Final Project: Churn Analysis

The objective of this project is to help PowerCo., a gas and electricity distribution company, anticipate customer churn that would ultimately help them save those customers. After analyzing the data and applying around 7 machine learning algorithms, we achieved an accuracy of 90% and a recall of 76% on the test data. This indicates our models' ability to predict customer churn and offers an opportunity to explore a discounted rate of 20% as a potential solution.

```
!pip install pandas-profiling

(Asad)
```

```
Collecting pandas-profiling
  Downloading pandas_profiling-3.6.6-py2.py3-none-any.whl (324 kB)
                                        324.4/324.4 kB 19.0 MB/s eta 0:00
Collecting ydata-profiling (from pandas-profiling)
  Downloading ydata_profiling-4.3.1-py2.py3-none-any.whl (352 kB)
                                        353.0/353.0 kB 32.8 MB/s eta 0:00
Requirement already satisfied: scipy<1.11,>=1.4.1 in /usr/local/lib/python3.1
Requirement already satisfied: pandas!=1.4.0,<2.1,>1.1 in /usr/local/lib/pyth
Requirement already satisfied: matplotlib<4,>=3.2 in /usr/local/lib/python3.1
Requirement already satisfied: pydantic<2,>=1.8.1 in /usr/local/lib/python3.1
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.
Collecting visions[type_image_path]==0.7.5 (from ydata-profiling->pandas-prof
  Downloading visions-0.7.5-py3-none-any.whl (102 kB)
                                        102.7/102.7 kB 11.2 MB/s eta 0:00
Requirement already satisfied: numpy<1.24,>=1.16.0 in /usr/local/lib/python3.
Collecting htmlmin==0.1.12 (from ydata-profiling->pandas-profiling)
  Downloading htmlmin-0.1.12.tar.gz (19 kB)
  Preparing metadata (setup.py) ... done
Collecting phik<0.13,>=0.11.1 (from ydata-profiling->pandas-profiling)
  Downloading phik-0.12.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_
                                        679.5/679.5 kB 44.2 MB/s eta 0:00
Requirement already satisfied: requests<3,>=2.24.0 in /usr/local/lib/python3.
Requirement already satisfied: tqdm<5,>=4.48.2 in /usr/local/lib/python3.10/d
Requirement already satisfied: seaborn<0.13,>=0.10.1 in /usr/local/lib/python
Collecting multimethod<2,>=1.4 (from ydata-profiling->pandas-profiling)
  Downloading multimethod-1.9.1-py3-none-any.whl (10 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in /usr/local/lib/pytho
Collecting typeguard<3,>=2.13.2 (from ydata-profiling->pandas-profiling)
  Downloading typeguard-2.13.3-py3-none-any.whl (17 kB)
Collecting imagehash==4.3.1 (from ydata-profiling->pandas-profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 296.5/296.5 kB 28.9 MB/s eta 0:00
Collecting wordcloud>=1.9.1 (from ydata-profiling->pandas-profiling)
  Downloading wordcloud-1.9.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 455.4/455.4 kB 41.2 MB/s eta 0:00
Collecting dacite>=1.8 (from ydata-profiling->pandas-profiling)
  Downloading dacite-1.8.1-py3-none-any.whl (14 kB)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist
Collecting tangled-up-in-unicode>=0.0.4 (from visions[type_image_path]==0.7.5-
  Downloading tangled_up_in_unicode-0.2.0-py3-none-any.whl (4.7 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.7/4.7 MB 84.2 MB/s eta 0:00:0
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/pyth
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/pyt
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB, GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, f1_score,

from sklearn.metrics import classification_report

from sklearn.pipeline import make_pipeline



# Models
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import RidgeClassifier, LogisticRegression

from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import GradientBoostingClassifier



from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.metrics import roc_curve, auc

(Asad and Parkash)
```

image.png

```
client = pd.read_csv('client_data.csv')
client.head()

(Asad)
```

	id	channel_sales	cons_12m	cons
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	
3	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	
4	149d57cf92fc41cf94415803a877cb4b	MISSING	4425	

5 rows × 26 columns

```
df = client
df.head()
```

(Asad)

	id	channel_sales	cons_12m	cons_
0	24011ae4ebbe3035111d65fa7c15bc57	foosdfpfkusacimwkcsosbicdxkicaua	0	
1	d29c2c54acc38ff3c0614d0a653813dd	MISSING	4660	
2	764c75f661154dac3a6c254cd082ea7d	foosdfpfkusacimwkcsosbicdxkicaua	544	
3	bba03439a292a1e166f80264c16191cb	lmkebamcaaclubfxadlmueccxoimlema	1584	
4	149d57cf92fc41cf94415803a877cb4b	MISSING	4425	

5 rows × 26 columns

```
df = df[['cons_12m', 'cons_gas_12m', 'cons_last_month',
       'forecast_cons_12m', 'forecast_cons_year', 'forecast_discount_energy',
       'forecast_meter_rent_12m', 'forecast_price_energy_off_peak',
       'forecast_price_energy_peak', 'forecast_price_pow_off_peak', 'has_gas',
       'imp_cons', 'margin_gross_pow_ele', 'margin_net_pow_ele', 'nb_prod_act',
       'net_margin', 'num_years_antig', 'pow_max', 'churn']]
```

(Asad)

```
df.has_gas = df.has_gas.map({'t' : 0, 'f' : 1})
```

(Asad)

```
df.shape
```

(Asad)

```
(14606, 19)
```

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

(Asad)

```
cons_12m                          0
cons_gas_12m                      0
cons_last_month                   0
forecast_cons_12m                 0
forecast_cons_year                0
forecast_discount_energy          0
forecast_meter_rent_12m           0
forecast_price_energy_off_peak    0
forecast_price_energy_peak        0
forecast_price_pow_off_peak       0
has_gas                           0
imp_cons                          0
margin_gross_pow_ele              0
margin_net_pow_ele                0
nb_prod_act                       0
net_margin                        0
num_years_antig                   0
pow_max                           0
churn                             0
dtype: int64
```

```
df.info()
```

(Asad)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14606 entries, 0 to 14605
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   cons_12m                        14606 non-null  int64  
 1   cons_gas_12m                    14606 non-null  int64  
 2   cons_last_month                 14606 non-null  int64  
 3   forecast_cons_12m               14606 non-null  float64
 4   forecast_cons_year              14606 non-null  int64  
 5   forecast_discount_energy        14606 non-null  float64
 6   forecast_meter_rent_12m         14606 non-null  float64
 7   forecast_price_energy_off_peak  14606 non-null  float64
 8   forecast_price_energy_peak      14606 non-null  float64
 9   forecast_price_pow_off_peak     14606 non-null  float64
 10  has_gas                         14606 non-null  int64  
 11  imp_cons                        14606 non-null  float64
 12  margin_gross_pow_ele            14606 non-null  float64
 13  margin_net_pow_ele              14606 non-null  float64
 14  nb_prod_act                     14606 non-null  int64  
 15  net_margin                      14606 non-null  float64
 16  num_years_antig                 14606 non-null  int64  
 17  pow_max                         14606 non-null  float64
 18  churn                           14606 non-null  int64  
dtypes: float64(11), int64(8)
memory usage: 2.1 MB
```

```
df.describe().T

(Asad)
```

	count	mean	std	min	25%
cons_12m	14606.0	159220.286252	573465.264198	0.0	5674.750000
cons_gas_12m	14606.0	28092.375325	162973.059057	0.0	0.000000
cons_last_month	14606.0	16090.269752	64364.196422	0.0	0.000000
forecast_cons_12m	14606.0	1868.614880	2387.571531	0.0	494.995000
forecast_cons_year	14606.0	1399.762906	3247.786255	0.0	0.000000
forecast_discount_energy	14606.0	0.966726	5.108289	0.0	0.000000
forecast_meter_rent_12m	14606.0	63.086871	66.165783	0.0	16.180000
forecast_price_energy_off_peak	14606.0	0.137283	0.024623	0.0	0.116340
forecast_price_energy_peak	14606.0	0.050491	0.049037	0.0	0.000000
forecast_price_pow_off_peak	14606.0	43.130056	4.485988	0.0	40.606701
has_gas	14606.0	0.818499	0.385446	0.0	1.000000
imp_cons	14606.0	152.786896	341.369366	0.0	0.000000
margin_gross_pow_ele	14606.0	24.565121	20.231172	0.0	14.280000
margin_net_pow_ele	14606.0	24.562517	20.230280	0.0	14.280000
nb_prod_act	14606.0	1.292346	0.709774	1.0	1.000000
net_margin	14606.0	189.264522	311.798130	0.0	50.712500
num_years_antig	14606.0	4.997809	1.611749	1.0	4.000000
pow_max	14606.0	18.135136	13.534743	3.3	12.500000
churn	14606.0	0.097152	0.296175	0.0	0.000000

```
df.head()
```

(Asad)

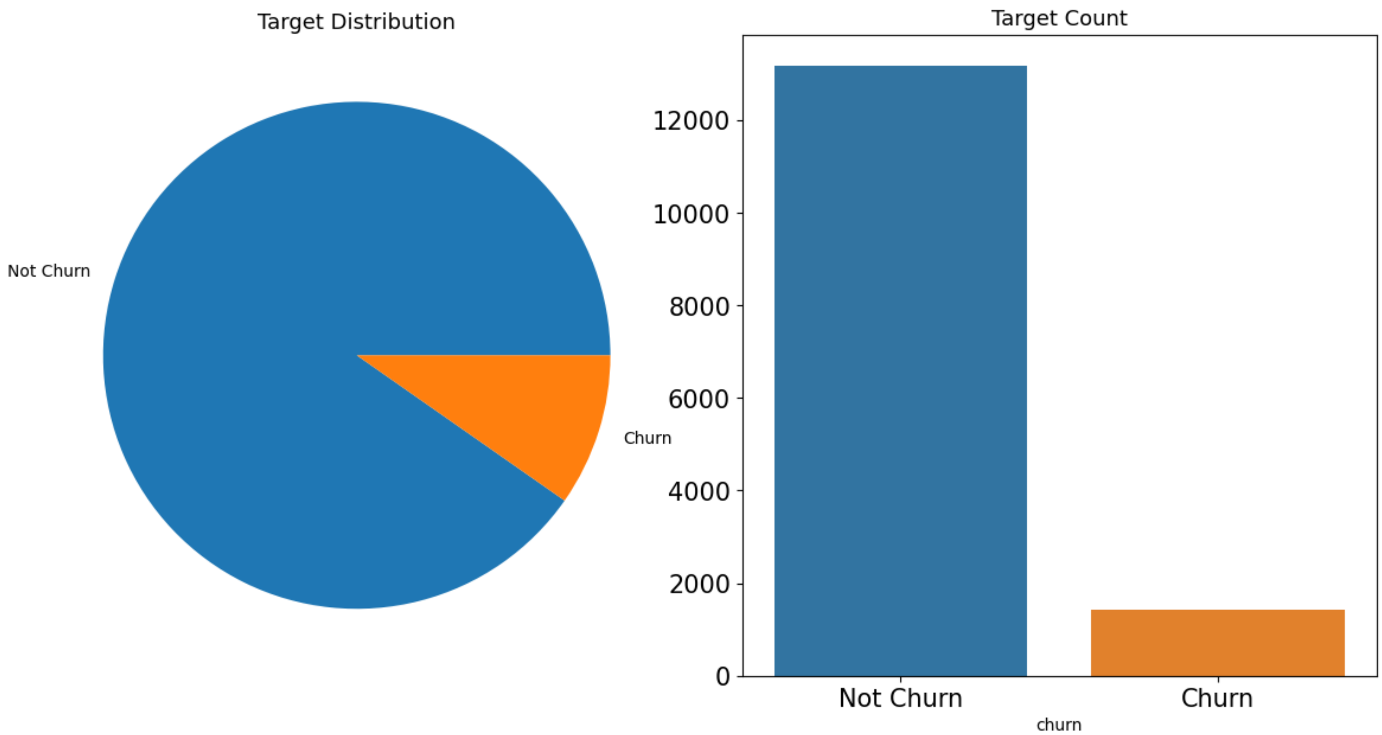
	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_ye
0	0	54946	0	0.00	
1	4660	0	0	189.95	
2	544	0	0	47.96	
3	1584	0	0	240.04	
4	4425	0	526	445.75	!

Comments:

The class variable *churn* is not balanced, so besides accuracy, we'll need to look at precision and recall as well. Sience churn has lower share, so our metric of interest is recall

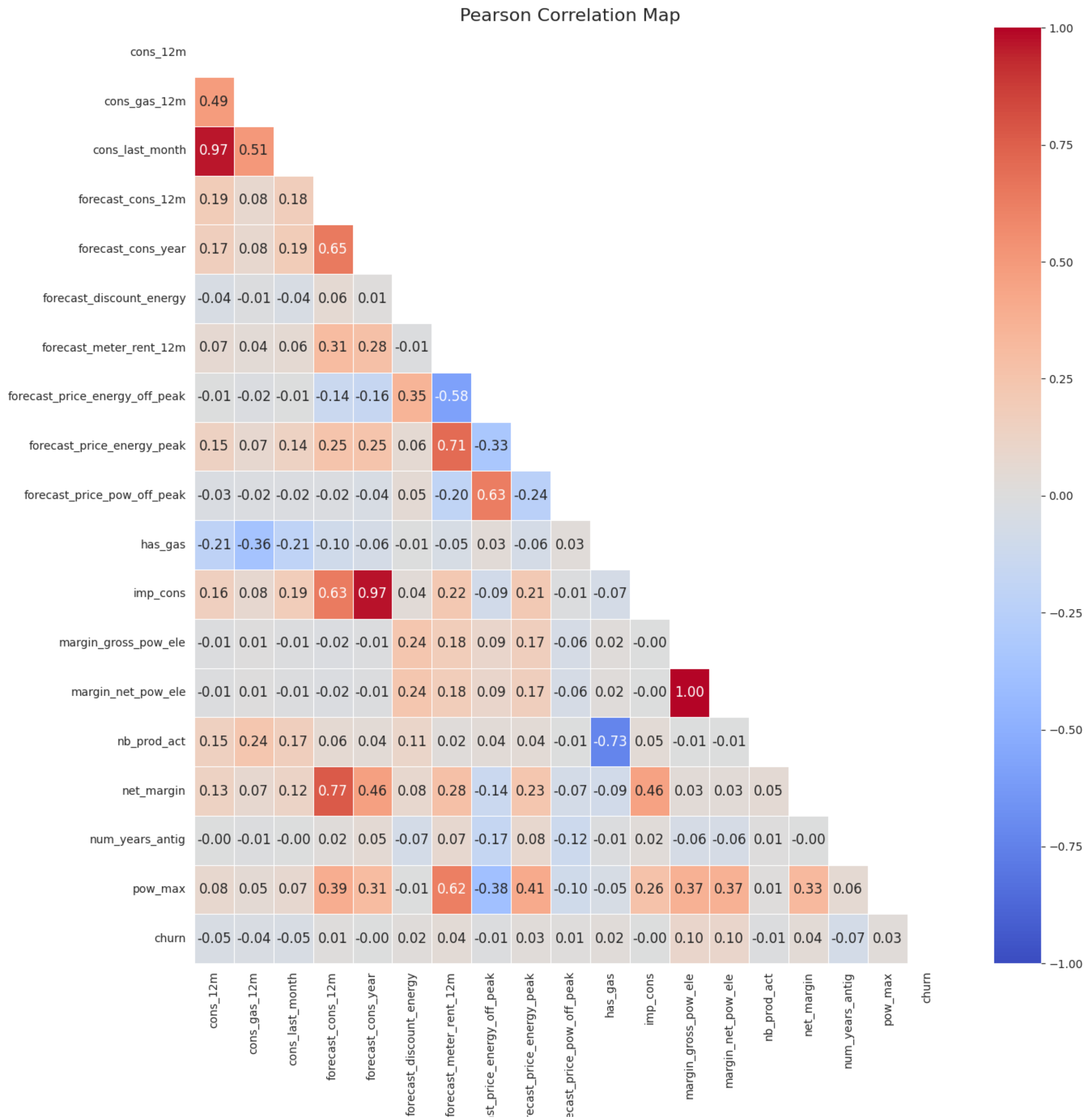
```
fig, axes  = plt.subplots(ncols = 2, figsize = (12, 6) , dpi = 100)
plt.tight_layout()

df.groupby('churn').count()['cons_12m'].plot(kind = 'pie', ax = axes[0], labels = |
sns.countplot(x = df['churn'], ax = axes[1])

axes[0].set_ylabel('')
axes[1].set_ylabel('')
axes[1].set_xticklabels(['Not Churn', 'Churn'])
axes[0].tick_params(axis='x', labelsize=15)
axes[0].tick_params(axis='y', labelsize=15)
axes[1].tick_params(axis='x', labelsize=15)
axes[1].tick_params(axis='y', labelsize=15)

axes[0].set_title('Target Distribution', fontsize=13)
axes[1].set_title('Target Count', fontsize=13)

plt.show()
```

(Asad)



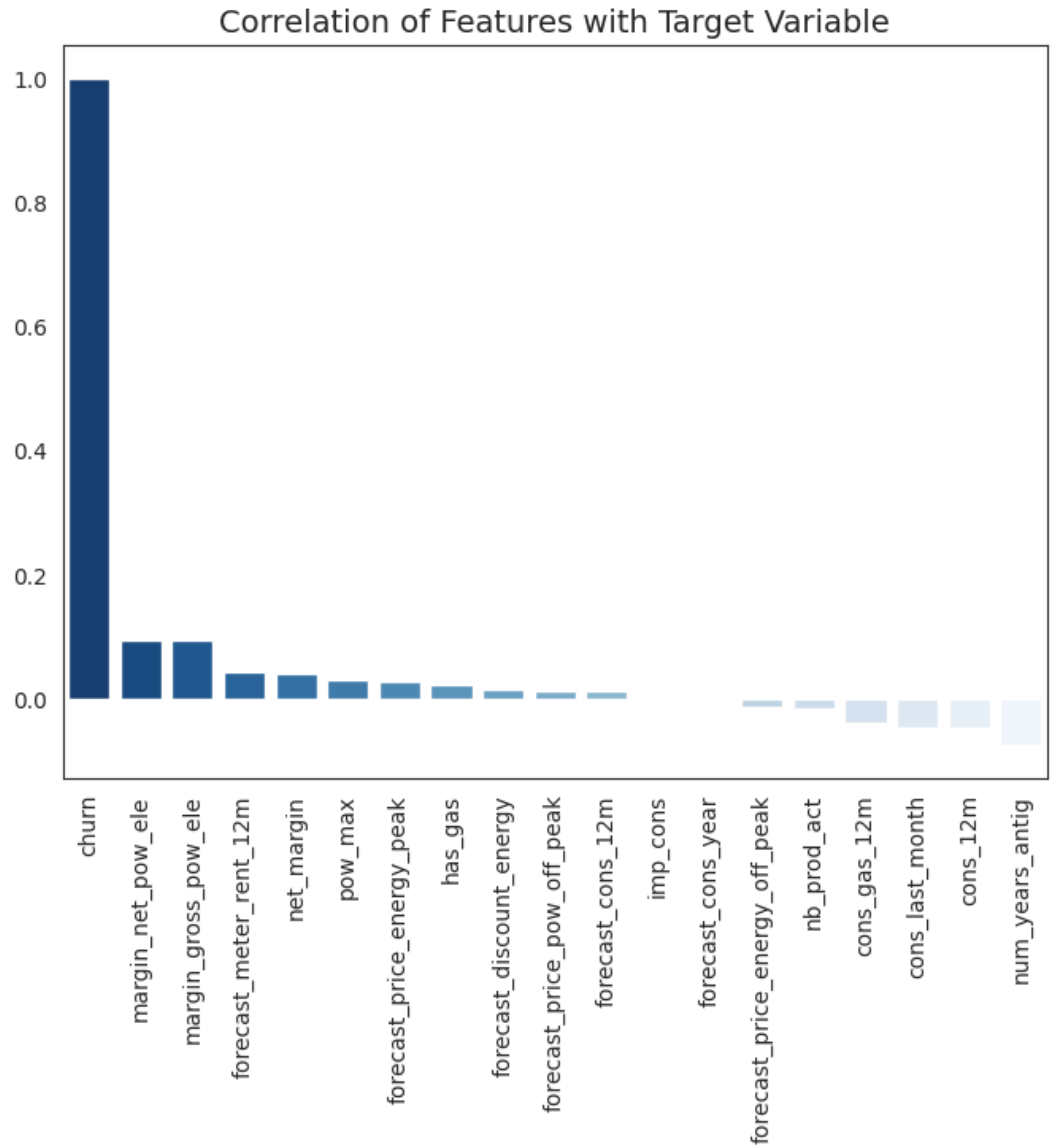

```
#Compute the Correlation matrix with all the features
sns.set_style("white")
corr_all = df.corr(method="pearson")
matrix = np.triu(corr_all)
fig, ax = plt.subplots(figsize=(15, 15))
axis_labels = df.columns
```

```
sns.heatmap(corr_all, xticklabels=axis_labels, yticklabels=axis_labels, annot=True,
            vmin=-1, vmax=1, mask=matrix, cmap="coolwarm", linewidth=0.4, linecolor
plt.xticks(rotation=90, size=10)
plt.yticks(rotation=0, size=10)
plt.title('Pearson Correlation Map', size=16)
plt.show()

(Asad)
```




```
# Compute the correlation matrix with the target variable
corr = df.corr()['churn']
corr_sort = corr.sort_values(ascending=False)
fig, ax = plt.subplots(figsize=(8, 6))
sns.barplot(x=corr_sort.index, y=corr_sort.values, palette="Blues_r")
plt.xticks(rotation=90, size=10)
plt.yticks(size=10)
plt.title('Correlation of Features with Target Variable', size=14)
plt.show()
```

(Asad)



Pandas Profiling: EDA

```
import pandas_profiling as pp
```

```
pp.ProfileReport(df)

(Asad)
```

Summarize dataset: 100% 317/317 [02:58<00:00, 1.10it/s, Completed]

Generate report structure: 100% 1/1 [00:19<00:00, 19.13s/it]

Render HTML: 100% 1/1 [00:13<00:00, 13.93s/it]

Overview

Dataset statistics

Number of variables	19
Number of observations	14606
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	2.1 MiB
Average record size in memory	152.0 B

Variable types

Numeric	17
Categorical	2

Categorical 2

Alerts

`cons_12m` is highly overall correlated with `cons_last_month` and 2 other fields (cons_last_month, forecast_cons_12m, net_margin)	**High correlation**
`cons_gas_12m` is highly overall correlated with `nb_prod_act`	**High correlation**

```
X = df.drop('churn', axis = 1)
y = df['churn']

(Parkash)
```

Normalising the data

```
cols = X.columns
std = StandardScaler()
X = std.fit_transform(X)
X = pd.DataFrame(data = X, columns = cols)
X.head()

(Parkash)
```

	cons_12m	cons_gas_12m	cons_last_month	forecast_cons_12m	forecast_cons_ye
0	-0.277655	0.164779	-0.249996	-0.782669	-0.431(
1	-0.269529	-0.172380	-0.249996	-0.703109	-0.431(
2	-0.276707	-0.172380	-0.249996	-0.762581	-0.431(
3	-0.274893	-0.172380	-0.249996	-0.682129	-0.431(
4	-0.269939	-0.172380	-0.241824	-0.595967	-0.269(

Splitting data into training and testing data with 80-20 rule.

```
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size = 0.20, random_stat
```

(Parkash)

```
xtrain.shape, xtest.shape, ytrain.shape, ytest.shape
```

(Parkash)

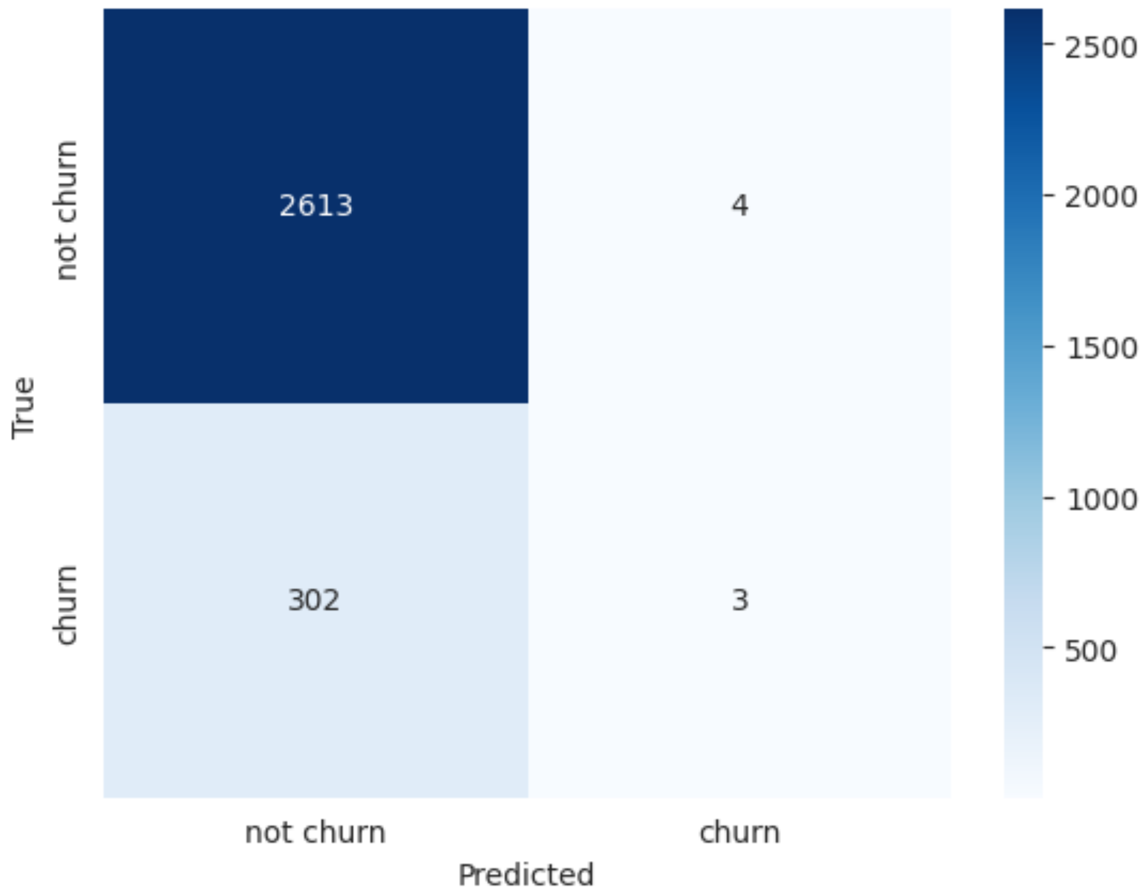
```
((11684, 18), (2922, 18), (11684,), (2922,))
```

Logistic Regression

```
target_names = ['not churn', 'churn']
lr = LogisticRegression()

lr.fit(xtrain,ytrain)

y_pred = lr.predict(xtest)

print(classification_report(ytest, y_pred, target_names=target_names))

LogisticAccuracy = 100*accuracy_score(y_pred, ytest)
LogisticPrecision = 100*precision_score(y_pred, ytest)
LogisticRecall = 100*recall_score(y_pred, ytest)
LogisticF1  = 100*f1_score(y_pred, ytest)

print("Accuracy score on test set:", LogisticAccuracy)
print("Precision score on test set:", LogisticPrecision)
print("Recall score on test set:", LogisticRecall)
print("F1 score on test set:", LogisticF1)


cm = confusion_matrix(ytest, y_pred)

sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=target_names, ytickl
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

(Asad)

```
              precision    recall  f1-score   support

   not churn       0.90      1.00      0.94      2617
       churn       0.43      0.01      0.02       305

    accuracy                           0.90      2922
   macro avg       0.66      0.50      0.48      2922
weighted avg       0.85      0.90      0.85      2922

Accuracy score on test set: 89.5277207392197
Precision score on test set: 0.9836065573770493
Recall score on test set: 42.857142857142854
F1 score on test set: 1.9230769230769231
```



KNN

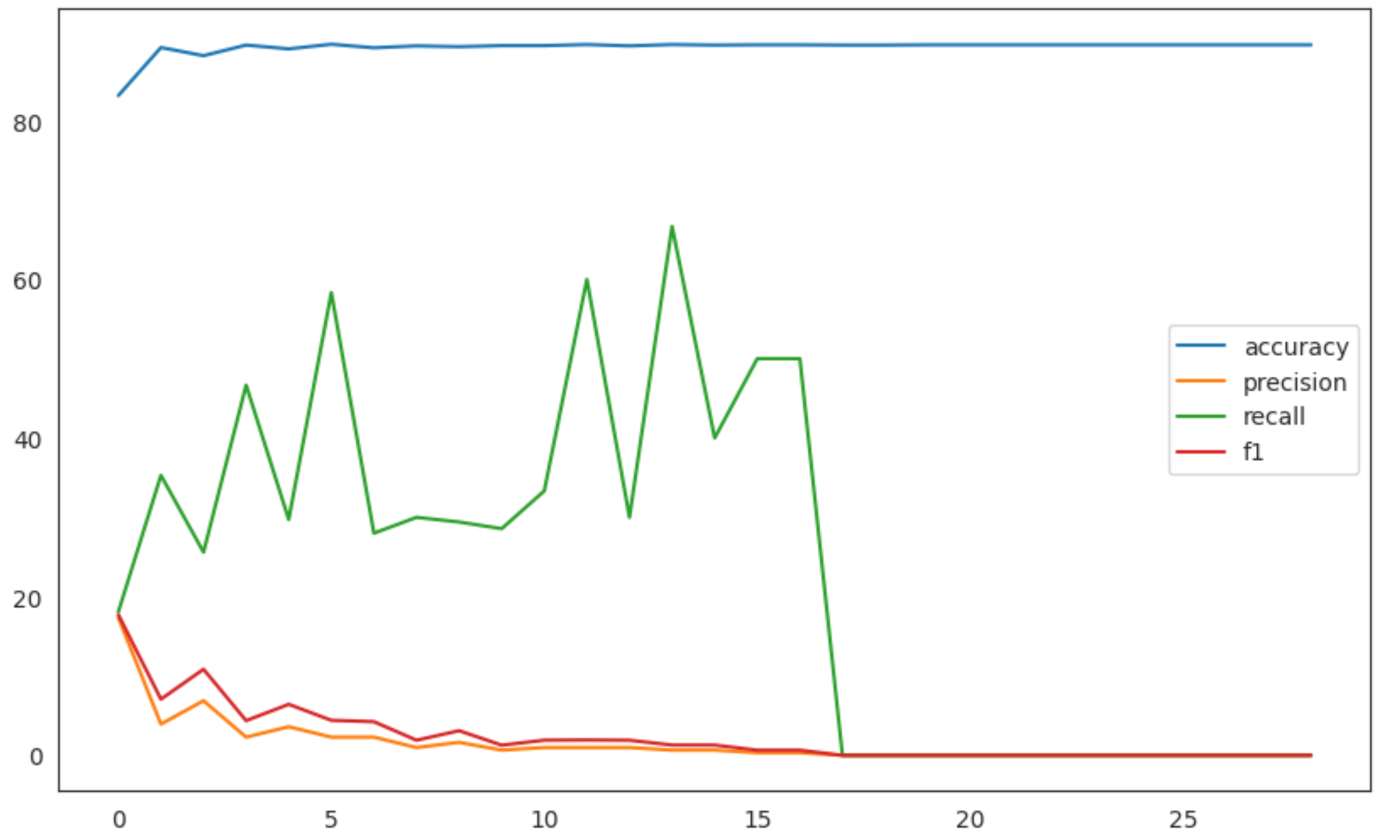
```
accuracy_list = []
precision_list = []
recall_list = []
f1_list = []
models = []

for k in range(1, 30):

    knn = KNeighborsClassifier(n_neighbors=k)
    models.append(knn)
    knn.fit(xtrain, ytrain)

    y_pred = knn.predict(xtest)

    accuracy = 100*accuracy_score(y_pred, ytest)
    precision = 100*precision_score(y_pred, ytest)
    recall = 100*recall_score(y_pred, ytest)
    f1  = 100*f1_score(y_pred, ytest)

    accuracy_list.append(accuracy)
    precision_list.append(precision)
    recall_list.append(recall)
    f1_list.append(f1)


(Parkash)
```

Comments

We see that as k increases, all the metrics decrease.

```
plt.figure(figsize=(10,6))
plt.plot(accuracy_list, label = 'accuracy')
plt.plot(precision_list, label = 'precision')
plt.plot(recall_list, label = 'recall')
plt.plot(f1_list, label = 'f1')
plt.legend()
plt.show()
```

(Parkash)

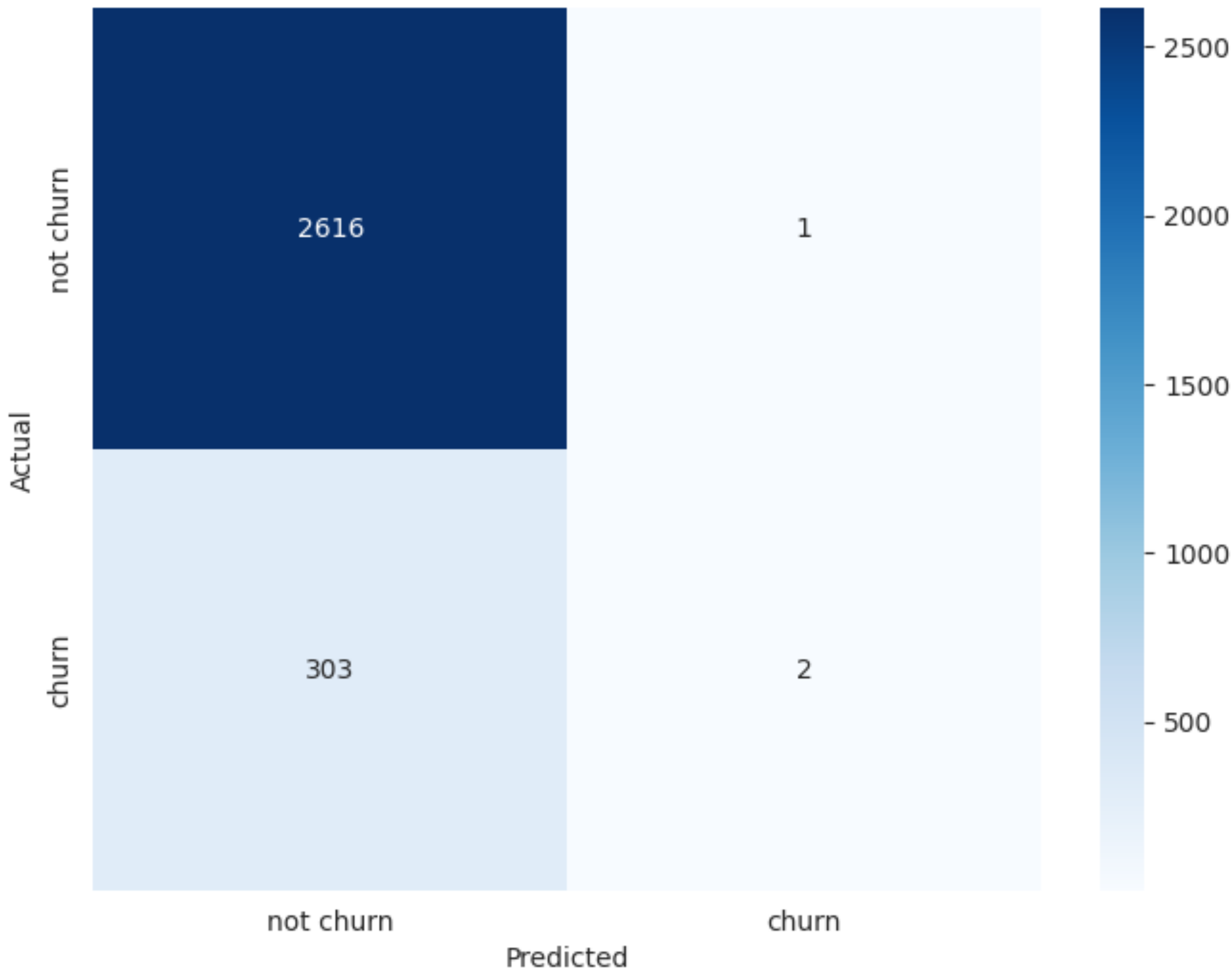



```
knn = KNeighborsClassifier(n_neighbors=14)
models.append(knn)
knn.fit(xtrain, ytrain)

y_pred = knn.predict(xtest)
```

```
print(classification_report(ytest, y_pred, target_names=target_names))


KnnAccuracy = 100*accuracy_score(y_pred, ytest)
KnnPrecision = 100*precision_score(y_pred, ytest)
KnnRecall = 100*recall_score(y_pred, ytest)
KnnF1  = 100*f1_score(y_pred, ytest)

print("Accuracy score on test set:", KnnAccuracy)
print("Precision score on test set:", KnnPrecision)
print("Recall score on test set:", KnnRecall)
print("F1 score on test set:", KnnF1)

cm = confusion_matrix(ytest, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=target_names, ytickl
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

(Parkash)

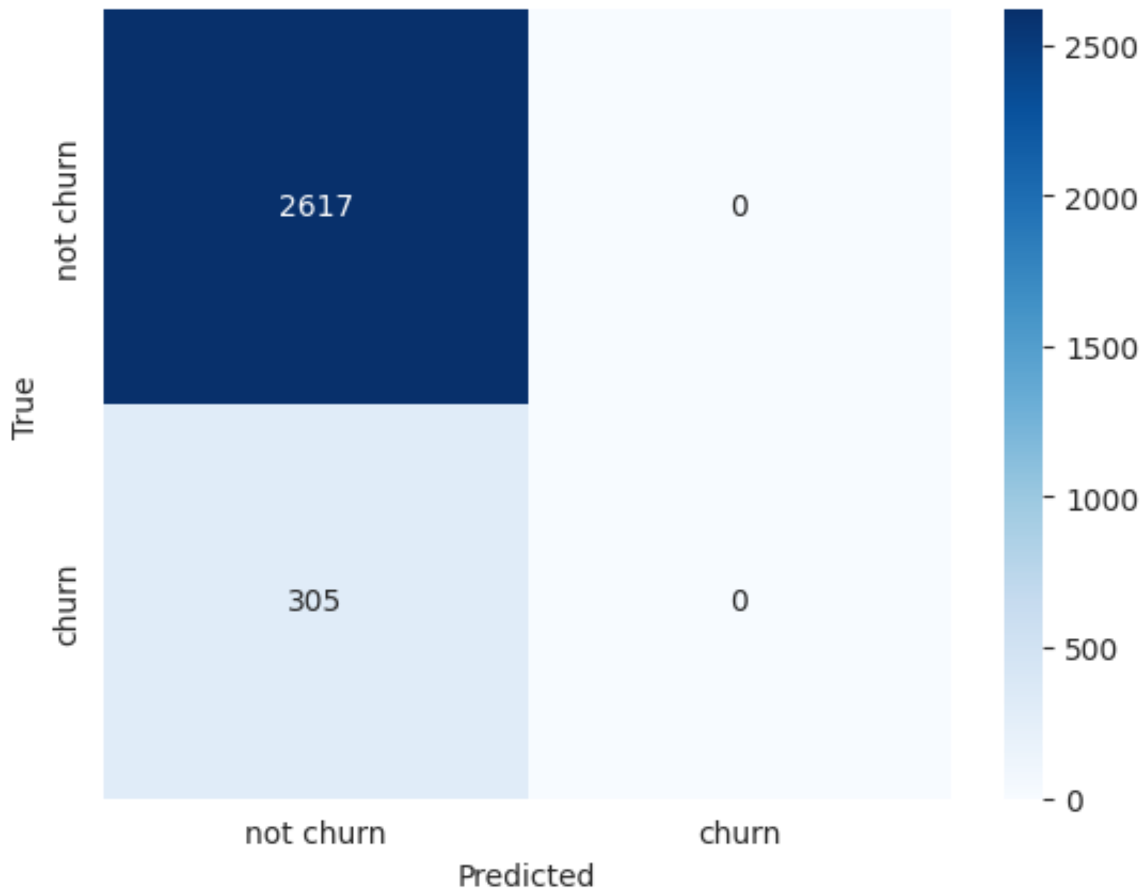
```
              precision    recall  f1-score   support

   not churn       0.90      1.00      0.95      2617
       churn       0.67      0.01      0.01       305

    accuracy                           0.90      2922
   macro avg       0.78      0.50      0.48      2922
weighted avg       0.87      0.90      0.85      2922

Accuracy score on test set: 89.59616700889802
Precision score on test set: 0.6557377049180327
Recall score on test set: 66.66666666666666
F1 score on test set: 1.2987012987012987
```



SVM

```
svc = SVC()
svc.fit(xtrain,ytrain)
y_pred = svc.predict(xtest)


print(classification_report(ytest, y_pred, target_names=target_names))

print("Accuracy score on test set:", accuracy)

SVMAccuracy = 100*accuracy_score(y_pred, ytest)
SVMPrecision = 100*precision_score(y_pred, ytest)
SVMRecall = 100*recall_score(y_pred, ytest)
SVMF1  = 100*f1_score(y_pred, ytest)

print("Accuracy score on test set:", SVMAccuracy)
print("Precision score on test set:", SVMPrecision)
print("Recall score on test set:", SVMRecall)
print("F1 score on test set:", SVMF1)

cm = confusion_matrix(ytest, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=target_names, ytickl
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()


(Parkash)
```

```
              precision    recall  f1-score   support

   not churn       0.90      1.00      0.94      2617
       churn       0.00      0.00      0.00       305

    accuracy                           0.90      2922
   macro avg       0.45      0.50      0.47      2922
weighted avg       0.80      0.90      0.85      2922

Accuracy score on test set: 89.56194387405887
Accuracy score on test set: 89.56194387405887
Precision score on test set: 0.0
Recall score on test set: 0.0
F1 score on test set: 0.0
```

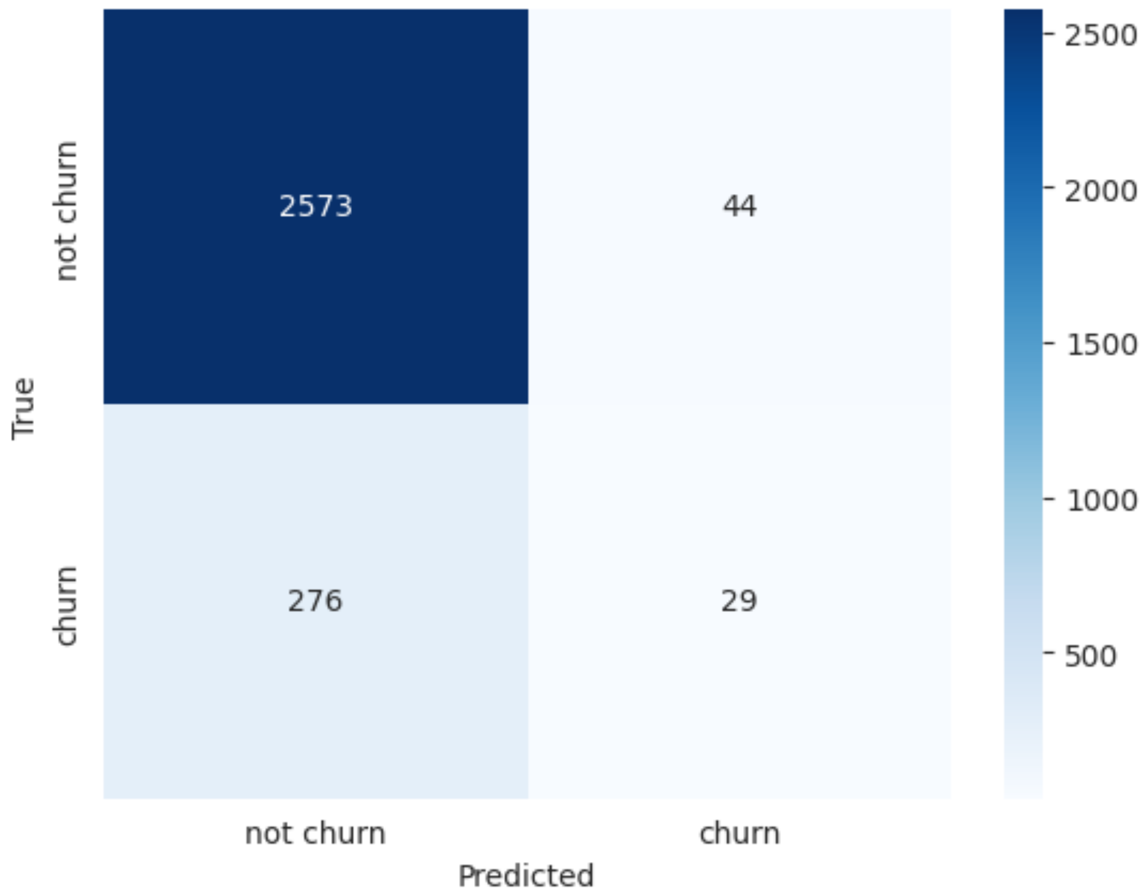


Decision Tree

```
dt = DecisionTreeClassifier(random_state = 4, max_depth=10)
dt.fit(xtrain, ytrain)
```

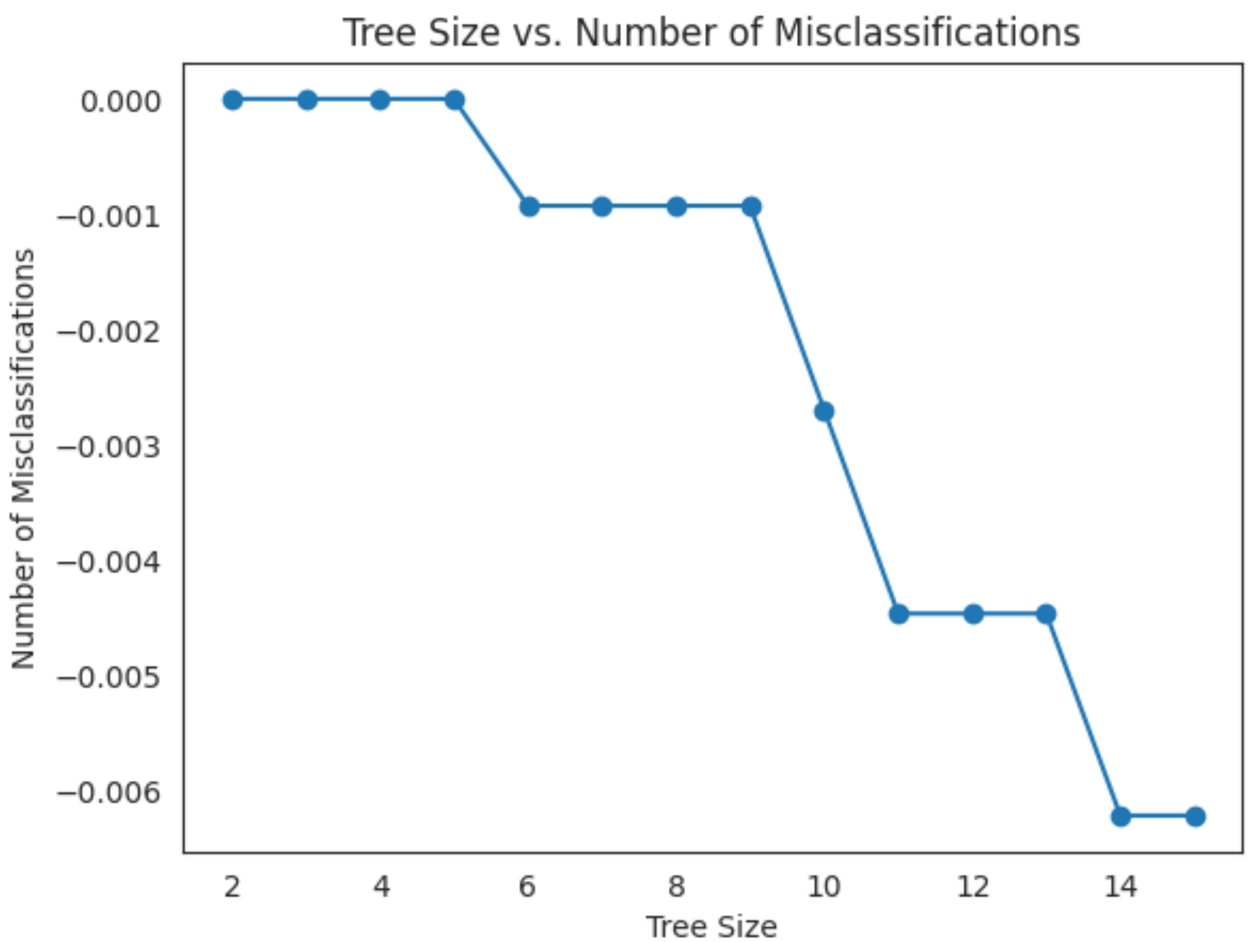
```
y_pred = dt.predict(xtest)

print(classification_report(ytest, y_pred, target_names=target_names))

print("Accuracy score on train set:", accuracy)

DTAccuracy = 100*accuracy_score(y_pred, ytest)
DTPrecision = 100*precision_score(y_pred, ytest)
DTRecall = 100*recall_score(y_pred, ytest)
DTF1  = 100*f1_score(y_pred, ytest)

print("Accuracy score on test set:", DTAccuracy)
print("Precision score on test set:", DTPrecision)
print("Recall score on test set:", DTRecall)
print("F1 score on test set:", DTF1)

cm = confusion_matrix(ytest, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=target_names, ytickl
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

(Asad)
```

```
              precision    recall  f1-score   support

   not churn       0.90      0.98      0.94      2617
       churn       0.40      0.10      0.15       305

    accuracy                           0.89      2922
   macro avg       0.65      0.54      0.55      2922
weighted avg       0.85      0.89      0.86      2922

Accuracy score on train set: 89.56194387405887
Accuracy score on test set: 89.0485968514716
Precision score on test set: 9.508196721311474
Recall score on test set: 39.726027397260275
F1 score on test set: 15.343915343915345
```




```
param_grid = {'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9,10,11, 12,13, 14, 15]}

kfold = KFold(n_splits=5, shuffle=True, random_state=4)
grid_search = GridSearchCV(DecisionTreeClassifier(random_state = 4), param_grid, cv
grid_search.fit(xtrain, ytrain)

tree_sizes = param_grid['max_leaf_nodes']
```

```
misclassifications = -grid_search.cv_results_['mean_test_score']

plt.plot(tree_sizes, misclassifications, marker='o')
plt.xlabel('Tree Size')
plt.ylabel('Number of Misclassifications')
plt.title('Tree Size vs. Number of Misclassifications')
plt.show()

min_misclassification = np.min(misclassifications)
best_tree_size = tree_sizes[np.argmin(misclassifications)]

print("Minimum number of misclassifications", min_misclassification)
print("Best tree size", best_tree_size)

(Asad)
```

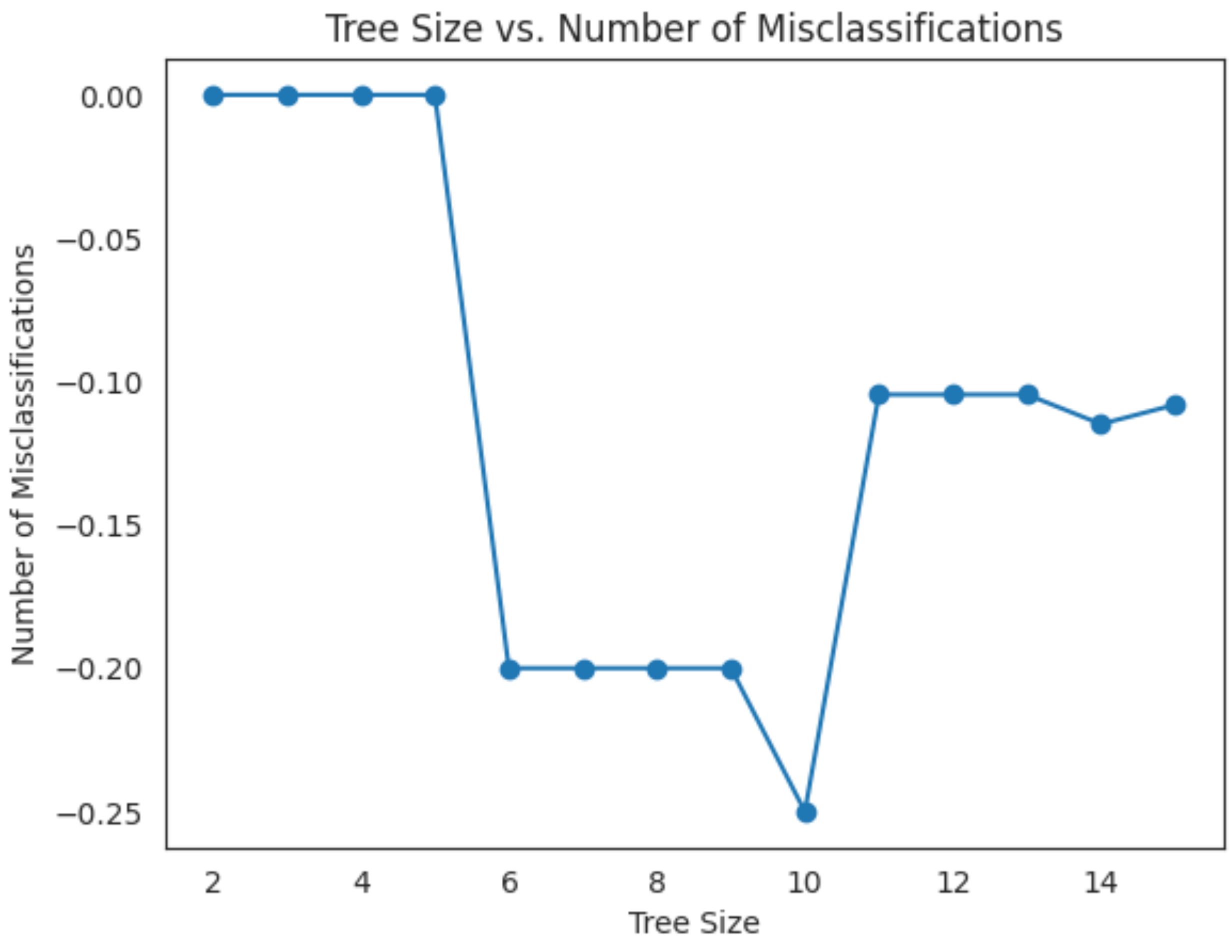



```
Minimum number of misclassifications -0.006207877090230031
Best tree size 14
```

```
param_grid = {'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9,10,11, 12,13, 14, 15]}
```

```
kfold = KFold(n_splits=5, shuffle=True, random_state=4)
grid_search = GridSearchCV(DecisionTreeClassifier(random_state = 4), param_grid, cv
grid_search.fit(xtrain, ytrain)

tree_sizes = param_grid['max_leaf_nodes']
misclassifications = -grid_search.cv_results_['mean_test_score']

plt.plot(tree_sizes, misclassifications, marker='o')
plt.xlabel('Tree Size')
plt.ylabel('Number of Misclassifications')
plt.title('Tree Size vs. Number of Misclassifications')
plt.show()

min_misclassification = np.min(misclassifications)
best_tree_size = tree_sizes[np.argmin(misclassifications)]

print("Minimum number of misclassifications", min_misclassification)
print("Best tree size", best_tree_size)
```

(Asad)




```
Minimum number of misclassifications -0.25
Best tree size 10
```

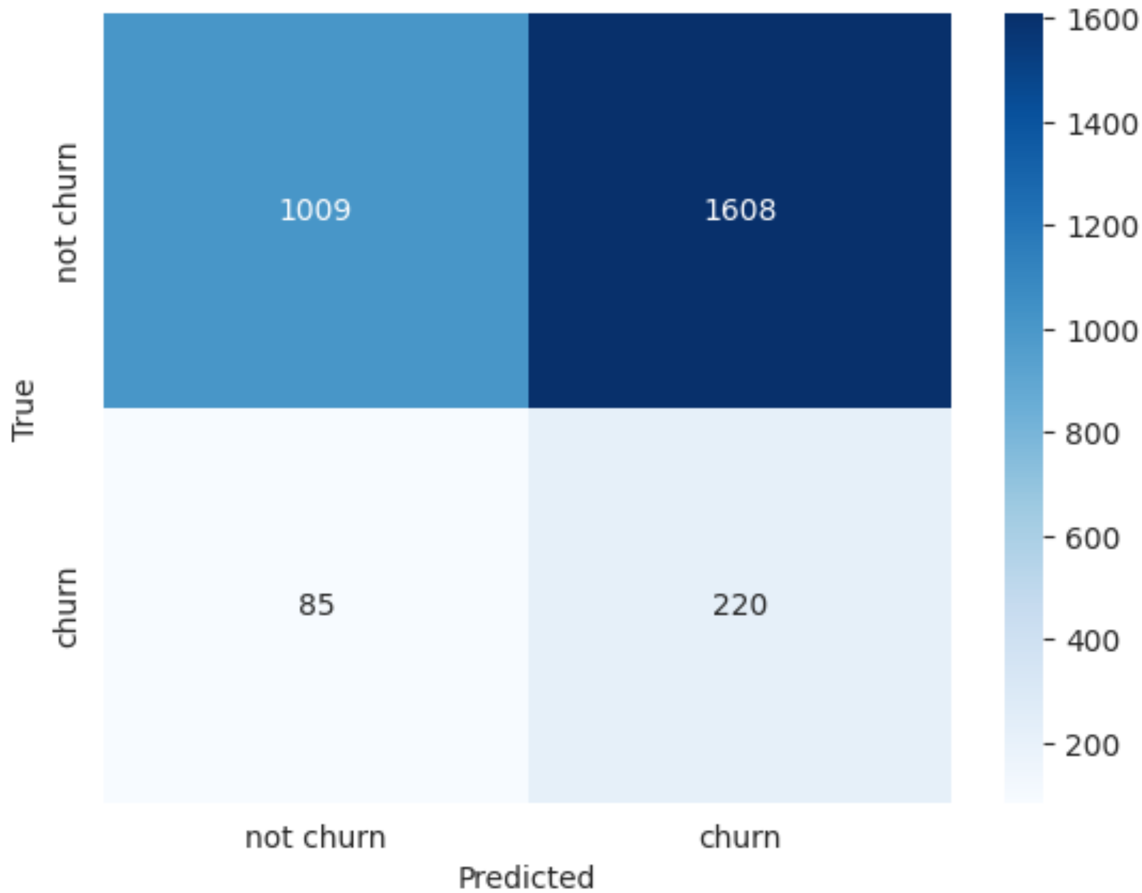
Gaussian Naive Bayes

```
gcla = GaussianNB()
gcla.fit(xtrain, ytrain)
y_pred = gcla.predict_proba(xtest)
y_pred = np.argmax(y_pred, axis = 1)

print(classification_report(ytest, y_pred, target_names=target_names))

print("Accuracy score on train set:", accuracy)

GNBAccuracy = 100*accuracy_score(y_pred, ytest)
GNBPrecision = 100*precision_score(y_pred, ytest)
GNBRecall = 100*recall_score(y_pred, ytest)
GNBF1  = 100*f1_score(y_pred, ytest)
```

```
print("Accuracy score on test set:", GNBAccuracy)
print("Precision score on test set:", GNBPrecision)
print("Recall score on test set:", GNBRecall)
print("F1 score on test set:", GNBF1)

cm = confusion_matrix(ytest, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=target_names, ytickl
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

(Asad)

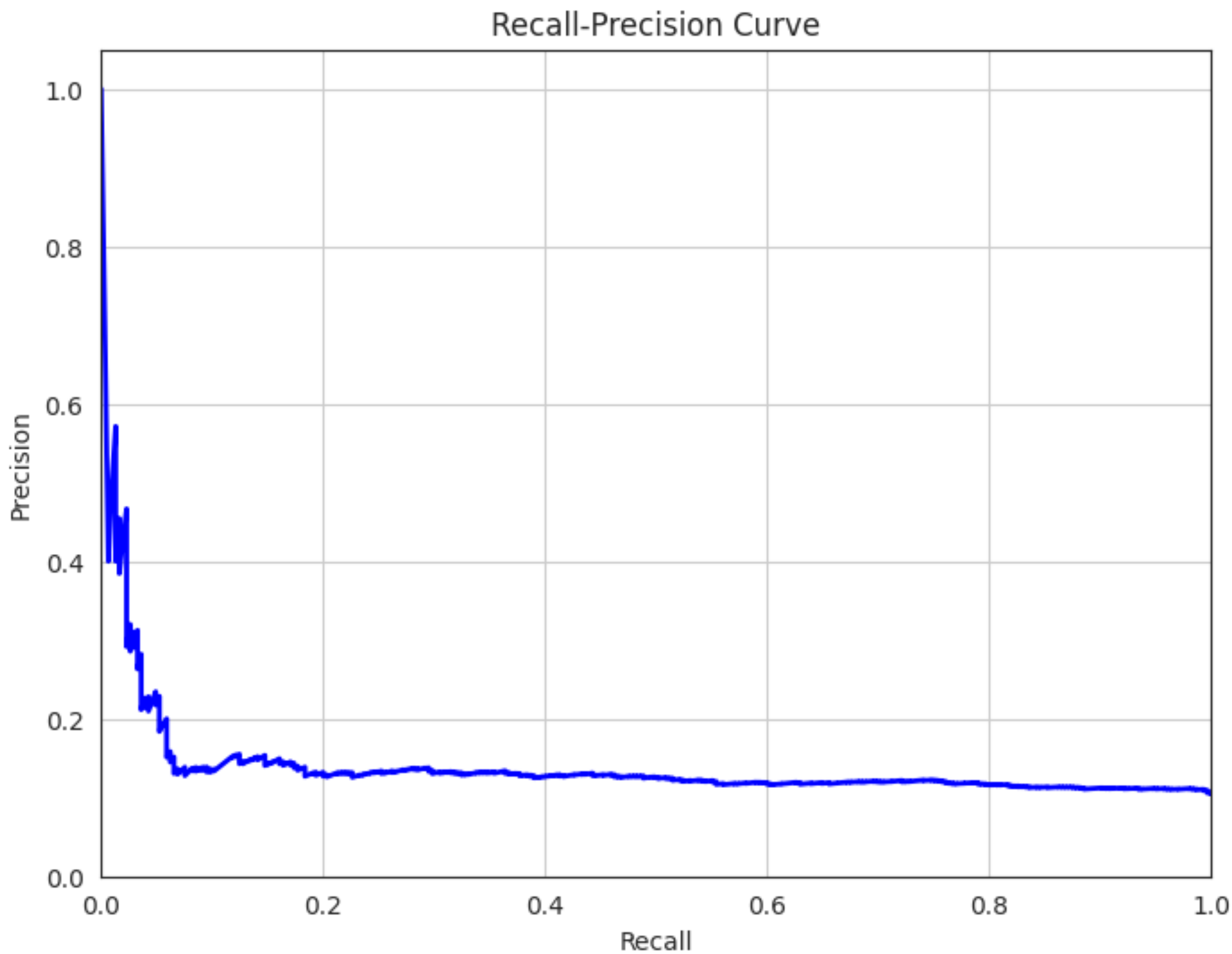
```
              precision    recall  f1-score   support

   not churn       0.92      0.39      0.54      2617
       churn       0.12      0.72      0.21       305

    accuracy                           0.42      2922
   macro avg       0.52      0.55      0.38      2922
weighted avg       0.84      0.42      0.51      2922

Accuracy score on train set: 89.56194387405887
Accuracy score on test set: 42.06023271731691
Precision score on test set: 72.1311475409836
Recall score on test set: 12.035010940919037
F1 score on test set: 20.628223159868732
```




```
from sklearn.metrics import precision_recall_curve

# Get predicted probabilities of positive class for test data
y_scores = gcla.predict_proba(xtest)[:, 1]

# Calculate precision and recall values
precision, recall, _ = precision_recall_curve(ytest, y_scores)
```

```
# Plot the recall-precision curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', linewidth=2)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Recall-Precision Curve')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.grid(True)
plt.show()
```

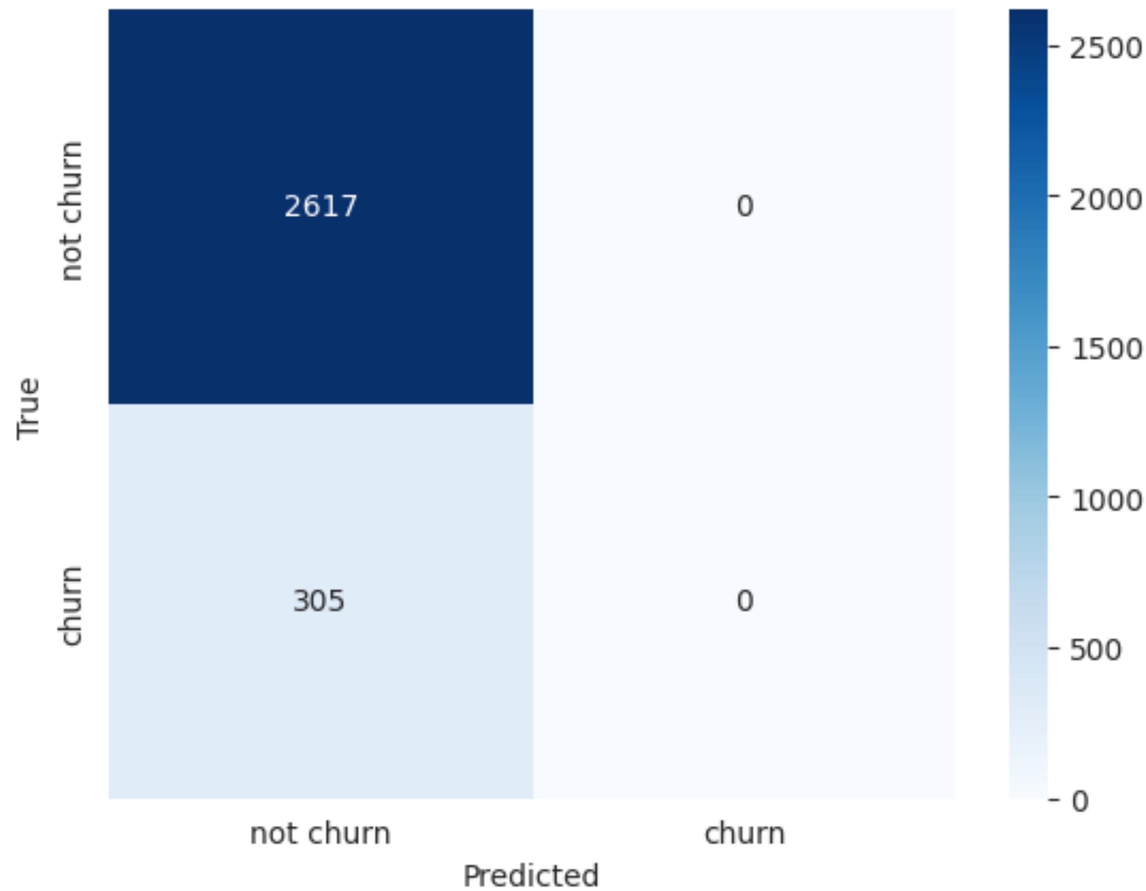
(Asad)



Gradient Boosting Classifier

```
gb = GradientBoostingClassifier(n_estimators=20)
gb.fit(xtrain,ytrain)
y_pred = gb.predict(xtest)
print(classification_report(ytest, y_pred, target_names=target_names))

print("Accuracy score on test set:", accuracy)

GBAccuracy = 100*accuracy_score(y_pred, ytest)
GBPrecision = 100*precision_score(y_pred, ytest)
GBRecall = 100*recall_score(y_pred, ytest)
GBF1  = 100*f1_score(y_pred, ytest)

print("Accuracy score on test set:", GBAccuracy)
print("Precision score on test set:", GBPrecision)
print("Recall score on test set:", GBRecall)
print("F1 score on test set:", GBF1)

cm = confusion_matrix(ytest, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=target_names, ytickl
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()


(Parkash)
```

```
              precision    recall  f1-score   support

   not churn       0.90      1.00      0.94      2617
       churn       0.00      0.00      0.00       305

    accuracy                           0.90      2922
   macro avg       0.45      0.50      0.47      2922
weighted avg       0.80      0.90      0.85      2922

Accuracy score on test set: 89.56194387405887
Accuracy score on test set: 89.56194387405887
Precision score on test set: 0.0
Recall score on test set: 0.0
F1 score on test set: 0.0
```

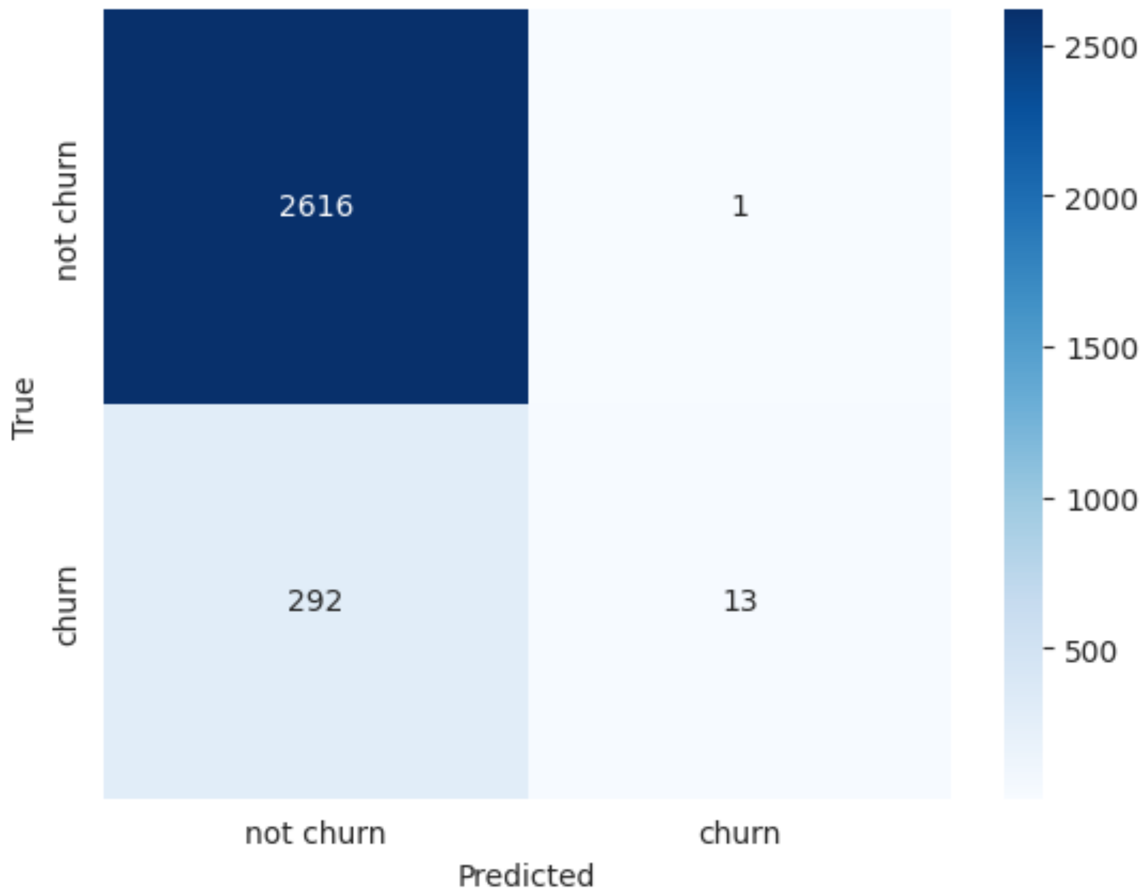


Random Forest

```
rf = RandomForestClassifier(random_state = 4)
rf.fit(xtrain,ytrain)
y_pred = rf.predict(xtest)
```

```
print(classification_report(ytest, y_pred, target_names=target_names))

print("Accuracy score on test set:", accuracy)

RFAccuracy = 100*accuracy_score(y_pred, ytest)
RFPrecision = 100*precision_score(y_pred, ytest)
RFRecall = 100*recall_score(y_pred, ytest)
RFF1  = 100*f1_score(y_pred, ytest)

print("Accuracy score on test set:", RFAccuracy)
print("Precision score on test set:", RFPrecision)
print("Recall score on test set:", RFRecall)
print("F1 score on test set:", RFF1)

cm = confusion_matrix(ytest, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=target_names, ytickl
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()


(Parkash)
```

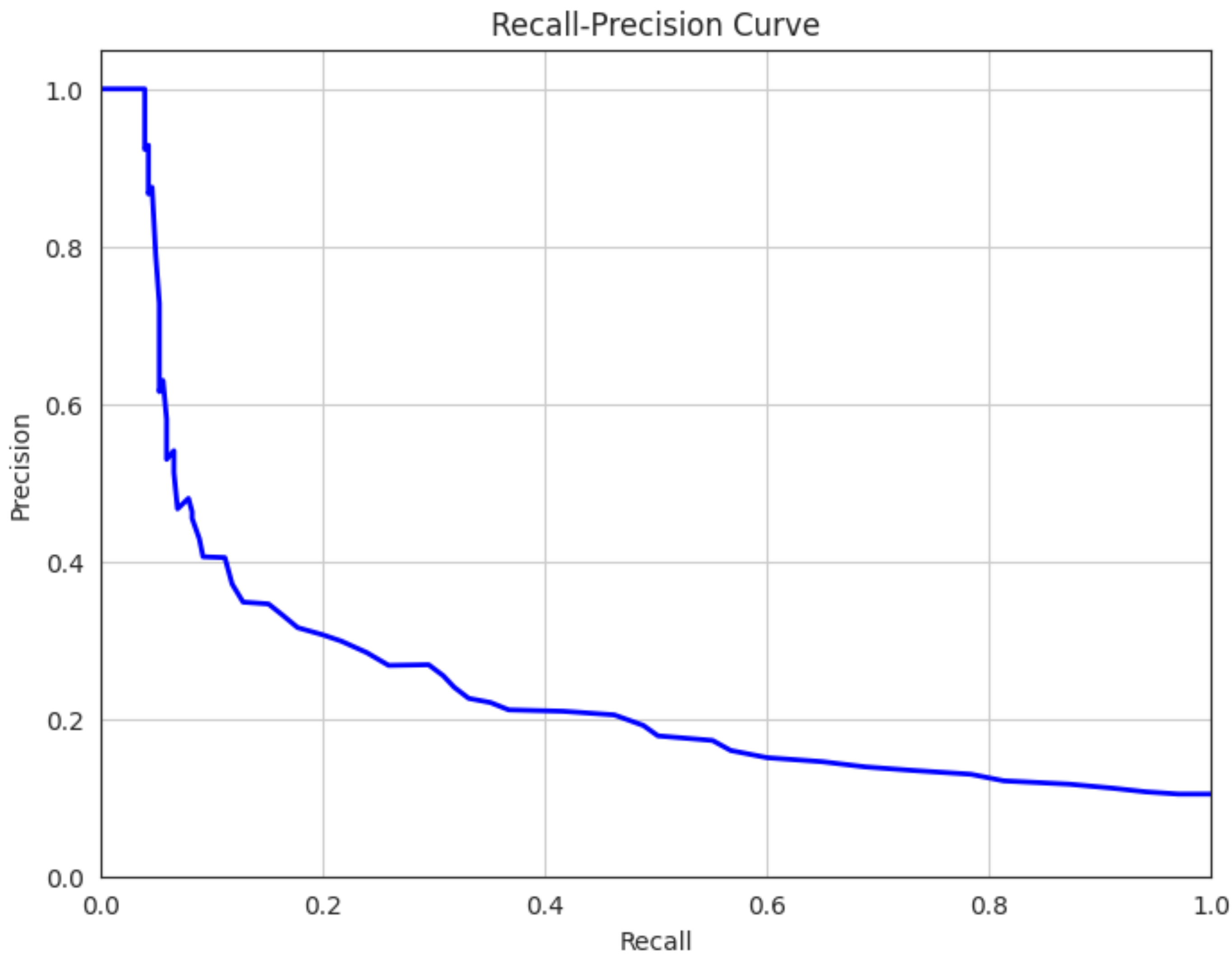
```
              precision    recall  f1-score   support

   not churn       0.90      1.00      0.95      2617
       churn       0.93      0.04      0.08       305

    accuracy                           0.90      2922
   macro avg       0.91      0.52      0.51      2922
weighted avg       0.90      0.90      0.86      2922

Accuracy score on test set: 89.56194387405887
Accuracy score on test set: 89.97262149212868
Precision score on test set: 4.2622950819672125
Recall score on test set: 92.85714285714286
F1 score on test set: 8.150470219435736
```




```
from sklearn.metrics import precision_recall_curve

# Get predicted probabilities of positive class for test data
y_scores = rf.predict_proba(xtest)[:, 1]

# Calculate precision and recall values
precision, recall, _ = precision_recall_curve(ytest, y_scores)
```

```
# Plot the recall-precision curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', linewidth=2)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Recall-Precision Curve')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.grid(True)
plt.show()
```

(Asad)



Plotting all metrics of all models togather

```
Accuracy_scores = [LogisticAccuracy, KnnAccuracy, SVMAccuracy, DTAccuracy, GNBAccur
Precision_scores = [LogisticPrecision, KnnPrecision, SVMPrecision, DTPrecision, GNE
Recall_scores = [LogisticRecall, KnnRecall, SVMRecall, DTRecall, GNBRecall, GBRecal
F1_scores = [LogisticF1, KnnF1, SVMF1, DTF1, GNBF1, GBF1, RFF1]
```

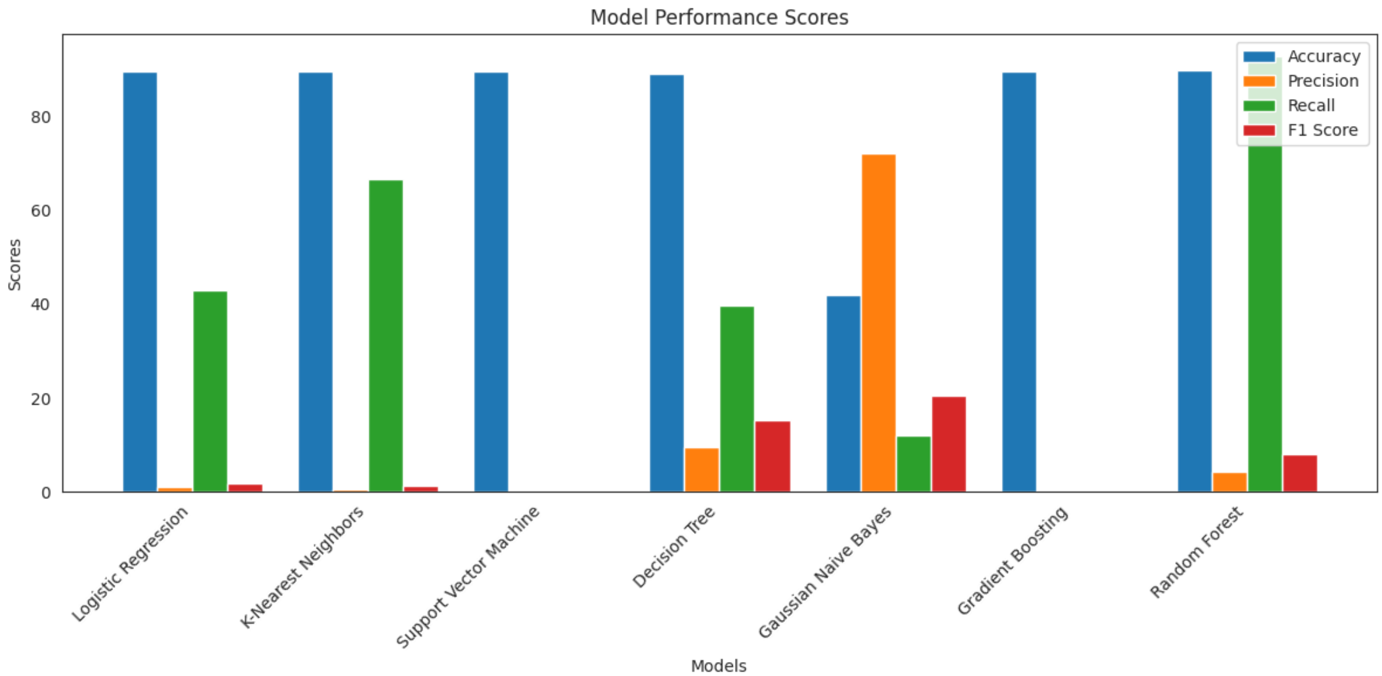
(Parkash)

```
import matplotlib.pyplot as plt

labels = ['Logistic Regression', 'K-Nearest Neighbors', 'Support Vector Machine', '
          'Gaussian Naive Bayes', 'Gradient Boosting', 'Random Forest']

legends = ['accuracy', 'precision', 'recall', 'f1']

# Create a figure and axis
plt.figure(figsize=(12, 6))
fig, ax = plt.subplots(figsize=(12, 6))

# Set the width of each bar
bar_width = 0.2

# Positions for the bars
x_positions = np.arange(len(labels))

# Plot the bars for each score type
ax.bar(x_positions - 1.5 * bar_width, Accuracy_scores, width=bar_width, label='Accu
ax.bar(x_positions - 0.5 * bar_width, Precision_scores, width=bar_width, label='Pre
ax.bar(x_positions + 0.5 * bar_width, Recall_scores, width=bar_width, label='Recall
ax.bar(x_positions + 1.5 * bar_width, F1_scores, width=bar_width, label='F1 Score')

# Set the x-axis ticks and labels
ax.set_xticks(x_positions)
ax.set_xticklabels(labels, rotation=45, ha='right')

# Set labels and title
ax.set_xlabel('Models')
ax.set_ylabel('Scores')
ax.set_title('Model Performance Scores')
```

```
# Add a legend
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```

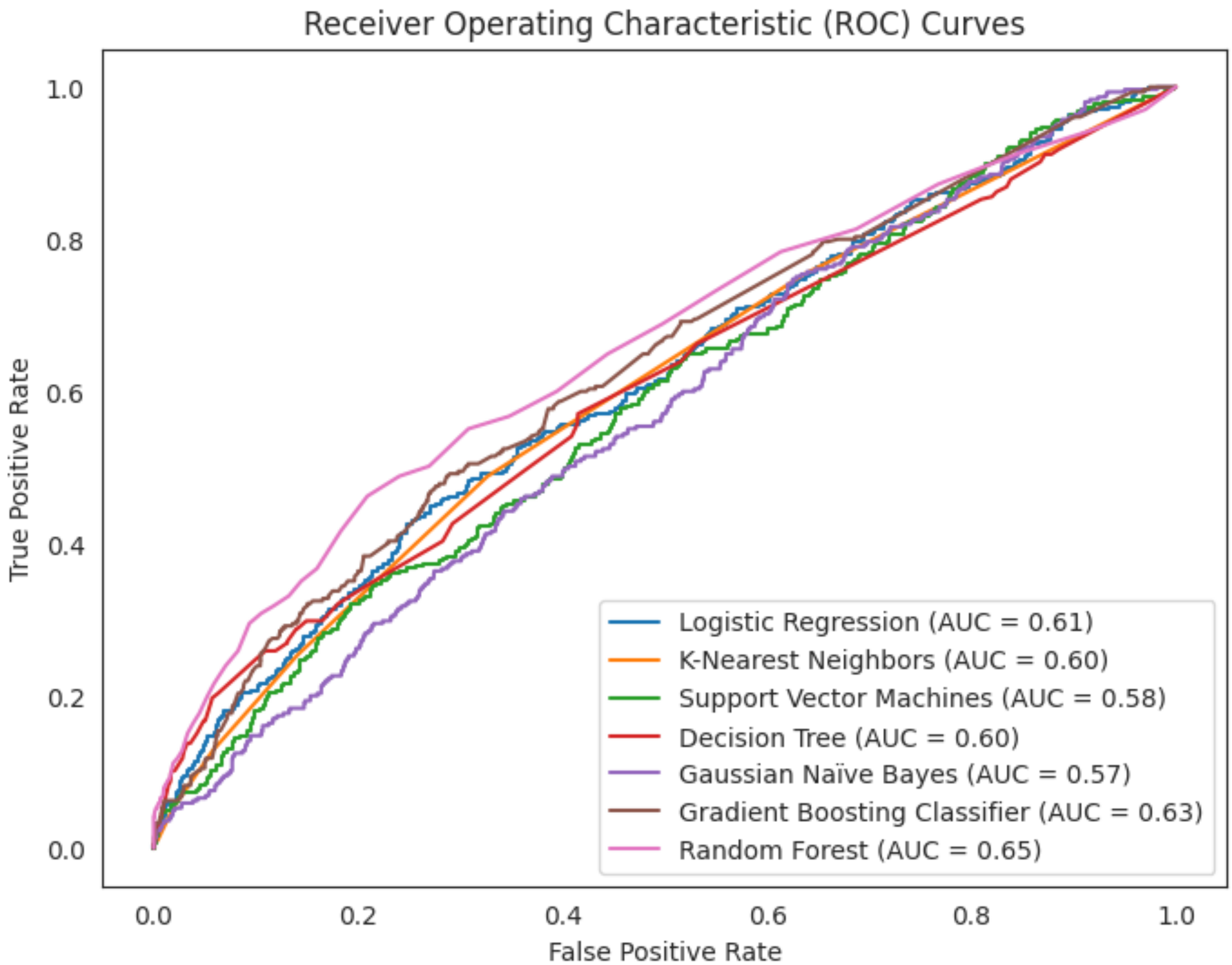
(Parkash)

```
<Figure size 1200x600 with 0 Axes>
```




```
y_pred_lr = lr.predict_proba(xtest)[:, 1]  # Logistic Regression
y_pred_knn = knn.predict_proba(xtest)[:, 1]  # K-Nearest Neighbors
```

```
y_pred_svm = svc.decision_function(xtest)  # Support Vector Machines
y_pred_dt = dt.predict_proba(xtest)[:, 1]  # Decision Tree
y_pred_gnb = gcla.predict_proba(xtest)[:, 1]  # Gaussian Naïve Bayes
y_pred_gbc = gb.predict_proba(xtest)[:, 1]  # Gradient Boosting Classifier
y_pred_rf = rf.predict_proba(xtest)[:, 1]  # Random Forest

# Compute FPR, TPR, and thresholds for each model
fpr_lr, tpr_lr, thresholds_lr = roc_curve(ytest, y_pred_lr)
fpr_knn, tpr_knn, thresholds_knn = roc_curve(ytest, y_pred_knn)
fpr_svm, tpr_svm, thresholds_svm = roc_curve(ytest, y_pred_svm)
fpr_dt, tpr_dt, thresholds_dt = roc_curve(ytest, y_pred_dt)
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(ytest, y_pred_gnb)
fpr_gbc, tpr_gbc, thresholds_gbc = roc_curve(ytest, y_pred_gbc)
fpr_rf, tpr_rf, thresholds_rf = roc_curve(ytest, y_pred_rf)

# Calculate AUC for each model
auc_lr = auc(fpr_lr, tpr_lr)
auc_knn = auc(fpr_knn, tpr_knn)
auc_svm = auc(fpr_svm, tpr_svm)
auc_dt = auc(fpr_dt, tpr_dt)
auc_gnb = auc(fpr_gnb, tpr_gnb)
auc_gbc = auc(fpr_gbc, tpr_gbc)
auc_rf = auc(fpr_rf, tpr_rf)

# Plot ROC curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, label='Logistic Regression (AUC = %0.2f)' % auc_lr)
plt.plot(fpr_knn, tpr_knn, label='K-Nearest Neighbors (AUC = %0.2f)' % auc_knn)
plt.plot(fpr_svm, tpr_svm, label='Support Vector Machines (AUC = %0.2f)' % auc_svm)
plt.plot(fpr_dt, tpr_dt, label='Decision Tree (AUC = %0.2f)' % auc_dt)
plt.plot(fpr_gnb, tpr_gnb, label='Gaussian Naïve Bayes (AUC = %0.2f)' % auc_gnb)
plt.plot(fpr_gbc, tpr_gbc, label='Gradient Boosting Classifier (AUC = %0.2f)' % auc
plt.plot(fpr_rf, tpr_rf, label='Random Forest (AUC = %0.2f)' % auc_rf)

# Add legend, axis labels, and title
plt.legend(loc='lower right')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curves')

# Show the plot
plt.show()


(Asad)
```



Conclusion:

```
Since our target variable churn is highly imbalanced, so we will consider recall as the

We can see that Random Forest and Gaussian Naive Bayes give us the optimal levels of Re
```

Parkash Meghwar & Asad Sajid