```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statistics
%matplotlib inline
```

```
#Loading DataSet
df_initial = pd.read_csv("/content/Telco_Customer_Churn.csv")
```

```
#df_initial.head()
df_initial.sample(2)
```

```
len(df_initial.columns)
```

```
     21
```

```
df_initial.columns
```

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

```
#Removing unwanted coloumns
df = df_initial.drop(['customerID','OnlineBackup','DeviceProtection','TotalCharges'] , axis = 1)
```

```
print("Now we are left with ",len(df.columns),"Columns")
df.columns
```

```
Now we are left with  17 Columns
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'Churn'],
      dtype='object')
```

EDA / Understand Data

```
df.describe()   #describe
```

.SeniorCitizen = a categorical col.

.50% customers have tenure less than 29 months

.Average Monthly charges are USD 64.76

```
df.info(verbose= True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 17 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   gender            7043 non-null   object 
 1   SeniorCitizen     7043 non-null   int64  
 2   Partner           7043 non-null   object 
 3   Dependents        7043 non-null   object 
 4   tenure            7043 non-null   int64  
 5   PhoneService      7043 non-null   object 
 6   MultipleLines     7043 non-null   object 
 7   InternetService   7043 non-null   object 
 8   OnlineSecurity    7043 non-null   object 
 9   TechSupport       7043 non-null   object 
 10  StreamingTV       7043 non-null   object 
 11  StreamingMovies   7043 non-null   object 
 12  Contract          7043 non-null   object 
 13  PaperlessBilling  7043 non-null   object 
 14  PaymentMethod     7043 non-null   object 
 15  MonthlyCharges    7043 non-null   float64
 16  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(14)
memory usage: 935.5+ KB
```

```
#percentage of Yes/No
print(100*df['Churn'].value_counts()/len(df['Churn']))
print ("")
print("#Data is highly imbalanced")
```

```
No     73.463013
Yes    26.536987
Name: Churn, dtype: float64
```

```
#Data is highly imbalanced
```

```
sns.countplot(df['gender'])
```

```
df['Contract'].value_counts().plot(kind='pie',autopct='%.2f')
```

```
#for mumarical data
plt.hist(df['MonthlyCharges'],bins=5)
```

```
sns.boxplot(df['MonthlyCharges'])
sns.distplot(df['MonthlyCharges'])
```

```
for i, predictor in enumerate(df):
    plt.figure(i)
    sns.countplot(data=df, x=predictor, hue='Churn')
```

```
df.duplicated().sum()
```

```
     28
```

```
df.isnull().sum()
```

```
     gender              0
     SeniorCitizen       0
     Partner             0
     Dependents          0
     tenure              0
     PhoneService        0
     MultipleLines       0
     InternetService     0
     OnlineSecurity      0
     TechSupport         0
     StreamingTV         0
     StreamingMovies     0
     Contract            0
     PaperlessBilling    0
     PaymentMethod       0
     MonthlyCharges      0
     Churn               0
     dtype: int64
```

```
print(df.describe())   # only for numerical cols
```

```
            SeniorCitizen       tenure  MonthlyCharges
     count    7043.000000  7043.000000     7043.000000
     mean        0.162147    32.371149       64.761692
     std         0.368612    24.559481       30.090047
     min         0.000000     0.000000       18.250000
     25%         0.000000     9.000000       35.500000
     50%         0.000000    29.000000       70.350000
     75%         0.000000    55.000000       89.850000
     max         1.000000    72.000000      118.750000
```

```
df.corr() #corelation between numarical cols
```

```
#Numarical Columns = SeniorCitizen , Tenure
#categorical colums = gender,SeniorCitizen,Partner,Dependents,tenure,PhoneService,MultipleLines,InternetService,OnlineSecuri
#                     OnlineBackup,DeviceProtection,TechSupport,StreamingTV,StreamingMovies,Contract,PaperlessBilling,Paymen
#                     MonthlyCharges,TotalCharges
```

```
#1. remove duplicated values
df.drop_duplicates(inplace = True)
```

```
# since we have so many values in tenure (we can make some group)

def make_group(a):
  if a<=5:
    return ("New_customer")
  elif (a>5 and a<=25):
    return ("old_customer")
  else :
    return ("Loyal_customer")
```

```
df['tenure'] = df['tenure'].map(make_group)  #map function to the tenures col.
```

```
df.tenure.value_counts()
```

```
Loyal_customer    3751
old_customer      1917
New_customer      1347
Name: tenure, dtype: int64
```

```
df.sample(3)
```

```
#Label Encoding = only for output col.
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df.Churn = le.fit_transform(df['Churn'])

# also we can do this with this
#df['Churn'] = np.where(df.Churn == 'Yes',1,0)
```

Data Spliting

```
input = df.iloc[:,:-1]
output = df.iloc[:,-1]
```

```
output
```

```
     0       0
     1       0
     2       1
     3       0
```

```
4       1
       ..
7038    0
7039    0
7040    0
7041    1
7042    0
Name: Churn, Length: 7015, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(input,output,test_size=0.2)
```

Encoding

```
#Nominal Encoding = for Nominal Data (ONE HOT ENCODING)
#Ordinal Encoding = for Ordinal Data
from sklearn.preprocessing import OneHotEncoder,OrdinalEncoder
#OHE = OneHotEncoder()
#OHE.fit_transform(df[["gender","Partner","Dependents","PhoneService","PhoneService","OnlineSecurity","TechSupport","Streami

# To complete this in one step
from sklearn.compose import ColumnTransformer

transformer = ColumnTransformer(transformers=[('OE',OrdinalEncoder(categories=[['No phone service','No','Yes'],['No','DSL','
    ,('OHE',OneHotEncoder(sparse=False,drop='first'),["gender","Partner","Dependents","PhoneService","PhoneService","TechSup
    ],remainder ='passthrough')
```

```
X_train = transformer.fit_transform(X_train)
```

```
X_test = transformer.transform(X_test)
```

```
X_train[3:7]#some random rows from Data
```

```
array([[ 1.  ,  1.  ,  1.  ,  1.  ,  1.  ,  1.  ,  1.  ,  0.  ,  0.  ,
         0.  ,  0.  ,  1.  ,  1.  ,  0.  ,  0.  ,  1.  ,  1.  ,  0.  ,
         0.  ,  0.  , 49.05],
       [ 1.  ,  1.  ,  1.  ,  2.  ,  2.  ,  2.  ,  1.  ,  1.  ,  0.  ,
         0.  ,  0.  ,  1.  ,  1.  ,  0.  ,  1.  ,  1.  ,  1.  ,  0.  ,
         0.  ,  0.  , 70.75],
       [ 0.  ,  1.  ,  1.  ,  1.  ,  1.  ,  1.  ,  1.  ,  2.  ,  0.  ,
         0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  1.  ,  0.  ,  0.  ,
         1.  ,  0.  , 29.75],
       [ 2.  ,  1.  ,  2.  ,  2.  ,  2.  ,  1.  ,  2.  ,  2.  ,  0.  ,
         1.  ,  1.  ,  1.  ,  1.  ,  0.  ,  1.  ,  0.  ,  0.  ,  0.  ,
         1.  ,  0.  , 75.8 ]])
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(5612, 21)
(5612,)
(1403, 21)
(1403,)
```

Apply Algorithms

```
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
#naive_base
from sklearn.naive_bayes import MultinomialNB

mnb = MultinomialNB().fit(X_train, y_train)

print("train shape: " + str(X_train.shape))
print("score on test: " + str(mnb.score(X_test, y_test)))
print("score on train: "+ str(mnb.score(X_train, y_train)))
```

```
     train shape: (5612, 21)
     score on test: 0.7405559515324305
     score on train: 0.7653243050605845
```

```
from sklearn.linear_model import SGDClassifier

sgd=SGDClassifier()
sgd.fit(X_train, y_train)

print("train shape: " + str(X_train.shape))
print("score on test: " + str(sgd.score(X_test, y_test)))
print("score on train: "+ str(sgd.score(X_train, y_train)))
```

```
     train shape: (5612, 21)
     score on test: 0.722024233784747
     score on train: 0.7574839629365645
```

```
#knearest

from sklearn.neighbors import KNeighborsClassifier

#knn = KNeighborsClassifier(n_neighbors=5,algorithm = 'ball_tree')
knn = KNeighborsClassifier(algorithm = 'brute', n_jobs=-1)

knn.fit(X_train, y_train)

print("train shape: " + str(X_train.shape))
print("score on test: " + str(knn.score(X_test, y_test)))
print("score on train: "+ str(knn.score(X_train, y_train)))
```

```
     train shape: (5612, 21)
     score on test: 0.7398431931575196
     score on train: 0.8344618674269423
```

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

# setting
# min_samples_split=10
# max_depth=4

adb = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2),n_estimators=100,learning_rate=0.5)
adb.fit(X_train, y_train)

print("train shape: " + str(X_train.shape))
print("score on test: " + str(adb.score(X_test, y_test)))
print("score on train: "+ str(adb.score(X_train, y_train)))
```

```
     train shape: (5612, 21)
     score on test: 0.7840342124019958
     score on train: 0.8200285103349965
```

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier

# setting
# min_samples_split=10
# max_depth=4

gbc = GradientBoostingClassifier(n_estimators=100)
gbc.fit(X_train, y_train)

print("train shape: " + str(X_train.shape))
print("score on test: " + str(gbc.score(X_test, y_test)))
print("score on train: "+ str(gbc.score(X_train, y_train)))
```

```
     train shape: (5612, 21)
     score on test: 0.7847469707769066
     score on train: 0.8230577334283677
```

```
from sklearn.ensemble import RandomForestClassifier
```