc_x.transform(xtest)

      print (xtrain[0:10, :])

      [[-0.58800883  1.32213142 -0.38144111 -0.53485572 -0.48461856  0.92665396]
       [-0.85144787 -0.75635446 -0.7311205  -0.53485572 -0.48461856 -0.30301232]
       [ 0.15905711  1.32213142  2.34605818 -0.53485572 -0.48461856  0.92665396]
       [-1.70074387  1.32213142 -1.78015869  0.5379296   0.65816439  0.92665396]
       [ 0.72525445  1.32213142 -0.59124874 -0.53485572 -0.48461856 -1.5326786 ]
       [ 0.61909245  1.32213142 -0.38144111  1.61071492  0.65816439 -0.30301232]
       [ 1.07519586  1.32213142  0.5976612  -0.53485572 -0.48461856 -1.5326786 ]
       [-1.37832594 -0.75635446  1.15714824  0.5379296  -0.48461856 -1.5326786 ]
       [ 1.54309505  1.32213142  0.66759708  0.5379296   0.65816439 -1.5326786 ]
       [ 1.39368186  1.32213142  0.10811005 -0.53485572 -0.48461856  0.92665396]]
```

```
[195] #training the data
      from sklearn.linear_model import LogisticRegression
      classifier = LogisticRegression(random_state = 0)
      classifier.fit(xtrain, ytrain)

      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=0, solver='lbfgs', tol=0.0001, verbose=0,
                         warm_start=False)
```

```
[196] #prediction on testing data
      y_pred = classifier.predict(xtest)
```

```
[197] #test by confusion matrix

      from sklearn.metrics import confusion_matrix
      cm = confusion_matrix(ytest, y_pred)

      print ("Confusion Matrix : \n", cm)

      Confusion Matrix :
       [[77 10]
       [19 44]]
```

Out of 150 :

TruePostive + TrueNegative = 77 + 44 =121

FalsePositive + FalseNegative = 19 + 10 = 29

```
[198] #Performance measure – Accuracy

      from sklearn.metrics import accuracy_score
      print ("Accuracy : ", accuracy_score(ytest, y_pred))

      Accuracy :  0.8066666666666666
```

>Accuracy of the Logistic Regression model = 80.66%

MODEL 3: Decision Tree Model

Using Decision Tree

```
[202] #Using Decision Tree
      from sklearn.tree import DecisionTreeClassifier
      model_dtree = DecisionTreeClassifier(criterion='entropy',random_state=7)
      model_dtree.fit(X_train,y_train)
      y_pred_dtree = model_dtree.predict(X_test)

      from sklearn.metrics import accuracy_score
      print("Accuracy Score:",accuracy_score(y_test,y_pred_dtree))

      Accuracy Score: 0.772093023255814
```

```
[204] from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
      conf_mat = confusion_matrix(y_test,y_pred_dtree)
      print(conf_mat)
      print(classification_report(y_test,y_pred_dtree))

      [[100  25]
       [ 24  66]]
                    precision    recall  f1-score   support

                 0       0.81      0.80      0.80       125
                 1       0.73      0.73      0.73        90

          accuracy                           0.77       215
         macro avg       0.77      0.77      0.77       215
      weighted avg       0.77      0.77      0.77       215
```

>Accuracy of the Decision Tree model = 77.2%

MODEL 4: Random Forest Model

Using Random Forest

```
[234] #Import Random Forest Model
      from sklearn.ensemble import RandomForestClassifier
      #xtrain, xtest, ytrain, ytest = train_test_split(X_train,y_train,test_size=0.3,random_state=0)
      #Create a Gaussian Classifier
      model_RF=RandomForestClassifier(n_estimators=100)

      #Train the model using the training sets y_pred=clf.predict(X_test)
      model_RF.fit(xtrain,ytrain)

      y_pred=model_RF.predict(xtest)
```

```
[235] # accuracy calculation
      from sklearn import metrics
      # Model Accuracy, how often is the classifier correct?
      print("Accuracy:",metrics.accuracy_score(ytest, y_pred))

      Accuracy: 0.7933333333333333
```

>Accuracy of the Random Forest model = 79.33%

To Summarize

Model>	Keras Classification Model	Logistic Regression Model	Decision Tree model	Random Forest model
Accuracy (in %)	85.16	80.66	77.2	79.33

>The Keras model shows the highest accuracy, and the Decision tree model shows least.

If we increase epochs in the Keras model we get a higher accuracy.

6. Do a few visualizations with the finalized data {i.e. after deleting extra attributes, and treating missing values.

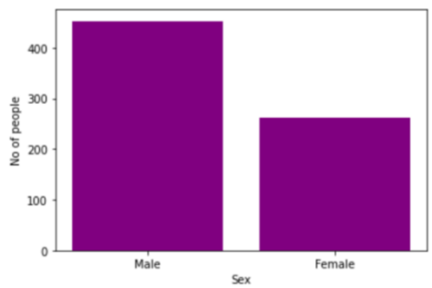
Graph (6.1): Plotting on cleaned data Survival on passengers' gender

Visualizations on cleaned data

```python
#count number of males and females
males = len(data_train[data_train['sex'] == 0 ])
females = len(data_train[data_train['sex']==1])
males, females
```

```
(453, 261)
```

```python
#Plotting survival on sex
sex = ['Male','Female']
values = [453, 261]
# Change the bar colors here
plt.bar(sex, values, color=['purple'])
plt.xlabel("Sex")
plt.ylabel("No of people")
plt.show()
```

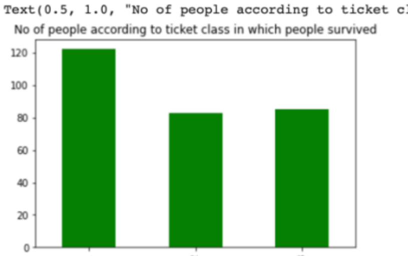


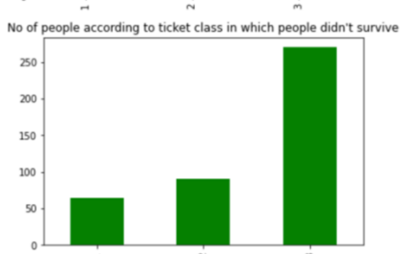
Graph (6.2): Plotting on cleaned data survival on pclass (ticket class)

```python
#Plotting with clean data survival on pclass
plt.figure(1)
data_train.loc[data_train['survived'] == 1, 'pclass'].value_counts().sort_index().plot.bar(color=['green'])
plt.title('No of people according to ticket class in which people survived')

plt.figure(2)
data_train.loc[data_train['survived'] == 0, 'pclass'].value_counts().sort_index().plot.bar(color=['green'])
plt.title('No of people according to ticket class in which people didn\'t survive')
```

```
Text(0.5, 1.0, "No of people according to ticket class in which people didn't survive")
```

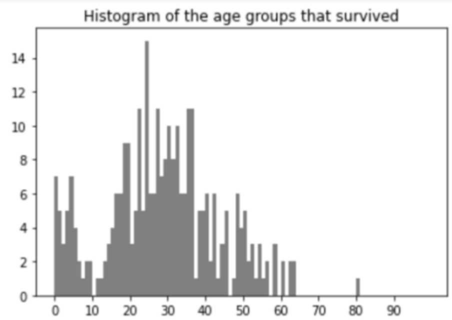


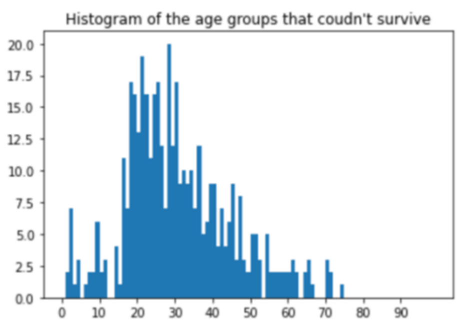


Graph (6.3): Plotting on cleaned data survival on age groups

```
#plotting with clean data survival on age groups
plt.figure(3)
age  = data_train.loc[train.survived == 1, 'age']
plt.title('Histogram of the age groups that survived')
plt.hist(age, np.arange(0,100,1),color=['gray'])
plt.xticks(np.arange(0,100,10))


plt.figure(4)
age  = data_train.loc[train.survived == 0, 'age']
plt.title('Histogram of the age groups that coudn\'t survive')
plt.hist(age, np.arange(0,100,1))
plt.xticks(np.arange(0,100,10))
```





7. Would you {and an assumed passenger e.g Woman / 30 year old} survived the trip?

>checked the survival status of females whose age are 30 or 24
If we look at the returned data, we can see the maximum of them have survived

9 out of 11 females have survived who are 30 years old.
14 out of 16 females have survived who are 24 years old.

According to these predictions chances of my survival are 87.5%

And the chances of survival of a 30-year-old female are 81.1%

```
[34] ans = data_train[(data_train.sex == 1) & (data_train.age == 30)]
     ans2 =data_train[(data_train.sex == 1) & (data_train.age == 24)]
     ans, ans2

(      passengerId  sex   age  sibsp  parch  pclass  survived
 79            80    1  30.0      0      0       3         1
 257          258    1  30.0      0      0       1         1
 309          310    1  30.0      0      0       1         1
 322          323    1  30.0      0      0       2         1
 520          521    1  30.0      0      0       1         1
 534          535    1  30.0      0      0       3         0
 537          538    1  30.0      0      0       1         1
 726          727    1  30.0      3      0       2         1
 747          748    1  30.0      0      0       2         1
 799          800    1  30.0      1      1       3         0
 842          843    1  30.0      0      0       1         1,
      passengerId  sex   age  sibsp  parch  pclass  survived
 142          143    1  24.0      1      0       3         1
 199          200    1  24.0      0      0       2         0
 247          248    1  24.0      0      2       2         1
 293          294    1  24.0      0      0       3         0
 310          311    1  24.0      0      0       1         1
 316          317    1  24.0      1      0       2         1
 341          342    1  24.0      3      2       1         1
 345          346    1  24.0      0      0       2         1
 369          370    1  24.0      0      0       1         1
 394          395    1  24.0      0      2       3         1
 437          438    1  24.0      2      3       2         1
 600          601    1  24.0      2      1       2         1
 615          616    1  24.0      1      2       2         1
 641          642    1  24.0      0      0       1         1
 710          711    1  24.0      0      0       1         1
 858          859    1  24.0      0      3       3         1)
```

```
[41] #survival of female, age=30
     ans[["survived"]].sum(), ans[["survived"]].count()

     (survived    9
      dtype: int64, survived    11
      dtype: int64)
```

```
#survival of female, age=24
ans2[["survived"]].sum(), ans2[["survived"]].count()

(survived    14
 dtype: int64, survived    16
 dtype: int64)
```

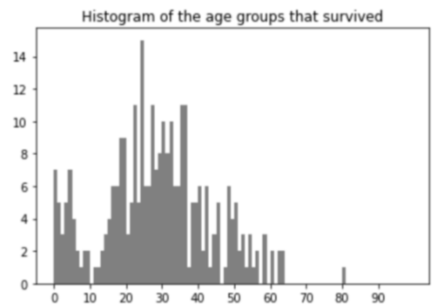
8. Short write-up [1 page] of you findings in layman terms... who survives.. who dies Which combination of attributes is best?

- Young or old people

>

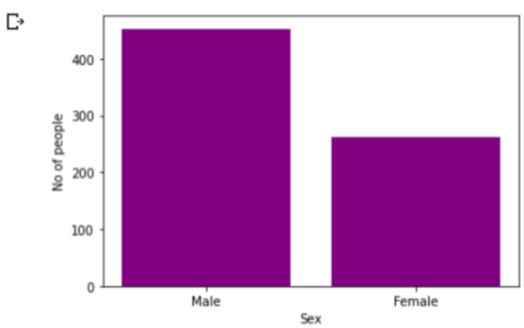
Referring to Graph (6.3), if we look at the first (orange) histogram that shows, we can observe that the survival chances are higher in the age group around 15 to 40 years.

It is quite evident that the age groups outside this group have very less chances to survive Moreover, Age 24 has a highest survival rate.



- Man or Female

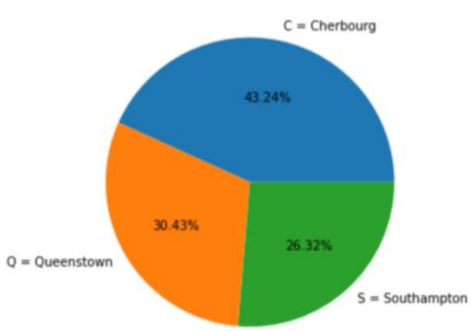
>From Graph (6.1) we can see that the survival rate of males is more than females. After cleaning the data, the number of men is 453 and that of females is 261.



- Boarding at one of the three embarquement/boarding places

>From Graph (3.4) we can see that the range of population embarking the ship from each location is 26% to 43% approximately, which according to me is not very evident to predict the survival rate.

If we think practically the place where a person boards does not make any difference when the accident occurred on the titanic. Had there been wide differences in the percentage of population from different embarkation, it would be essential for the prediction.



- Rich or poor {correlates to decks}

  >From Graph (6.2) we can see that

  People in first class had the highest survival rate as a group vs. those in second or third class.

  People in 3rd class has the highest death rate. That part maybe more prone to the area on the ship that hit the iceberg.

  The 1st class are generally more secured and also during the rescue operation they are given priority to get on life boat.