

REPORT

**CUSTOMER CHURN PREDICTION FOR A
TELECOM COMPANY**

BY → Arijeet Goswami

ACKNOWLEDGMENTS

The Internship opportunity that I had with Pratinik Infotech was a great change for learning and understanding the intricacies of the subject of Data-visualization, Machine-learning in Data Science; and also, for personal as well as professional development. I am very obliged to have a chance to interact with so many professionals who guided me throughout the traineeship project and made it a great learning curve for me. Firstly, I express my deepest gratitude and special thanks to the Development Team of Pratinik Infotech who allowed me to carry out my internship at their esteemed organization. Also, I express my thanks to the team for making me understand the details of the Data Science profile and training me in the same so that I can carry out the project properly and with maximum client satisfaction and also for spending his valuable time despite his busy schedule. I would also like to thank the team of Pratinik Infotech and my colleagues who made the working environment productive and very conducive.

TABLE OF CONTENT

Acknowledgments i

Abstract iii

Sr. No.	Topic	Page No.
1	Introduction	
	1.1 About The Project	
	1.2 Objective And Deliverables	
2	Methodology	
	2.1 Language and Platform Used	
3	Implementation	
	3.1	
	3.2	
	3.3	
	3.4	
	3.5	
	3.6	
	3.7	
	3.8	

INTRODUCTION

1.1 About the Project:

This document outlines a project focused on predicting customer churn within a telecom company. The primary goal is to identify customers who are likely to discontinue their services, enabling the company to implement proactive retention strategies. This project addresses this challenge by developing a predictive model that can accurately forecast which customers are at high risk of churning. By leveraging historical customer data, the project aims to uncover patterns and indicators associated with churn behavior.

1.2 Objective And Deliverables:

Objectives

The key objectives of this customer churn prediction project are:

1. Data Collection and Preparation:

- To gather relevant historical customer data, including demographic information, service usage patterns, billing details, contract types, and customer service interactions.
- To clean and preprocess the raw data, handling missing values, outliers, and inconsistencies to ensure data quality and suitability for modeling.
- To perform necessary feature engineering to create new variables that might improve model performance (e.g., tenure, average monthly charges, data usage per month).

2. Exploratory Data Analysis (EDA):

- To conduct comprehensive EDA to understand the underlying patterns, distributions, and correlations within the dataset.
- To identify key factors and attributes that are most strongly associated with customer churn.
- To visualize relationships between different features and the target variable (churn) to gain actionable insights.

3. Model Development and Evaluation:

- To develop predictive models using **Logistic Regression** to estimate the probability of a customer churning.
- To develop predictive models using **Support Vector Machines (SVM)**, a powerful algorithm for classification tasks, to identify churners.
- To train and validate both models using appropriate cross-validation techniques to ensure generalization ability.
- To evaluate the performance of the models using various metrics such as accuracy, precision, recall, F1-score, and AUC-ROC curve.
- To compare the performance of Logistic Regression and SVM models to determine which algorithm provides better predictive power for this specific dataset.

4. Identification of Key Churn Drivers:

- To identify the most influential features or factors that contribute significantly to customer churn based on the insights from the developed models. This will help the telecom company understand *why* customers are churning.

Deliverables

The successful completion of this project will result in the following deliverables:

1. **Cleaned and Preprocessed Dataset:** A well-structured and cleaned dataset ready for machine learning model training, along with documentation of the preprocessing steps.
2. **Exploratory Data Analysis Report:** A detailed report summarizing key findings from the EDA, including visualizations, statistical summaries, and identified churn drivers.
3. **Developed Machine Learning Models:**
 - A trained **Logistic Regression model** capable of predicting customer churn probabilities.
 - A trained **Support Vector Machine (SVM) model** for churn classification.
 - The model files and associated code for deployment.

4. Model Performance Evaluation Report:

A comprehensive report detailing the performance metrics of both Logistic Regression and SVM models, including comparative analysis and justification for the chosen best-performing model.

5. Insights and Recommendations: A document outlining the key factors contributing to churn, derived from model interpretability. This will include actionable recommendations for the telecom company to design effective customer retention strategies, such as targeted promotions, improved customer service protocols, or tailored product offerings for at-risk customers.

6. Codebase: A well-commented and organized codebase containing all the scripts for data loading, preprocessing, model training, evaluation, and visualization.

METHODOLOGY

2.1.1 Language and Platform Used:

Colab:

python Colab stands out as a hassle-free Jupyter Notebook service, offering seamless accessibility without the need for setup. It grants users complimentary access to robust computing resources, including GPUs and TPUs. This platform is particularly tailored to cater to the needs of machine learning, data science, and educational endeavors. Moreover, you have the opportunity to enhance the aesthetic appeal of your Python project notebook by incorporating mathematical equations, graphs, tables, images, and various other graphics. Furthermore, Colab empowers you to seamlessly code data visualizations in Python, with the platform rendering the code into visually engaging assets.

Major features are:

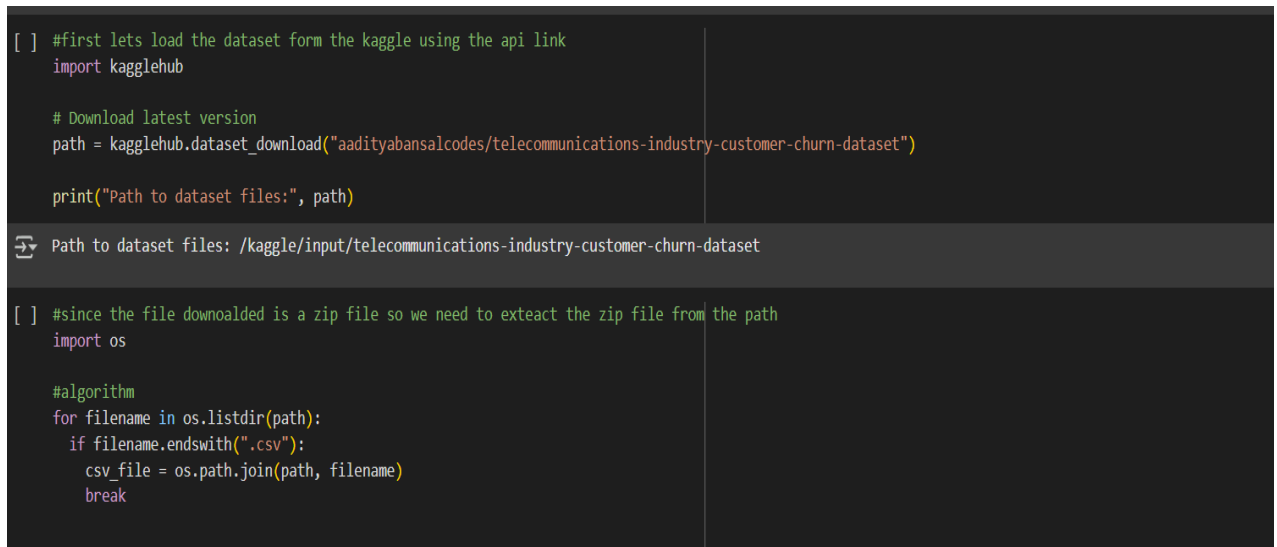
- Google Colab provides pre-installed libraries so that you can import the required library for code usage. Libraries include NumPy, Pandas, Matplotlib, PyTorch, TensorFlow more ML libraries.
- You can import data directly from your local machine, seamlessly mount Google Drive to a Colab instance, fetch remote data, or clone GitHub repositories directly into Colab, ensuring convenience and flexibility in data sourcing.
- We can also train models on audio and text
- Provide popular and pre-installed for data-science

Python:

Python is a high-level, interpreted programming language known for its readability and simplicity. Its clear syntax and extensive libraries make it an excellent choice for beginners. Python supports multiple programming paradigms, including object-oriented, imperative, and functional programming. It's widely used in web development, data science, artificial intelligence, and automation. The language's vast and active community contributes to its continuous development and robust ecosystem.

IMPLEMENTATION

3.1 Obtain and Review Data



```
[ ] #first lets load the dataset form the kaggle using the api link
import kagglehub

# Download latest version
path = kagglehub.dataset_download("aadiyabansalcodes/telecommunications-industry-customer-churn-dataset")

print("Path to dataset files:", path)

Path to dataset files: /kaggle/input/telecommunications-industry-customer-churn-dataset

[ ] #since the file downloalded is a zip file so we need to exteact the zip file from the path
import os

#algorithm
for filename in os.listdir(path):
    if filename.endswith(".csv"):
        csv_file = os.path.join(path, filename)
        break
```

Figure 1: loading of data-set from Kaggle

Data Acquisition from Kaggle

This image shows the initial steps to acquire the dataset for your project.

- **Downloading the dataset:** You're using the kagglehub library to download the "telecommunications-industry-customer-churn-dataset" from Kaggle. This is a standard and efficient way to get publicly available datasets.
- **Path to dataset:** The output confirms that the dataset has been downloaded to /kaggle/input/telecommunications-industry-customer-churn-dataset.
- **Extracting the CSV:** Since Kaggle datasets are often downloaded as zip files, you're then using the os library to list the contents of the downloaded directory and extract the .csv file. This ensures you have the raw data ready for loading.

3.2 LOADING AND DEPENDENCIES AND LIBRARAY

```
#loading dependencies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt    #data vizualization libraries
import seaborn as sns
from sklearn.preprocessing import LabelEncoder    #provides vizulization of categorical features
from imblearn.over_sampling import SMOTE    #for class imbalance in the target column    #oversampling technique --> to uniformly build taget class
from sklearn.model_selection import train_test_split, cross_val_score
```

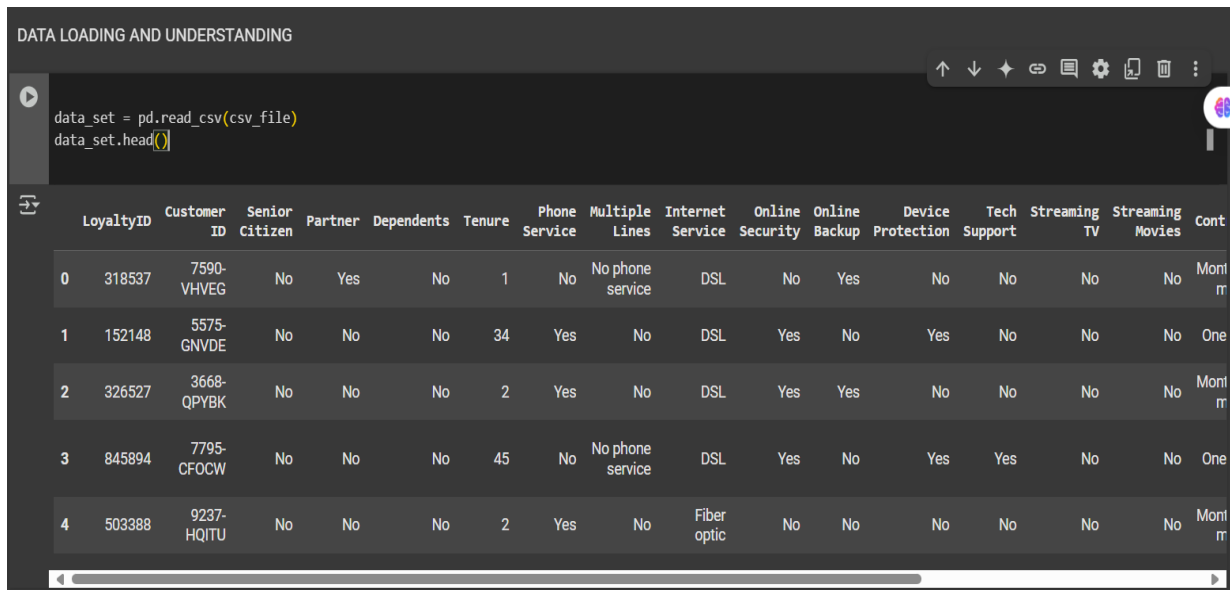
Figure 2: Libraries and dependencies

Loading Dependencies

This image highlights the essential libraries you're importing for your data analysis and machine learning tasks.

- **pandas as pd:** This is crucial for data manipulation and analysis, primarily for working with DataFrames.
- **numpy as np:** Used for numerical operations, especially array manipulations.
- **matplotlib.pyplot as plt and seaborn as sns:** These are your go-to libraries for data visualization, allowing you to create various plots and charts to understand your data better.
- **sklearn.preprocessing.LabelEncoder:** This will likely be used to convert categorical text data into numerical labels, which is a common preprocessing step for machine learning models.
- **imblearn.over_sampling.SMOTE:** This indicates you're aware of potential class imbalance in your target variable (churn). SMOTE (Synthetic Minority Over-sampling Technique) is a popular technique to address this by generating synthetic samples for the minority class, aiming to uniformly balance the target classes.
- **sklearn.model_selection.train_test_split, cross_val_score:** These are fundamental for splitting your data into training and testing sets (to evaluate your model's performance on unseen data) and for performing cross-validation (a more robust way to evaluate model performance).

3.3 DATA_SET OVERVIEW:



The screenshot shows a Jupyter Notebook interface with the title "DATA LOADING AND UNDERSTANDING". The code cell contains the following Python code:

```
data_set = pd.read_csv(csv_file)
data_set.head()
```

The output of the code is a preview of the first 5 rows of the DataFrame. The columns are: LoyaltyID, Customer ID, Senior Citizen, Partner, Dependents, Tenure, Phone Service, Multiple Lines, Internet Service, Online Security, Online Backup, Device Protection, Tech Support, Streaming TV, Streaming Movies, and Churn. The rows are indexed 0 to 4.

	LoyaltyID	Customer ID	Senior Citizen	Partner	Dependents	Tenure	Phone Service	Multiple Lines	Internet Service	Online Security	Online Backup	Device Protection	Tech Support	Streaming TV	Streaming Movies	Churn
0	318537	7590-VHVEG	No	Yes	No	1	No	No phone service	DSL	No	Yes	No	No	No	No	Monthly
1	152148	5575-GNVDE	No	No	No	34	Yes	No	DSL	Yes	No	Yes	No	No	No	One
2	326527	3668-QPYBK	No	No	No	2	Yes	No	DSL	Yes	Yes	No	No	No	No	Monthly
3	845894	7795-CFOCW	No	No	No	45	No	No phone service	DSL	Yes	No	Yes	Yes	No	No	One
4	503388	9237-HQITU	No	No	No	2	Yes	No	Fiber optic	No	No	No	No	No	No	Monthly

Figure 3: DATA_SET

Here, you're loading your extracted CSV file into a pandas DataFrame and viewing the first few rows.

- **pd.read_csv(csv_file):** This command reads the data from the identified CSV file into a pandas DataFrame named data_set.
- **data_set.head():** This displays the first 5 rows of your DataFrame, giving you a quick overview of the columns and the type of data they contain. From this output, we can see columns like LoyaltyID, Customer ID, Senior Citizen, Tenure, Phone Service, Internet Service, Monthly Charges, Total Charges, and Churn, among others. This immediately tells us what kind of features you'll be working with to predict churn.

```

#INITIAL INSPECTION FOR THE DATA
#SHAPE DETERMINING
data_shape = data_set.shape
print(f"the number of rows and column is {data_shape}")

...

#displaying max clouns together
data_col = pd.set_option("display.max_columns",None)
print(f"the columns are {data_set.head()}")
...

#for seeing the types of data as well knowing wetehr the data is null or not
data_information = data_set.info()
print(f"the data_set information is {data_information}")

```

Figure 4: overview of data-set utilized

These images show you performing essential initial inspections of your dataset.

- **data_set.shape:** You're determining the number of rows and columns in your DataFrame. The output clearly states "the number of rows and column is (7043, 21)", meaning your dataset has 7043 observations (customers) and 21 features.
- **pd.set_option("display.max_columns", None):** This is a good practice to ensure that when you display the head of the DataFrame, all columns are shown and not truncated, which is helpful for comprehensive viewing.
- **data_set.info():** This is a very important step. It provides a concise summary of your DataFrame, including:
 - **Column names:** All 21 column names are listed.
 - **Non-Null Count:** This tells you if there are any missing values. In this case, all columns show "7043 non-null", indicating no missing values (which is great!).
 - **Dtype:** This shows the data type of each column. You have float64 for Monthly Charges, int64 for LoyaltyID, Tenure, and Customer ID, and object for most other categorical columns.
 - **Memory Usage:** Shows the memory footprint of your DataFrame.
- **"the data_set information is None":** This output seems to be an anomaly in the printing. Data_set.info() itself prints the information directly to the console and returns None. So, while the information is displayed, attempting to print(f"the data_set information is {data_information}") where data_information holds the return value of data_set.info() would indeed print None. The actual info() output (the table above "the data_set information is None") is what's important here.

```
the number of rows and column is(7043, 21)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
#   Column                Non-Null Count  Dtype
---  -
0   LoyaltyID              7043 non-null   int64
1   Customer ID            7043 non-null   object
2   Senior Citizen          7043 non-null   object
3   Partner                 7043 non-null   object
4   Dependents              7043 non-null   object
5   Tenure                  7043 non-null   int64
6   Phone Service           7043 non-null   object
7   Multiple Lines          7043 non-null   object
8   Internet Service        7043 non-null   object
9   Online Security         7043 non-null   object
10  Online Backup           7043 non-null   object
11  Device Protection       7043 non-null   object
12  Tech Support            7043 non-null   object
13  Streaming TV            7043 non-null   object
14  Streaming Movies        7043 non-null   object
15  Contract                7043 non-null   object
16  Paperless Billing        7043 non-null   object
17  Payment Method          7043 non-null   object
18  Monthly Charges         7043 non-null   float64
19  Total Charges           7043 non-null   object
20  Churn                   7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
the data_set information is None
```

Figure 5: brief description for the dataset

Dataset Summary

- **Shape of DataFrame:**
 - **Rows:** 7,043
 - **Columns:** 21
- **Type:** <class 'pandas.core.frame.DataFrame'>
- **RangeIndex:** 0 to 7042

Data Types Overview

- **Numerical (int64):** LoyaltyID, Tenure
- **Float (float64):** Monthly Charges
- **Categorical (object):** 18 columns (e.g., Partner, Internet Service, Churn, etc.)

The image displays the summary output of a Pandas DataFrame, likely from a telecom dataset used to analyze customer behavior and churn patterns. This summary, generated using the `.info()` method in Python, reveals that the dataset contains **7,043 rows and 21 columns**, indicating a fairly large sample size for analysis. Each column in the DataFrame represents a specific attribute or feature related to a customer, such as their tenure with the company, the services they use, payment method, and whether they have churned.

The data types vary across the columns: there are **two integer columns** (LoyaltyID and Tenure), **one float column** (Monthly Charges), and **18 object-type columns**, which typically represent categorical or string data such as Customer ID, Internet Service, or Churn. All the columns are complete, with no missing values, as evidenced by each having 7043 non-null entries

```
data_set_col = pd.set_option("display.max_columns",None)
print(f"the cols of data are {data_set.head()}")
```

the cols of data are	LoyaltyID	Customer ID	Senior Citizen	Partner	Dependents	Tenure	\
0	318537	7590-VHVEG	No	Yes	No	1	
1	152148	5575-GMVDE	No	No	No	34	
2	326527	3668-QPYBK	No	No	No	2	
3	845894	7795-CFOCW	No	No	No	45	
4	503388	9237-HQITU	No	No	No	2	
Phone Service	Multiple Lines	Internet Service	Online Security	\			
0	No	No phone service	DSL	No			
1	Yes	No	DSL	Yes			
2	Yes	No	DSL	Yes			
3	No	No phone service	DSL	Yes			
4	Yes	No	Fiber optic	No			
Online Backup	Device Protection	Tech Support	Streaming TV	Streaming Movies	\		
0	Yes	No	No	No	No		
1	No	Yes	No	No	No		
2	Yes	No	No	No	No		
3	No	Yes	Yes	No	No		
4	No	No	No	No	No		
Contract	Paperless Billing	Payment Method	\				
0	Month-to-month	Yes	Electronic check				
1	One year	No	Mailed check				
2	Month-to-month	Yes	Mailed check				
3	One year	No	Bank transfer (automatic)				
4	Month-to-month	Yes	Electronic check				
Monthly Charges	Total Charges	Churn					
0	29.85	29.85	No				
1	56.95	1889.5	No				
2	53.85	108.15	Yes				
3	42.30	1840.75	No				
4	70.70	151.65	Yes				

Figure 6: Displaying All Columns (Head)

Displaying All Columns (Head)

Similar to the previous display, this image reaffirms the use of `pd.set_option("display.max_columns", None)` to ensure all columns are visible when you view the head of your DataFrame. It provides a more complete visual representation of the different features in your dataset, across multiple lines, making it easier to grasp the breadth of information available.

```
[ ] #displaying the columns
data_columns = data_set.columns
print(f"the columns of data sets are {data_columns}")
```

```
the columns of data sets are Index(['LoyaltyID', 'Customer ID', 'Senior Citizen', 'Partner', 'Dependents',
'LoyaltyID', 'Customer ID', 'Senior Citizen', 'Partner', 'Dependents',
'Tenure', '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'],
dtype='object')
```

Figure 7: Displaying All Columns Names from the data_set

This image displays the retrieval of all column names using `data_set.columns`, which returns an Index of the DataFrame's 21 labels. It helps confirm the exact features you'll work with in your churn prediction model. This step is essential for efficient data cleaning, transformation, and feature selection.

3.4 DATA CLEANING:

cleaning of the data set

```
[ ] #dropping of the columns
#dropping the customer_id columns as not required for modelling
data_set = data_set.drop('Customer ID',axis=1)
print(f"the updated data_set columns is {data_set.columns}")

the updated data_set columns is Index(['LoyaltyID', 'Senior Citizen', 'Partner', 'Dependents', 'Tenure',
    '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'],
    dtype='object')
```

Figure 8: Dropping of Columns:

- The Customer ID column was dropped from the data_set as it was not required for modeling.
- The updated columns are: LoyaltyID, Senior Citizen, Partner, Dependents, Tenure, 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.

```
#lets see the unique values to determine wether the columns are catgorical or numerical
for col in data_set.columns:
    unique_values = data_set[col].unique()
    print(f"the unique values of {col} is {unique_values}")
    print('*'*50)
```

Figure 9: Examining Unique Values:

- The unique values for each column were printed to determine if they are categorical or numerical.
 - Binary/Categorical: Senior Citizen (No, Yes), Partner (No, Yes), Dependents (No, Yes), Phone Service (No, Yes), Multiple Lines (No phone service, No, Yes), Internet Service (DSL, Fiber optic, No), Online Security (No, Yes, 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), Paperless Billing (Yes, No), Churn (No, Yes).
 - Categorical (multi-level): Contract (Month-to-month, One year, Two year), Payment Method (Electronic check, Mailed check, Bank transfer (automatic), credit card (automatic)).

```

the unique values of LoyaltyID is [318537 152148 326527 ... 155157 731782 353947]
the unique values of Senior Citizen is ['No' 'Yes']
the unique values of Partner is ['Yes' 'No']
the unique values of Dependents is ['No' 'Yes']
the unique values of Tenure is [ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
 5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26  0
39]
the unique values of Phone Service is ['No' 'Yes']
the unique values of Multiple Lines is ['No phone service' 'No' 'Yes']
the unique values of Internet Service is ['DSL' 'Fiber optic' 'No']
the unique values of Online Security is ['No' 'Yes' 'No internet service']
the unique values of Online Backup is ['Yes' 'No' 'No internet service']
the unique values of Device Protection is ['No' 'Yes' 'No internet service']
the unique values of Tech Support is ['No' 'Yes' 'No internet service']
the unique values of Streaming TV is ['No' 'Yes' 'No internet service']
the unique values of Streaming Movies is ['No' 'Yes' 'No internet service']
the unique values of Contract is ['Month-to-month' 'One year' 'Two year']
the unique values of Paperless Billing is ['Yes' 'No']
the unique values of Payment Method is ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
'Credit card (automatic)']

```

Figure 10: Displaying of Unique Values:

- Numerical/Potentially Numerical: LoyaltyID (unique identifiers), Tenure (integer values ranging from 0 to 72), Monthly Charges (not explicitly shown but implied numerical), Total Charges (not explicitly shown but implied numerical).

```

#lets print for the numerical columns
numerical_list = []
categorical_list = []
for cols in data_set.columns:
    unique_values = data_set[cols].unique()
    if data_set[cols].dtype == "int64" or data_set[cols].dtype == "float64":
        numerical_list.append(cols)
        print(f"the numerical columns are {numerical_list}")
    else:
        categorical_list.append(cols)
        print(f"the categorical columns are {categorical_list}")

the numerical columns are ['LoyaltyID']
the categorical columns are ['Senior Citizen']
the categorical columns are ['Senior Citizen', 'Partner']
the categorical columns are ['Senior Citizen', 'Partner', 'Dependents']
the numerical columns are ['LoyaltyID', 'Tenure']
the categorical columns are ['Senior Citizen', 'Partner', 'Dependents', 'Phone Service']
the categorical columns are ['Senior Citizen', 'Partner', 'Dependents', 'Phone Service', 'Multiple Lines']
the categorical columns are ['Senior Citizen', 'Partner', 'Dependents', 'Phone Service', 'Multiple Lines', 'Internet Service']
the categorical columns are ['Senior Citizen', 'Partner', 'Dependents', 'Phone Service', 'Multiple Lines', 'Internet Service', 'Online Security']
the categorical columns are ['Senior Citizen', 'Partner', 'Dependents', 'Phone Service', 'Multiple Lines', 'Internet Service', 'Online Security', 'Online Backup']
the categorical columns are ['Senior Citizen', 'Partner', 'Dependents', 'Phone Service', 'Multiple Lines', 'Internet Service', 'Online Security', 'Online Backup', 'Device Protection']
the categorical columns are ['Senior Citizen', 'Partner', 'Dependents', 'Phone Service', 'Multiple Lines', 'Internet Service', 'Online Security', 'Online Backup', 'Device Protection', 'Tech Support']
the categorical columns are ['Senior Citizen', 'Partner', 'Dependents', 'Phone Service', 'Multiple Lines', 'Internet Service', 'Online Security', 'Online Backup', 'Device Protection', 'Tech Support', 'Streaming TV']
the categorical columns are ['Senior Citizen', 'Partner', 'Dependents', 'Phone Service', 'Multiple Lines', 'Internet Service', 'Online Security', 'Online Backup', 'Device Protection', 'Tech Support', 'Streaming TV', 'Streaming Movies']
the numerical columns are ['LoyaltyID', 'Tenure', 'Monthly Charges']
the categorical columns are ['Senior Citizen', 'Partner', 'Dependents', 'Phone Service', 'Multiple Lines', 'Internet Service', 'Online Security', 'Online Backup', 'Device Protection', 'Tech Support', 'Streaming TV', 'Streaming Movies', 'Contract']
the categorical columns are ['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']

```

Figure 11: Division of Unique Values into numerical and categorical list

- This image shows an attempt to programmatically classify columns into numerical_list and categorical_list based on their data type (dtype).
numerical_list = [] and categorical_list = [] initialize empty lists to store column names.
- The for loop iterates through each column.
if data_set[cols]. dtype == "int64" or data_set[cols].dtype == "float64": checks if the column's data type is an integer or a float. This is a standard way to identify numerical columns in pandas.

```
DETERMINING THE NULL VALUES

#1st lets see the number of missing values
missing_values = data_set.isnull().sum()
print(f"the missing values are {missing_values}")
```

Figure 12: Determining Null value from the Data-set:

- The code checked for missing values across all columns using `data_set.isnull().sum()`.
- The output shows that there are 0 missing values for all columns, including LoyaltyID, Senior Citizen, Partner, Dependents, Tenure, 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, and Total Charges.

```
the missing values are LoyaltyID      0
Senior Citizen      0
Partner      0
Dependents      0
Tenure      0
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      0
Churn      0
dtype: int64
```

Figure 13: Overview of Null value from the Data-set:

- The output clearly indicates that all columns have 0 missing values. This is a good initial finding, suggesting no explicit NaN (Not a Number) or None values are present in the dataset.
- However, it's important to remember that "missing" can also be represented by empty strings ("") or specific placeholder values ("?", "-") that `isnull()` might not detect directly. The next images address this.

NOW CONVERSION OF VALUES

```
[ ] #CONVERTING THE TOTAL CHAGES COLUMN TO INTEGER
...
data_set['Total Charges'] = data_set["Total Charges"].astype(float)
print(data_set.info())
#
...
'\ndata_set['Total Charges'] = data_set["Total Charges"].astype(float)\nprint(data_set.info())\n#\n'
```

Figure 14: Potential error during conversion of values in dataset column:

- This image shows an attempt to convert the Total Charges column to a numerical data type (float in this case).
- It's a common requirement because financial or quantitative columns are often imported as 'object' (string) type if they contain any non-numeric characters or empty strings.
- `data_set['Total Charges'].astype(float)` tries to cast every value in that column to a floating-point number.

```
[ ] #since the problem has arise of the string cannot be coverted to float
#lets tackle the problem (empty_space)

#1 filter the space dataSset on the data-frame
filtering_the_total_charges = data_set[data_set['Total Charges']==' ']
print(f"the total chagers with space is {filtering_the_total_charges}")
print("\n"*50)

#lets detrmine the length of the filtering_total_charegs_data
lentgth_of_total_charges_data = len(filtering_the_total_charges)
print(f"the length of the filtering_the_total_charges_data is {lentgth_of_total_charges_data}")
#there are 11 rows where the total chanegs value is empty
```

Figure 15: Detrmining the empty spaces and its length

- The problem is specifically identified as (empty_space). This means that instead of actual nulls (which `isnull()` checked), there are strings consisting of just a space character (' ').
- `filtering_the_total_charges = data_set[data_set["Total Charges"]==" "]` is a crucial line. It performs boolean indexing to filter the DataFrame. It creates a new DataFrame (`filtering_the_total_charges`) containing only those rows where the Total Charges column has a value exactly equal to a single space string.
- The first print statement displays these problematic rows.
- The second block of code calculates the number of such rows: `len(filtering_the_total_charges)`.
- The final comment "`#there are 11 rows where the total chanegs value is empty`" summarizes the finding.

the total chagers with space is			LoyaltyID	Senior Citizen	Partner	Dependents	Tenure	Phone Service	\
488	344543	No	Yes	Yes	0	No			
753	150036	No	No	Yes	0	Yes			
936	497688	No	Yes	Yes	0	Yes			
1082	158969	No	Yes	Yes	0	Yes			
1340	470044	No	Yes	Yes	0	No			
3331	937662	No	Yes	Yes	0	Yes			
3826	821083	No	Yes	Yes	0	Yes			
4380	947028	No	Yes	Yes	0	Yes			
5218	135257	No	Yes	Yes	0	Yes			
6670	317862	No	Yes	Yes	0	Yes			
6754	392646	No	No	Yes	0	Yes			

Multiple Lines		Internet Service	Online Security	\
488	No phone service	DSL	Yes	
753	No	No	No internet service	
936	No	DSL	Yes	
1082	Yes	No	No internet service	
1340	No phone service	DSL	Yes	
3331	No	No	No internet service	
3826	Yes	No	No internet service	
4380	No	No	No internet service	
5218	No	No	No internet service	
6670	Yes	DSL	No	
6754	Yes	DSL	Yes	

Online Backup	Device Protection	Tech Support	\
488	No	Yes	
753	No internet service	No internet service	
936	Yes	Yes	No
1082	No internet service	No internet service	
1340	Yes	Yes	Yes
3331	No internet service	No internet service	
3826	No internet service	No internet service	
4380	No internet service	No internet service	

Figure 15: ouput for determining the empty spaces of data_set

- Significance: This step is vital for data quality. It identifies a common data entry error where "missing" numerical data might be represented by spaces or empty strings, which are not NaN but still prevent numerical conversion.
- This output confirms the presence of 11 rows where the Total Charges column contains an empty space.
- Crucially, you can observe that the Tenure column for all these 11 rows is 0. This is a common pattern in churn prediction datasets: new customers (Tenure = 0) might not have accrued any Total Charges yet, leading to an empty string instead of 0.0.
- This finding informs the next steps in data cleaning, which would likely involve either replacing these empty strings with 0.0, the median/mean, or dropping these rows, depending on the strategy chosen. Given Tenure=0, replacing with 0.0 is usually the most appropriate.

```
[ ] #for better observant #note filtering the data from a dataframe
data_set[data_set['Total Charges']==' ']
```

	Tenure	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
is	0	No	No phone service	DSL	Yes	No	Yes	Yes	Yes	No	Two year	Yes	Bank transfer (automatic)	52.55		No
is	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	20.25		No
is	0	Yes	No	DSL	Yes	Yes	Yes	No	Yes	Yes	Two year	No	Mailed check	80.85		No
is	0	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	25.75		No
is	0	No	No phone service	DSL	Yes	Yes	Yes	Yes	Yes	No	Two year	No	Credit card (automatic)	56.05		No
is	0	Yes	No	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	19.85		No
is	0	Yes	Yes	No	No internet service	No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	25.35		No

Figure 16: vizualizing the empty spaces more clearly

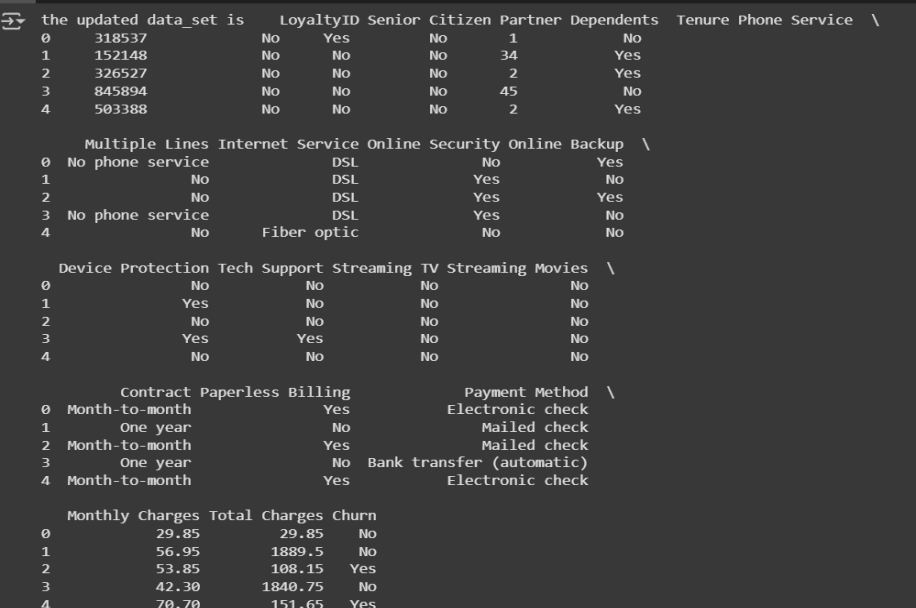
- This image serves as a visual confirmation of the previous step where rows with empty spaces in the Total Charges column were identified.
- The code `data_set[data_set['Total Charges']==' ']` filters the DataFrame to display only those rows where the Total Charges column explicitly contains a single space character. This is done for "better observant" to inspect the characteristics of these specific rows.
 - Content/Output (Partial view of DataFrame):
- The image shows several rows where:
 - Tenure is 0. This is a strong indicator that these are new customers who haven't accumulated any total charges yet.
 - Monthly Charges have values (e.g., 52.55, 20.25, 80.85), which makes sense for new customers who have just signed up for services.
 - Total Charges column is not explicitly shown in the screenshot for these rows, but based on the filtering condition, we know it contains ' ' for these entries.
- This observation reinforces the understanding that these aren't truly "missing" values in the sense of NaN, but rather represent 0 for customers with 0 tenure, expressed as an empty string.

replace the missing values in the data

```
[ ] #replacing the missng values in the totoal charges
data_set['Total Charges'] = data_set['Total Charges'].replace(" ", "0.0")
print(f"the updated data_set is {data_set.head()}")
```

Figure 17: Replacing missing values in the data frame

- This image shows the solution to the problem identified in the previous steps: replacing the empty space strings in Total Charges.
- `data_set['Total Charges'].replace(" ", "0.0")`: This is the core operation. It uses the pandas `replace()` method on the Total Charges column.
- `" "` is the value to be found (the empty space string).
- `"0.0"` is the value to replace it with. Using `"0.0"` ensures that the replacement is a string representation of a float, which will be correctly converted to a float in the next step. Since these rows correspond to `Tenure=0`, logically their Total Charges should also be 0.
- `data_set['Total Charges'] = ...` assigns the modified Series back to the Total Charges column in the DataFrame.



	the updated data_set is	LoyaltyID	Senior	Citizen	Partner	Dependents	Tenure	Phone Service	\
0	318537	No	Yes	No	1	No			
1	152148	No	No	No	34	Yes			
2	326527	No	No	No	2	Yes			
3	845894	No	No	No	45	No			
4	503388	No	No	No	2	Yes			
Multiple Lines Internet Service Online Security Online Backup \									
0	No phone service	DSL	No	Yes		Yes			
1	No	DSL	Yes	No		No			
2	No	DSL	Yes	Yes		Yes			
3	No phone service	DSL	Yes	No		No			
4	No	Fiber optic	No			No			
Device Protection Tech Support Streaming TV Streaming Movies \									
0	No	No	No	No		No			
1	Yes	No	No	No		No			
2	No	No	No	No		No			
3	Yes	Yes	No	No		No			
4	No	No	No	No		No			
Contract Paperless Billing Payment Method \									
0	Month-to-month	Yes	Electronic check						
1	One year	No	Mailed check						
2	Month-to-month	Yes	Mailed check						
3	One year	No	Bank transfer (automatic)						
4	Month-to-month	Yes	Electronic check						
Monthly Charges Total Charges Churn									
0	29.85	29.85	No						
1	56.95	1889.5	No						
2	53.85	108.15	Yes						
3	42.30	1840.75	No						
4	70.70	151.65	Yes						

Figure 18: updated data-set after performing cleaning operation

- It shows the first 5 rows of the `data_set`.
- Total Charges now contains numerical values, confirming successful replacement. Rows with `Tenure=0` may not appear in `head()`,
- The data looks clean in the displayed head, and the values are now consistent with numerical data, preparing the column for a type conversion.

```
#lets convert the total_charges column into the floatting value
data_set['Total Charges']= data_set['Total Charges'].astype(float)
print(f"the upadets data set: {data_set.info()}")

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   LoyaltyID                             7043 non-null   int64
1   Senior Citizen                         7043 non-null   object
2   Partner                               7043 non-null   object
3   Dependents                             7043 non-null   object
4   Tenure                                 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                                7043 non-null   object
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB
the upadets data set: None
```

Figure 19: performing coversion operation (figure→14)

- This image shows the final step in cleaning the Total Charges column: converting its data type to float.
- The comment #lets convert the total_charges column into the floating value states the goal.
- `data_set["Total Charges"] = data_set["Total Charges"].astype(float)`: Now that the empty spaces have been replaced with "0.0", this operation should successfully convert the entire column to a float data type without errors.
- `print(f"the upadets data set: {data_set.info()}")`: This line prints the `info()` method of the DataFrame. `data_set.info()` provides a concise summary of the DataFrame, including.

```
[ ] data_set.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
#   Column                Non-Null Count  Dtype
---  -
0   LoyaltyID              7043 non-null   int64
1   Senior Citizen         7043 non-null   object
2   Partner                7043 non-null   object
3   Dependents             7043 non-null   object
4   Tenure                 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                  7043 non-null   object
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB
```

Figure 20: updated information about the data-set (figure→14)

- `print(f'the upadets data set: {data_set.info ()}')`: This line prints the `info ()` method of the DataFrame. `data_set.info ()` provides a concise summary of the DataFrame, including:
 - The number of entries (rows).
 - The total number of columns.
 - A list of columns with their non-null count and data type (Dtype).
 - Memory usage.
- Output (below the code, starting with `<class 'pandas.core.frame.DataFrame'>`):

3.4 EXPLORATORY DATA ANALYSIS (EDA):

```
3. EDA -> EXploratory Data Analysis

[ ] #lets cekck the columns names
data_columns = data_set.columns
print(f"the columns of the data_set are {data_columns}")
print(""*50)

#viweing the data
data_view = data_set.head()
print(f"the data_set is {data_view}")
```

Figure 21: Reviewing the Data_set

The image shows two distinct code blocks, both aiming to provide a preliminary understanding of the data_set.

1. Checking Column Names:

- The first block is commented with #lets cekck the columns names.
- `data_columns = data_set.columns` assigns the column names of the data_set DataFrame to the data_columns variable.
- `print(f"the columns of the data_set are {data_columns}")` is intended to display these column names.
- `print(""*50)` is a simple line of code to print a separator for better readability in the output.
- This step is fundamental in EDA as it immediately reveals the features available in the dataset, which is crucial for understanding what information is at hand for churn prediction.

2. Viewing the Data Head:

- The second block is commented with #viweing the data (note the typo, "viewing").
- `data_view = data_set.head()` calls the head() method on the data_set DataFrame, which, by default, returns the first 5 rows of the DataFrame. This is assigned to data_view.
- `print(f"the data_set is {data_view}")` will then print these initial rows.
- Viewing the head of the data provides a quick visual inspection of the actual values, allowing for early detection of potential data entry errors, inconsistencies, or unexpected data types. It's an essential first look before diving deeper into analysis.

```

the columns of the data_set are Index(['LoyaltyID', 'Senior Citizen', 'Partner', 'Dependents', 'Tenure',
'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'],
dtype='object')
*****
the data_set is
  LoyaltyID Senior Citizen Partner Dependents Tenure Phone Service \
0  318537      No      Yes      No      1      No
1  152148      No      No      No     34      Yes
2  326527      No      No      No      2      Yes
3  845894      No      No      No     45      No
4  503388      No      No      No      2      Yes

  Multiple Lines Internet Service Online Security Online Backup \
0 No phone service      DSL      No      Yes
1      No      DSL      Yes      No
2      No      DSL      Yes      Yes
3 No phone service      DSL      Yes      No
4      No      Fiber optic      No      No

  Device Protection Tech Support Streaming TV Streaming Movies \
0      No      No      No      No
1      Yes      No      No      No
2      No      No      No      No
3      Yes      Yes      No      No
4      No      No      No      No

  Contract Paperless Billing      Payment Method \
0 Month-to-month      Yes      Electronic check
1      One year      No      Mailed check
2 Month-to-month      Yes      Mailed check
3      One year      No      Bank transfer (automatic)
4 Month-to-month      Yes      Electronic check

```

Figure 22: Output of Initial Data Exploration.

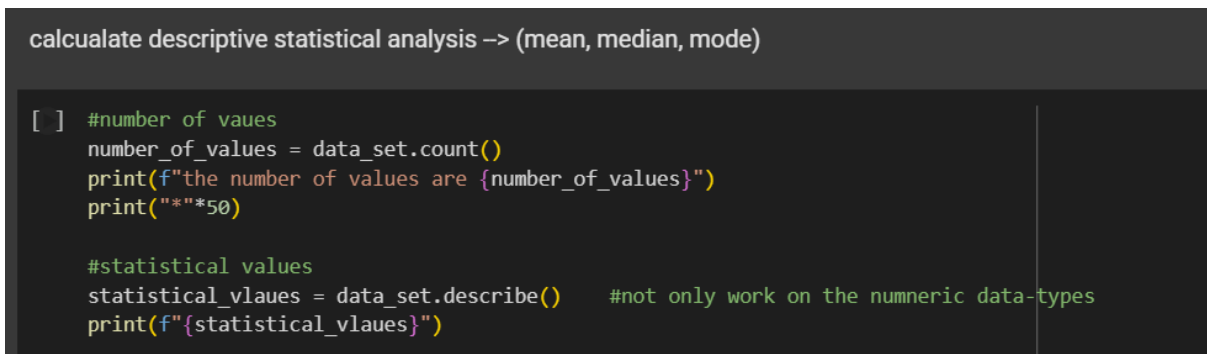
1. Column Names:

- The first output line the columns of the data_set are Index(['LoyaltyID', 'Senior Citizen', 'Partner', 'Dependents', 'Tenure', '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'], dtype='object')
- This clearly lists all 20 columns present in the data_set DataFrame. These column names are critical as they represent the features and the target variable (Churn) for your customer churn prediction model. The dtype='object' for the Index indicates that the column names themselves are stored as Python objects (strings).

2. Head of the Dataset:

- It shows the first 5 rows (indexed 0 through 4) of the DataFrame, along with all 20 columns.
- This preview is invaluable for understanding the nature of the data in each column:
 - LoyaltyID appears to be a unique identifier.

- Senior Citizen, Partner, Dependents, Phone Service, Multiple Lines, Online Security, Online Backup, Device Protection, Tech Support, Streaming TV, Streaming Movies, Paperless Billing, Churn seem to be categorical features, often binary ('Yes'/'No') or with a few discrete categories.
 - Tenure, Monthly Charges, Total Charges are numerical features.
 - Internet Service, Contract, Payment Method are also categorical but with more than two possible values.
- This visual inspection helps confirm that the data has loaded correctly and provides a qualitative sense of the data types before formal type checking.



```
calcuulate descriptive statistical analysis -> (mean, median, mode)

[ ] #number of vaues
    number_of_values = data_set.count()
    print(f"the number of values are {number_of_values}")
    print(""*50)

    #statistical values
    statistical_vlaues = data_set.describe()    #not only work on the numneric data-types
    print(f"{statistical_vlaues}")
```

Figure 23: Count and Statistical analysis

The image shows two main code blocks, both related to statistical summaries of the `data_set`.

1. Counting Values (Non-Nulls):

- The block is commented with `#number of vaues` (note the typo, "values").
- `number_of_values = data_set.count()` calls the `count()` method on the DataFrame. This method returns the number of non-null observations for each column.
- `print(f'the number of values are {number_of_values}')` prints the result.
- `print(""*50)` adds another separator.
- Checking the count of values per column is a critical first step in identifying missing data. If any column had a count less than the total number of rows, it would indicate missing values, which would need to be addressed in data preprocessing.

2. Calculating Statistical Values (Descriptive Statistics):

- The block is commented with `#statistical values`.
- `statistical_vlaues = data_set.describe()` calls the `describe()` method. This powerful method generates descriptive statistics that summarize the central

tendency, dispersion, and shape of a dataset's distribution, excluding NaN values.

- The inline comment `#not only work on the nummeric data-types` is partially misleading; `describe()` by default primarily provides statistics for *numerical* columns. If you want to include categorical columns, you typically need to specify `include='all'` or `include=['object']`.
- `print(f'{{statistical_vlaues}}')` prints the resulting statistical summary.
- This step is crucial for understanding the distribution of numerical features, identifying potential outliers, and gaining insights into the typical range and spread of values, all of which are vital for feature engineering and model selection.

```
the number of values are LoyaltyID 7043
Senior Citizen 7043
Partner 7043
Dependents 7043
Tenure 7043
Phone Service 7043
Multiple Lines 7043
Internet Service 7043
Online Security 7043
Online Backup 7043
Device Protection 7043
Tech Support 7043
Streaming TV 7043
Streaming Movies 7043
Contract 7043
Paperless Billing 7043
Payment Method 7043
Monthly Charges 7043
Total Charges 7043
Churn 7043
dtype: int64
*****

```

	LoyaltyID	Tenure	Monthly Charges	Total Charges
count	7043.000000	7043.000000	7043.000000	7043.000000
mean	550382.651001	32.371149	64.761692	2279.734304
std	260776.118690	24.559481	30.090047	2266.794470
min	100346.000000	0.000000	18.250000	0.000000
25%	323604.500000	9.000000	35.500000	398.550000
50%	548704.000000	29.000000	70.350000	1394.550000
75%	776869.000000	55.000000	89.850000	3786.600000
max	999912.000000	72.000000	118.750000	8684.800000

. Figure 24: Output of Descriptive Statistics

□ Count of Non-Null Values Per Column:

- The first output section, preceded by the number of values are, lists each column name and its corresponding count.
- For every column, the count is 7043. This is a very important finding: it confirms that there are no missing values in any of the columns in your `data_set`. This simplifies the data preprocessing stage as you won't need to handle NaNs through imputation or removal.
- The `dtype: int64` at the bottom indicates that the counts are integers.

□ Descriptive Statistics for Numerical Columns:

- Following a line of asterisks
*****, this section displays the output of `data_set.describe()`.
- It provides a summary table for the numerical columns: LoyaltyID, Tenure, Monthly Charges, and Total Charges.
- For each of these columns, the following statistics are presented:
 - count: Confirms 7043 non-null values for each.
 - mean: The average value.
 - std: The standard deviation, indicating the spread of data.
 - min: The minimum value.
 - 25%: The first quartile (Q1).
 - 50%: The median (second quartile, Q2).
 - 75%: The third quartile (Q3).
 - max: The maximum value.

Insights from this table:

- **LoyaltyID:** While numerical, this is likely an identifier, so its statistical properties like mean and std deviation aren't typically meaningful in a predictive sense.
- **Tenure:** Shows a range from 0 to 72, with a mean of approximately 32.5 months. The spread suggests a mix of new and long-term customers.
- **Monthly Charges:** Ranges from 18.25 to 118.75, with a mean of around 64.76. This indicates the varying monthly service costs for customers.
- **Total Charges:** Ranges from 0 to 8684.80, with a mean of about 2279.73. The wide range and the fact that the minimum is 0 (possibly for new customers who haven't accumulated charges yet) are notable.

Understanding the distribution of the numeric features

```
[ ] #histogram function

def histo_plot(data_set, column_name):
    #setup for plotting the graph
    plt.figure(figsize=(10,5))
    #utilizing the seaborn
    sns.histplot(data_set[column_name], kde=True)
    plt.title(f"Histogrammic distribution of the {column_name}")

    #lets calculate the statistical value of the data_set column --> mean(), median()
    mean_value = data_set[column_name].mean()
    print(f"the mean value of {column_name} is {mean_value}")
    median_value = data_set[column_name].median()
    print(f"the median value of {column_name} is {median_value}")
    mode_value = data_set[column_name].mode()
    print(f"the mode value of the {column_name} is {mode_value}")

    #lets add vertical lines to these mean, median, mode value for understanding
    plt.axvline(mean_value, color='red', linestyle='--', label = 'Mean')
    plt.axvline(median_value, color='green', linestyle='solid', label = 'Median')

    plt.legend()
    plt.show()
```

Figure 25: Understanding the Distribution of Numerical Features (Code)

This image displays the Python code for a `histo_plot` function. This function is designed to visualize the distribution of numerical features in your dataset.

- **Key Functionality:**
 - It uses `seaborn.histplot` to create histograms, which show the frequency distribution of a given numerical column. The `kde=True` argument adds a Kernel Density Estimate line, providing a smoothed representation of the distribution.
 - It calculates and prints the mean, median, and mode of the specified column, providing key central tendency measures.
 - It adds vertical lines to the histogram, indicating the positions of the mean (red dashed line) and median (green solid line), which helps in visually understanding the skewness and central tendency of the data.
- **Relevance to Churn Prediction:** Understanding the distribution of numerical features like Monthly Charges, Total Charges, and Tenure is crucial for data preprocessing. For instance, highly skewed data might require transformation (e.g., log transformation) to improve model performance. Outliers, visible in histograms, can also be identified and handled.

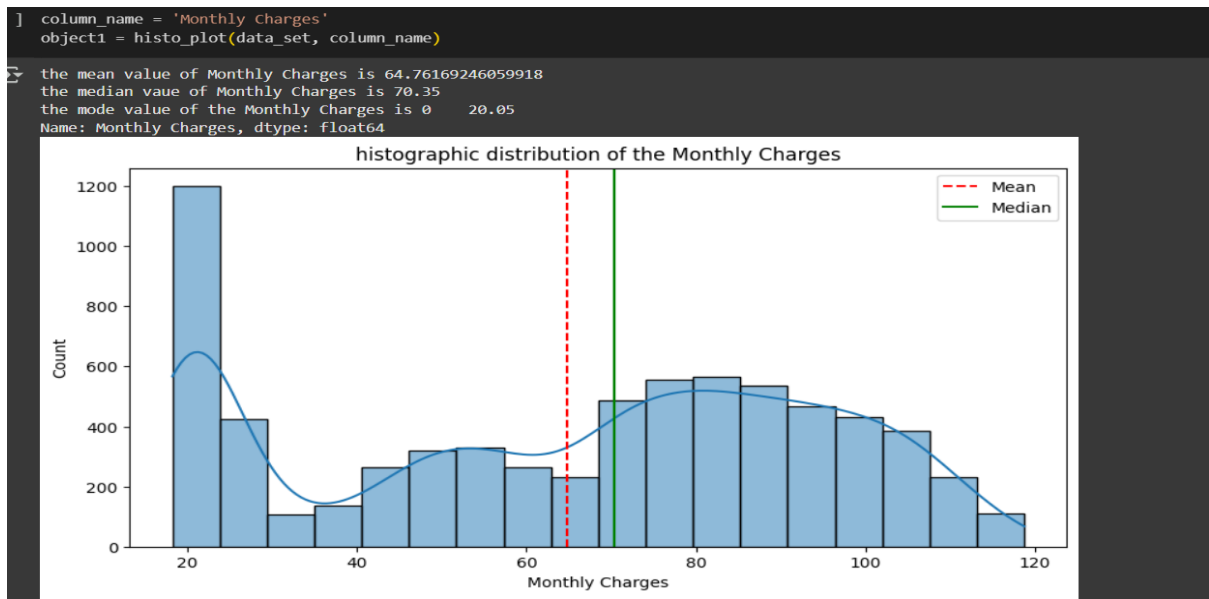


Figure 26: Histogram for Monthly Charges

This image shows the output of the `histo_plot` function applied to the 'Monthly Charges' column.

- Key Observations:
 - The histogram displays the distribution of monthly charges among customers.
 - The mean value is approximately 64.76, and the median is 70.35. The mode is around 20 and 28.05.
 - The distribution appears somewhat bimodal, with peaks around 20 and 70-80, suggesting distinct groups of customers based on their monthly spending (e.g., low-cost plans vs. higher-tier plans).
 - The mean (red line) is slightly to the left of the median (green line) at the higher peak, indicating a slight left skew or potentially the influence of lower charges pulling the mean down.
- Relevance to Churn Prediction: This visualization helps identify typical monthly spending patterns. High churn rates might be associated with customers in specific charge ranges.

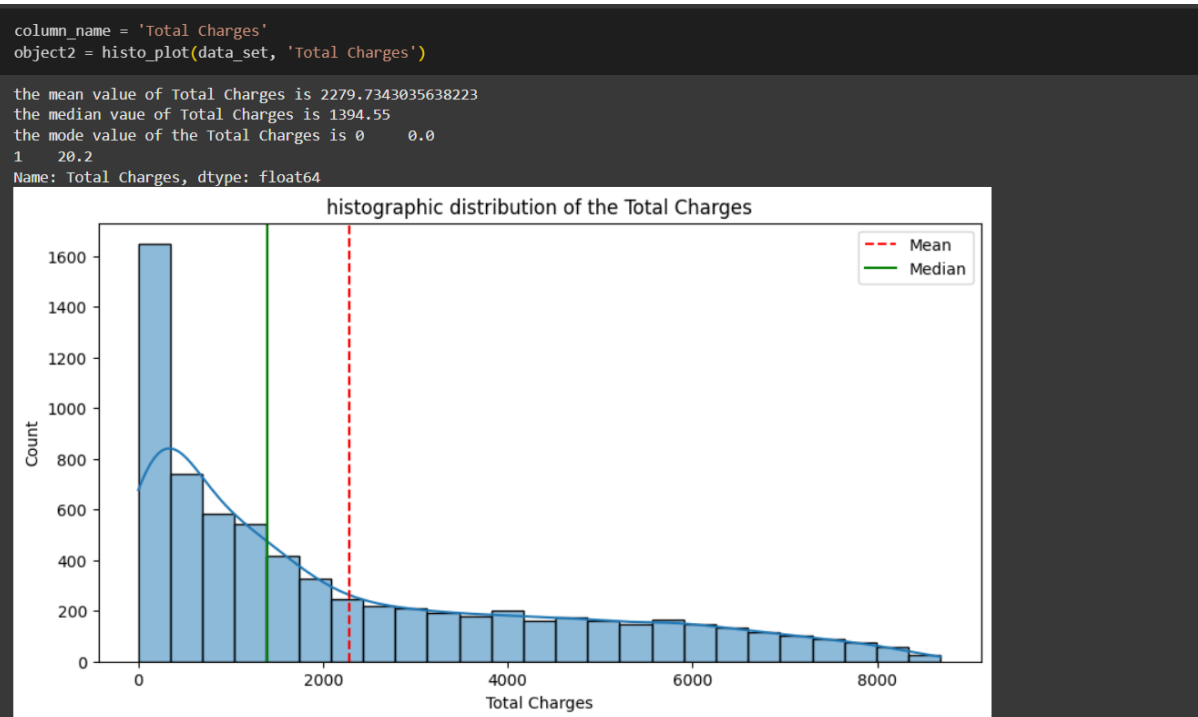


Figure 27: Histogram for Total Charges

This image displays the output of the `histo_plot` function for the 'Total Charges' column.

- **Key Observations:**
 - The histogram shows the distribution of total charges incurred by customers.
 - The mean value is approximately 2279, and the median is 1394.55. The mode is around 0 and 20.2.
 - The distribution is highly right-skewed, with a large number of customers having low total charges (likely new customers) and a long tail extending towards higher total charges (long-term customers). The mean is significantly higher than the median due to this skewness.
- **Relevance to Churn Prediction:** This feature is critical as it often correlates with customer loyalty. Newer customers (low total charges) might be more prone to churn than long-term customers with high total charges. The skewness indicates a need for careful handling during model training.

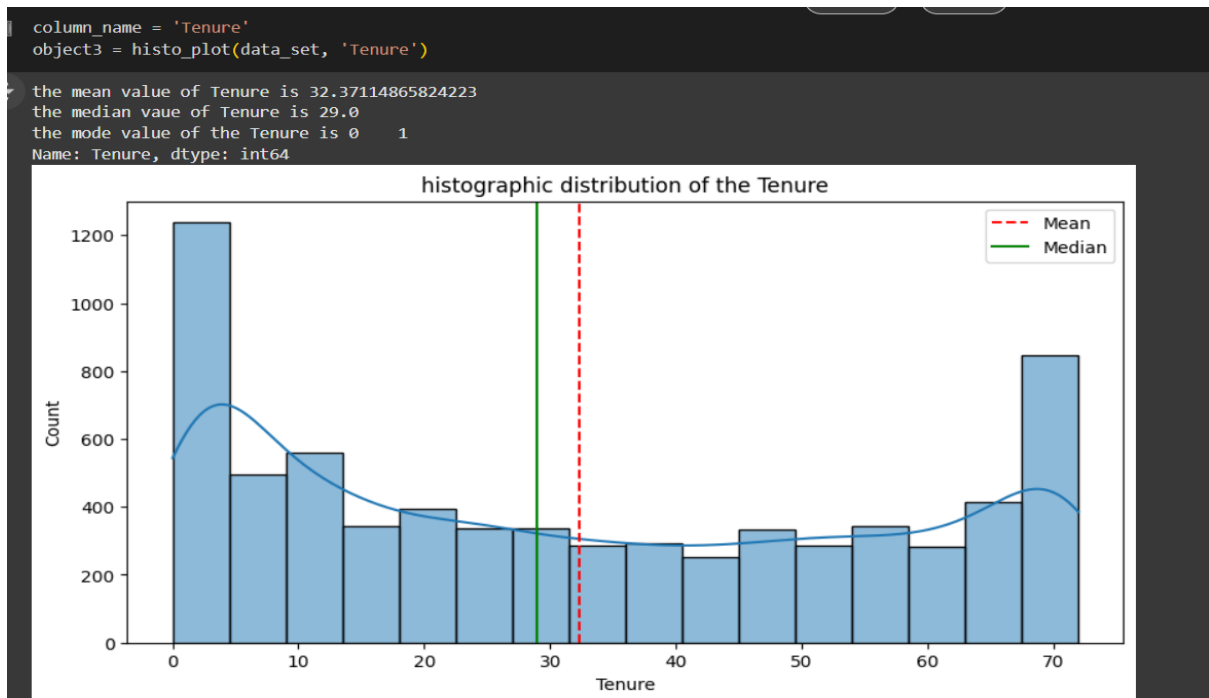


Figure 27: Histogram for Tenure

This image presents the output of the `histo_plot` function for the 'Tenure' column (how long a customer has been with the company).

- Key Observations:
 - The histogram illustrates the distribution of customer tenure.
 - The mean tenure is approximately 32.37 months, and the median is 29.0 months. The mode is at 0 and 1 months.
 - The distribution is somewhat U-shaped, with peaks at very low tenure (new customers) and very high tenure (long-term loyal customers). This indicates that the company has both a significant number of new customers and a strong base of long-standing customers.
- Relevance to Churn Prediction: Tenure is a strong predictor of churn. New customers (low tenure) often have higher churn rates as they are still evaluating the service, while very long-term customers are usually highly loyal. This U-shape highlights the importance of targeting both new customers with retention strategies and maintaining the satisfaction of loyal customers.

3.4 MACHINE LEARNING MODEL :

```
[ ] data_set.columns

⇒ Index(['CustomerID', 'Age', 'Gender', 'Tenure', 'MonthlyCharges',
        'ContractType', 'InternetService', 'TotalCharges', 'TechSupport',
        'Churn'],
        dtype='object')

**here we will divide the columns into x and y labels

1. y = Churn → which will be predicted
2. x = columns which we will choose based on which the Y will predicted

[ ] #here we will divide the

[ ] #create x and y variables
    #global variables which will be used throughout
    Y = data_set[["Churn"]] # for the prediction
    X = data_set[["Age", "Gender", "MonthlyCharges", "Tenure"]]

[ ] X
    #Y
```

Figure 28: Data Columns and X/Y Variable Division

This image showcases the initial step of identifying the columns in your dataset and then explicitly defining which columns will serve as your independent variables (features, X) and your dependent variable (target, Y).

- **Key Functionality/Information:**

- **data_set.columns:** This line of code lists all the columns present in your data_set DataFrame. The columns include CustomerID, Age, Gender, Tenure, MonthlyCharges, ContractType, InternetService, TotalCharges, TechSupport, and Churn.
- **X and Y Labels Division:** This section clearly states the objective:
 - **Y = Churn:** The 'Churn' column is identified as the target variable that your model will predict. This is a binary classification problem (customer churns or not).
 - **X = columns:** The remaining columns (or a selected subset of them) will be used as features to predict Y.

- **Variable Creation Code:** The subsequent code block explicitly creates these X and Y variables:
 - `y = data_set[["Churn"]]`: Extracts the 'Churn' column into the y DataFrame.
 - `x = data_set[["Age", "Gender", "MonthlyCharges", "Tenure"]]`: Selects a specific set of features (Age, Gender, MonthlyCharges, Tenure) to form the x DataFrame.
- **Relevance to Churn Prediction:** This is a fundamental step in any supervised machine learning project. Clearly defining your target variable (Churn) and selecting relevant features (X) is crucial for building a predictive model. The choice of X features here suggests you've identified these as potentially strong predictors of churn.

**** HERE WE WILL USE LAMBDA FUNCTION ****

1. First lets focus on the X variables

[] x

	Age	Gender	MonthlyCharges	Tenure
0	49	Male	88.35	4
1	43	Male	36.67	0
2	51	Female	63.79	2
3	60	Female	102.34	8
4	42	Male	69.01	32
...
995	42	Male	37.14	41
996	62	Male	80.93	9
997	51	Female	111.72	15
998	39	Male	65.67	68
999	50	Male	56.67	1

1000 rows × 4 columns

[] y

	Churn
0	Yes
1	Yes
2	Yes
3	Yes
4	Yes
...	...
995	Yes
996	Yes
997	Yes
998	Yes
999	Yes

1000 rows × 1 columns

Figure 29: Ouput of Data Columns and X/Y Variable Division

- Key Observations:
 - The table displays the first few and last few rows of the X DataFrame.
 - It contains four columns: Age, Gender, MonthlyCharges, and Tenure.
 - The table shows the 'Churn' column, indicating whether a customer has churned ('Yes') or not.
 - The values are categorical: 'Yes'. (While only 'Yes' is shown in the snippet, it's implied that 'No' is also a possible value, making it a binary classification problem).

```
[ ] #LETS COVERT IT INTO TO NUMERICAL VLAUES OF BOTH THE COLUMNS
#LETS MAKE IT GLOBAL VARIABLES
X["Gender"] = X["Gender"].apply(lambda x: 1 if x=="Male" else 0)
Y["Churn"] = Y["Churn"].apply(lambda x:1 if x=="Yes" else 0)

[ ] #LETS CHECK THE COLUMNS THE VARIABLES
print(X)

print('*' * 80)
print(Y)
```

Figure 30: Converting Categorical Variables to Numerical Values (Code)

Key Functionality:

- Gender Encoding:
 - `X["Gender"] = X["Gender"].apply(lambda x: 1 if x=="Male" else 0)`: This line performs binary encoding on the 'Gender' column in the X DataFrame. It assigns 1 if the Gender is 'Male' and 0 otherwise (implying 'Female' becomes 0).
- Churn Encoding:
 - `Y["Churn"] = Y["Churn"].apply(lambda x: 1 if x=="Yes" else 0)`: This line performs binary encoding on the 'Churn' column in the Y DataFrame. It assigns 1 if Churn is 'Yes' and 0 otherwise (implying 'No' becomes 0).
- Verification: The `print(X)` and `print(Y)` statements are included to display the transformed DataFrames, allowing you to verify the successful encoding.
- **Relevance to Churn Prediction:** This is a critical data preprocessing step. Most machine learning algorithms require numerical input. By converting 'Gender' and 'Churn' into numerical (binary) formats, you make them suitable for model training. This particular method is a simple form of one-hot encoding for binary categorical variables.

```
[1000 rows x 4 columns]
```

	Age	Gender	MonthlyCharges	Tenure
0	49	1	88.35	4
1	43	1	36.67	0
2	51	0	63.79	2
3	60	0	102.34	8
4	42	1	69.01	32
..
995	42	1	37.14	41
996	62	1	80.93	9
997	51	0	111.72	15
998	39	1	65.67	68
999	50	1	56.67	1

```
*****
```

	Churn
0	1
1	1
2	1
3	1
4	1
..	...
995	1
996	1
997	1
998	1
999	1

```
[1000 rows x 1 columns]
```

Figure 31: Output of Numerical Conversion

Transformed X DataFrame: The 'Gender' column now contains only 0s and 1s instead of 'Male' and 'Female'. For example, row 0 now has Gender as 1 (which means Male), and row 2 has Gender as 0 (which means Female). The numerical columns Age, MonthlyCharges, and Tenure remain unchanged.

- Transformed Y DataFrame: The 'Churn' column now also contains only 0s and 1s. Given the snippet from image 27 part-2.png only showed 'Yes', it's expected that all rows here will now be 1 (assuming 'Yes' maps to 1), but in a complete dataset, you would see a mix of 0s (for 'No') and 1s (for 'Yes').
- The dimensions remain 1000 rows x 4 columns for X and 1000 rows x 1 column for Y.

□ **Relevance to Churn Prediction:** This output confirms that your data is now in a fully numerical format, which is a prerequisite for applying most machine learning algorithms. This prepares your dataset for the next stages of model building, such as splitting into training and testing sets, and then training your chosen churn prediction model.

```
[ ] from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.2,random_state=42)

    #here we will use 20 percent of the data for testing and 80 percent for training
```

```
[ ] x
```



	Age	Gender	MonthlyCharges	Tenure
0	49	1	88.35	4
1	43	1	36.67	0
2	51	0	63.79	2
3	60	0	102.34	8
4	42	1	69.01	32
...
995	42	1	37.14	41
996	62	1	80.93	9
997	51	0	111.72	15
998	39	1	65.67	68
999	50	1	56.67	1

```
[ ] from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
```



```
#lets train the x data
x_train = scaler.fit_transform(x_train)

#now save the data

import joblib
joblib.dump(scaler,"scaler.pkl")
```



```
['scaler.pkl']
```

```
[ ] # NOW for testing part
    x_test = scaler.fit_transform(x_test)
```

```
[ ] #lets check it
    x_train
    x_test
```

```

array([[ 0.57309103, -0.88640526,  1.08859098, -1.03475948],
 [ 0.98354119, -0.88640526,  1.64456522, -0.9281382 ],
 [-1.88960992,  1.12815215,  1.59448733, -0.44834246],
 [-0.5556469 , -0.88640526,  1.42707365,  0.3513171 ],
 [-1.17132214,  1.12815215, -0.70799902, -0.71489565],
 [ 0.98354119,  1.12815215, -0.72773775, -0.55496374],
 [-0.96609706, -0.88640526,  0.95626838,  0.72449157],
 [-0.96609706,  1.12815215, -1.44710481,  0.29800647],
 [ 0.67570357,  1.12815215, -1.57833081,  0.03145328],
 [ 0.76087198, -0.88640526,  0.86890622,  3.33671282],
 [ 0.45303436, -0.88640526, -1.45478098, -1.03475948],
 [ 0.06002834, -0.88640526, -0.7500352 ,  0.19138519],
 [ 0.88092865, -0.88640526,  0.42807458, -0.98144884],
 [-0.65825944, -0.88640526,  1.47349622,  0.19138519],
 [ 0.26525341, -0.88640526, -1.55201251, -0.55496374],
 [-0.86348452,  1.12815215, -1.0066387 , -0.28841055],
 [ 0.16264088,  1.12815215, -0.97885826,  3.07015963],
 [ 1.08615373, -0.88640526, -0.61478834,  0.19138519],
 [ 0.16264088,  1.12815215, -1.48877546, -0.98144884],
 [-0.14519674, -0.88640526, -1.33452094, -0.9281382 ],
 [ 0.65825944,  1.12815215,  0.53883079, -0.60827438],
 [ 0.26525341, -0.88640526, -1.55786102, -0.76820629],
 [ 0.47047849, -0.88640526, -1.09217319,  0.08476391],
 [ 0.06002834, -0.88640526, -0.54095087, -0.76820629],
 [ 0.57309103, -0.88640526,  1.58132818,  0.99104476],
 [ 0.67570357, -0.88640526,  1.00561521,  0.83111284],
 [-0.45303436,  1.12815215,  1.15877313, -1.03475948],
 [ 1.59921643,  1.12815215, -0.93389782,  0.99104476],
 [ 0.06002834, -0.88640526, -0.16335627, -0.9281382 ],
 [-0.65825944, -0.88640526, -0.12643753,  0.19138519],
 [-1.47915976, -0.88640526, -0.08074603,  0.56455965],
 [-1.27393468,  1.12815215, -0.34027378, -0.55496374],
 [ 0.47047849, -0.88640526, -0.48319681,  0.93773412],
 [-1.37654722,  1.12815215,  0.54614143, -0.075168 ],
 [ 0.06002834,  1.12815215,  1.140131 , -0.9281382 ],

```

MODELLING

```
# @title MODELLING
```

```
[ ] #LETS GO WITHH ACCURACY SCORE ---> takes values between 0 and 1
from sklearn.metrics import accuracy_score
def model_performace(prediction):
    print("the accuracy score is {}".format(accuracy_score(Y_test,prediction)))
```

HYPER PARAMETER TUNNING FOR THE MODELS

```
[ ] # @title HYPER PARAMETER TUNNING FOR THE MODELS
```

```
#here we are gone use GRID SEARCH CV
```

```
from sklearn.model_selection import GridSearchCV #--> utlize for svc
```


[] Start coding or [generate](#) with AI.

▼ Support Vector Classifier MODEL

[] # @title Support Vector Classifier MODEL

[] #IMPORTING THE MODEL
from sklearn.svm import SVC

#creating a variable name SVM
svm = SVC()

...

Valid parameters are: ['C', 'break_ties', 'cache_size', 'class_weight', 'coef0', 'decision_function_shape', 'degree', 'gamma', 'kernel', 'max_iter', 'prot

```
#Setting up parameters
param_grid = {
    "C" : [0.01,0.1,0.5,1],
    "kernel" : ["Linear","rbf","poly"],
}

#initilizing varaibale svc_model

svc_model = GridSearchCV(svm,param_grid,cv=5)

#lets fit the model

svc_model.fit(X_train,Y_train)

#lets check the parameters
print(svc_model.best_params_)

print("*** * 80)

#lets predict for the X_test --> 20percent data

svc_model.predict(X_test)
```

```
#model performace
```

```
prediction2 = svc_model.predict(X_test)  
print(model_performace(prediction2))
```

```
{'C': 0.01, 'kernel': 'poly'}
```

```
*****
```

```
the accuraccy score is 0.885
```

```
None
```

```
#now lets dump the svc model
```

```
import joblib  
joblib.dump(svc_model,"model.pkl")
```

```
['model.pkl']
```


3.5 DEPLOYMENT OF MODEL:

```
import streamlit as st
import joblib
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore") # Ignore potential warnings from sklearn/joblib

# Load the trained model and scaler
try:
    model = joblib.load('model.pkl') # Your trained SVC model from GridSearchCV
    scaler = joblib.load('scaler.pkl') # Your trained StandardScaler
except FileNotFoundError:
    st.error("Model or scaler file not found. Make sure 'model.pkl' and 'scaler.pkl' are in the same directory.")
    st.stop() # Stop execution if files are missing

# --- Streamlit UI ---

st.title("Telecom Customer Churn Prediction")
st.write("Enter customer details to predict churn.")

# Input fields for the features used in the model
# Features were: "Age", "Gender", "MonthlyCharges", "Tenure"

age = st.number_input("Age", min_value=10, max_value=100, value=30)
gender = st.selectbox("Gender", ["Male", "Female"])
monthly_charges = st.number_input("Monthly Charges", min_value=0.0, value=130.0)
tenure = st.number_input("Tenure (months)", min_value=0, max_value=100, value=12)

# Convert gender input to numerical (Male=1, Female=0 as per your notebook)
gender_encoded = 1 if gender == "Male" else 0

# Create a DataFrame with the input data
input_data = pd.DataFrame({
    "Age": [age],
    "Gender": [gender_encoded],
    "MonthlyCharges": [monthly_charges],
    "Tenure": [tenure]
```

```

age = st.number_input("Age", min_value=10, max_value=100, value=30)
gender = st.selectbox("Gender", ["Male", "Female"])
monthly_charges = st.number_input("Monthly Charges", min_value=0.0, value=130.0)
tenure = st.number_input("Tenure (months)", min_value=0, max_value=100, value=12)

# Convert gender input to numerical (Male=1, Female=0 as per your notebook)
gender_encoded = 1 if gender == "Male" else 0

# Create a DataFrame with the input data
input_data = pd.DataFrame({
    "Age": [age],
    "Gender": [gender_encoded],
    "MonthlyCharges": [monthly_charges],
    "Tenure": [tenure]
})

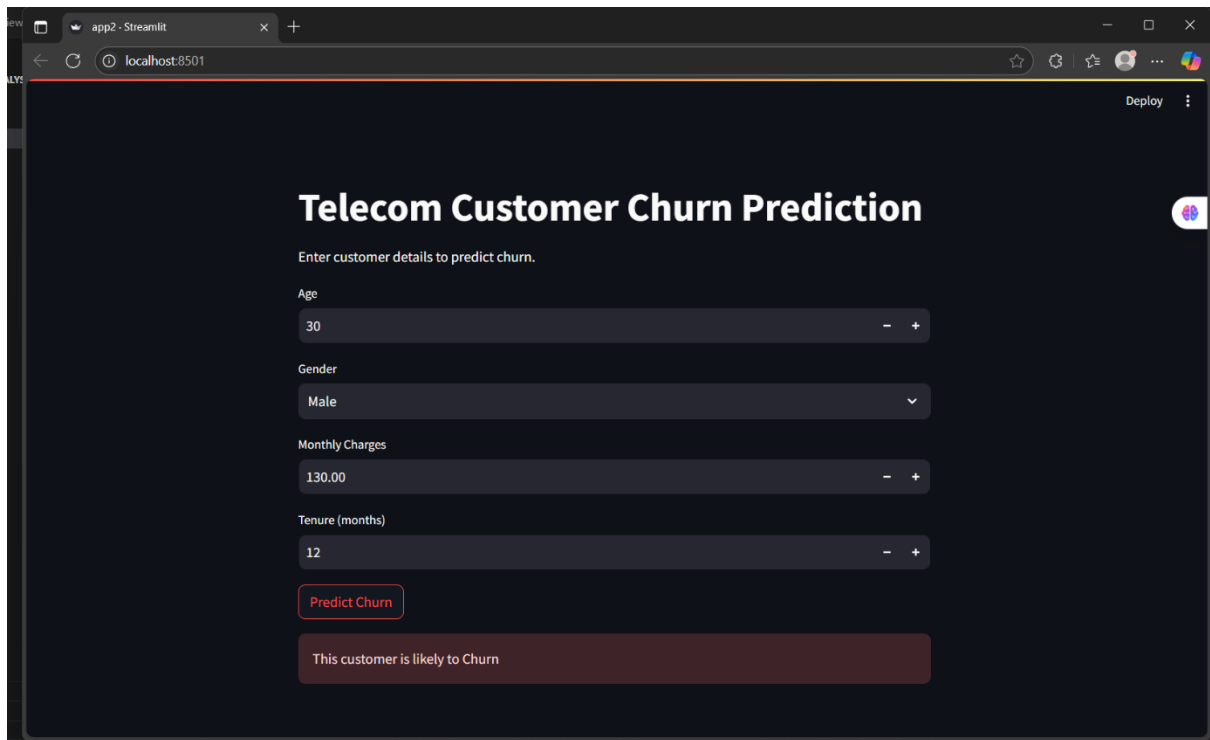
# Button to make prediction
if st.button("Predict Churn"):
    # Preprocess the input data using the loaded scaler
    # The scaler expects input as a DataFrame or array, and will output a numpy array
    scaled_input = scaler.transform(input_data)

    # Make prediction
    prediction = model.predict(scaled_input) # model is the GridSearchCV object

    st.balloons()

    # Interpret the prediction
    # Your Y column 'Churn' was encoded: Yes=1, No=0
    if prediction[0] == 1:
        result = "This customer is likely to Churn"
        st.error(result) # Use st.error for visual emphasis on churn
    else:
        result = "This customer is unlikely to Churn"
        st.success(result) # Use st.success for visual emphasis on no churn

```



Project link

[ArijeetGoswami/-CUSTOMER-CHURN-PREDICTION-FOR-A-TELECOM-COMPANY:](#)

[ArijeetGoswami/CUSTOMER-CHURN-PREDICTION-FOR-A-TELECOM-COMPANY-2:
with streamlit](#)