LIBRARIES

*# Import necessary libraries*

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

1. PANDAS

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008. Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

2. NUMPY

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. NumPy stands for Numerical Python.

Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

3. SEABORN

Seaborn is a library for making statistical graphics in Python. It builds on top of [matplotlib](https://matplotlib.org/) and integrates closely with [pandas](https://pandas.pydata.org/) data structures.

Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

4. MATPLOTLIB

Matplotlib is a powerful and very popular data visualization library in Python. In this tutorial, we will discuss how to create line plots, bar plots, and scatter plots in Matplotlib using stock market data in 2022. These are the foundational plots that will allow you to start understanding, visualizing, and telling stories about data. Data visualization is an essential skill for all data analysts and Matplotlib is one of the most popular libraries for creating visualizations.

5. SCIKIT-LEARN

Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

6. JOBLIB

Joblib is a Python library that simplifies parallel processing, result caching, and task distribution, particularly useful for computationally intensive tasks and large datasets, offering a lightweight pipelining solution.

7. XGBoost

XGBoost (eXtreme Gradient Boosting) is a powerful, open-source machine learning library that uses gradient boosted decision trees, a supervised learning boosting algorithm, known for its speed, efficiency, and ability to scale well with large datasets.

*# Import necessary libraries*

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.preprocessing **import** LabelEncoder, StandardScaler

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.ensemble **import** RandomForestClassifier

**from** xgboost **import** XGBClassifier

**from** sklearn.metrics **import** accuracy\_score, classification\_report, confusion\_matrix

CODE EXPLANATION

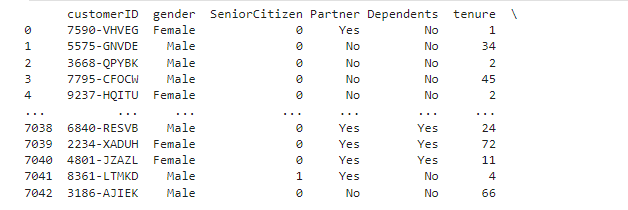
1. .pd**.**read\_csv

The pandas read\_csv function is one of the most commonly used pandas functions, particularly for data preprocessing. It is invaluable for tasks such as importing data from CSV files into the Python environment for further analysis. This function is capable of reading a CSV file from both your local machine and from a URL directly. What’s more, using pandas to read csv files comes with a plethora of options to customize your data loading process to fit your specific needs.

*# Load the dataset*

df **=** pd**.**read\_csv(r"C:\Users\HP\Desktop\tec/WA\_Fn-UseC\_-Telco-Customer-Churn.csv")

print(df)



2. **.**dropna()

The dropna() method removes the rows that contains NULL values. The dropna() method returns a new DataFrame object unless the inplace parameter is set to True, in that case the dropna() method does the removing in the original DataFrame instead.

*#Remove Rows*

new\_df **=** df**.**dropna()

print(new\_df**.**to\_string())

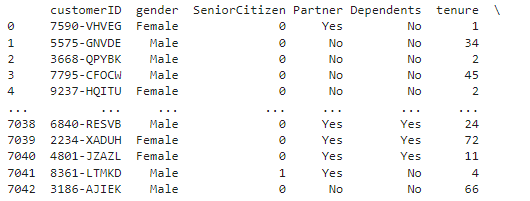
3. .fillna()

The fillna() method replaces the NULL values with a specified value. The fillna() method returns a new DataFrame object unless the inplace parameter is set to True, in that case the fillna() method does the replacing in the original DataFrame instead.

*#Replace Empty Values*

new\_df**.**fillna(130, inplace **=** **True**)

print(new\_df)

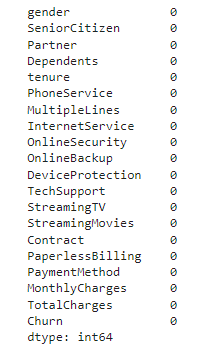


4. **.**isnull()**.**sum()

To check this, we can use the function **dataframe.isnull()** in pandas. It will return True for missing components and False for non-missing cells. However, when the dimension of a dataset is large, it could be difficult to figure out the existence of missing values. In general, we may just want to know if there are any missing values at all before we try to find where they are. The function **dataframe.isnull().values.any()**returns True when there is at least onemissing value occurring in the data. The function **dataframe.isnull().sum().sum()**returns the number of missing values in the dataset.

*# Check for missing values*

print(new\_df**.**isnull()**.**sum())



5. .drop() Method:

This method is a core function for modifying DataFrames by deleting specific rows or columns.

*# Drop customerID as it is not useful for prediction*

new\_df**.**drop(columns**=**['customerID'], inplace**=True**)

print(new\_df**.**to\_string())

6.  *Convert categorical variables to numerical using Label Encoding*

le **=** LabelEncoder()

categorical\_columns **=** new\_df**.**select\_dtypes(include**=**['object'])**.**columns

**for** col **in** categorical\_columns:

new\_df[col] **=** le**.**fit\_transform(new\_df[col])

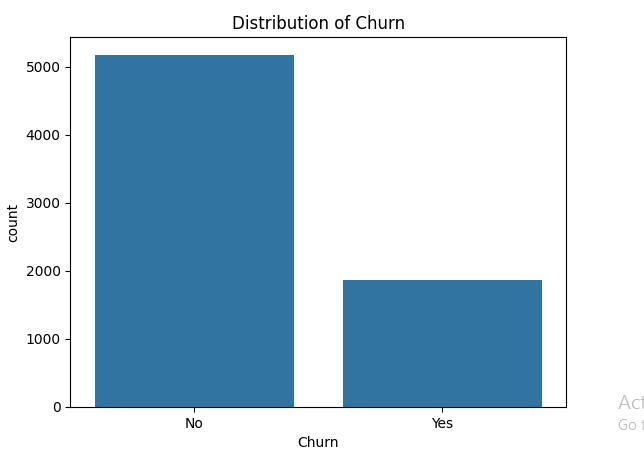
*7.* The plot() function is used to draw points (markers) in a diagram. By default, the plot() function draws a line from point to point. The function takes parameters for specifying points in the diagram. Parameter 1 is an array containing the points on the **x-axis**.

*# Distribution of Churn*

sns**.**countplot(x**=**'Churn', data**=**df)

plt**.**title('Distribution of Churn')

plt**.**show()



*8. # Random Forest*

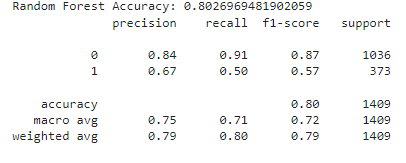
rf\_model **=** RandomForestClassifier()

rf\_model**.**fit(X\_train, y\_train)

y\_pred\_rf **=** rf\_model**.**predict(X\_test)

print("Random Forest Accuracy:", accuracy\_score(y\_test, y\_pred\_rf))

print(classification\_report(y\_test, y\_pred\_rf))



*9. # XGBoost*

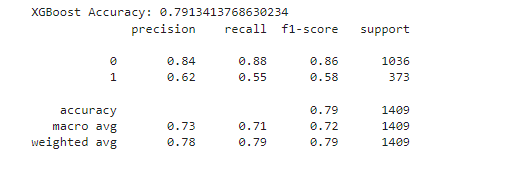
xgb\_model **=** XGBClassifier()

xgb\_model**.**fit(X\_train, y\_train)

y\_pred\_xgb **=** xgb\_model**.**predict(X\_test)

print("XGBoost Accuracy:", accuracy\_score(y\_test, y\_pred\_xgb))

print(classification\_report(y\_test, y\_pred\_xgb))



*10. # Confusion Matrix for Random Forest*

conf\_matrix **=** confusion\_matrix(y\_test, y\_pred\_rf)

sns**.**heatmap(conf\_matrix, annot**=True**, fmt**=**'d', cmap**=**'Blues')

plt**.**title('Confusion Matrix for Random Forest')

plt**.**show()

**import** joblib

joblib**.**dump(rf\_model, 'churn\_prediction\_model.pkl')

11. # Correlation Heatmap

plt.figure(figsize=(12, 8))

sns.heatmap(new\_df.corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap')

plt.show()

