DATA MINING TECHNIQUES

TELECOM USER CHURN PREDICTION

NAME: S PRAVEEN **REGISTER NO:19MIS1003**

COURSE CODE: SWE2009 **FACULTY: DR PATTABIRAMAN V**

ABSTRACT:

Telecom Companies are working hard to survive in this competitive market depending on multiple strategies. One among the strategies proposed by the company is to the decrease the potential of customers' churn known as "the customer movement from one provider to another".Any business wants to maximize the number of customers. To achieve this goal, it is important not only to try to attract new ones, but also to retain existing ones. Retaining a client will cost the company less than attracting a new one. In addition, a new client may be weakly interested in business services and it will be difficult to work with him, while old clients already have the necessary data on interaction with the service. Predicting customers' churn can be a very useful thing for the companies to boost their sales.

PROBLEM DEFINITION:

Churn prediction is one of the most popular Big Data use cases in business. It consists of detecting customers who are likely to cancel a subscription to a service. This can be telecom companies, SaaS companies, and any other company that sells a service for a monthly fee. Customer churn is a major problem and one of the most important concerns for large companies. Therefore, finding factors that increase customer churn is important to take necessary actions to reduce this churn. The main contribution of our work is to develop a churn prediction model which assists telecom operators to predict customers who are most likely subject to churn. The model developed in this work uses machine learning techniques on big data platform and builds a new way of features' engineering and selection.

DATASET:

I have used the Telecom Churn Dataset from data world website. This dataset tracks a fictional telecommunications company, Telco. It's customer churn data sourced by the IBM Developer Platform. It includes a target label indicating whether or not the customer left within the last month, and other dependent features that cover demographics, services that each customer has signed up for, and customer account information. It has data for 7043 clients, with 20 features.

```
dataoverview(data_df, 'Overview of the dataset')
```

```
Overview of the dataset:

Number of rows:  7043

Number of features: 21

Data Features:
['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetServic
e', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'Paperle
ssBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn']

Missing values: 0

Unique values:
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      1586
TotalCharges        6531
Churn                  2
dtype: int64
```

https://data.world/sumitrock/teclocomplete/workspace/file?filename=telco_churn.csv

METHODOLOGY:

Customer Churn dataset churn.csv from data.world.com is used for this project. Initially I have explored the dataset w.r.t to the target variable and then explored the numeric features. Then the dataset was preprocessed for converting the data into a data representation that is suitable for machine learning algorithms.

Once the major factors for the churn are realised then the model is built. I have performed five data mining algorithms namely Logistic Regression, SVC, RandomForrest, DecisionTree and GaussianNB to infer the accuracies for the datset. Then I have done hyperparameter tuning for the logistic regression model for improving the accuracy. Finally a WebApp whixh predicts the customer churn by user inputs has been made based on the model created.

ALGORITHMS USED:

Five data mining classification algorithms namely Logistic Regression, SVC, RandomForrest, DecisionTree and GaussianNB(Naïve Bayes) have been used for this project. These algorithms were imported into the code for building the model. Out of these five Logistic regression gave the best accuracy. So then I did hyperparameter tuning for further increasing the accuracy that we got by LogisticRegression.

CODE:

Model Building:

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


#Import libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
pd.set_option('display.max_columns', None)

import plotly.express as px #for visualization
import matplotlib.pyplot as plt #for visualization

#Read the dataset
data_df = pd.read_csv("../data/churn.csv")


# In[2]:


#Get overview of the data
def dataoveriew(df, message):
    print(f'{message}:\n')
    print('Number of rows: ', df.shape[0])
    print("\nNumber of features:", df.shape[1])
    print("\nData Features:")
    print(df.columns.tolist())
    print("\nMissing values:", df.isnull().sum().values.sum())
    print("\nUnique values:")
    print(df.nunique())

dataoveriew(data_df, 'Overview of the dataset')


# ### Explore Target variable
```

```
# In[3]:


target_instance = data_df["Churn"].value_counts().to_frame()
target_instance = target_instance.reset_index()
target_instance = target_instance.rename(columns={'index': 'Category'})
fig = px.pie(target_instance, values='Churn', names='Category',
color_discrete_sequence=["green", "red"],
             title='Distribution of Churn')
fig.show()


# ### Exploratory Data Analysis

# In[4]:


#Defining bar chart function
def bar(feature, df=data_df ):
    #Groupby the categorical feature
    temp_df = df.groupby([feature, 'Churn']).size().reset_index()
    temp_df = temp_df.rename(columns={0:'Count'})
    #Calculate the value counts of each distribution and it's corresponding
Percentages
    value_counts_df = df[feature].value_counts().to_frame().reset_index()
    categories = [cat[1][0] for cat in value_counts_df.iterrows()]
    #Calculate the value counts of each distribution and it's corresponding
Percentages
    num_list = [num[1][1] for num in value_counts_df.iterrows()]
    div_list = [element / sum(num_list) for element in num_list]
    percentage = [round(element * 100,1) for element in div_list]
    #Defining string formatting for graph annotation
    #Numeric section
    def num_format(list_instance):
        formatted_str = ''
        for index,num in enumerate(list_instance):
            if index < len(list_instance)-2:
                formatted_str=formatted_str+f'{num}%, ' #append to empty
string(formatted_str)
            elif index == len(list_instance)-2:
                formatted_str=formatted_str+f'{num}% & '
            else:
                formatted_str=formatted_str+f'{num}%'
        return formatted_str
    #Categorical section
    def str_format(list_instance):
        formatted_str = ''
        for index, cat in enumerate(list_instance):
            if index < len(list_instance)-2:
                formatted_str=formatted_str+f'{cat}, '
            elif index == len(list_instance)-2:
                formatted_str=formatted_str+f'{cat} & '
            else:
                formatted_str=formatted_str+f'{cat}'
        return formatted_str


    #Running the formatting functions
    num_str = num_format(percentage)
    cat_str = str_format(categories)
```

```
    #Setting graph framework
    fig = px.bar(temp_df, x=feature, y='Count', color='Churn',
title=f'Churn rate by {feature}', barmode="group",
color_discrete_sequence=["green", "red"])
    fig.add_annotation(
                text=f'Value count of distribution of {cat_str}
are<br>{num_str} percentage respectively.',
                align='left',
                showarrow=False,
                xref='paper',
                yref='paper',
                x=1.4,
                y=1.3,
                bordercolor='black',
                borderwidth=1)
    fig.update_layout(
        # margin space for the annotations on the right
        margin=dict(r=400),
    )

    return fig.show()


# In[5]:


#Gender feature plot
bar('gender')
#SeniorCitizen feature plot
data_df.loc[data_df.SeniorCitizen==0,'SeniorCitizen'] = "No"   #convert 0
to No in all data instances
data_df.loc[data_df.SeniorCitizen==1,'SeniorCitizen'] = "Yes"  #convert 1
to Yes in all data instances
bar('SeniorCitizen')
#Partner feature plot
bar('Partner')
#Dependents feature plot
bar('Dependents')


# In[6]:


bar('PhoneService')
bar('MultipleLines')
bar('InternetService')
bar('OnlineSecurity')
bar('OnlineBackup')
bar('DeviceProtection')
bar('TechSupport')
bar('StreamingTV')
bar('StreamingMovies')


# In[7]:


bar('Contract')
bar('PaperlessBilling')
bar('PaymentMethod')
```

```
# ### Explore Numeric features

# In[8]:


data_df.dtypes


# In[9]:


try:
    data_df['TotalCharges'] = data_df['TotalCharges'].astype(float)
except ValueError as ve:
    print (ve)


# In[10]:


data_df['TotalCharges'] =
pd.to_numeric(data_df['TotalCharges'],errors='coerce')
#Fill the missing values with with the median value
data_df['TotalCharges'] =
data_df['TotalCharges'].fillna(data_df['TotalCharges'].median())


# In[11]:


# Defining the histogram plotting function
def hist(feature):
    group_df = data_df.groupby([feature, 'Churn']).size().reset_index()
    group_df = group_df.rename(columns={0: 'Count'})
    fig = px.histogram(group_df, x=feature, y='Count', color='Churn',
marginal='box', title=f'Churn rate frequency to {feature} distribution',
color_discrete_sequence=["green", "red"])
    fig.show()


# In[12]:


hist('tenure')
hist('MonthlyCharges')
hist('TotalCharges')


# ***
# **Customer account information**: The tenure histogram is rightly skewed
and shows that majority of customers has been with the telecom company for
just the first few months (0-9 months) and the highest rate of churn is
also in that first few months (0-9months). 75% of customers who end up
leaving Telcom company  do so within their first 30 months. The monthly
charge histogram shows that clients with higher monthly charges have a
higher churn rate (This suggests that discounts and promotions can be an
enticing reason for customers to stay). The total charge trend is quite
depict due to variation in frequency.
# Lets bin the numeric features into 3 sections based on quantiles (low,
```

```
medium and high to get more information from it).
# ***

# In[13]:


#Create an empty dataframe
bin_df = pd.DataFrame()

#Update the binning dataframe
bin_df['tenure_bins'] =  pd.qcut(data_df['tenure'], q=3, labels= ['low',
'medium', 'high'])
bin_df['MonthlyCharges_bins'] =  pd.qcut(data_df['MonthlyCharges'], q=3,
labels= ['low', 'medium', 'high'])
bin_df['TotalCharges_bins'] =  pd.qcut(data_df['TotalCharges'], q=3,
labels= ['low', 'medium', 'high'])
bin_df['Churn'] = data_df['Churn']

#Plot the bar chart of the binned variables
bar('tenure_bins', bin_df)
bar('MonthlyCharges_bins', bin_df)
bar('TotalCharges_bins', bin_df)


# ### Data preprocessing

# In[14]:


# The customerID column isnt useful as the feature us used for
identification of customers.
data_df.drop(["customerID"],axis=1,inplace = True)

# Encode categorical features

#Defining the map function
def binary_map(feature):
    return feature.map({'Yes':1, 'No':0})

## Encoding target feature
data_df['Churn'] = data_df[['Churn']].apply(binary_map)

# Encoding gender category
data_df['gender'] = data_df['gender'].map({'Male':1, 'Female':0})

#Encoding other binary category
binary_list = ['SeniorCitizen', 'Partner', 'Dependents', 'PhoneService',
'PaperlessBilling']
data_df[binary_list] = data_df[binary_list].apply(binary_map)

#Encoding the other categoric features with more than two categories
data_df = pd.get_dummies(data_df, drop_first=True)


# In[15]:


# Checking the correlation between features
corr = data_df.corr()

fig = px.imshow(corr,width=1000, height=1000)
```

```
fig.show()


# In[16]:


import statsmodels.api as sm
import statsmodels.formula.api as smf

#Change variable name seperators to '_'
all_columns = [column.replace(" ", "_").replace("(", "_").replace(")",
"_").replace("-", "_") for column in data_df.columns]

#Effect the change to the dataframe column names
data_df.columns = all_columns

#Prepare it for the GLM formula
glm_columns = [e for e in all_columns if e not in ['customerID', 'Churn']]
glm_columns = ' + '.join(map(str, glm_columns))

#Fiting it to the Generalized Linear Model
glm_model = smf.glm(formula=f'Churn ~ {glm_columns}', data=data_df,
family=sm.families.Binomial())
res = glm_model.fit()
print(res.summary())


# In[17]:


np.exp(res.params)


# In[18]:


#feature scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
data_df['tenure'] = sc.fit_transform(data_df[['tenure']])
data_df['MonthlyCharges'] = sc.fit_transform(data_df[['MonthlyCharges']])
data_df['TotalCharges'] = sc.fit_transform(data_df[['TotalCharges']])


# #### Creating a baseline model

# In[19]:


# Import Machine learning algorithms
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB

#Import metric for performance evaluation
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

#Split data into train and test sets
```

```python
from sklearn.model_selection import train_test_split
X = data_df.drop('Churn', axis=1)
y = data_df['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=50)


# In[20]:


def modeling(alg, alg_name, params={}):
    model = alg(**params) #Instantiating the algorithm class and unpacking
parameters if any
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    #Performance evaluation
    def print_scores(alg, y_true, y_pred):
        print(alg_name)
        acc_score = accuracy_score(y_true, y_pred)
        print("accuracy: ",acc_score)
        pre_score = precision_score(y_true, y_pred)
        print("precision: ",pre_score)
        rec_score = recall_score(y_true, y_pred)
        print("recall: ",rec_score)
        f_score = f1_score(y_true, y_pred, average='weighted')
        print("f1_score: ",f_score)

    print_scores(alg, y_test, y_pred)
    return model


# In[21]:


# Running logistic regression model
log_model = modeling(LogisticRegression, 'Logistic Regression')


# In[22]:


# Feature selection to improve model building
from sklearn.feature_selection import RFECV
from sklearn.model_selection import StratifiedKFold
log = LogisticRegression()
rfecv = RFECV(estimator=log, cv=StratifiedKFold(10, random_state=50,
shuffle=True), scoring="accuracy")
rfecv.fit(X, y)


# In[23]:


plt.figure(figsize=(8, 6))
plt.plot(range(1, len(rfecv.grid_scores_)+1), rfecv.grid_scores_)
plt.grid()
plt.xticks(range(1, X.shape[1]+1))
plt.xlabel("Number of Selected Features")
plt.ylabel("CV Score")
plt.title("Recursive Feature Elimination (RFE)")
```

```python
plt.show()

print("The optimal number of features: {}".format(rfecv.n_features_))


# In[24]:


#Saving dataframe with optimal features
X_rfe = X.iloc[:, rfecv.support_]

#Overview of the optimal features in comparison with the intial dataframe
print("\"X\" dimension: {}".format(X.shape))
print("\"X\" column list:", X.columns.tolist())
print("\"X_rfe\" dimension: {}".format(X_rfe.shape))
print("\"X_rfe\" column list:", X_rfe.columns.tolist())


# In[25]:


# Splitting data with optimal features
X_train, X_test, y_train, y_test = train_test_split(X_rfe, y,
test_size=0.3, random_state=50)


# In[26]:


# Running logistic regression model
log_model = modeling(LogisticRegression, 'Logistic Regression
Classification')


# In[27]:


### Trying other machine learning algorithms: SVC
svc_model = modeling(SVC, 'SVC Classification')


# In[28]:


#Random forest
rf_model = modeling(RandomForestClassifier, "Random Forest Classification")


# In[29]:


#Decision tree
dt_model = modeling(DecisionTreeClassifier, "Decision Tree Classification")


# In[30]:


#Naive bayes
nb_model = modeling(GaussianNB, "Naive Bayes Classification")
```

```
# In[31]:


## Improve best model by hyperparameter tuning
# define model
model = LogisticRegression()

# define evaluation
from sklearn.model_selection import RepeatedStratifiedKFold
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

# define search space
from scipy.stats import loguniform
space = dict()
space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']
space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
space['C'] = loguniform(1e-5, 1000)

# define search
from sklearn.model_selection import RandomizedSearchCV
search = RandomizedSearchCV(model, space, n_iter=500, scoring='accuracy',
n_jobs=-1, cv=cv, random_state=1)

# execute search
result = search.fit(X_rfe, y)
#summarize result
#print('Best Score: %s' % result.best_score_)
#print('Best Hyperparameters: %s' % result.best_params_)


# In[32]:


params = result.best_params_
params


# In[33]:


#Improving the Logistic Regression model
log_model = modeling(LogisticRegression, 'Logistic Regression
Classification', params=params)


# In[34]:


#Saving best model
import joblib
#Sava the model to disk
filename = 'model.sav'
joblib.dump(log_model, filename)


# In[ ]:
```

```
# In[ ]:
```

App.py:

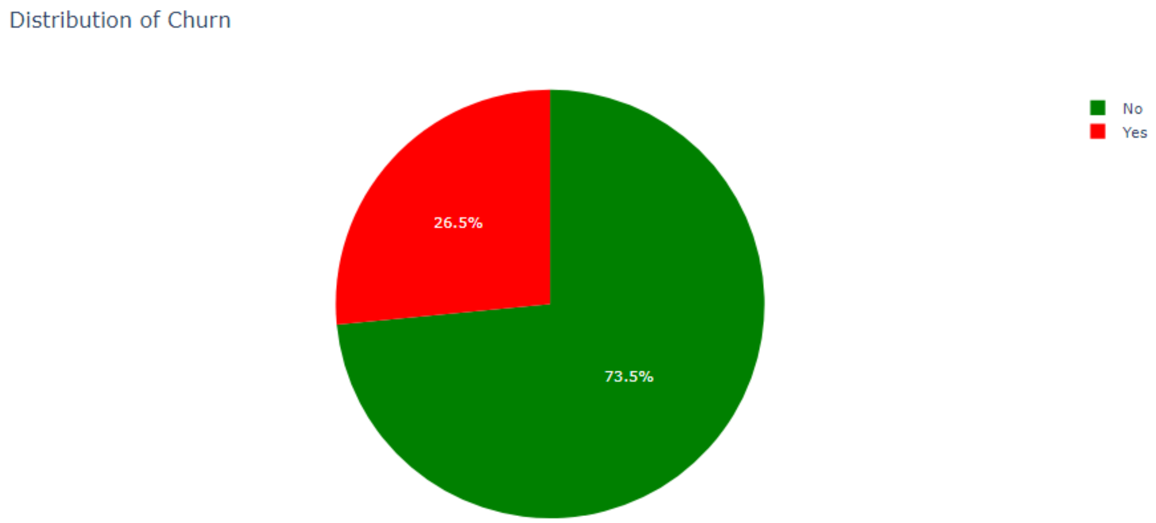
```
#Import libraries
import streamlit as st
import pandas as pd
import numpy as np
from PIL import Image

#load the model from disk
import joblib
model = joblib.load(r"./notebook/model.sav")

#Import python scripts
from preprocessing import preprocess

def main():
    #Setting Application title
    st.title('Telecom User Churn Prediction App(19MIS1003)')

      #Setting Application description
    st.markdown("""
     :dart:  This app is made using Streamlit(Open-source python
framework). This app is made to predict customer churn for telecom
companys' use case. This app is functional for online and batch of data. \n
    """)
    st.markdown("<h3></h3>", unsafe_allow_html=True)

    #Setting Application sidebar default
    image = Image.open('App.jpg')
    add_selectbox = st.sidebar.selectbox(
   "How would you like to predict?", ("Online", "Batch"))
    st.sidebar.info('This app is created to predict Customer Churn')
    st.sidebar.image(image)

    if add_selectbox == "Online":
        st.info("Input data below")
        #Based on our optimal features selection
        st.subheader("Customer Data")
        seniorcitizen = st.selectbox('Senior Citizen:', ('Yes', 'No'))
        dependents = st.selectbox('Dependent:', ('Yes', 'No'))
        gender = st.selectbox('Gender:', ('Male', 'Female'))
        partner = st.selectbox('Partner:', ('Yes', 'No'))


        st.subheader("Payment data")
        tenure = st.slider('Number of months the customer has stayed with
the company', min_value=0, max_value=72, value=0)
        contract = st.selectbox('Contract', ('Month-to-month', 'One year',
'Two year'))
        paperlessbilling = st.selectbox('Paperless Billing', ('Yes', 'No'))
        PaymentMethod = st.selectbox('PaymentMethod',('Electronic check',
'Mailed check', 'Bank transfer (automatic)','Credit card (automatic)'))
        monthlycharges = st.number_input('The amount charged to the
```

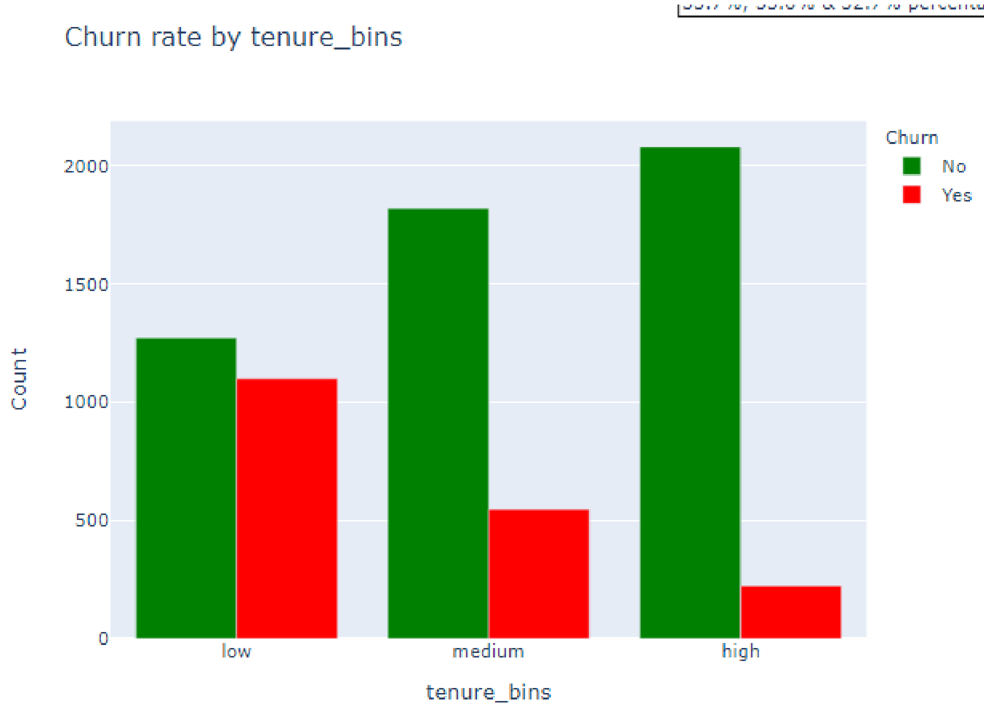
```
customer monthly', min_value=0, max_value=500, value=0)
        totalcharges = st.number_input('The total amount charged to the
customer',min_value=0, max_value=10000, value=0)

        st.subheader("Services signed up for")
        mutliplelines = st.selectbox("Does the customer have multiple
lines",('Yes','No','No phone service'))
        phoneservice = st.selectbox('Phone Service:', ('Yes', 'No'))
        internetservice = st.selectbox("Does the customer have internet
service", ('DSL', 'Fiber optic', 'No'))
        onlinesecurity = st.selectbox("Does the customer have online
security",('Yes','No','No internet service'))
        onlinebackup = st.selectbox("Does the customer have online
backup",('Yes','No','No internet service'))
        techsupport = st.selectbox("Does the customer have technology
support", ('Yes','No','No internet service'))
        streamingtv = st.selectbox("Does the customer stream TV",
('Yes','No','No internet service'))
        streamingmovies = st.selectbox("Does the customer stream movies",
('Yes','No','No internet service'))
        industry_avg=210
        data = {
                'SeniorCitizen': seniorcitizen,
                'Dependents': dependents,
                'Gender': gender,
                'Partner': partner,
                'tenure':tenure,
                'PhoneService': phoneservice,
                'MultipleLines': mutliplelines,
                'InternetService': internetservice,
                'OnlineSecurity': onlinesecurity,
                'OnlineBackup': onlinebackup,
                'TechSupport': techsupport,
                'StreamingTV': streamingtv,
                'StreamingMovies': streamingmovies,
                'Contract': contract,
                'PaperlessBilling': paperlessbilling,
                'PaymentMethod':PaymentMethod,
                'MonthlyCharges': monthlycharges,
                'TotalCharges': totalcharges
                }
        features_df = pd.DataFrame.from_dict([data])
        st.markdown("<h3></h3>", unsafe_allow_html=True)
        st.write('Overview of input is shown below')
        st.markdown("<h3></h3>", unsafe_allow_html=True)
        st.dataframe(features_df)


        #Preprocess inputs
        preprocess_df = preprocess(features_df, 'Online')

        prediction = model.predict(preprocess_df)

        if st.button('Predict'):
            if prediction == 1:
                st.warning('Yes, the customer will terminate the service.')
            else:
                st.success('No, the customer is happy with Telco
Services.')
        if monthlycharges > industry_avg:
            st.warning('The tariff plan is higher compared to other telecom
```

```
providers. Customers may churn if no measures are taken.')
        elif monthlycharges < industry_avg:
            st.success('The tariff plan is fairly less compared to other
telecom providers. Customers may stay longer.')

    else:
        st.subheader("Dataset upload")
        uploaded_file = st.file_uploader("Choose a file")
        if uploaded_file is not None:
            data = pd.read_csv(uploaded_file)
            #Get overview of data
            st.write(data.head())
            st.markdown("<h3></h3>", unsafe_allow_html=True)
            #Preprocess inputs
            preprocess_df = preprocess(data, "Batch")
            if st.button('Predict'):
                #Get batch prediction
                prediction = model.predict(preprocess_df)
                prediction_df = pd.DataFrame(prediction,
columns=["Predictions"])
                prediction_df = prediction_df.replace({1:'Yes, the customer
will terminate the service.',
                                                    0:'No, the customer is
happy with Telco Services.'})
            if monthlycharges > industry_avg:
                st.warning('The tariff plan is higher compared to other
telecom providers. Customers may churn if no measures are taken.')
            elif monthlycharges < industry_avg:
                st.success('The tariff plan is fairly less compared to
other telecom providers. Customers may stay longer.')
                st.markdown("<h3></h3>", unsafe_allow_html=True)
                st.subheader('Prediction')
                st.write(prediction_df)

if __name__ == '__main__':
        main()
```

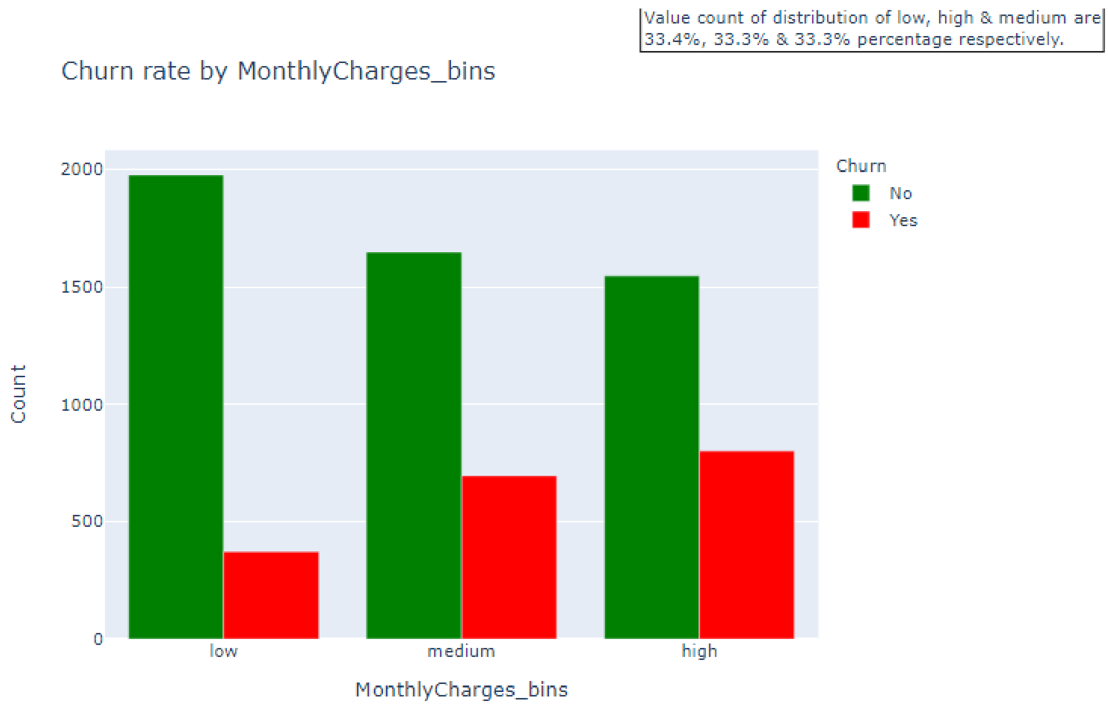
RESULTS:

This is the pie representation of the people/customers who have churned or moved out of the company in the last month according to the dataset that we have used.

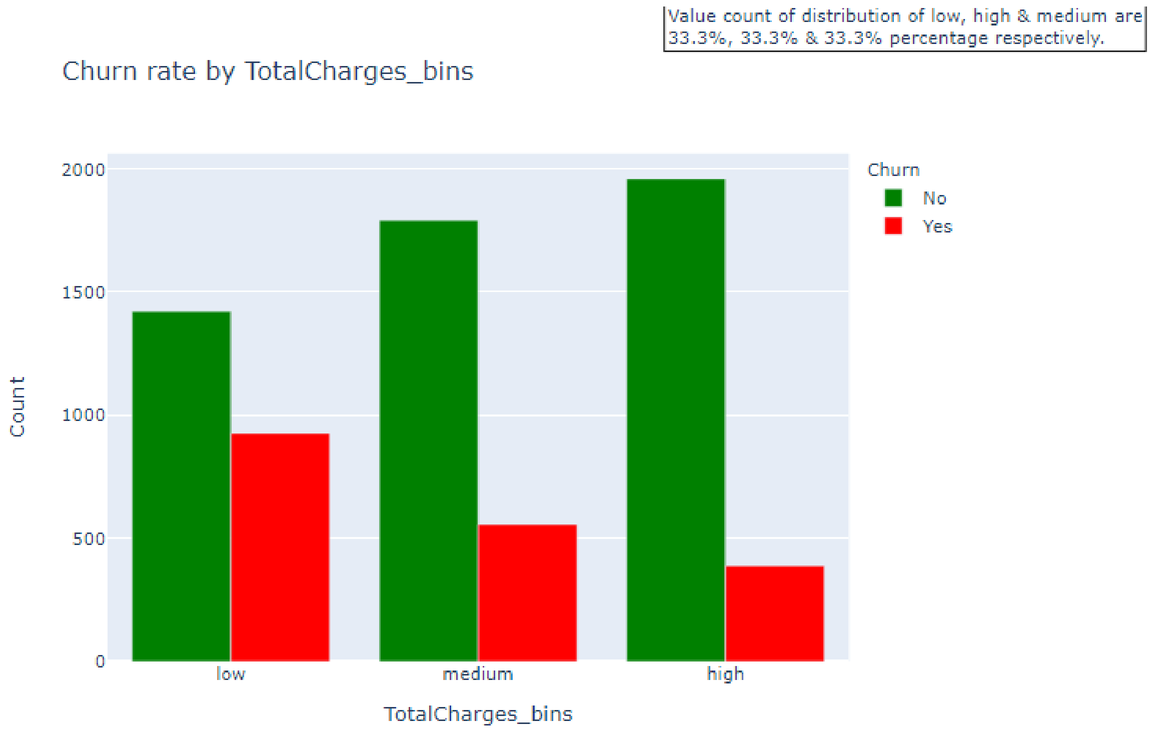




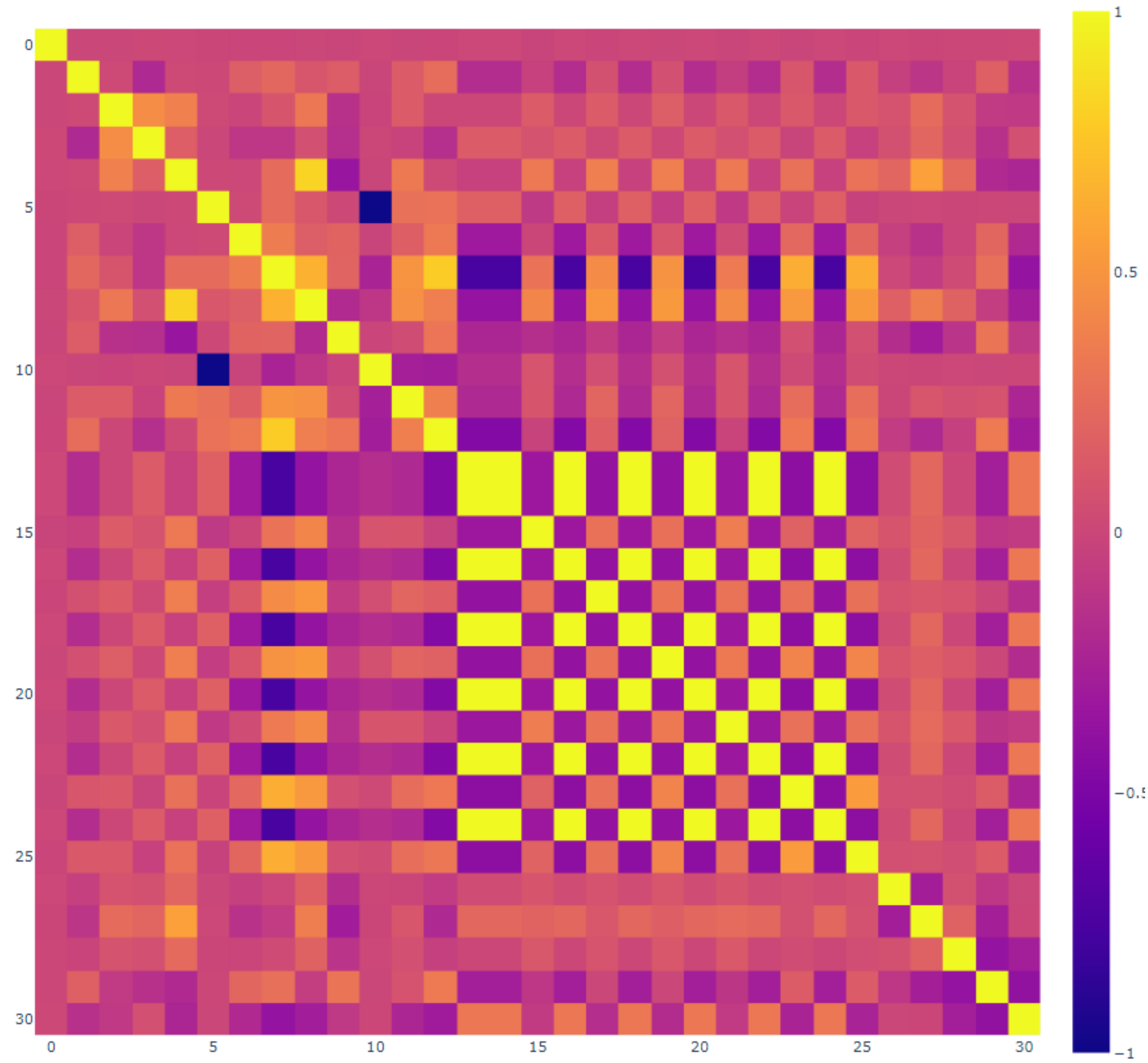
The higher the tenure period for which the customer is using the telecom service the lesser the churn. As tenure in high category has drastical difference compared to the tenure in low category as it is almost identical



The customer churn is comparatively higher when the monthly charges that they are paying is high.



Another interesting conclusion where the customer churn is comparitevely less when the custoemrs are paying high toal charges per month . This indicates that customers might be interested in high tariff plans or customers are not so considering tariff price they are paying.



This graph is the correlation of features ranging from 1 to -1.

```python
# Running logistic regression model
log_model = modeling(LogisticRegression, 'Logistic Regression Classification')
```

```
Logistic Regression Classification
accuracy:  0.8017037387600567
precision:  0.6374501992031872
recall:  0.5745062836624776
f1_score:  0.7982762676502377
```

```python
### Trying other machine learning algorithms: SVC
svc_model = modeling(SVC, 'SVC Classification')
```

```
SVC Classification
accuracy:  0.7993374349266446
precision:  0.6494382022471911
recall:  0.518850987432675
f1_score:  0.7916082146150322
```

```python
#Random forest
rf_model = modeling(RandomForestClassifier, "Random Forest Classification")
```

```
Random Forest Classification
accuracy:  0.7860861334595362
precision:  0.6082474226804123
recall:  0.5296229802513465
f1_score:  0.7811142593105196
```

```python
#Decision tree
dt_model = modeling(DecisionTreeClassifier, "Decision Tree Classification")
```

```
Decision Tree Classification
accuracy:  0.7302413629910081
precision:  0.4889267461669506
recall:  0.5152603231597845
f1_score:  0.7324655009979134
```

```python
#Naive bayes
nb_model = modeling(GaussianNB, "Naive Bayes Classification")
```

```
Naive Bayes Classification
accuracy:  0.6540463795551349
precision:  0.4257679180887372
recall:  0.895870736086176
f1_score:  0.6729702812977834
```

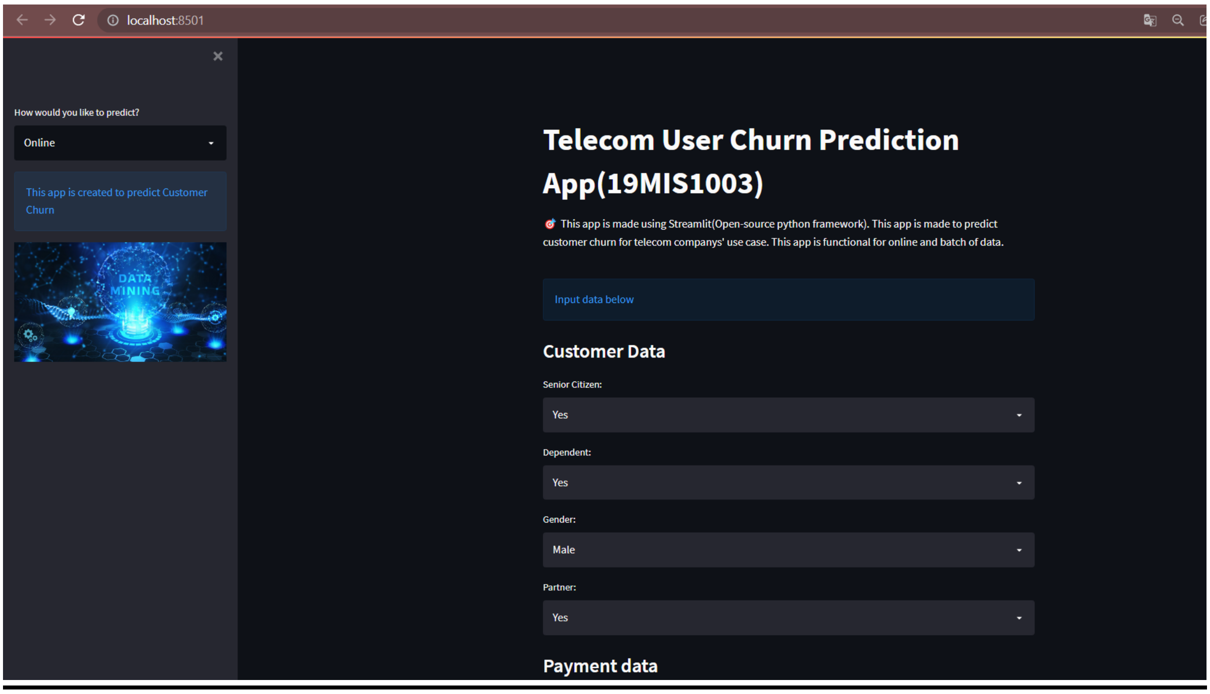
The accuracy, precision, recall and f1 score of the five data mining algorithms have been given. Here we can see that LogisticRegression has given the highest accuracy and NaiveBayes model has performed with the least accuracy.

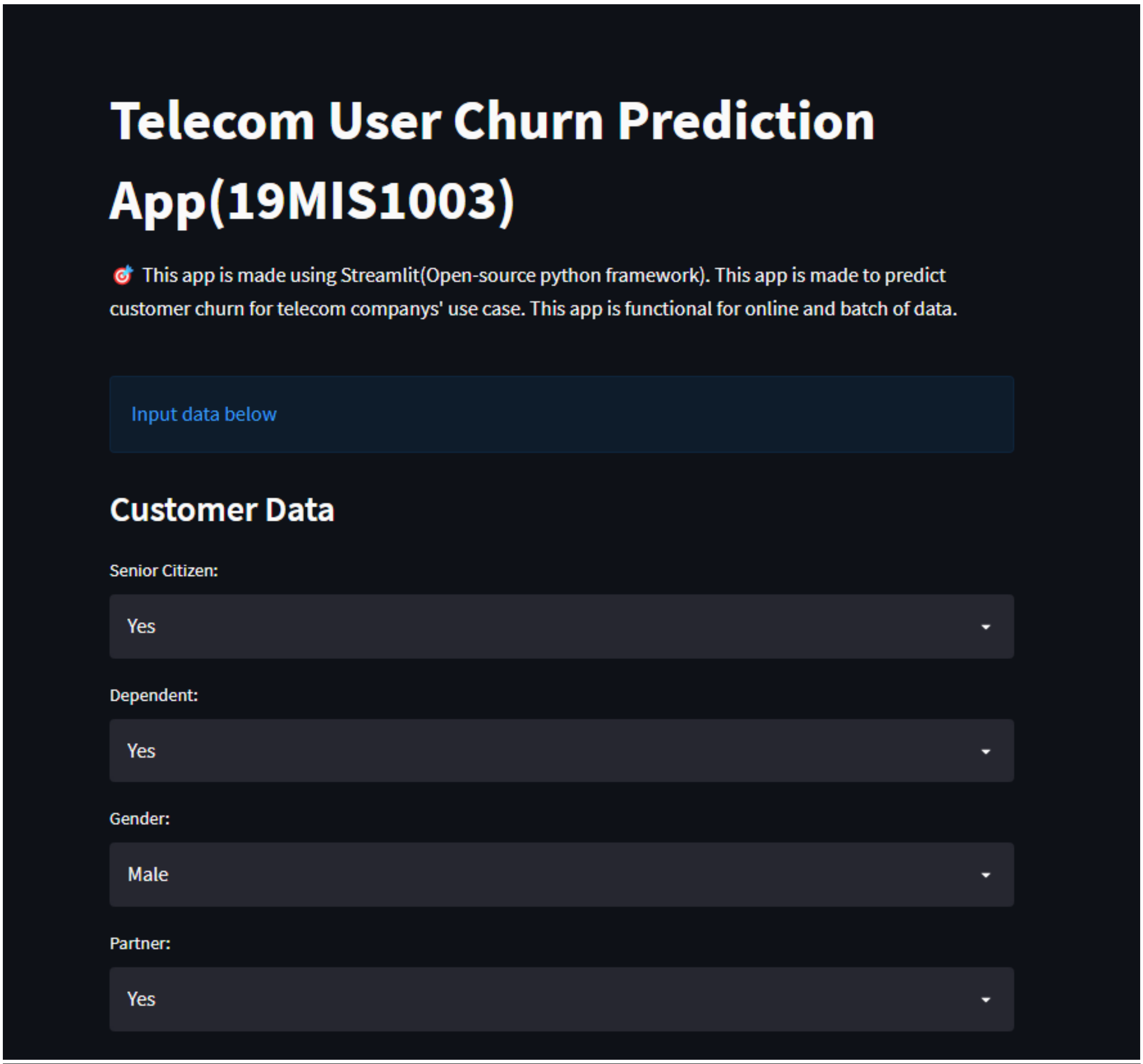
```python
#Improving the Logistic Regression model
log_model = modeling(LogisticRegression, 'Logistic Regression Classification', params=params)
```

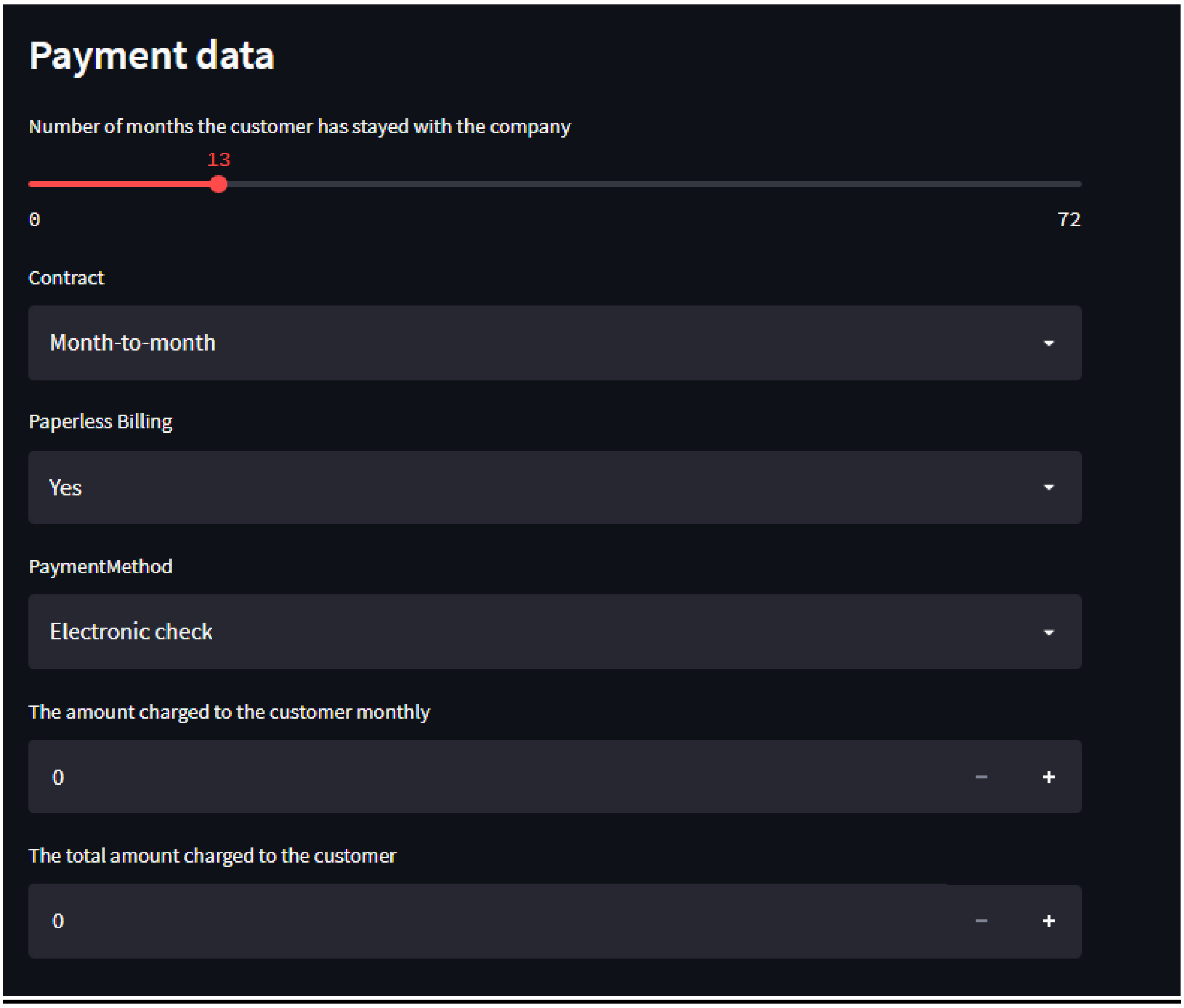
```
Logistic Regression Classification
accuracy:  0.8031235210601041
precision:  0.6407185628742516
recall:  0.5763016157989228
f1_score:  0.7996532493520713
```

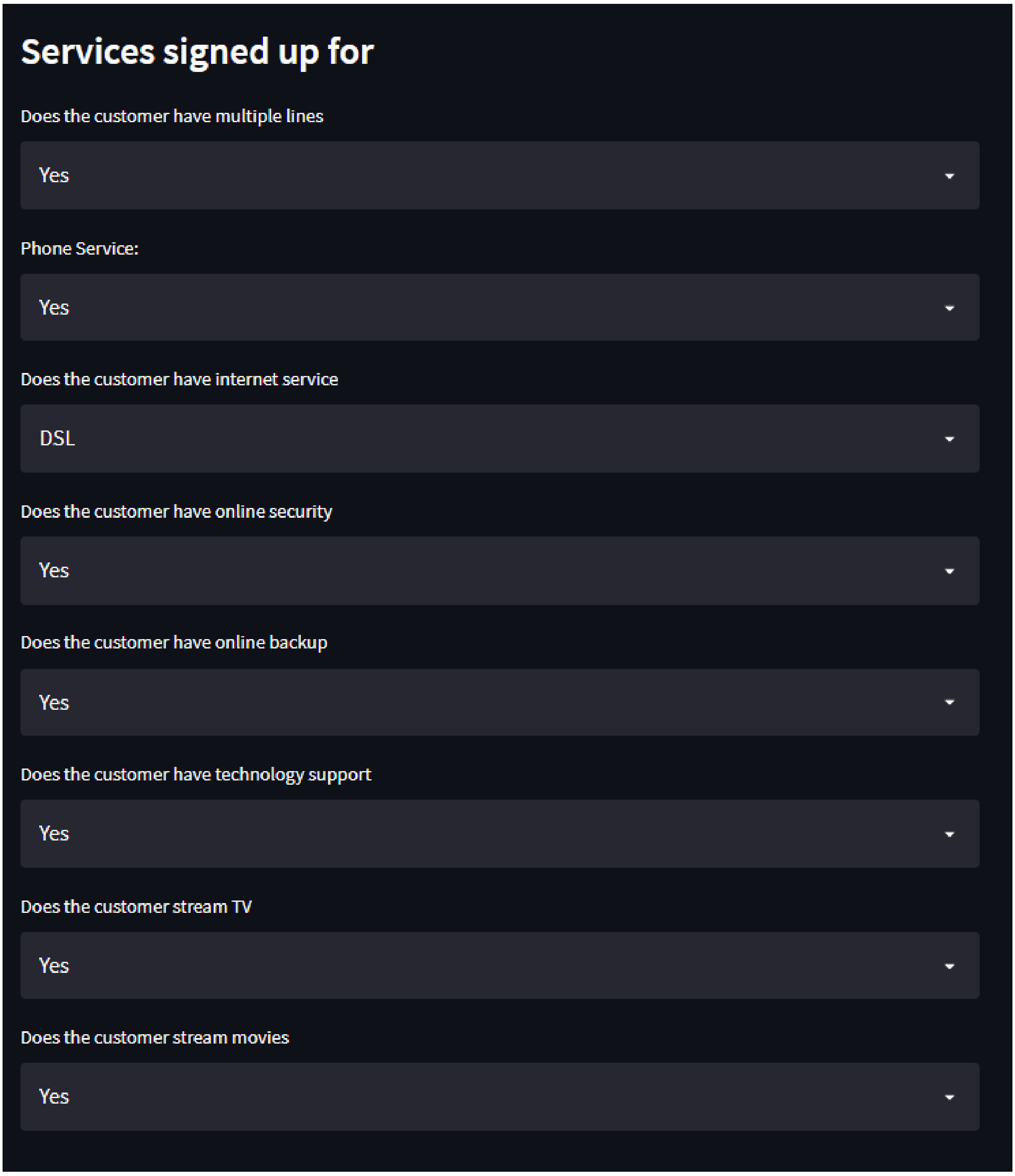
As logisticRegression gave the highest accuracy, I did hyperparameter tuning to check if there is further improvement in the accuracy, and the accuracy improved from 0.801 to 0.803.

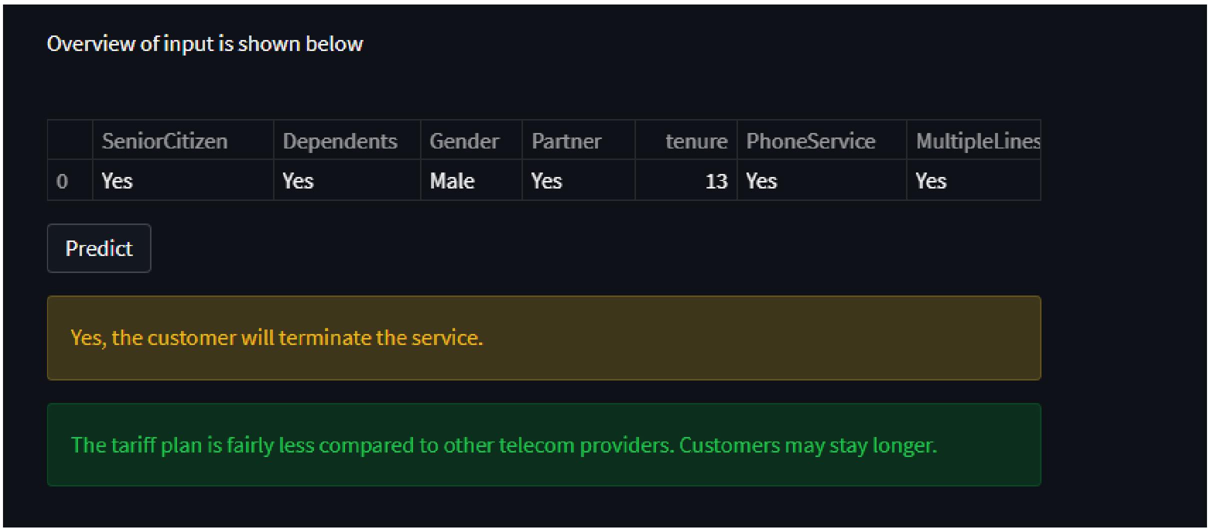
SCREENSHOTS OF THE LOCAL APP




Telecom User Churn Prediction App(19MIS1003)
🎯 This app is made using Streamlit(Open-source python framework). This app is made to predict customer churn for telecom companys' use case. This app is functional for online and batch of data.
Input data below
Customer Data
Senior Citizen:
Yes
Dependent:
Yes
Gender:
Male
Partner:
Yes


Payment data
Number of months the customer has stayed with the company
13
0
72
Contract
Month-to-month
Paperless Billing
Yes
PaymentMethod
Electronic check
The amount charged to the customer monthly
0
The total amount charged to the customer
0


Services signed up for
Does the customer have multiple lines
Yes
Phone Service:
Yes
Does the customer have internet service
DSL
Does the customer have online security
Yes
Does the customer have online backup
Yes
Does the customer have technology support
Yes
Does the customer stream TV
Yes
Does the customer stream movies
Yes



CONCLUSION:

Having researched about the telecom customer churn problem, the factors which mostly influence the customer to churn have been identified with python. Every factor has been depicted with its influence on the churn with a graph. More crucial data like comparison of average fee per month for the telecom sector vs the fee for XYZ company turned out to be beneficial. I would like to do further research on this data and would build a improved model if possible. The streamlit based webapp has been trained with the same model and can be used to predict the customer churn with a set of inputs.

REFERENCES:

1. P. K. Dalvi, S. K. Khandge, A. Deomore, A. Bankar and V. A. Kanade, "Analysis of customer churn prediction in telecom industry using decision trees and logistic regression," 2016 Symposium on Colossal Data Analysis and Networking (CDAN), 2016, pp. 1-4, doi: 10.1109/CDAN.2016.7570883.
2. https://learnpython.com/blog/
3. https://neptune.ai/blog/streamlit-guide-machine-learning