Name - Omkar Phansopkar

UID - 2022701007

D3 - CSE-DS

Dataset - Telco customer churn: IBM dataset

https://www.kaggle.com/datasets/yeanzc/telco-customer-churn-ibm-dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remc
```

```
!pip install branca plotly scikit-learn scipy imblearn --quiet
!pip install h3 folium --quiet
```

```
import numpy as np
from scipy import stats
from IPython.display import Image
import branca.colormap as cm
import pandas as pd
import seaborn as sns
import plotly.express as px
from plotly.offline import init_notebook_mode, iplot
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
import h3
import matplotlib
import imblearn
import os
import folium
from sklearn.impute import SimpleImputer
```

Data Description

CustomerID: A unique ID that identifies each customer.

Count: A value used in reporting/dashboarding to sum up the number of customers in a filtered set.

Country: The country of the customer's primary residence.

State: The state of the customer's primary residence.

City: The city of the customer's primary residence.

Zip Code: The zip code of the customer's primary residence.

Lat Long: The combined latitude and longitude of the customer's primary residence.

Latitude: The latitude of the customer's primary residence.

Longitude: The longitude of the customer's primary residence.

Gender: The customer's gender: Male, Female

Senior Citizen: Indicates if the customer is 65 or older: Yes, No

Partner: Indicate if the customer has a partner: Yes, No

Dependents: Indicates if the customer lives with any dependents: Yes, No. Dependents could be children, parents, grandparents, etc.

Tenure Months: Indicates the total amount of months that the customer has been with the company by the end of the quarter specified above.

Phone Service: Indicates if the customer subscribes to home phone service with the company: Yes, No

Multiple Lines: Indicates if the customer subscribes to multiple telephone lines with the company: Yes, No

Internet Service: Indicates if the customer subscribes to Internet service with the company: No, DSL, Fiber Optic, Cable.

Online Security: Indicates if the customer subscribes to an additional online security service provided by the company: Yes, No

Online Backup: Indicates if the customer subscribes to an additional online backup service provided by the company: Yes, No

Device Protection: Indicates if the customer subscribes to an additional device protection plan for their Internet equipment provided by the company: Yes, No

Tech Support: Indicates if the customer subscribes to an additional technical support plan from the company with reduced wait times: Yes, No

Streaming TV: Indicates if the customer uses their Internet service to stream television programing from a third party provider: Yes, No. The company does not charge an additional fee for this service.

Streaming Movies: Indicates if the customer uses their Internet service to stream movies from a third party provider: Yes, No. The company does not charge an additional fee for this service.

Contract: Indicates the customer's current contract type: Month-to-Month, One Year, Two Year.

Paperless Billing: Indicates if the customer has chosen paperless billing: Yes, No

Payment Method: Indicates how the customer pays their bill: Bank Withdrawal, Credit Card, Mailed Check

Monthly Charge: Indicates the customer's current total monthly charge for all their services from the company.

Total Charges: Indicates the customer's total charges, calculated to the end of the quarter specified above.

Churn Label: Yes = the customer left the company this quarter. No = the customer remained with the company. Directly related to Churn Value.

Churn Value: 1 = the customer left the company this quarter. 0 = the customer remained with the company. Directly related to Churn Label.

Churn Score: A value from 0-100 that is calculated using the predictive tool IBM SPSS Modeler. The model incorporates multiple factors known to cause churn. The higher the score, the more likely the customer will churn.

CLTV: Customer Lifetime Value. A predicted CLTV is calculated using corporate formulas and existing data. The higher the value, the more valuable the customer. High value customers should be monitored for churn.

Churn Reason: A customer's specific reason for leaving the company. Directly related to Churn Category.

Source This dataset is detailed in: https://community.ibm.com/community/user/businessanalytics/blogs/steven-macko/2019/07/11/telco-customer-churn-1113

Downloaded from: https://community.ibm.com/accelerators/?context=analytics&query=telco%20churn&type=Data&product=Cognos%20Analytics

There are several related datasets as documented in: https://community.ibm.com/community/user/businessanalytics/blogs/steven-macko/2018/09/12/base-samples-for-ibm-cognos-analytics

```
data = pd.read_excel('/content/drive/MyDrive/datasets/fds_customer_churn/Telco_customer_churn.xlsx')
data.sample()
```

	CustomerID	Count	Country	State	City	Zip Code	Lat Long	Latitude
1565	4192-GORJT	1	United States	California	Fremont	94555	37.555473, -122.080312	37.555473

1 rows × 33 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 33 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   CustomerID         7043 non-null   object 
 1   Count              7043 non-null   int64  
 2   Country            7043 non-null   object 
 3   State              7043 non-null   object 
 4   City               7043 non-null   object 
 5   Zip Code           7043 non-null   int64  
 6   Lat Long           7043 non-null   object 
 7   Latitude           7043 non-null   float64
 8   Longitude          7043 non-null   float64
 9   Gender             7043 non-null   object 
 10  Senior Citizen     7043 non-null   object 
 11  Partner            7043 non-null   object 
 12  Dependents         7043 non-null   object 
 13  Tenure Months      6995 non-null   float64
 14  Phone Service      7043 non-null   object 
 15  Multiple Lines     7043 non-null   object 
 16  Internet Service   7043 non-null   object 
 17  Online Security    7043 non-null   object 
 18  Online Backup      7043 non-null   object 
 19  Device Protection  7043 non-null   object 
 20  Tech Support       7043 non-null   object 
 21  Streaming TV       7043 non-null   object 
 22  Streaming Movies   7043 non-null   object 
 23  Contract           7043 non-null   object 
 24  Paperless Billing  7043 non-null   object 
 25  Payment Method     7043 non-null   object 
```

```
 26  Monthly Charges    7043 non-null   float64
 27  Total Charges      7043 non-null   object
 28  Churn Label        7043 non-null   object
 29  Churn Value        7043 non-null   int64
 30  Churn Score        7043 non-null   int64
 31  CLTV               7043 non-null   int64
 32  Churn Reason       1869 non-null   object
dtypes: float64(4), int64(5), object(24)
memory usage: 1.8+ MB
```

- what services customers use,
- type of contract
- the lifetime of the client in the service
- payment method
- the amount of monthly payments of customers and their total costs in the service,
- customer locations,
- gender and age of the client
- reason for churn (for clients in the churn)

```
data['Total Charges']
```

```
0        108.15
1        151.65
2         820.5
3       3046.05
4        5036.3
         ...
7038     1419.4
7039     1990.5
7040     7362.9
7041     346.45
7042     6844.5
Name: Total Charges, Length: 7043, dtype: object
```

```
data['Total Charges'] = pd.to_numeric(data['Total Charges'], errors='coerce')
data['Total Charges']
```

```
0        108.15
1        151.65
2        820.50
3       3046.05
4       5036.30
         ...
7038    1419.40
7039    1990.50
7040    7362.90
7041     346.45
7042    6844.50
Name: Total Charges, Length: 7043, dtype: float64
```

```
data.isnull().sum()
```

```
CustomerID            0
Count                 0
Country               0
State                 0
City                  0
Zip Code              0
Lat Long              0
Latitude              0
Longitude             0
Gender                0
Senior Citizen        0
Partner               0
Dependents            0
Tenure Months        48
Phone Service         0
Multiple Lines        0
Internet Service      0
Online Security       0
Online Backup         0
Device Protection     0
Tech Support          0
Streaming TV          0
Streaming Movies      0
Contract              0
Paperless Billing     0
Payment Method        0
Monthly Charges       0
Total Charges        11
Churn Label           0
```

```
Churn Value            0
Churn Score            0
CLTV                   0
Churn Reason        5174
dtype: int64
```

```
data.isnull().sum() / len(data)
```

```
CustomerID           0.000000
Count                0.000000
Country              0.000000
State                0.000000
City                 0.000000
Zip Code             0.000000
Lat Long             0.000000
Latitude             0.000000
Longitude            0.000000
Gender               0.000000
Senior Citizen       0.000000
Partner              0.000000
Dependents           0.000000
Tenure Months        0.006815
Phone Service        0.000000
Multiple Lines       0.000000
Internet Service     0.000000
Online Security      0.000000
Online Backup        0.000000
Device Protection    0.000000
Tech Support         0.000000
Streaming TV         0.000000
Streaming Movies     0.000000
Contract             0.000000
Paperless Billing    0.000000
Payment Method       0.000000
Monthly Charges      0.000000
Total Charges        0.001562
Churn Label          0.000000
Churn Value          0.000000
Churn Score          0.000000
CLTV                 0.000000
Churn Reason         0.734630
dtype: float64
```

```
data[data['Tenure Months'].isnull()]
```

	CustomerID	Count	Country	State	City	Zip Code	Lat Long	Latitude	Longitude	Gender	...	Contract	Paper Bil
217	9944-HKVVB	1	United States	California	King City	93930	36.220761, -120.980777	36.220761	-120.980777	Female	...	Month-to-month	
258	9524-EGPJC	1	United States	California	Walnut Creek	94595	37.862128, -122.075197	37.862128	-122.075197	Female	...	Month-to-month	
332	6818-WOBHJ	1	United States	California	Fulton	95439	38.493888, -122.777141	38.493888	-122.777141	Female	...	Month-to-month	
416	9391-EOYLI	1	United States	California	Lewiston	96052	40.704293, -122.803899	40.704293	-122.803899	Male	...	Month-to-month	
419	3068-OMWZA	1	United States	California	Round Mountain	96084	40.923558, -122.059933	40.923558	-122.059933	Male	...	Month-to-month	
438	2058-DCJBE	1	United States	California	Los Angeles	90012	34.065875, -118.238728	34.065875	-118.238728	Male	...	Month-to-month	
515	9497-QCMMS	1	United States	California	Escondido	92026	33.21846, -117.116916	33.218460	-117.116916	Male	...	Month-to-month	
568	3023-GFLBR	1	United States	California	San Bernardino	92408	34.084909, -117.258107	34.084909	-117.258107	Female	...	Month-to-month	
573	9061-TIHDA	1	United States	California	Murrieta	92563	33.581045, -117.14719	33.581045	-117.147190	Male	...	Month-to-month	
637	9221-OTIVJ	1	United States	California	Dunlap	93621	36.789213, -119.140338	36.789213	-119.140338	Female	...	Month-to-month	
666	0407-BDJKB	1	United States	California	San Francisco	94114	37.758085, -122.434801	37.758085	-122.434801	Male	...	Month-to-month	
669	3269-ATYWD	1	United States	California	San Francisco	94123	37.800254, -122.436975	37.800254	-122.436975	Male	...	Month-to-month	
749	9992-RRAMN	1	United States	California	Riverbank	95367	37.734971, -120.954271	37.734971	-120.954271	Male	...	Month-to-month	
824	5649-ANRML	1	United States	California	Clipper Mills	95930	39.562239, -121.14836	39.562239	-121.148360	Male	...	Month-to-month	
830	8393-DLHGA	1	United States	California	Palermo	95968	39.435756, -121.552071	39.435756	-121.552071	Male	...	Month-to-month	
842	5519-NPHVG	1	United States	California	Klamath River	96050	41.816595, -122.948287	41.816595	-122.948287	Female	...	Month-to-month	
850	5498-TXHLF	1	United States	California	Whitmore	96096	40.637105, -121.906949	40.637105	-121.906949	Female	...	Month-to-month	
937	0133-BMFZO	1	United States	California	Alhambra	91803	34.074736, -118.145959	34.074736	-118.145959	Female	...	Month-to-month	
942	8623-TMRBY	1	United States	California	Chula Vista	91915	32.605012, -116.97595	32.605012	-116.975950	Male	...	Month-to-month	
944	3208-YPIOE	1	United States	California	La Mesa	91942	32.782501, -117.01611	32.782501	-117.016110	Male	...	Month-to-month	
945	2612-RANWT	1	United States	California	Mount Laguna	91948	32.830852, -116.444601	32.830852	-116.444601	Female	...	Month-to-month	
956	3194-ORPIK	1	United States	California	San Diego	92113	32.697098, -117.116587	32.697098	-117.116587	Female	...	Month-to-month	
1055	0618-XWMSS	1	United States	California	Wasco	93280	35.652242, -119.4464	35.652242	-119.446400	Male	...	Month-to-month	
1056	5276-KQWHG	1	United States	California	Woody	93287	35.710244, -118.881679	35.710244	-118.881679	Female	...	Month-to-month	
[illegible]	2357-	[illegible]	United	[illegible]	[illegible]	[illegible]	37.454924,	[illegible]	[illegible]	[illegible]		Month-to-	

1099	COQEK	1	States	California	Atherton	94027	-122.203168	37.454924	-122.203168	Female	...	month
1157	5449-FIBXJ	1	United States	California	San Jose	95127	37.375156, -121.795867	37.375156	-121.795867	Male	...	Month-to-month
1158	9837-BMCLM	1	United States	California	Stockton	95202	37.959706, -121.287669	37.959706	-121.287669	Male	...	Month-to-month
1244	5729-KLZAR	1	United States	California	Sacramento	95817	38.550722, -121.457314	38.550722	-121.457314	Female	...	Month-to-month
1246	8740-CRYFY	1	United States	California	Sacramento	95823	38.475465, -121.443625	38.475465	-121.443625	Male	...	Month-to-month
1475	6284-AHOOQ	1	United States	California	Avenal	93204	35.916943, -120.129921	35.916943	-120.129921	Male	...	Month-to-month
1480	4818-DRBQT	1	United States	California	Farmersville	93223	36.29878, -119.201028	36.298780	-119.201028	Male	...	Month-to-month
1484	4391-RESHN	1	United States	California	Lemon Cove	93244	36.462671, -118.997291	36.462671	-118.997291	Male	...	Month-to-month
1528	3500-RMZLT	1	United States	California	Fresno	93702	36.739385, -119.753649	36.739385	-119.753649	Female	...	Month-to-month
1534	3422-LYEPQ	1	United States	California	Carmel By The Sea	93921	36.554618, -121.922239	36.554618	-121.922239	Male	...	Month-to-month
1538	9507-EXLTT	1	United States	California	Seaside	93955	36.625114, -121.823565	36.625114	-121.823565	Female	...	Month-to-month

```
imputer = SimpleImputer(strategy='mean')
data['Tenure Months'] = imputer.fit_transform(data[['Tenure Months']])
data['Tenure Months'].isnull().sum()
```

```
0
```

```
data.groupby('Churn Label')['CustomerID'].nunique()
```

```
Churn Label
No     5174
Yes    1869
Name: CustomerID, dtype: int64
```

```
data[data['Total Charges'].isna()]
```

	CustomerID	Count	Country	State	City	Zip Code	Lat Long	Latitu
2234	4472-LVYGI	1	United	California	San	92408	34.084909,	34.084

```
data.dropna(subset=['Country', 'State', 'City'], inplace=True)
```

2438	3115-CZMZD	1	United	California	Independence	93526	36.869584,	36.869

```
data['calc_charges'] = data['Monthly Charges'] * data['Tenure Months']
```

2568	5709-LVOEO	1	United	California	San Mateo	94401	37.590421,	37.590

```
data.columns
```

```
Index(['CustomerID', 'Count', 'Country', 'State', 'City', 'Zip Code',
       'Lat Long', 'Latitude', 'Longitude', 'Gender', 'Senior Citizen',
       'Partner', 'Dependents', 'Tenure Months', 'Phone Service',
       'Multiple Lines', 'Internet Service', 'Online Security',
       'Online Backup', 'Device Protection', 'Tech Support', 'Streaming TV',
       'Streaming Movies', 'Contract', 'Paperless Billing', 'Payment Method',
       'Monthly Charges', 'Total Charges', 'Churn Label', 'Churn Value',
       'Churn Score', 'CLTV', 'Churn Reason', 'calc_charges'],
      dtype='object')
```

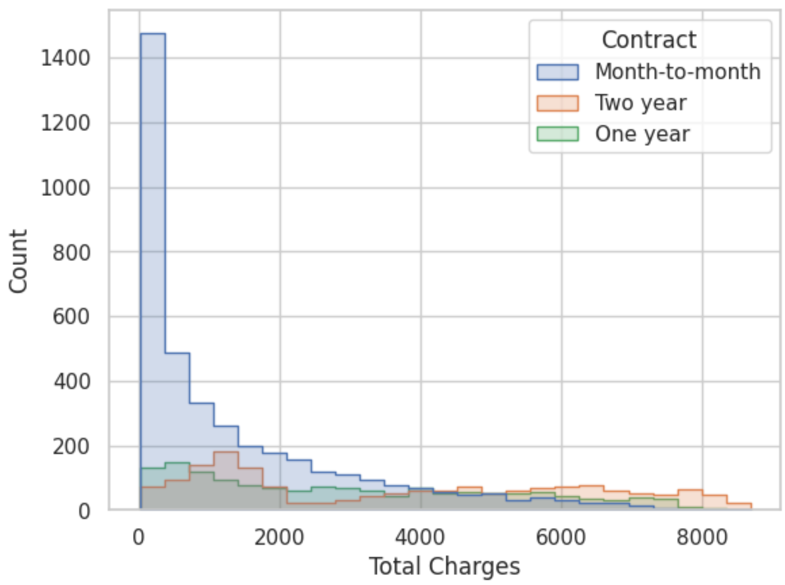
calculating difference between Total Charges and calculated charges

```
data['diff_in_charges'] = data['Total Charges'] - data['calc_charges']
data['diff_in_charges']
```

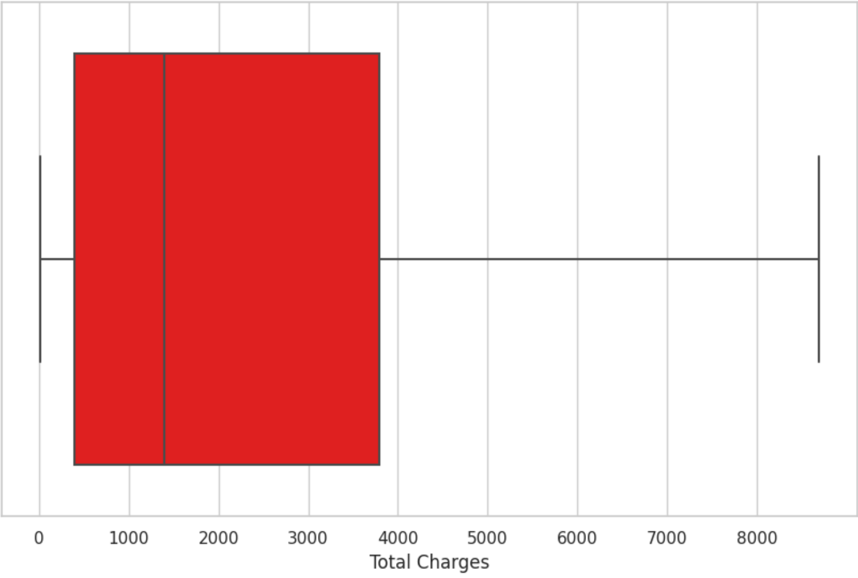
```
0         0.45
1        10.25
2        23.30
3       111.65
4       -45.00
         ...
7038   -103.40
7039    -44.70
7040    -67.50
7041     20.85
7042   -128.40
Name: diff_in_charges, Length: 7043, dtype: float64
```

```
# Charges distribution by contract
sns.set(style="whitegrid")
sns.histplot(data=data, x="Total Charges", hue="Contract", element="step", common_norm=False)
```

```
<Axes: xlabel='Total Charges', ylabel='Count'>
```




```
# How are total charges distributed ?
plt.figure(figsize=(10, 6))
ax = sns.boxplot(data=data, x="Total Charges", color="red")
ax.set_xticks(range(0, 9000, 1000))
plt.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.1)
```




```
# Accounting irregularities?
data.groupby('Contract')[['Total Charges','diff_in_charges']].quantile([.50,.80,.90,.95])
```

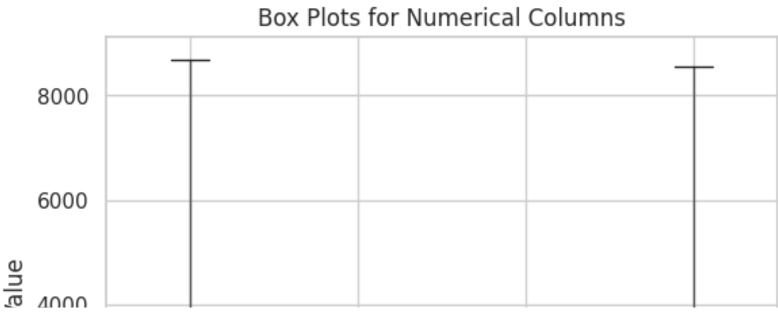
Contract		Total Charges	diff_in_charges
Month-to-month	**0.50**	679.5500	0.0000
	0.80	2485.7300	24.6800
	0.90	3844.0600	54.2000
	0.95	4966.9200	87.3650
One year	**0.50**	2657.5500	0.7750
	0.80	5286.4600	55.0500
	0.90	6341.2500	92.2000
	0.95	7072.4725	133.3375
Two year	**0.50**	3623.9500	0.5000
	0.80	6399.2400	61.5300
	0.90	7457.6100	97.5700
	0.95	7922.3400	139.1800

```
data['Total Charges'] = np.where(data['Total Charges'].isna() == True,data['calc_charges'], data['Total Charges'])
```

```
data.select_dtypes(include=['number']).columns
```

```
Index(['Count', 'Zip Code', 'Latitude', 'Longitude', 'Tenure Months',
       'Monthly Charges', 'Total Charges', 'Churn Value', 'Churn Score',
       'CLTV', 'calc_charges', 'diff_in_charges'],
      dtype='object')
```

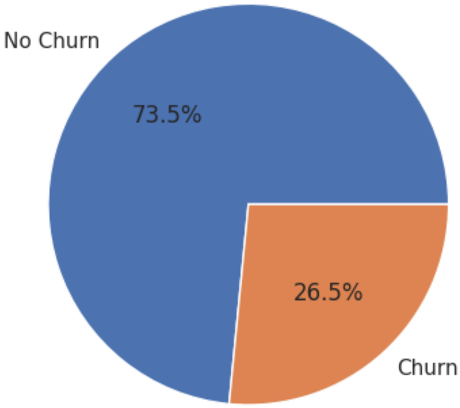
```
data[['Total Charges', 'Churn Score',
      'calc_charges']].boxplot()
plt.title('Box Plots for Numerical Columns')
plt.ylabel('Value')
plt.show()
```




```
data = data.drop(['calc_charges','diff_in_charges'], axis=1)
```

```
# Glance of churn non churn
plt.pie(data['Churn Label'].value_counts(), labels=['No Churn','Churn'], autopct='%1.1f%%')
```

```
([<matplotlib.patches.Wedge at 0x7b729f523e50>,
  <matplotlib.patches.Wedge at 0x7b729f523d60>],
 [Text(-0.7393678277834757, 0.8144539368428056, 'No Churn'),
  Text(0.7393677515287918, -0.8144540060674139, 'Churn')],
 [Text(-0.4032915424273503, 0.44424760191425755, '73.5%'),
  Text(0.4032915008338864, -0.4442476396731348, '26.5%')])
```




```
data.groupby(['Country','State'])['CustomerID'].count()
```

```
Country        State
United States  California    7043
Name: CustomerID, dtype: int64
```

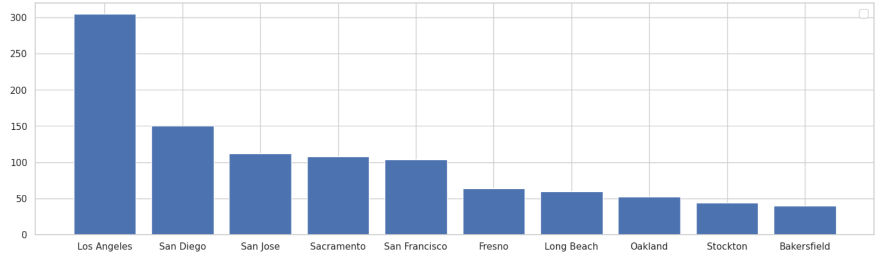
```
data['City'].nunique()
```

```
1129
```

```
# Geolocation scatter plot
fig = px.scatter_mapbox(data.groupby(['Latitude','Longitude'])['CustomerID'].count().reset_index(), lat="Latitude", lon="Lo
fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

```
# How many customers in each city ?
f, ax = plt.subplots(figsize=(18,5))
plt.bar(
  data.groupby('City')['CustomerID'].count().sort_values(ascending=False).head(
    10).index,
  data.groupby('City')['CustomerID'].count().sort_values(ascending=False).head(10).values,
)
ax.legend(fontsize = 14)
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Not
<matplotlib.legend.Legend at 0x7b729c7ddf00>
```




```
# Geolocation & number of clients
hex_level = 5

data['hex_id'] = data.apply(lambda x: h3.geo_to_h3(x['Latitude'], x['Longitude'], hex_level), axis=1)

hex_counts = data.groupby('hex_id')['CustomerID'].count().reset_index(name='total_clients')
hex_counts['center'] = hex_counts['hex_id'].apply(lambda x: h3.h3_to_geo(x))


color_range = [hex_counts['total_clients'].min(), hex_counts['total_clients'].max()]
colormap = cm.LinearColormap(["purple","red","orange","yellow","green"],vmin = min(color_range), vmax = max(color_range))

mean_lat, mean_lon = hex_counts['center'].apply(lambda x: x[0]).mean(), hex_counts['center'].apply(lambda x: x[1]).mean()
map_center = [mean_lat, mean_lon]
m = folium.Map(location=map_center, zoom_start=6, tiles='Stamen Terrain')

for _, row in hex_counts.iterrows():
    folium.Polygon(
        locations=h3.h3_to_geo_boundary(row['hex_id']),
        fill=True,
        fill_color=colormap(row['total_clients']),
        fill_opacity=0.7,
        stroke=False,
        tooltip=f"Number of clients: {row['total_clients']}"
    ).add_to(m)

colormap.caption = 'Number of clients'
m.add_child(colormap)

m
```


```
churn = data.assign(churn_clients = np.where(data['Churn Label']=='Yes',data['CustomerID'],None)).groupby(['hex_id']).agg({
```

```
data.groupby(['hex_id'])['CustomerID'].count()
```

```
hex_id
85280043fffffff     4
8528004bfffffff     4
8528004ffffffff     4
85280207fffffff     4
8528020bfffffff     4
                   ..
85485b03fffffff     5
85485b33fffffff     5
85485b63fffffff     5
85485babfffffff     5
85485bb7fffffff    10
Name: CustomerID, Length: 714, dtype: int64
```

```
clients = data.groupby(['hex_id'])['CustomerID'].count().reset_index()
clients.sample(5)
```

	hex_id	CustomerID
481	8529a227fffffff	4
162	8528305bfffffff	8
629	8529ab8bfffffff	4
573	8529a8b3fffffff	4
146	85283007fffffff	8

```
# Hex id for plotting purpose
churn_data = clients.join(churn.set_index(['hex_id']), on=['hex_id'])
churn_data
```

	hex_id	CustomerID	churn_clients
0	85280043fffffff	4	1
1	8528004bfffffff	4	1
2	8528004ffffffff	4	1
3	85280207fffffff	4	1
4	8528020bfffffff	4	1
...	...	...	...
709	85485b03fffffff	5	1
710	85485b33fffffff	5	1
711	85485b63fffffff	5	0
712	85485babfffffff	5	3
713	85485bb7fffffff	10	3

714 rows × 3 columns

```
churn_data['churn_rate'] = churn_data['churn_clients']/churn_data['CustomerID']
churn_data['churn_rate']
```

```
0       0.25
1       0.25
```

```
2      0.25
3      0.25
4      0.25
       ...
709    0.20
710    0.20
711    0.00
712    0.60
713    0.30
Name: churn_rate, Length: 714, dtype: float64
```

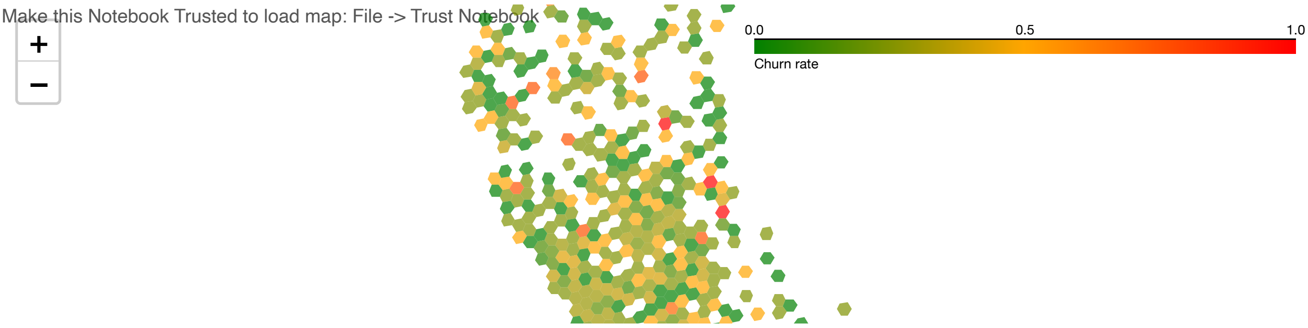
```
churn_data
```

	hex_id	CustomerID	churn_clients	churn_rate
0	85280043fffffff	4	1	0.25
1	8528004bfffffff	4	1	0.25
2	8528004ffffffff	4	1	0.25
3	85280207fffffff	4	1	0.25
4	8528020bfffffff	4	1	0.25
...	...	...	...	...
709	85485b03fffffff	5	1	0.20
710	85485b33fffffff	5	1	0.20
711	85485b63fffffff	5	0	0.00
712	85485babfffffff	5	3	0.60
713	85485bb7fffffff	10	3	0.30

714 rows × 4 columns

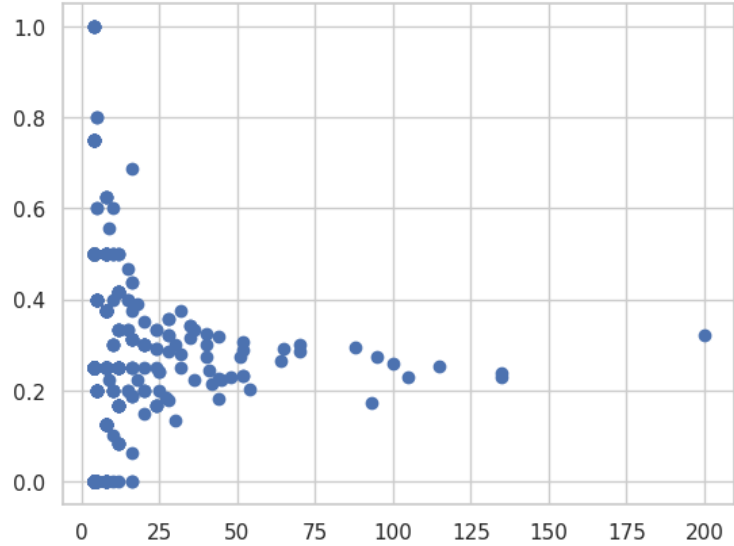
```
churn_data['center'] = churn_data['hex_id'].apply(lambda x: h3.h3_to_geo(x))

color_range = [churn_data['churn_rate'].min(), churn_data['churn_rate'].max()]
colormap = cm.LinearColormap(["green","orange","red"],vmin = min(color_range), vmax = max(color_range))

mean_lat, mean_lon = churn_data['center'].apply(lambda x: x[0]).mean(), churn_data['center'].apply(lambda x: x[1]).mean()
map_center = [mean_lat, mean_lon]
m = folium.Map(location=map_center, zoom_start=6,  width='100%', height='80%',tiles='Stamen Terrain')

for _, row in churn_data.iterrows():
    folium.Polygon(
        locations=h3.h3_to_geo_boundary(row['hex_id']),
        fill=True,
        fill_color=colormap(row['churn_rate']),
        fill_opacity=0.7,
        stroke=False,
        tooltip=f"Churn rate: {row['churn_rate']}<br>Number of customers: {row['CustomerID']}"
    ).add_to(m)

colormap.caption = 'Churn rate'
m.add_child(colormap)

m
```

Make this Notebook Trusted to load map: File -> Trust Notebook



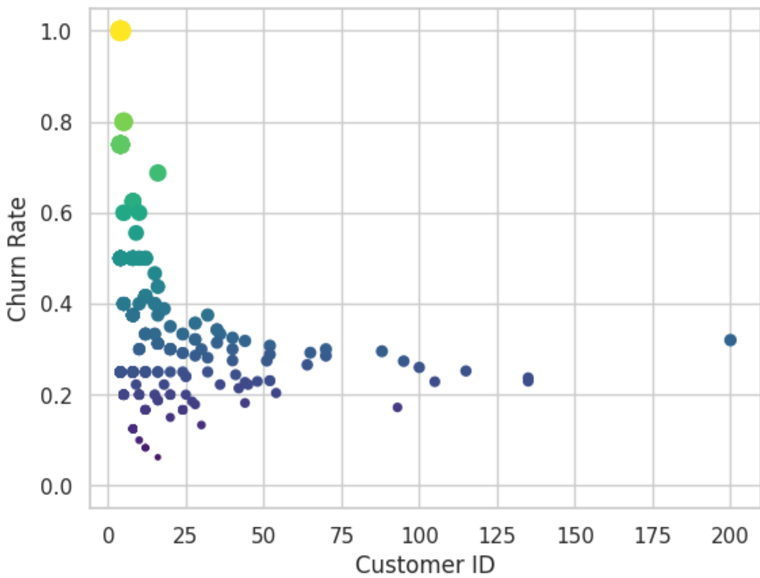

```
plt.scatter(
  churn_data['CustomerID'],
  churn_data['churn_rate'],
)
```

```
<matplotlib.collections.PathCollection at 0x7b729a390fa0>
```



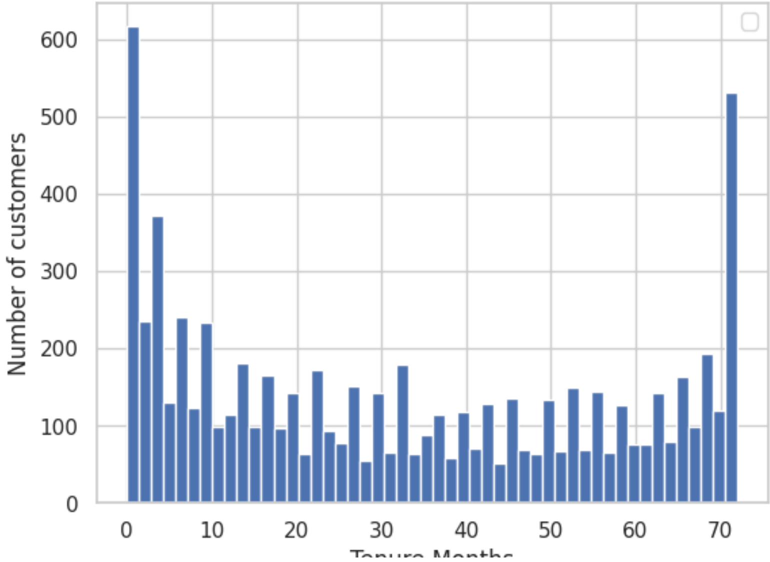

```
fig, ax = plt.subplots()
ax.scatter(churn_data['CustomerID'], churn_data['churn_rate'], s=churn_data['churn_rate']*100, c=churn_data['churn_rate'],
ax.set_xlabel('Customer ID')
ax.set_ylabel('Churn Rate')
```

```
Text(0, 0.5, 'Churn Rate')
```




```
plt.hist(data['Tenure Months'], bins=50)
plt.legend()
plt.xlabel('Tenure Months')
plt.ylabel('Number of customers')
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an
Text(0, 0.5, 'Number of customers')
```




```
data.groupby('Churn Label')['Tenure Months'].quantile([.50,.75,.90,.95])
```

```
Churn Label
No          0.50    38.0
            0.75    61.0
            0.90    71.0
            0.95    72.0
Yes         0.50    10.0
            0.75    31.0
            0.90    50.0
            0.95    60.0
Name: Tenure Months, dtype: float64
```

```
data.groupby('Churn Label')['Tenure Months'].mean()
```

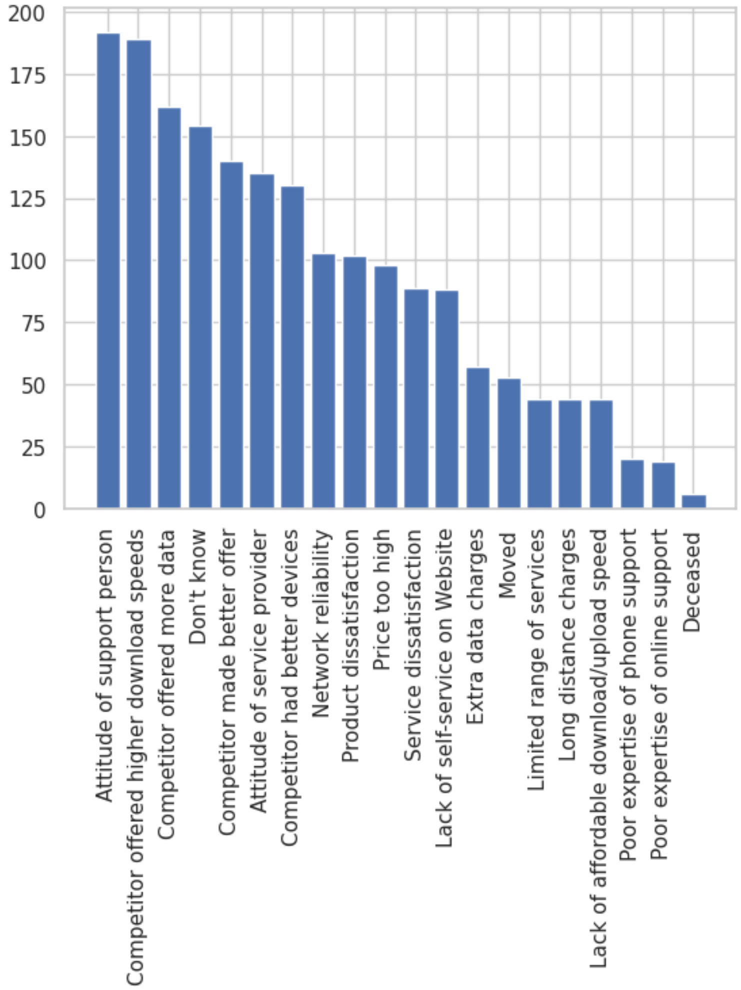
```
Churn Label
No     37.578893
Yes    18.274855
Name: Tenure Months, dtype: float64
```

```
grouped = data.groupby(['Churn Reason'])['CustomerID'].count().reset_index().sort_values('CustomerID',
                                                                                          ascending=False)
grouped
```

	Churn Reason	CustomerID
1	Attitude of support person	192

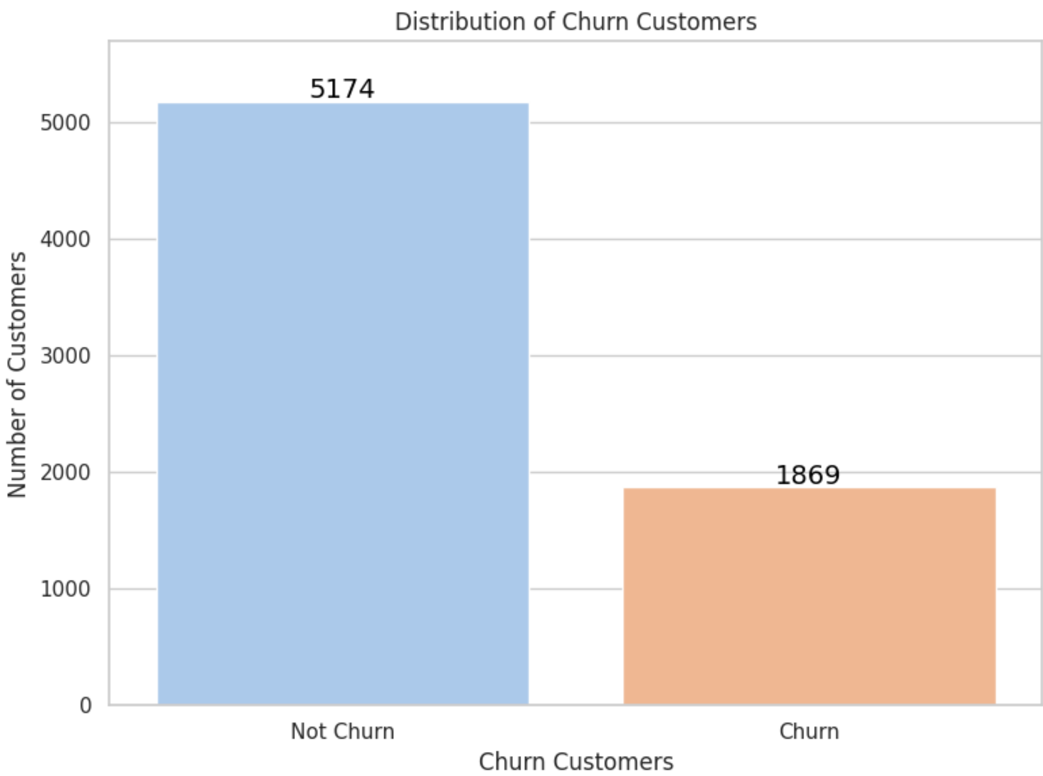
```
# What is frequency of each reason for churning out ?
plt.bar(grouped['Churn Reason'], grouped['CustomerID'])
plt.xticks(rotation=90)
```

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
 [Text(0, 0, 'Attitude of support person'),
  Text(1, 0, 'Competitor offered higher download speeds'),
  Text(2, 0, 'Competitor offered more data'),
  Text(3, 0, "Don't know"),
  Text(4, 0, 'Competitor made better offer'),
  Text(5, 0, 'Attitude of service provider'),
  Text(6, 0, 'Competitor had better devices'),
  Text(7, 0, 'Network reliability'),
  Text(8, 0, 'Product dissatisfaction'),
  Text(9, 0, 'Price too high'),
  Text(10, 0, 'Service dissatisfaction'),
  Text(11, 0, 'Lack of self-service on Website'),
  Text(12, 0, 'Extra data charges'),
  Text(13, 0, 'Moved'),
  Text(14, 0, 'Limited range of services'),
  Text(15, 0, 'Long distance charges'),
  Text(16, 0, 'Lack of affordable download/upload speed'),
  Text(17, 0, 'Poor expertise of phone support'),
  Text(18, 0, 'Poor expertise of online support'),
  Text(19, 0, 'Deceased')])
```

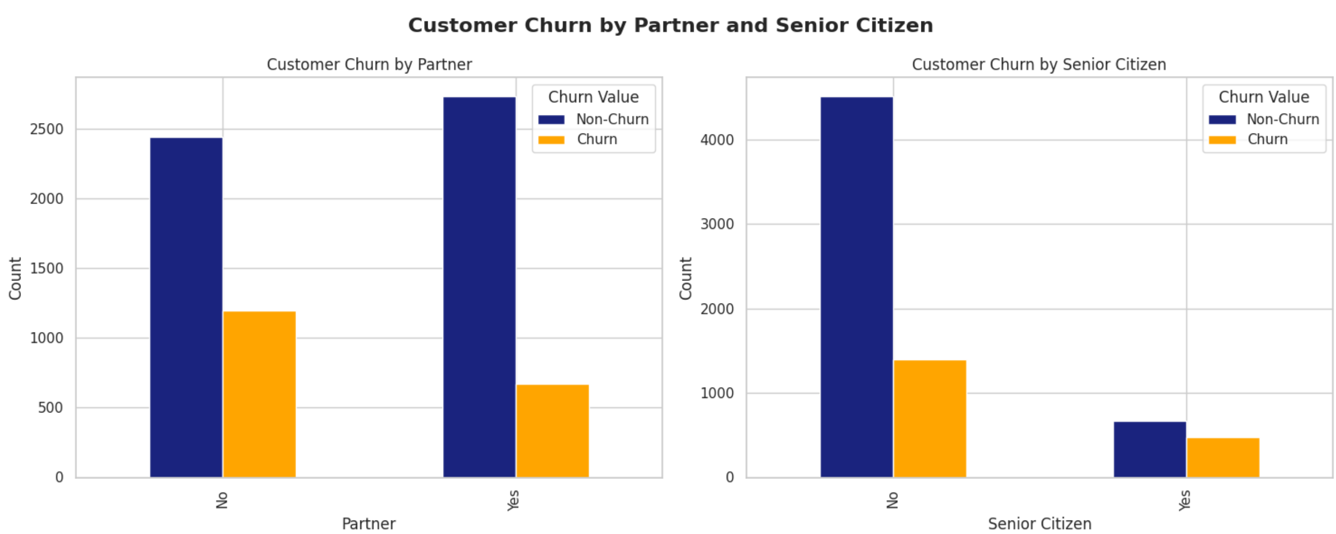



```
# number of customers who churned and not churned
exit_counts = data['Churn Value'].value_counts()
exit_percentages = exit_counts


sns.set_style('whitegrid')
plt.figure(figsize=(8,6))
ax = sns.barplot(x=exit_counts.index, y=exit_counts.values, palette='pastel')
ax.set(xlabel='Churn Customers', ylabel='Number of Customers', title='Distribution of Churn Customers')
plt.xticks([0, 1], ['Not Churn', 'Churn'])
plt.ylim(top=max(exit_counts.values)*1.1)

# add counting number on top of each bar
```

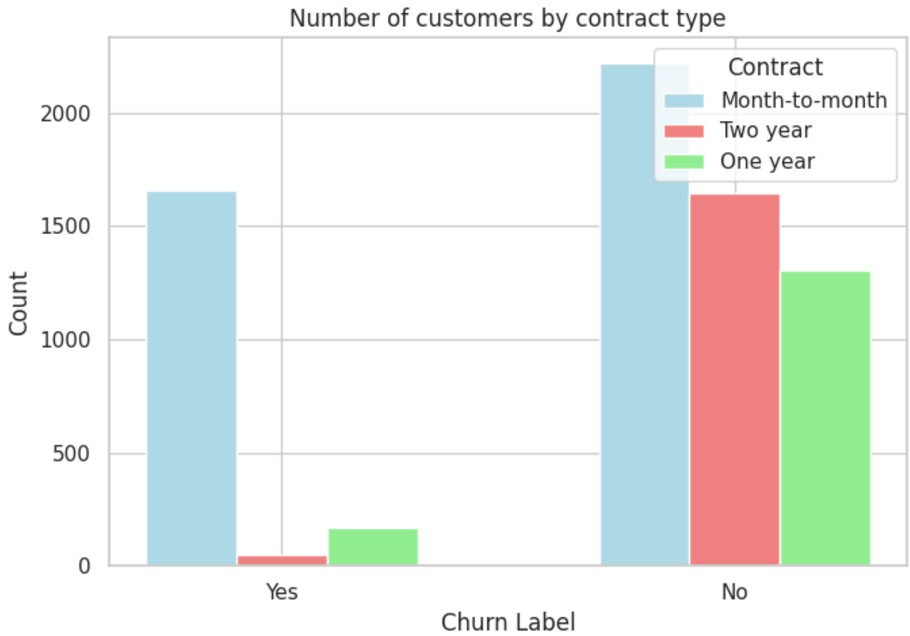
```
for i, v in enumerate(exit_percentages):
    ax.text(i, exit_counts.values[i]+30, f'{v}', fontsize=14, color='black', ha='center')

plt.tight_layout()
plt.show()
```



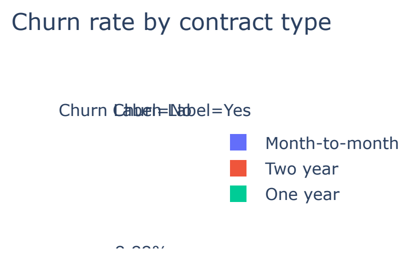

```
# Does having partner or being senior citizen affect churns ?
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15,6))

colors = ['#1a237e', '#FFA500']
partner_churn = data.groupby(['Partner', 'Churn Value'])['Churn Value'].count().unstack().plot(ax=ax1, kind='bar', color=co
ax1.set_xlabel('Partner')
ax1.set_ylabel('Count')
ax1.set_title('Customer Churn by Partner')
ax1.legend(['Non-Churn', 'Churn'], title='Churn Value', loc='upper right')

senior_churn = data.groupby(['Senior Citizen', 'Churn Value'])['Churn Value'].count().unstack().plot(ax=ax2, kind='bar', co
ax2.set_xlabel('Senior Citizen')
ax2.set_ylabel('Count')
ax2.set_title('Customer Churn by Senior Citizen')
ax2.legend(['Non-Churn', 'Churn'], title='Churn Value', loc='upper right')

plt.suptitle('Customer Churn by Partner and Senior Citizen', fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()
```



```
# Contract wise churned or not ?
churn_labels = data['Churn Label'].unique()
contracts = data['Contract'].unique()

fig, ax = plt.subplots(figsize=(7, 5))

contract_colors = {'Month-to-month': 'lightblue', 'One year': 'lightgreen', 'Two year': 'lightcoral'}

width = 0.2
x = np.arange(len(churn_labels))

for i, contract in enumerate(contracts):
    contract_data = data[data['Contract'] == contract]
    counts = [contract_data[contract_data['Churn Label'] == label]['Churn Label'].count() for label in churn_labels]
    ax.bar(x + i * width, counts, width=width, label=contract, color=contract_colors[contract])

ax.set_xlabel('Churn Label')
ax.set_ylabel('Count')
ax.set_title('Number of customers by contract type')
ax.set_xticks(x + width)
ax.set_xticklabels(churn_labels)
ax.legend(title='Contract', loc='upper right')

plt.tight_layout()
plt.show()
```




```
#
grouped = data.groupby(['Contract', 'Churn Label'])['CustomerID'].count().reset_index()


fig = px.pie(data.groupby(['Contract','Churn Label'])['CustomerID'].count().reset_index(),
             values='CustomerID',
            names='Contract',
            facet_col = 'Churn Label',
            title = 'Churn rate by contract type')

fig.show()
```



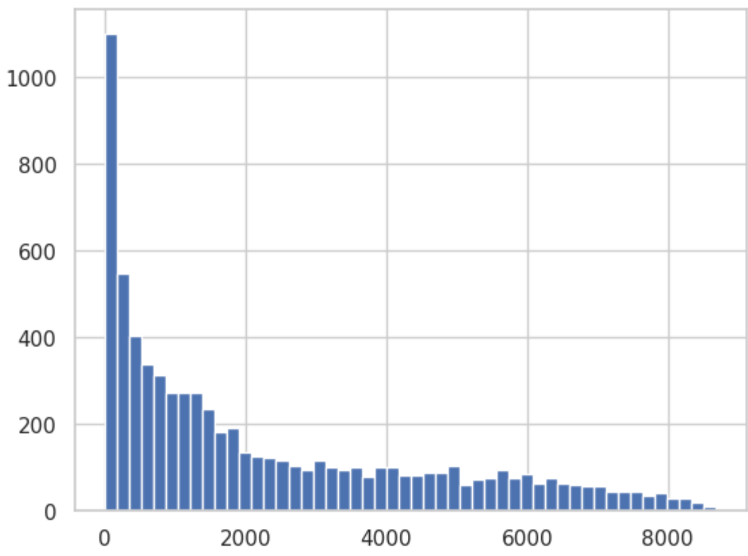

```
# Does contract type affect Tenure months & churn ?
data.groupby(['Contract','Churn Label'])['Tenure Months'].mean()
```

```
Contract        Churn Label
Month-to-month  No             21.054141
                Yes            14.350878
One year        No             41.674063
                Yes            44.963855
Two year        No             56.602914
                Yes            61.270833
Name: Tenure Months, dtype: float64
```

Total charges

```
plt.hist(data['Total Charges'], bins=50)
```

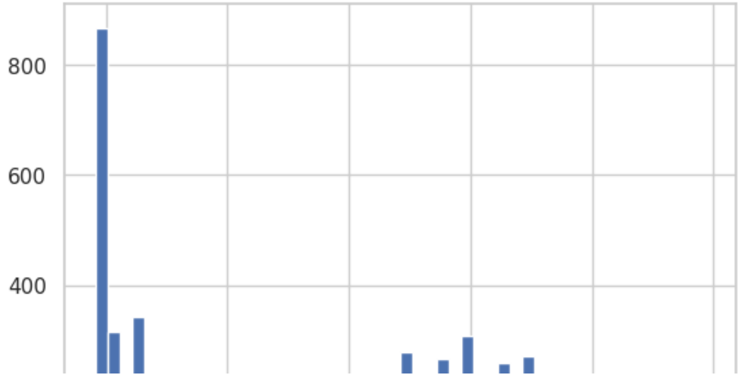
```
(array([1101.,  547.,  404.,  337.,  312.,  270.,  272.,  270.,  235.,
         180.,  190.,  135.,  123.,  121.,  116.,  104.,   93.,  115.,
          99.,   92.,  100.,   79.,   99.,  100.,   82.,   80.,   86.,
          86.,  102.,   60.,   71.,   75.,   93.,   74.,   83.,   62.,
          73.,   62.,   58.,   56.,   57.,   44.,   44.,   44.,   35.,
          41.,   29.,   26.,   18.,    8.]),
 array([   0.   ,  173.696,  347.392,  521.088,  694.784,  868.48 ,
        1042.176, 1215.872, 1389.568, 1563.264, 1736.96 , 1910.656,
        2084.352, 2258.048, 2431.744, 2605.44 , 2779.136, 2952.832,
        3126.528, 3300.224, 3473.92 , 3647.616, 3821.312, 3995.008,
        4168.704, 4342.4  , 4516.096, 4689.792, 4863.488, 5037.184,
        5210.88 , 5384.576, 5558.272, 5731.968, 5905.664, 6079.36 ,
        6253.056, 6426.752, 6600.448, 6774.144, 6947.84 , 7121.536,
        7295.232, 7468.928, 7642.624, 7816.32 , 7990.016, 8163.712,
        8337.408, 8511.104, 8684.8  ]),
 <BarContainer object of 50 artists>)
```



Monthly Charges

```
plt.hist(data['Monthly Charges'], bins=50)
```

```
(array([868., 316.,  66., 343.,  13.,  59.,  25.,  25.,  74.,   8.,  62.,
         32.,  57., 194.,  19., 190., 107.,  96., 204.,  43., 148.,  85.,
         67., 125.,  48., 278., 116., 166., 265.,  66., 308., 156., 150.,
        259.,  82., 270., 130., 165., 213.,  93., 236., 114., 149., 177.,
         82., 115.,  58.,  50.,  54.,  17.]),
 array([ 18.25,  20.26,  22.27,  24.28,  26.29,  28.3 ,  30.31,  32.32,
         34.33,  36.34,  38.35,  40.36,  42.37,  44.38,  46.39,  48.4 ,
         50.41,  52.42,  54.43,  56.44,  58.45,  60.46,  62.47,  64.48,
         66.49,  68.5 ,  70.51,  72.52,  74.53,  76.54,  78.55,  80.56,
         82.57,  84.58,  86.59,  88.6 ,  90.61,  92.62,  94.63,  96.64,
         98.65, 100.66, 102.67, 104.68, 106.69, 108.7 , 110.71, 112.72,
        114.73, 116.74, 118.75]),
 <BarContainer object of 50 artists>)
```




```
data.groupby('Churn Label')['Monthly Charges'].quantile([.50,.75,.95,.99])
```

```
Churn Label
No           0.50     64.4250
             0.75     88.4000
             0.95    108.4175
             0.99    115.1000
Yes          0.50     79.6500
             0.75     94.2000
             0.95    105.6100
             0.99    111.1320
Name: Monthly Charges, dtype: float64
```

```
corr_df = data.copy()
```

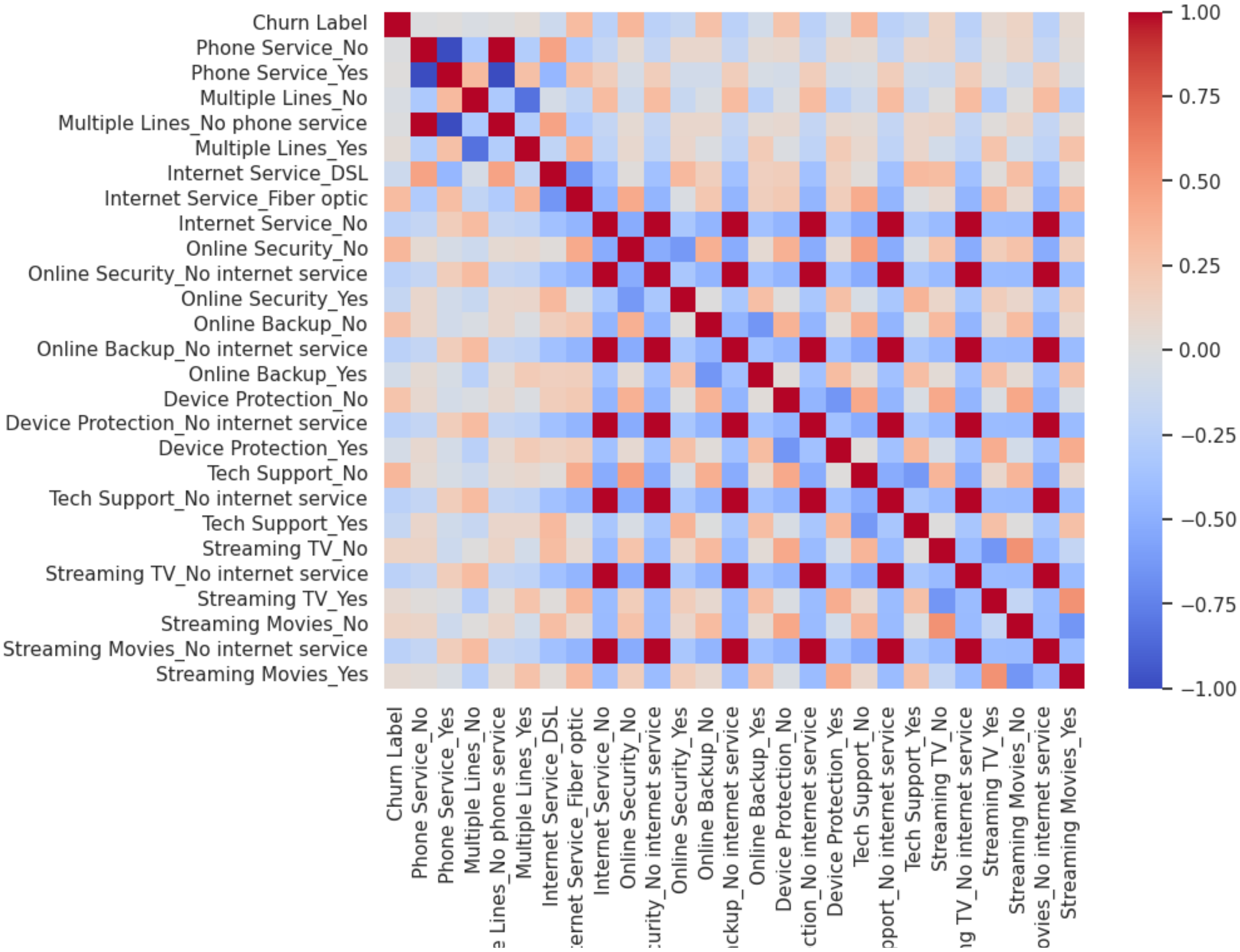
```
corr_df['Churn Label'].replace(to_replace='Yes', value=1, inplace=True)
corr_df['Churn Label'].replace(to_replace='No',  value=0, inplace=True)
```

```
df_dummies = pd.get_dummies(corr_df[['Churn Label','Phone Service','Multiple Lines','Internet Service','Online Security',
                                     'Online Backup','Device Protection','Tech Support','Streaming TV',
                                     'Streaming Movies']])
df_dummies.head()
```

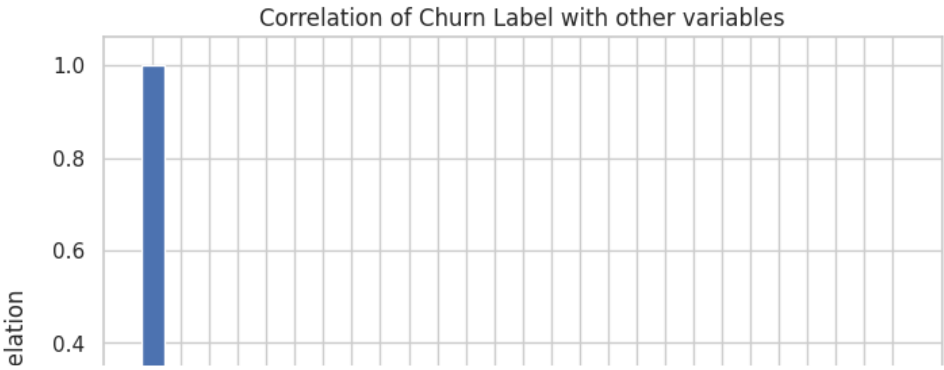
	Churn Label	Phone Service_No	Phone Service_Yes	Multiple Lines_No	Multiple Lines_No phone service	Multiple Lines_Yes	Internet Service_DSL	Internet Service_Fiber optic	Internet Service_No	Online Security_No	..
0	1	0	1	1	0	0	1	0	0	0	
1	1	0	1	1	0	0	0	1	0	1	
2	1	0	1	0	0	1	0	1	0	1	
3	1	0	1	0	0	1	0	1	0	1	
4	1	0	1	0	0	1	0	1	0	1	

5 rows × 27 columns

```
# Plot correlation of features
plt.figure(figsize=(9, 7))
sns.heatmap(df_dummies.corr(), annot=False, cmap='coolwarm')

plt.show()
```

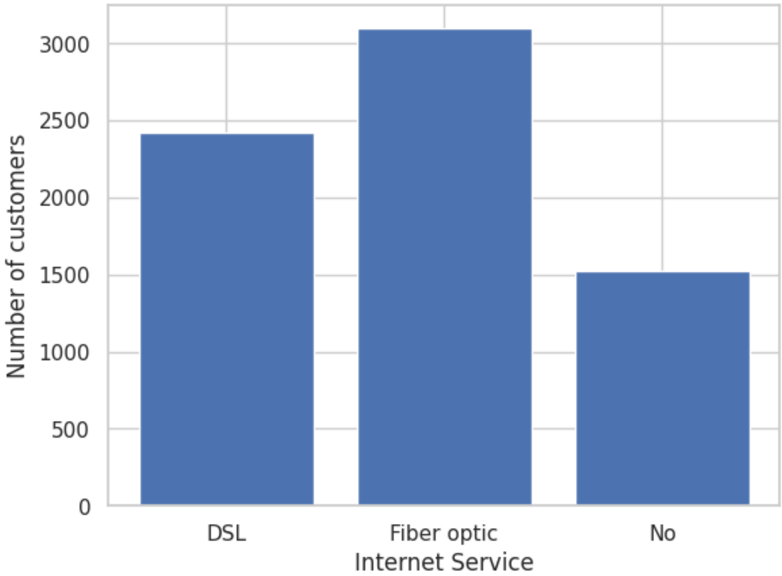



```
# Correlation of Churn Label with other variables

fig, ax = plt.subplots(figsize=(8, 6))
ax.bar(df_dummies.corr()['Churn Label'].sort_values(ascending=False).index,
       df_dummies.corr()['Churn Label'].sort_values(ascending=False).values)
ax.set_title('Correlation of Churn Label with other variables')
ax.set_xlabel('Variables')
ax.set_ylabel('Correlation')
plt.xticks(rotation=90)
plt.show()
```



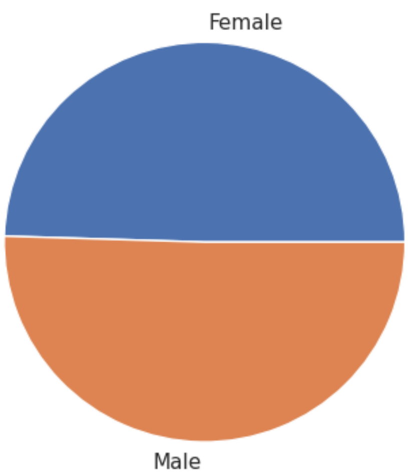

```
# Customers subscribed to each internet service
internet_services = data.groupby('Internet Service')['CustomerID'].count().reset_index()
plt.bar(internet_services['Internet Service'], internet_services['CustomerID'])
plt.xlabel('Internet Service')
plt.ylabel('Number of customers')
```

```
Text(0, 0.5, 'Number of customers')
```

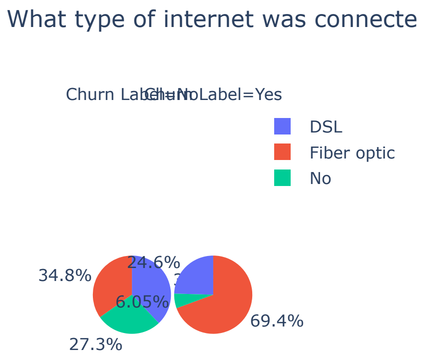



```
data.groupby('Gender')['CustomerID'].count().plot(kind='pie')
plt.ylabel('')
```

```
Text(0, 0.5, '')
```

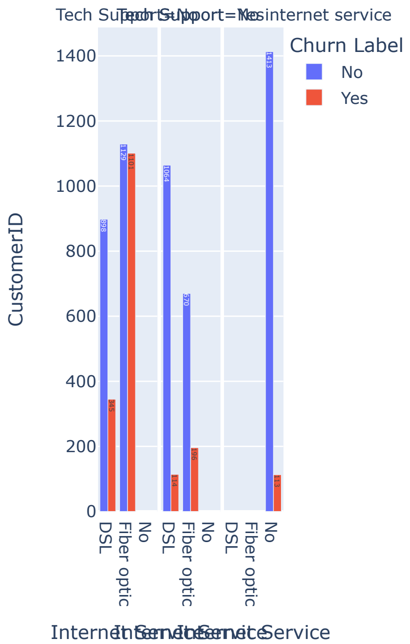



```
#
fig = px.pie(data.groupby(['Internet Service','Churn Label'])['CustomerID'].count().reset_index(),
             values='CustomerID',
             facet_col = 'Churn Label',
             names='Internet Service',
            title = 'What type of internet was connected to the clients who left the service?')
fig.show()
```



Tech Support

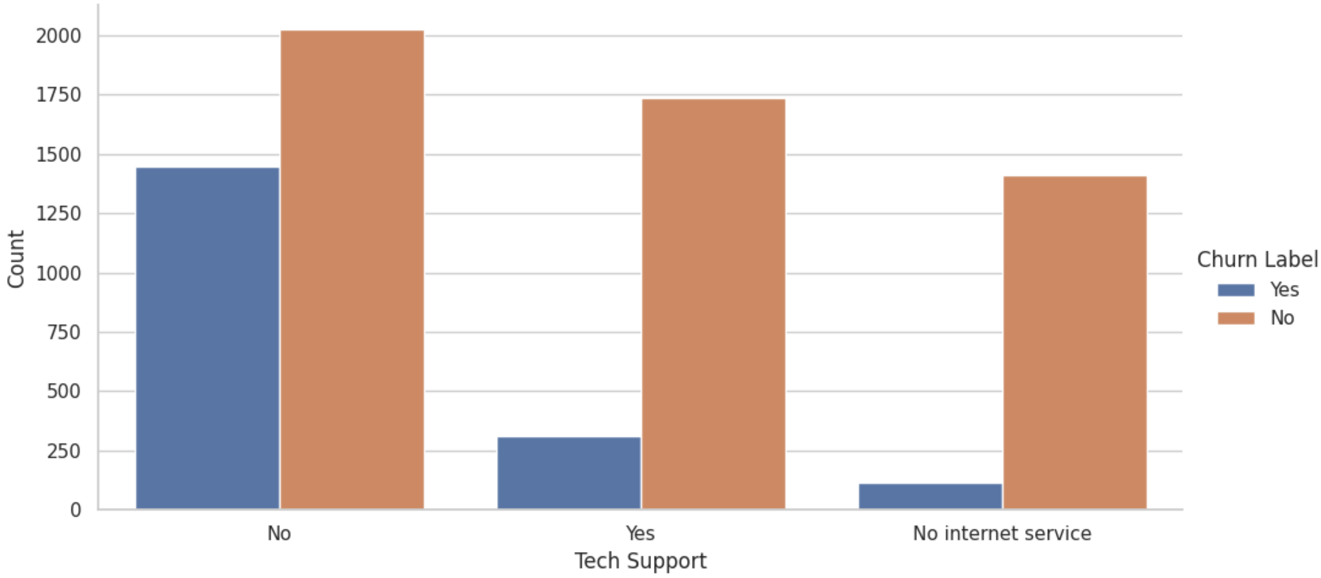
```
# Tech support opted ?
fig = px.bar(data.groupby(['Internet Service', 'Tech Support', 'Churn Label'])['CustomerID'].count().reset_index(),
             x="Internet Service",
             y="CustomerID",
             color="Churn Label",
             text = 'CustomerID',
             barmode="group",
             facet_col="Tech Support"
            )
fig.show()
```



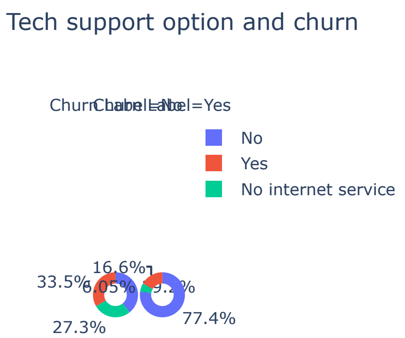

```
# Assuming 'data' is a DataFrame with the required data
plt.figure(figsize=(12, 6))  # Adjust the figure size as needed

# Create a catplot with two pie charts for each 'Churn Label'
g = sns.catplot(data=data, x='Tech Support', kind='count', hue='Churn Label', aspect=2)
g.set_axis_labels('Tech Support', 'Count')
g.set_titles('Tech support option and churn ({col_name})')
```

```
plt.show()
```

<Figure size 1200x600 with 0 Axes>



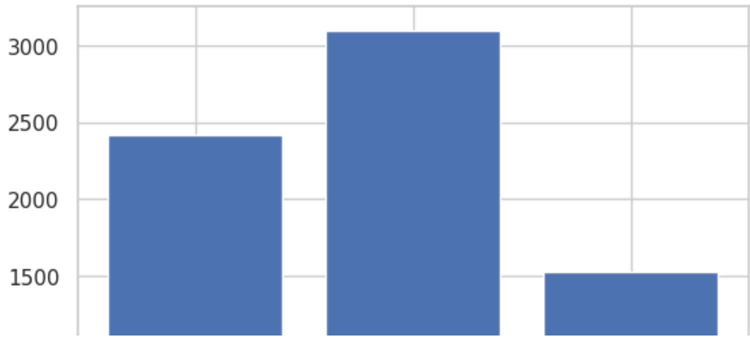

```
fig = px.pie(data.groupby(['Tech Support','Churn Label'])['CustomerID'].count().reset_index(),
             values='CustomerID',
             facet_col = 'Churn Label',
             hole = .5,
             names='Tech Support',
            title = 'Tech support option and churn')
fig.show()
```



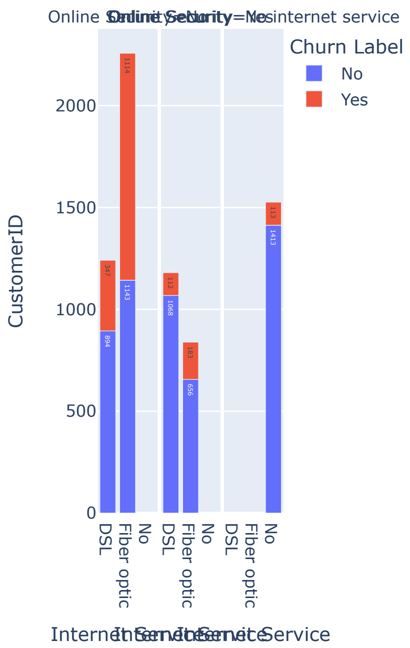

```
internet_services = data.groupby('Internet Service')['CustomerID'].count().reset_index()
```

```
plt.bar(internet_services['Internet Service'], internet_services['CustomerID'])
```

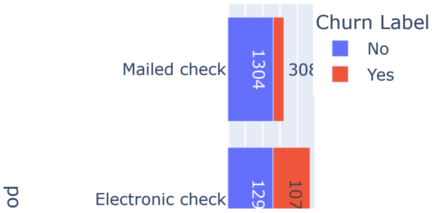
```
<BarContainer object of 3 artists>
```



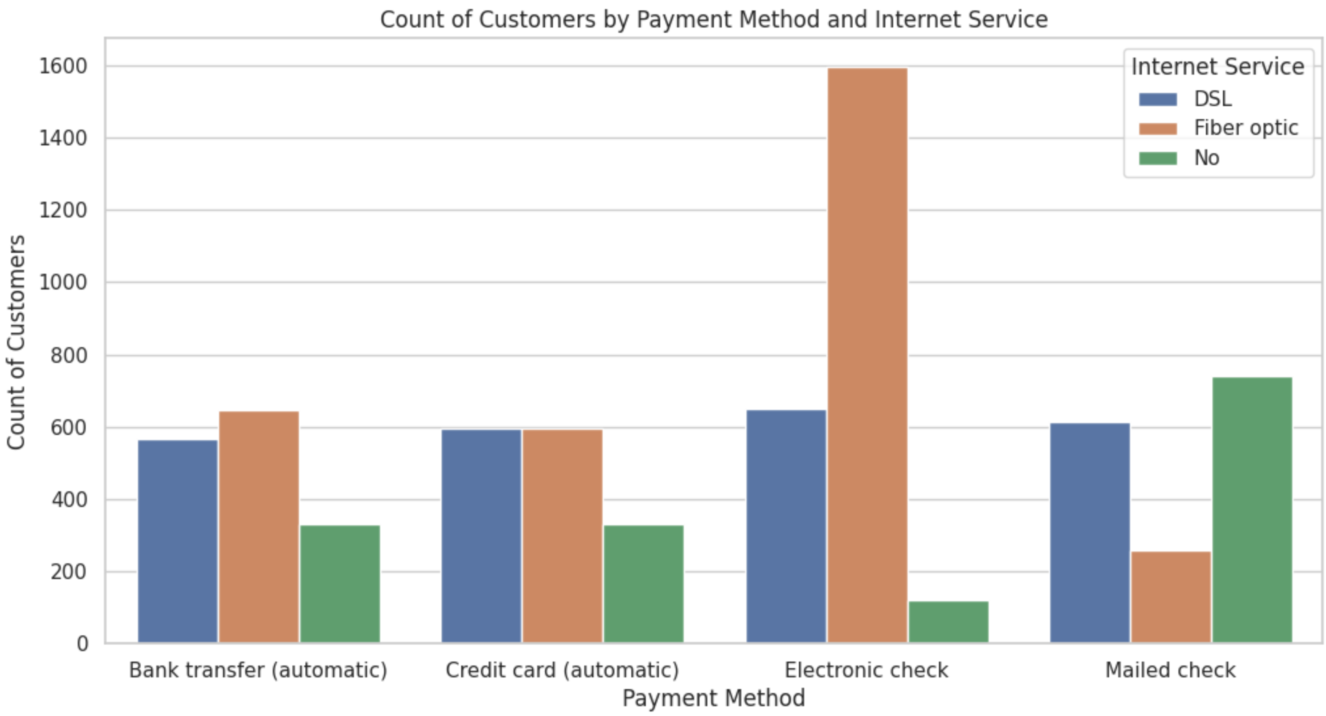

```
fig = px.bar(data.groupby(['Internet Service','Online Security',
                                                  'Churn Label'])['CustomerID'].count().reset_index(),
             x="Internet Service",
             y="CustomerID",
             color="Churn Label",
             #barmode="group",
             text = 'CustomerID',
             facet_col = 'Online Security'
            )
fig.show()
```




```
fig = px.bar(data.groupby(['Payment Method',
                                                  'Churn Label'])['CustomerID'].count().reset_index(),
             x="CustomerID",
             y="Payment Method",
             color="Churn Label",
             text = 'CustomerID'
            )
fig.show()
```



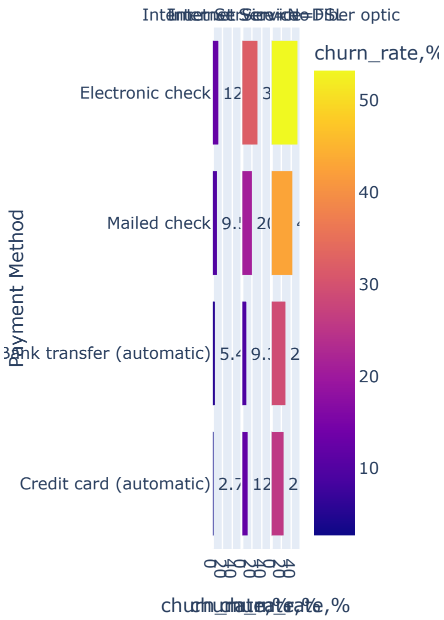

```
grouped = data.groupby(['Payment Method', 'Internet Service'])['CustomerID'].count().reset_index()

plt.figure(figsize=(12, 6))
sns.barplot(data=grouped, x='Payment Method', y='CustomerID', hue='Internet Service')

plt.title("Count of Customers by Payment Method and Internet Service")
plt.xlabel("Payment Method")
plt.ylabel("Count of Customers")

plt.show()
```




```
churn_pm = data.assign(churn_clients = np.where(data['Churn Label']== 'Yes',data['CustomerID'],None))\
   .groupby(['Payment Method','Internet Service']).agg({'churn_clients':'count'}).reset_index()


pm_clients = data.groupby(['Payment Method','Internet Service'])['CustomerID'].count().reset_index()


pm_data = pm_clients.join(churn_pm.set_index(['Payment Method','Internet Service']), on=['Payment Method','Internet Service


pm_data
```

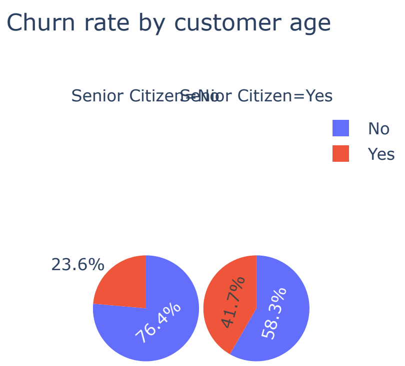
	Payment Method	Internet Service	CustomerID	churn_clients
0	Bank transfer (automatic)	DSL	566	53
1	Bank transfer (automatic)	Fiber optic	646	187

```
pm_data['churn_rate,%'] = round(((pm_data['churn_clients']/pm_data['CustomerID']) * 100),2)

# Churn rate % based on internet service & distributed based on payment method
fig = px.bar(pm_data.sort_values('churn_rate,%'),
             x='churn_rate,%',
             y='Payment Method',
             facet_col = 'Internet Service',
             color = 'churn_rate,%',
             text = 'churn_rate,%')
fig.show()
```




```
fig = px.bar(data.groupby(['Gender', 'Churn Label'])['CustomerID'].count().reset_index(),
             x="CustomerID",
             y="Gender",
             color="Churn Label",
             text = 'CustomerID'
            )
fig.show()
```

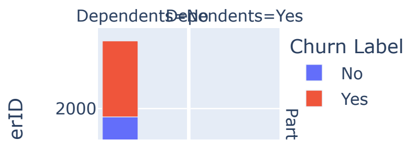
```
fig = px.pie(data.groupby(['Senior Citizen','Churn Label'])['CustomerID'].count().reset_index(),
            values='CustomerID',
            names='Churn Label',
            facet_col = 'Senior Citizen',
            color = 'Churn Label',
            title = 'Churn rate by customer age')

fig.show()
```



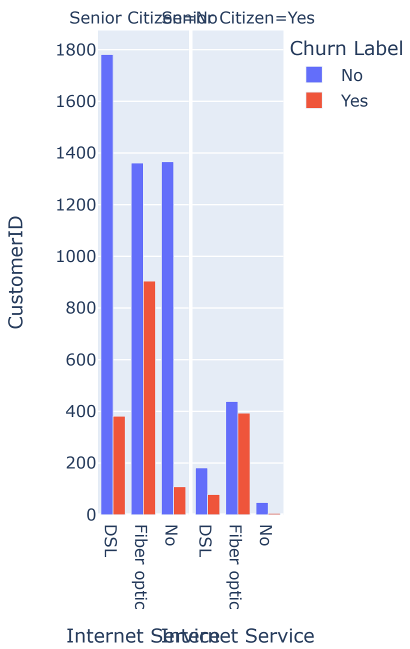

```
data.groupby('Senior Citizen')['CustomerID'].count()
```

```
    Senior Citizen
    No     5901
    Yes    1142
    Name: CustomerID, dtype: int64
```

```
fig = px.bar(data.groupby(['Senior Citizen','Partner',
                                        'Dependents','Churn Label'])['CustomerID'].count().reset_index(),
             x="Senior Citizen",
             y="CustomerID",
             color="Churn Label",
             #barmode="group",
             facet_row="Partner",
             facet_col = 'Dependents'
            )
fig.show()
```




```
fig = px.bar(data.groupby(['Senior Citizen','Internet Service','Churn Label'])['CustomerID'].count().reset_index(),
             x="Internet Service",
             y="CustomerID",
             color="Churn Label",
             barmode="group",
             facet_col = 'Senior Citizen'
            )
fig.show()
```




```
# data.to_excel('/content/drive/MyDrive/datasets/fds_customer_churn/Telco_customer_churn_working.xlsx')
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

Name - Omkar Phansopkar

UID - 2022701007

D3 - CSE-DS

Dataset - Telco customer churn: IBM dataset

https://www.kaggle.com/datasets/yeanzc/telco-customer-churn-ibm-dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install xgboost --quiet
```

```
import joblib
import numpy as np
import pandas as pd
import seaborn as sns
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
import imblearn
import os
import folium

from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, f1_score,recall_score, precision_score
from sklearn.metrics import average_precision_score, roc_auc_score, roc_curve, auc
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
WORKING_DIR = '/content/drive/MyDrive/datasets/fds_customer_churn'
```

```
data = pd.read_excel(f'{WORKING_DIR}/Telco_customer_churn_working.xlsx', index_col=False)
data.sample(5)
```

	Unnamed: 0	CustomerID	Count	Country	State	City	Zip Code	Lat Long
3511	3511	5242-UOWHD	1	United States	California	Anaheim	92802	33.807864, -117.923782
1782	1782	6230-BSUXY	1	United States	California	Paramount	90723	33.897122, -118.164432
2116	2116	9498-FIMXL	1	United States	California	Oceanside	92054	33.351059, -117.420557
5479	5479	4393-RYCRE	1	United States	California	Macdoel	96058	41.769709, -121.92063
2240	2240	0505-SPOOW	1	United States	California	Riverside	92508	33.885499, -117.324959

5 rows × 35 columns

```
data = data.drop(['Unnamed: 0','Country','State','Count','Zip Code','Churn Reason','City','Churn Score','Churn Value','CLT
                  'Latitude','Longitude'], axis = 1)
```

```
data.sample(2)
```

	Gender	Senior Citizen	Partner	Dependents	Tenure Months	Phone Service	Multiple Lines	Internet Service
1598	Female	No	No	No	1	Yes	No	Fiber optic
4342	Male	No	Yes	Yes	23	Yes	No	No

2 rows × 21 columns

```
data.columns
```

```
Index(['Gender', 'Senior Citizen', 'Partner', 'Dependents', 'Tenure Months',
       'Phone Service', 'Multiple Lines', 'Internet Service',
       'Online Security', 'Online Backup', 'Device Protection', 'Tech Support',
       'Streaming TV', 'Streaming Movies', 'Contract', 'Paperless Billing',
       'Payment Method', 'Monthly Charges', 'Total Charges', 'Churn Label',
       'hex_id'],
      dtype='object')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             7043 non-null   object 
 1   Senior Citizen     7043 non-null   object 
 2   Partner            7043 non-null   object 
 3   Dependents         7043 non-null   object 
 4   Tenure Months      7043 non-null   int64  
 5   Phone Service      7043 non-null   object 
 6   Multiple Lines     7043 non-null   object 
 7   Internet Service   7043 non-null   object 
 8   Online Security    7043 non-null   object 
 9   Online Backup      7043 non-null   object 
 10  Device Protection  7043 non-null   object 
 11  Tech Support       7043 non-null   object 
 12  Streaming TV       7043 non-null   object 
 13  Streaming Movies   7043 non-null   object 
 14  Contract           7043 non-null   object 
 15  Paperless Billing  7043 non-null   object 
 16  Payment Method     7043 non-null   object 
 17  Monthly Charges    7043 non-null   float64
 18  Total Charges      7043 non-null   float64
 19  Churn Label        7043 non-null   object 
 20  hex_id             7043 non-null   object 
dtypes: float64(2), int64(1), object(18)
memory usage: 1.1+ MB
```

```
data['Churn Label'].replace(to_replace='Yes', value=1, inplace=True)
data['Churn Label'].replace(to_replace='No',  value=0, inplace=True)
```

```
data['Churn Label'].value_counts()
```

```
0    5174
1    1869
Name: Churn Label, dtype: int64
```

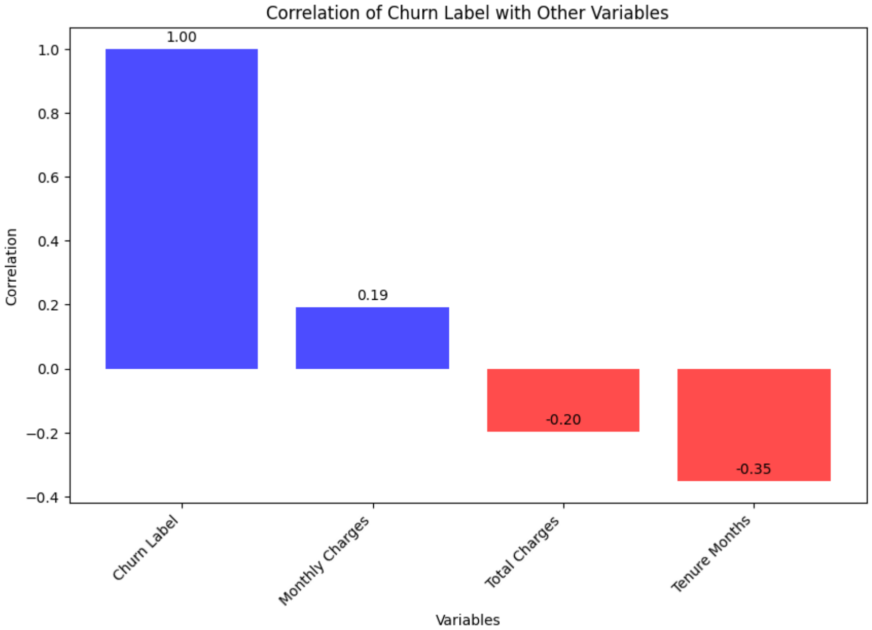
```
data.corr()
```

```
<ipython-input-43-c44ded798807>:1: FutureWarning: The default value of numeri
  data.corr()
```

	Tenure Months	Monthly Charges	Total Charges	Churn Label
Tenure Months	1.000000	0.247900	0.826178	-0.352229
Monthly Charges	0.247900	1.000000	0.651174	0.193356
Total Charges	0.826178	0.651174	1.000000	-0.198324
Churn Label	-0.352229	0.193356	-0.198324	1.000000

```
corr = data.corr()['Churn Label'].sort_values(ascending=False)

plt.figure(figsize=(10, 6))
bars = plt.bar(corr.index, corr, color=np.where(corr > 0, 'b', 'r'), alpha=0.7)

plt.title('Correlation of Churn Label with Other Variables')
plt.xlabel('Variables')
plt.ylabel('Correlation')
plt.xticks(rotation=45, ha='right')

for bar in bars:
    height = bar.get_height()
    plt.annotate(f'{height:.2f}', xy=(bar.get_x() + bar.get_width() / 2, height),
                 xytext=(0, 3), textcoords='offset points', ha='center', va='bottom')

plt.show()
```

```
<ipython-input-44-b1b0091519ea>:1: FutureWarning: The default value of numeri
  corr = data.corr()['Churn Label'].sort_values(ascending=False)
```




```
# Columns to apply nominal encoding
categorical_columns = []
for col in data.columns:
    if data[col].nunique() < 20:
      categorical_columns.append(col)
      print(col, "->", data[col].unique())

categorical_columns.remove("Churn Label")
categorical_columns
```

```
Gender -> ['Male' 'Female']
Senior Citizen -> ['No' 'Yes']
Partner -> ['No' 'Yes']
Dependents -> ['No' 'Yes']
Phone Service -> ['Yes' 'No']
Multiple Lines -> ['No' 'Yes' 'No phone service']
Internet Service -> ['DSL' 'Fiber optic' 'No']
Online Security -> ['Yes' 'No' 'No internet service']
Online Backup -> ['Yes' 'No' 'No internet service']
Device Protection -> ['No' 'Yes' 'No internet service']
Tech Support -> ['No' 'Yes' 'No internet service']
Streaming TV -> ['No' 'Yes' 'No internet service']
Streaming Movies -> ['No' 'Yes' 'No internet service']
Contract -> ['Month-to-month' 'Two year' 'One year']
Paperless Billing -> ['Yes' 'No']
Payment Method -> ['Mailed check' 'Electronic check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn Label -> [1 0]
```

```
['Gender',
 'Senior Citizen',
 'Partner',
 'Dependents',
 'Phone Service',
 'Multiple Lines',
 'Internet Service',
 'Online Security',
 'Online Backup',
 'Device Protection',
 'Tech Support',
 'Streaming TV',
 'Streaming Movies',
 'Contract',
 'Paperless Billing',
 'Payment Method']
```

```
data.columns
```

```
Index(['Gender', 'Senior Citizen', 'Partner', 'Dependents', 'Tenure Months',
       'Phone Service', 'Multiple Lines', 'Internet Service',
       'Online Security', 'Online Backup', 'Device Protection', 'Tech Support',
       'Streaming TV', 'Streaming Movies', 'Contract', 'Paperless Billing',
       'Payment Method', 'Monthly Charges', 'Total Charges', 'Churn Label',
       'hex_id'],
      dtype='object')
```

```
data.head(2)
```

	Gender	Senior Citizen	Partner	Dependents	Tenure Months	Phone Service	Multiple Lines	Internet Service	S
0	Male	No	No	No	2	Yes	No	DSL	
1	Female	No	No	Yes	2	Yes	No	Fiber optic	

2 rows × 21 columns

```
data.columns
```


```
Index(['Gender', 'Senior Citizen', 'Partner', 'Dependents', 'Tenure Months',
       'Phone Service', 'Multiple Lines', 'Internet Service',
       'Online Security', 'Online Backup', 'Device Protection', 'Tech Support',
       'Streaming TV', 'Streaming Movies', 'Contract', 'Paperless Billing',
       'Payment Method', 'Monthly Charges', 'Total Charges', 'Churn Label',
       'hex_id'],
      dtype='object')
```


```
def encode_data(df):
    # Nominal encoding
    encoder = OneHotEncoder(sparse_output=False, drop='first')

    encoded_data = encoder.fit_transform(data[categorical_columns])
    encoded_data = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(categorical_columns))

    newdf = pd.concat([df, encoded_data], axis=1)
    return newdf.drop(columns=[*categorical_columns, 'hex_id'])

data = encode_data(data)
data.sample(3)
```

	Tenure Months	Monthly Charges	Total Charges	Churn Label	Gender_Male	Senior Citizen_Yes	Partner_Yes	D
3766	71	99.00	7061.65	0	0.0	0.0	1.0	
1199	27	85.25	2287.25	1	0.0	1.0	0.0	
3544	1	49.85	49.85	0	1.0	0.0	0.0	

3 rows × 31 columns

```
data.groupby('Churn Label')['Churn Label'].count()
```

```
Churn Label
0    5174
```

```
1    1869
Name: Churn Label, dtype: int64
```

```
len(data.columns)
```

```
31
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.py:198: RuntimeWarning: divide by zero e
  vif = 1. / (1. - r_squared_i)
                                 Feature          VIF
0                          Tenure Months     7.542697
1                        Monthly Charges   865.450546
2                          Total Charges    10.862958
3                            Churn Label     1.429089
4                            Gender_Male     1.002024
5                     Senior Citizen_Yes     1.135728
6                            Partner_Yes     1.344290
7                         Dependents_Yes     1.273662
8                      Phone Service_Yes  1771.930567
9          Multiple Lines_No phone service    60.982066
10                    Multiple Lines_Yes     7.281349
11          Internet Service_Fiber optic   148.398568
12                   Internet Service_No          inf
13     Online Security_No internet service        inf
14                   Online Security_Yes     6.339317
15       Online Backup_No internet service        inf
16                     Online Backup_Yes     6.783802
17   Device Protection_No internet service        inf
18                 Device Protection_Yes     6.927119
19        Tech Support_No internet service        inf
20                      Tech Support_Yes     6.473185
21        Streaming TV_No internet service        inf
22                      Streaming TV_Yes    24.073788
23    Streaming Movies_No internet service        inf
24                  Streaming Movies_Yes    24.131570
25                     Contract_One year     1.632110
26                     Contract_Two year     2.629070
27                 Paperless Billing_Yes     1.212537
28  Payment Method_Credit card (automatic)     1.560576
29        Payment Method_Electronic check     1.983301
30            Payment Method_Mailed check     1.860244
11
```

```
over = SMOTE(sampling_strategy = 1)

x = data.drop("Churn Label", axis = 1).values
y = data['Churn Label'].values

x,y = over.fit_resample(x,y)

x,y
```

```
(array([[2.00000000e+00, 5.38500000e+01, 1.08150000e+02, ...,
         0.00000000e+00, 0.00000000e+00, 1.00000000e+00],
        [2.00000000e+00, 7.07000000e+01, 1.51650000e+02, ...,
         0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
        [8.00000000e+00, 9.96500000e+01, 8.20500000e+02, ...,
         0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
        ...,
        [2.00000000e+00, 8.72689071e+01, 1.83367799e+02, ...,
         0.00000000e+00, 8.49336565e-02, 0.00000000e+00],
        [1.00000000e+00, 4.25625192e+01, 4.25625192e+01, ...,
         0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
        [4.62333136e+01, 1.07678530e+02, 4.87392507e+03, ...,
         0.00000000e+00, 1.00000000e+00, 0.00000000e+00]]),
 array([1, 1, 1, ..., 1, 1, 1]))
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state =2, test_size = 0.2)
```

```
x_test
```

```
array([[1.70000000e+01, 8.82500000e+01, 1.46065000e+03, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       [8.00000000e+00, 5.13000000e+01, 4.11600000e+02, ...,
        1.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 7.11070051e+01, 7.11070051e+01, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       ...,
       [6.90000000e+01, 4.62500000e+01, 3.12140000e+03, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.70000000e+01, 5.46000000e+01, 9.34800000e+02, ...,
```

```
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 7.47689188e+01, 7.47689188e+01, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00]])
```

```
x_train
```

```
array([[3.00000000e+00, 2.05500000e+01, 5.74000000e+01, ...,
        1.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 4.02000000e+01, 4.02000000e+01, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       [1.50000000e+01, 2.54000000e+01, 3.99600000e+02, ...,
        1.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       ...,
       [6.00000000e+00, 1.95500000e+01, 1.24450000e+02, ...,
        0.00000000e+00, 0.00000000e+00, 1.00000000e+00],
       [5.40000000e+01, 1.04100000e+02, 5.64580000e+03, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 2.07741153e+01, 2.07741153e+01, ...,
        0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

```
x_train[0]
```

```
array([ 3.  , 20.55, 57.4 ,  0.  ,  0.  ,  0.  ,  1.  ,  1.  ,  0.  ,
        0.  ,  0.  ,  1.  ,  1.  ,  0.  ,  1.  ,  0.  ,  1.  ,  0.  ,
        1.  ,  0.  ,  1.  ,  0.  ,  1.  ,  0.  ,  1.  ,  0.  ,  1.  ,
        1.  ,  0.  ,  0.  ])
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Filter only the numerical columns
numerical_data = data.select_dtypes(include=['number'])

# VIF for each feature
vif = pd.DataFrame()
vif["Feature"] = numerical_data.columns
vif["VIF"] = [variance_inflation_factor(numerical_data.values, i) for i in range(numerical_data.shape[1])]


# Select features with VIF below threshold 6
remove_features = vif[vif["VIF"] < 6]["Feature"].values
remove_features
```

```
array(['Monthly Charges', 'Churn Label'], dtype=object)
```

```
data.drop(columns = remove_features)
```

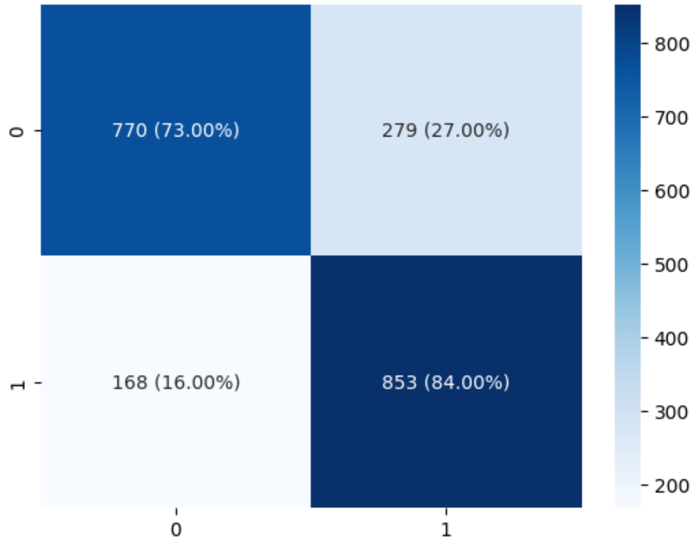
	Gender	Senior Citizen	Partner	Dependents	Tenure Months	Phone Service	Multiple Lines	Internet Service
0	Male	No	No	No	2	Yes	No	DSL

```
def test_model(model, x_train, y_train, x_test, y_test):
    model.fit(x_train, y_train)

    scaler = StandardScaler()
    x_train = scaler.fit_transform(x_train)

    predictions = model.predict(x_test)
    c_matrix = confusion_matrix(y_test, predictions)

    percentages = (c_matrix / np.sum(c_matrix, axis=1)[:, np.newaxis]).round(2) * 100
    labels = [[f"{c_matrix[i, j]} ({percentages[i, j]:.2f}%)" for j in range(c_matrix.shape[1])] for i in range(c_matrix.s
    labels = np.asarray(labels)

    sns.heatmap(c_matrix, annot=labels, fmt='', cmap='Blues')

    print("ROC AUC: ", '{:.2%}'.format(roc_auc_score(y_test, predictions)))
    print("Model accuracy: ", '{:.2%}'.format(accuracy_score(y_test, predictions)))
    print(classification_report(y_test, predictions))

def predict_sample(model, sample):
    prediction = model.predict([sample])[0]
    scaler = StandardScaler()
    sample = scaler.fit_transform([sample])
    prediction = model.predict(sample)[0]
    return prediction

def save_model(model, filename):
    try:
        joblib.dump(model, filename)
        print(f"Model saved to {filename}")
    except Exception as e:
        print(f"Error saving the model: {str(e)}")

# logistic_reg = LogisticRegression()
# test_model(logistic_reg, x_train, y_train, x_test, y_test)
# print(predict_sample(logistic_reg, x_test[0]))
# print(predict_sample(logistic_reg, x_test[1]))



# Train & test different models
def test_model(model, x_train, y_train, x_test, y_test):
    model.fit(x_train, y_train)

    predictions = model.predict(x_test)
    c_matrix = confusion_matrix(y_test, predictions)

    percentages = (c_matrix / np.sum(c_matrix, axis=1)[:, np.newaxis]).round(2) * 100
    labels = [[f"{c_matrix[i, j]} ({percentages[i, j]:.2f}%)" for j in range(c_matrix.shape[1])] for i in range(c_matrix.s
    labels = np.asarray(labels)

    sns.heatmap(c_matrix, annot=labels, fmt='', cmap='Blues')

    print("ROC AUC: ", '{:.2%}'.format(roc_auc_score(y_test, predictions)))
    print("Model accuracy: ", '{:.2%}'.format(accuracy_score(y_test, predictions)))

    mae = mean_absolute_error(y_test, predictions)
    rmse = np.sqrt(mean_squared_error(y_test, predictions))
    print(f'Mean Absolute Error (MAE): {mae}')
    print("Mean squared error (MSE)", mean_squared_error(y_test, predictions))
    print(f'Root Mean Squared Error (RMSE): {rmse}')
    print("Sum squared error (SSE): ", np.sum((y_test - predictions) ** 2))
    print("Sum squared total (SST): ", np.sum((y_test - np.mean(y_test)) ** 2))
    print("R^2: ", r2_score(y_test, predictions))
    print("\n\n")
    print(classification_report(y_test, predictions))

# Predict single sample
def predict_sample(model, sample):
    prediction = model.predict([sample])[0]
    return prediction

# Save pickle
def save_model(model, filename):
    try:
        joblib.dump(model, filename)
```

```
        print(f"Model saved to {filename}")
    except Exception as e:
        print(f"Error saving the model: {str(e)}")
```

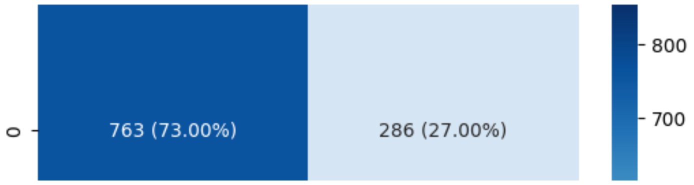
```
logistic_reg = LogisticRegression()
test_model(logistic_reg, x_train, y_train, x_test, y_test)
save_model(logistic_reg, f"{WORKING_DIR}/logistic_reg.pkl")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
  n_iter_i = _check_optimize_result(
ROC AUC:  78.47%
Model accuracy:  78.41%
Mean Absolute Error (MAE): 0.21594202898550724
Mean squared error (MSE) 0.21594202898550724
Root Mean Squared Error (RMSE): 0.4646956304781736
Sum squared error (SSE):  447
Sum squared total (SST):  517.4053140096617
R^2:  0.13607381312737532


              precision    recall  f1-score   support

           0       0.82      0.73      0.78      1049
           1       0.75      0.84      0.79      1021

    accuracy                           0.78      2070
   macro avg       0.79      0.78      0.78      2070
weighted avg       0.79      0.78      0.78      2070

Model saved to /content/drive/MyDrive/datasets/fds_customer_churn/logistic_re
```



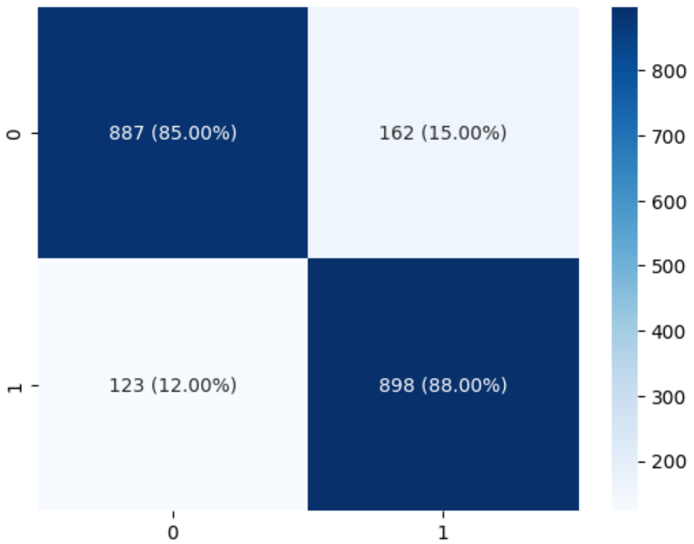

```
logistic_reg_high = LogisticRegression(max_iter=4000)
test_model(logistic_reg_high, x_train, y_train, x_test, y_test)
save_model(logistic_reg_high, f"{WORKING_DIR}/logistic_reg_high.pkl")
```

```
ROC AUC:  78.24%
Model accuracy:  78.16%
Mean Absolute Error (MAE): 0.21835748792270532
Mean squared error (MSE) 0.21835748792270532
Root Mean Squared Error (RMSE): 0.4672873718844811
Sum squared error (SSE):  452
Sum squared total (SST):  517.4053140096617
R^2:  0.12641020924736834


              precision    recall  f1-score   support

           0       0.82      0.73      0.77      1049
           1       0.75      0.84      0.79      1021

    accuracy                           0.78      2070
   macro avg       0.79      0.78      0.78      2070
weighted avg       0.79      0.78      0.78      2070

Model saved to /content/drive/MyDrive/datasets/fds_customer_churn/logistic_re
```



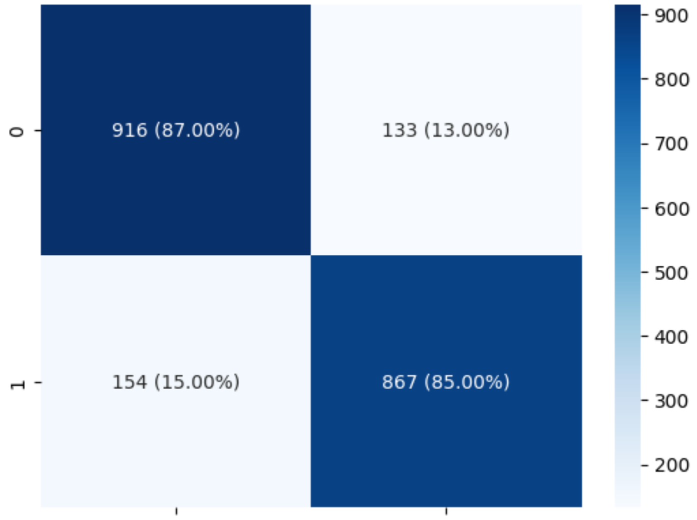

```
xgb = XGBClassifier(learning_rate= 0.01,max_depth = 3,n_estimators = 1000)
test_model(xgb,x_train,y_train,x_test,y_test)
save_model(xgb, f"{WORKING_DIR}/xgb.pkl")
```

```
ROC AUC:  86.25%
Model accuracy:  86.23%
Mean Absolute Error (MAE): 0.13768115942028986
Mean squared error (MSE) 0.13768115942028986
Root Mean Squared Error (RMSE): 0.3710541192606408
Sum squared error (SSE):  285
Sum squared total (SST):  517.4053140096617
R^2:  0.44917457883960177


              precision    recall  f1-score   support

           0       0.88      0.85      0.86      1049
           1       0.85      0.88      0.86      1021

    accuracy                           0.86      2070
   macro avg       0.86      0.86      0.86      2070
weighted avg       0.86      0.86      0.86      2070

Model saved to /content/drive/MyDrive/datasets/fds_customer_churn/xgb.pkl
```




```
rf_classifier = RandomForestClassifier(n_estimators=500, random_state=42)
test_model(rf_classifier,x_train,y_train,x_test,y_test)
save_model(rf_classifier, f"{WORKING_DIR}/rf_classifier.pkl")
```

```
ROC AUC:  86.12%
Model accuracy:  86.14%
Mean Absolute Error (MAE): 0.1386473429951691
Mean squared error (MSE) 0.1386473429951691
Root Mean Squared Error (RMSE): 0.37235378740543124
Sum squared error (SSE):  287
Sum squared total (SST):  517.4053140096617
R^2:  0.4453091372875989


              precision    recall  f1-score   support

           0       0.86      0.87      0.86      1049
           1       0.87      0.85      0.86      1021

    accuracy                           0.86      2070
   macro avg       0.86      0.86      0.86      2070
weighted avg       0.86      0.86      0.86      2070

Model saved to /content/drive/MyDrive/datasets/fds_customer_churn/rf_classifi
```




```
y_test[[3,9,20]]
```

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

```
print(predict_sample(logistic_reg_high, x_test[3]))
print(predict_sample(logistic_reg_high, x_test[9]))
print(predict_sample(logistic_reg_high, x_test[20]))
```

```
1
0
1
```

```
print(predict_sample(xgb, x_test[3]))
print(predict_sample(xgb, x_test[9]))
print(predict_sample(xgb, x_test[20]))
```

```
1
0
1
```

```
print(predict_sample(rf_classifier, x_test[3]))
print(predict_sample(rf_classifier, x_test[9]))
print(predict_sample(rf_classifier, x_test[20]))
```

```
1
0
1
```