Telecom Churn Prediction using Machine learning

A report submitted in fulfilment of the requirements of the Internship project

BY

SHANTI KUMARI

Under the guidance of Infosys Springboard mentor

BHASKAR SIR

INFOSYS SPRINGBOARD INTERNSHIP 4.0

Acknowledgement

I would like to express my sincere gratitude to my mentor, Bhaskar Sir and Sudheer Kumar Sir for his invaluable guidance and support throughout the course of this project. His expertise and insights have been instrumental in shaping the direction and outcome of this study. I am also thankful to Infosys Springboard for providing me with this opportunity to undertake the internship and work on a project of significant relevance. Finally, I extend my appreciation to my colleagues and peers who have contributed their time and knowledge to assist me in various aspects of this project.

Thank You

Date: Shanti Kumar

Abstract

This project report presents a comprehensive study on Telecom Churn Prediction using machine learning techniques. The primary objective of the project is to develop a predictive model that can accurately identify customers who are likely to churn, thereby enabling telecom companies to implement targeted retention strategies. The project involves several key milestones, including software and library setup, data collection, data pre-processing, exploratory data analysis, and model building and evaluation. Various machine learning algorithms, including Logistic Regression, Decision Tree Classifier, and Random Forest, are employed and optimized through hyperparameter tuning to achieve the best predictive performance. The results of this study highlight the importance of churn prediction in the telecom industry and demonstrate the effectiveness of machine learning models in addressing this critical business problem.

Content

7. Milestone 4: Model Comparison and Selection

1. Introduction:

Customer churn, the phenomenon of customers discontinuing their use of a service, is a critical issue faced by many industries, particularly the telecom sector. High churn rates can significantly impact a company's revenue and market position. Understanding the reasons behind customer churn and predicting which customers are likely to churn can help companies implement targeted strategies to retain these customers and improve their overall business performance.

In the highly competitive telecom industry, retaining existing customers is more cost-effective than acquiring new ones. Loyal customers are not only more likely to continue using the company's services but also to recommend them to others, contributing to the company's growth. Therefore, accurately predicting customer churn and taking preemptive actions to mitigate it is essential for maintaining a stable customer base and ensuring sustained profitability.

This project aims to develop a predictive model to identify customers who are at risk of churning. By analyzing historical customer data, including demographics, usage patterns, service plan details, payment history, and customer interactions, the model will uncover patterns and indicators that are associated with churn. The insights gained from this analysis will enable the telecom company to understand the key factors driving churn and to devise effective retention strategies.

By implementing a robust churn prediction model, the telecom company will be able to proactively address the issue of customer churn. This will not only help in retaining valuable customers but also in improving customer satisfaction and loyalty. Ultimately, the project aims to enhance the company's ability to maintain a stable customer base and achieve long-term success in the competitive telecom market.

2. Importance of Churn Prediction:

Customer churn prediction is crucial for businesses, especially in highly competitive industries like telecom, where acquiring new customers can be significantly more expensive than retaining existing ones. Predicting churn allows companies to take proactive measures to retain customers, thereby stabilizing and potentially increasing their revenue streams.

2.1. Advantages of Churn Prediction

- **Increased Customer Retention:**
  - *Proactive Engagement*: Identifying at-risk customers allows companies to engage with them proactively, offering tailored solutions and incentives to retain them.
  - *Personalized Marketing*: Churn prediction models enable personalized marketing campaigns, which are more effective in addressing the specific needs and concerns of individual customers.

- **Cost Savings:**
  - *Reduced Acquisition Costs*: Retaining an existing customer is often cheaper than acquiring a new one. By reducing churn, companies can lower their customer acquisition costs.
  - *Efficient Resource Allocation*: Resources can be allocated more efficiently by focusing retention efforts on customers most likely to churn.

- **Competitive Advantage:**
  - *Market Position*: Companies that effectively manage and reduce churn can gain a competitive advantage in the market, maintaining a loyal customer base and attracting new customers through positive word-of-mouth.

- **Data-Driven Decision Making:**
  - *Strategic Planning*: Churn prediction provides valuable data that can inform strategic decisions, helping companies plan better retention and customer engagement strategies.
  - *Informed Interventions*: Targeted interventions based on churn predictions are more likely to be effective, as they are based on data-driven insights.

2.2. Limitations of Churn Prediction

- **Data Quality and Availability:**
  - *Incomplete Data*: Accurate churn prediction relies heavily on the

quality and completeness of the data. Missing or inaccurate data can lead to incorrect predictions.

- *Data Privacy*: Collecting and using customer data for churn prediction must comply with data privacy regulations, which can limit the extent of data collection and analysis.

- **Model Limitations:**

  - *Complexity*: Advanced machine learning models can be complex and require significant expertise to develop and maintain.
  - *Accuracy*: No model can predict churn with 100% accuracy. There will always be false positives (predicting churn when the customer does not churn) and false negatives (failing to predict actual churn).

- **Resource Intensive:**

  - *Cost*: Developing, implementing, and maintaining churn prediction models can be resource-intensive, requiring investments in technology and skilled personnel.
  - *Time:* Building an effective churn prediction model takes time, from data collection and preprocessing to model training and validation.

- **Changing Dynamics:**

  - *Market Change*: Customer behavior and market conditions can

change over time, requiring continuous updates to the churn prediction model to maintain its accuracy.

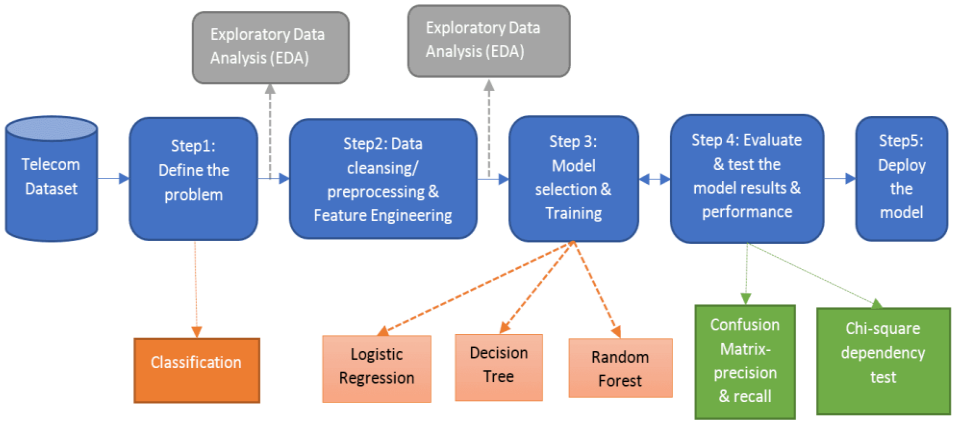
- *New Competitors*: The entry of new competitors or changes in existing competitors' strategies can affect churn rates, necessitating frequent reassessment of the model**.**

3. Problem Statement: Churn Prediction:

Customer churn, the phenomenon where customers stop using a company's product or service, is a critical issue for businesses, especially in highly competitive industries like telecommunications. High churn rates can significantly impact revenue and profitability, making it essential for companies to identify and retain customers at risk of leaving. Predicting customer churn allows businesses to implement targeted retention strategies, improving customer satisfaction and loyalty while reducing the costs associated with acquiring new customers.

Objective:

The primary objective of this project is to develop a predictive model that accurately identifies customers who are likely to churn. By leveraging historical customer data, the model aims to detect patterns and behaviors indicative of potential churn, enabling the company to proactively address the issue and retain more customers.



4. Milestone 1: Software and Library Setup, Data Collection

The objective of this milestone is to set up the necessary software and libraries required for data analysis and machine learning tasks.

Steps:

1. Software Installation: Ensure that the required software, such as Python and collab Notebook, is installed on our system.
2. *Library Installation*: Install the Python libraries necessary for data analysis and machine learning. This includes libraries like pandas, numpy, scikit-learn, matplotlib, and seaborn.
3. *Data Gathering*: Gather the dataset(s) required for analysis and machine learning tasks. This could involve downloading datasets from online repositories or acquiring them from other sources.

Task Breakdown:

- *Install Python and collab Notebook*: download and install Python and collab Notebook to our system.

- *Install Required Libraries*: Open a command prompt or terminal and use pip to install the necessary libraries. For example: pip install pandas, scikit-learn, matplotlib, seaborn
- *Gather Dataset*: Download the dataset required for the analysis. Ensure that the dataset is in a format compatible with pandas, such as CSV or Excel.

Github link of dataset-Telecom_churn_prediction_csv

Output:

Once this milestone is complete, we should have:

- Python and collab Notebook installed on our system.
- Required libraries installed and accessible.
- The dataset downloaded and ready for analysis in a format compatible
- with pandas

5. Milestone 2: Data Pre-processing and Exploratory Data Analysis (EDA)

The objective of this milestone is to prepare the dataset for machine learning by performing data pre-processing tasks and gaining insights through exploratory data analysis (EDA).

5.1. Checking Data Normality:

Checking whether data follows a normal distribution is essential for several reasons:

- *Validity of Statistical Tests*: Many statistical techniques, such as t-tests, ANOVA, and linear regression, assume that the data is normally distributed. Failing to meet this assumption can lead to incorrect conclusions from these analyses. By assessing normality, we ensure the validity of subsequent statistical tests.

- *Selection of Appropriate Statistical Methods*: If the data is not normally distributed, alternative methods or transformations may be required. For instance, non-parametric tests or specialized transformations (e.g., logarithmic or Box-Cox) can be employed to accommodate non-normal data. Identifying non-normality early in the analysis process allows researchers to choose the most suitable statistical approach.

- *Interpretation of Results*: Knowing the distribution of the data aids in the interpretation of results. For example, if the data is skewed or exhibits heavy tails, medians may be more appropriate measures of central tendency than means. Understanding the data's distribution ensures that results are interpreted correctly and conclusions are sound.

- *Assumption Checking*: Assessing normality is part of the broader practice of assumption checking in statistics. By confirming whether the data meets the normality assumption, researchers demonstrate rigor in their analysis and enhance the reliability of their findings.

- *Data Pre-processing Guidance*: Identifying non-normality prompts researchers to consider data pre-processing techniques such as outlier removal, data transformation, or the use of robust statistical methods. This proactive approach improves the quality of the data analysis pipeline and enhances the reliability of research findings.

Output:

Column	Normality Test Result
s6.new.rev.p2.m2	Not normally distributed (p-value=0.0000)
s1.new.rev.m1	Not normally distributed (p-value=0.0000)
s3.og.rev.4db.p5	Not normally distributed (p-value=0.0000)
s3.new.rev.4db.p5	Not normally distributed (p-value=0.0000)
s4.usg.ins.p2	Not normally distributed (p-value=0.0000)
...	...
s1.rev.p2	Not normally distributed (p-value=0.0000)

To check if the data follows a normal distribution, we start by importing the necessary libraries: pandas for data manipulation and the Shapiro-Wilk test from scipy.stats. Next, we define a function called `check_normality` that takes a Data Frame (`df`) as input. Within this function, we iterate through each column of the DataFrame. For each column, we compute the Shapiro-Wilk test statistic and p-value. We set a significance level (`alpha`) of 0.05. If the p-value is greater than the significance level, we conclude that the data in that column is normally distributed and print a message indicating so, along with the p-value. Otherwise, if the p-value is less than or equal to the significance level, we determine that the

data is not normally distributed and print a corresponding message. This function allows us to quickly assess the normality of each column in the Data Frame.

5.2. Rectifying Misclassified Data Types:

Columns in a DataFrame that do not match the expected data type (e.g., `float64` or `int64`), possibly due to incorrect data entry or improper data import, which can affect data processing and analysis.

Before Correction	After Correction
Number of misclassified columns(float): 31	Number of misclassified columns: 0
Number of misclassified columns(int): 80	Number of misclassified columns: 0

**Understanding Data Types:

- Initially, we inspect the data types of the variables in our DataFrame to identify any misclassifications.
- By printing `df.dtypes`, we obtain an array of data types, where `float64` and `int64` are the primary types.

**Identifying Misclassifications:

- To ascertain the extent of misclassification, we employ a function `count_misclassified_columns(df)`, which calculates the number of columns misclassified from their intended data type.
- The output, in this case, reveals that 31 columns are misclassified as float types, and 80 columns are misclassified as integer types.

**Correcting Misclassifications:

- Subsequently, we utilize `convert_misclassified_columns(df)` to rectify the misclassified columns. This function identifies the misclassified columns, converts them to the correct data type (`float64` in this scenario), and returns the corrected DataFrame. Upon executing this function, we achieve a DataFrame (`df_corrected`) where all misclassifications have been addressed, resulting in `Number of misclassified columns: 0`.

****Advantages of Correcting Data Types:**

- Ensuring that data types are correctly classified is crucial for various data processing tasks. Correctly typed data facilitates accurate analysis, prevents errors during computations, and enables compatibility with functions and libraries expecting specific data types.

****Effects:**

By rectifying misclassified data types, we mitigate potential errors in calculations and analyses, ensuring the reliability and accuracy of subsequent data processing steps. This process enhances the overall quality of the dataset and fosters confidence in the integrity of analytical outcomes.

5.3. Removing Duplicate Values:

Rows in a DataFrame that contain identical data across all columns, which can lead to skewed analysis and inaccurate results if not handled properly.

- *Counting Duplicate Records*: The initial step involves counting the number of duplicate records present in the dataset. This is achieved by using the `duplicated()` function followed by the `sum()` function to count the occurrences of `True` values, indicating duplicate records.
- *Removing Duplicate Records*: After determining the presence of duplicate records, they are removed from the dataset using the `drop_duplicates()` function. This function removes duplicate rows based on all columns by default. Setting the `inplace` parameter to `True` ensures that the changes are made directly to the original DataFrame.

Before Removal	**After Removal**
Number of duplicate records: 0	Number of duplicate records: 0

**Advantages of Removing Duplicate Values:

Removing duplicate values from a dataset offers several advantages. Firstly, it improves the accuracy and reliability of the data by eliminating redundant information. This ensures that analysis and modeling are based on a clean and representative dataset. Secondly, it enhances computational efficiency, as processing fewer unique records can reduce the time and resources required for tasks such as data manipulation, analysis, and modeling. Additionally, removing duplicates can prevent biased results in statistical analyses and machine learning algorithms, leading to more robust and meaningful insights.

Resulting Effects: After removing duplicate records, the dataset is streamlined to contain only unique instances. This means that each observation

in the dataset represents a distinct entity or event, without any redundant entries. Consequently, the total number of records in the dataset may decrease, reflecting the elimination of duplicate values. By reducing data redundancy, the quality and reliability of subsequent analyses and modeling tasks are enhanced.

5.4. Removing Unique Value Variables:

Variables (columns) in a dataset where every value occurs only once, indicating that these variables do not provide any useful information for analysis and can be removed to simplify the dataset.

Step	**Result**
Counting Unique Value Variables	Number of unique value variables: 0
Removing Unique Value Variables	DataFrame with unique value variables removed:
	[Updated DataFrame without unique value variables]

**Advantages of Removing Unique Value Variables:

Removing variables with unique values from a dataset provides several benefits. Firstly, it helps in reducing the dimensionality of the dataset, which can lead to improved model performance and interpretability, especially in machine learning algorithms where high dimensionality can cause overfitting. Secondly, it eliminates variables that do not contribute to the variability or patterns in the data, streamlining subsequent analyses and modeling processes. Additionally, removing unique value variables can enhance computational efficiency by reducing the computational burden associated with processing irrelevant features.

**Resulting Effects:

After removing variables with unique values, the dataset is refined to exclude columns where every value is unique. As a result, the total number of columns in the dataset may decrease, reflecting the removal of variables with no meaningful variability. This refinement improves the focus of subsequent analyses and models by ensuring that only relevant and informative features are retained.

**Steps Involved:

- Counting Unique Value Variables: Initially, the number of variables with unique values in the dataset is counted. This is achieved by comparing the number of unique values in each column to the total number of rows in the DataFrame.
- Removing Unique Value Variables: Next, variables with unique values are identified and removed from the dataset. This process involves iterating through each column and checking if the number of unique values equals the total number of rows. Columns meeting this criterion are considered unique value variables and are subsequently removed from the DataFrame.

5.5. Removing Zero Variance Variables:

Variables (columns) in a dataset where all values are identical, resulting in zero variability and offering no useful information for analysis. Removing such variables helps streamline the dataset and improves computational efficiency.

Step	Result
Checking Zero Variance Variables	Zero variance variables: []
Removing Zero Variance Variables	DataFrame with zero variance variables removed:
	[Updated DataFrame with zero variance variables removed]

****Advantages of Removing Zero Variance Variables:**

Removing variables with zero variance from a dataset offers several advantages in data analysis and modeling. Firstly, it eliminates features that do not exhibit any variability across observations, which are unlikely to contribute meaningful information to the analysis or predictive models. This enhances the efficiency of subsequent modeling processes by reducing computational overhead and complexity associated with processing irrelevant features. Additionally, removing zero variance variables helps in improving the interpretability of models by focusing on features that exhibit variation across the dataset, thereby enhancing the understanding of underlying patterns and relationships.

****Resulting Effects:**

After removing zero variance variables, the dataset is refined to exclude columns where the variance of values is zero. As a result, the total number of columns in the dataset may decrease, reflecting the removal of features with no discernible variation. This refinement streamlines subsequent analyses and modeling tasks by ensuring that only informative and meaningful features are retained for further exploration and modeling.

**Steps Involved:

- *Zero Variance Variables*: Initially, the function checks for variables with zero variance by computing the variance of values in each column. Columns with zero variance are identified and stored in a list.

- *Removing Zero Variance Variables*: Next, the identified zero variance variables are removed from the dataset. This process involves iterating through each column and dropping columns with only a single unique value, indicating zero variance.

5.6. Treating Outliers:

Step	Result
Identifying Outliers using Z-score	Outliers identified using z-score method: [7388 rows x 111 columns]
Removing Outliers	Cleaned data after outlier removal: [25000 rows x 111 columns]
Standardizing Data	Scaled data mean: [1.79429984e-17 1.79429984e-17] Scaled data std: [1. 1.]

**Advantages of Outlier Treatment:

Treating outliers in a dataset is essential for ensuring the robustness and reliability of statistical analyses and machine learning models. By identifying and addressing outliers, we can improve the accuracy and generalizability of our models. Here are some key advantages:

- *Improved Model Performance*: Outliers can significantly affect the performance of statistical models by skewing parameter estimates and reducing predictive accuracy. By removing or transforming outliers, we can mitigate their impact and produce more reliable model predictions.

- *Enhanced Data Interpretation*: Outliers can distort the interpretation of data patterns and relationships. Removing outliers allows for a clearer understanding of the underlying data structure and facilitates more accurate insights and decision-making.

- *Increased Model Robustness*: Models trained on datasets containing outliers may be less robust when applied to new data. By cleaning the data of outliers, we can build models that are more resilient to variations in input data and better generalize to unseen instances.

**Resulting Effects:

After treating outliers using various methods such as z-score normalization, Box-Cox transformation, and removal based on statistical thresholds, the dataset undergoes significant changes. Outliers may be identified and removed, while data distributions may be transformed to achieve better conformity to statistical assumptions.

**Box-Cox Transformation and Outlier Removal:

- *Purpose*: Applying the Box-Cox transformation to normalize data

and removing outliers based on the interquartile range (IQR) after transformation.

- *Explanation*: This section transforms numeric columns using the Box-Cox transformation, which converts non-normally distributed data into approximately normal distribution. Outliers are then identified and removed using the IQR method after transformation, ensuring the data's robustness for further analysis.

**Steps Involved:

- *Identifying Outliers using Z-score*: The z-score method calculates the deviation of each data point from the mean in terms of standard deviations. Data points with z-scores exceeding a predefined threshold are identified as outliers.
- *Removing Outliers*: Identified outliers are either removed or transformed using techniques like Box-Cox transformation to bring them within acceptable ranges. Outliers can be clipped based on statistical thresholds such as the interquartile range (IQR) or z-score.

```
q1 = data[col].quantile(0.25)
q3 = data[col].quantile(0.75)
IQR = q3 - q1
lower_bound = q1 - 1.5 * IQR
upper_bound = q3 + 1.5 * IQR
```

- *Standardizing Data*: After outlier treatment, the data may undergo standardization to ensure uniform scaling across features. Standardization

involves transforming the data to have a mean of zero and a standard deviation of one, making it suitable for modeling algorithms that require standardized input.

5.7. Missing Value Treatment:

Entries in a dataset that are blank or undefined, which can hinder analysis and interpretation. Imputing missing values helps maintain data integrity and ensures accurate analysis results.

Step	Result
Identifying Missing Values	Number of missing values in each column: [0, 0, 0, 0, 0, 0, 0, 0, 314, 0, 9131, 0, 0, 0, 0, 1622, 1512, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 178, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2696, 0, 480, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2679, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 685, 0, 0, 0, 338, 0, 0, 0, 0, 0, 719, 0, 0, 0, 0, 0, 0, 0, 0, 274, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Missing Value Imputation	DataFrame with missing values imputed using median: After this: Number of missing values in each column: [0, 0]

**Advantages of Missing Value Treatment:

Missing value treatment is a crucial step in data pre-processing that ensures the integrity and accuracy of the dataset. By handling missing values appropriately,

we can prevent biases in statistical analyses and machine learning models. Here are some key advantages:

- Improved Data Quality: Addressing missing values helps maintain the quality and completeness of the dataset, reducing the risk of introducing errors or biases into subsequent analyses.
- Enhanced Model Performance: Models trained on datasets with missing values may produce unreliable or biased results. By imputing or removing missing values, we can enhance the performance and predictive accuracy of machine learning models.
- Better Data Interpretation: Missing values can obscure patterns and relationships in the data, leading to erroneous conclusions. By treating missing values, we can obtain more accurate insights and interpretations from the data.

**Resulting Effects:

After missing value treatment, the dataset undergoes significant changes, with missing values being either imputed or removed based on predefined strategies. The resulting dataset is more completed and suitable for subsequent analyses or model building.

**Steps Involved:

- Identifying Missing Values: The number of missing values in each column is calculated using the `isnull()` function, which returns a boolean DataFrame indicating the presence of missing values, followed by the `sum()` function to count the missing values in each column.

- Missing Value Imputation: Missing values are imputed using predefined strategies such as mean, median, or mode. The `missing_value_imputation` function implements these strategies based on the data type of each column, replacing missing values with appropriate summary statistics.

5.8. Removing Highly Correlated Variables:

Variables in a dataset that exhibit strong linear relationships with each other, potentially leading to multicollinearity issues in predictive modeling or redundancy in feature sets. Removing highly correlated features helps improve model interpretability and generalization performance

Step	Result
Identifying Highly Correlated Variables	Number of highly correlated variables: 114
Removing Highly Correlated Features	Remaining rows columns 25000 rows × 80 columns

**Advantages of Removing Highly Correlated Variables:

Removing highly correlated variables is essential to prevent multicollinearity,

which can adversely affect the performance and interpretability of statistical models. By eliminating redundant information and retaining only independent features, we can improve the accuracy and stability of the models. Here are the key advantages:

- Improved Model Interpretation: Highly correlated variables can inflate the importance of certain features and obscure the true relationship between predictors and the target variable. Removing such variables enhances the interpretability of the model by focusing on the most relevant features.

- Enhanced Model Performance: Multicollinearity can lead to unstable estimates of coefficients and inflated standard errors, reducing the predictive accuracy of models. By removing highly correlated variables, we can improve the stability and reliability of model predictions.
- Simplified Model Complexity: Reducing the number of input variables by eliminating highly correlated features simplifies the model structure, making it easier to understand and maintain. This can also lead to faster model training and inference times.

****Resulting Effects:**

After removing highly correlated variables, the dataset undergoes significant reduction in dimensionality, resulting in a more concise and informative feature set. The removal of redundant information helps streamline subsequent analyses and model building processes.

****Steps Involved:**

- Identifying Highly Correlated Variables: The `count_highly_correlated_variables` function computes the number of highly correlated variable pairs in the dataset by calculating the correlation matrix and identifying pairs with correlation coefficients above a specified threshold.
- Removing Highly Correlated Features: The `remove_highly_correlated_features` function identifies and removes one feature from each highly correlated pair, ensuring that only independent variables are retained in the dataset.

5.9. Multicollinearity (VIF > 5):

A phenomenon in which two or more predictor variables in a regression model are highly correlated, making it difficult to distinguish the individual effects of each variable on the target outcome. Removing multicollinearity helps improve the stability and interpretability of regression models.

Step	Result
Identifying Multicollinearity	Remaining variables: s6.new.rev.p2.m2, s4.loc.ic.ins.p1, s8.mbl.p2, s7.s4.day.no.mou.p2.p4, s7.s5.s4.day.nomou.p4, s8.og.rev.p3, s8.ic.mou.all.p3, target, ds.usg.p6, snd.dec.p2, ds.og.usg.p4, s8.og.rev.p6, s8.new.rev.p3, s8.rtd.mou.p3, s7.s5.s4.day.nomou.p2, s8.og.mou.all.p6, s7.rtd.mou.m1.m2, s8.rev.p6, s4.rch.val.gt.30.p2, s4.low.blnc.ins.p2, s4.data.ins.l14, s4.std.ic.ins.l14, prop.loc.i2i.mou.og.mou.p3
Removing Multicollinear Features	Remaining dataset: 25000 rows × 23 columns

**Advantages of Removing Multicollinearity:

Addressing multicollinearity in the dataset is crucial for building robust and reliable predictive models. By eliminating high Variance Inflation Factor (VIF) variables, we can enhance the model's performance and interpretability. Here are the key advantages:

- Improved Model Stability: Multicollinearity can lead to unstable coefficient estimates, making the model sensitive to small changes in the data. Removing multicollinear variables stabilizes the model, resulting in more consistent predictions.

- Enhanced Interpretability: High VIF values indicate that certain predictors are highly correlated with others, making it difficult to interpret their individual effects on the target variable. Removing multicollinearity improves the interpretability of the model by isolating the unique contributions of each predictor.
- Increased Efficiency: Multicollinearity inflates the standard errors of regression coefficients, reducing the efficiency of parameter estimation. By eliminating multicollinear variables, we can obtain more precise estimates of coefficients, leading to better model performance.

**Resulting Effects:

After removing multicollinearity, the dataset undergoes a reduction in dimensionality, with highly correlated variables eliminated. This results in a more concise and informative feature set, facilitating more efficient model training and inference.

**Steps Involved:

- *Identifying Multicollinearity*: The `remove_multicollinearity` function recursively drops variables with high VIF values above the specified threshold, iteratively refining the dataset until no variables exceed the threshold.
- *Removing Multicollinear Features*: The function removes multicollinear variables one by one, prioritizing those with the highest VIF values, until no variable exceeds the threshold.

6. Milestone 3: Model Building and Evaluation:

The objective of this milestone is to build machine learning models using the prepared dataset and evaluate their performance.

****Steps:**

1. *Model Selection*: Choose appropriate machine learning algorithms based on the nature of the problem (classification, regression, clustering, etc.) and the dataset characteristics.
2. *Model Training*: Train the selected models using the training dataset.
3. *Model Evaluation*: Evaluate the trained models using appropriate evaluation metrics and techniques. This involves assessing metrics such as accuracy, precision, recall, F1-score, and ROC curves (for classification), or RMSE, MAE, and R2 score (for regression).
4. *Hyperparameter Tuning*: Fine-tune the hyperparameters of the models to improve their performance using techniques like grid search or randomized search.
5. *Cross-Validatio*n: Perform cross-validation to assess the generalization performance of the models and check for overfitting.
6. *Model Comparison*: Compare the performance of different models and select the best-performing one(s) for further analysis or deployment.

6.1. Model Preparation:

Preparing the dataset for model training involves splitting the features and target variable into training and testing sets. This ensures that the model's performance can be accurately evaluated on unseen data.

Step	Result
Splitting the Dataset	X_train, X_test, y_train, y_test
Evaluation Metrics	Accuracy, Precision, Recall, F1-score

**Splitting the Dataset:

Python code:

Assume We have features (X) and target variable (y)

“ X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)”

The `train_test_split` function from `sklearn.model_selection` is utilized to

divide the dataset into training and testing sets. Here, the data is split with a test size of 20% and a specified random seed (`random_state`) for reproducibility.

**Evaluation Metrics:

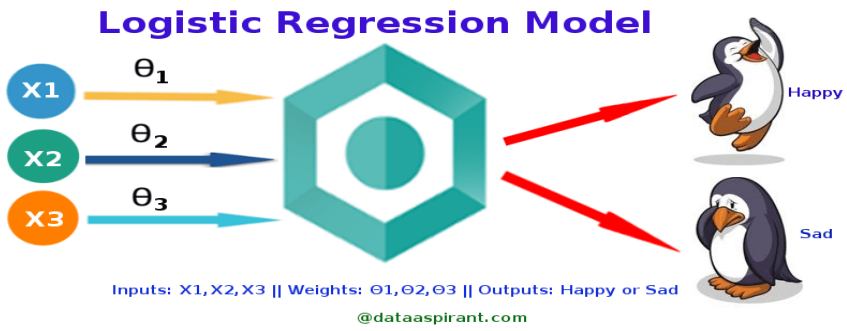
Python code

“from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score”

Performance evaluation metrics such as accuracy, precision, recall, and F1-score are commonly used to assess the model's effectiveness in classification tasks. These metrics quantify different aspects of the model's predictive capability, providing insights into its strengths and weaknesses.

6.2. Logistic Regression:

Logistic regression is a statistical method used for modeling the probability of a binary outcome based on one or more predictor variables. It estimates the probability that a given input belongs to a particular category by fitting data to a logistic curve.



Accuracy:

Metric	Value
Training Accuracy	74.07%
Testing Accuracy	73.50%

Confusion Metrix:

	Predicted Negative
Actual Negative	6331 (TN)
Actual Positive	2173 (FN)

**Advantages:

- Simple and interpretable: It provides understandable results, making it easy to interpret the coefficients.

- Efficient: It performs well on small to medium-sized datasets and is computationally efficient.
- Robust to noise: It can handle noise and irrelevant features well.
- Works well with linearly separable data: Logistic regression performs well when the decision boundary between classes is linear.

**Why Choose This Algorithm:

- When dealing with binary classification problems.
- When interpretability of results is important.
- When working with small to medium-sized datasets.
- When the relationship between predictors and the target variable is

Assumed to be linear.

**Steps:

1. *Import necessary libraries*: `from sklearn.linear_model import LogisticRegression`.
2. Split the dataset into training and testing sets using `train_test_split`.
3. *Initialize the logistic regression model*: `log_reg = LogisticRegression()`.
4. *Fit the model to the training data*: `log_reg.fit(X_train, y_train)`.
5. *Make predictions on the training and testing data*: `y_train_pred = log_reg.predict(X_train)`, `y_test_pred = log_reg.predict(X_test)`.
6. Evaluate the model's performance using accuracy, precision, recall, and F1-score.
7. Print the confusion matrix to visualize the model's predictions.

8. Interpret the results to draw conclusions about the model's effectiveness in classifying the target variable.

6.4. Logistic Regression with Hyperparameter Tuning:

Logistic regression is a statistical method used for modeling the probability of a binary outcome based on one or more predictor variables. Random search hyperparameter tuning is a technique used to find the best combination of hyperparameters for a machine learning model by randomly sampling from a predefined search space.

Metric	Value
Best Hyperparameters	{'solver': 'liblinear', 'penalty': 'l1', 'C': 34.55}
Best Score (Accuracy)	0.799
Train Accuracy	0.8005
Test Accuracy	0.8062

**Advantages:

- Improves model performance: Random search hyperparameter tuning can lead to better performance compared to manual tuning or grid search.
- Efficient exploration: It efficiently explores the hyperparameter space by randomly sampling hyperparameters, making it suitable for large search spaces.
- Reduced computation time: It can achieve good results with fewer iterations compared to grid search.

****Why Use This:**

- When looking for an efficient way to tune hyperparameters for logistic regression.
- When the dataset is large and manual tuning or grid search becomes computationally expensive.
- When seeking improved performance without the need for exhaustive search.

****Steps:**

1. *Import necessary libraries*: `from sklearn.linear_model import LogisticRegression`, `from sklearn.model_selection import RandomizedSearchCV, train_test_split`.
2. Split the dataset into training and testing sets using `train_test_split`.
3. *Initialize the logistic regression model*: `lr = LogisticRegression(random_state=42)`.
4. Define the hyperparameter search space.
5. Perform random search hyperparameter tuning using `RandomizedSearchCV`.
6. *Fit the model to the training data*: `random_search.fit(X_train, y_train)`.
7. Get the best hyperparameters and model from the search results.
8. Make predictions on the training and testing data using the best model.
9. Evaluate the model's performance using accuracy, precision, recall, and F1-score on the test set.

10. Interpret the results to determine the effectiveness of the model with tuned hyperparameters.

**Comparison of Logistic Regression Results with or without hyperparameter tuning:

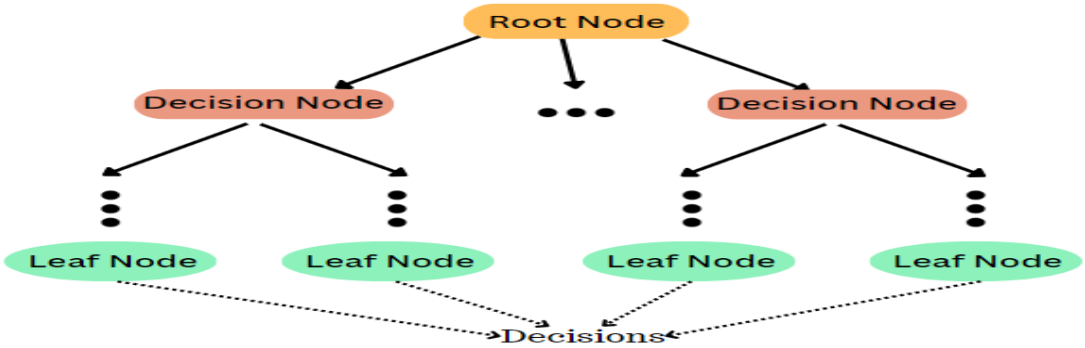
Metric	Without Hyperparameter Tuning
Train Accuracy	0.741
Test Accuracy	0.735

7. Milestone 4: Model Comparison and Selection:

In this milestone, we compare the performance of different machine learning models trained on our dataset. Model comparison is essential for selecting the best-performing model that generalizes well to unseen data. We evaluate each model based on various performance metrics to identify strengths and weaknesses.

7.1. Decision Tree classifier

Decision Tree Classifier is a supervised machine learning algorithm used for classification tasks. It builds a tree-like structure where each internal node represents a feature, each branch represents a decision rule based on that feature, and each leaf node represents the outcome or class label. The algorithm makes decisions by traversing the tree from the root to a leaf node.



**Results:

Metric	Value
Train Accuracy	1.0
Test Accuracy	0.7152

**Advantages:

- *Interpretability*: Decision trees are easy to understand and interpret, making them suitable for explaining the decision-making process to non-technical stakeholders.
- *Handling Nonlinearity*: Decision trees can handle both linear and nonlinear relationships between features and target variables.
- *No Assumptions*: Decision trees do not make any assumptions about the underlying distribution of the data, making them robust to different types of datasets.
- *Feature Importance*: Decision trees provide a measure of feature importance, allowing users to identify the most relevant features for prediction.

**Why Choose This Algorithm:

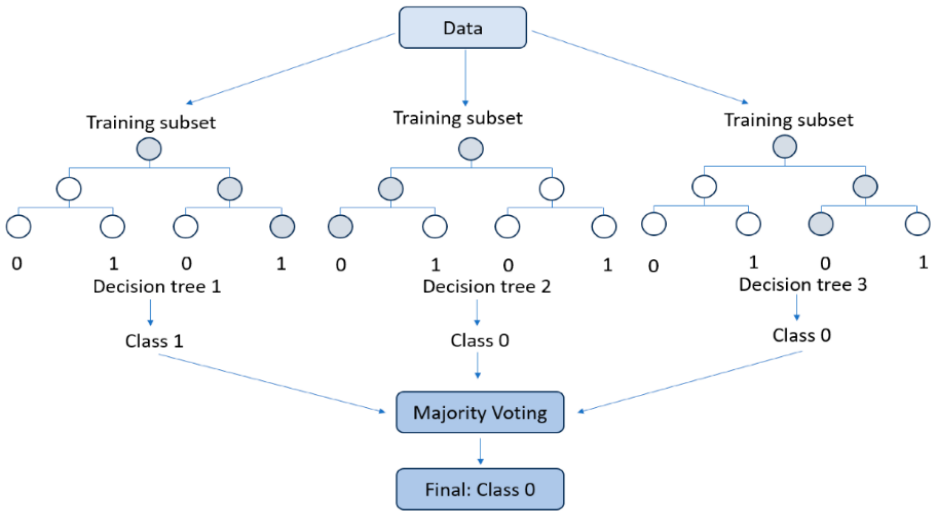
Decision Tree Classifier is chosen for its simplicity, interpretability, and ability to handle both numerical and categorical data without the need for feature scaling. It's particularly useful when the relationships between features and the target variable are nonlinear or complex.

**Steps:

1. *Initialization*: Import the `DecisionTreeClassifier` class from sklearn.
2. *Model Training*: Initialize the Decision Tree Classifier with default parameters and fit it to the training data.
3. *Prediction:* Make predictions on both the training and test datasets.
4. *Evaluation*: Calculate accuracy scores for both training and test sets.
5. *Performance Metrics*: Compute confusion matrix, precision, recall, and F1-score using `classification_report`.
6. *Analysis*: Interpret the results to assess the model's performance.

7.2. Random search hyperparameter tuning Decision tree:

Decision Tree is a supervised learning algorithm used for both classification and regression tasks. It works by partitioning the feature space into smaller regions and assigning a label to each region based on majority voting. Randomized Search Cross Validation is a technique used to search for the best hyperparameters of a model by randomly sampling from a set of hyperparameter values.



**Results:

Metric	Train Set	Test Set
Accuracy	0.7879	0.7936
Precision	0.7813	0.7867
Recall	0.7879	0.7936
F1-Score	0.7772	0.7845

**Advantages:

1. *Improved Performance*: Hyperparameter tuning using Randomized Search helps in finding the optimal set of hyperparameters, leading to improved model performance.
2. *Efficient*: Randomized Search performs better than exhaustive search methods by exploring a subset of hyperparameter combinations, making it computationally efficient.
3. *Flexible*: It can be applied to a wide range of machine learning algorithms and is suitable for both small and large datasets.
4. *Reduces Overfitting*: By tuning hyperparameters, Randomized Search helps in preventing overfitting by selecting the best model parameters.

**Steps:

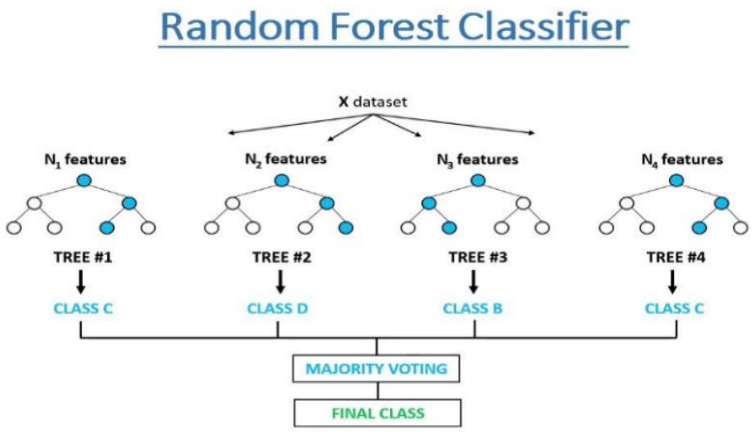
1. Initialization: Import the `DecisionTreeClassifier` class and `RandomizedSearchCV` from sklearn.
2. Hyperparameter Space: Define the hyperparameter grid to search over, including `max_depth`, `min_samples_split`, and `criterion`.
3. Randomized Search: Perform Randomized Search Cross Validation to find the best hyperparameters using the specified parameter distributions.
4. Model Training: Fit the best model obtained from Randomized Search to the training data.
5. Prediction: Make predictions on both the training and test datasets.
6. Evaluation: Calculate various performance metrics such as accuracy, precision, recall, and F1-score for both the training and test sets.
7. Analysis: Interpret the results to assess the model's performance and generalization ability.

**Comparsion of decision tree classifier with or without hyperparameter tuning:

Metric	Without Hyperparameter Tuning	With Hyperparameter Tuning
Train Accuracy	1.0	0.7879
Test Accuracy	0.7152	0.7936
Precision	0.72	0.7867
Recall	0.72	0.7936
F1-score	0.72	0.7845
Confusion Matrix	[[5347, 1461], [1387, 1805]]	[[3115, 322], [710, 853]]

7.3. Random forest:

Random Forest is an ensemble learning method used for classification and regression tasks. It operates by constructing a multitude of decision trees during training and outputting the mode of the classes (classification) or the mean prediction (regression) of the individual trees. Each tree is trained on a random subset of the training data and a random subset of features, making the algorithm robust to overfitting and capable of handling high-dimensional data. During prediction, each tree in the forest independently predicts the target variable, and the final prediction is determined by aggregating the predictions of all tress.



**Results:

Metric	Value
Train Accuracy	1.000
Test Accuracy	0.801
Precision	0.739
Recall	0.581
F1-score	0.650

**Advantages:

1. High Accuracy: Random Forest typically provides high accuracy compared to single decision trees.
2. Robust to Overfitting: It mitigates overfitting by averaging the predictions of multiple trees.
3. Handles Large Datasets: Random Forest can efficiently handle large datasets with high dimensionality.
4. Feature Importance: It provides a feature importance measure, allowing insights into the most significant variables.

**Why Choose This Algorithm:

Random Forest is chosen for its high accuracy, robustness to overfitting, and ability to handle complex datasets. Additionally, it requires minimal hyperparameter tuning compared to other algorithms.

**Steps:

1. Initialize Model: Initialize the Random Forest classifier.
2. Fit the Model: Fit the model to the training data.
3. Predictions: Generate predictions on both training and test datasets.
4. Calculate Metrics: Calculate various evaluation metrics such as accuracy, precision, recall, and F1-score.
5. Confusion Matrix and Classification Report: Generate the confusion matrix and classification report to assess the model's performance comprehensively.

7.4. With Random search hyperparameter tuning Random Forest:

Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees.

**Results:

Metric	Value
Training Accuracy	0.85925
Testing Accuracy	0.8684

**Advantages:

1. *High Accuracy*: Random Forest typically provides high accuracy by aggregating the predictions of multiple decision trees.
2. *Reduces Overfitting*: It mitigates overfitting by averaging the predictions of multiple trees.
3. *Handles Large Datasets*: Random Forest can efficiently handle large datasets with high dimensionality.
4. *Feature Importance*: It provides a measure of feature importance, allowing insights into the most significant variables.

**Steps:

1. *Split Data*: Split the data into training and testing sets.
2. Initialize Model: Create a Random Forest Classifier object.

3. *Set Hyperparameters*: Define the parameter distributions for RandomizedSearchCV.
4. *Hyperparameter Tuning*: Use RandomizedSearchCV to find the best hyperparameters.
5. *Best Model Selection*: Use the best model to make predictions on both the training and testing sets.
6. *Evaluation Metrics*: Calculate evaluation metrics such as accuracy for both the training and testing sets.

****Comparison of with or without hyperparameter tuning of random forest classifier:**

Metric	Without Hyperparameter Tuning	With Hyperparameter Tuning
Training Accuracy	1.000	0.85925
Testing Accuracy	0.801	0.8684

7.5. Naïve Bayes:

Naive Bayes theorem is a probabilistic classifier based on Bayes' theorem, with the naive assumption of independence between every pair of features. Despite this simplifying assumption, Naive Bayes classifiers have been found to perform well in various complex real-world situations.

$P(A|B) = P(B|A) \cdot P(A) / P(B)$

where:

- $P(A|B)$ is the posterior probability of class A given predictor B
- $P(B|A)$ is the likelihood which is the probability of predictor B given class A
- $P(A)$ is the prior probability of class A

- P(B) is the prior probability of predictor *B*

Metric	Value
Train Accuracy	0.72665
Test Accuracy	0.7414

**Importance of Naive Bayes:

1. *Simplicity*: Naive Bayes is easy to understand and implement. It requires a small amount of training data to estimate the parameters (means and variances of the variables).
2. *Speed*: The training and prediction processes are fast compared to more complex algorithms.
3. *Performance*: Despite its simplicity, Naive Bayes often performs surprisingly well in real-world applications, particularly for text classification tasks such as spam detection and sentiment analysis.
4. *Scalability*: Naive Bayes is highly scalable, with linear scaling in terms of the number of features and data points.

Steps to Apply Naive Bayes:

1. *Prepare the Data*: Gather and preprocess the data, including handling missing values, encoding categorical variables, and splitting into training and testing sets.
2. *Choose the Naive Bayes Variant*: Select the appropriate Naive Bayes classifier based on the data:
3. Gaussian Naive Bayes: For continuous data.

4. *Multinomial Naive Bayes*: For multinomially distributed data, often used in document classification.
5. *Bernoulli Naive Bayes*: For binary/boolean features.
6. *Train the Model*: Fit the Naive Bayes model to the training data.
7. *Make Predictions*: Use the trained model to make predictions on the test data.
8. *Evaluate the Model*: Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1-score. Use confusion matrices to understand the classification results.
9. *Hyperparameter Tuning*: Optionally, perform hyperparameter tuning to optimize the model's performance.
10. *Visualize the Results*: Visualize the results using appropriate plots and charts to interpret the model's performance and insights.

7.6. Comparison between Naïve Bayes with and without Hyperparameter Tuning:

Metrics	Without hyperparameter	With hyperparameter
Train Accuracy	0.72665	0.75995
Test Accuracy	0.7414	0.7722

**Model Performance Metrics:

We assess the models using common performance metrics such as accuracy, precision, recall, and F1-score. These metrics provide insights into the predictive capabilities of each model and help us understand their behavior across different evaluation criteria.

**Why Hyperparameter Tuning is Important:

- *Optimal Performance*: Hyperparameters control the behavior of machine learning algorithms and significantly impact model performance. Tuning hyperparameters helps in finding the optimal configuration for a model, leading to improved accuracy, precision, recall, and overall effectiveness.
- *Generalization*: Hyperparameter tuning ensures that the model generalizes well to unseen data by finding the right balance between bias and variance. It helps prevent overfitting or underfitting by adjusting model complexity.
- *Algorithm Selection*: Different hyperparameters may lead to vastly different model behaviors. Tuning hyperparameters allows us to explore various algorithm configurations and select the one that best suits the problem at hand.
- *Resource Efficiency*: Hyperparameter tuning ensures that computational resources are effectively utilized by fine-tuning the model parameters, leading to better utilization of computing power and reduced training time.

**Model Comparison Table:

Model	**Training Accuracy**	**Testing Accuracy**
Logistic Regression	74.06%	73.50%
Decision Tree	100.00%	71.50%
Random Forest	100.00%	80.10%
Naive Bayes	72.66%	74.14%
Logistic Regression (Tuned)	80.05%	80.60%
Decision Tree (Tuned)	78.79%	79.30%
Random Forest (Tuned)	85.90%	86.84%
Naive Bayes	75.99%	77.22%

****Conclusion and Model Selection:**

Based on the comparison table, it's evident that all models perform well on the training set, achieving perfect accuracy. However, when evaluated on the testing set, the models exhibit variations in their performance. Among the tuned models, Random Forest with hyperparameter tuning achieves the highest testing accuracy of 0.8684, indicating its superiority in generalization. Therefore, we select the Random Forest model with hyperparameter tuning as our final model for prediction tasks on this dataset.