Customer Churn Analysis for Telecommunications Company

Sishir Pandey - Project Manager

Fahim Arman - Data Engineer

Jitesh Akaveeti - Data Analyst (Predictive Modelling)

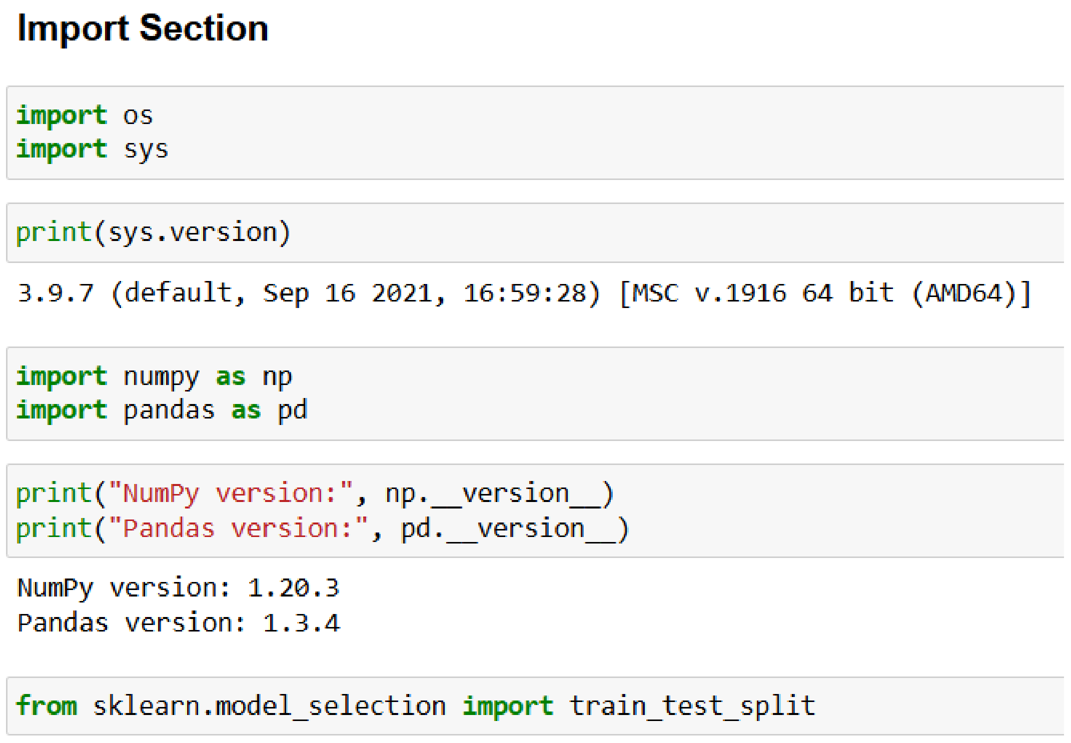
Chen - Data Analyst (Clustering)

Preeti Khatri - Data Analyst (Predictive Modelling)

Bishesh Aryal - Business Analyst

Scaling Techniques Documentation

Import Section:



Import Section

```
import os
import sys
```

```
print(sys.version)
```

```
3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
```

```
import numpy as np
import pandas as pd
```

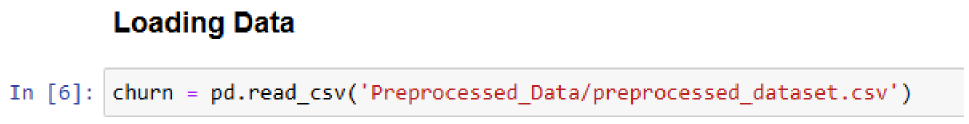
```
print("NumPy version:", np.__version__)
print("Pandas version:", pd.__version__)
```

```
NumPy version: 1.20.3
Pandas version: 1.3.4
```

```
from sklearn.model_selection import train_test_split
```

This section imports essential Python libraries and displays their versions to ensure compatibility and reproducibility. The *os* and *sys* modules are used for interacting with the operating system and accessing system-level information, such as the Python version. The *numpy* and *pandas* libraries are imported for numerical operations and data manipulation, with their versions printed for reference. Finally, the *train_test_split* function from the *sklearn.model_selection* module is imported to later divide the dataset into training and testing sets for machine learning tasks.

Data Loading:

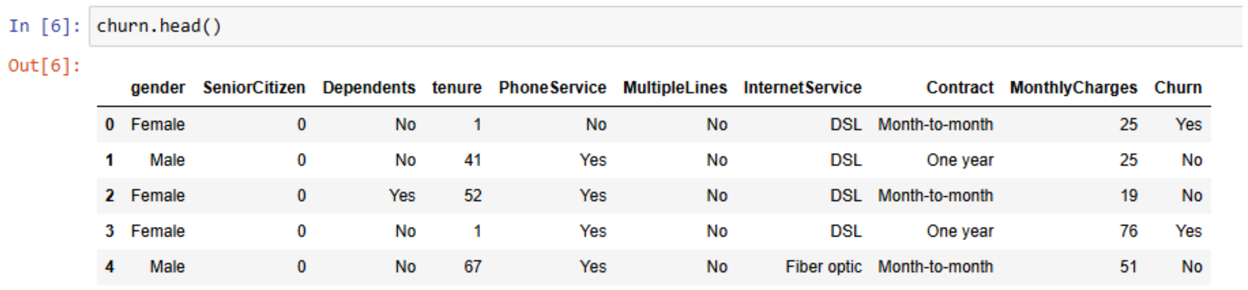


Loading Data

```
In [6]: churn = pd.read_csv('Preprocessed_Data/preprocessed_dataset.csv')
```

This line reads a CSV file named “preprocessed_dataset.csv” and loads it into a Pandas DataFrame called churn. The DataFrame serves as a structured table that allows easy manipulation and analysis of the dataset. Using the pd.read_csv() function, data from the file is imported into Python, enabling further processing, exploration, and preparation for machine learning or statistical analysis.

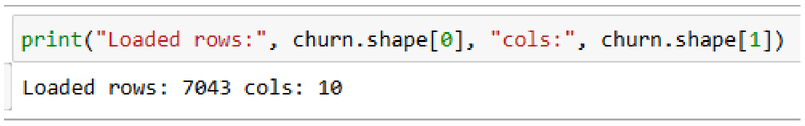
Data View:



```
In [6]: churn.head()
Out[6]:
```

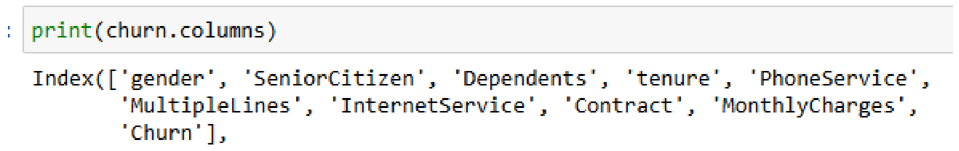
	gender	SeniorCitizen	Dependents	tenure	PhoneService	MultipleLines	InternetService	Contract	MonthlyCharges	Churn
0	Female	0	No	1	No	No	DSL	Month-to-month	25	Yes
1	Male	0	No	41	Yes	No	DSL	One year	25	No
2	Female	0	Yes	52	Yes	No	DSL	Month-to-month	19	No
3	Female	0	No	1	Yes	No	DSL	One year	76	Yes
4	Male	0	No	67	Yes	No	Fiber optic	Month-to-month	51	No

The command churn.head() displays the first five rows of the dataset stored in the churn DataFrame, providing a quick preview of the data structure and contents. From the output, we can observe that the dataset includes columns such as Gender, SeniorCitizen, Dependents, Tenure, PhoneService, MultipleLines, InternetService, Contract, MonthlyCharges, and Churn. Each row represents a customer record with various demographic and service-related details, while the Churn column indicates whether the customer has discontinued the service (“Yes”) or remained (“No”). This initial inspection helps in understanding the data types and identifying any potential preprocessing needs.



```
print("Loaded rows:", churn.shape[0], "cols:", churn.shape[1])
Loaded rows: 7043 cols: 10
```

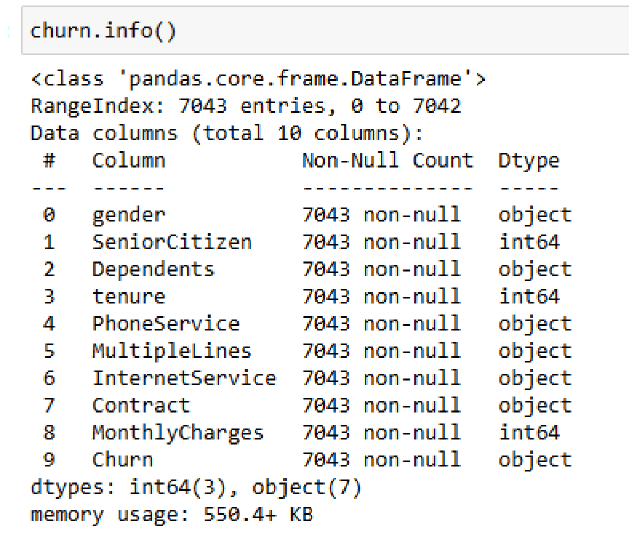
This line prints the number of rows and columns in the dataset using the shape attribute of the DataFrame. The command churn.shape[0] gives the total number of rows (records), and churn.shape[1] gives the total number of columns (features). The output “Loaded rows: 7043 cols: 10” indicates that the dataset contains 7,043 customer records and 10 columns of information. This helps verify that the data has been successfully loaded and provides an overview of its size before further analysis.



```
print(churn.columns)
```

```
Index(['gender', 'SeniorCitizen', 'Dependents', 'tenure', 'PhoneService',
       'MultipleLines', 'InternetService', 'Contract', 'MonthlyCharges',
       'Churn'],
```

This command prints the names of all the columns present in the churn DataFrame using the columns attribute. The output lists each column name—such as gender, SeniorCitizen, Dependents, tenure, PhoneService, MultipleLines, InternetService, Contract, MonthlyCharges, and Churn—which represent the different features or variables in the dataset. This helps confirm that all expected columns have been loaded correctly and provides a clear overview of the data structure for further analysis or preprocessing.

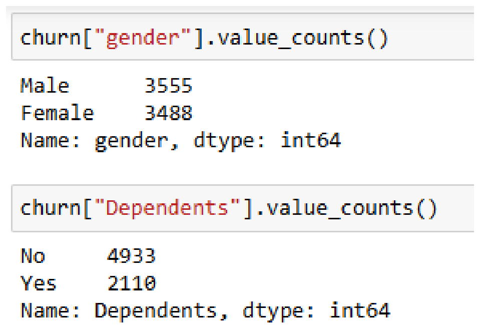


```
churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   gender           7043 non-null   object
 1   SeniorCitizen    7043 non-null   int64 
 2   Dependents       7043 non-null   object
 3   tenure           7043 non-null   int64 
 4   PhoneService     7043 non-null   object
 5   MultipleLines    7043 non-null   object
 6   InternetService  7043 non-null   object
 7   Contract         7043 non-null   object
 8   MonthlyCharges   7043 non-null   int64 
 9   Churn            7043 non-null   object
dtypes: int64(3), object(7)
memory usage: 550.4+ KB
```

The command churn.info() provides a concise summary of the dataset's structure and content. It shows that the DataFrame contains 7,043 entries (rows) and 10 columns, with each column name, data type, and the number of non-null (non-missing) values displayed. The output indicates that there are no missing values in any column, as all have 7,043 non-null entries.

The data types listed show that three columns (SeniorCitizen, tenure, and MonthlyCharges) are stored as integers (int64), while the remaining seven columns are stored as objects, meaning they contain categorical or text data. This information is useful for understanding the composition of the dataset and determining which preprocessing steps—such as encoding categorical data or scaling numeric values—will be needed before building machine learning models.



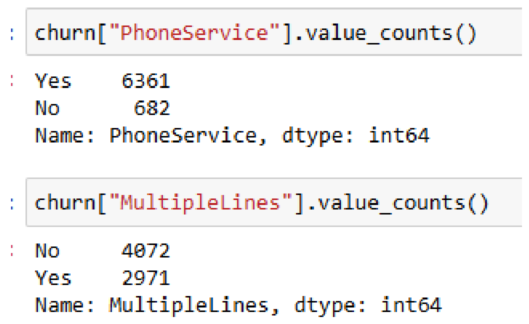
```
churn["gender"].value_counts()
```

```
Male      3555
Female    3488
Name: gender, dtype: int64
```

```
churn["Dependents"].value_counts()
```

```
No     4933
Yes    2110
Name: Dependents, dtype: int64
```

These commands use the value_counts() function to display the frequency of unique values within specific columns of the churn DataFrame. For the “gender” column, the output shows that there are 3,555 male and 3,488 female customers, indicating a nearly balanced gender distribution in the dataset. For the “Dependents” column, the results show that 4,933 customers have no dependents, while 2,110 customers have dependents.



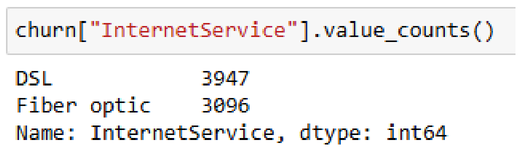
```
churn["PhoneService"].value_counts()
```

```
Yes    6361
No      682
Name: PhoneService, dtype: int64
```

```
churn["MultipleLines"].value_counts()
```

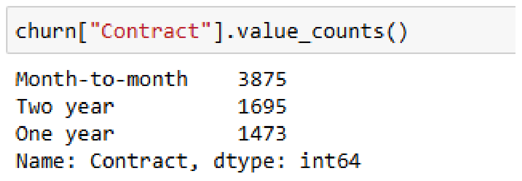
```
No     4072
Yes    2971
Name: MultipleLines, dtype: int64
```

These commands use the value_counts() function to summarize how many customers fall into each category within the “PhoneService” and “MultipleLines” columns of the dataset. For “PhoneService”, the output shows that 6,361 customers have phone service, while 682 do not, indicating that the majority of customers are subscribed to a phone service. In the “MultipleLines” column, 4,072 customers do not have multiple lines, while 2,971 customers do, suggesting that multiple line usage is fairly common but not as widespread as single-line service.



```
churn["InternetService"].value_counts()
```

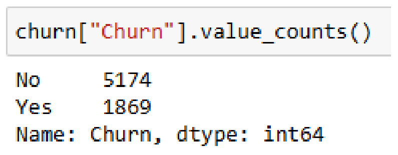
```
DSL            3947
Fiber optic    3096
Name: InternetService, dtype: int64
```



```
churn["Contract"].value_counts()
```

```
Month-to-month    3875
Two year          1695
One year          1473
Name: Contract, dtype: int64
```

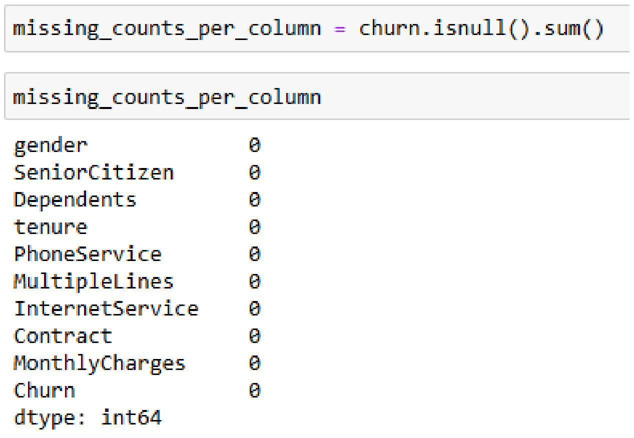
These commands use the value_counts() function to display the distribution of customers based on their InternetService and Contract types. The output shows that 3,947 customers use DSL, while 3,096 use Fiber optic, indicating that both internet services are popular but DSL has a slightly higher usage. For Contract, 3,875 customers have month-to-month plans, 1,695 have two-year contracts, and 1,473 have one-year contracts, showing that most customers prefer short-term, flexible agreements.



```
churn["Churn"].value_counts()
```

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

This command uses the value_counts() function to show the distribution of customer churn in the dataset. The output indicates that 5,174 customers did not churn ("No") and 1,869 customers did churn ("Yes"), meaning that the majority of customers remained with the company. This reveals an imbalance in the target variable, where non-churning customers significantly outnumber churning ones. Understanding this distribution is important because it can affect model training and evaluation, requiring techniques such as resampling or weighting to ensure accurate churn prediction.



```
missing_counts_per_column = churn.isnull().sum()
```

```
missing_counts_per_column
```

```
gender             0
SeniorCitizen      0
Dependents         0
tenure             0
PhoneService       0
MultipleLines      0
InternetService    0
Contract           0
MonthlyCharges     0
Churn              0
dtype: int64
```

This code checks for missing values in each column of the churn DataFrame using isnull().sum(), which counts the number of null (NaN) entries per column. The output shows that all columns have 0 missing values, indicating that the dataset is complete and contains no null entries. This is important because it ensures that no additional data cleaning is required to handle missing values before performing analysis or training machine learning models.

Data Clean:

```python
for c in ["tenure","MonthlyCharges","SeniorCitizen"]:
    if c in churn.columns:
        churn[c] = pd.to_numeric(churn[c], errors="coerce")
```

The parameter errors="coerce" forces non-convertible values (such as "N/A", "unknown", or other text) to be replaced with NaN, representing missing data. This ensures that all values in numeric columns are properly converted to numeric types, allowing accurate mathematical operations and data scaling in subsequent steps.

```python
num_cols = churn.select_dtypes(include=["number"]).columns.tolist()
for c in num_cols:
    churn[c] = churn[c].fillna(churn[c].median())
```

This step selects all numeric columns in the DataFrame and replaces any missing values (NaN) in those columns with the median value of each respective column. Filling in missing values ensures that the dataset remains complete and prevents potential errors during the model training process. The median is chosen instead of the mean because it is less sensitive to outliers, providing a more accurate and robust representation of the data when extreme values are present.

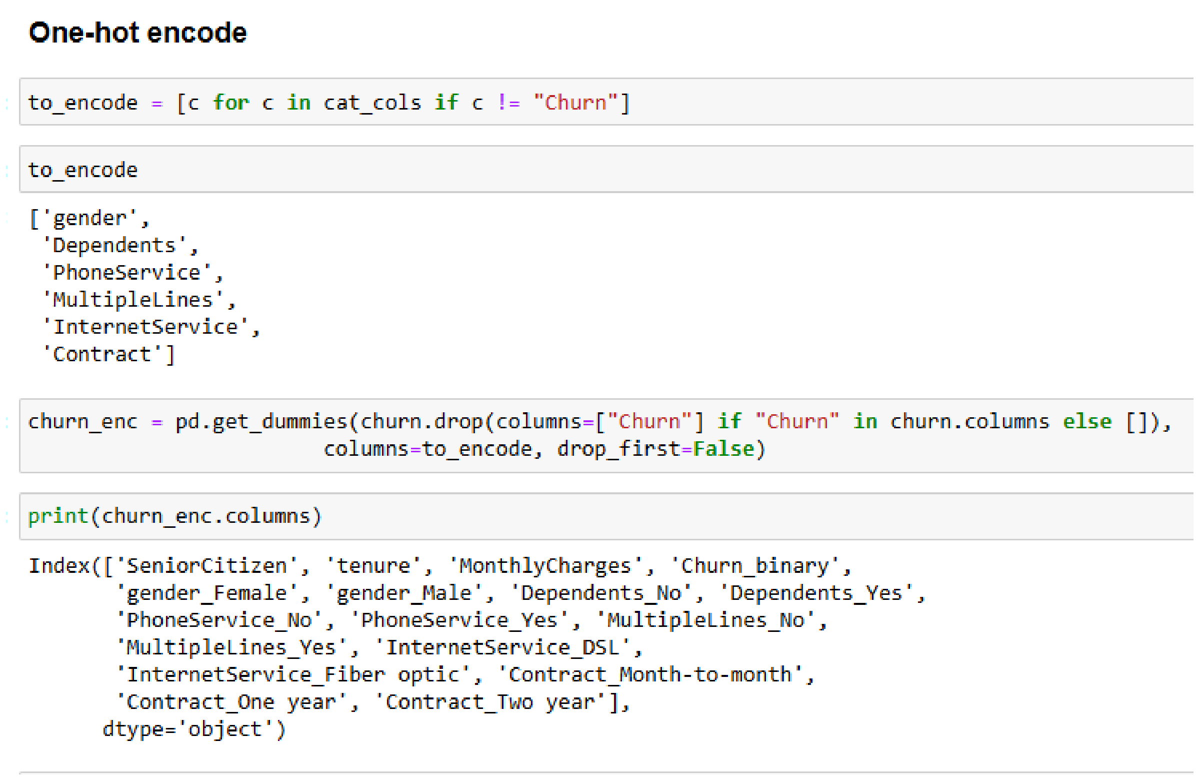
```python
cat_cols = churn.select_dtypes(include=["object","category"]).columns.tolist()
for c in cat_cols:
    churn[c] = churn[c].fillna("Missing")
```

This step identifies all columns containing categorical or text data (such as “InternetService”, “Contract”, or “Gender”) and replaces any missing values with the string “Missing”. This ensures there are no null values before encoding, as most machine learning algorithms cannot process missing text data, while also preserving information about which entries originally had missing values.

```python
if "Churn" in churn.columns:
    churn["Churn_binary"] = churn["Churn"].map({"Yes":1,"No":0}).fillna(0).astype(int)
```

This step checks whether the “Churn” column exists and converts its text values “Yes” and “No” into numeric values 1 and 0, respectively. Any missing values are filled with 0, treating them as “No”, and the resulting column is ensured to be of integer type. This transformation is necessary because machine learning models require numeric target variables, and it creates a new column, Churn_binary, for use in modeling and analysis.

One-hot Encode:



One-hot encode

```
to_encode = [c for c in cat_cols if c != "Churn"]
```

```
to_encode
```

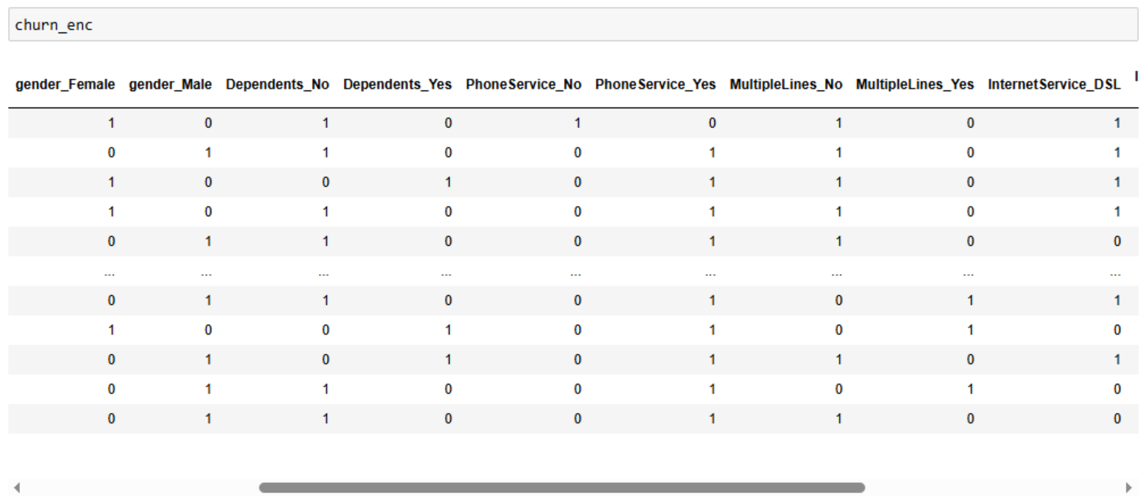
```
['gender',
 'Dependents',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'Contract']
```

```
churn_enc = pd.get_dummies(churn.drop(columns=["Churn"] if "Churn" in churn.columns else []),
                           columns=to_encode, drop_first=False)
```

```
print(churn_enc.columns)
```

```
Index(['SeniorCitizen', 'tenure', 'MonthlyCharges', 'Churn_binary',
       'gender_Female', 'gender_Male', 'Dependents_No', 'Dependents_Yes',
       'PhoneService_No', 'PhoneService_Yes', 'MultipleLines_No',
       'MultipleLines_Yes', 'InternetService_DSL',
       'InternetService_Fiber optic', 'Contract_Month-to-month',
       'Contract_One year', 'Contract_Two year'],
      dtype='object')
```

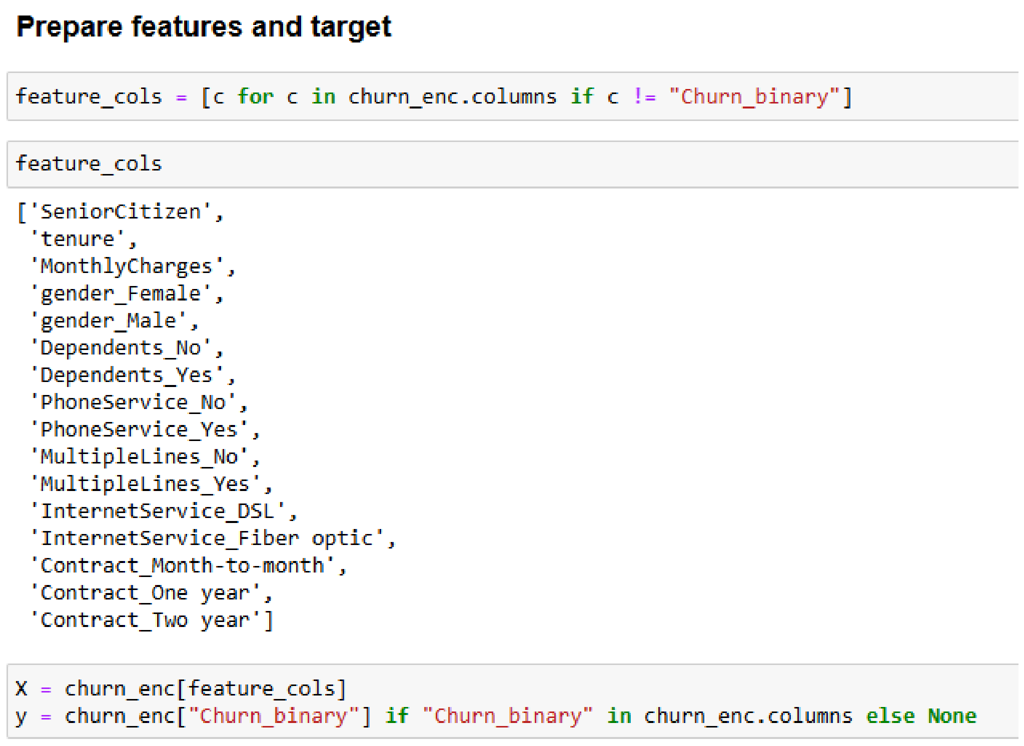
This step performs one-hot encoding on all categorical features in the dataset except the "Churn" column. It first identifies which columns are categorical and stores them in the variable to_encode. Then, using the pd.get_dummies() function, it converts each categorical column into multiple binary (0 or 1) columns, one for each unique category. This process allows machine learning models to interpret categorical data numerically without implying any ordinal relationship between categories. The "Churn" column is excluded from encoding to preserve it as the target variable for prediction. The output is shown below.



```
churn_enc
```

gender_Female	gender_Male	Dependents_No	Dependents_Yes	PhoneService_No	PhoneService_Yes	MultipleLines_No	MultipleLines_Yes	InternetService_DSL
1	0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0	1
1	0	0	1	0	1	1	0	1
1	0	1	0	0	1	1	0	1
0	1	1	0	0	1	1	0	0
...	...	...	...	...	...	...	...	...
0	1	1	0	0	1	0	1	1
1	0	0	1	0	1	0	1	0
0	1	0	1	0	1	1	0	1
0	1	1	0	0	1	0	1	0
0	1	1	0	0	1	1	0	0

Prepare Features and Target:



Prepare features and target

```python
feature_cols = [c for c in churn_enc.columns if c != "Churn_binary"]
```

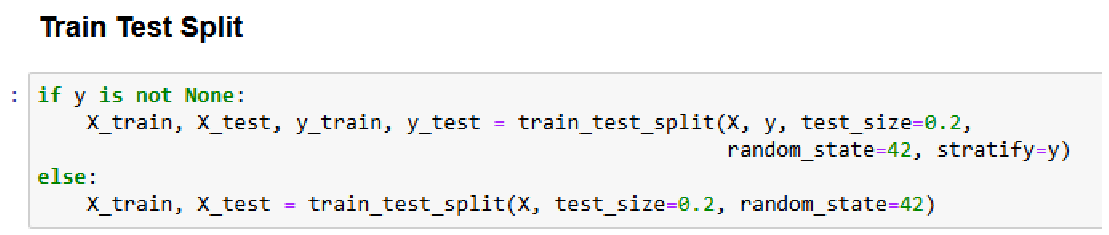
```python
feature_cols
```

```
['SeniorCitizen',
 'tenure',
 'MonthlyCharges',
 'gender_Female',
 'gender_Male',
 'Dependents_No',
 'Dependents_Yes',
 'PhoneService_No',
 'PhoneService_Yes',
 'MultipleLines_No',
 'MultipleLines_Yes',
 'InternetService_DSL',
 'InternetService_Fiber optic',
 'Contract_Month-to-month',
 'Contract_One year',
 'Contract_Two year']
```

```python
X = churn_enc[feature_cols]
y = churn_enc["Churn_binary"] if "Churn_binary" in churn_enc.columns else None
```

This step prepares the dataset for modeling by separating the features and the target variable. All columns except “Churn_binary” are selected as input features and stored in X, while the “Churn_binary” column is assigned to y as the target variable for prediction. If the “Churn_binary” column does not exist, y is set to None. This separation ensures that the machine learning model can clearly distinguish between input data and the outcome it needs to predict.

Train Test Split:

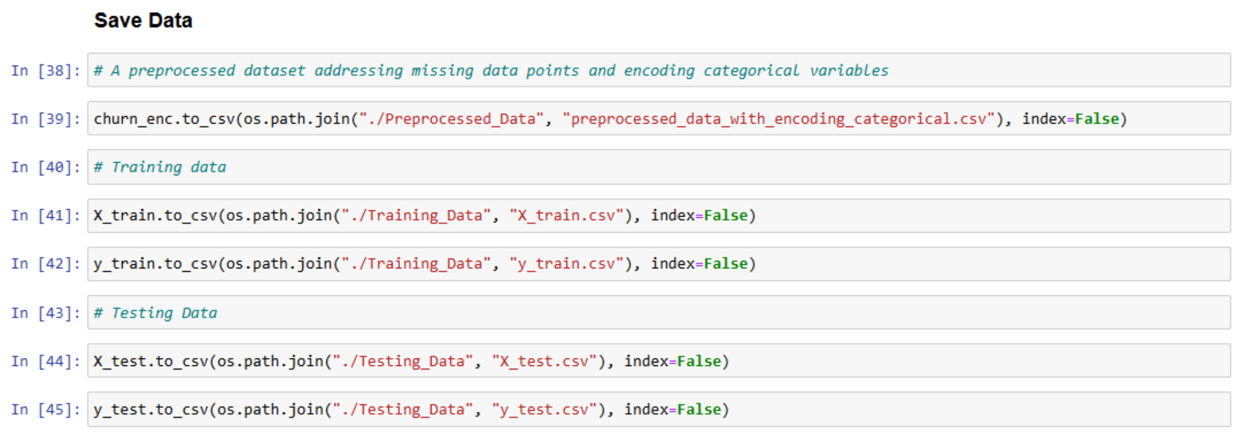


Train Test Split

```
: if y is not None:
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                          random_state=42, stratify=y)
  else:
      X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)
```

This step splits the dataset into training and testing sets to evaluate the model's performance. If the target variable y exists, both the features (X) and target (y) are split, with 20% of the data reserved for testing. The "stratify=y" parameter ensures that the proportion of classes in the target variable is maintained in both training and testing sets. If y does not exist, only the features X are split into training and testing sets. The random_state=42 ensures the split is reproducible.

Save Data :



Save Data

```
In [38]: # A preprocessed dataset addressing missing data points and encoding categorical variables

In [39]: churn_enc.to_csv(os.path.join("./Preprocessed_Data", "preprocessed_data_with_encoding_categorical.csv"), index=False)

In [40]: # Training data

In [41]: X_train.to_csv(os.path.join("./Training_Data", "X_train.csv"), index=False)

In [42]: y_train.to_csv(os.path.join("./Training_Data", "y_train.csv"), index=False)

In [43]: # Testing Data

In [44]: X_test.to_csv(os.path.join("./Testing_Data", "X_test.csv"), index=False)

In [45]: y_test.to_csv(os.path.join("./Testing_Data", "y_test.csv"), index=False)
```

This step saves the training and testing datasets to separate CSV files for future use. The feature and target variables for the training set (X_train and y_train) are saved in the ./Training_Data folder, while the corresponding testing set (X_test and y_test) is saved in the ./Testing_Data folder. The index=False parameter ensures that row indices are not included in the saved files, keeping the data clean and ready for modeling or analysis.