Churn_Analysis_

November 12, 2022

1 Churn Customers

1.0.1 Introduction

This IBM Sample Dataset has information about Telco customers and if they left the company within the last month (churn).

Basic information: Customers who left within the last month – the column is called Churn. Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies. Customer account information – how long they've been a customer, contract, payment method, paperless billing, monthly charges, and total charges. Demographic info about customers – gender, age range, and if they have partners and dependents. There are 21 columns with 19 features.

Objective I will explore the data and try to answer some questions like:

Customer churn measures how and why are customers leaving the business

1.0.2 Importing Libraries

[ ]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
```

1.0.3 Loading data

[ ]:
```
df = pd.read_csv("Churndata.csv")
df.sample(5)
```

[ ]:
```
      customerID  gender  SeniorCitizen  … MonthlyCharges TotalCharges  Churn
3821  1833-VGRUM  Female              1  …          74.20         74.2    Yes
4831  4654-GGUII  Female              0  …          40.20       711.95     No
6528  4957-SREEC    Male              0  …          20.35       1458.1     No
6126  9190-MFJLN    Male              1  …          95.90       1777.9    Yes
4267  1227-UDMZR  Female              0  …          91.15       6637.9     No

[5 rows x 21 columns]
```

1.0.4 Pre-processing

[ ]:
```
#Checking for datatypes
#Wesee that Total charges is an object. We need to change that to float
df.dtypes
```

[ ]:
```
customerID          object
gender              object
SeniorCitizen        int64
Partner             object
Dependents          object
tenure               int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
DeviceProtection    object
TechSupport         object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling    object
PaymentMethod       object
MonthlyCharges     float64
TotalCharges        object
Churn               object
dtype: object
```

[ ]:
```
df.TotalCharges.values
```

[ ]:
```
array(['29.85', '1889.5', '108.15', …, '346.45', '306.6', '6844.5'],
      dtype=object)
```

[ ]:
```
df['TotalCharges'] = df['TotalCharges'].replace(" ", 0).astype('float64')
```

[ ]:
```
df.dtypes
```

[ ]:
```
customerID          object
gender              object
SeniorCitizen        int64
Partner             object
Dependents          object
tenure               int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
```

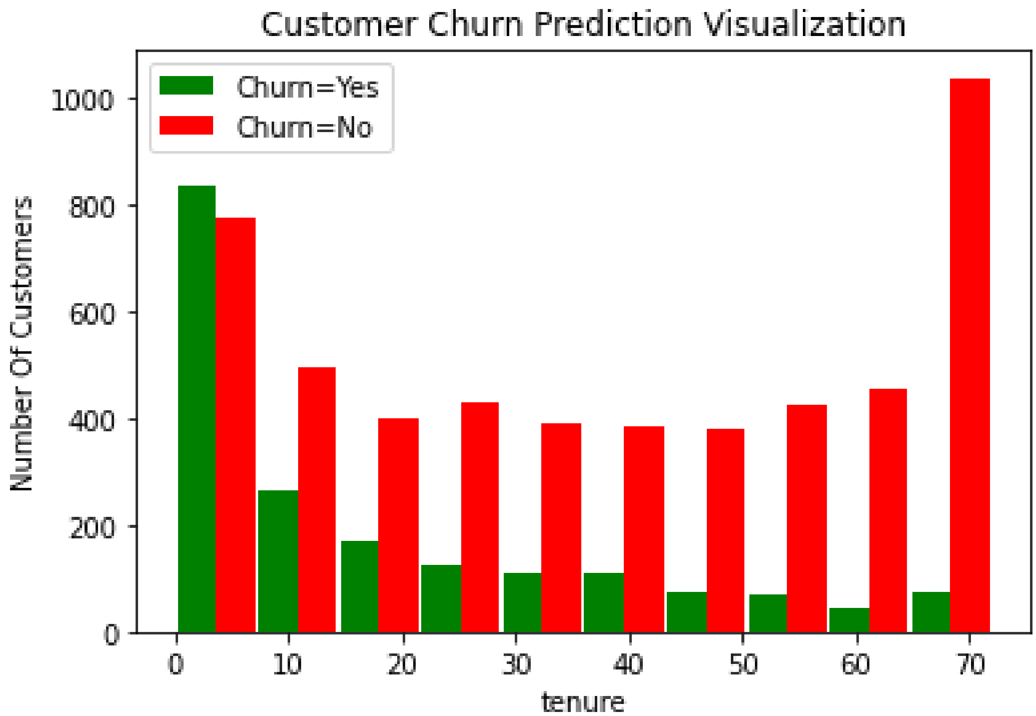
```
OnlineBackup          object
DeviceProtection      object
TechSupport           object
StreamingTV           object
StreamingMovies       object
Contract              object
PaperlessBilling      object
PaymentMethod         object
MonthlyCharges       float64
TotalCharges         float64
Churn                 object
dtype: object
```

1.0.5 Data Visualization

Viz1 : No. of customers Vs. Tenure

[ ]:
```
tenure_churn_no = df[df.Churn=='No'].tenure
tenure_churn_yes = df[df.Churn=='Yes'].tenure

plt.xlabel("tenure")
plt.ylabel("Number Of Customers")
plt.title("Customer Churn Prediction Visualization")

blood_sugar_men = [113, 85, 90, 150, 149, 88, 93, 115, 135, 80, 77, 82, 129]
blood_sugar_women = [67, 98, 89, 120, 133, 150, 84, 69, 89, 79, 120, 112, 100]

plt.hist([tenure_churn_yes, tenure_churn_no], rwidth=0.95,␣
 ↪color=['green','red'],label=['Churn=Yes','Churn=No'])
plt.legend()

#In this graph, we have the number of customers who will be leaving and not␣
 ↪leaving the company, and we have the tenure of months that they have been␣
 ↪apart of the company till now.
```

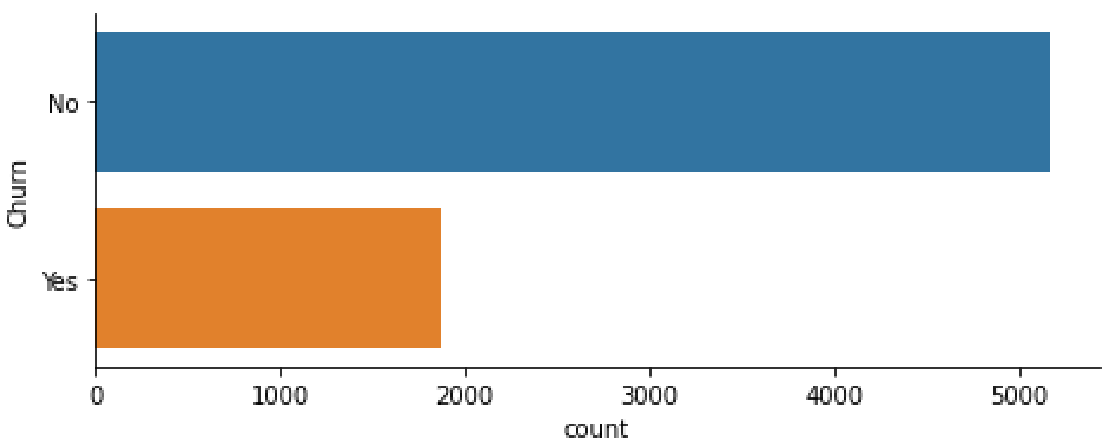
[ ]:
```
<matplotlib.legend.Legend at 0x7f877bf79cf8>
```



Churn Vs. Count Churn: No - 72.4%

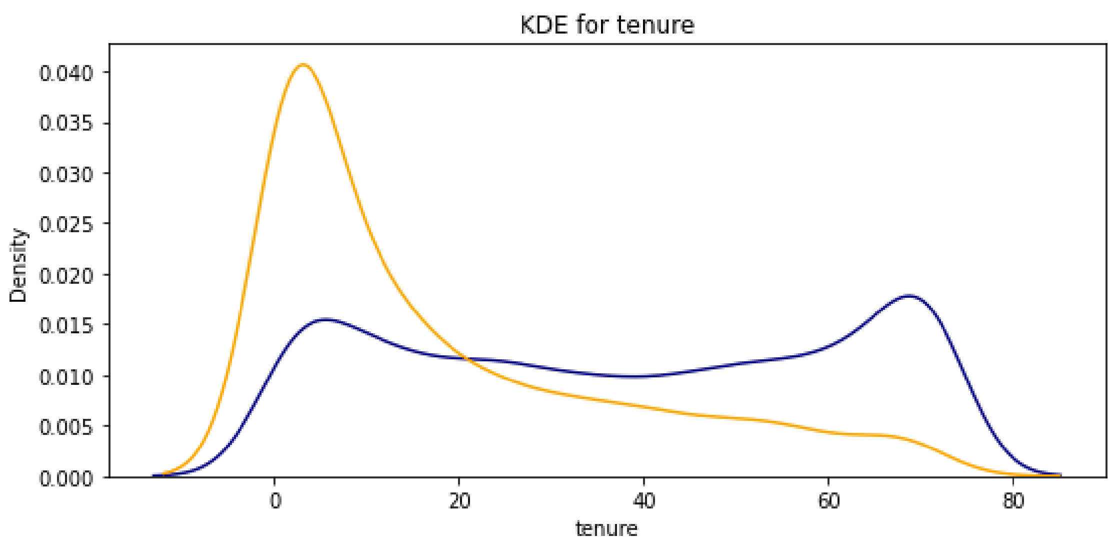
Churn: Yes - 27.6%

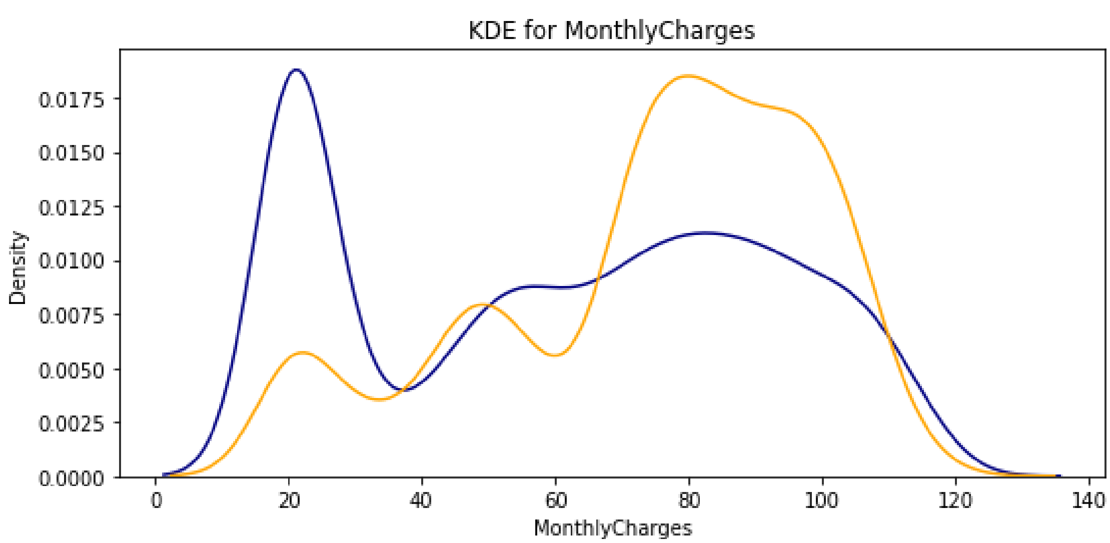
```
[ ]: ax = sns.catplot(y="Churn", kind="count", data=df, height=2.6, aspect=2.5,␣
     ↪orient='h')
```

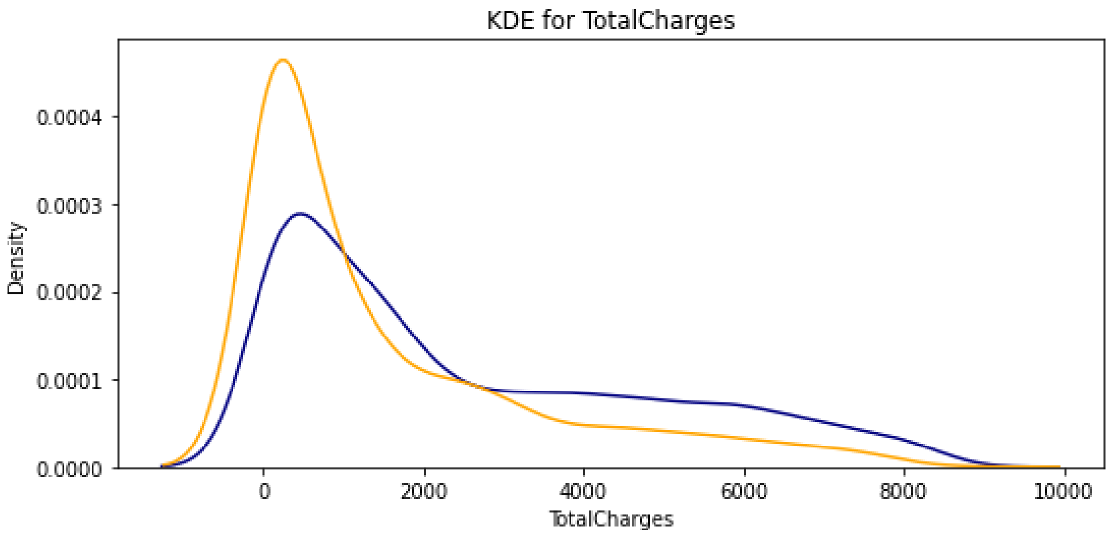


Numeric Features There are only three numerical columns: tenure, monthly charges and total charges. The probability density distribution can be estimate using the seaborn kdeplot function.

```
[ ]: def kdeplot(feature):
         plt.figure(figsize=(9, 4))
         plt.title("KDE for {}".format(feature))
         ax0 = sns.kdeplot(df[df['Churn'] == 'No'][feature].dropna(), color= 'navy',␣
      ↪label= 'Churn: No')
         ax1 = sns.kdeplot(df[df['Churn'] == 'Yes'][feature].dropna(), color=␣
      ↪'orange', label= 'Churn: Yes')
     kdeplot('tenure')
     kdeplot('MonthlyCharges')
     kdeplot('TotalCharges')
```







From the plots above we can conclude that:

Recent clients are more likely to churn.

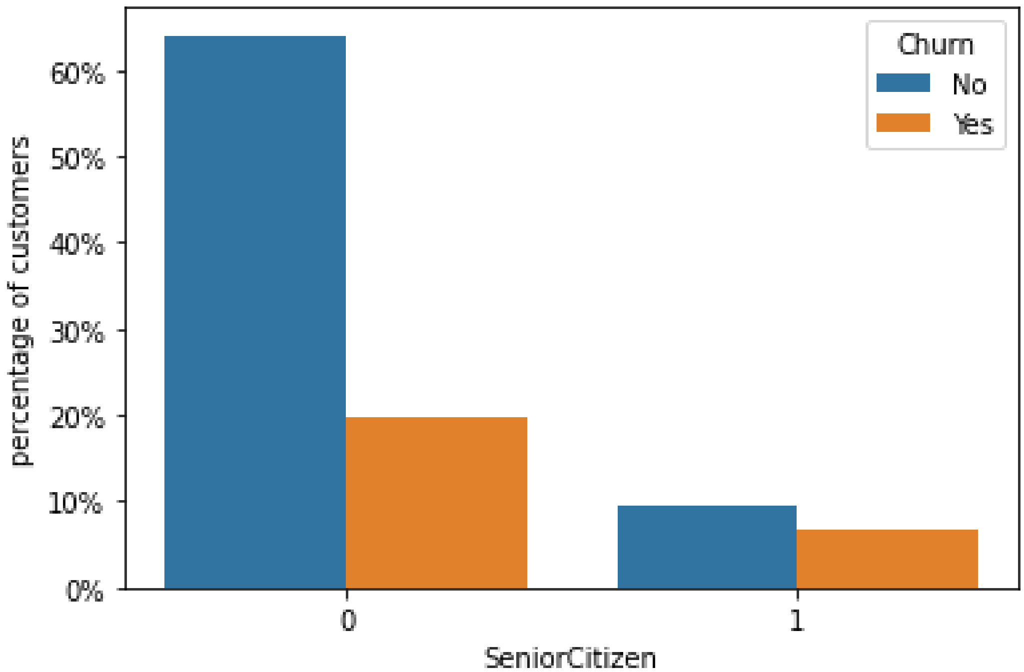
Clients with higher MonthlyCharges are also more likely to churn.

Tenure and MonthlyCharges are probably important features.

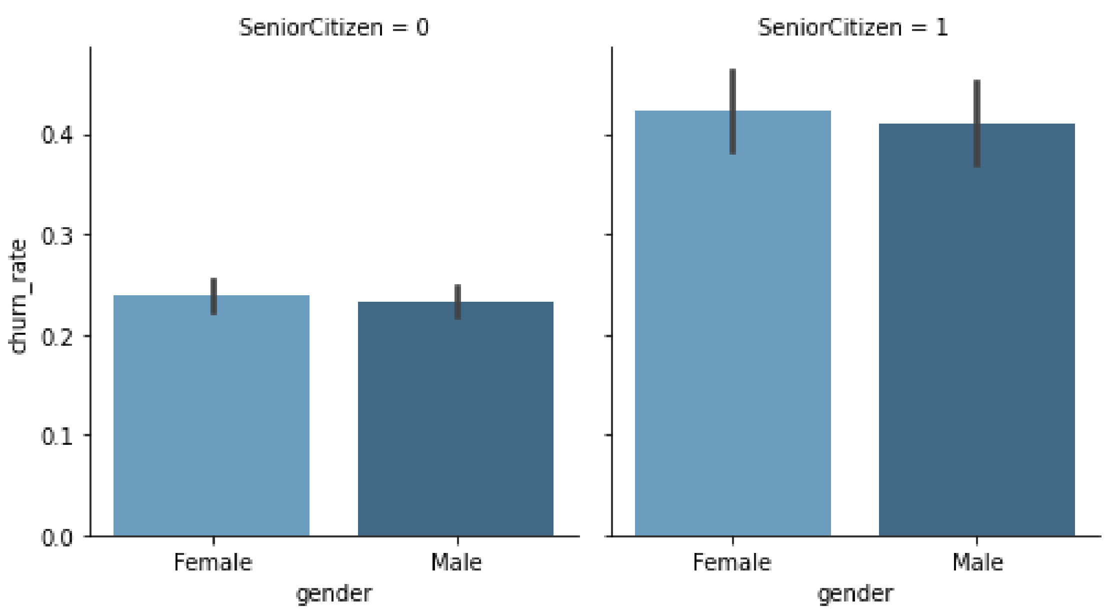
Categorical feautures This dataset has 16 categorical features.

Gender and Age(SeniorCitizen)

[ ]:
```
def barplot_percentages(feature, orient='v', axis_name="percentage of␣
 ↪customers"):
    ratios = pd.DataFrame()
    g = df.groupby(feature)["Churn"].value_counts().to_frame()
    g = g.rename({"Churn": axis_name}, axis=1).reset_index()
    g[axis_name] = g[axis_name]/len(df)
    if orient == 'v':
        ax = sns.barplot(x=feature, y= axis_name, hue='Churn', data=g,␣
 ↪orient=orient)
        ax.set_yticklabels(['{:,.0%}'.format(y) for y in ax.get_yticks()])
    else:
        ax = sns.barplot(x= axis_name, y=feature, hue='Churn', data=g,␣
 ↪orient=orient)
        ax.set_xticklabels(['{:,.0%}'.format(x) for x in ax.get_xticks()])
    ax.plot()
barplot_percentages("SeniorCitizen")
```



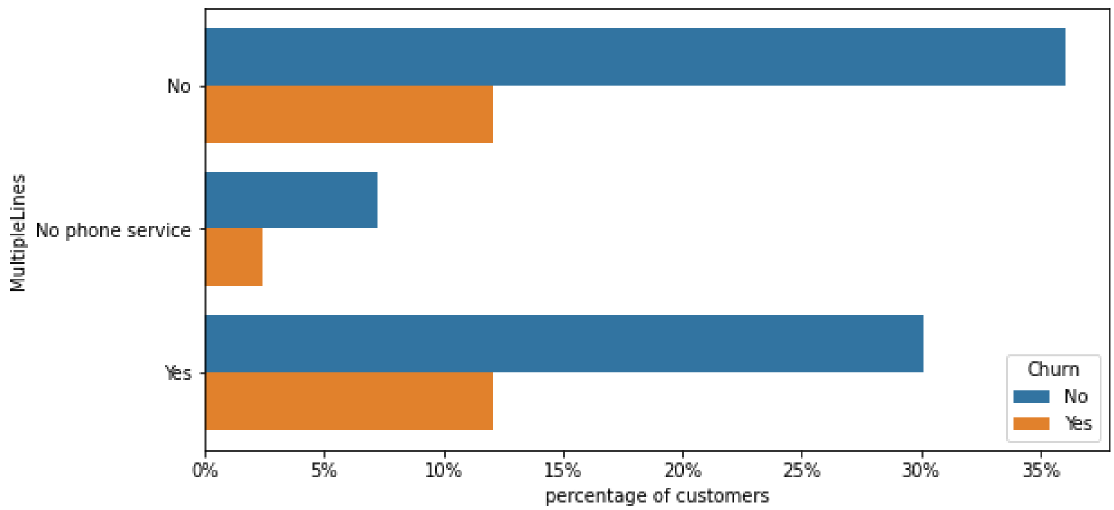
[ ]:
```
df['churn_rate'] = df['Churn'].replace("No", 0).replace("Yes", 1)
g = sns.FacetGrid(df, col="SeniorCitizen", height=4, aspect=.9)
ax = g.map(sns.barplot, "gender", "churn_rate", palette = "Blues_d", order=␣
↪['Female', 'Male'])
```



Gender is not an indicative of churn. SeniorCitizens are only 16% of customers, but they have a much higher churn rate: 42% against 23% for non-senior customers.

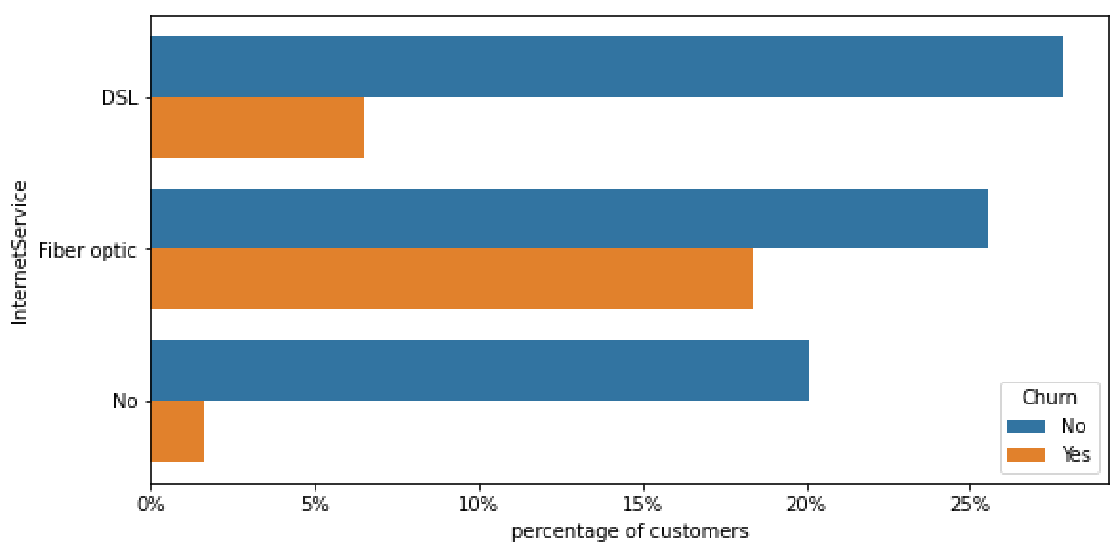
Phone and Internet services There are only two features here: If the client has phone and if he has more than one line. Both can be summed up in one chart:

[ ]:
```
plt.figure(figsize=(9, 4.5))
barplot_percentages("MultipleLines", orient='h')
```



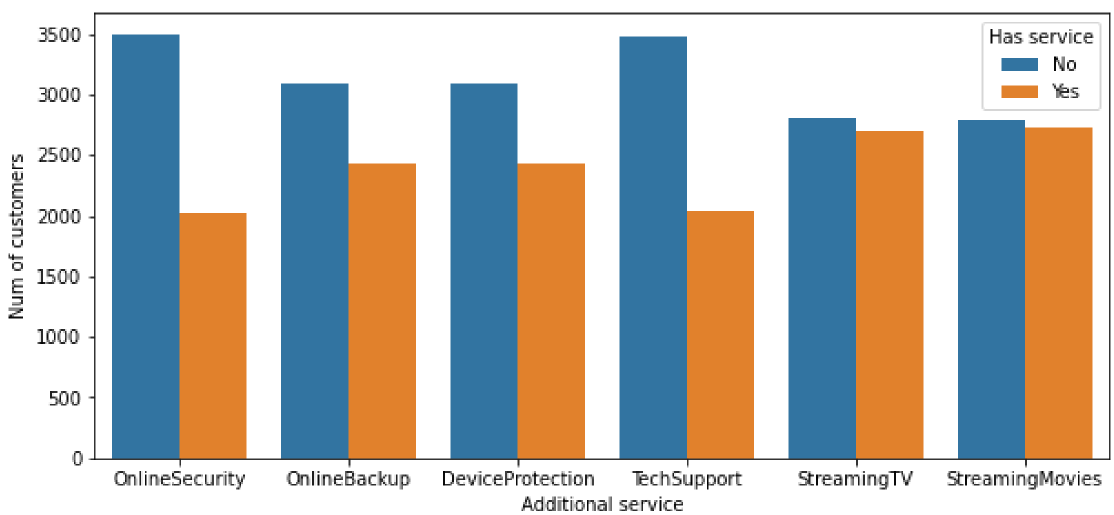
Customer with multiple lines have higher of churn rate.

[ ]:
```
plt.figure(figsize=(9, 4.5))
barplot_percentages("InternetService", orient="h")
```

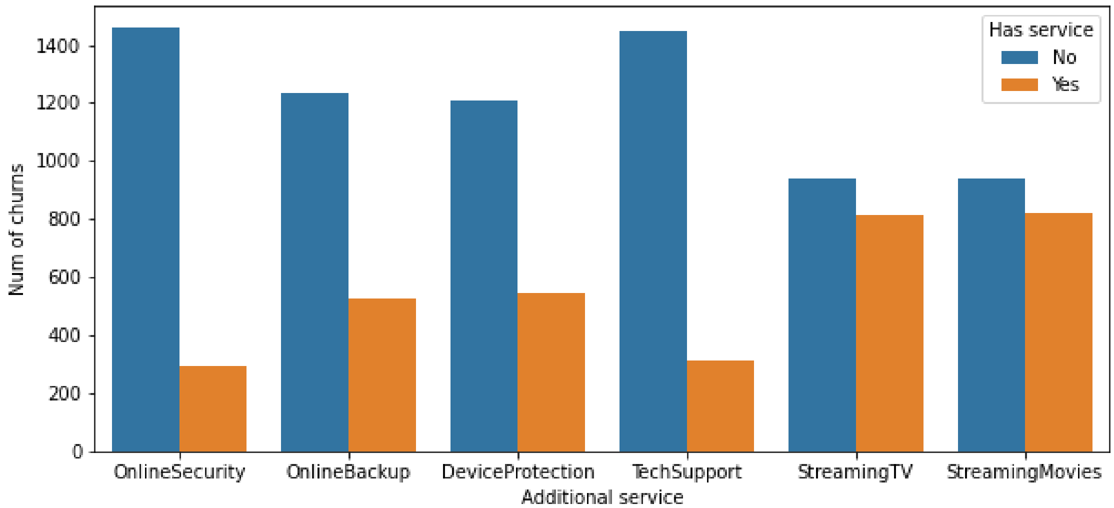


Additional service analysis The first plot shows the total number of customers for each additional service, while the second shows the number of clients that churn.

[ ]:
```
#There are six additional services for customers with internet:
cols = ["OnlineSecurity", "OnlineBackup", "DeviceProtection", "TechSupport",␣
↪"StreamingTV", "StreamingMovies"]
df1 = pd.melt(df[df["InternetService"] != "No"][cols]).rename({'value': 'Has␣
↪service'}, axis=1)
plt.figure(figsize=(10, 4.5))
ax = sns.countplot(data=df1, x='variable', hue='Has service')
ax.set(xlabel='Additional service', ylabel='Num of customers')
plt.show()
```



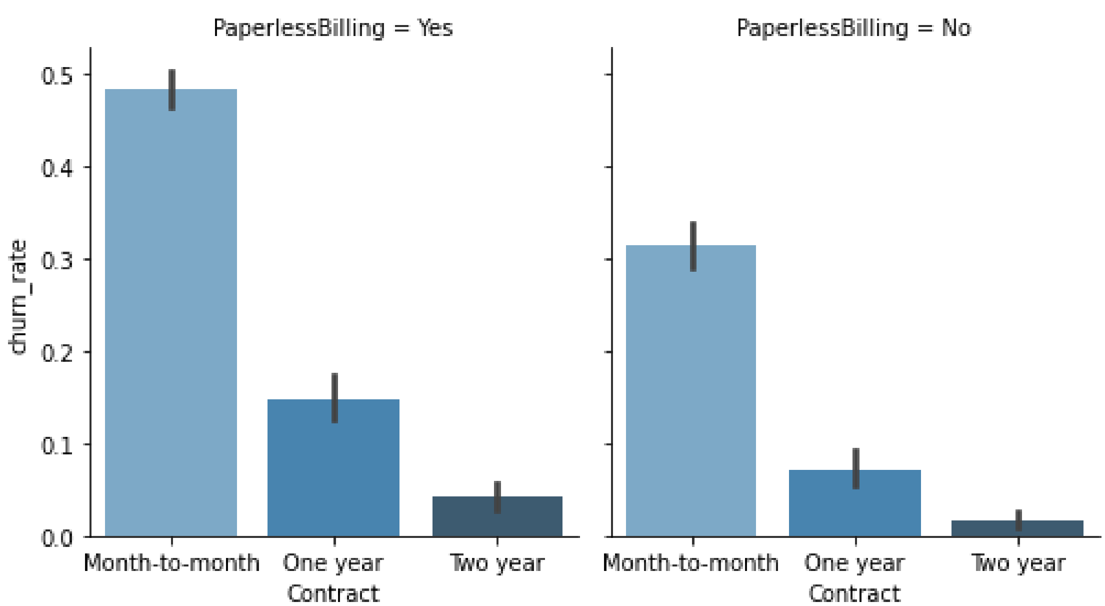
[ ]:
```
plt.figure(figsize=(10, 4.5))
df1 = df[(df.InternetService != "No") & (df.Churn == "Yes")]
df1 = pd.melt(df1[cols]).rename({'value': 'Has service'}, axis=1)
ax = sns.countplot(data=df1, x='variable', hue='Has service', hue_order=['No',␣
↪'Yes'])
ax.set(xlabel='Additional service', ylabel='Num of churns')
plt.show()
```



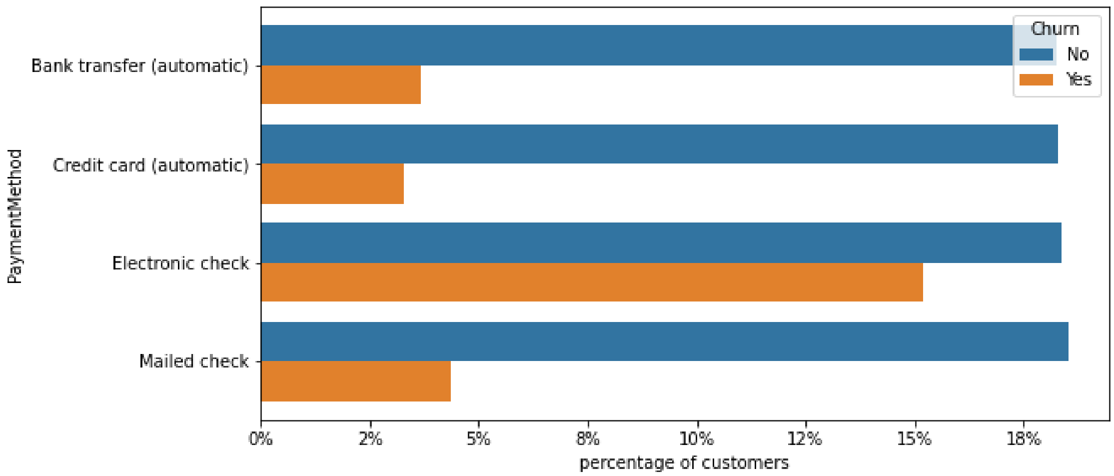
Customers with the OnlineSecurity, Backup,Protection and tech support are more unlikely to churn.

Contract and Payment Customers with paperless billing are more probable to churn. The preferred payment method is Electronic check with around 35% of customers. This also has a very high churn rate.

[ ]:
```
g = sns.FacetGrid(df, col="PaperlessBilling", height=4, aspect=.9)
ax = g.map(sns.barplot, "Contract", "churn_rate", palette = "Blues_d", order=␣
 ↪['Month-to-month', 'One year', 'Two year'])
```

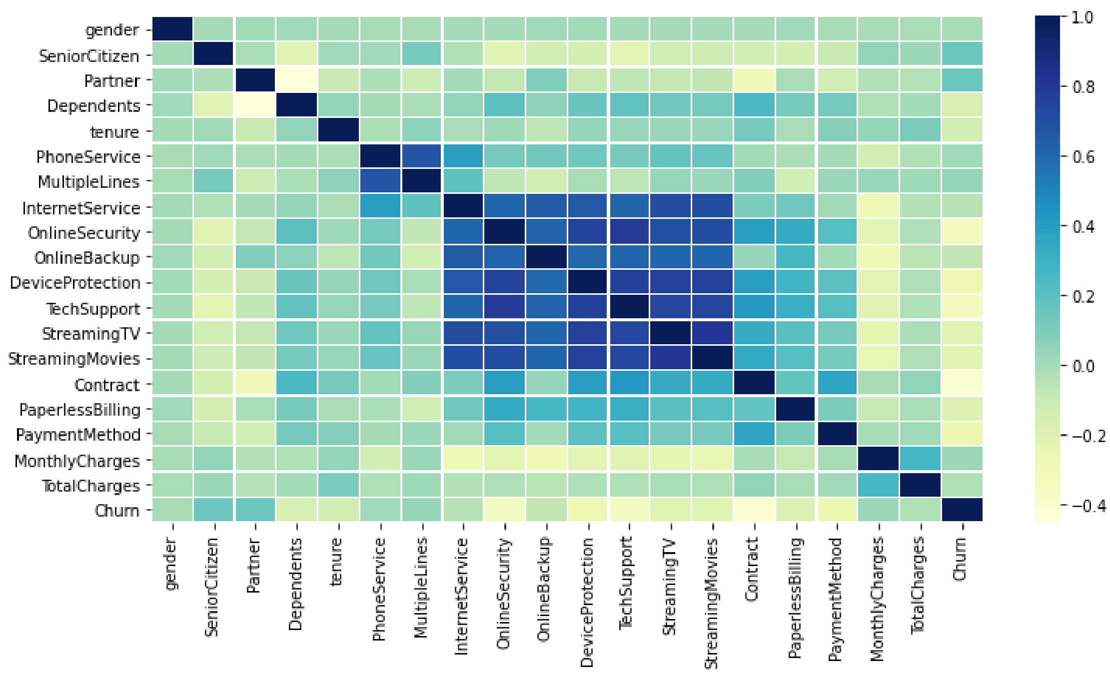


[ ]:
```
plt.figure(figsize=(9, 4.5))
barplot_percentages("PaymentMethod", orient='h')
```



Correlation between features

[ ]:
```
plt.figure(figsize=(12, 6))
df.drop(['customerID', 'churn_rate'],
        axis=1, inplace=True)
corr = df.apply(lambda x: pd.factorize(x)[0]).corr()
ax = sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,
                 linewidths=.2, cmap="YlGnBu")
```



1.0.6 Pre-process to build model

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

[ ]:
```
   gender  SeniorCitizen Partner  … MonthlyCharges  TotalCharges Churn
0  Female              0     Yes  …          29.85         29.85    No
1    Male              0      No  …          56.95       1889.50    No
2    Male              0      No  …          53.85        108.15   Yes
3    Male              0      No  …          42.30       1840.75    No
4  Female              0      No  …          70.70        151.65   Yes

[5 rows x 20 columns]
```

[ ]:
```
#Let's print unique values in object columns to see data values
def print_unique_col_values(df):
       for column in df:
            if df[column].dtypes=='object':
                print(f'{column}: {df[column].unique()}')
```

[ ]:
```
print_unique_col_values(df)
```

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

[ ]:
```
#Some of the columns have no internet service or no phone service, that can be␣
↪replaced with a simple No
df.replace('No internet service','No',inplace=True)
df.replace('No phone service','No',inplace=True)
```

[ ]:
```
print_unique_col_values(df)
```

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

[ ]:
```
df
```

[ ]:
```
      gender  SeniorCitizen Partner  … MonthlyCharges  TotalCharges Churn
0     Female              0     Yes  …          29.85         29.85    No
1       Male              0      No  …          56.95       1889.50    No
2       Male              0      No  …          53.85        108.15   Yes
3       Male              0      No  …          42.30       1840.75    No
4     Female              0      No  …          70.70        151.65   Yes
…        …              …       …    …            …             …     …
7038    Male              0     Yes  …          84.80       1990.50    No
7039  Female              0     Yes  …         103.20       7362.90    No
7040  Female              0     Yes  …          29.60        346.45    No
7041    Male              1     Yes  …          74.40        306.60   Yes
7042    Male              0      No  …         105.65       6844.50    No

[7043 rows x 20 columns]
```

[ ]:
```
#Here, converting Yes= 1, No =0

yes_no_columns =
 ↪['Partner','Dependents','PhoneService','MultipleLines','OnlineSecurity','OnlineBackup',
 ↪'DeviceProtection','TechSupport','StreamingTV','StreamingMovies','PaperlessBilling','Churn'
for col in yes_no_columns:
    df[col].replace({'Yes': 1,'No': 0},inplace=True)
```

[ ]:
```
for col in df:
    print(f'{col}: {df[col].unique()}')
```

```
gender: ['Female' 'Male']
```

```
SeniorCitizen: [0 1]
Partner: [1 0]
Dependents: [0 1]
tenure: [ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26  0
 39]
PhoneService: [0 1]
MultipleLines: [0 1]
InternetService: ['DSL' 'Fiber optic' 'No']
OnlineSecurity: [0 1]
OnlineBackup: [1 0]
DeviceProtection: [0 1]
TechSupport: [0 1]
StreamingTV: [0 1]
StreamingMovies: [0 1]
Contract: ['Month-to-month' 'One year' 'Two year']
PaperlessBilling: [1 0]
PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
MonthlyCharges: [29.85 56.95 53.85 … 63.1  44.2  78.7 ]
TotalCharges: [  29.85 1889.5   108.15 …  346.45  306.6  6844.5 ]
Churn: [0 1]
```

[ ]:
```
df['gender'].replace({'Female':1,'Male':0},inplace=True)
```

[ ]:
```
df.gender.unique()
```

[ ]:
```
array([1, 0])
```

One hot encoding for categorical columns

[ ]:
```
df1 = pd.get_dummies(data=df,␣
 ↪columns=['InternetService','Contract','PaymentMethod'])
df1.columns
```

[ ]:
```
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'PaperlessBilling', 'MonthlyCharges', 'TotalCharges', 'Churn',
       'InternetService_DSL', 'InternetService_Fiber optic',
       'InternetService_No', 'Contract_Month-to-month', 'Contract_One year',
       'Contract_Two year', 'PaymentMethod_Bank transfer (automatic)',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check'],
      dtype='object')
```

[ ]:
```
df1.sample(5)
```

[ ]:
```
      gender  ...  PaymentMethod_Mailed check
6383       0  ...                           1
5006       1  ...                           1
1189       0  ...                           0
1956       1  ...                           0
4435       0  ...                           0

[5 rows x 27 columns]
```

[ ]:
```
df1.dtypes
```

[ ]:
```
gender                                       int64
SeniorCitizen                                int64
Partner                                      int64
Dependents                                   int64
tenure                                       int64
PhoneService                                 int64
MultipleLines                                int64
OnlineSecurity                               int64
OnlineBackup                                 int64
DeviceProtection                             int64
TechSupport                                  int64
StreamingTV                                  int64
StreamingMovies                              int64
PaperlessBilling                             int64
MonthlyCharges                             float64
TotalCharges                               float64
Churn                                        int64
InternetService_DSL                          uint8
InternetService_Fiber optic                  uint8
InternetService_No                           uint8
Contract_Month-to-month                      uint8
Contract_One year                            uint8
Contract_Two year                            uint8
PaymentMethod_Bank transfer (automatic)      uint8
PaymentMethod_Credit card (automatic)        uint8
PaymentMethod_Electronic check               uint8
PaymentMethod_Mailed check                   uint8
dtype: object
```

[ ]:
```
cols_to_scale = ['tenure','MonthlyCharges','TotalCharges']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df1[cols_to_scale] = scaler.fit_transform(df1[cols_to_scale])
```

[ ]:
```python
for col in df1:
    print(f'{col}: {df1[col].unique()}')
```

```
gender: [1 0]
SeniorCitizen: [0 1]
Partner: [1 0]
Dependents: [0 1]
tenure: [0.01388889 0.47222222 0.02777778 0.625      0.11111111 0.30555556
 0.13888889 0.38888889 0.86111111 0.18055556 0.22222222 0.80555556
 0.68055556 0.34722222 0.95833333 0.72222222 0.98611111 0.29166667
 0.16666667 0.41666667 0.65277778 1.         0.23611111 0.375
 0.06944444 0.63888889 0.15277778 0.97222222 0.875      0.59722222
 0.20833333 0.83333333 0.25       0.91666667 0.125      0.04166667
 0.43055556 0.69444444 0.88888889 0.77777778 0.09722222 0.58333333
 0.48611111 0.66666667 0.40277778 0.90277778 0.52777778 0.94444444
 0.44444444 0.76388889 0.51388889 0.5        0.56944444 0.08333333
 0.05555556 0.45833333 0.93055556 0.31944444 0.79166667 0.84722222
 0.19444444 0.27777778 0.73611111 0.55555556 0.81944444 0.33333333
 0.61111111 0.26388889 0.75       0.70833333 0.36111111 0.
 0.54166667]
PhoneService: [0 1]
MultipleLines: [0 1]
OnlineSecurity: [0 1]
OnlineBackup: [1 0]
DeviceProtection: [0 1]
TechSupport: [0 1]
StreamingTV: [0 1]
StreamingMovies: [0 1]
PaperlessBilling: [1 0]
MonthlyCharges: [0.11542289 0.38507463 0.35422886 … 0.44626866 0.25820896
0.60149254]
TotalCharges: [0.00343704 0.21756402 0.01245279 … 0.03989153 0.03530306
0.78810105]
Churn: [0 1]
InternetService_DSL: [1 0]
InternetService_Fiber optic: [0 1]
InternetService_No: [0 1]
Contract_Month-to-month: [1 0]
Contract_One year: [0 1]
Contract_Two year: [0 1]
PaymentMethod_Bank transfer (automatic): [0 1]
PaymentMethod_Credit card (automatic): [0 1]
PaymentMethod_Electronic check: [1 0]
PaymentMethod_Mailed check: [0 1]
```

Train test split

```
[ ]: X = df1.drop('Churn',axis='columns')
     y = df1['Churn']

     from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
      ↪2,random_state=5)
```

```
[ ]: X_train.shape
```

```
[ ]: (5634, 26)
```

```
[ ]: X_test.shape
```

```
[ ]: (1409, 26)
```

```
[ ]: X_train[:10]
```

```
[ ]:       gender  …  PaymentMethod_Mailed check
     5860       1  …                            0
     2458       0  …                            0
     5879       0  …                            1
     4708       1  …                            0
     1293       0  …                            0
     2242       0  …                            0
     1444       0  …                            0
     3269       0  …                            0
     101        1  …                            0
     4191       1  …                            0

     [10 rows x 26 columns]
```

```
[ ]: len(X_train.columns)
```

```
[ ]: 26
```

1.0.7 Build a model (ANN) in tensorflow/keras

```
[ ]: model = keras.Sequential([
         keras.layers.Dense(26, input_shape=(26,), activation='relu'),
         keras.layers.Dense(15, activation='relu'),
         keras.layers.Dense(1, activation='sigmoid')
     ])

     # opt = keras.optimizers.Adam(learning_rate=0.01)

     model.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])
```

```python
model.fit(X_train, y_train, epochs=100)
```

```
Epoch 1/100
177/177 [==============================] - 0s 1ms/step - loss: 0.5035 -
accuracy: 0.7474
Epoch 2/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4302 -
accuracy: 0.7930
Epoch 3/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4229 -
accuracy: 0.7980
Epoch 4/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4181 -
accuracy: 0.8055
Epoch 5/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4145 -
accuracy: 0.8067
Epoch 6/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4132 -
accuracy: 0.8046
Epoch 7/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4118 -
accuracy: 0.8090
Epoch 8/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4093 -
accuracy: 0.8080
Epoch 9/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4086 -
accuracy: 0.8094
Epoch 10/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4080 -
accuracy: 0.8083
Epoch 11/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4061 -
accuracy: 0.8080
Epoch 12/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4037 -
accuracy: 0.8126
Epoch 13/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4040 -
accuracy: 0.8131
Epoch 14/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4034 -
accuracy: 0.8115
Epoch 15/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4013 -
accuracy: 0.8131
```

```
Epoch 16/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4001 -
accuracy: 0.8168
Epoch 17/100
177/177 [==============================] - 0s 1ms/step - loss: 0.4010 -
accuracy: 0.8131
Epoch 18/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3993 -
accuracy: 0.8154
Epoch 19/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3996 -
accuracy: 0.8156
Epoch 20/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3985 -
accuracy: 0.8165
Epoch 21/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3971 -
accuracy: 0.8175
Epoch 22/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3961 -
accuracy: 0.8181
Epoch 23/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3952 -
accuracy: 0.8154
Epoch 24/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3941 -
accuracy: 0.8175
Epoch 25/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3955 -
accuracy: 0.8175
Epoch 26/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3925 -
accuracy: 0.8197
Epoch 27/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3931 -
accuracy: 0.8170
Epoch 28/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3924 -
accuracy: 0.8179
Epoch 29/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3910 -
accuracy: 0.8172
Epoch 30/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3905 -
accuracy: 0.8193
Epoch 31/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3909 -
accuracy: 0.8206
```

```
Epoch 32/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3895 -
accuracy: 0.8195
Epoch 33/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3878 -
accuracy: 0.8181
Epoch 34/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3887 -
accuracy: 0.8179
Epoch 35/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3874 -
accuracy: 0.8202
Epoch 36/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3864 -
accuracy: 0.8191
Epoch 37/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3863 -
accuracy: 0.8200
Epoch 38/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3862 -
accuracy: 0.8181
Epoch 39/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3860 -
accuracy: 0.8179
Epoch 40/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3858 -
accuracy: 0.8211
Epoch 41/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3836 -
accuracy: 0.8200
Epoch 42/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3842 -
accuracy: 0.8223
Epoch 43/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3827 -
accuracy: 0.8204
Epoch 44/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3831 -
accuracy: 0.8220
Epoch 45/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3831 -
accuracy: 0.8222
Epoch 46/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3817 -
accuracy: 0.8198
Epoch 47/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3817 -
accuracy: 0.8214
```

```
Epoch 48/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3812 -
accuracy: 0.8222
Epoch 49/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3809 -
accuracy: 0.8227
Epoch 50/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3799 -
accuracy: 0.8211
Epoch 51/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3797 -
accuracy: 0.8218
Epoch 52/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3796 -
accuracy: 0.8220
Epoch 53/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3788 -
accuracy: 0.8232
Epoch 54/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3785 -
accuracy: 0.8246
Epoch 55/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3776 -
accuracy: 0.8211
Epoch 56/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3783 -
accuracy: 0.8236
Epoch 57/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3771 -
accuracy: 0.8245
Epoch 58/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3762 -
accuracy: 0.8220
Epoch 59/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3757 -
accuracy: 0.8248
Epoch 60/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3759 -
accuracy: 0.8237
Epoch 61/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3750 -
accuracy: 0.8268
Epoch 62/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3752 -
accuracy: 0.8243
Epoch 63/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3758 -
accuracy: 0.8255
```

```
Epoch 64/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3735 -
accuracy: 0.8250
Epoch 65/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3732 -
accuracy: 0.8271
Epoch 66/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3738 -
accuracy: 0.8248
Epoch 67/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3735 -
accuracy: 0.8229
Epoch 68/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3715 -
accuracy: 0.8236
Epoch 69/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3713 -
accuracy: 0.8241
Epoch 70/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3718 -
accuracy: 0.8252
Epoch 71/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3720 -
accuracy: 0.8259
Epoch 72/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3714 -
accuracy: 0.8241
Epoch 73/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3712 -
accuracy: 0.8264
Epoch 74/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3703 -
accuracy: 0.8262
Epoch 75/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3696 -
accuracy: 0.8230
Epoch 76/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3690 -
accuracy: 0.8282
Epoch 77/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3691 -
accuracy: 0.8285
Epoch 78/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3691 -
accuracy: 0.8252
Epoch 79/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3690 -
accuracy: 0.8259
```

```
Epoch 80/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3676 -
accuracy: 0.8259
Epoch 81/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3681 -
accuracy: 0.8239
Epoch 82/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3675 -
accuracy: 0.8296
Epoch 83/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3661 -
accuracy: 0.8236
Epoch 84/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3651 -
accuracy: 0.8278
Epoch 85/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3655 -
accuracy: 0.8273
Epoch 86/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3653 -
accuracy: 0.8259
Epoch 87/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3651 -
accuracy: 0.8278
Epoch 88/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3653 -
accuracy: 0.8241
Epoch 89/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3637 -
accuracy: 0.8289
Epoch 90/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3647 -
accuracy: 0.8284
Epoch 91/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3630 -
accuracy: 0.8294
Epoch 92/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3647 -
accuracy: 0.8271
Epoch 93/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3635 -
accuracy: 0.8246
Epoch 94/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3623 -
accuracy: 0.8269
Epoch 95/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3625 -
accuracy: 0.8271
```

```
Epoch 96/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3611 -
accuracy: 0.8277
Epoch 97/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3608 -
accuracy: 0.8291
Epoch 98/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3611 -
accuracy: 0.8300
Epoch 99/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3616 -
accuracy: 0.8280
Epoch 100/100
177/177 [==============================] - 0s 1ms/step - loss: 0.3607 -
accuracy: 0.8285
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x7f877bfcde80>
```

[ ]:
```
model.evaluate(X_test, y_test)
```

```
45/45 [==============================] - 0s 949us/step - loss: 0.4588 -
accuracy: 0.7779
```

[ ]:
```
[0.458802729845047, 0.7778566479682922]
```

[ ]:
```
yp = model.predict(X_test)
yp[:5]
```

[ ]:
```
array([[0.36123043],
       [0.50069755],
       [0.32254955],
       [0.76979315],
       [0.09441373]], dtype=float32)
```

[ ]:
```
y_pred = []
for element in yp:
    if element > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

[ ]:
```
y_pred[:10]
```

[ ]:
```
[0, 1, 0, 1, 0, 1, 1, 1, 0, 0]
```

[ ]:
```
y_test[:10]
```

[ ]:
```
4213    1
5035    0
3713    1
1720    0
234     0
4558    1
40      0
3455    1
5944    1
1089    0
Name: Churn, dtype: int64
```

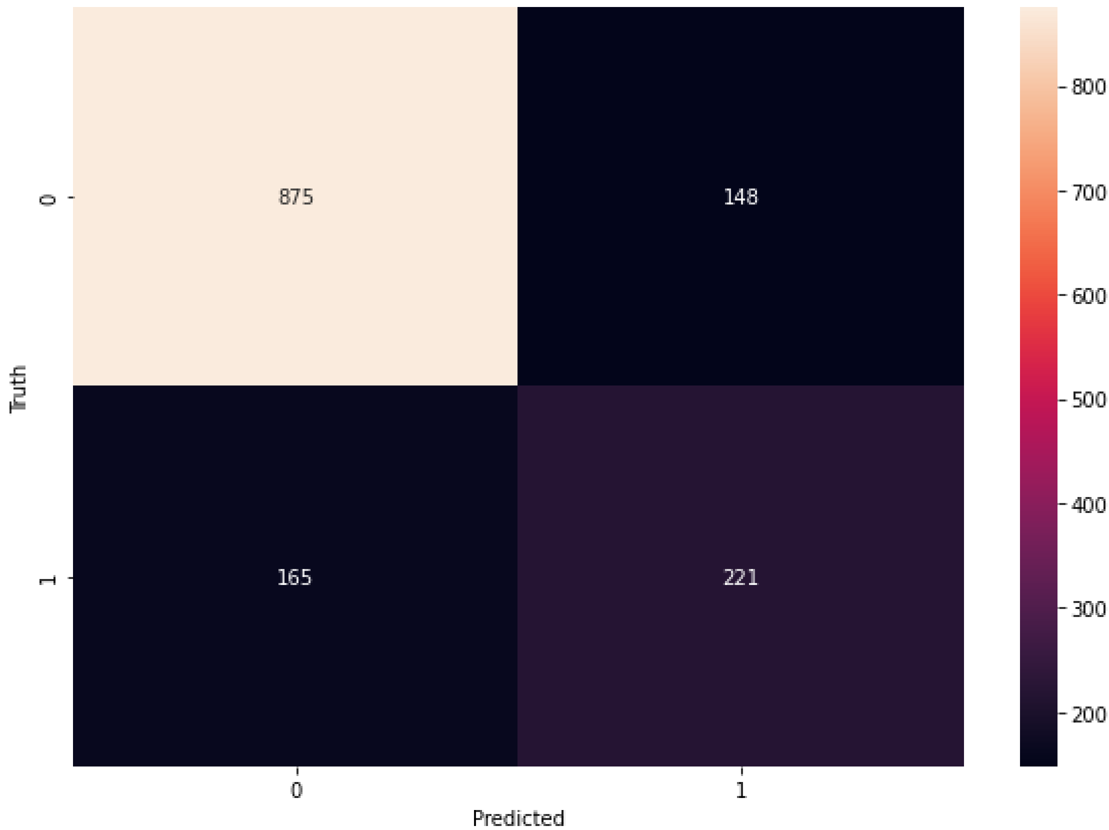
Confusion matrix

[ ]:
```python
from sklearn.metrics import confusion_matrix , classification_report

print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.86      0.85      1023
           1       0.60      0.57      0.59       386

    accuracy                           0.78      1409
   macro avg       0.72      0.71      0.72      1409
weighted avg       0.77      0.78      0.78      1409
```

[ ]:
```python
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_pred)

plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

[ ]:
```
Text(69.0, 0.5, 'Truth')
```



[ ]:
```
y_test.shape
```

[ ]: (1409,)

Accuracy

[ ]:
```
round((862+229)/(862+229+137+179),2)
```

[ ]: 0.78

Precision for 0 class. i.e. Precision for customers who did not churn

[ ]:
```
round(862/(862+179),2)
```

[ ]: 0.83

Precision for 1 class. i.e. Precision for customers who actually churned

[ ]:
```
round(229/(229+137),2)
```

[ ]: 0.63

Recall for 0 class

[ ]:
```
round(862/(862+137),2)
```

[ ]:
```
0.86
```

[ ]:
```
round(229/(229+179),2)
```

[ ]:
```
0.56
```

[ ]:
```

```