

(7153CEM)

Course work

Big Data Analytics and Data Visualisation

Module Leader:

Dr. Anup Pandey

Student Name:

Sandeep Dasarathan Kumanan

Student ID:

(13530374)

Title:

An Enhanced Tesco customer churn prediction
using Machine learning

Table of Contents

Abstract.....	3
1. INTRODUCTION	3
2. Related work	4
3. Dataset Section.....	4
3.1 Pyspark Installation.....	4
3.2 Dataset	5
3.3 Implementation	5
3.3.1 Loading Data.....	5
3.3.2 Data Pre-Processing	6
3.3.3 Exploration Data Analysis	6
4. Methodology	7
4.1 Decision Tree:.....	7
4.2 Random Forest.....	7
4.3 Gradient Boosting Tree	8
4.4 Logistic Regression	8
4.5 Classification results.....	8
5. Experimental section.....	8
6. Result Discussion.....	9
7. Conclusion	10
8. Social ethics	10
9. Reference	11
Appendix:	12
A. Code:	12
B. Output:.....	18

Abstract

This study examines the enrolment trends within the Tesco retail store Clubcard program. It aims to forecast the number of customers who will either maintain or discontinue their membership using various machine-learning classification techniques. The research evaluates four distinct modules: Decision Trees, Random Forest, Gradient Boosting Trees, and Logistic Regression, drawing comparisons among their predictive performance.

Big data analysis and machine learning techniques were implemented using PySpark. The findings were then presented visually using tools like Tableau, along with popular Python libraries such as matplotlib, pandas, and seaborn. The outcome of the analysis revealed that among various models, Gradient Boosting Trees exhibited superior performance.

Keywords—Machine Learning, Big Data, PySpark, Pipeline, Tableau, Decision Trees, Random Forest, Gradient Boosting Trees, Logistic Regression

1. INTRODUCTION

This document was prepared as a component of the course covering Big Data management and data visualization. The assignment centers around the concept of handling large volumes of data and encompasses tasks such as identifying and choosing suitable analytical methods for analyzing such data, followed by a thorough examination of the outcomes. The project entails the utilization of Apache Spark, more precisely PySpark, which seamlessly integrates with the Python programming language. Apache Spark stands as a flexible framework for performing analytics on extensive datasets through its set of proprietary libraries.

The primary goal of this project is to examine the behavior of Tesco's Clubcard customers, determining whether they choose to maintain their membership or discontinue it. Additionally, the project aims to forecast levels of fine particulate matter. . To accomplish these objectives, specific goals were established:

1. Utilize PySpark for the purpose of loading and manipulating data.
2. Examine and dissect data using suitable methods and approaches.
3. Utilize Tableau to create visual representations of data and analyze the discoveries made.

4. Identify suitable methods from the field of machine learning to make predictions.
5. In the subsequent section (Section 2), we will delve into the process of establishing PySpark, detailing the dataset used, conducting an in-depth analysis of the data, and ultimately, executing the Machine Learning implementations.

2. Related work

In previous studies, researchers have utilized a range of methods such as machine learning, data mining, and combinations of these techniques to predict churn. These approaches assist businesses in recognizing and anticipating customers who are likely to leave, aiding in strategic decision-making. Among these methods, Random Forest stands out as one of the most frequently employed techniques to predict and address issues related to customer churn (*Ullah, B. Raza, A. K. Malik, M. Imran, S. U. Islam and S. W. Kim, 2019*). a churn prediction model for free-to-play mobile games. Their approach involves creating player behavioral features that accurately reflect player churn tendencies. These features are then used to build churn prediction models. To implement their proposed churn model, the researchers applied classification algorithms. (*S. -K. Lee, S. -J. Hong, S. -I. Yang and H. Lee, 2016*). The aim of this project is to achieve high accuracy using a dataset that has been divided into categorical and numerical features. Three machine learning models, namely Decision Trees (DT), Naïve Bayes (NB), and Random Forest (RF), have been created. These models are used to determine if a customer will subscribe to a term deposit. The ultimate goal is to identify the potential customers who are likely to subscribe (*I. Kaur and J. Kaur, 2020*)

3. Dataset Section

3.1 Pyspark Installation

The configuration of PySpark was established within the Google Colab environment. This choice offers numerous advantages, including the presence of pre-installed libraries, the option to leverage a Graphics Processing Unit (GPU), and the convenience of seamlessly switching between R and Python, if the need arises.

The Pyspark package can be easily installed using the Python package installer, pip. This package is particularly useful for conducting local testing of your tasks or for executing them on a pre-existing cluster that operates on Yarn, Standalone, or Mesos ([figure 1](#)).

Python enthusiasts make use of 'pip,' a package manager, to conveniently add or remove external packages not included in the core Python library. With pip, it's possible to effortlessly install, uninstall, update, or even revert various Python libraries that extend the capabilities of the language.

The pip command starts collecting the Pyspark package and installing it ([figure 2](#)).

An issue involving a py4j error arose, requiring Spark and PySpark to have the exact same versions. As a solution, it was determined that installing PySpark version 3.4.1 specifically would prevent this error from occurring. You can verify the installed version by using the command shown in ([Figure 3](#)).

3.2 Dataset

This dataset was taken from Kaggle. This dataset portrays various customers of Tesco who are enrolled in the Clubcard membership program. The columns provide information about these customers' characteristics. On the left, there is a column indicating whether a customer has decided to discontinue their membership. By analyzing this dataset, we can anticipate potential future actions of customers and extract valuable insights, such as determining whether adjustments are needed for offers or if certain offers should be phased out. . The raw original data contain about 7,043 rows and 21 columns and in this data, our target column is LeftMembership which contains categorical columns.

3.3 Implementation

3.3.1 Loading Data

When dealing with Big Data, the Spark RDD and SQL modules are commonly employed. Among these, Spark SQL stands out as a component designed for managing structured data. Unlike Spark RDD, it offers more detailed insights into both the arrangement of data and the processes applied to it. However, ,it's worth noting that these modules share the same underlying execution engine for carrying out computations.

The project primarily utilizes the Spark SQL module for conducting computations, while visualizations are created using Tableau.

In [Figure 4](#) below, a new spark context and session were created and read the CSV file containing this dataset.

In [figure 5](#), The initial display presents the first 21 rows of the dataset. These rows encompass various categorical data points such as customer ID, gender, partner status, presence of dependents, phone service availability, . multiple lines usage, type of internet service, online security and backup choices, device protection, tech support availability, streaming TV and movie options, contract type, paperless billing preference, payment method, and whether the customer has left the membership. Additionally, the dataset includes numerical values like senior citizen status, tenure, monthly charges, and total charges.

3.3.2 Data Pre-Processing

Pre-processing involves the inspection of data, counting the data in the dataset, type of dataset, and checking the types of data in each and every column in the dataset using printSchema that might cause issues to process data. There were problems with reading and processing data in some of the columns, therefore there was a need to check the type of columns – [Figure 6](#), [Figure 7](#) and [Figure 8](#).

This code in [Figure 9](#) checks missing values in the spark data frame) and almost every column had missing values. It was decided to perform feature imputation instead of dropping those value.

3.3.3 Exploration Data Analysis

The task involves conducting exploratory data analysis on the Tesco retail store churn prediction dataset using Tableau 2023.2 Desktop version. The dataset, imported as a CSV file, needs to be cleaned using the data interpreter. With 21 variables in the dataset, the main focus is understanding customer preferences for membership methods and identifying factors that contribute to increased club card sign-ups at Tesco retail stores.

By leveraging Tableau, the dataset is comprehensively explored by establishing relationships between variables and the target variable. A specific visualization [Figure 10](#) showcases a comparison between payment methods and monthly charges, revealing how customers make payments. . Another visualization [Figure 11](#) takes the form of a circular chart to display how many senior citizens are interested in streaming TV and streaming movies, with a gender-based split.

In [Figure 12](#), a pie chart is employed to reveal the distribution of customer contracts, considering the contract duration and categorizing it based on the payment method. [Figure 13](#) employs a stacked bar approach to display how senior citizens show interest in the club through online means, while also considering internet service, online backup, and online security.

A horizontal bar graph in [Figure 14](#) highlights differences related to dependents, phone service, contracts, monthly charges, and total charges. [Figure 15](#) involves a packed bubble visualization, which compares LeftMembership and Tenure with factors like Tech Support, Multi Lines, and Device Protection.

[Figure 16](#) utilizes a Treemap to demonstrate relationships between Tenure, paperbilling, internet service, phone service, and payment method. . Lastly, [Figure 17](#) employs a box-and-whisker plot to illustrate features like dependents and partners alongside Tenure, monthly charges, and total charges.

Overall, Tableau is a crucial tool in comprehending the Tesco retail store churn prediction dataset, allowing for insightful visualizations and data-driven insights into customer behavior and preferences.

4. Methodology

4.1 Decision Tree:

Decision tree classification in PySpark involves using the machine learning library of Apache Spark to create a predictive model based on decision tree algorithms. Decision trees split the dataset into subsets based on the values of input features, aiming to make accurate predictions for classification tasks.

To begin, it's necessary to bring in the library indicated in [Figure 18](#),

Next, we will associate a column with the labelCol and another column with the featuresCol, as illustrated in [Figure 19](#).

Decision tree accuracy refers to the measure of how well a decision tree model can correctly predict outcomes or classify instances within a dataset. This metric gauges the percentage of instances for which the decision tree's predictions match the actual outcomes. A higher accuracy score indicates that the decision tree model is making more accurate predictions, while a lower score suggests that the model's predictions are less reliable. Decision tree accuracy is a crucial evaluation metric in assessing the performance of decision tree models and their suitability for various tasks, such as classification or regression problems. To find the accuracy of the decision tree module and I used to show the error values of module, shown in [figure 20](#).

Module Weight of Recall, f1, Weight Precision shown in [figure 21](#).

4.2 Random Forest

This random forest classifier is a supervised learning algorithm that can be used to predict and classify the data from the data frame. It is created from subsets of data, and the result is based on the average or majority. The random forest performed well in finding the missing value in the data frame.

Overfitting gives a problem in this random forest unless enough number trees are created to improve accuracy and working principle of Random Forest shown in [figure 22](#).

To find the accuracy of the Random Forest module and I used to show the error values of a module, shown in [Figure 23](#).

Module Weight of Recall, f1, Weight Precision shown in [figure 24](#).

4.3 Gradient Boosting Tree

The Gradient Boosting Tree (GBT) classification algorithm is renowned for its remarkable accuracy in various machine learning tasks. It stands out as a powerful ensemble learning technique that combines the strengths of multiple decision trees to create a robust and high-performing predictive model.

To find the accuracy of the Gradient Boosting Tree (GBT) module and I used to show the error values of the module, shown in [figure 25](#).

Module Weight of Recall, f1, Weight Precision shown in [figure 26](#).

4.4 Logistic Regression

Logistic Regression is a widely used classification algorithm in machine learning that is particularly suited for binary classification problems, where the goal is to predict one of two possible outcomes. One essential aspect of assessing the effectiveness of any classification algorithm is its accuracy.

Calculation of Accuracy: The accuracy is calculated using the formula:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100\%$$

To find the accuracy of the Logistic Regression Classification module and I used to show the error values of the module, shown in [Figure 27](#).

Module Weight of Recall, f1, Weight Precision shown in [figure 28](#).

4.5 Classification results

The result will shown in [figure 29](#).

5. Experimental section

The necessary libraries were brought in for Decision Trees, Random Forest, and Gradient Boosting Trees, Logistic Regression Classification. Each of these models was then trained and put to the test and it shown in [figure 30](#).

The dataset contains a significant number of categorical values, making it challenging to directly utilize the data in a model. This has led to the emergence of errors during the fitting process. To address this issue, we intend to employ the StringIndexer () technique. This approach involves transforming each categorical value into a corresponding numerical

representation. Additionally, we'll have the flexibility to rename columns as needed. . The visual representation of these changes is provided in [Figure 31](#).

In this piece of code, we employ the VectorAssembler feature. Its purpose is to merge multiple columns together, consolidating them into a single column that we conveniently name as "features". This combined column is a key component for making predictions. We integrate this newly formed column into a collection known as featureCols, which holds the features used in the prediction process. Refer to [Figure 32](#) for a visual representation.

We will now divide the dataset into two categories: one will be used to train the module, and the other will be used to test its performance and calculate accuracy. The dataset will be divided in a 70-30 ratio, with 70% allocated for training and the remaining 30% for testing. This division will be done randomly, facilitated by employing the 'randomSplit' function in the code shows in [figure 33](#).

The train data was divided randomly, and around 70% of the data was assigned to the 'train_data' as illustrated in [Figure 34](#).

In [Figure 35](#), we designated 30% of the data for testing purposes within the 'test_data'.

Now we going to use the pipeline, several guides were used to build a pipeline in [figure 36](#).

Now we going to do paramGrid, The PySpark library's module pyspark.ml.tuning offers a comprehensive set of functions designed for model tuning purposes. Prior to conducting hyperparameter tuning, it's essential to establish a parameter grid. This grid will serve as the foundation for both hyperparameter tuning and the process of cross-validation shown [Figure 37](#).

We will now proceed with the CrossValidator technique, which involves selecting the best model by dividing the dataset into distinct, randomly separated folds. Each of these folds serves as an independent training and testing set shown in [figure 38](#).

6. Result Discussion

The outcome revealed that out of the instances considered, 7043 were accurately categorized by the Gradient Boosted Tree (GBT) model. This model achieved an accuracy rate of 77.7%, indicating the proportion of correct predictions, and a test error of 22.7%, representing the extent of inaccuracies in its predictions. The Decision Tree (DT) exhibited an accuracy of 76.5% with a corresponding test error of 23.4%. Similarly, the Logistic Regression (LGR)

model also achieved 76.5% accuracy and a test error of 23.4%. The final model, the Random Forest Classifier (RF), achieved an accuracy of 76.9%, while its test error stood at 23.0%

In general, the Gradient Boosted Tree (GBT) algorithm yielded favorable outcomes when compared to other classification methods in anticipating the count of customers who withdrew their membership from the Tesco retail Clubcard dataset. To assess the model's effectiveness, metrics such as accuracy, F1 score, recall, and precision were employed using the `MulticlassClassificationEvaluator` in PySpark.

7. Conclusion

This research delved into the application of PySpark's machine learning classification techniques in conjunction with the Clubcard dataset from Tesco's retail stores, revealing a promising strategy. The central aim was to predict the attrition of Clubcard memberships. Notably, the Gradient Boosted Tree (GBT) algorithm stood out as exceptionally effective, consistently outperforming other classifiers in predictive capabilities. Thorough assessment using metrics such as accuracy, F1 score, recall, and precision consistently showcased impressive results for the GBT algorithm. It displayed a remarkable ability to identify patterns within the data, leading to accurate forecasts regarding membership attrition. This achievement can be attributed to the algorithm's ensemble learning structure, which combines the strengths of various weak learners to create a robust predictive model. The outcomes of this study underscore the potential of PySpark and machine learning algorithms in advancing the comprehension of customer behavior within retail membership programs. The GBT's capacity to predict membership attrition with accuracy can assist businesses like Tesco in developing targeted strategies to retain customers and enhance overall satisfaction. This investigation highlights the importance of utilizing advanced analytical techniques to extract valuable insights from customer data, enabling well-informed business decisions that foster sustainable growth and improved customer relationships in the retail industry.

8. Social ethics

Social Consideration: Applying this machine learning algorithm to the Tesco retail store club card dataset enhances marketing processes while ensuring no negative impact on individuals and communities.

Ethical: The customer's provided personal information from the retail store is utilized to predict whether they will maintain or discontinue their membership. This data will remain secure and safeguarded from any external parties.

Legal Consideration: This dataset remains free from misuse by third-party vendors; any manipulation or unauthorized commercial use will result in legal consequences for the involved vendor.

Professional concerns: As developers work on creating a machine learning algorithm for this dataset, it's important for them to maintain a professional approach. This ensures that the aforementioned considerations hold the utmost importance during the development process.

9. Reference

1. Ullah, B. Raza, A. K. Malik, M. Imran, S. U. Islam and S. W. Kim, "A Churn Prediction Model Using Random Forest: Analysis of Machine Learning Techniques for Churn Prediction and Factor Identification in Telecom Sector," in IEEE Access, vol. 7, pp. 60134-60149, 2019, doi: 10.1109/ACCESS.2019.2914999.
2. S. -K. Lee, S. -J. Hong, S. -I. Yang and H. Lee, "Predicting churn in mobile free-to-play games," 2016 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South), 2016, pp. 1046-1048, doi: 10.1109/ICTC.2016.7763364.
3. I. Kaur and J. Kaur, "Customer Churn Analysis and Prediction in Banking Industry using Machine Learning," 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC), Wagnaghat, India, 2020, pp. 434-437, doi: 10.1109/PDGC50313.2020.9315761.

Appendix:

A. Code:

```
!pip install pyspark
```

```
import pyspark
print(pyspark.__version__)
```

```
from pyspark import SparkFiles

from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler,
OneHotEncoder
from pyspark.ml.classification import
DecisionTreeClassifier, RandomForestClassifier, LogisticRegression, GBTCla
ssifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

```
spark =
SparkSession.builder.appName("Tesco_retail_cub_card").getOrCreate()
```

```
df = spark.read.csv('Tesco.csv', header= True, inferSchema=True)
```

```
df.show()
```

```
df.describe().show()
```

```
type(df)
```

```
df.columns
```

```
df.printSchema()
```

```
stringIndexer =
StringIndexer(inputCols=['gender', 'SeniorCitizen', 'Partner', 'Dependents',
'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'On
lineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMo
vies', 'PaperlessBilling', 'LeftMembership'],
outputCols=['gender_index', 'SeniorCitizen_index', 'Partner_index', 'Depen
dents_index', 'PhoneService_index', 'MultipleLines_index', 'InternetServic
```

```
e_index', 'OnlineSecurity_index', 'OnlineBackup_index', 'DeviceProtection_index', 'TechSupport_index', 'StreamingTV_index', 'StreamingMovies_index', 'PaperlessBilling_index', 'label']])
```

```
assembler=VectorAssembler(inputCols=['gender_index', 'SeniorCitizen_index', 'Partner_index', 'Dependents_index', 'PhoneService_index', 'MultipleLines_index', 'InternetService_index', 'OnlineSecurity_index', 'OnlineBackup_index', 'DeviceProtection_index', 'TechSupport_index', 'StreamingTV_index', 'StreamingMovies_index', 'PaperlessBilling_index'], outputCol='features')
```

```
train_data, test_data = df.randomSplit([0.7, 0.3], seed=42)
```

```
train_data.show()
```

```
train_data.describe().show()
```

```
test_data.show(10)
```

```
test_data.describe().show()
```

```
dt = DecisionTreeClassifier(labelCol='label', featuresCol='features')
```

```
pipeline = Pipeline(stages=[stringIndexer, assembler, dt])
```

```
paramGrid = ParamGridBuilder() \
    .addGrid(dt.maxDepth, [3, 5, 7]) \
    .addGrid(dt.minInstancesPerNode, [1, 3, 5]) \
    .build()
```

```
crossval = CrossValidator(estimator=pipeline,
    estimatorParamMaps=paramGrid,
    evaluator=MulticlassClassificationEvaluator(
        labelCol='label', predictionCol='prediction',
metricName='accuracy'),
    numFolds=5)
```

```
cvModel = crossval.fit(train_data)
```

```
best_model = cvModel.bestModel
```

```
predictions = best_model.transform(test_data)
```

```
predictions.select("prediction", "label", "features").show(5)
```

```
evaluator = MulticlassClassificationEvaluator(labelCol="label",  
predictionCol="prediction", metricName="accuracy")  
accuracy1 = evaluator.evaluate(predictions)
```

```
print(f"DecisionTree Test Accuracy: ",accuracy1)  
print("Test Error = %g" % (1.0 - accuracy1))
```

```
evaluator = MulticlassClassificationEvaluator(labelCol="label",  
predictionCol="prediction",  
metricName="f1")  
f1_dt = evaluator.evaluate(predictions)  
print("f1 for Decision Tree Classifier:", f1_dt)
```

```
evaluator = MulticlassClassificationEvaluator(labelCol="label",  
predictionCol="prediction",  
metricName="weightedPrecision")  
weightedPrecision_dt = evaluator.evaluate(predictions)  
print("weightedPrecision for Decision Tree Classifier:",  
weightedPrecision_dt)
```

```
evaluator = MulticlassClassificationEvaluator(labelCol="label",  
predictionCol="prediction",  
metricName="weightedRecall")  
weightedRecall_dt = evaluator.evaluate(predictions)  
print("weightedRecall for Decision Tree Classifier:",  
weightedRecall_dt)
```

```
print("Accuracy: ",accuracy1)  
print("f1 for Random Forest Classifier:", f1_dt)  
print("weightedPrecision for Random Forest Classifier:",  
weightedPrecision_dt)  
print("weightedRecall for Random Forest Classifier:",  
weightedRecall_dt)
```

```
rf = RandomForestClassifier(labelCol="label", featuresCol="features")
```

```
pipeline1 = Pipeline(stages=[stringIndexer, assembler, rf])
```

```
paramGrid1 = ParamGridBuilder() \  
    .addGrid(rf.numTrees, [10, 20, 30]) \  
    .addGrid(rf.maxDepth, [5, 10, 15]) \  
    .build()
```

```
cross_validator1 = CrossValidator(estimator=pipeline1,  
                                  estimatorParamMaps=paramGrid1,  
                                  evaluator=MulticlassClassificationEvaluator(labelCol="label", metricName="accuracy"),  
                                  numFolds=5, seed=42)
```

```
cv_model1 = cross_validator1.fit(train_data)
```

```
predictions = cv_model1.transform(test_data)
```

```
predictions.select("prediction", "label", "features").show(5)
```

```
evaluator2 = MulticlassClassificationEvaluator(labelCol="label",  
metricName="accuracy")
```

```
accuracy2 = evaluator2.evaluate(predictions)  
print("Random Forest accuracy = ", accuracy2)  
print("Test Error= %g" %(1.0-accuracy2))
```

```
evaluator2 = MulticlassClassificationEvaluator(labelCol="label",  
predictionCol="prediction",  
metricName="f1")  
f1_rf = evaluator2.evaluate(predictions)  
print("f1 for Random Forest Classifier:", f1_rf)
```

```
evaluator2 = MulticlassClassificationEvaluator(labelCol="label",  
predictionCol="prediction",  
metricName="weightedPrecision")  
weightedPrecision_rf = evaluator2.evaluate(predictions)  
print("weightedPrecision for Random Forest Classifier:",  
weightedPrecision_rf)
```

```
evaluator2 = MulticlassClassificationEvaluator(labelCol="label",  
predictionCol="prediction",  
metricName="weightedRecall")
```

```
weightedRecall_rf = evaluator2.evaluate(predictions5)
print("weightedRecall for Random Forest Classifier:",
weightedRecall_rf)
```

```
print("Accuracy: ",accuracy2)
print("f1 for Random Forest Classifier:", f1_rf)
print("weightedPrecision for Random Forest Classifier:",
weightedPrecision_rf)
print("weightedRecall for Random Forest Classifier:",
weightedRecall_rf)
```

```
gbt = GBTClassifier(labelCol="label", featuresCol="features",
maxIter=10)
pipeline = Pipeline(stages=[stringIndexer, assembler, gbt])
```

```
model = pipeline.fit(train_data)
```

```
predictions = model.transform(test_data)
```

```
predictions.select("prediction", "label", "features").show(5)
```

```
evaluator3 = MulticlassClassificationEvaluator(
    labelCol="label", predictionCol="prediction",
    metricName="accuracy")
accuracy3 = evaluator3.evaluate(predictions)
print("Accuracy: ",accuracy3)
print("Test Error = %g" % (1.0 - accuracy3))
```

```
evaluator3 = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction",
metricName="f1")
f1_gbt = evaluator3.evaluate(predictions)
print("f1 for GBT Classifier:", f1_gbt)
```

```
evaluator3 = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction",
metricName="weightedPr
ecision")
weightedPrecision_gbt = evaluator3.evaluate(predictions)
print("weightedPrecision for GBT Classifier:", weightedPrecision_gbt)
```



```
evaluator3 = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction",
metricName="weightedRecall")
weightedRecall_gbt = evaluator3.evaluate(predictions5)
print("weightedRecall for GBT Classifier:", weightedRecall_gbt)
```

```
print("Accuracy: ",accuracy3)
print("f1 for GBT Classifier:", f1_gbt)
print("weightedPrecision for GBT Classifier:", weightedPrecision_gbt)
print("weightedRecall for GBT Classifier:", weightedRecall_gbt)

from pyspark.ml.classification import LogisticRegression
```

```
logistic_regression = LogisticRegression(featuresCol="features",
labelCol="label")
pipeline5 = Pipeline(stages=[stringIndexer, assembler,
logistic_regression])
```

```
model = pipeline5.fit(train_data)
```

```
predictions5 = model.transform(test_data)
```

```
predictions5.select("prediction", "label", "features").show(5)
```

```
evaluator5 = MulticlassClassificationEvaluator(
    labelCol="label", predictionCol="prediction",
    metricName="accuracy")
accuracy5 = evaluator5.evaluate(predictions5)
print("Accuracy: ",accuracy5)
print("Test Error = %g" % (1.0 - accuracy5))
```

```
evaluator5 = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction",
metricName="f1")
f1_lr = evaluator5.evaluate(predictions5)
print("f1 for Logistic Regression Classifier:", f1_lr)
```

```
evaluator5 = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction",
metricName="weightedPrecision")
weightedPrecision_lr = evaluator5.evaluate(predictions5)
print("weightedPrecision for Logistic Regression Classifier:",
weightedPrecision_lr)
```

```
evaluator5 = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction",
metricName="weightedRecall")
weightedRecall_lr = evaluator5.evaluate(predictions5)
print("weightedRecall for Logistic Regression Classifier:",
weightedRecall_lr)
```

```
print("Accuracy: ",accuracy5)
print("f1 for Logistic Regression Classifier:", f1_lr)
print("weightedPrecision for Logistic Regression Classifier:",
weightedPrecision_lr)
print("weightedRecall for Logistic Regression Classifier:",
weightedRecall_lr)
```

B. Output:


✓ 32s  !pip install pyspark

Figure 1: installation of pyspark

```
Collecting pyspark
  Downloading pyspark-3.4.1.tar.gz (310.8 MB)
    310.8/310.8 MB 4.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.4.1-py2.py3-none-any.whl size=311285388 sha256=72c8b210dbc8699aa45aee2061b2ad208e50b7d91b45a6924c
  Stored in directory: /root/.cache/pip/wheels/0d/77/a3/ff2f74cc9ab41f8f594dabf0579c2a7c6de920d584206e0834
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.4.1
```

Figure 2: Collecting the PySpark package and installing it.

✓ 0s [3] `import pyspark`
`print(pyspark.__version__)`
3.4.1

Figure 3: PySpark Version

```
[ ] spark = SparkSession.builder.appName("Tesco_retail_cub_card").getOrCreate()

[ ] df = spark.read.csv('Tesco.csv',header= True,inferSchema=True)

[ ] df.show()
```

Figure 4: Reading data.

customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport
7590-VHVES	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No	Nc
5575-GWDE	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes	Nc
3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No	Nc
7795-CFOCM	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Yes	Yes
9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No	Nc
9305-CDSKC	Female	0	No	No	8	Yes	Yes	Fiber optic	No	No	Yes	Nc
1452-KIOVK	Male	0	No	Yes	22	Yes	Yes	Fiber optic	No	Yes	No	Nc
6713-OKOMC	Female	0	No	No	10	No	No phone service	DSL	Yes	No	No	Nc
7892-POOKP	Female	0	Yes	No	28	Yes	Yes	Fiber optic	No	No	Yes	Yes
6388-TABGU	Male	0	No	Yes	62	Yes	No	DSL	Yes	Yes	No	Nc
9763-GRSKD	Male	0	Yes	Yes	13	Yes	No	DSL	Yes	No	No	Nc
7469-LKBCI	Male	0	No	No	16	Yes	No	No	No internet service	No internet service	No internet service	No internet service
8091-TTVAX	Male	0	Yes	No	58	Yes	Yes	Fiber optic	No	No	Yes	Nc
0280-XJGEX	Male	0	No	No	49	Yes	Yes	Fiber optic	No	Yes	Yes	Nc
5129-JLPIS	Male	0	No	No	25	Yes	No	Fiber optic	Yes	No	Yes	Yes
3655-SNQYZ	Female	0	Yes	Yes	69	Yes	Yes	Fiber optic	Yes	Yes	Yes	Yes
8191-XW5ZG	Female	0	No	No	52	Yes	No	No	No internet service	No internet service	No internet service	No internet service
9959-WOFKT	Male	0	No	Yes	71	Yes	Yes	Fiber optic	Yes	No	Yes	Nc
4190-MFLUM	Female	0	Yes	Yes	10	Yes	No	DSL	No	No	Yes	Yes
4183-MYFRB	Female	0	No	No	21	Yes	No	Fiber optic	No	Yes	Yes	Nc

only showing top 20 rows

Figure 5: Dataset containing environmental data.

```
✓ [28] type(df)
0s
pyspark.sql.dataframe.DataFrame
```

Figure 6: Data Frame type

```
✓ 0s [37] df.schema.names

['customerID',
 'gender',
 'SeniorCitizen',
 'Partner',
 'Dependents',
 'tenure',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod',
 'MonthlyCharges',
 'TotalCharges',
 'LeftMembership']
```

Figure 7: getting column name from the dataset.

```
✓ 0s [30] df.printSchema()

root
|-- customerID: string (nullable = true)
|-- gender: string (nullable = true)
|-- SeniorCitizen: integer (nullable = true)
|-- Partner: string (nullable = true)
|-- Dependents: string (nullable = true)
|-- tenure: integer (nullable = true)
|-- PhoneService: string (nullable = true)
|-- MultipleLines: string (nullable = true)
|-- InternetService: string (nullable = true)
|-- OnlineSecurity: string (nullable = true)
|-- OnlineBackup: string (nullable = true)
|-- DeviceProtection: string (nullable = true)
|-- TechSupport: string (nullable = true)
|-- StreamingTV: string (nullable = true)
|-- StreamingMovies: string (nullable = true)
|-- Contract: string (nullable = true)
|-- PaperlessBilling: string (nullable = true)
|-- PaymentMethod: string (nullable = true)
|-- MonthlyCharges: double (nullable = true)
|-- TotalCharges: string (nullable = true)
|-- LeftMembership: string (nullable = true)
```

```
✓ 2s [ ] df.count()

7043
```

Figure 8: Columns type and counting data.

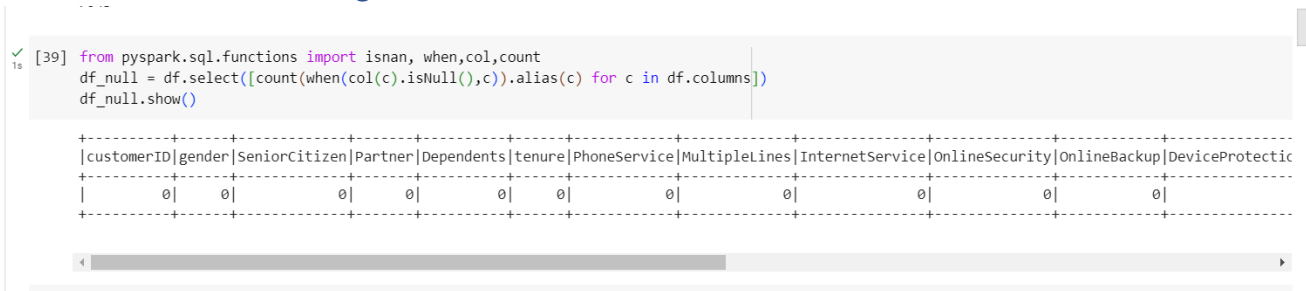


Figure 9: Checking the null values.

Tableau:

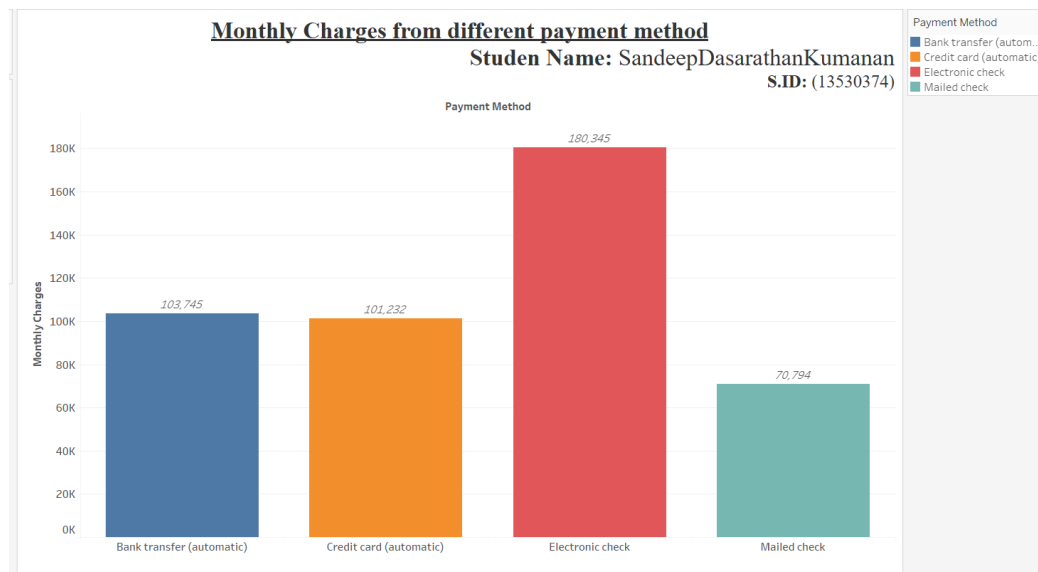


Figure 10: Monthly Charges from different payment method

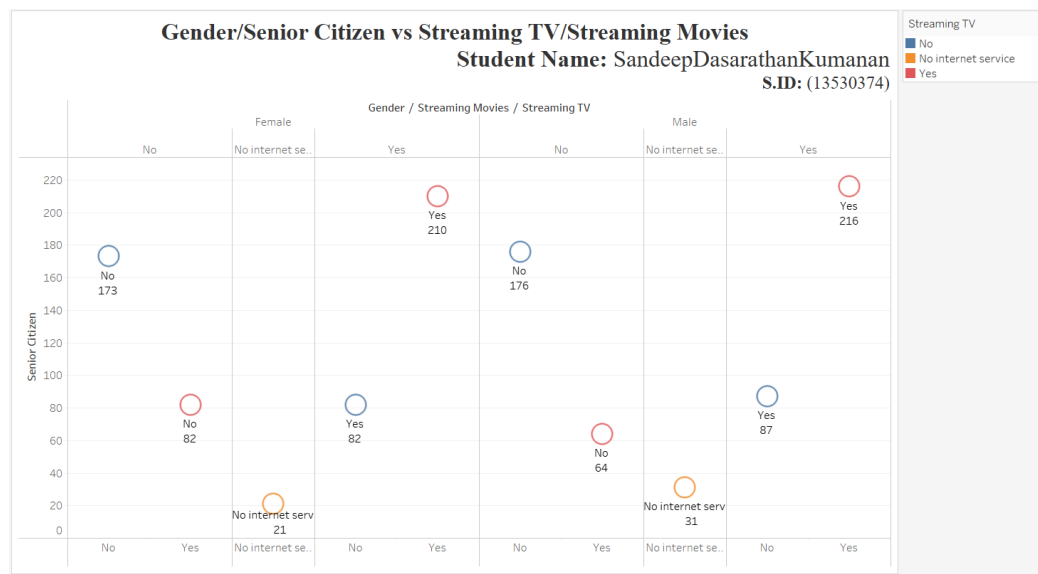


Figure 11: Gender/Senior Citizen vs Streaming TV/Streaming Movies



Figure 12: Contract of Customers

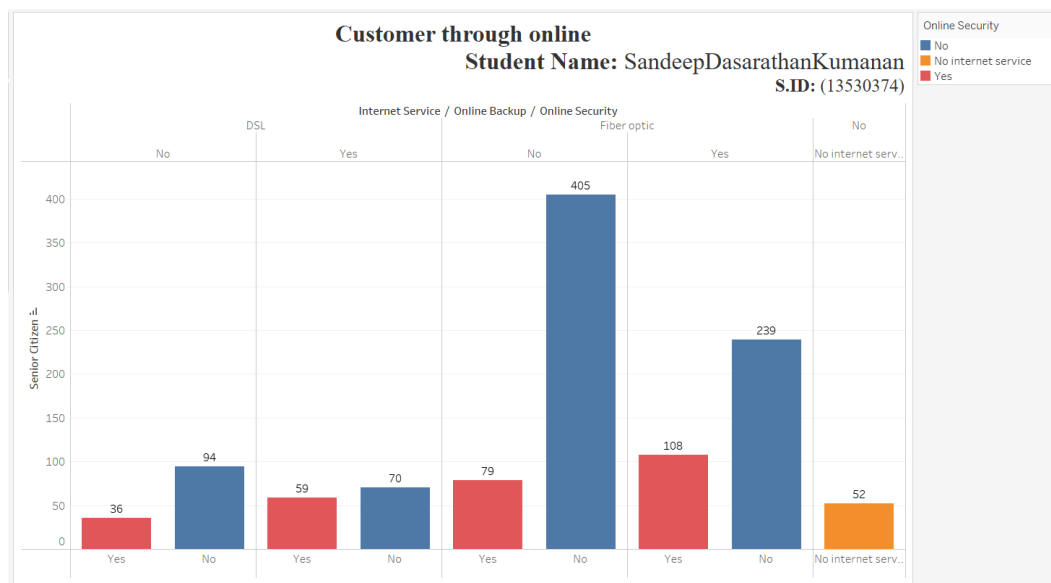


Figure 13: Customer through online

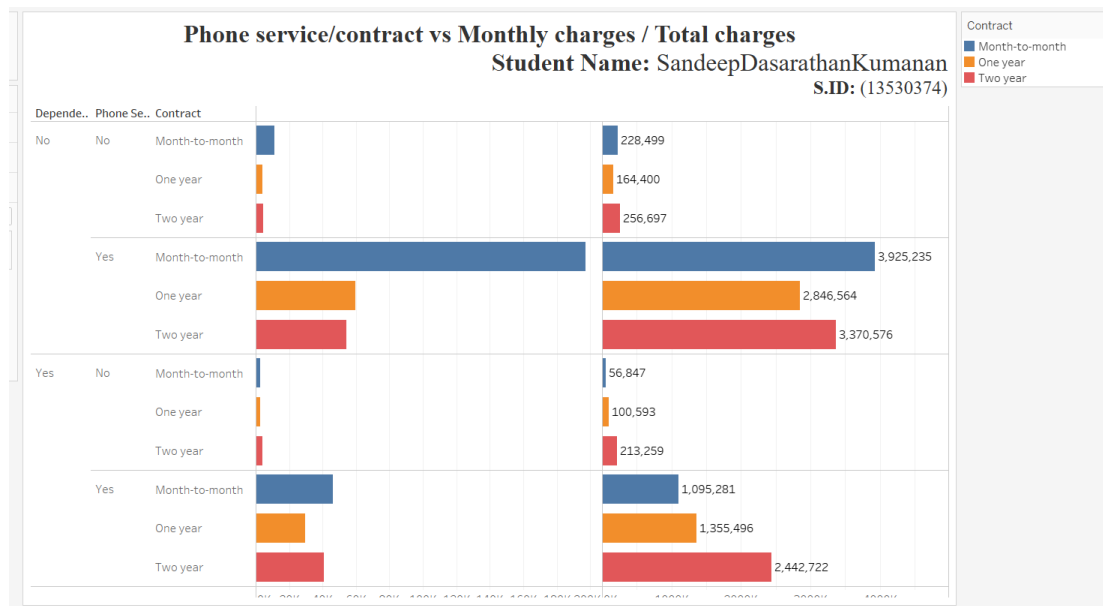


Figure 14: Phone service/contract vs Monthly charges / Total charges

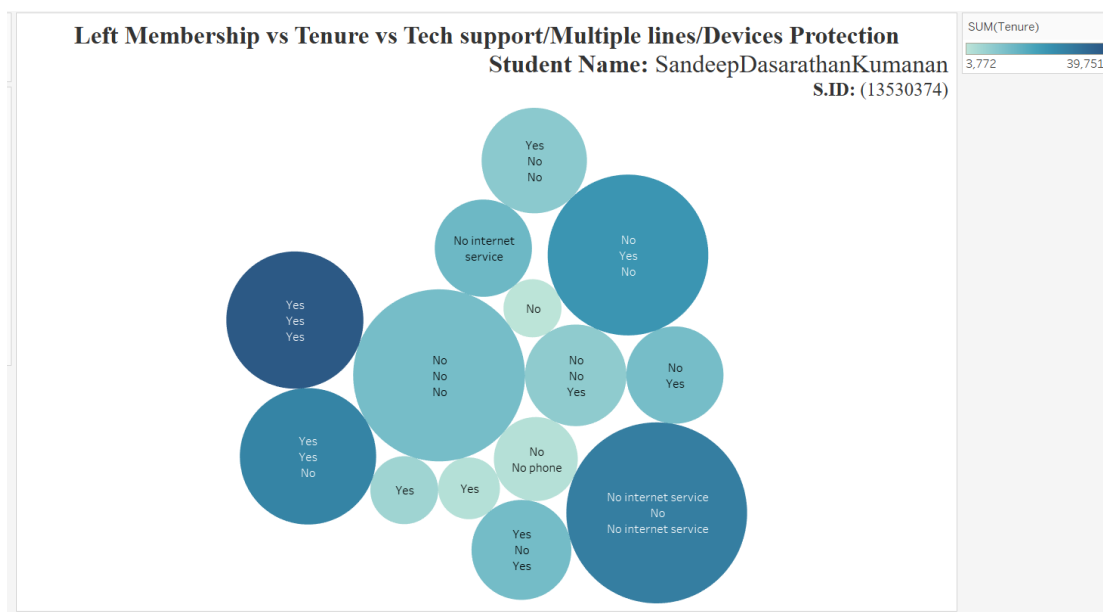


Figure 15: Left Membership vs Tenure vs Tech support/Multiple lines/Devices Protection

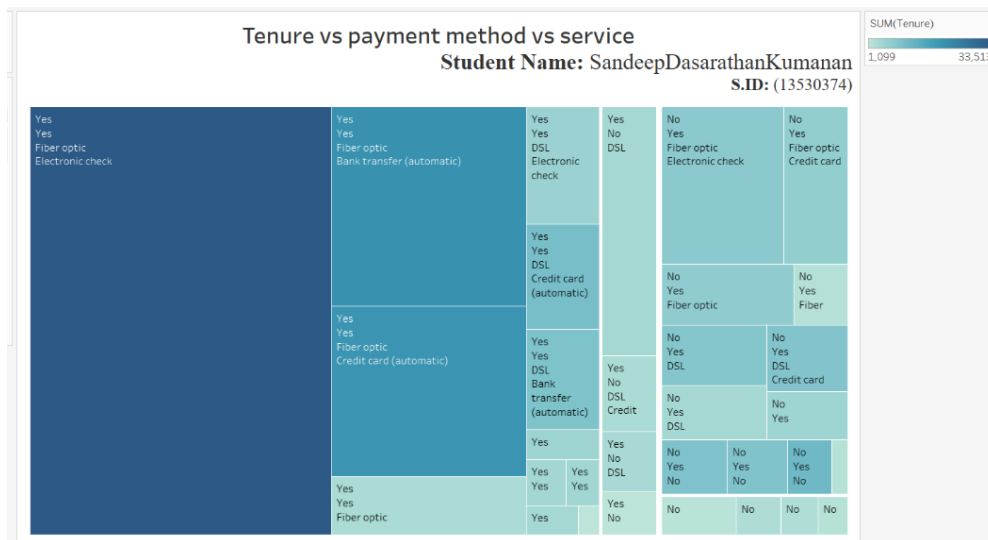


Figure 16: Tenure vs payment method vs service

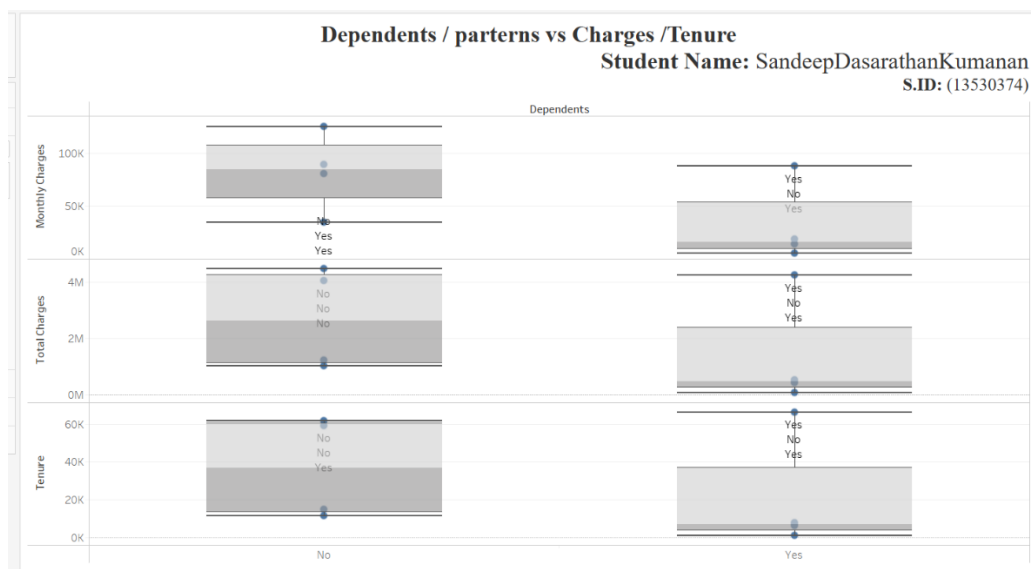


Figure 17: Dependents / partners vs Charges /Tenure

```
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import DecisionTreeClassifier, RandomForestClassifier, LogisticRegression, GBTClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

Figure 18: Implementing the algorithm.

▼ Decision Tress

```
[ ] dt = DecisionTreeClassifier(labelCol='label', featuresCol='features')
```

Figure 19: Fitting the value to features and target column.


```
[ ] evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy1 = evaluator.evaluate(predictions)
```

```
[ ] print(f"DecisionTree Test Accuracy: ",accuracy1)
print("Test Error = %" % (1.0 - accuracy1))
```

DecisionTree Test Accuracy: 0.7653213751868461
Test Error = 0.234679

Figure 20: Accuracy of Decision Tree

```
[26] evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",metricName="f1")
f1_dt = evaluator.evaluate(predictions)
print("f1 for Decision Tree Classifier:", f1_dt)
```

f1 for Decision Tree Classifier: 0.7458719582506969

```
[27] evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",metricName="weightedPrecision")
weightedPrecision_dt = evaluator.evaluate(predictions)
print("weightedPrecision for Decision Tree Classifier:", weightedPrecision_dt)
```

weightedPrecision for Decision Tree Classifier: 0.7445230723434431

```
[28] evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",metricName="weightedRecall")
weightedRecall_dt = evaluator.evaluate(predictions)
print("weightedRecall for Decision Tree Classifier:", weightedRecall_dt)
```

weightedRecall for Decision Tree Classifier: 0.7653213751868461

```
[29] print("Accuracy: ",accuracy1)
print("f1 for Decision Tree Classifier:", f1_dt)
print("weightedPrecision for Decision Tree Classifier:", weightedPrecision_dt)
print("weightedRecall for Decision Tree Classifier:", weightedRecall_dt)
```

Accuracy: 0.7653213751868461
f1 for Decision Tree Classifier: 0.7458719582506969
weightedPrecision for Decision Tree Classifier: 0.7445230723434431
weightedRecall for Decision Tree Classifier: 0.7653213751868461

Figure 21: Decision Tree Weight of Recall, f1, Weight Precision values

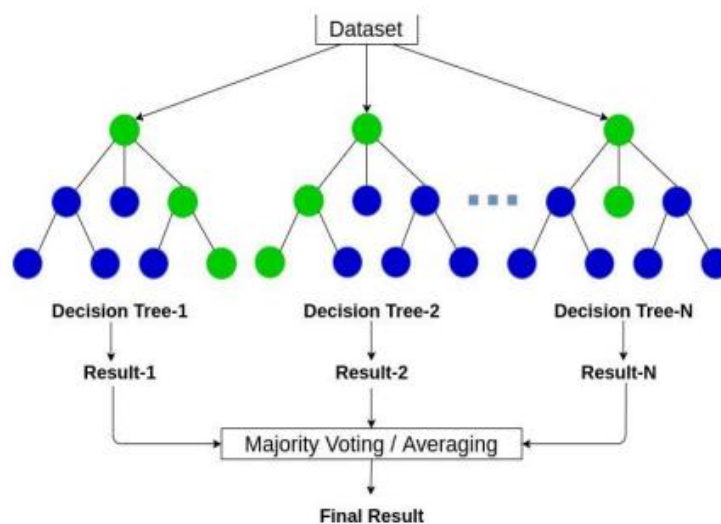


Figure 22: Working principle of Random Forest.

```
[ ] evaluator2 = MulticlassClassificationEvaluator(labelCol="label", metricName="accuracy")
```

```
[ ] accuracy2 = evaluator2.evaluate(predictions)
print("Random Forest accuracy = ",accuracy2)
print("Test Error= %g" %(1.0-accuracy2))
```

Random Forest accuracy = 0.7698056801195815
Test Error= 0.230194

Figure 23: Accuracy of Random Forest.

```
[ ] evaluator2 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",metricName="f1")
f1_rf = evaluator2.evaluate(predictions)
print("f1 for Random Forest Classifier:", f1_rf)
```

f1 for Random Forest Classifier: 0.758889647835172

```
[ ] evaluator2 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",metricName="weightedPrecision")
weightedPrecision_rf = evaluator2.evaluate(predictions)
print("weightedPrecision for Random Forest Classifier:", weightedPrecision_rf)
```

weightedPrecision for Random Forest Classifier: 0.7561859707331716

```
[ ] evaluator2 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",metricName="weightedRecall")
weightedRecall_rf = evaluator2.evaluate(predictions)
print("weightedRecall for Random Forest Classifier:", weightedRecall_rf)
```

weightedRecall for Random Forest Classifier: 0.7658196312904832

```
[ ] print("Accuracy: ",accuracy2)
print("f1 for Random Forest Classifier:", f1_rf)
print("weightedPrecision for Random Forest Classifier:", weightedPrecision_rf)
print("weightedRecall for Random Forest Classifier:", weightedRecall_rf)
```

Accuracy: 0.7698056801195815
f1 for Random Forest Classifier: 0.758889647835172
weightedPrecision for Random Forest Classifier: 0.7561859707331716
weightedRecall for Random Forest Classifier: 0.7658196312904832

Figure 24: Random Forest Weight of Recall, f1, Weight Precision values

```
[ ] evaluator3 = MulticlassClassificationEvaluator(
    labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy3 = evaluator3.evaluate(predictions)
print("Accuracy: ",accuracy3)
print("Test Error = %g" %(1.0 - accuracy3))
```

Accuracy: 0.772795216741405
Test Error = 0.227205

Figure 25: Accuracy of Gradient Boosting Tree.

```
[ ] evaluator3 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
f1_gbt = evaluator3.evaluate(predictions)
print("f1 for GBT Classifier:", f1_gbt)

f1 for GBT Classifier: 0.758889647835172

[ ] evaluator3 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedPrecision")
weightedPrecision_gbt = evaluator3.evaluate(predictions)
print("weightedPrecision for GBT Classifier:", weightedPrecision_gbt)

weightedPrecision for GBT Classifier: 0.7561859707331716

[ ] evaluator3 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedRecall")
weightedRecall_gbt = evaluator3.evaluate(predictions)
print("weightedRecall for GBT Classifier:", weightedRecall_gbt)

weightedRecall for GBT Classifier: 0.7658196312904832

print("Accuracy: ", accuracy3)
print("f1 for GBT Classifier:", f1_gbt)
print("weightedPrecision for GBT Classifier:", weightedPrecision_gbt)
print("weightedRecall for GBT Classifier:", weightedRecall_gbt)

Accuracy: 0.772795216741405
f1 for GBT Classifier: 0.758889647835172
weightedPrecision for GBT Classifier: 0.7561859707331716
weightedRecall for GBT Classifier: 0.7658196312904832
```

Figure 26: Gradient Boosting Tree Weight of Recall, f1, Weight Precision values.

```
[ ] evaluator5 = MulticlassClassificationEvaluator(
    labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy5 = evaluator5.evaluate(predictions5)
print("Accuracy: ", accuracy5)
print("Test Error = %g" % (1.0 - accuracy5))

Accuracy: 0.7658196312904834
Test Error = 0.23418
```

Figure 27: Accuracy of GBT

```
[ ] evaluator5 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
f1_lr = evaluator5.evaluate(predictions5)
print("f1 for Logistic Regression Classifier:", f1_lr)

f1 for Logistic Regression Classifier: 0.7311190615367359

[ ] evaluator5 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedPrecision")
weightedPrecision_lr = evaluator5.evaluate(predictions5)
print("weightedPrecision for Logistic Regression Classifier:", weightedPrecision_lr)

weightedPrecision for Logistic Regression Classifier: 0.742845072399753

evaluator5 = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="weightedRecall")
weightedRecall_lr = evaluator5.evaluate(predictions5)
print("weightedRecall for Logistic Regression Classifier:", weightedRecall_lr)

weightedRecall for Logistic Regression Classifier: 0.7658196312904832

print("Accuracy: ", accuracy5)
print("f1 for Logistic Regression Classifier:", f1_lr)
print("weightedPrecision for Logistic Regression Classifier:", weightedPrecision_lr)
print("weightedRecall for Logistic Regression Classifier:", weightedRecall_lr)

Accuracy: 0.7658196312904834
f1 for Logistic Regression Classifier: 0.7311190615367359
weightedPrecision for Logistic Regression Classifier: 0.742845072399753
weightedRecall for Logistic Regression Classifier: 0.7658196312904832
```

Figure 28: Logistic Regression Weight of Recall, f1, Weight Precision values.

	Decision Tree	Random Forest	GBT Classification	Logistic Regression
Accuracy	0.76582	0.765321	0.772795	0.772795
Recall	0.765321375	0.772795217	0.772795217	0.765819631
f1 score	0.745871958	0.745064428	0.758889648	0.731119062
Precision	0.744523072	0.752474476	0.756185971	0.742845072

Figure 29: Table result of classification.

```
[24]
from pyspark import SparkFiles

from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import DecisionTreeClassifier, RandomForestClassifier, LogisticRegression, GBTClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

Figure 30: importing the StringIndexer and VectorAssembler features.

```
[14] stringIndexer = StringIndexer(inputCols=['gender','SeniorCitizen','Partner','Dependents','PhoneService','MultipleLines','InternetService','Onl
outputCols=['gender_index','SeniorCitizen_index','Partner_index','Dependents_index','PhoneService_index','Multipl
```

Figure 31: Converting the categorical values to numeric.

```
[ ] assembler=VectorAssembler(inputCols=['gender_index','SeniorCitizen_index','Partner_index','Dependents_index','PhoneService_index','MultipleLine:
outputCol='features')
```

Figure 32: Fitting the columns to features using VectorAssembler.

Train and Test Split

```
[ ] train_data, test_data = df.randomSplit([0.7, 0.3], seed=42)
```

Figure 33: Train and test split the dataset.

```
[ ] train_data.show()
```

hSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	LeftMembership
Yes	Yes	No	One year	Yes	Mailed check	65.6	593.3	No
No	No	Yes	Month-to-month	No	Mailed check	59.9	542.4	No
No	Yes	Yes	Month-to-month	Yes	Electronic check	98.0	1237.85	Yes
Yes	Yes	No	Month-to-month	Yes	Mailed check	83.9	267.4	Yes
Yes	Yes	Yes	Month-to-month	Yes	Credit card (auto...	69.4	571.45	No
Yes	No	No	Two year	Yes	Credit card (auto...	84.65	5377.8	No
Yes	Yes	No	Two year	No	Credit card (auto...	45.2	2460.55	No
Yes	Yes	Yes	Two year	Yes	Credit card (auto...	116.8	8456.75	No
No	No	No	Month-to-month	Yes	Electronic check	68.95	351.5	No
Yes	Yes	Yes	One year	No	Mailed check	61.25	1993.2	No
No	No	No	Month-to-month	Yes	Electronic check	72.1	72.1	No
No	No	Yes	One year	No	Credit card (auto...	62.7	2791.5	Yes
service	No internet service	No internet service	One year	No	Electronic check	25.2	1306.3	No
No	Yes	No	Month-to-month	Yes	Electronic check	83.75	1849.95	No
No	No	No	Month-to-month	No	Bank transfer (au...	30.5	30.5	Yes
Yes	Yes	Yes	One year	Yes	Bank transfer (au...	103.7	5656.75	No
service	No internet service	No internet service	Two year	No	Credit card (auto...	20.4	1090.6	No
service	No internet service	No internet service	One year	No	Bank transfer (au...	20.45	900.9	No
service	No internet service	No internet service	One year	No	Electronic check	20.55	1343.4	No
No	Yes	No	Month-to-month	Yes	Bank transfer (au...	89.8	4959.6	No

Figure 34: Train data

```
[ ] test_data.show(10)
```

customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup
0004-TLHLJ	Male	0	No	No	4	Yes	No	Fiber optic	No	No
0013-SMEOE	Female	1	Yes	No	71	Yes	No	Fiber optic	Yes	Yes
0015-UOCOJ	Female	1	No	No	7	Yes	No	DSL	Yes	No
0016-QLJIS	Female	0	Yes	Yes	65	Yes	Yes	DSL	Yes	Yes
0019-EFAEP	Female	0	No	No	72	Yes	Yes	Fiber optic	Yes	Yes
0019-GFNTW	Female	0	No	No	56	No	No phone service	DSL	Yes	Yes
0020-INWCK	Female	0	Yes	Yes	71	Yes	Yes	Fiber optic	No	Yes
0023-HGHML	Male	1	No	No	1	No	No phone service	DSL	No	No
0023-XUOPT	Female	0	Yes	No	13	Yes	Yes	Fiber optic	No	Yes
0030-FNXPP	Female	0	No	No	3	Yes	No	No	No internet service	No internet service

only showing top 10 rows

Figure 35: Test data

```
[ ] pipeline = Pipeline(stages=[stringIndexer, assembler, dt])
```

Figure 36: Pipeline

```
[ ] paramGrid = ParamGridBuilder() \
    .addGrid(dt.maxDepth, [3, 5, 7]) \
    .addGrid(dt.minInstancesPerNode, [1, 3, 5]) \
    .build()
```

Figure 37: paramGrid

```
[ ] crossval = CrossValidator(estimator=pipeline, estimatorParamMaps=paramGrid,  
                             evaluator=MulticlassClassificationEvaluator(  
                                 labelCol='label', predictionCol='prediction', metricName='accuracy'),  
                             numFolds=5)
```

Figure 38: Cross Validator