IMPORTING LIBRARIES & DATASET

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [2]:
```
df = pd.read_csv('telco.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLine
0	7590-VHVEG	Female	0	Yes	No	1	No	No phor servic
1	5575-GNVDE	Male	0	No	No	34	Yes	N
2	3668-QPYBK	Male	0	No	No	2	Yes	N
3	7795-CFOCW	Male	0	No	No	45	No	No phor servic
4	9237-HQITU	Female	0	No	No	2	Yes	N

5 rows × 21 columns

EXPLORATORY DATA ANALYSIS

In [4]:
```
print('Number of Rows:', df.shape[0])
print('Number of Columns:', df.shape[1])
print('Number of Missing Values: ', df.isnull().sum().sum())
print('Number of Unique: \n', df.nunique())
```

```
Number of Rows: 7043
Number of Columns: 21
Number of Missing Values:  0
Number of Unique:
 customerID          7043
gender                 2
SeniorCitizen          2
Partner                2
Dependents             2
tenure                73
PhoneService           2
MultipleLines          3
InternetService        3
OnlineSecurity         3
OnlineBackup           3
DeviceProtection       3
TechSupport            3
StreamingTV            3
StreamingMovies        3
Contract               3
PaperlessBilling       2
PaymentMethod          4
MonthlyCharges      1585
TotalCharges        6531
Churn                  2
dtype: int64
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

In [6]:
```
df = df[df['TotalCharges']!=" "]
```

In [7]:
```
df['TotalCharges'] = df['TotalCharges'].astype('float')
```

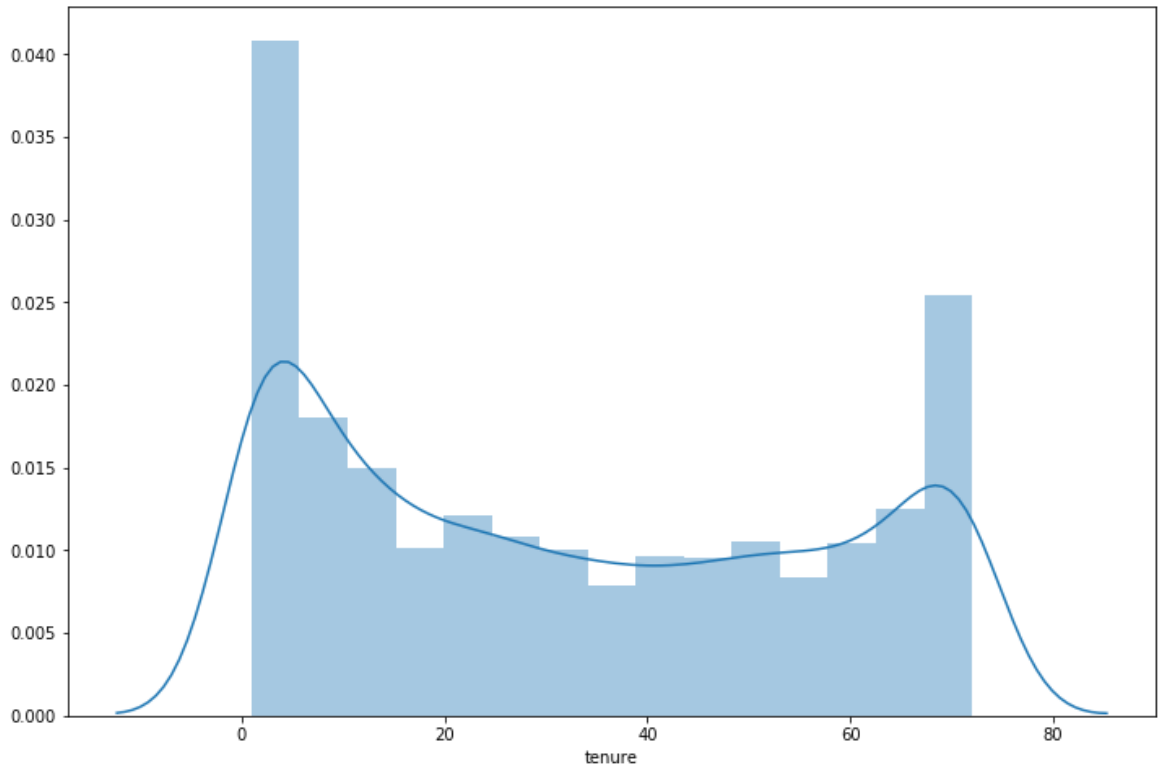
In [8]:
```
df.columns
```

Out[8]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

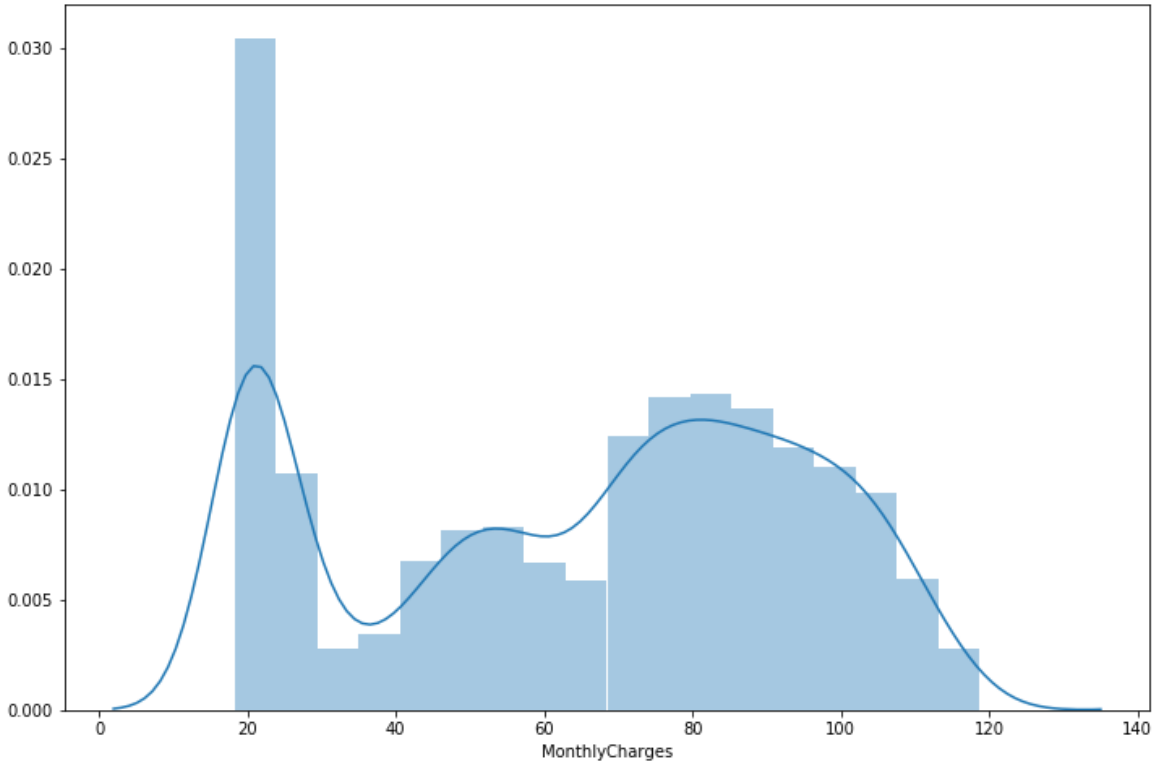
In [9]:
```
uni=[]
for i in df.columns:
    if df[i].nunique()<5:
        print(i, df[i].unique())
```

```
gender ['Female' 'Male']
SeniorCitizen [0 1]
Partner ['Yes' 'No']
Dependents ['No' 'Yes']
PhoneService ['No' 'Yes']
MultipleLines ['No phone service' 'No' 'Yes']
InternetService ['DSL' 'Fiber optic' 'No']
OnlineSecurity ['No' 'Yes' 'No internet service']
OnlineBackup ['Yes' 'No' 'No internet service']
DeviceProtection ['No' 'Yes' 'No internet service']
TechSupport ['No' 'Yes' 'No internet service']
StreamingTV ['No' 'Yes' 'No internet service']
StreamingMovies ['No' 'Yes' 'No internet service']
Contract ['Month-to-month' 'One year' 'Two year']
PaperlessBilling ['Yes' 'No']
PaymentMethod ['Electronic check' 'Mailed check' 'Bank transfer (automati
c)'
 'Credit card (automatic)']
Churn ['No' 'Yes']
```

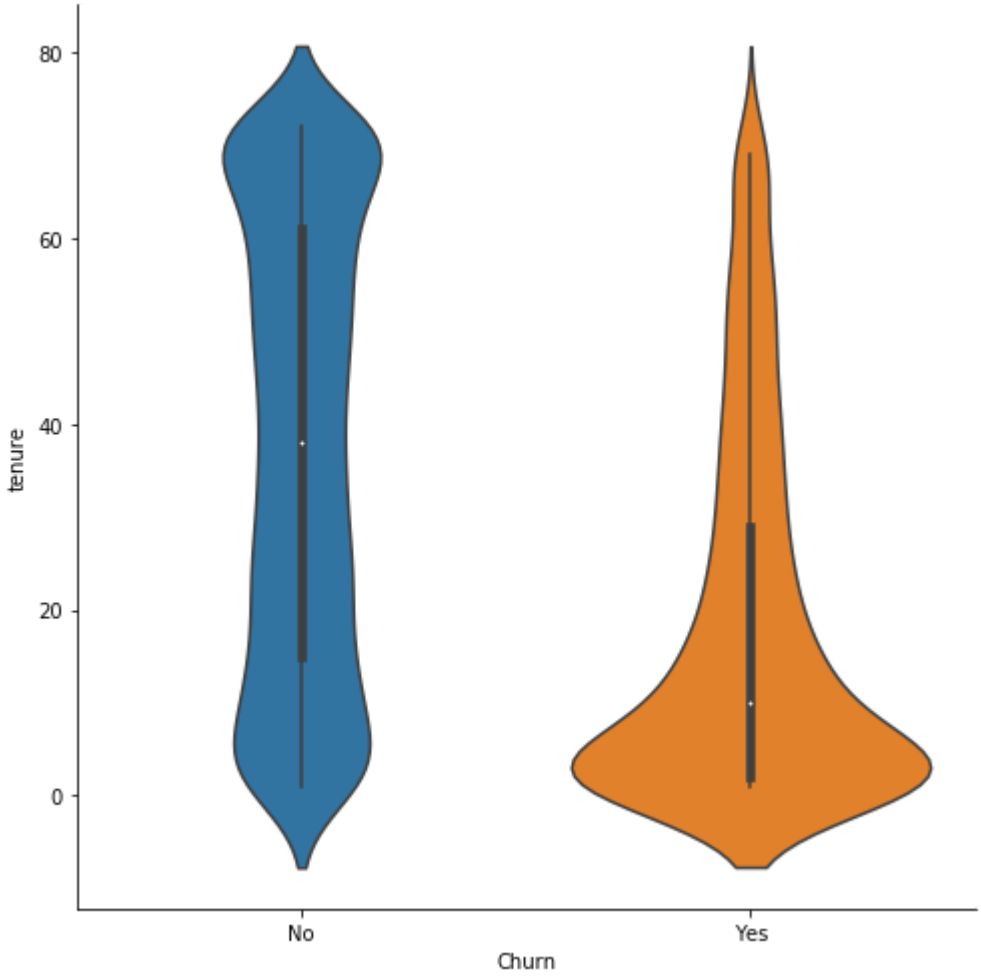
In [10]:
```
plt.figure(figsize=(12,8))
sns.distplot(df['tenure'])
plt.show()
```



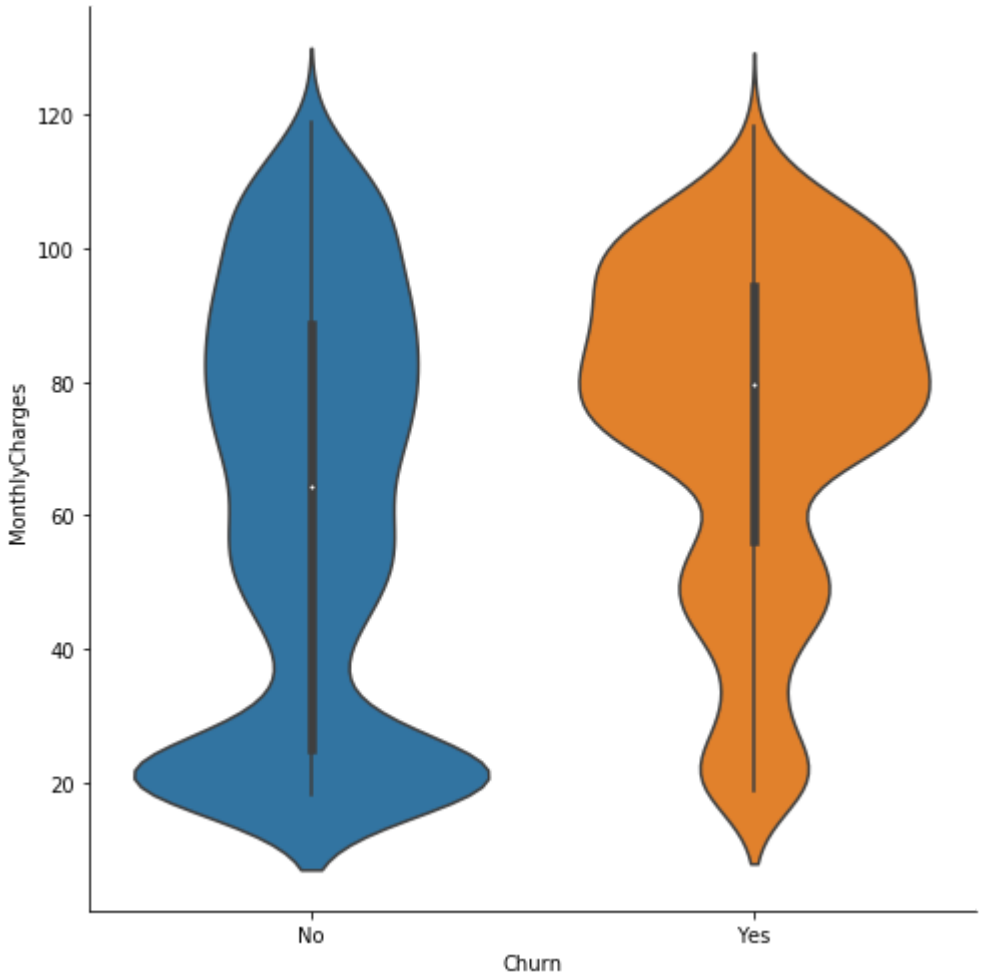
In [11]:
```
plt.figure(figsize=(12,8))
sns.distplot(df['MonthlyCharges'])
plt.show()
```



In [12]:
```
sns.catplot(data=df, x='Churn', y='tenure', kind='violin', height=7, aspect=1
plt.show()
```



In [13]:
```
sns.catplot(data=df, x='Churn', y='MonthlyCharges', kind='violin', height=7,
plt.show()
```



In [14]:
```
cat_col=[]
for i in df.columns:
    if df[i].nunique()<10:
        cat_col.append(i)
```

In [15]:
```
cat_df = df[cat_col]
```

In [16]:
```
cat_df
```

Out[16]:

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService
0	Female	0	Yes	No	No	No phone service	DSL
1	Male	0	No	No	Yes	No	DSL
2	Male	0	No	No	Yes	No	DSL
3	Male	0	No	No	No	No phone service	DSL
4	Female	0	No	No	Yes	No	Fiber optic
...	...	...	...	...	...	...	...
7038	Male	0	Yes	Yes	Yes	Yes	DSL
7039	Female	0	Yes	Yes	Yes	Yes	Fiber optic
7040	Female	0	Yes	Yes	No	No phone service	DSL
7041	Male	1	Yes	No	Yes	Yes	Fiber optic
7042	Male	0	No	No	Yes	No	Fiber optic

7032 rows × 17 columns

In [17]:
```
temp = pd.concat([pd.crosstab(cat_df[x], cat_df['Churn']) for x in cat_df.col
```

In [18]:
```
temp.columns = ['attribute1', 'attribute2', 'No', 'Yes']
```

In [19]:
```
temp['churn_rate'] = temp['Yes']/(temp['Yes']+temp['No'])
```

In [20]:
```
temp
```

Out[20]:

	attribute1	attribute2	No	Yes	churn_rate
0	gender	Female	2544	939	0.269595
1	gender	Male	2619	930	0.262046
2	SeniorCitizen	0	4497	1393	0.236503
3	SeniorCitizen	1	666	476	0.416813
4	Partner	No	2439	1200	0.329761
5	Partner	Yes	2724	669	0.197171
6	Dependents	No	3390	1543	0.312791
7	Dependents	Yes	1773	326	0.155312
8	PhoneService	No	510	170	0.250000
9	PhoneService	Yes	4653	1699	0.267475
10	MultipleLines	No	2536	849	0.250812
11	MultipleLines	No phone service	510	170	0.250000
12	MultipleLines	Yes	2117	850	0.286485
13	InternetService	DSL	1957	459	0.189983
14	InternetService	Fiber optic	1799	1297	0.418928
15	InternetService	No	1407	113	0.074342
16	OnlineSecurity	No	2036	1461	0.417787
17	OnlineSecurity	No internet service	1407	113	0.074342
18	OnlineSecurity	Yes	1720	295	0.146402
19	OnlineBackup	No	1854	1233	0.399417
20	OnlineBackup	No internet service	1407	113	0.074342
21	OnlineBackup	Yes	1902	523	0.215670
22	DeviceProtection	No	1883	1211	0.391403
23	DeviceProtection	No internet service	1407	113	0.074342
24	DeviceProtection	Yes	1873	545	0.225393
25	TechSupport	No	2026	1446	0.416475
26	TechSupport	No internet service	1407	113	0.074342
27	TechSupport	Yes	1730	310	0.151961
28	StreamingTV	No	1867	942	0.335351
29	StreamingTV	No internet service	1407	113	0.074342
30	StreamingTV	Yes	1889	814	0.301147
31	StreamingMovies	No	1843	938	0.337289
32	StreamingMovies	No internet service	1407	113	0.074342
33	StreamingMovies	Yes	1913	818	0.299524

	attribute1	attribute2	No	Yes	churn_rate
34	Contract	Month-to-month	2220	1655	0.427097
35	Contract	One year	1306	166	0.112772
36	Contract	Two year	1637	48	0.028487
37	PaperlessBilling	No	2395	469	0.163757
38	PaperlessBilling	Yes	2768	1400	0.335893
39	PaymentMethod	Bank transfer (automatic)	1284	258	0.167315
40	PaymentMethod	Credit card (automatic)	1289	232	0.152531
41	PaymentMethod	Electronic check	1294	1071	0.452854
42	PaymentMethod	Mailed check	1296	308	0.192020

FEATURE ENGINEERING

In [21]:
```
df.drop('customerID', axis=1, inplace=True)
```

In [22]:
```
df.head()
```

Out[22]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetSe
0	Female	0	Yes	No	1	No	No phone service	
1	Male	0	No	No	34	Yes	No	
2	Male	0	No	No	2	Yes	No	
3	Male	0	No	No	45	No	No phone service	
4	Female	0	No	No	2	Yes	No	Fiber

In [23]:
```
target_col = ['Churn']
```

In [24]:
```
bin_cols = df.columns[df.nunique()<3].tolist()
```

In [25]:
```
bin_cols = [x for x in bin_cols if x not in target_col]
```

In [26]:
```
cat_cols = df.columns[df.nunique()<10].tolist()
```

In [27]:
```
cat_cols = [x for x in cat_cols if x not in target_col + bin_cols]
```

In [28]:
```
num_cols = [x for x in df.columns if x not in cat_cols+target_col+bin_cols]
```

In [29]:
```
target_col, num_cols, bin_cols, cat_cols
```

Out[29]:
```
(['Churn'],
 ['tenure', 'MonthlyCharges', 'TotalCharges'],
 ['gender',
  'SeniorCitizen',
  'Partner',
  'Dependents',
  'PhoneService',
  'PaperlessBilling'],
 ['MultipleLines',
  'InternetService',
  'OnlineSecurity',
  'OnlineBackup',
  'DeviceProtection',
  'TechSupport',
  'StreamingTV',
  'StreamingMovies',
  'Contract',
  'PaymentMethod'])
```

In [30]:
```
scale = StandardScaler()
```

In [31]:
```
num_cols_std = scale.fit_transform(df[num_cols])
```

In [32]:
```
num_cols_std  = pd.DataFrame(num_cols_std, columns=num_cols)
```

In [33]:
```
encoder = LabelEncoder()
```

In [34]:
```
for i in bin_cols:
    num_cols_std[i] = encoder.fit_transform(df[i])
```

In [35]:
```
num_cols_std
```

Out[35]:

	tenure	MonthlyCharges	TotalCharges	gender	SeniorCitizen	Partner	Dependents	P
0	-1.280248	-1.161694	-0.994194	0	0	1	0	
1	0.064303	-0.260878	-0.173740	1	0	0	0	
2	-1.239504	-0.363923	-0.959649	1	0	0	0	
3	0.512486	-0.747850	-0.195248	1	0	0	0	
4	-1.239504	0.196178	-0.940457	0	0	0	0	
...	...	...	...	...	...	...	...	
7027	-0.343137	0.664868	-0.129180	1	0	1	1	
7028	1.612573	1.276493	2.241056	0	0	1	1	
7029	-0.872808	-1.170004	-0.854514	0	0	1	1	
7030	-1.158016	0.319168	-0.872095	1	1	1	0	
7031	1.368109	1.357932	2.012344	1	0	0	0	

7032 rows × 9 columns

In [36]:
```
cat_cols_dum = pd.get_dummies(df[cat_cols])
```

In [37]:
```
num_cols_std.shape, cat_cols_dum.shape
```

Out[37]:
```
((7032, 9), (7032, 31))
```

In [38]:
```
cat_cols_dum.reset_index(drop=True, inplace=True)
num_cols_std.reset_index(drop=True, inplace=True)
```

In [39]:
```
df_final = pd.concat([num_cols_std, cat_cols_dum], axis=1, )
```

In [40]:
```
df_final.shape, df['Churn'].shape
```

Out[40]:
```
((7032, 40), (7032,))
```

In [41]:
```
y = encoder.fit_transform(df['Churn'])
```

In [42]:
```
X = df_final
```

In [43]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_si
```

MODEL BUILDING

In [44]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score, confusion_matrix, classification_
```

I Logistics Regression

In [45]:
```
model_log = LogisticRegression()
```

In [46]:
```
model_log.fit(X_train, y_train)
```

Out[46]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
                   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 [47]:
```
y_log_pred = model_log.predict(X_test)
```

In [48]:
```
print("Accuracy Score :", accuracy_score(y_test, y_log_pred)), print("Confusi
```

```
Accuracy Score : 0.8100113765642776
Confusion Matrix:
 [[1156  135]
 [ 199  268]]
```

Out[48]:
```
(None, None)
```

In [49]:
```
print(classification_report(y_test, y_log_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.90      0.87      1291
           1       0.67      0.57      0.62       467

    accuracy                           0.81      1758
   macro avg       0.76      0.73      0.74      1758
weighted avg       0.80      0.81      0.81      1758
```

II Random Forest Classification

In [50]:
```
model_rfc = RandomForestClassifier()
```

In [51]:
```
model_rfc.fit(X_train, y_train)
```

Out[51]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [52]:
```
y_rfc_pred = model_rfc.predict(X_test)
```

In [53]:
```
print("Accuracy Score :", accuracy_score(y_test, y_rfc_pred)), print("Confusi
```

```
Accuracy Score : 0.7957906712172924
Confusion Matrix:
 [[1158  133]
 [ 226  241]]
```

Out[53]:
```
(None, None)
```

In [54]:
```
print(classification_report(y_test, y_rfc_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1291
           1       0.64      0.52      0.57       467

    accuracy                           0.80      1758
   macro avg       0.74      0.71      0.72      1758
weighted avg       0.79      0.80      0.79      1758
```

XG Boost

In [56]:
```
model_xgb = xgb.XGBClassifier(max_depth=5, learning_rate=0.08, objective= 'bi
```

In [57]:
```
model_xgb.fit(X_train, y_train)
```

Out[57]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.08, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=-1, num_parallel_tree=1,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [58]:
```
y_xgb_pred = model_xgb.predict(X_test)
```

In [59]:
```
accuracy_score(y_test, y_xgb_pred)
```

Out[59]:
```
0.8088737201365188
```

In [60]:
```
confusion_matrix(y_test, y_xgb_pred)
```

Out[60]:
```
array([[1157,  134],
       [ 202,  265]], dtype=int64)
```

In [61]:
```
print(classification_report(y_test, y_xgb_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.90      0.87      1291
           1       0.66      0.57      0.61       467

    accuracy                           0.81      1758
   macro avg       0.76      0.73      0.74      1758
weighted avg       0.80      0.81      0.80      1758
```

IV Artificial Neural Network

In [62]:
```
model_ann = Sequential()
```

In [63]:
```
model_ann.add(Dense(input_dim=40, units=64, activation='relu'))
model_ann.add(Dropout(0.2))
model_ann.add(Dense(units=64, activation='relu'))
model_ann.add(Dropout(0.2))
model_ann.add(Dense(units=64, activation='relu'))
model_ann.add(Dropout(0.2))
model_ann.add(Dense(units=32, activation='relu'))
model_ann.add(Dropout(0.2))
model_ann.add(Dense(units=1, activation='sigmoid'))
model_ann.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc
```

In [64]:
```
model_ann.fit(X_train, y_train, batch_size=50, epochs=35, validation_data=(X_
```

```
Epoch 1/35
 88/106 [=====================>......] - ETA: 0s - loss: 0.5273 - accu
racy: 0.7302WARNING:tensorflow:Callbacks method `on_test_batch_begin` is
slow compared to the batch time (batch time: 0.0000s vs `on_test_batch_b
egin` time: 0.0010s). Check your callbacks.
106/106 [==============================] - 0s 3ms/step - loss: 0.5139 -
accuracy: 0.7419 - val_loss: 0.4387 - val_accuracy: 0.7878
Epoch 2/35
106/106 [==============================] - 0s 2ms/step - loss: 0.4506 -
accuracy: 0.7759 - val_loss: 0.4304 - val_accuracy: 0.7952
Epoch 3/35
106/106 [==============================] - 0s 3ms/step - loss: 0.4398 -
accuracy: 0.7907 - val_loss: 0.4165 - val_accuracy: 0.8072
Epoch 4/35
106/106 [==============================] - 0s 3ms/step - loss: 0.4346 -
accuracy: 0.7943 - val_loss: 0.4169 - val_accuracy: 0.8038
Epoch 5/35
106/106 [==============================] - 0s 3ms/step - loss: 0.4319 -
accuracy: 0.7950 - val_loss: 0.4240 - val_accuracy: 0.8055
```

In [65]:
```
y_ann_pred = model_ann.predict_classes(X_test)
```

```
WARNING:tensorflow:From <ipython-input-65-25eb986ee273>:1: Sequential.predi
ct_classes (from tensorflow.python.keras.engine.sequential) is deprecated a
nd will be removed after 2021-01-01.
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your mode
l does multi-class classification   (e.g. if it uses a `softmax` last-layer
activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model d
oes binary classification   (e.g. if it uses a `sigmoid` last-layer activat
ion).
```

In [66]:
```
accuracy_score(y_test, y_ann_pred)
```

Out[66]:
```
0.7918088737201365
```

In [67]:
```
confusion_matrix(y_test, y_ann_pred)
```

Out[67]:
```
array([[1124,  167],
       [ 199,  268]], dtype=int64)
```

In [68]:
```
print(classification_report(y_test, y_ann_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.87      0.86      1291
           1       0.62      0.57      0.59       467

    accuracy                           0.79      1758
   macro avg       0.73      0.72      0.73      1758
weighted avg       0.79      0.79      0.79      1758
```

In [70]:
```
# XGB performs slightly better than other model. Better Feature Engineer can
```

In [ ]:
```

```