```
# iranian churn dataset
# my drive
# https://drive.google.com/file/d/1BKpDDOHVj8ClwJZbv88VCLIr_SedvY_s/view?usp=sharing
# churn mean: the rate at which customers stop doing business with a company over a given period of time.
```

```
import pandas as pd
import numpy as np

# Data Loading (take csv file from drive)
url='https://drive.google.com/file/d/1BKpDDOHVj8ClwJZbv88VCLIr_SedvY_s/view?usp=sharing'
url='https://drive.google.com/uc?id=' + url.split('/')[-2]
df = pd.read_csv(url)
print("Data Frame:")
df.head()
```

```
Data Frame:
```

	Call Failure	Complains	Subscription Length	Charge Amount	Seconds of Use	Frequency of use	Frequency of SMS	Distin Call Numbe
0	8	0	38	0	4370	71	5	
1	0	0	39	0	318	5	7	
2	10	0	37	0	2453	60	359	
3	10	0	38	0	4198	66	1	
4	3	0	38	0	2393	58	2	

```
# Change Column Name for Ease in detection
df.rename(columns = {"Call  Failure": "Call_Failure", "Subscription  Length": "Subscription_Length", "Charge  Amount": "Charge_Amount",
                   "Seconds of Use": "Seconds_of_Use", "Frequency of use": "Frequency_of_use", "Frequency of SMS": "Frequency_of_SMS", "Disti
                   "Age Group": "Age_Group", "Tariff Plan": "Tariff_Plan", "Customer Value": "Customer_Value"},inplace=True)
print(df.columns)
```

```
Index(['Call_Failure', 'Complains', 'Subscription_Length', 'Charge_Amount',
       'Seconds_of_Use', 'Frequency_of_use', 'Frequency_of_SMS',
       'Distinct_Called_Numbers', 'Age_Group', 'Tariff_Plan', 'Status', 'Age',
       'Customer_Value', 'FN', 'FP', 'Churn'],
      dtype='object')
```

```
# Data Processing
print("Missing value:")
print(df.isna().sum())#missing values
print("catagorical value:")
print(df.info())#catagorical values
# Not have missing values
```

```
Missing value:
Call_Failure              0
Complains                 0
Subscription_Length       0
Charge_Amount             0
Seconds_of_Use            0
Frequency_of_use          0
Frequency_of_SMS          0
Distinct_Called_Numbers   0
Age_Group                 0
Tariff_Plan               0
Status                    0
Age                       0
Customer_Value            0
FN                        0
FP                        0
Churn                     0
dtype: int64
catagorical value:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3150 entries, 0 to 3149
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Call_Failure            3150 non-null   int64
 1   Complains               3150 non-null   int64
```

```
 2   Subscription_Length      3150 non-null   int64
 3   Charge_Amount            3150 non-null   int64
 4   Seconds_of_Use           3150 non-null   int64
 5   Frequency_of_use         3150 non-null   int64
 6   Frequency_of_SMS         3150 non-null   int64
 7   Distinct_Called_Numbers  3150 non-null   int64
 8   Age_Group                3150 non-null   int64
 9   Tariff_Plan              3150 non-null   int64
 10  Status                   3150 non-null   int64
 11  Age                      3150 non-null   int64
 12  Customer_Value           3150 non-null   float64
 13  FN                       3150 non-null   float64
 14  FP                       3150 non-null   float64
 15  Churn                    3150 non-null   int64
dtypes: float64(3), int64(13)
memory usage: 393.9 KB
None
```

```
print("statistical value:")
print(df.describe())#statistical propotion
```

```
statistical value:
       Call_Failure    Complains  Subscription_Length  Charge_Amount  \
count   3150.000000  3150.000000          3150.000000    3150.000000
mean       7.627937     0.076508            32.541905       0.942857
std        7.263886     0.265851             8.573482       1.521072
min        0.000000     0.000000             3.000000       0.000000
25%        1.000000     0.000000            30.000000       0.000000
50%        6.000000     0.000000            35.000000       0.000000
75%       12.000000     0.000000            38.000000       1.000000
max       36.000000     1.000000            47.000000      10.000000

       Seconds_of_Use  Frequency_of_use  Frequency_of_SMS  \
count     3150.000000       3150.000000       3150.000000
mean      4472.459683         69.460635         73.174921
std       4197.908687         57.413308        112.237560
min          0.000000          0.000000          0.000000
25%       1391.250000         27.000000          6.000000
50%       2990.000000         54.000000         21.000000
75%       6478.250000         95.000000         87.000000
max      17090.000000        255.000000        522.000000

       Distinct_Called_Numbers    Age_Group  Tariff_Plan       Status  \
count              3150.000000  3150.000000  3150.000000  3150.000000
mean                 23.509841     2.826032     1.077778     1.248254
std                  17.217337     0.892555     0.267864     0.432069
min                   0.000000     1.000000     1.000000     1.000000
25%                  10.000000     2.000000     1.000000     1.000000
50%                  21.000000     3.000000     1.000000     1.000000
75%                  34.000000     3.000000     1.000000     1.000000
max                  97.000000     5.000000     2.000000     2.000000

                Age  Customer_Value           FN           FP        Churn
count  3150.000000     3150.000000  3150.000000  3150.000000  3150.000000
mean     30.998413      470.972916   423.875624    98.304688     0.157143
std       8.831095      517.015433   465.313890    50.724492     0.363993
min      15.000000        0.000000     0.000000    60.000000     0.000000
25%      25.000000      113.801250   102.421125    61.380125     0.000000
50%      30.000000      228.480000   205.632000    72.848000     0.000000
75%      30.000000      788.388750   709.549875   128.838875     0.000000
max      55.000000     2165.280000  1948.752000   266.528000     1.000000
```

```
# Count distinct value in Target
print(df["Churn"].value_counts())
print("0: No, ", "1: Yes")
```

```
0    2655
1     495
Name: Churn, dtype: int64
0: No,  1: Yes
```

```
# Divide the set in Features and Target
X = df
y = df.pop("Churn")
print(X)
print(y)
```

```
3149             8          1                  11             2

      Seconds_of_Use  Frequency_of_use  Frequency_of_SMS  \
0               4370                71                 5
1                318                 5                 7
2               2453                60               359
3               4198                66                 1
4               2393                58                 2
...              ...               ...               ...
3145            6697               147                92
3146            9237               177                80
3147            3157                51                38
3148            4695                46               222
3149            1792                25                 7

      Distinct_Called_Numbers  Age_Group  Tariff_Plan  Status  Age  \
0                          17          3            1       1   30
1                           4          2            1       2   25
2                          24          3            1       1   30
3                          35          1            1       1   15
4                          33          1            1       1   15
...                       ...        ...          ...     ...  ...
3145                       44          2            2       1   25
3146                       42          5            1       1   55
3147                       21          3            1       1   30
3148                       12          3            1       1   30
3149                        9          3            1       1   30

      Customer_Value         FN        FP
0            197.640   177.8760   69.7640
1             46.035    41.4315   60.0000
2           1536.520  1382.8680  203.6520
3            240.020   216.0180   74.0020
4            145.805   131.2245   64.5805
...              ...        ...       ...
3145         721.980   649.7820  122.1980
3146         261.210   235.0890   76.1210
3147         280.320   252.2880   78.0320
3148        1077.640   969.8760  157.7640
3149         100.680    90.6120   60.0680

[3150 rows x 15 columns]
0       0
1       0
2       0
3       0
4       0
       ..
3145    0
3146    0
3147    0
3148    0
3149    1
Name: Churn, Length: 3150, dtype: int64
```

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Extract numerical features into a separate DataFrame
numerical_features = X.select_dtypes(include=['float64', 'int64'])

# Standardization
standard_scaler = StandardScaler()
X = pd.DataFrame(standard_scaler.fit_transform(numerical_features),
                                 columns=numerical_features.columns)

print(X)
```

```
      Call_Failure  Complains  Subscription_Length  Charge_Amount  \
0         0.051229  -0.287830             0.636726      -0.619962
1        -1.050285  -0.287830             0.753384      -0.619962
2         0.326608  -0.287830             0.520069      -0.619962
3         0.326608  -0.287830             0.636726      -0.619962
4        -0.637217  -0.287830             0.636726      -0.619962
...            ...        ...                  ...            ...
3145      1.841189  -0.287830            -1.579761       0.695109
3146      1.290432  -0.287830            -1.813076       0.037573
3147      0.739675  -0.287830            -1.696418       2.010180
3148     -0.086460  -0.287830            -2.513019       0.695109
3149      0.051229   3.474268            -2.513019       0.695109

      Seconds_of_Use  Frequency_of_use  Frequency_of_SMS  \
0          -0.024411          0.026816         -0.607513
```

```
1         -0.989807        -1.122926        -0.589691
2         -0.481140        -0.164807         2.547012
3         -0.065390        -0.060285        -0.643157
4         -0.495435        -0.199648        -0.634246
...             ...              ...              ...
3145       0.530000         1.350761         0.167752
3146       1.135160         1.873371         0.060819
3147      -0.313410        -0.321590        -0.313447
3148       0.053021        -0.408692         1.326193
3149      -0.638624        -0.774519        -0.589691

      Distinct_Called_Numbers  Age_Group  Tariff_Plan    Status       Age  \
0                   -0.378158   0.194941    -0.290409 -0.574662 -0.113074
1                   -1.133331  -0.925616    -0.290409  1.740153 -0.679346
2                    0.028473   0.194941    -0.290409 -0.574662 -0.113074
3                    0.667466  -2.046172    -0.290409 -0.574662 -1.811888
4                    0.551285  -2.046172    -0.290409 -0.574662 -1.811888
...                       ...        ...          ...       ...       ...
3145                 1.190278  -0.925616     3.443420 -0.574662 -0.679346
3146                 1.074097   2.436055    -0.290409 -0.574662  2.718281
3147                -0.145797   0.194941    -0.290409 -0.574662 -0.113074
3148                -0.668609   0.194941    -0.290409 -0.574662 -0.113074
3149                -0.842880   0.194941    -0.290409 -0.574662 -0.113074

      Customer_Value        FN        FP
0          -0.528759 -0.528759 -0.562750
1          -0.822036 -0.822036 -0.755272
2           2.061285  2.061285  2.077183
3          -0.446775 -0.446775 -0.479188
4          -0.629033 -0.629033 -0.664956
...              ...       ...       ...
3145        0.485570  0.485570  0.471116
3146       -0.405783 -0.405783 -0.437406
3147       -0.368815 -0.368815 -0.399726
3148        1.173589  1.173589  1.172387
3149       -0.716326 -0.716326 -0.753931

[3150 rows x 15 columns]
```

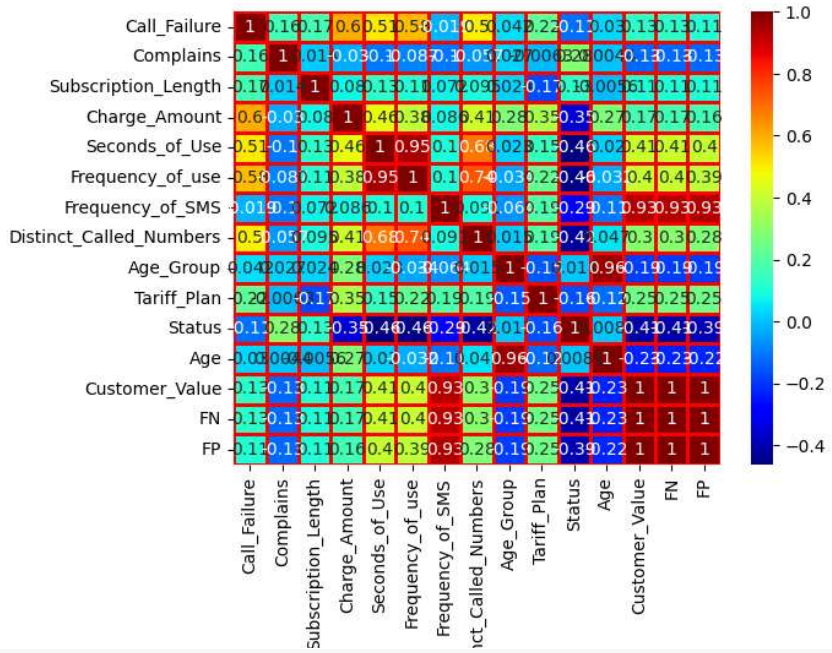
```
# Feature Selection Trying Done After split for avoid over fitting!!
```

```
# split the Data into two test and train with 3 ratios
from sklearn.model_selection import train_test_split
# 70 -30
x_train1,x_test1,y_train1,y_test1 = train_test_split(X,y,test_size=0.3,random_state=0)
# 80 -20
x_train2,x_test2,y_train2,y_test2 = train_test_split(X,y,test_size=0.2,random_state=0)
# 60 -40
x_train3,x_test3,y_train3,y_test3 = train_test_split(X,y,test_size=0.4,random_state=0)
print("X Train: ")
print("70/30: ",x_train1.shape)
print("80/20: ",x_train2.shape)
print("60/40: ",x_train3.shape)
print("X Test: ")
print("70/30: ",x_test1.shape)
print("80/20: ",x_test2.shape)
print("60/40: ",x_test3.shape)
```

```
X Train:
70/30:  (2205, 15)
80/20:  (2520, 15)
60/40:  (1890, 15)
X Test:
70/30:  (945, 15)
80/20:  (630, 15)
60/40:  (1260, 15)
```

```
# Feature Selection
# For xtrain1
import seaborn as sns

cor = x_train1.corr()
sns.heatmap(cor,annot=True,linewidth=2, linecolor="r", cmap="jet")
```

```
<Axes: >
```




```
# function for Feature Selection
def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr

corr_features = correlation(x_train1, 0.8)# let 0.8 as threshold
x_train1.drop(corr_features,axis=1)
x_test1.drop(corr_features,axis=1)
print(corr_features)
```

```
{'Frequency_of_use', 'FP', 'FN', 'Age', 'Customer_Value'}
```

```
# Feature Selection
# For xtrain2
import seaborn as sns

cor = x_train2.corr()
sns.heatmap(cor,annot=True,linewidth=2, linecolor="r", cmap="jet")
```

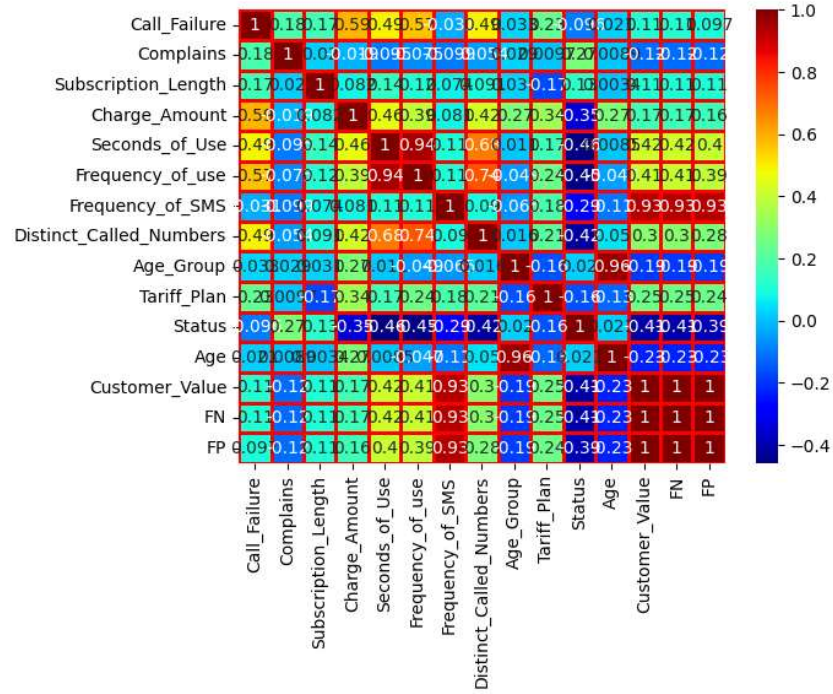
```
<Axes: >
```


```
# function for Feature Selection
def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr

corr_features = correlation(x_train1, 0.8)# let 0.8 as threshold
x_train2.drop(corr_features,axis=1)
x_test2.drop(corr_features,axis=1)
print(corr_features)
```

```
{'Frequency_of_use', 'FP', 'FN', 'Age', 'Customer_Value'}
```

```
# Feature Selection
# For xtrain3
import seaborn as sns

cor = x_train3.corr()
sns.heatmap(cor,annot=True,linewidth=2, linecolor="r", cmap="jet")
```

```
<Axes: >
```




```
# function for Feature Selection
def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
```

```
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr

corr_features = correlation(x_train1, 0.8)# let 0.8 as threshold
x_train3.drop(corr_features,axis=1)
x_test3.drop(corr_features,axis=1)
print(corr_features)
```

```
{'Frequency_of_use', 'FP', 'FN', 'Age', 'Customer_Value'}
```

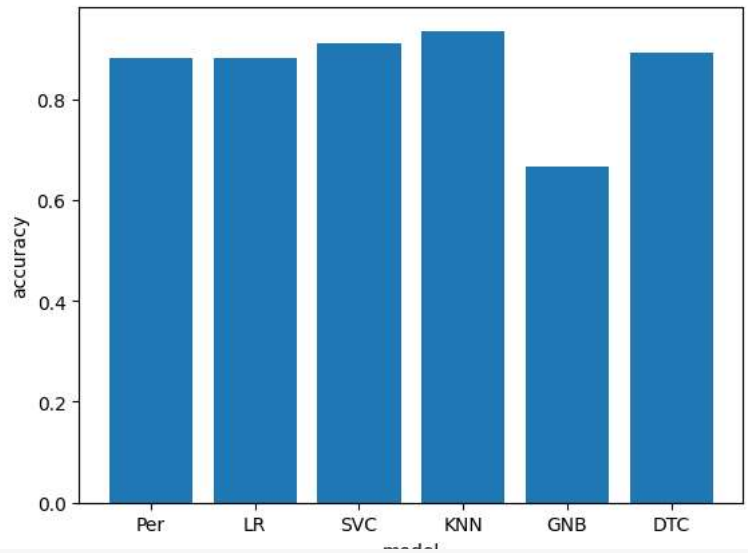
Classification

Perceptron, LogisticRegression, SVC, KNeighborsClassifier, DecisionTreeClassifier, GaussianNB

```
# Classification 6 modals
# use perceptron, LogisticRegression, SVC, KNeighborsClassifier, DecisionTreeClassifier, GaussianNB
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.linear_model import Perceptron, LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as pyt

# Perceptron
clf1 = Perceptron(alpha=0.001, l1_ratio=0.2, max_iter=1000)

# logistic regresiion
clf2 = LogisticRegression(penalty="l2", tol=0.001, C=1.0, max_iter=10000)

# svc
clf3 = SVC(C=1.0, kernel="rbf")


# knn
clf4 = KNeighborsClassifier(n_neighbors=5, p=2, metric="minkowski")

# Gaussian NB
clf5 = GaussianNB(priors=None)

# tree
clf6 = DecisionTreeClassifier(criterion="gini", splitter="best", max_depth=3)

clf = [clf1,clf2,clf3,clf4,clf5,clf6]
clf_name = ["Per", "LR", "SVC", "KNN" , "GNB", "DTC"]
```

```
# predict churn using x_train1
acc={}
T={}
for model, model_name in zip(clf, clf_name):
  model.fit(x_train1, y_train1)
  pred = model.predict(x_test1)
  acc[model_name] = accuracy_score(pred,y_test1)
  precision = precision_score(y_test1, pred)
  recall = recall_score(y_test1, pred)
  f1 = f1_score(y_test1, pred)
  # Generate a classification report
  report = classification_report(y_test1, pred)
  T[model_name] = report
for i,j in acc.items():
  print(i,":",j)
for i,j in acc.items():
  print(i,":",T[i])

pyt.ylabel("accuracy")
pyt.xlabel("model")
pyt.bar(acc.keys(),acc.values())
pyt.show()
```

```
Per : 0.8804232804232804
LR : 0.8825396825396825
SVC : 0.91005291005291
KNN : 0.9354497354497354
GNB : 0.6666666666666666
DTC : 0.8920634920634921
Per :               precision    recall  f1-score   support

           0       0.88      0.99      0.93       776
           1       0.86      0.40      0.54       169

    accuracy                           0.88       945
   macro avg       0.87      0.69      0.74       945
weighted avg       0.88      0.88      0.86       945

LR :               precision    recall  f1-score   support

           0       0.89      0.98      0.93       776
           1       0.84      0.43      0.56       169

    accuracy                           0.88       945
   macro avg       0.86      0.70      0.75       945
weighted avg       0.88      0.88      0.87       945

SVC :               precision    recall  f1-score   support

           0       0.91      0.99      0.95       776
           1       0.94      0.53      0.68       169

    accuracy                           0.91       945
   macro avg       0.92      0.76      0.81       945
weighted avg       0.91      0.91      0.90       945

KNN :               precision    recall  f1-score   support

           0       0.95      0.97      0.96       776
           1       0.85      0.78      0.81       169

    accuracy                           0.94       945
   macro avg       0.90      0.88      0.89       945
weighted avg       0.93      0.94      0.93       945

GNB :               precision    recall  f1-score   support

           0       0.97      0.61      0.75       776
           1       0.34      0.92      0.50       169

    accuracy                           0.67       945
   macro avg       0.66      0.76      0.62       945
weighted avg       0.86      0.67      0.71       945

DTC :               precision    recall  f1-score   support

           0       0.92      0.96      0.94       776
           1       0.75      0.60      0.66       169

    accuracy                           0.89       945
   macro avg       0.83      0.78      0.80       945
weighted avg       0.89      0.89      0.89       945
```



```
# predict churn using x_train2

acc={}
T={}
for model, model_name in zip(clf, clf_name):
  model.fit(x_train2, y_train2)
  pred = model.predict(x_test2)
  acc[model_name] = accuracy_score(pred,y_test2)
  precision = precision_score(y_test2, pred)
  recall = recall_score(y_test2, pred)
  f1 = f1_score(y_test2, pred)

  # Generate a classification report
  report = classification_report(y_test2, pred)
  T[model_name] = report
for i,j in acc.items():
  print(i,":",j)
for i,j in acc.items():
  print(i,":",T[i])

pyt.ylabel("accuracy")
pyt.xlabel("model")
pyt.bar(acc.keys(),acc.values())
pyt.show()
classi = {"KNN": "Accuracy-94.28571428571428 %"}
```

```
Per : 0.8031746031746032
LR : 0.8825396825396825
SVC : 0.9079365079365079
KNN : 0.9428571428571428
GNB : 0.6666666666666666
DTC : 0.8888888888888888
Per :               precision    recall  f1-score   support

           0       0.94      0.81      0.87       521
           1       0.46      0.76      0.57       109

    accuracy                           0.80       630
   macro avg       0.70      0.79      0.72       630
weighted avg       0.86      0.80      0.82       630

LR :               precision    recall  f1-score   support

           0       0.89      0.98      0.93       521
           1       0.80      0.43      0.56       109

    accuracy                           0.88       630
   macro avg       0.84      0.70      0.75       630
weighted avg       0.88      0.88      0.87       630

SVC :               precision    recall  f1-score   support

           0       0.91      0.99      0.95       521
           1       0.93      0.50      0.65       109

    accuracy                           0.91       630
```

```
# predict churn using x_train3
acc={}
T={}
for model, model_name in zip(clf, clf_name):
  model.fit(x_train3, y_train3)
  pred = model.predict(x_test3)
  acc[model_name] = accuracy_score(pred,y_test3)
  precision = precision_score(y_test3, pred)
  recall = recall_score(y_test3, pred)
  f1 = f1_score(y_test3, pred)

  # Generate a classification report
  report = classification_report(y_test3, pred)
  T[model_name] = report
for i,j in acc.items():
  print(i,":",j)
for i,j in acc.items():
  print(i,":",T[i])

pyt.ylabel("accuracy")
pyt.xlabel("model")
pyt.bar(acc.keys(),acc.values())
pyt.show()
```

```
Per : 0.8357142857142857
LR : 0.8896825396825396
SVC : 0.9142857142857143
KNN : 0.9373015873015873
GNB : 0.6714285714285714
DTC : 0.8976190476190476
Per :               precision    recall  f1-score   support

           0       0.84      1.00      0.91      1048
           1       0.69      0.04      0.08       212

    accuracy                           0.84      1260
   macro avg       0.76      0.52      0.49      1260
weighted avg       0.81      0.84      0.77      1260

LR :               precision    recall  f1-score   support

           0       0.90      0.98      0.94      1048
           1       0.82      0.44      0.57       212

    accuracy                           0.89      1260
   macro avg       0.86      0.71      0.75      1260
weighted avg       0.88      0.89      0.88      1260

SVC :               precision    recall  f1-score   support

           0       0.91      0.99      0.95      1048
           1       0.93      0.53      0.68       212

    accuracy                           0.91      1260
   macro avg       0.92      0.76      0.81      1260
weighted avg       0.92      0.91      0.90      1260

KNN :               precision    recall  f1-score   support

           0       0.95      0.97      0.96      1048
           1       0.84      0.77      0.80       212

    accuracy                           0.94      1260
   macro avg       0.90      0.87      0.88      1260
weighted avg       0.94      0.94      0.94      1260

GNB :               precision    recall  f1-score   support

           0       0.98      0.62      0.76      1048
           1       0.33      0.93      0.49       212

    accuracy                           0.67      1260
   macro avg       0.65      0.77      0.62      1260
weighted avg       0.87      0.67      0.71      1260

DTC :               precision    recall  f1-score   support

           0       0.90      0.99      0.94      1048
           1       0.93      0.42      0.58       212

    accuracy                           0.90      1260
   macro avg       0.91      0.71      0.76      1260
```

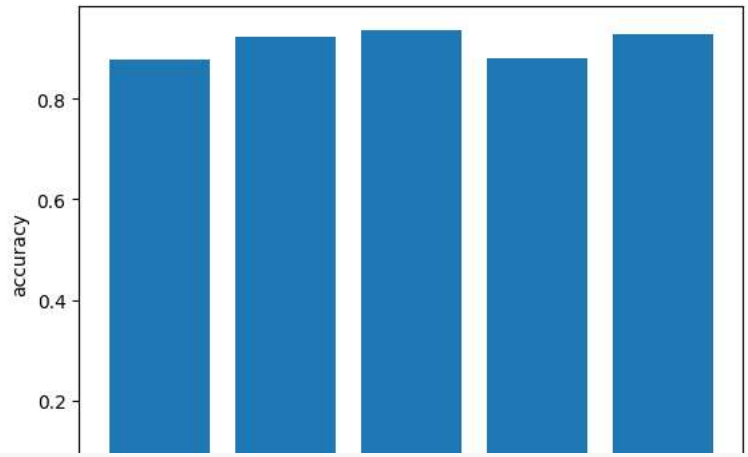
Regression (Threshold aproach)

Linear Regression , DicisionTreeRegression, RandomforestRegressor, GradientBoosting Regressor, SVR

```
# Regression done using thresold technique as it has output 0 1
# Linear Regression , DicisionTreeRegression, RandomforestRegressor, GradientBoosting Regressor, SVR
# For (70-30) test split
import numpy as np
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

acc={}
# Linear Regression
linear_reg = LinearRegression()
linear_reg.fit(x_train1, y_train1)
```

```
linear_reg_predictions = linear_reg.predict(x_test1)
linear_reg_binary_predictions = np.where(linear_reg_predictions > 0.5, 1, 0)

# Decision Tree Regression
dt_reg = DecisionTreeRegressor(random_state=42)
dt_reg.fit(x_train1, y_train1)
dt_reg_predictions = dt_reg.predict(x_test1)
dt_reg_binary_predictions = np.where(dt_reg_predictions > 0.5, 1, 0)

# Random Forest Regression
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(x_train1, y_train1)
rf_reg_predictions = rf_reg.predict(x_test1)
rf_reg_binary_predictions = np.where(rf_reg_predictions > 0.5, 1, 0)

# Support Vector Regression (SVR)
svr_reg = SVR(kernel='linear')
svr_reg.fit(x_train1, y_train1)
svr_reg_predictions = svr_reg.predict(x_test1)
svr_reg_binary_predictions = np.where(svr_reg_predictions > 0.5, 1, 0)

# Gradient Boosting Regression
gb_reg = GradientBoostingRegressor(random_state=42)
gb_reg.fit(x_train1, y_train1)
gb_reg_predictions = gb_reg.predict(x_test1)
gb_reg_binary_predictions = np.where(gb_reg_predictions > 0.5, 1, 0)


print("Linear Regression Metrics:")
print("Accuracy:", accuracy_score(y_test1, linear_reg_binary_predictions))
print("Precision:", precision_score(y_test1, linear_reg_binary_predictions))
print("Recall:", recall_score(y_test1, linear_reg_binary_predictions))
print("F1 Score:", f1_score(y_test1, linear_reg_binary_predictions))
acc["LR"] = accuracy_score(y_test1, linear_reg_binary_predictions)
print()

print("Decision Tree Regression Metrics:")
print("Accuracy:", accuracy_score(y_test1, dt_reg_binary_predictions))
print("Precision:", precision_score(y_test1, dt_reg_binary_predictions))
print("Recall:", recall_score(y_test1, dt_reg_binary_predictions))
print("F1 Score:", f1_score(y_test1, dt_reg_binary_predictions))
acc["DTR"] = accuracy_score(y_test1, dt_reg_binary_predictions)
print()

print("Random Forest Regression Metrics:")
print("Accuracy:", accuracy_score(y_test1, rf_reg_binary_predictions))
print("Precision:", precision_score(y_test1, rf_reg_binary_predictions))
print("Recall:", recall_score(y_test1, rf_reg_binary_predictions))
print("F1 Score:", f1_score(y_test1, rf_reg_binary_predictions))
acc["RFR"] = accuracy_score(y_test1, rf_reg_binary_predictions)
print()

print("Support Vector Regression Metrics:")
print("Accuracy:", accuracy_score(y_test1, svr_reg_binary_predictions))
print("Precision:", precision_score(y_test1, svr_reg_binary_predictions))
print("Recall:", recall_score(y_test1, svr_reg_binary_predictions))
print("F1 Score:", f1_score(y_test1, svr_reg_binary_predictions))
acc["SVR"] = accuracy_score(y_test1, svr_reg_binary_predictions)
print()

print("Gradient Boosting Regression Metrics:")
print("Accuracy:", accuracy_score(y_test1, gb_reg_binary_predictions))
print("Precision:", precision_score(y_test1, gb_reg_binary_predictions))
print("Recall:", recall_score(y_test1, gb_reg_binary_predictions))
print("F1 Score:", f1_score(y_test1, gb_reg_binary_predictions))
acc["GB"] = accuracy_score(y_test1, gb_reg_binary_predictions)

pyt.ylabel("accuracy")
pyt.xlabel("model")
pyt.bar(acc.keys(),acc.values())
pyt.show()
```

```
Linear Regression Metrics:
Accuracy: 0.8793650793650793
Precision: 0.8021978021978022
Recall: 0.4319526627218935
F1 Score: 0.5615384615384615

Decision Tree Regression Metrics:
Accuracy: 0.9238095238095239
Precision: 0.8012422360248447
Recall: 0.7633136094674556
F1 Score: 0.7818181818181819

Random Forest Regression Metrics:
Accuracy: 0.9375661375661376
Precision: 0.8571428571428571
Recall: 0.7810650887573964
F1 Score: 0.8173374613003095

Support Vector Regression Metrics:
Accuracy: 0.8804232804232804
Precision: 0.8414634146341463
Recall: 0.40828402366863903
F1 Score: 0.549800796812749

Gradient Boosting Regression Metrics:
Accuracy: 0.9301587301587302
Precision: 0.8551724137931035
Recall: 0.7337278106508875
F1 Score: 0.7898089171974522
```




```
# Regression done using thresold technique as it has output 0 1
# Linear Regression , DicisionTreeRegression, RandomforestRegressor, GradientBoosting Regressor, SVR
# For (80-20) test split

import numpy as np
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

acc={}
# Linear Regression
linear_reg = LinearRegression()
linear_reg.fit(x_train2, y_train2)
linear_reg_predictions = linear_reg.predict(x_test2)
linear_reg_binary_predictions = np.where(linear_reg_predictions > 0.5, 1, 0)

# Decision Tree Regression
dt_reg = DecisionTreeRegressor(random_state=42)
dt_reg.fit(x_train2, y_train2)
dt_reg_predictions = dt_reg.predict(x_test2)
dt_reg_binary_predictions = np.where(dt_reg_predictions > 0.5, 1, 0)

# Random Forest Regression
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(x_train2, y_train2)
rf_reg_predictions = rf_reg.predict(x_test2)
rf_reg_binary_predictions = np.where(rf_reg_predictions > 0.5, 1, 0)
```

```
# Support Vector Regression (SVR)
svr_reg = SVR(kernel='linear')
svr_reg.fit(x_train2, y_train2)
svr_reg_predictions = svr_reg.predict(x_test2)
svr_reg_binary_predictions = np.where(svr_reg_predictions > 0.5, 1, 0)

# Gradient Boosting Regression
gb_reg = GradientBoostingRegressor(random_state=42)
gb_reg.fit(x_train2, y_train2)
gb_reg_predictions = gb_reg.predict(x_test2)
gb_reg_binary_predictions = np.where(gb_reg_predictions > 0.5, 1, 0)


print("Linear Regression Metrics:")
print("Accuracy:", accuracy_score(y_test2, linear_reg_binary_predictions))
print("Precision:", precision_score(y_test2, linear_reg_binary_predictions))
print("Recall:", recall_score(y_test2, linear_reg_binary_predictions))
print("F1 Score:", f1_score(y_test2, linear_reg_binary_predictions))
acc["LR"] = accuracy_score(y_test2, linear_reg_binary_predictions)
print()

print("Decision Tree Regression Metrics:")
print("Accuracy:", accuracy_score(y_test2, dt_reg_binary_predictions))
print("Precision:", precision_score(y_test2, dt_reg_binary_predictions))
print("Recall:", recall_score(y_test2, dt_reg_binary_predictions))
print("F1 Score:", f1_score(y_test2, dt_reg_binary_predictions))
acc["DTR"] = accuracy_score(y_test2, dt_reg_binary_predictions)
print()

print("Random Forest Regression Metrics:")
print("Accuracy:", accuracy_score(y_test2, rf_reg_binary_predictions))
print("Precision:", precision_score(y_test2, rf_reg_binary_predictions))
print("Recall:", recall_score(y_test2, rf_reg_binary_predictions))
print("F1 Score:", f1_score(y_test2, rf_reg_binary_predictions))
acc["RFR"] = accuracy_score(y_test2, rf_reg_binary_predictions)
print()

print("Support Vector Regression Metrics:")
print("Accuracy:", accuracy_score(y_test2, svr_reg_binary_predictions))
print("Precision:", precision_score(y_test2, svr_reg_binary_predictions))
print("Recall:", recall_score(y_test2, svr_reg_binary_predictions))
print("F1 Score:", f1_score(y_test2, svr_reg_binary_predictions))
acc["SVR"] = accuracy_score(y_test2, svr_reg_binary_predictions)
print()

print("Gradient Boosting Regression Metrics:")
print("Accuracy:", accuracy_score(y_test2, gb_reg_binary_predictions))
print("Precision:", precision_score(y_test2, gb_reg_binary_predictions))
print("Recall:", recall_score(y_test2, gb_reg_binary_predictions))
print("F1 Score:", f1_score(y_test2, gb_reg_binary_predictions))
acc["GB"] = accuracy_score(y_test2, gb_reg_binary_predictions)

pyt.ylabel("accuracy")
pyt.xlabel("model")
pyt.bar(acc.keys(),acc.values())
pyt.show()
```

```
Linear Regression Metrics:
Accuracy: 0.8825396825396825
Precision: 0.8070175438596491
Recall: 0.42201834862385323
F1 Score: 0.5542168674698795

Decision Tree Regression Metrics:
Accuracy: 0.9222222222222223
Precision: 0.8061224489795918
Recall: 0.7247706422018348
F1 Score: 0.7632850241545894

Random Forest Regression Metrics:
Accuracy: 0.9333333333333333
Precision: 0.845360824742268
Recall: 0.7522935779816514
F1 Score: 0.796116504854369

Support Vector Regression Metrics:
Accuracy: 0.8825396825396825
Precision: 0.8431372549019608
Recall: 0.3944954128440367
F1 Score: 0.5375000000000001

Gradient Boosting Regression Metrics:
Accuracy: 0.9349206349206349
Precision: 0.8541666666666666
Recall: 0.7522935779816514
F1 Score: 0.8
```

```
# Regression done using thresold technique as it has output 0 1
# Linear Regression , DicisionTreeRegression, RandomforestRegressor, GradientBoosting Regressor, SVR
# For (60-40) test split

import numpy as np
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

acc={}
# Linear Regression
linear_reg = LinearRegression()
linear_reg.fit(x_train3, y_train3)
linear_reg_predictions = linear_reg.predict(x_test3)
linear_reg_binary_predictions = np.where(linear_reg_predictions > 0.5, 1, 0)

# Decision Tree Regression
dt_reg = DecisionTreeRegressor(random_state=42)
dt_reg.fit(x_train3, y_train3)
dt_reg_predictions = dt_reg.predict(x_test3)
dt_reg_binary_predictions = np.where(dt_reg_predictions > 0.5, 1, 0)

# Random Forest Regression
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
rf_reg.fit(x_train3, y_train3)
rf_reg_predictions = rf_reg.predict(x_test3)
rf_reg_binary_predictions = np.where(rf_reg_predictions > 0.5, 1, 0)

# Support Vector Regression (SVR)
svr_reg = SVR(kernel='linear')
svr_reg.fit(x_train3, y_train3)
svr_reg_predictions = svr_reg.predict(x_test3)
svr_reg_binary_predictions = np.where(svr_reg_predictions > 0.5, 1, 0)

# Gradient Boosting Regression
gb_reg = GradientBoostingRegressor(random_state=42)
gb_reg.fit(x_train3, y_train3)
gb_reg_predictions = gb_reg.predict(x_test3)
gb_reg_binary_predictions = np.where(gb_reg_predictions > 0.5, 1, 0)

print("Linear Regression Metrics:")
```

```
print("Accuracy:", accuracy_score(y_test3, linear_reg_binary_predictions))
print("Precision:", precision_score(y_test3, linear_reg_binary_predictions))
print("Recall:", recall_score(y_test3, linear_reg_binary_predictions))
print("F1 Score:", f1_score(y_test3, linear_reg_binary_predictions))
acc["LR"] = accuracy_score(y_test3, linear_reg_binary_predictions)
print()

print("Decision Tree Regression Metrics:")
print("Accuracy:", accuracy_score(y_test3, dt_reg_binary_predictions))
print("Precision:", precision_score(y_test3, dt_reg_binary_predictions))
print("Recall:", recall_score(y_test3, dt_reg_binary_predictions))
print("F1 Score:", f1_score(y_test3, dt_reg_binary_predictions))
acc["DTR"] = accuracy_score(y_test3, dt_reg_binary_predictions)
print()

print("Random Forest Regression Metrics:")
print("Accuracy:", accuracy_score(y_test3, rf_reg_binary_predictions))
print("Precision:", precision_score(y_test3, rf_reg_binary_predictions))
print("Recall:", recall_score(y_test3, rf_reg_binary_predictions))
print("F1 Score:", f1_score(y_test3, rf_reg_binary_predictions))
acc["RFR"] = accuracy_score(y_test3, rf_reg_binary_predictions)
print()

print("Support Vector Regression Metrics:")
print("Accuracy:", accuracy_score(y_test3, svr_reg_binary_predictions))
print("Precision:", precision_score(y_test3, svr_reg_binary_predictions))
print("Recall:", recall_score(y_test3, svr_reg_binary_predictions))
print("F1 Score:", f1_score(y_test3, svr_reg_binary_predictions))
acc["SVR"] = accuracy_score(y_test3, svr_reg_binary_predictions)
print()

print("Gradient Boosting Regression Metrics:")
print("Accuracy:", accuracy_score(y_test3, gb_reg_binary_predictions))
print("Precision:", precision_score(y_test3, gb_reg_binary_predictions))
print("Recall:", recall_score(y_test3, gb_reg_binary_predictions))
print("F1 Score:", f1_score(y_test3, gb_reg_binary_predictions))
acc["GB"] = accuracy_score(y_test3, gb_reg_binary_predictions)

pyt.ylabel("accuracy")
pyt.xlabel("model")
pyt.bar(acc.keys(),acc.values())
pyt.show()
```

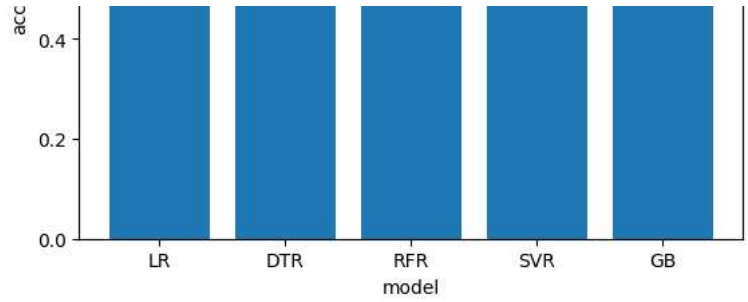
```
Linear Regression Metrics:
Accuracy: 0.8920634920634921
Precision: 0.7753623188405797
Recall: 0.5047169811320755
F1 Score: 0.6114285714285714

Decision Tree Regression Metrics:
Accuracy: 0.9261904761904762
Precision: 0.8082901554404145
Recall: 0.7358490566037735
F1 Score: 0.7703703703703704

Random Forest Regression Metrics:
Accuracy: 0.942063492063492
Precision: 0.8677248677248677
Recall: 0.7735849056603774
F1 Score: 0.8179551122194514
```

Result

```
Recall: 0.42452830188679247
```

```
print("This Result on basis of accuracy:")
print("For Classification:")
print("KNeighborsClassifier is best with",classi["KNN"])
print("With 80 20 split")
print()
print("For Regression:")
print("Random Forest Regression is best with Accuracy-",accuracy_score(y_test3, dt_reg_binary_predictions))
print("With 60 40 split")
```

```
This Result on basis of accuracy:
For Classification:
KNeighborsClassifier is best with Accuracy-94.28571428571428 %
With 80 20 split

For Regression:
Random Forest Regression is best with Accuracy- 0.9261904761904762
With 60 40 split
```



✓ 0s completed at 8:08 AM