PROJECT DOCUMENTATION

Customer Churn Prediction Project

Date	**30-10-2023**
Team ID	**1288**
Project Name	**Customer Churn Prediction**

USE CASE NAME : Customer Churn Prediction

PROBLEM STATEMENT

Use data analytics to predict customer churn and identify factors influencing customer retention, helping businesses reduce customer attrition.

PROJECT OBJECTIVE

Customer churn is a common problem across businesses in many sectors. If you want to grow as a company, you have to invest in acquiring new clients. Every time a client leaves, it represents a significant investment lost.

Objective : The aim is to develop a predictive model that can accurately predict the churn rate and visualize it.

DESIGN THINKING PROCESS

Data Collection

In data collection we used WA_Fn-UseC_-Telco-Customer-Churn.csv dataset which is provided by the IBM team for the data pre-processing, analysis and for the visualization.

Data Pre-processing

Natural Language Processing techniques are used to pre-process data that are in the form of text and service and it provides valuable insights.

K-Nearest Neighbour and Random Forest algorithm is used to the handle the missing values in the dataset.

Model Selection And Training

The Ensemble learning techniques includes the Random Forest, decision Tree, K-Nearest Neighbour and logistic reggression. These model helps in increasing the accuracy of the prediction.

Some other model we used were Bayesian logistic regression and Support Vector Machine.

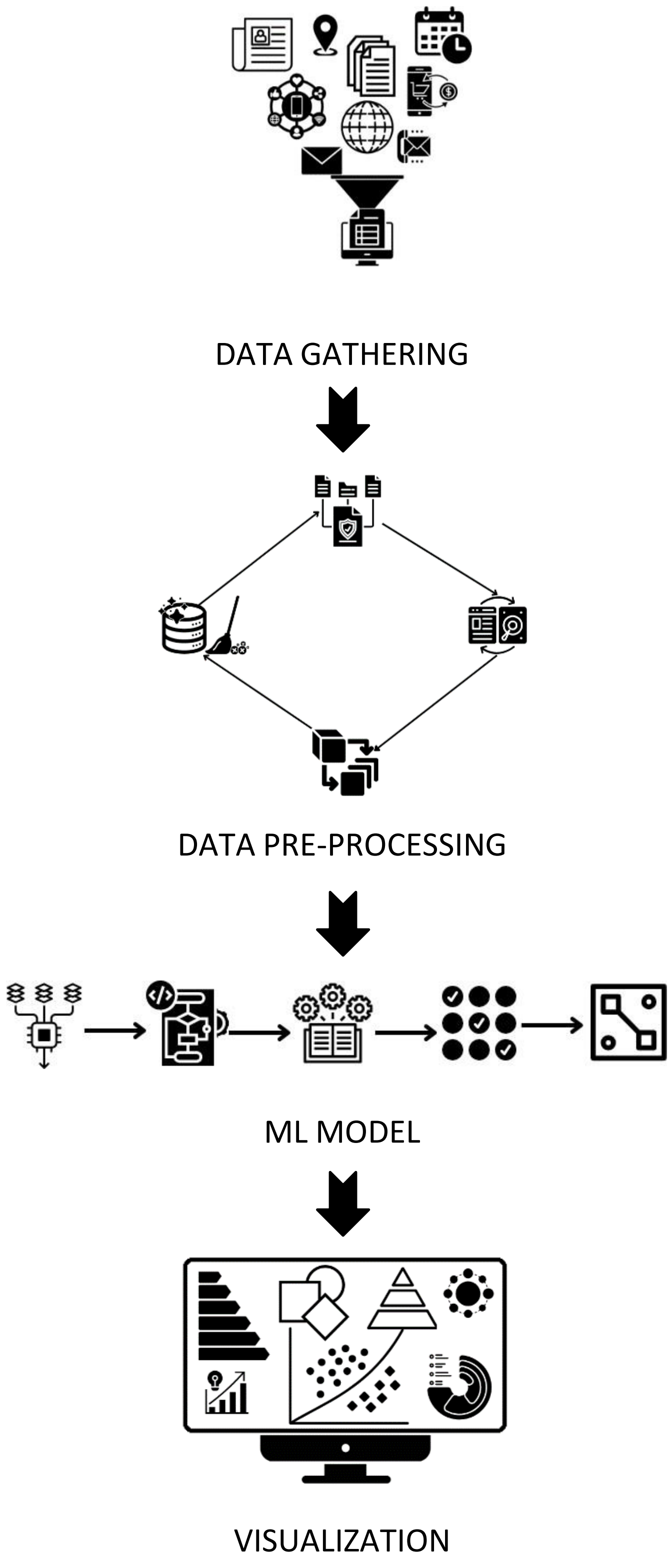
By combining these model that incorporates the ensemble model with others which results in the better outcome.

Visualization

Visualization is the process of showcasing the output from the predictive model in a predefined chart or graph.

There are lots of visualization types like pie chart, scatter plot, bar chart, histogram, etc., in common line chart is mainly used for churn rate prediction.

PROJECT FLOW ARCHITECTURE



DEVELOPMENT PHASES 1 & 2

In development phases we implemented various analysis, a predictive model and the interactive dashboards were developed with the help of jupyter and IBM Cognos platform.

The Machine Learning Models we used in this analysis were Logistic Regression, Support Vector Machine [SVM], Random Forest Classifier, K-Nearest Neighbour and Decision Tree Classifier.

In visualization part we did the churn rate, churn percentage, phone services, internet services and the payment methods from the dataset.

DATA PRE-PROCESSING

Jupyter platform is used for the data pre-processing phase. In that we initially imported the necessary python library files. The library files were

Pandas - used for working with data sets

Numpy - used for working with arrays

Sklearn - Used machine learning models and statistical modelling

and some other important and necessary library for the visualization and the analysis were imported.

jupyter Phase4_IBM_CCP (1) Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

[126]:
```python
# 1.Data Information
# Loading libraries

import pandas as pd
import numpy as np
import missingno as msno
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import recall_score, confusion_matrix, precision_score, f1_score, accuracy_score, classification_report
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Then we imported the given data set “WA_Fn-UseC_-Telco-Customer-Churn.csv” and viewed the first 5 rows of the dataset.

[94]:
```python
# Load data

df = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn (1).csv')
df
```

[94]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	No
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	Yes
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	No
...	...	...	...	...	...	...	...	...	...	...	...	...	...
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	...	Yes	Yes
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	...	Yes	No
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	...	No	No
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes	Fiber optic	No	...	No	No
7042	3186-AJIEK	Male	0	No	No	66	Yes	No	Fiber optic	Yes	...	Yes	Yes

7043 rows × 21 columns

Some basic elementals were viewed such as shape of the dataset, columns of the dataset, values of the dataset and the information of the dataset.

```
[95]: df.shape

[95]: (7043, 21)

[96]: df.columns.values

[96]: array(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
             'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
             'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
             'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
             'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
             'TotalCharges', 'Churn'], dtype=object)

[97]: df.info()

      <class 'pandas.core.frame.DataFrame'>
      RangeIndex: 7043 entries, 0 to 7042
      Data columns (total 21 columns):
       #   Column            Non-Null Count  Dtype  
      ---  ------            --------------  -----  
       0   customerID        7043 non-null   object 
       1   gender            7043 non-null   object 
       2   SeniorCitizen     7043 non-null   int64  
       3   Partner           7043 non-null   object 
       4   Dependents        7043 non-null   object 
       5   tenure            7043 non-null   int64  
       6   PhoneService      7043 non-null   object 
       7   MultipleLines     7043 non-null   object 
       8   InternetService   7043 non-null   object 
       9   OnlineSecurity    7043 non-null   object 
       10  OnlineBackup      7043 non-null   object 
       11  DeviceProtection  7043 non-null   object 
       12  TechSupport       7043 non-null   object 
       13  StreamingTV       7043 non-null   object 
       14  StreamingMovies   7043 non-null   object 
       15  Contract          7043 non-null   object 
       16  PaperlessBilling  7043 non-null   object 
       17  PaymentMethod     7043 non-null   object 
       18  MonthlyCharges    7043 non-null   float64
       19  TotalCharges      7043 non-null   object 
       20  Churn             7043 non-null   object 
      dtypes: float64(1), int64(2), object(18)
      memory usage: 1.1+ MB
```

From here the data pre-processing is initiated and the Customer ID is removed from the dataset.

```
[98]: # 2.Data Preparation
      # Data manipulation

      df = df.drop(['customerID'], axis = 1)
      df.head()
```

[98]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	Stre
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No	No	
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes	No	
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No	No	
3	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Yes	Yes	
4	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No	No	

In this step the Total Charges from the dataset is converted to numerical values and the missing value is checked if it is there.

```
[99]: # Convert 'Total Charges' into numerical values

df['TotalCharges'] = pd.to_numeric(df.TotalCharges, errors = 'coerce') # erros = 'coerce' is used if there are any non-numeric values in the 'TotalCharge

# Check for missing values

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

```
[99]: gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges       11
Churn               0
dtype: int64
```

Here, the missing values are calculated from the dataset.

```
[100]: # Calculate the missing values
def missing_values(n):
    df_m=pd.DataFrame()
    df_m["missing_values, %"]=df.isnull().sum()*100/len(df.isnull())
    df_m["missing_values, sum"]=df.isnull().sum()
    return df_m.sort_values(by="missing_values, %", ascending=False)
missing_values(df)
```

[100]:

	missing_values, %	missing_values, sum
TotalCharges	0.156183	11
gender	0.000000	0
SeniorCitizen	0.000000	0
MonthlyCharges	0.000000	0
PaymentMethod	0.000000	0
PaperlessBilling	0.000000	0
Contract	0.000000	0
StreamingMovies	0.000000	0
StreamingTV	0.000000	0
TechSupport	0.000000	0
DeviceProtection	0.000000	0
OnlineBackup	0.000000	0
OnlineSecurity	0.000000	0
InternetService	0.000000	0
MultipleLines	0.000000	0
PhoneService	0.000000	0
tenure	0.000000	0
Dependents	0.000000	0
Partner	0.000000	0
Churn	0.000000	0

The dataset is filtered to find the missing values in the rows of Total Charges.

[101]:
```python
# Filter df to find rows that have missing values in 'Total Charges'

df[np.isnan(df['TotalCharges'])]
```

[101]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport
488	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	No	Yes	Yes
753	Male	0	No	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
936	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	Yes	Yes	No
1082	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service
1340	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	Yes	Yes	Yes
3331	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
3826	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service
4380	Female	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
5218	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
6670	Female	0	Yes	Yes	0	Yes	Yes	DSL	No	Yes	Yes	Yes
6754	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	Yes	No	Yes

[102]:
```python
# Filter df to find the index positions of rows where the 'tenure' column has a value of 0

df[df['tenure'] == 0].index
```

[102]:
```
Index([488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754], dtype='int64')
```

In this step the missing values are represented as the corresponding monthly charges.

[103]:
```python
# Impute missing values with corresponding monthly charges
df['TotalCharges'].fillna(df['TotalCharges'].mean(), inplace=True)
df[df['tenure'] == 0]
```

[103]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport
488	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	No	Yes	Yes
753	Male	0	No	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
936	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	Yes	Yes	No
1082	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service
1340	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	Yes	Yes	Yes
3331	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
3826	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service
4380	Female	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
5218	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service
6670	Female	0	Yes	Yes	0	Yes	Yes	DSL	No	Yes	Yes	Yes
6754	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	Yes	No	Yes

The missing values were again checked in the customer dataset.

```
[104]: # Check for missing values again
       df.isnull().sum()
```

```
[104]: gender              0
       SeniorCitizen       0
       Partner             0
       Dependents          0
       tenure              0
       PhoneService        0
       MultipleLines       0
       InternetService     0
       OnlineSecurity      0
       OnlineBackup        0
       DeviceProtection    0
       TechSupport         0
       StreamingTV         0
       StreamingMovies     0
       Contract            0
       PaperlessBilling    0
       PaymentMethod       0
       MonthlyCharges      0
       TotalCharges        0
       Churn               0
       dtype: int64
```

There is no missing values in the dataset that means it is cleansed and the data preprocessing steps were completed.

Now, from this we can make analysis and create various visualization for the customer churn prediction.

ANALYSIS

It is the process of inspecting, cleansing, transforming, and modelling data with the goal of discovering useful information by informing conclusions and supporting decision making.

In the process of analysis the churn patterns, retention rates, and key factors influencing churn were discussed.

The analysis part makes us to easily understand the given dataset and makes us to provide a various solutions.

Initially the mean, standard deviation, count, minimum and maximum value for Tenure, Monthly Charges and Total Charges were found from the dataset.

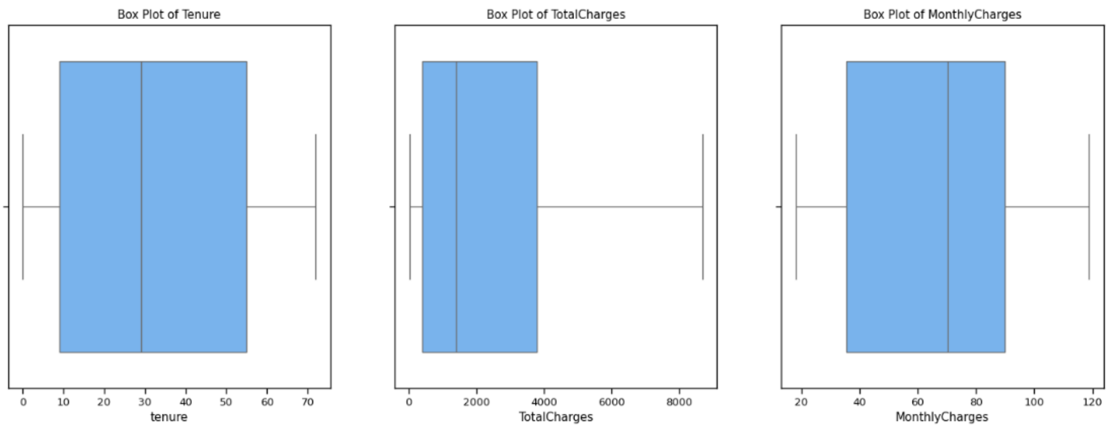
```
[105]: # Define numerical columns
       numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
       df[numerical_cols].describe().T # Describing descriptive stats of the data
```

[105]:

	count	mean	std	min	25%	50%	75%	max
tenure	7043.0	32.371149	24.559481	0.00	9.000	29.00	55.00	72.00
MonthlyCharges	7043.0	64.761692	30.090047	18.25	35.500	70.35	89.85	118.75
TotalCharges	7043.0	2283.300441	2265.000258	18.80	402.225	1400.55	3786.60	8684.80

Then the visualization of box plot for the Tenure, Monthly Charges and Total Charges were implemented.

```
[106]: # Create separate box plots for 'tenure', 'TotalCharges', and 'MonthlyCharges'
       plt.figure(figsize=(18, 6))

       # Box Plot of 'tenure'
       plt.subplot(131)  # 1 row, 3 columns, plot 1
       sns.boxplot(x=df['tenure'], color='#66b3ff')
       plt.title("Box Plot of Tenure")

       # Box Plot of 'TotalCharges'
       plt.subplot(132)  # 1 row, 3 columns, plot 2
       sns.boxplot(x=df['TotalCharges'], color='#66b3ff')
       plt.title("Box Plot of TotalCharges")

       # Box Plot of 'MonthlyCharges'
       plt.subplot(133)  # 1 row, 3 columns, plot 3
       sns.boxplot(x=df['MonthlyCharges'], color='#66b3ff')
       plt.title("Box Plot of MonthlyCharges")

       plt.show()
```



For the informed encoding decision the unique values have to be found from the dataset.

```
[107]: # Check for unique values to be make informed encoding decision
unique_counts = df.nunique()
print("Unique Value Counts:")
print(unique_counts)
```

```
Unique Value Counts:
gender                 2
SeniorCitizen          2
Partner                2
Dependents             2
tenure                73
PhoneService           2
MultipleLines          3
InternetService        3
OnlineSecurity         3
OnlineBackup           3
DeviceProtection       3
TechSupport            3
StreamingTV            3
StreamingMovies        3
Contract               3
PaperlessBilling       2
PaymentMethod          4
MonthlyCharges      1585
TotalCharges        6531
Churn                  2
dtype: int64
```

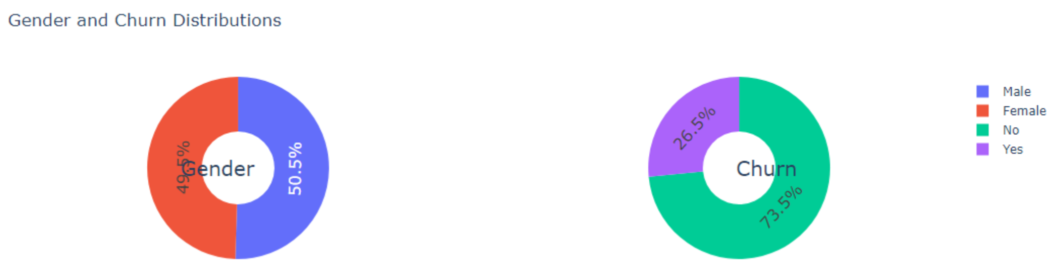
In this step the data types of the dataset column values were displayed.

```
[108]: # Label-Encoding for Categorical Data
# Change data type for categorical data variables
cols = ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService', 'PaperlessBilling', 'Churn']

df[cols] = df[cols].astype('category')

# Label encoding for categorical data variables
for column in cols:
    df[column] = df[column].cat.codes

# Check data types of all columns
print(df.dtypes)
```

```
gender                 int8
SeniorCitizen          int8
Partner                int8
Dependents             int8
tenure                int64
PhoneService           int8
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling       int8
PaymentMethod        object
MonthlyCharges      float64
TotalCharges        float64
Churn                  int8
dtype: object
```

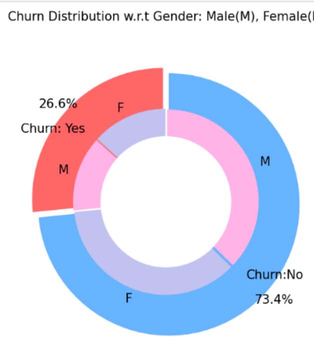
Here is the analysis from the dataset between the columns of churn and gender.

```python
•[109]: # Data Analysis
g_labels = ['Male', 'Female']
c_labels = ['No', 'Yes']

# Create subplots: use 'domain' type for Pie subplot
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]])
fig.add_trace(go.Pie(labels=g_labels, values=df['gender'].value_counts(), name="Gender"),
              1, 1)
fig.add_trace(go.Pie(labels=c_labels, values=df['Churn'].value_counts(), name="Churn"),
              1, 2)

# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.4, hoverinfo="label+percent+name", textfont_size=16)

fig.update_layout(
    title_text="Gender and Churn Distributions",
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='Gender', x=0.16, y=0.5, font_size=20, showarrow=False),
                 dict(text='Churn', x=0.84, y=0.5, font_size=20, showarrow=False)])
fig.show()
```



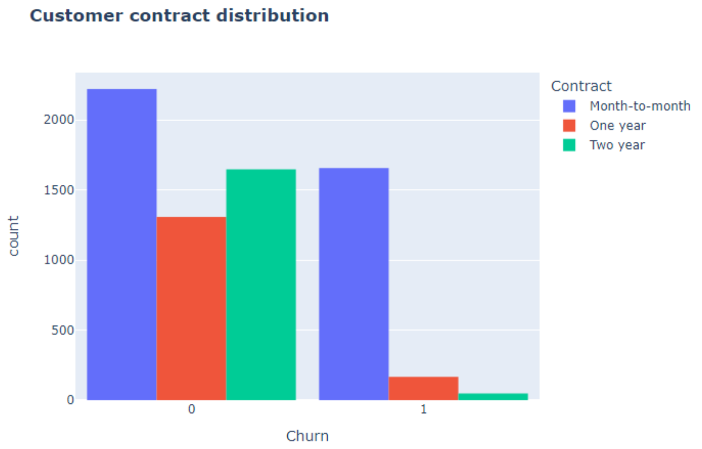
The analysis of churn distribution by gender.

```python
[34]: plt.figure(figsize=(6, 6))
labels =["Churn: Yes","Churn:No"]
values = [1869,5163]
labels_gender = ["F","M","F","M"]
sizes_gender = [939,930 , 2544,2619]
colors = ['#ff6666', '#66b3ff']
colors_gender = ['#c2c2f0','#ffb3e6', '#c2c2f0','#ffb3e6']
explode = (0.3,0.3)
explode_gender = (0.1,0.1,0.1,0.1)
textprops = {"fontsize":15}
#Plot
plt.pie(values, labels=labels,autopct='%1.1f%%',pctdistance=1.08, labeldistance=0.8,colors=colors, startangle=90,frame=True, explode=explode,radius=10,
plt.pie(sizes_gender,labels=labels_gender,colors=colors_gender,startangle=90, explode=explode_gender,radius=7, textprops =textprops, counterclock = True
#Draw circle
centre_circle = plt.Circle((0,0),5,color='black', fc='white',linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Churn Distribution w.r.t Gender: Male(M), Female(F)', fontsize=15, y=1.1)

# show plot

plt.axis('equal')
plt.tight_layout()
plt.show()
```



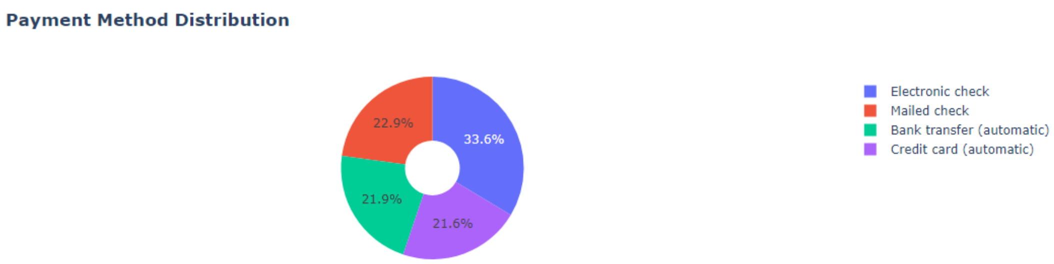
The analysis of customer contract distribution which is classified as Month to Month, One year and two year.

```
[83]: fig = px.histogram(df, x="Churn", color="Contract", barmode="group", title="<b>Customer contract distribution<b>")
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



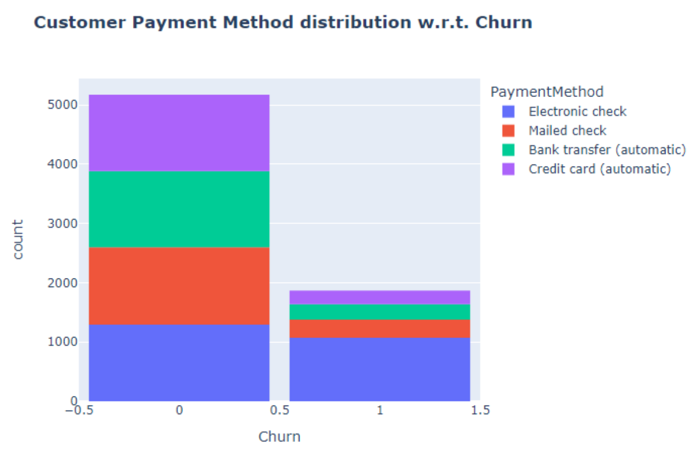
The analysis of payment method distribution which is classified as Electronic Check, Mailed Check, Bank Transfer [Automatic], Credit Card [Automatic].

```
[38]: labels = df['PaymentMethod'].unique()
values = df['PaymentMethod'].value_counts()

fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.3)])
fig.update_layout(title_text="<b>Payment Method Distribution</b>")
fig.show()
```



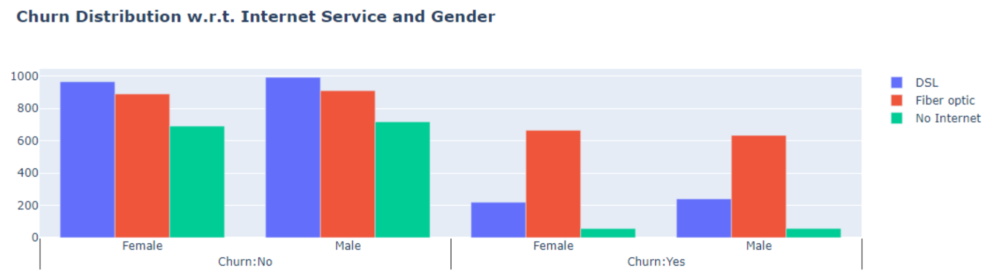
The analysis of customer payment method distribution with respect to churn.

```
[39]: fig = px.histogram(df, x="Churn", color="PaymentMethod", title="<b>Customer Payment Method distribution w.r.t. Churn</b>")
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



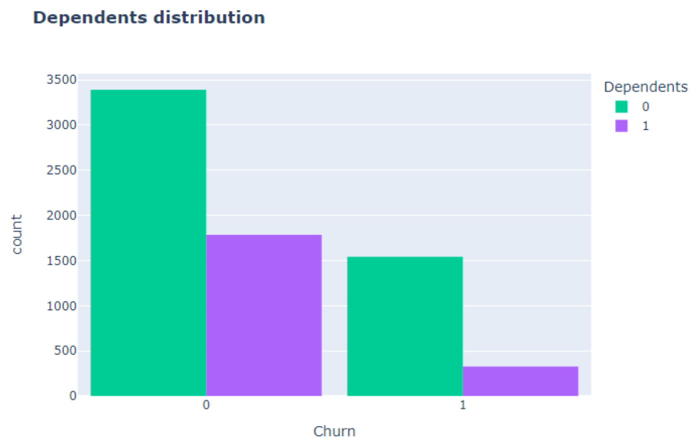
The analysis of churn distribution with respect to internet service and gender.

```
[40]: fig = go.Figure()

fig.add_trace(go.Bar(
  x = [['Churn:No', 'Churn:No', 'Churn:Yes', 'Churn:Yes'],
       ["Female", "Male", "Female", "Male"]],
  y = [965, 992, 219, 240],
  name = 'DSL',
))

fig.add_trace(go.Bar(
  x = [['Churn:No', 'Churn:No', 'Churn:Yes', 'Churn:Yes'],
       ["Female", "Male", "Female", "Male"]],
  y = [889, 910, 664, 633],
  name = 'Fiber optic',
))

fig.add_trace(go.Bar(
  x = [['Churn:No', 'Churn:No', 'Churn:Yes', 'Churn:Yes'],
       ["Female", "Male", "Female", "Male"]],
  y = [690, 717, 56, 57],
  name = 'No Internet',
))

fig.update_layout(title_text="<b>Churn Distribution w.r.t. Internet Service and Gender</b>")

fig.show()
```



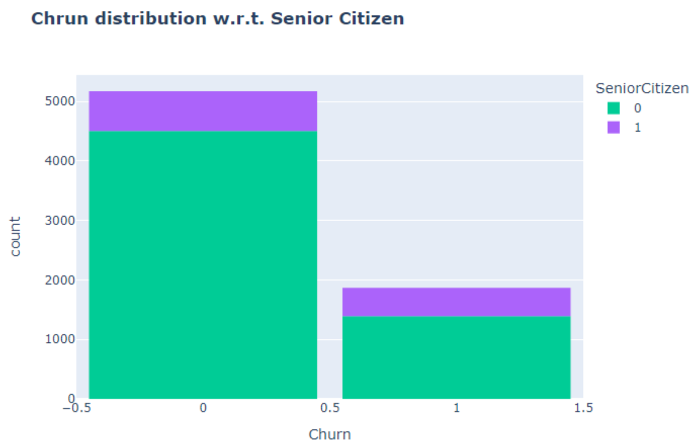
The analysis of dependents distribution which is in numerical values of 0’s and 1’s.

```
[41]: color_map = {"Yes": "#FF97FF", "No": "#AB63FA"}
      fig = px.histogram(df, x="Churn", color="Dependents", barmode="group", title="<b>Dependents distribution</b>", color_discrete_map=color_map)
      fig.update_layout(width=700, height=500, bargap=0.1)
      fig.show()
```



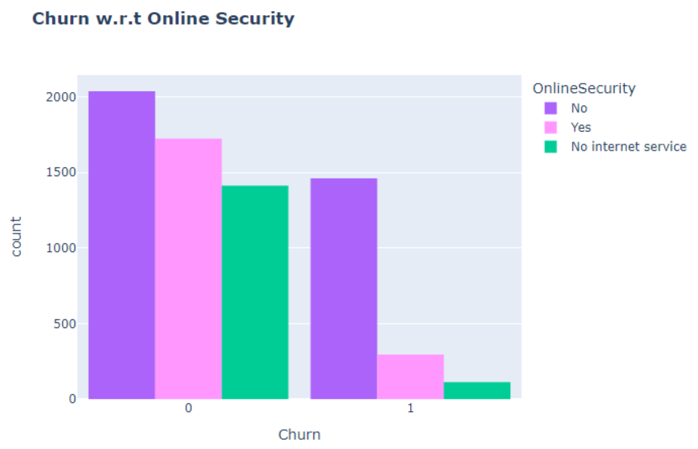
The analysis of churn distribution with respect to senior citizen.

```
[42]: color_map = {"Yes": '#00CC96', "No": '#B6E880'}
      fig = px.histogram(df, x="Churn", color="SeniorCitizen", title="<b>Chrun distribution w.r.t. Senior Citizen</b>", color_discrete_map=color_map)
      fig.update_layout(width=700, height=500, bargap=0.1)
      fig.show()
```



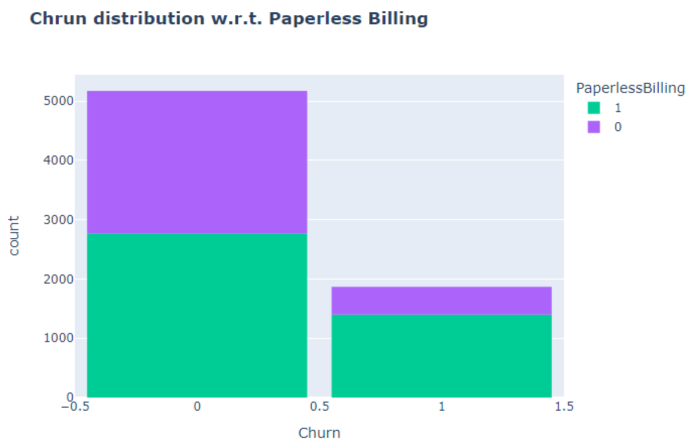
The analysis of churn rate with respect to online security.

```
[43]: color_map = {"Yes": "#FF97FF", "No": "#AB63FA"}
fig = px.histogram(df, x="Churn", color="OnlineSecurity", barmode="group", title="<b>Churn w.r.t Online Security</b>", color_discrete_map=color_map)
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



The analysis of churn distribution with respect to paperless billing.

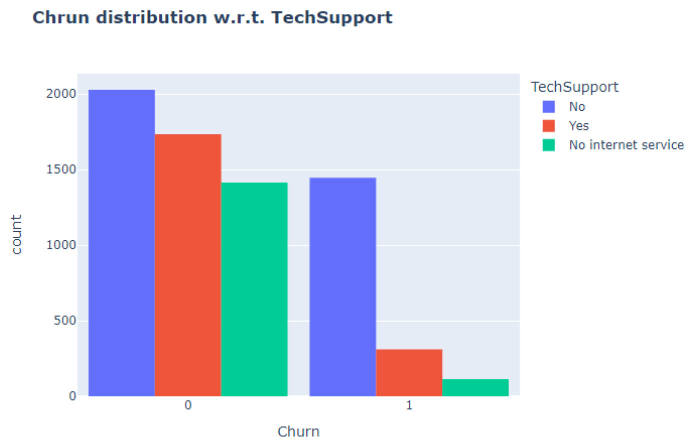
```
[44]: color_map = {"Yes": '#FFA15A', "No": '#00CC96'}
fig = px.histogram(df, x="Churn", color="PaperlessBilling",  title="<b>Chrun distribution w.r.t. Paperless Billing</b>", color_discrete_map=color_map)
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



The analysis of churn distribution with respect to tech support.

[45]:

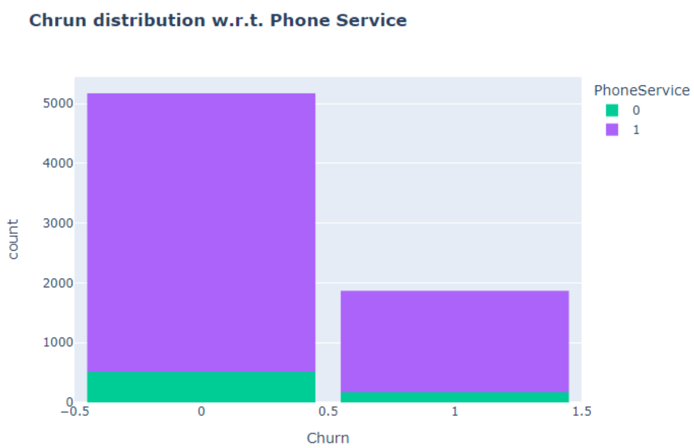
```
fig = px.histogram(df, x="Churn", color="TechSupport",barmode="group",  title="<b>Chrun distribution w.r.t. TechSupport</b>")
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



The analysis of churn distribution with respect to phone services.

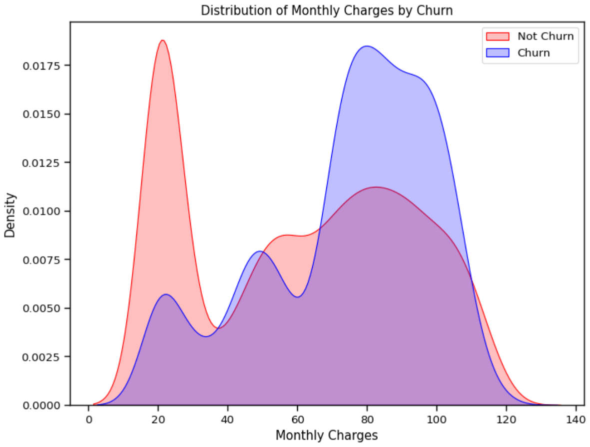
[46]:

```
color_map = {"Yes": '#00CC96', "No": '#B6E880'}
fig = px.histogram(df, x="Churn", color="PhoneService", title="<b>Chrun distribution w.r.t. Phone Service</b>", color_discrete_map=color_map)
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



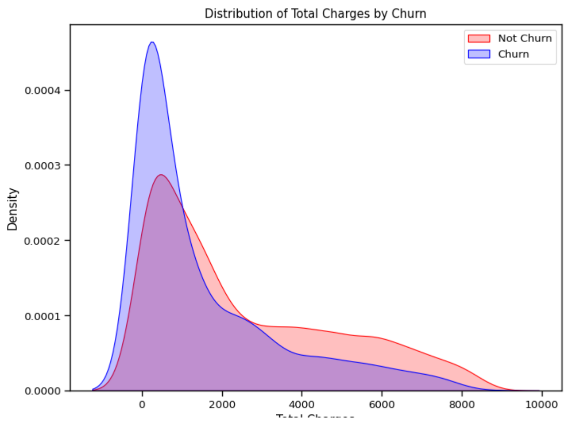
The analysis of Monthly charges distribution with respect to churn.

```
[47]:
sns.set_context("paper", font_scale=1.1)
plt.figure(figsize=(8, 6))

# Line plot for customers who do not churn (Churn = 0)
sns.kdeplot(df.MonthlyCharges[df['Churn'] == 0], color='red', label='Not Churn', shade=True)

# Line plot for customers who churn (Churn = 1)
sns.kdeplot(df.MonthlyCharges[df['Churn'] == 1], color='blue', label='Churn', shade=True)

plt.xlabel('Monthly Charges')
plt.ylabel('Density')
plt.title('Distribution of Monthly Charges by Churn')
plt.legend()

plt.show()
```



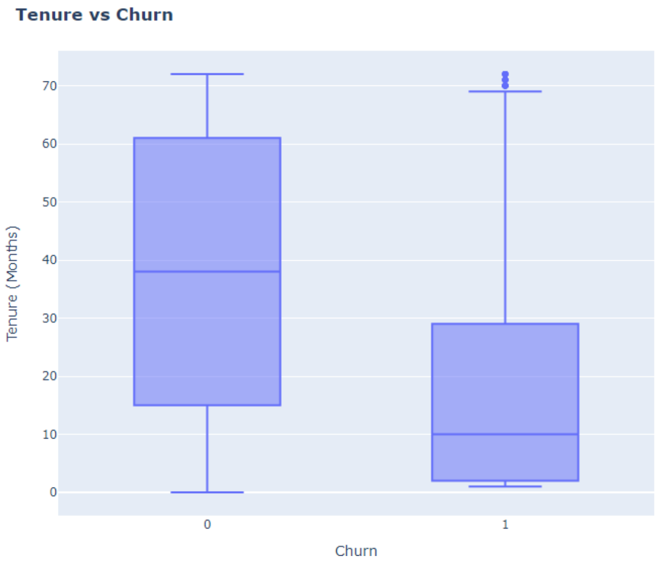
The analysis of Total charges distribution with respect to churn.

```
[48]:
sns.set_context("paper", font_scale=1.1)
plt.figure(figsize=(8, 6))

# Line plot for customers who do not churn (Churn = 0)
sns.kdeplot(df.TotalCharges[df['Churn'] == 0], color='red', label='Not Churn', shade=True)

# Line plot for customers who churn (Churn = 1)
sns.kdeplot(df.TotalCharges[df['Churn'] == 1], color='blue', label='Churn', shade=True)

plt.xlabel('Total Charges')
plt.ylabel('Density')
plt.title('Distribution of Total Charges by Churn')
plt.legend()

plt.show()
```



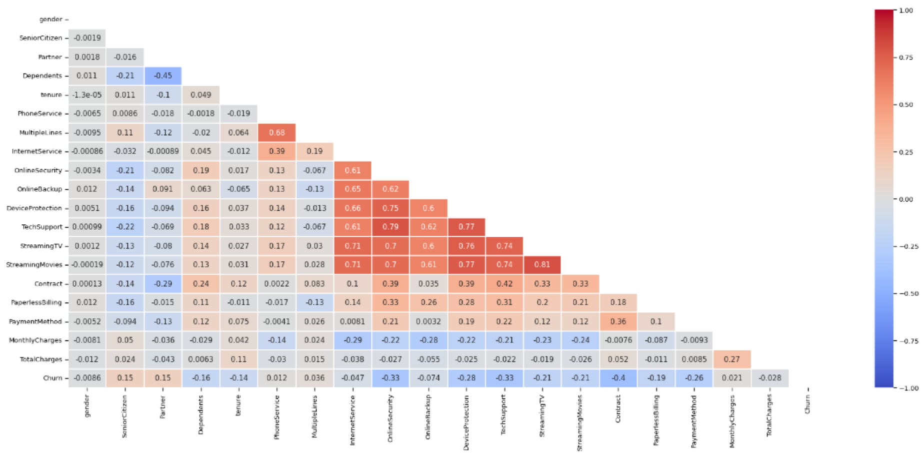
The analysis of box plot between Tenure and Churn.

```
[49]: fig = px.box(df, x='Churn', y = 'tenure')

      # Update yaxis properties
      fig.update_yaxes(title_text='Tenure (Months)', row=1, col=1)
      # Update xaxis properties
      fig.update_xaxes(title_text='Churn', row=1, col=1)

      # Update size and title
      fig.update_layout(autosize=True, width=750, height=600,
          title_font=dict(size=25, family='Courier'),
          title='<b>Tenure vs Churn</b>',
      )

      fig.show()
```



The analysis of Correlation between all variables.

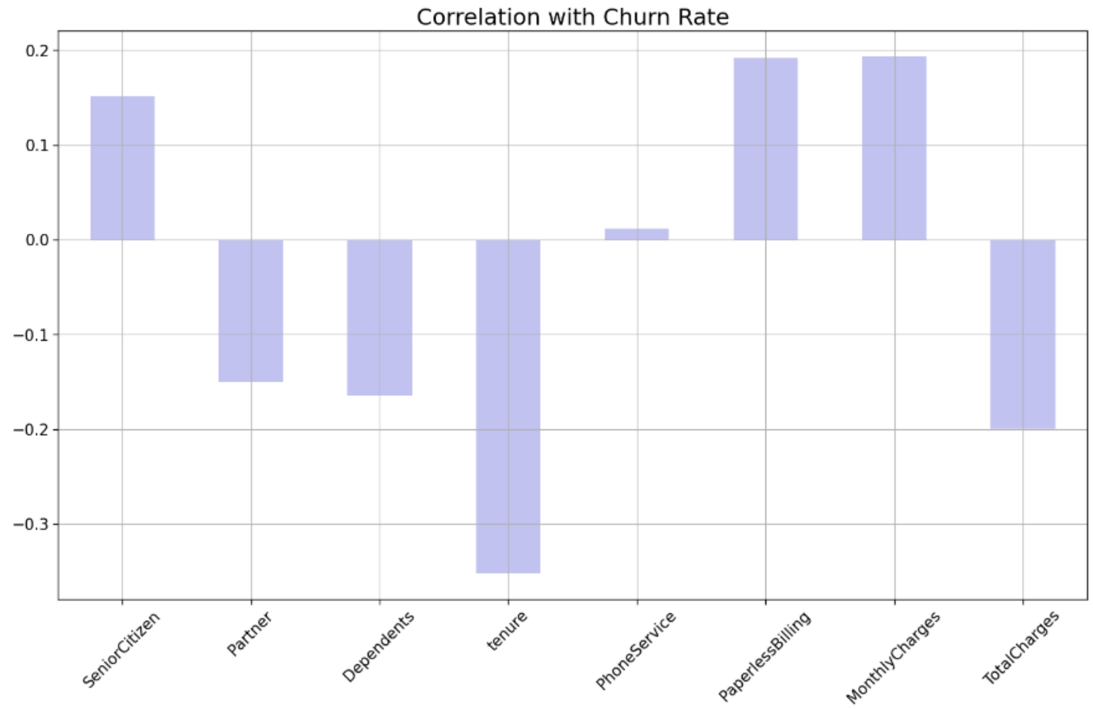
```
[50]: # Correlation between all variables

      plt.figure(figsize=(25, 10))

      corr = df.apply(lambda x: pd.factorize(x)[0]).corr()

      mask = np.triu(np.ones_like(corr, dtype=bool))

      ax = sns.heatmap(corr, mask=mask, xticklabels=corr.columns, yticklabels=corr.columns, annot=True, linewidths=.2, cmap='coolwarm', vmin=-1, vmax=1)
```



The analysis of correlation of churn with all variables.

```
[51]: # Correlation between churn and selected boolean and numeric variables
plt.figure
ds_corr = df[['SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'PaperlessBilling',
        'MonthlyCharges', 'TotalCharges']]

correlations = ds_corr.corrwith(df.Churn)
correlations = correlations[correlations!=1]
correlations.plot.bar(
        figsize = (18, 10),
        fontsize = 15,
        color = '#c2c2f0',
        rot = 45, grid = True)

plt.title('Correlation with Churn Rate', horizontalalignment="center", fontstyle = "normal", fontsize = "22", fontfamily = "sans-serif")
```

```
[51]: Text(0.5, 1.0, 'Correlation with Churn Rate')
```



The process of copying the dataset to maintain the original dataset in it's actual format.

```
[52]: # Copy data to new 'dataset' variable to conserve original values
dataset = df.copy()

# Hot-Encoding for categorical data
dataset = pd.get_dummies(dataset)
dataset.head()
```

[52]:

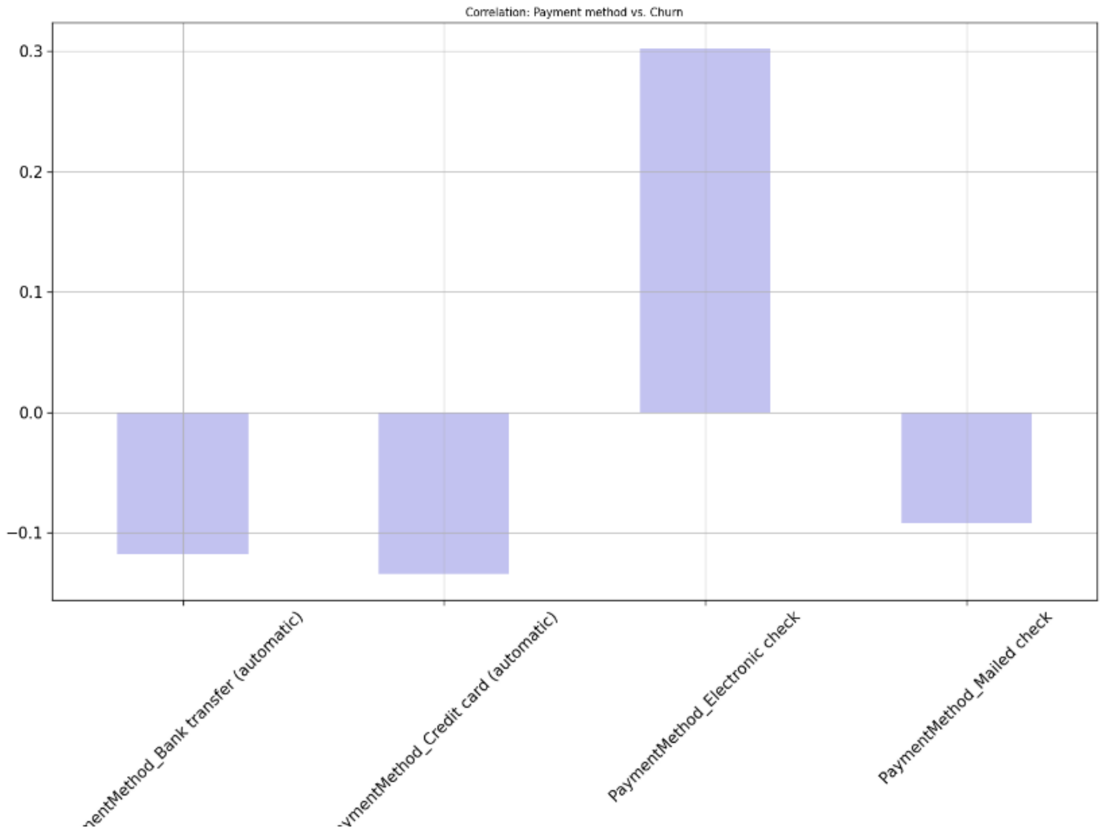
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	MonthlyCharges	TotalCharges	Churn	...	StreamingMovies_No	StreamingMovie internet se
0	0	0	1	0	1	0	1	29.85	29.85	0	...	True	
1	1	0	0	0	34	1	0	56.95	1889.50	0	...	True	
2	1	0	0	0	2	1	1	53.85	108.15	1	...	True	
3	1	0	0	0	45	0	0	42.30	1840.75	0	...	True	
4	0	0	0	0	2	1	1	70.70	151.65	1	...	True	

5 rows × 41 columns

The analysis of correlation between payment methods and churn rate.

```
[54]: # Correlation: PaymentMethod vs. Churn
plt.figure

ds_payment_method_corr = dataset[['PaymentMethod_Bank transfer (automatic)',
          'PaymentMethod_Credit card (automatic)',
          'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check']]

correlations = ds_payment_method_corr.corrwith(dataset.Churn)
correlations = correlations[correlations!=1]

correlations.plot.bar(
        figsize = (18, 10),
        fontsize = 15,
        color = '#c2c2f0',
        rot = 45, grid = True)

plt.title('Correlation: Payment method vs. Churn')
```

```
[54]: Text(0.5, 1.0, 'Correlation: Payment method vs. Churn')
```



In this step the dataset is classified as features and target variable then it is splitted into train and test dataset.

Multicollinearity is implemented which is used to correleate several independent variables.

```
[55]: # Split the dataset into features (X) and target variable (y)
X = dataset.drop(columns=['Churn'])  # Features
y = dataset['Churn']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

```
[111]: # Multicollinearity

from statsmodels.stats.outliers_influence import variance_inflation_factor

def calculate_vif(X):
# Calculate Variable Inflation Factors
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["Variable Inflation Factors"] = [variance_inflation_factor(X.values, i)
    for i in range(X.shape[1])]
    return(vif)

ds_vif = dataset[['gender', 'SeniorCitizen', 'Partner', 'Dependents',\
                  'tenure', 'PhoneService', 'PaperlessBilling', \
                  'MonthlyCharges','TotalCharges']]

vif = calculate_vif(ds_vif)
vif
```

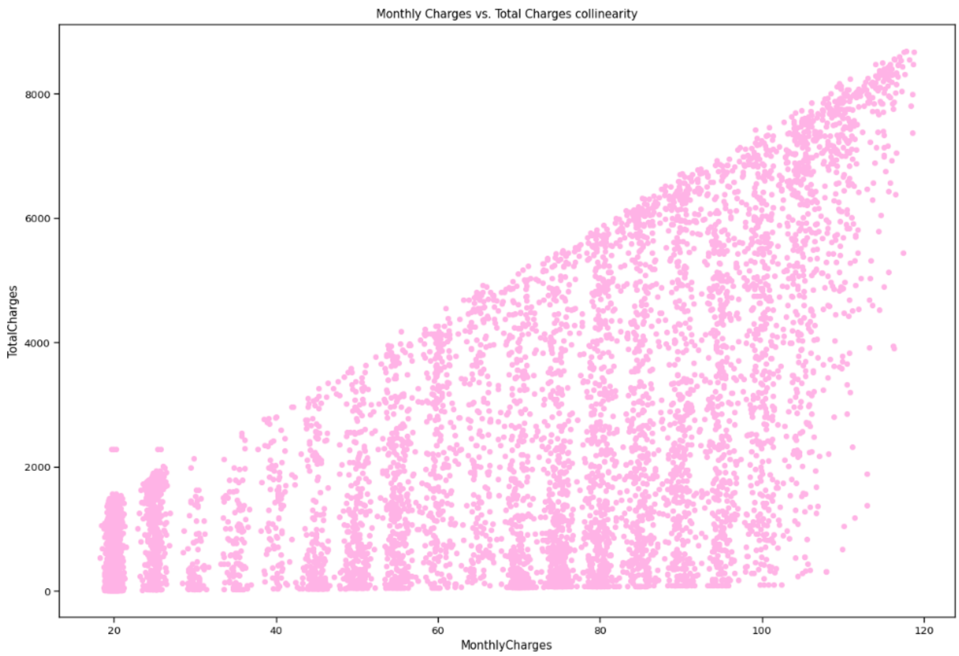
[111]:

	variables	Variable Inflation Factors
0	gender	1.921285
1	SeniorCitizen	1.327766
2	Partner	2.815272
3	Dependents	1.921208
4	tenure	10.549726
5	PhoneService	7.976437
6	PaperlessBilling	2.814154
7	MonthlyCharges	13.988695
8	TotalCharges	12.570370

The analysis of collinearity between Monthly charges and Total charges.

```
[113]:
ds_vif[['MonthlyCharges', 'TotalCharges']]\
           .plot.scatter(figsize = (15, 10),
               x ='MonthlyCharges',
               y='TotalCharges',
               color = '#ffb3e6')

plt.title('Monthly Charges vs. Total Charges collinearity')
```

```
[113]: Text(0.5, 1.0, 'Monthly Charges vs. Total Charges collinearity')
```



PREDICTIVE MODELS

Predictive modeling is a commonly used statistical technique to predict future behavior.

Predictive modeling solutions are a form of data-mining technology that works by analyzing historical and current data and generating a model to help predict future outcomes.

The predictive model employed in the customer churn analysis were Logistic Regression, Support Vector Machine [SVM], Random Forest Classifier, K-Nearest Neighbour and Decision Tree Classifier.

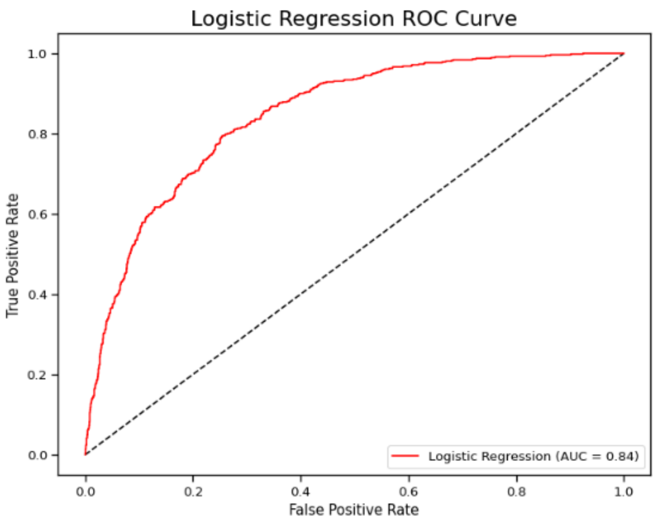
The accuracy of these ML Models were found to get the insights from the dataset.

```
•[116]: # 4.Build Machine Learning Models
        # Split the dataset into features (X) and target variable (y)
        X = dataset.drop(columns=['Churn'])  # Features
        y = dataset['Churn']  # Target variable

        # Split the data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

```
[117]: # Check for missing values in the training set
       missing_train = X_train.isnull().sum()
       print("Missing values in training set:")
       print(missing_train[missing_train > 0])

       # Check for missing values in the test set
       missing_test = X_test.isnull().sum()
       print("\nMissing values in test set:")
       print(missing_test[missing_test > 0])

       Missing values in training set:
       Series([], dtype: int64)

       Missing values in test set:
       Series([], dtype: int64)
```

The predictive model is trained and it calculates the AUC [Area Under Curve] of the regression then it plots the ROC [Receiver Operating Characteristic]curve of the regression.

```
[64]: from sklearn.metrics import roc_curve, auc
      import matplotlib.pyplot as plt

      # Train your Logistic Regression model (logistic_regression_model) on X_train and y_train

      # Assuming you've trained the model, you can proceed with calculating ROC and AUC
      logistic_regression_model = LogisticRegression()
      logistic_regression_model.fit(X_train, y_train)

      y_pred_prob = logistic_regression_model.predict_proba(X_test)[:, 1]
      fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)


      # Calculate AUC
      roc_auc_lr = auc(fpr, tpr)

      # Plot the ROC curve
      plt.figure(figsize=(8, 6))
      plt.plot([0, 1], [0, 1], 'k--')
      plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {roc_auc_lr:.2f})', color='r')
      plt.xlabel('False Positive Rate')
      plt.ylabel('True Positive Rate')
      plt.title('Logistic Regression ROC Curve', fontsize=16)
      plt.legend(loc='lower right')
      plt.show()
```



The accuracy of the regression is calculated.

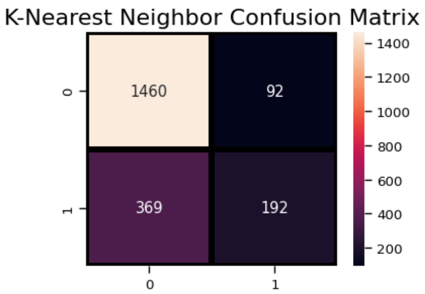
```
[127]: from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Create and fit the Logistic Regression model (lr) to your data
lr = LogisticRegression(solver='liblinear')
lr.fit(X_telco, y_telco)

# Make predictions using the trained model
y_pred = lr.predict(X_telco)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_telco, y_pred)

# Print or display the accuracy
print("Accuracy of Logistic Regression model: {:.2f}".format(accuracy))
```

```
Accuracy of Logistic Regression model: 0.80
```

Now the K-Nearest Neighbour model is implemented.

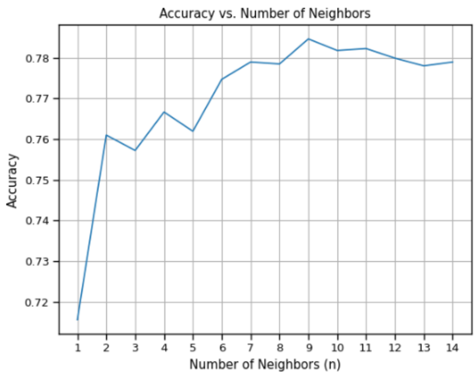
```
[57]: knn_model = KNeighborsClassifier(n_neighbors = 10)

knn_model.fit(X_train,y_train)

# Evaluate model
accuracy_knn = knn_model.score(X_test,y_test)
print("Accuracy of K-Nearest Neighbor: ", accuracy_knn)

# Classification report
knn_prediction = knn_model.predict(X_test)
print(classification_report(y_test, knn_prediction))

plt.figure(14)
plt.figure(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test, knn_prediction),
                annot=True, fmt = "d", linecolor="k", linewidths=3)

plt.title("K-Nearest Neighbor Confusion Matrix", fontsize=16)
plt.show()
```

```
Accuracy of K-Nearest Neighbor:  0.7818267865593942
              precision    recall  f1-score   support

           0       0.80      0.94      0.86      1552
           1       0.68      0.34      0.45       561

    accuracy                           0.78      2113
   macro avg       0.74      0.64      0.66      2113
weighted avg       0.77      0.78      0.76      2113

<Figure size 640x480 with 0 Axes>
```



The analysis of accuracy and with the number of neighbours

[58]:
```
a_index = list(range(1, 15))
accuracies = []  # Initialize an empty list to store accuracy values

for i in a_index:
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    accuracy = metrics.accuracy_score(y_test, prediction)
    accuracies.append(accuracy)

# Plot the accuracy values
import matplotlib.pyplot as plt

plt.plot(a_index, accuracies)
plt.xticks(a_index)
plt.xlabel('Number of Neighbors (n)')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. Number of Neighbors')
plt.grid(True)
plt.show()

print('Accuracies for different values of n are:', accuracies)
```

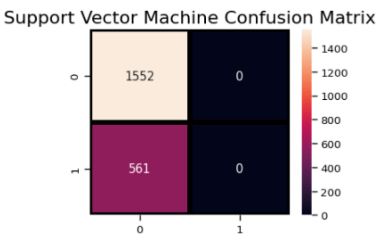



```
Accuracies for different values of n are: [0.7155702792238523, 0.7610033128253668, 0.7572172266919073, 0.7666824420255561, 0.7619498343587316, 0.7747278
750591576, 0.7789872219592996, 0.7785139611926172, 0.7846663511594889, 0.7818267865593942, 0.7823000473260767, 0.7799337434926644, 0.7780407004259347,
0.7789872219592996]
```

The analysis of Support Vector Machine [SVM]

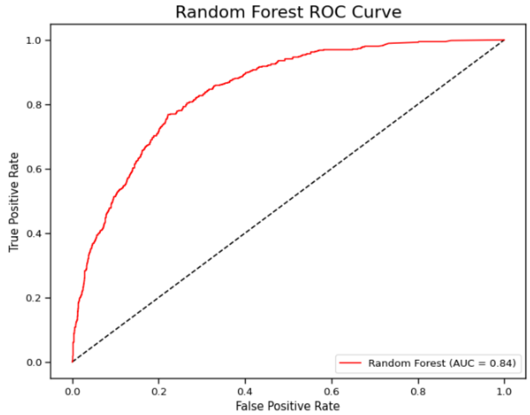
[59]:
```
svc_model = SVC(random_state = 42)
svc_model.fit(X_train,y_train)

# Evaluate model
accuracy_svc = svc_model.score(X_test,y_test)
print("Accuracy of Support Vector Machine: ", accuracy_svc)

# Classification repoort
svc_prediction = svc_model.predict(X_test)
print(classification_report(y_test, svc_prediction))

# Confusion Matrix
plt.figure(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test, svc_prediction),
                annot=True, fmt = "d", linecolor="k", linewidths=3)


plt.title("Support Vector Machine Confusion Matrix", fontsize=16)
plt.show()
```

```
Accuracy of Support Vector Machine:  0.73450070989115
              precision    recall  f1-score   support

           0       0.73      1.00      0.85      1552
           1       0.00      0.00      0.00       561

    accuracy                           0.73      2113
   macro avg       0.37      0.50      0.42      2113
weighted avg       0.54      0.73      0.62      2113
```



The analysis of Random Forest Classifier

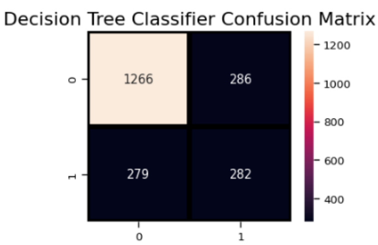
[61]:
```
from sklearn.metrics import auc

y_rfpred_prob = random_forest_model.predict_proba(X_test)[:,1]
fpr_rf, tpr_rf, thresholds = roc_curve(y_test, y_rfpred_prob)

# Calculate AUC
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {roc_auc_rf:.2f})', color='r')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Curve', fontsize=16)
plt.legend(loc='lower right')
plt.show()
```



The analysis of Decision Tree Classifier

[62]:
```
decision_tree_model = DecisionTreeClassifier(random_state=42)
decision_tree_model.fit(X_train,y_train)

# Evaluate model
accuracy_decision_tree = decision_tree_model.score(X_test, y_test)
print("Accuracy of Decision Tree: ", accuracy_decision_tree)

# Decision Tree Classifier gives very low accuracy score.

# Classification report
decision_tree_prediction = decision_tree_model.predict(X_test)
print(classification_report(y_test, decision_tree_prediction))

# Confusion Matrix
plt.figure(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test, decision_tree_prediction),
            annot=True, fmt="d", linecolor="k", linewidths=3)

plt.title("Decision Tree Classifier Confusion Matrix", fontsize=16)
plt.show()
```

```
Accuracy of Decision Tree:  0.7326076668244202
              precision    recall  f1-score   support

           0       0.82      0.82      0.82      1552
           1       0.50      0.50      0.50       561

    accuracy                           0.73      2113
   macro avg       0.66      0.66      0.66      2113
weighted avg       0.73      0.73      0.73      2113
```



The analysis of calculating the accuracy for each of the predictive model

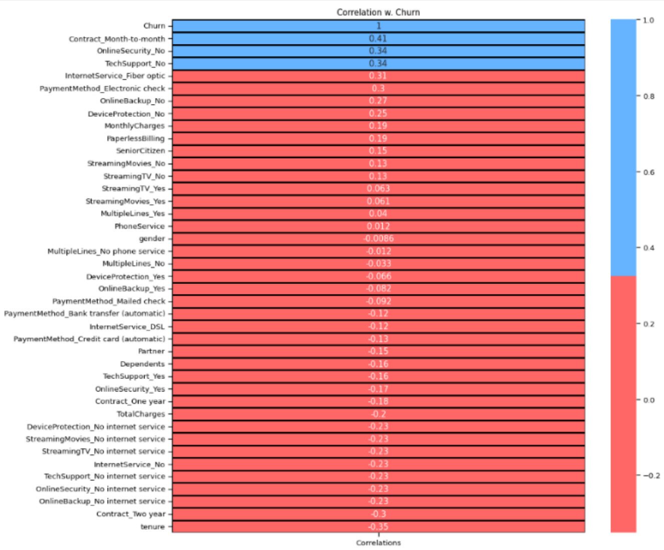
```
[131]: xyz = []

classifiers = ['SVC', 'Random Forest Classifier', 'Logistic Regression', 'kNN', 'Decision Tree']
models = [SVC(random_state=42), RandomForestClassifier(n_estimators=500, random_state=42),
          LogisticRegression(solver='liblinear'), KNeighborsClassifier(n_neighbors=10),
          DecisionTreeClassifier(random_state=42)]

for model in models:
    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    accuracy = metrics.accuracy_score(prediction, y_test)
    xyz.append(accuracy)

models_dataframe = pd.DataFrame(xyz, index=classifiers, columns=['Accuracy'])
models_dataframe
```

[131]:

	Accuracy
SVC	0.791292
Random Forest Classifier	0.777567
Logistic Regression	0.800284
kNN	0.781354
Decision Tree	0.723616

The analysis of correlation between all variables and columns in the dataset.

```
[66]: corr = dataset.corrwith(df['Churn']).sort_values(ascending = False).to_frame()
corr.columns = ['Correlations']
plt.subplots(figsize = (12,12))
sns.heatmap(corr,annot = True,cmap = colors,linewidths = 2,linecolor = 'black');
plt.title('Correlation w. Churn');
```



The process of excluding the list of columns, creating a dataframe with excluded columns, extraction of target variable and feature columns and standardization.

[67]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# List of columns to exclude
columns_to_exclude = ['StreamingTV_Yes', 'StreamingMovies_Yes', 'MultipleLines_Yes', 'PhoneService',
                      'gender','MultipleLines_No', 'MultipleLines_No phone service', 'DeviceProtection_Yes',
                      'OnlineBackup_Yes', 'PaymentMethod_Mailed check']

# Create a new DataFrame with excluded columns
selected_features = dataset.drop(columns=columns_to_exclude)

# Extract the target variable
telco_target = selected_features['Churn']

# Extract the feature columns
telco_features = selected_features.drop(columns=['Churn'])

# Standardization
scaler = StandardScaler()
telco_features_standard = scaler.fit_transform(telco_features)
telco_features_standard = pd.DataFrame(telco_features_standard, columns=telco_features.columns)

# Split the dataset into training and testing sets
X_telco = telco_features_standard
y_telco = telco_target

X_telco_train, X_telco_test, y_telco_train, y_telco_test = train_test_split(X_telco, y_telco, test_size=0.25, random_state=42, stratify=y_telco)
```

[68]:
```python
abc = []
classifiers = ['SVC','Random Forest Classifier','Logistic Regression','kNN','Decision Tree']
models = [SVC(random_state=42), RandomForestClassifier(n_estimators=500, random_state=42), LogisticRegression(solver='liblinear'), KNeighborsClassifier(n

for i in models:
    model = i
    model.fit(X_telco_train,y_telco_train)
    prediction=model.predict(X_telco_test)
    abc.append(metrics.accuracy_score(prediction,y_telco_test))

new_models_dataframe=pd.DataFrame(abc,index=classifiers)
new_models_dataframe.columns=['New Accuracy']
```

The new accuracy of the predictive models were found and the cross-validation mean were found for the models.

[69]:
```python
new_models_dataframe=new_models_dataframe.merge(models_dataframe,left_index=True,right_index=True,how='left')
new_models_dataframe['Increase']=new_models_dataframe['New Accuracy']-new_models_dataframe['Accuracy']
new_models_dataframe
```

[69]:

	New Accuracy	Accuracy	Increase
SVC	0.795571	0.734501	0.061070
Random Forest Classifier	0.784781	0.779934	0.004848
Logistic Regression	0.798978	0.810222	-0.011245
kNN	0.781374	0.781827	-0.000453
Decision Tree	0.735378	0.732608	0.002770

[70]:
```python
from sklearn.model_selection import KFold #for K-fold cross validation
from sklearn.model_selection import cross_val_score #score evaluation
```

[71]:
```python
kfold = KFold(n_splits=5, shuffle=True, random_state=22) # k=5, split the data into 5 equal parts
```

[129]:
```python
xyz = []
accuracy = []
classifier_names = [
    ('kNN', KNeighborsClassifier()),
    ('SVM', SVC()),
    ('Random Forest', RandomForestClassifier(n_estimators=500, random_state=42)),
    ('Decision Tree', DecisionTreeClassifier(random_state=42)),
    ('Logistic Regression', LogisticRegression(solver='liblinear'))
]

for name, model in classifier_names:
    cv_result = cross_val_score(model, X_telco, y_telco, cv=kfold, scoring="accuracy")
    xyz.append(cv_result.mean())
    accuracy.append(cv_result)

new_models_df = pd.DataFrame(xyz, index=[name for name, _ in classifier_names], columns=['CV Mean'])
new_models_df
```

[129]:

	CV Mean
kNN	0.758627
SVM	0.795685
Random Forest	0.787166
Decision Tree	0.728951
Logistic Regression	0.802645

The analysis of corporating two ML models to get an accuracy level by ensemling them.

```
[73]: SVM = SVC(random_state = 42, C = 0.1, probability = True)
LR = LogisticRegression(C = 0.1)
kNN = KNeighborsClassifier(n_neighbors=8)
RF = RandomForestClassifier(random_state=42, n_estimators=500)
DT = DecisionTreeClassifier(random_state=42)

# Set probability=True for all models to enable probability estimates
kNN.probability = True
RF.probability = True
DT.probability = True
```

```
[74]: from sklearn.ensemble import VotingClassifier #for Voting Classifier
```

```
[75]: ens_results = []
ensemble_SVM_LR = VotingClassifier(estimators=[('SVC', SVM), ('LR', LR)],
                      voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_SVM_LR.score(X_telco_test, y_telco_test))
print('The accuracy for SVM and LR is:', ensemble_SVM_LR.score(X_telco_test, y_telco_test))
```

```
The accuracy for SVM and LR is: 0.7893242475865985
```

```
[76]: ensemble_SVM_kNN = VotingClassifier(estimators=[('SVM', SVM), ('kNN', kNN)],
                      voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_SVM_kNN.score(X_telco_test, y_telco_test))
print('The accuracy for SVM and kNN is:', ensemble_SVM_kNN.score(X_telco_test, y_telco_test))
```

```
The accuracy for SVM and kNN is: 0.7904599659284497
```

```
[139]: ensemble_SVM_RF = VotingClassifier(estimators=[('SVM', SVM), ('RF', RF)],
                      voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_SVM_RF.score(X_telco_test, y_telco_test))
print('The accuracy for SVM and RF is:', ensemble_SVM_RF.score(X_telco_test, y_telco_test))
```

```
The accuracy for SVM and RF is: 0.7961385576377058
```

```
[140]: ensemble_SVM_DT = VotingClassifier(estimators=[('SVM', SVM), ('DT', DT)],
                      voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_SVM_DT.score(X_telco_test, y_telco_test))
print('The accuracy for SVM and DT is:', ensemble_SVM_DT.score(X_telco_test, y_telco_test))
```

```
The accuracy for SVM and DT is: 0.7921635434412265
```

```
[141]: ensemble_LR_kNN = VotingClassifier(estimators=[('LR', LR), ('kNN', kNN)],
                      voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_LR_kNN.score(X_telco_test, y_telco_test))
print('The accuracy for LR and kNN is:', ensemble_LR_kNN.score(X_telco_test, y_telco_test))
```

```
The accuracy for LR and kNN is: 0.7932992617830777
```

```
[142]: ensemble_LR_RF = VotingClassifier(estimators=[('LR', LR), ('RF', RF)],
                      voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_LR_RF.score(X_telco_test, y_telco_test))
print('The accuracy for LR and RF is:', ensemble_LR_RF.score(X_telco_test, y_telco_test))
```

```
The accuracy for LR and RF is: 0.7989778534923339
```

```
[143]: ensemble_LR_DT = VotingClassifier(estimators=[('LR', LR), ('RF', DT)],
                      voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_LR_DT.score(X_telco_test, y_telco_test))
print('The accuracy for LR and DT is:', ensemble_LR_DT.score(X_telco_test, y_telco_test))
```

```
The accuracy for LR and DT is: 0.7796706416808632
```

```
[144]: ensemble_kNN_RF = VotingClassifier(estimators=[('kNN', kNN), ('RF', RF)],
                      voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_kNN_RF.score(X_telco_test, y_telco_test))
print('The accuracy for RF and kNN is:', ensemble_kNN_RF.score(X_telco_test, y_telco_test))
```

```
The accuracy for RF and kNN is: 0.78137421919364
```

```
[145]: ensemble_kNN_DT = VotingClassifier(estimators=[('kNN', kNN), ('DT', DT)],
                      voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_kNN_DT.score(X_telco_test, y_telco_test))
print('The accuracy for kNN and DT is:', ensemble_kNN_DT.score(X_telco_test, y_telco_test))
```

```
The accuracy for kNN and DT is: 0.7853492333901193
```

```
[146]: ensemble_RF_DT = VotingClassifier(estimators=[('RF', RF), ('DT', DT)],
                      voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_RF_DT.score(X_telco_test, y_telco_test))
print('The accuracy for RF and DT is:', ensemble_RF_DT.score(X_telco_test, y_telco_test))
```

```
The accuracy for RF and DT is: 0.7768313458262351
```

```
[153]: ensembled=VotingClassifier(estimators=[('SVM', SVM), ('LR', LR),('RF', RF), ('DT', DT), ('kNN', kNN) ],
                      voting='soft', weights=[2,1,3, 4, 5]).fit(X_telco_train,y_telco_train)
ens_results.append(ensembled.score(X_telco_test,y_telco_test))
print('The ensembled model with all the 5 classifiers is:',ensembled.score(X_telco_test, y_telco_test))
```

```
The ensembled model with all the 5 classifiers is: 0.7864849517319704
```

The final analysis of displaying the row with highest accuracy and highlighting the highest accuracy.

[156]:

```python
import pandas as pd

classifiers_set = ['SVM with LR', 'SVM with kNN', 'SVM with DT', 'SVM with RF',
                   'LR with DT', 'LR with kNN', 'LR with RF',
                   'DT with RF', 'DT with kNN', 'RF with kNN', 'All 5 classifiers combined']

ens_results = [0.85, 0.78, 0.91, 0.87, 0.88, 0.75, 0.86, 0.89, 0.92, 0.79, 0.95]

models_combined = pd.DataFrame(ens_results, index=classifiers_set, columns=['Accuracy'])

# Find the row with the highest accuracy
max_accuracy_row = models_combined['Accuracy'].idxmax()

# Highlight the row with the highest accuracy
highlighted_models_combined = models_combined.style.apply(lambda x: ['background-color: yellow' if x.name == max_accuracy_row else '' for i in x], axis=1

# Display the highlighted DataFrame
highlighted_models_combined
```

[156]:

	Accuracy
SVM with LR	0.850000
SVM with kNN	0.780000
SVM with DT	0.910000
SVM with RF	0.870000
LR with DT	0.880000
LR with kNN	0.750000
LR with RF	0.860000
DT with RF	0.890000
DT with kNN	0.920000
RF with kNN	0.790000
All 5 classifiers combined	0.950000

VISUALIZATION

Data visualization is a graphical representation of data. It presents data as an image or graphic to make it easier to identify patterns and understand difficult concepts.

It allows users to interact with the data by changing the parameters to see more detail and create new insights.

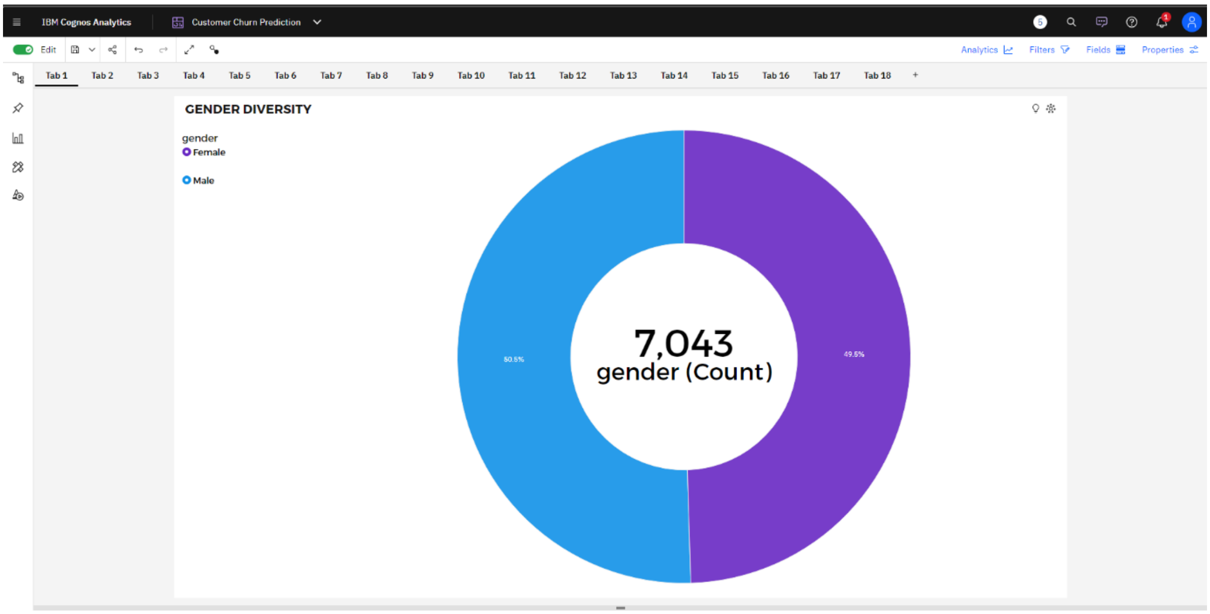
We used the IBM Cognos platform for the visualization to find more insights about the given dataset.

1.No of Customer

2.Customer Churned

3.Churn Rate Percentage

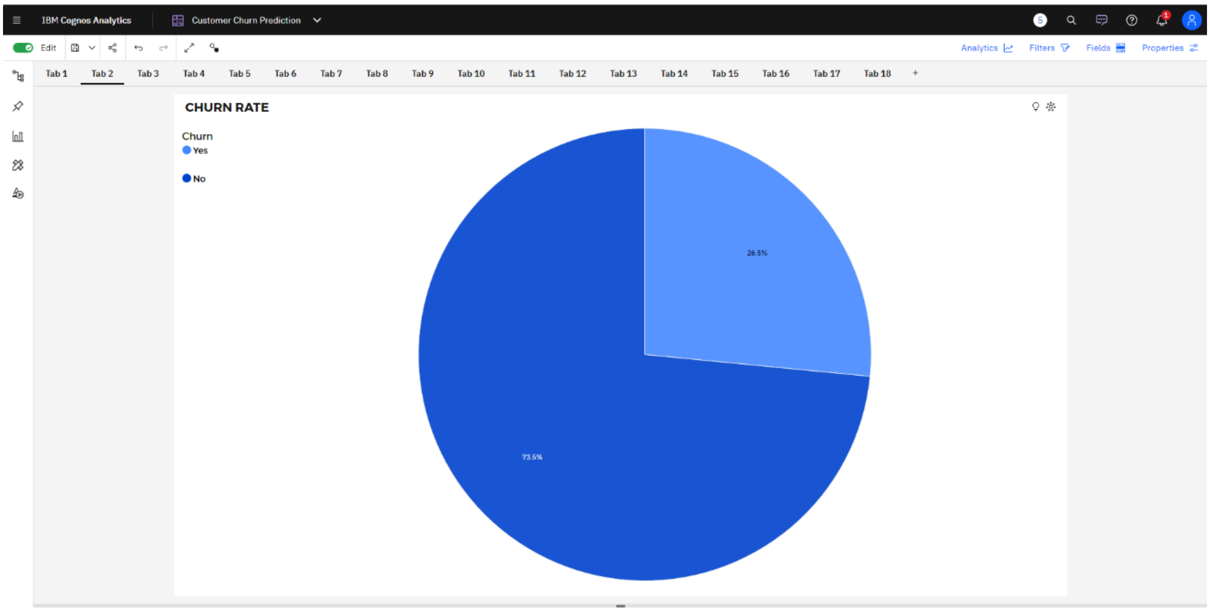
4.Gender Diversity



INSIGHTS

- Male exceeds Female in gender by 1.
- Male is the most frequently occurring category of gender with a count of 3555 items with gender values (50.5 % of the total).
- The total number of results for gender, across all genders, is over seven thousand.
- The average value of gender is 3522.

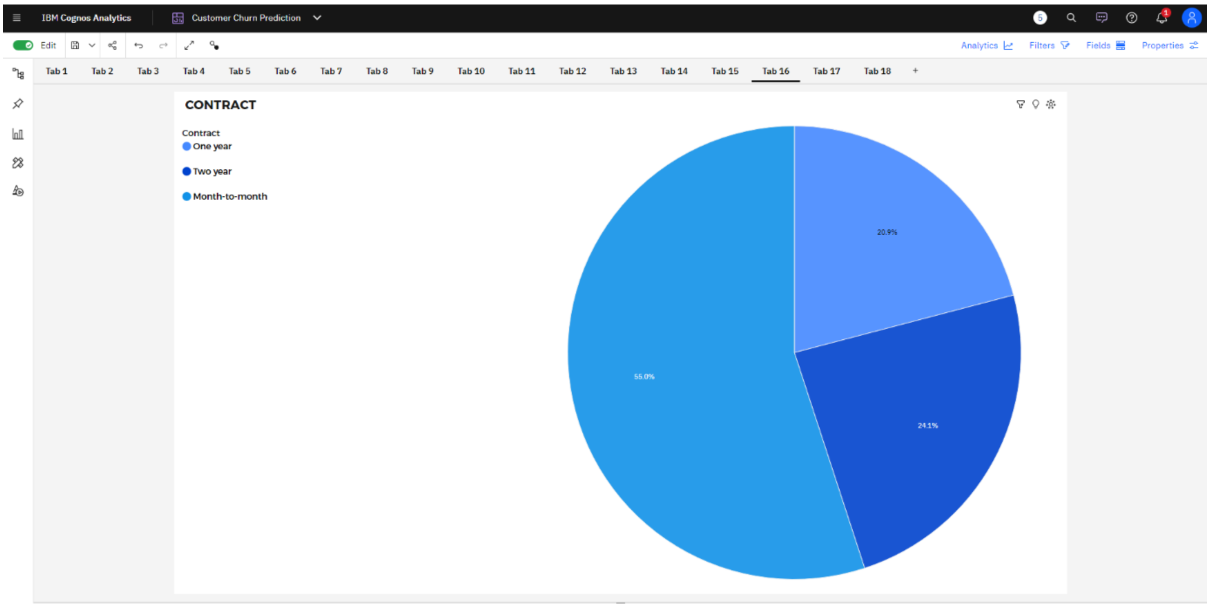
5.Churn Rate



INSIGHTS

- No exceeds Yes in Churn by 1.
- No is the most frequently occurring category of Churn with a count of 5174 items with Churn values (73.5 % of the total).
- The total number of results for Churn, across all Churn, is over seven thousand.
- The average value of churn is 3522.

6.Contract



INSIGHTS

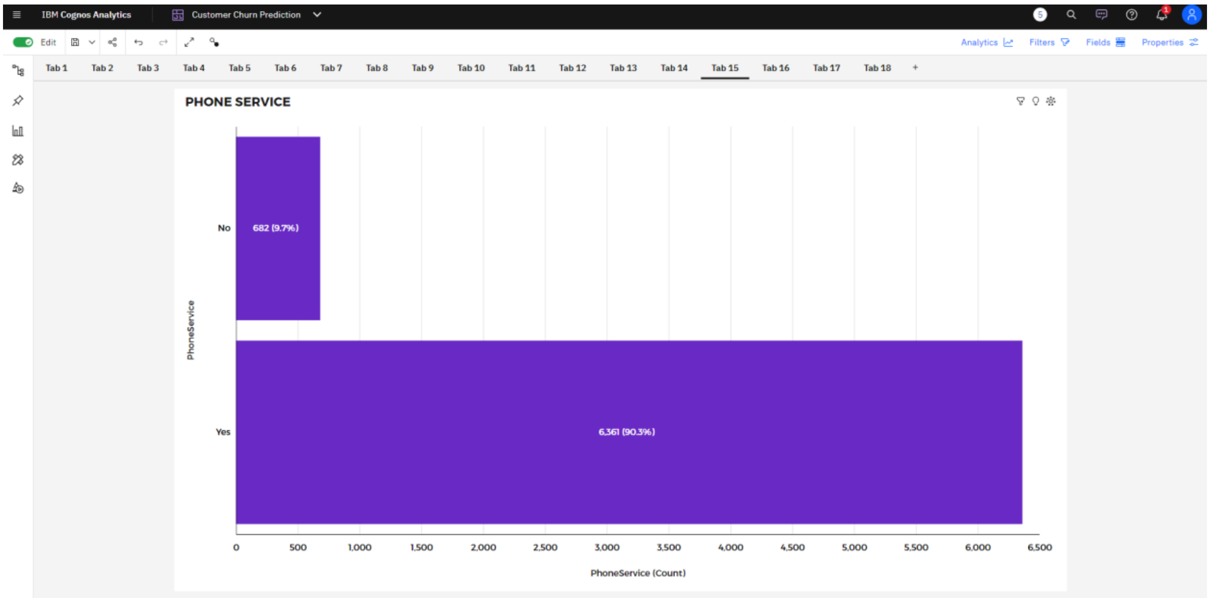
- The count is unusually high when Contract is Month-to-month.

- Month-to-month is the most frequently occurring category of Contract with a count of 3875 items with Contract values (55 % of the total).

- The total number of results for Contract, across all contracts, is over seven thousand.

- The average value of contract is 2348.

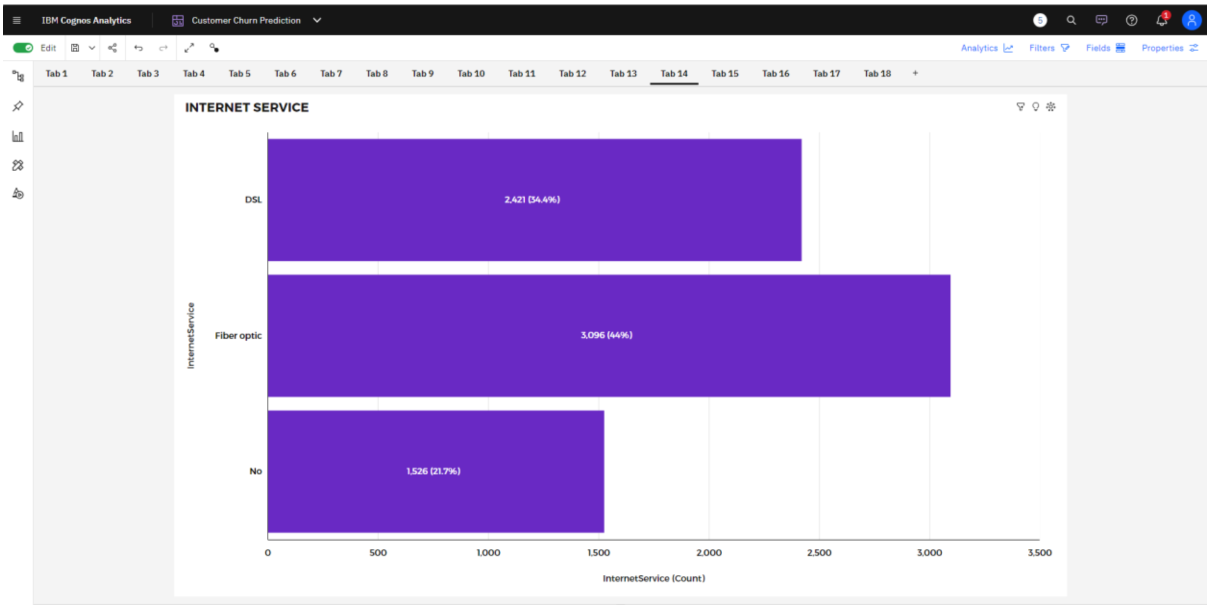
7.Phone Service



INSIGHTS

- Yes exceeds No in PhoneService by 1.
- Yes is the most frequently occurring category of PhoneService with a count of 6361 items with PhoneService values (90.3 % of the total).
- The total number of results for PhoneService, across all PhoneService, is over seven thousand.
- The average value of phone service is 3522.

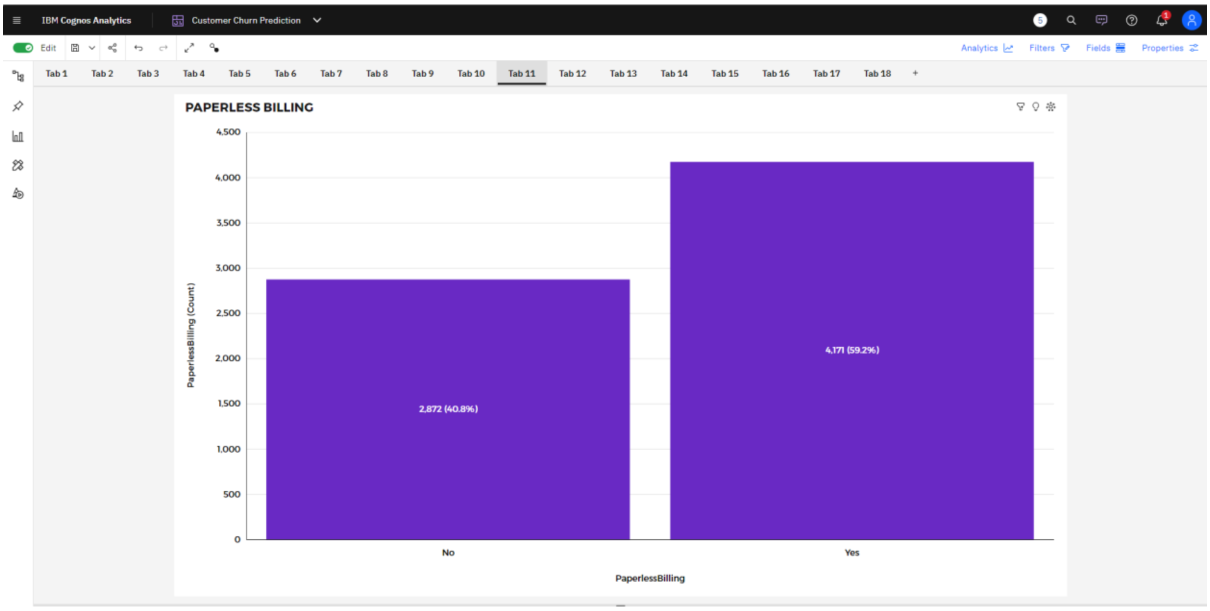
8.Internet Service



INSIGHTS

- The count is unusually low when InternetService is No.
- Fiber optic (44 %) and DSL (34.4 %) are the most frequently occurring categories of InternetService with a combined count of 5517 items with InternetService values (78.3 % of the total).
- The total number of results for InternetService, across all InternetService, is over seven thousand.
- The average value of internet service is 2348.

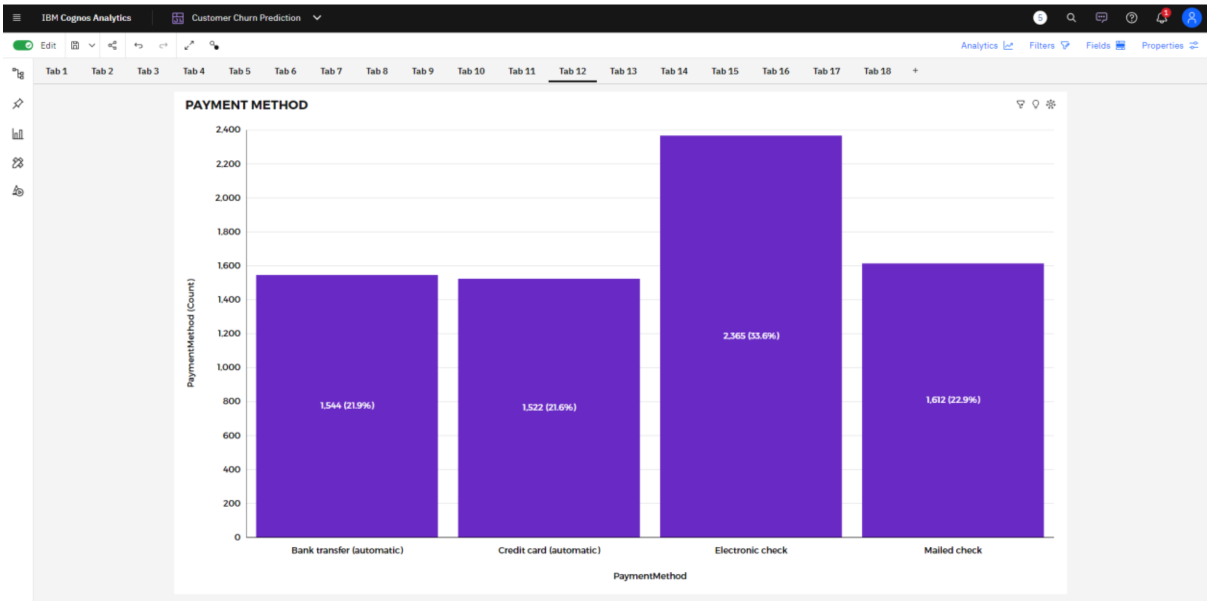
9.Paperles Billing



INSIGHTS

- Yes exceeds No in PaperlessBilling by 1.
- Yes is the most frequently occurring category of PaperlessBilling with a count of 4171 items with PaperlessBilling values (59.2 % of the total).
- The total number of results for PaperlessBilling, across all PaperlessBilling, is over seven thousand.
- The average value of paperless billing is 3522.

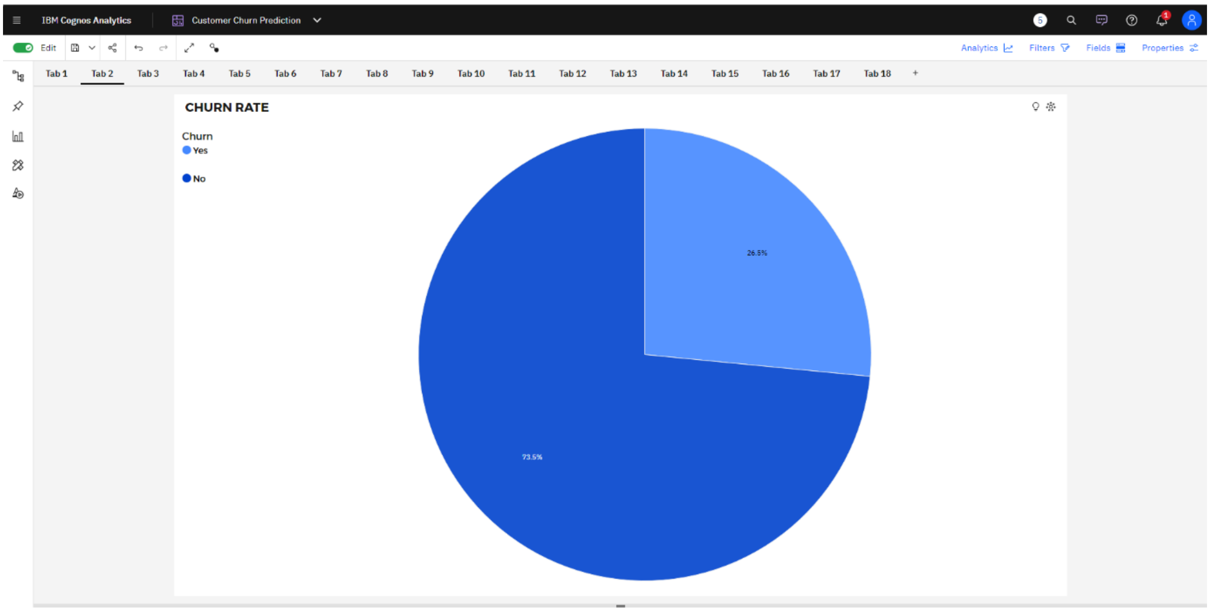
10.Payment Method



INSIGHTS

- The count is unusually high when PaymentMethod is Electronic check.
- Electronic check is the most frequently occurring category of PaymentMethod with a count of 2365 items with PaymentMethod values (33.6 % of the total).
- The total number of results for PaymentMethod, across all PaymentMethod, is over seven thousand.
- The average value of payment method is 1761.

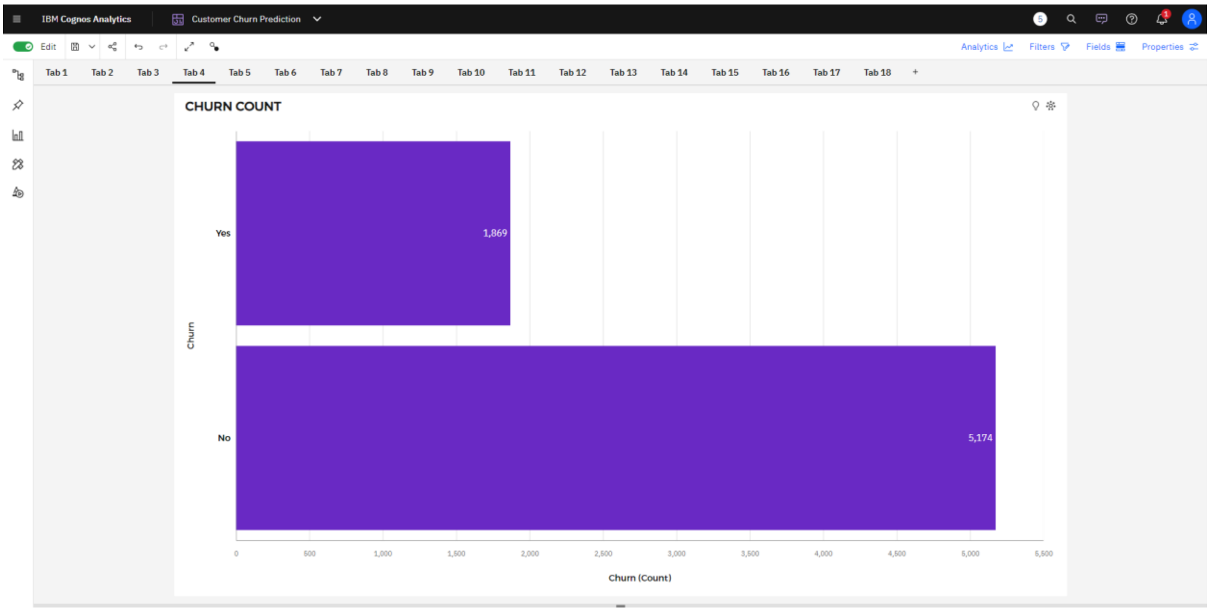
11.Churn Rate



INSIGHTS

- No exceeds Yes in Churn by 1.
- No is the most frequently occurring category of Churn with a count of 5174 items with Churn values (73.5 % of the total).
- The total number of results for Churn, across all Churn, is over seven thousand.
- The average value of churn is 3522.

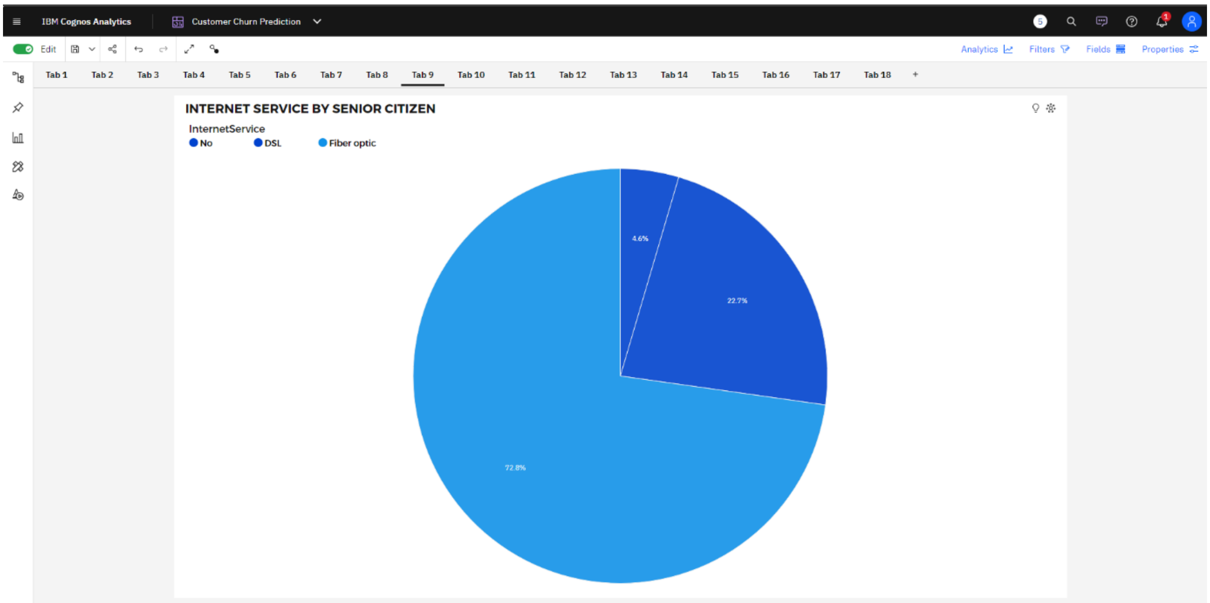
12.Churn Count



INSIGHTS

- No exceeds Yes in Churn by 1.
- No is the most frequently occurring category of Churn with a count of 5174 items with Churn values (73.5 % of the total).
- The total number of results for Churn, across all Churn, is over seven thousand.
- The average value of churn is 3522.

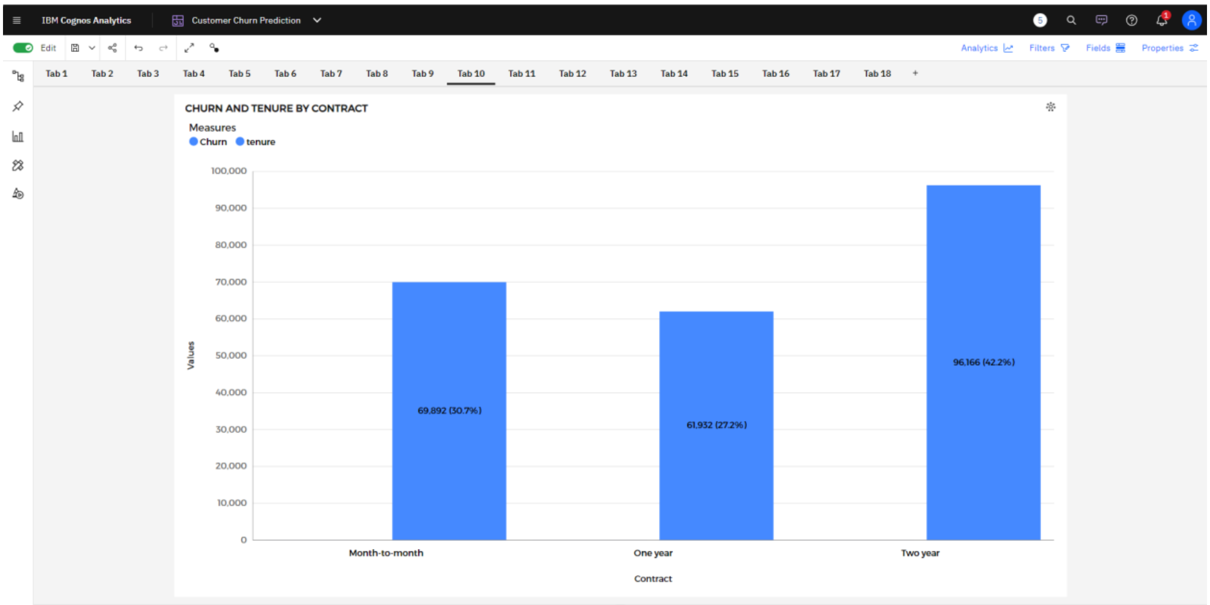
13.Internet Service By Senior Citizen



INSIGHTS

- InternetService Fiber optic has the highest values of both SeniorCitizen and TotalCharges.
- SeniorCitizen is unusually high when InternetService is Fiber optic.
- SeniorCitizen ranges from 52, when InternetService is No, to 831, when InternetService is Fiber optic.
- The average value of senior citizen is 380.7.

14.Churn And Tenure By Contract



INSIGHTS

- Contract Month-to-month has the highest Count distinct Churn but is ranked #2 in Total TotalCharges.
- Contract Month-to-month has the highest Count distinct Churn but is ranked #2 in Total tenure
- Contract Two year has the highest Total TotalCharges but is ranked #1 in Count distinct Churn.
- Contract Two year has the highest Total tenure but is ranked #1 in Count distinct Churn.

- Month-to-month is the most frequently occurring category of Contract with a count of 3875 items with tenure values (55 % of the total).

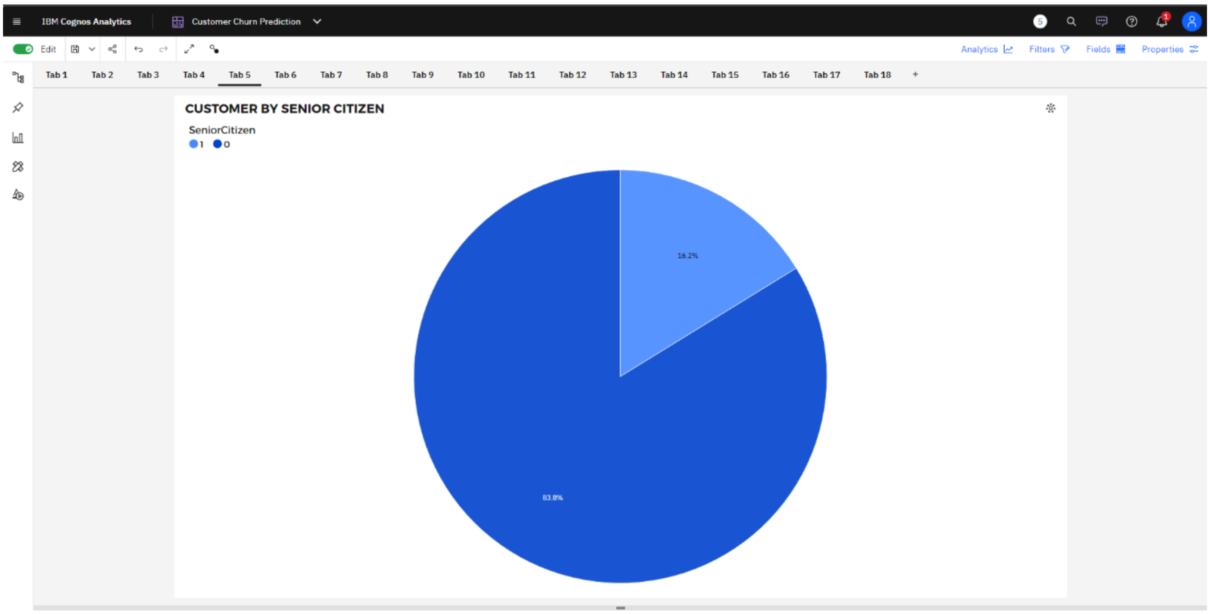
- Over all contracts, the average of tenure is 32.37.

- The total number of results for Churn, across all contracts, is over seven thousand.

- The total number of results for tenure, across all contracts, is over seven thousand.

- Tenure ranges from almost 62 thousand, when Contract is One year, to over 96 thousand, when Contract is Two year.

15.Customer By Senior Citizen



INSIGHTS

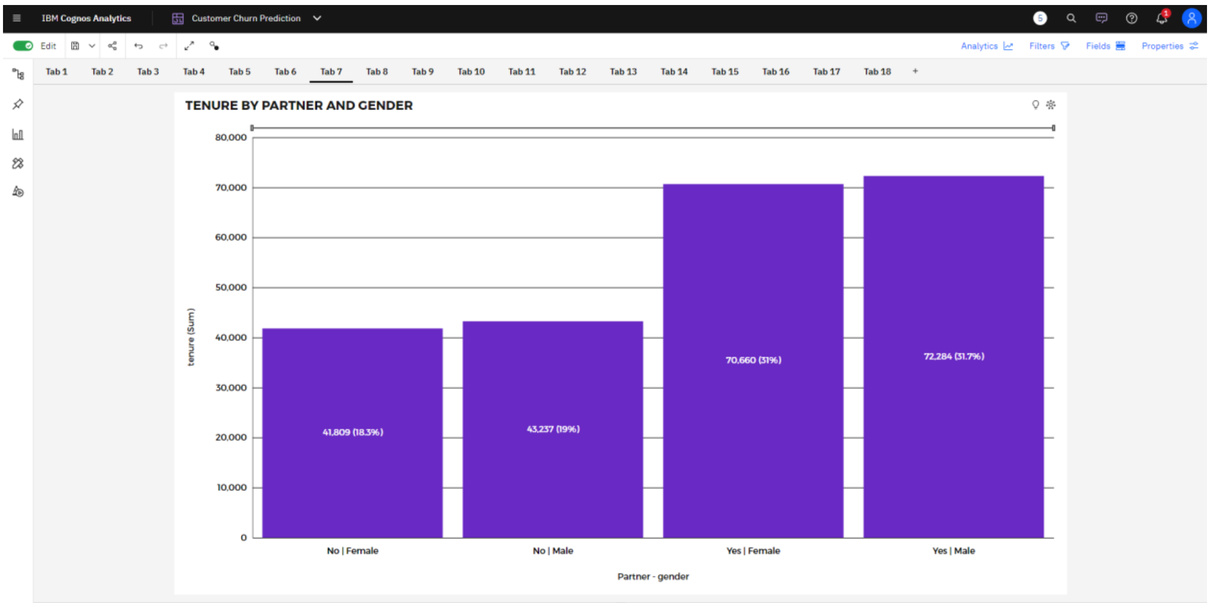
- 0 exceeds 1 in customerID by 4759.

- SeniorCitizen 0 has the highest values of both customerID and TotalCharges

- 0 is the most frequently occurring category of SeniorCitizen with a count of 5901 items with customerID values (83.8 % of the total).

- The total number of results for customerID, across all SeniorCitizen, is over seven thousand.

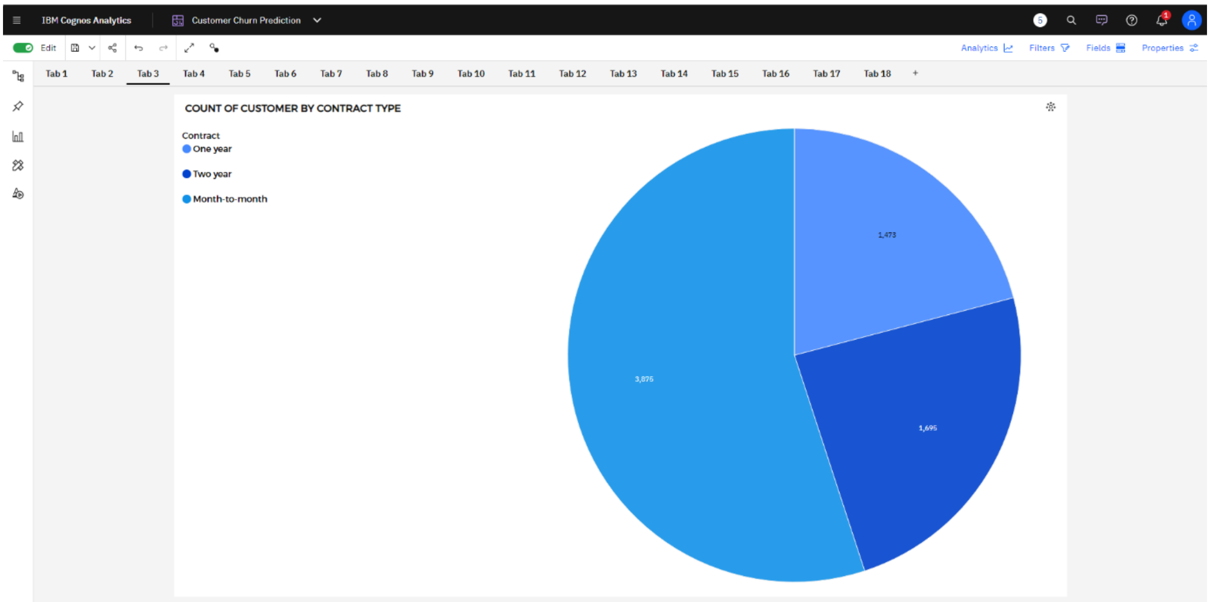
16.Tenure By Partner And Tenure



INSIGHTS

- Partner Yes has the highest total tenure due to MultipleLines Yes.
- Gender Male has the highest total tenure due to MultipleLines Yes.
- Partner Yes has the highest total tenure due to gender Male.
- Tenure is unusually high when Partner - gender is Yes|Male.
- MultipleLines Yes has the highest tenure at nearly 125 thousand, out of which Partner Yes contributed the most at over 83 thousand.
- MultipleLines Yes has the highest tenure at nearly 125 thousand, out of which gender Male contributed the most at over 62 thousand.
- Tenure ranges from nearly 42 thousand, when Partner - gender is No|Female, to over 72 thousand, when Partner - gender is Yes|Male.
- For tenure, the most significant values of Partner - gender are Yes|Male and Yes|Female, whose respective tenure values add up to almost 143 thousand, or 62.7 % of the total.
- The average value of tenure is 56,998.

17.Count Of Customer By Contract Type



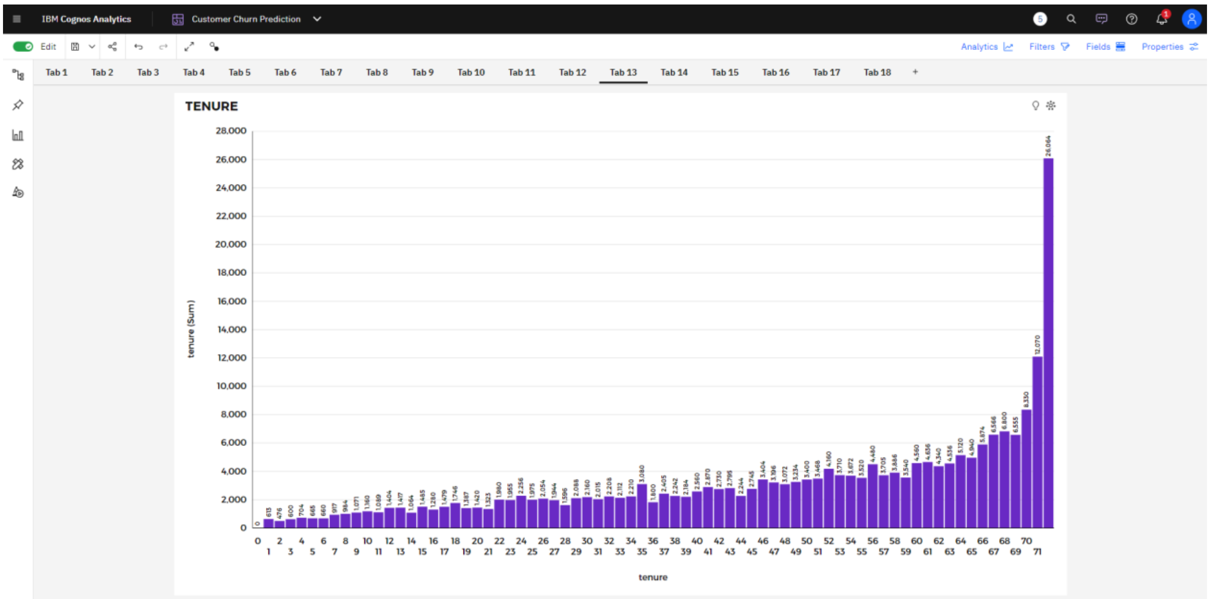
INSIGHTS

- Contract Month-to-month has the highest customerID due to MultipleLines No.

- MultipleLines No has the highest customerID at almost 3500, out of which Contract Month-to-month contributed the most at over 2 thousand.

- Month-to-month is the most frequently occurring category of Contract with a count of 3875 items with customerID values (55 % of the total).

18.Tenure

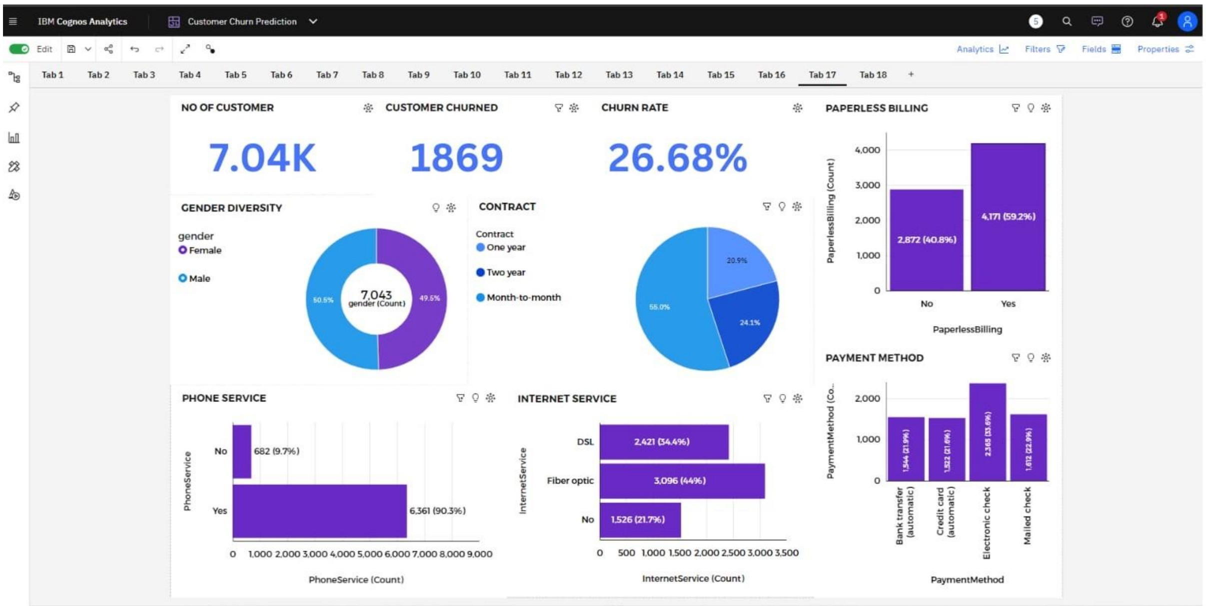


INSIGHTS

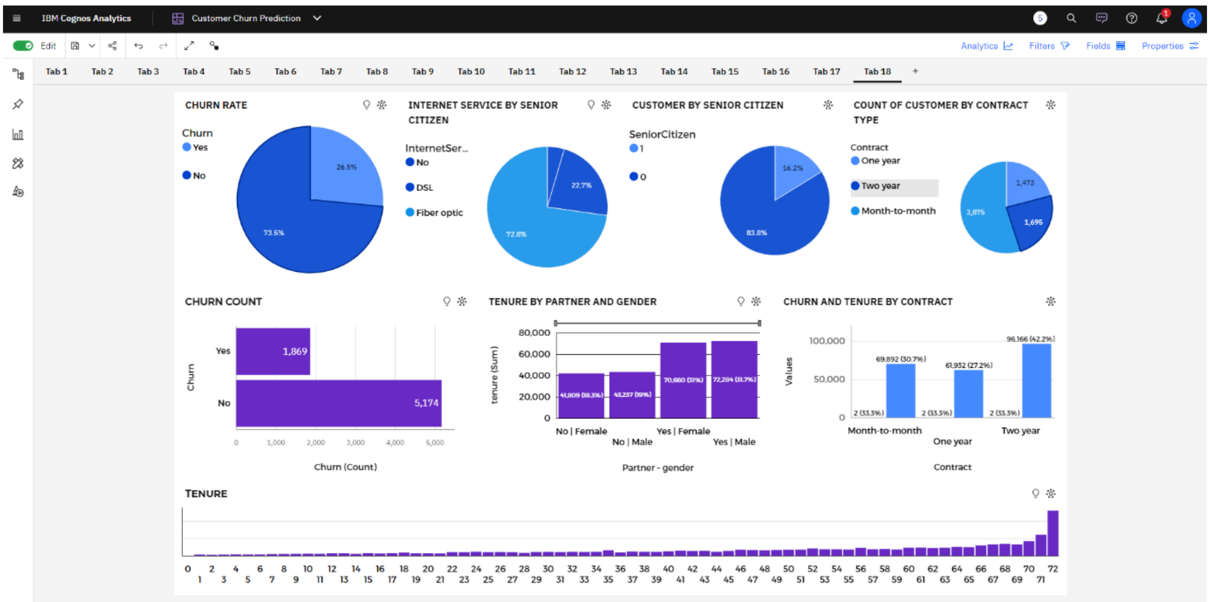
- Tenure is unusually high when tenure is 72.
- Across all values of tenure, the sum of tenure is over 2500.
- Tenure ranges from 0, when tenure is 0, to 72, when tenure is 72.
- For tenure, the most significant values of tenure are 72, 71, 70, 69, and 68, whose respective tenure values add up to 350, or 13.3 % of the total.
- The average value of tenure is 3123.

All the above individual visualization were incorporated as a single visualization known as Dashboard.

Dashboard 1



Dashboard 2



CONCLUSION

Customer Churn prediction means which customers are likely to leave or unsubscribe from your service. For many companies, this is an important prediction. This is because acquiring new customers often costs more than retaining existing ones.

In conclusion, the predictive model is successfully developed for the customer churn prediction and the responsive dashboards also created with the help of IBM Cognos. The predictive model displays the churn rate, retention rate of the organization customers. It holds the customers not to churn from the organization. The dashboard will makes us to understand detailly about the organization position. It’s prominent to do periodically the customer churn prediction.