PREDICTING CUSTOMER CHURN FOR A TELECOMMUNICATIONS COMPANY

Big Data Analytics and Visualisation using Pyspark and Tableau.

Student Name: Suleiman Adamu Yakubu

SID: 11796778

Module Code: 7153CEM

Module Name: Big Data Analytics and Data Visualisation

Module Leader: Dr Marwan Fuad

I can confirm that all the work submitted is my own: YES

Abstract

This paper analyses customer churn in telecommunication companies using a combination of algorithms to tease out trends and patterns in the data. The insights gleaned can help lower customer attrition rates by making us more knowledgeable about the more profitable sections of the customer base, and help improve services.

Big Data Analytics was supplied via PySpark and all Machine Learning (ML) algorithms were made available using MLLib of Apache Spark in a Jupyter Notebook. The data visualisations were made possible by Tableau Online.

Link to the dataset: https://www.kaggle.com/blastchar/telco-customer-churn

Contents

1. Introduction

Customer churn refers to the phenomenon where businesses lose customers due to one reason or another. In the telecommunications industry particularly, it means that the customer has typically moved to a competitor. This is obviously undesirable for companies for a number of reasons: it is cheaper for them to maintain older customers than it is to gain a new customer, and a lost customer indicates revenue loss. In most cases, the company has no idea why the customer has churned.

This report is meant to analyse the selected dataset, glean insights, and view trends that will enable the identification of the reasons for customer churn and will suggest ways to reduce the phenomenon.

As this report is the coursework of the Big Data and Data Analytics module. It is meant to demonstrate big data analytics concepts and how insight can be gleaned from data. This report also covers how to install and set up a number of programs such as Tableau (for Data Visualisation), PySpark (for Big Data Analytics), and Jupyter Notebook for the programming aspect of the report.

We will employ a combination of data analysis algorithms to analyse the dataset in question and make sense of the results.

1.1 Aim and Objectives

The aim of the project is to analyse the selected dataset using a combination of algorithms such as the Random Forest, Regression, and KNN and predict churn. The following objectives support this aim

1. Install Spark, Pyspark, and Jupyter Notebook.

2. Perform Exploratory Data Analysis and visualisation of the dataset using Pyspark and Tableau
3. Select algorithms to perform analysis of the data and build models.
4. Compare the performance of the selected algorithms
5. Diagnose the reasons behind the customer churn and proffer solutions if possible.

2. Setup and Configuration

This section of the report covers the tools used and how they were installed. The work was implemented using Pyspark, which provides an interface to Apache Spark using Python. Apache Spark on the other hand, is an engine that allows machine learning, data science, and data engineering tasks to be executed in a distributed manner. This is especially important when dealing with Big Data.

Other tools used include; Tableau (for Data visualisation) and Jupyter Notebooks for programming. All of these were installed on a Linux Ubuntu 20.04 partition with 4GB of RAM and 50GB of hard disk available

2.1 Pyspark installation

To successfully install Pyspark on the machine in question, the instructions in Blismos Academy's video (Blismos Academy, 2021), were followed.

Pyspark has the following requirements: Apache Spark, Java, and Python. Furthermore, to gain a programming environment for the project, Jupyter Notebook was installed via Anaconda. To configure Jupyter Notebook for Pyspark, instructions from The Intuitive Vibe's (The Intuitive Vibe, 2020) YouTube video were followed.

The steps of the installation are as follows:

1. Open a terminal window

2. Install Java
3. Download and extract Apache Spark
4. Install Python and pip
5. Write the appropriate environmental variables to the path.
6. Install Pyspark

The following figures cover the steps taken to install Pyspark.

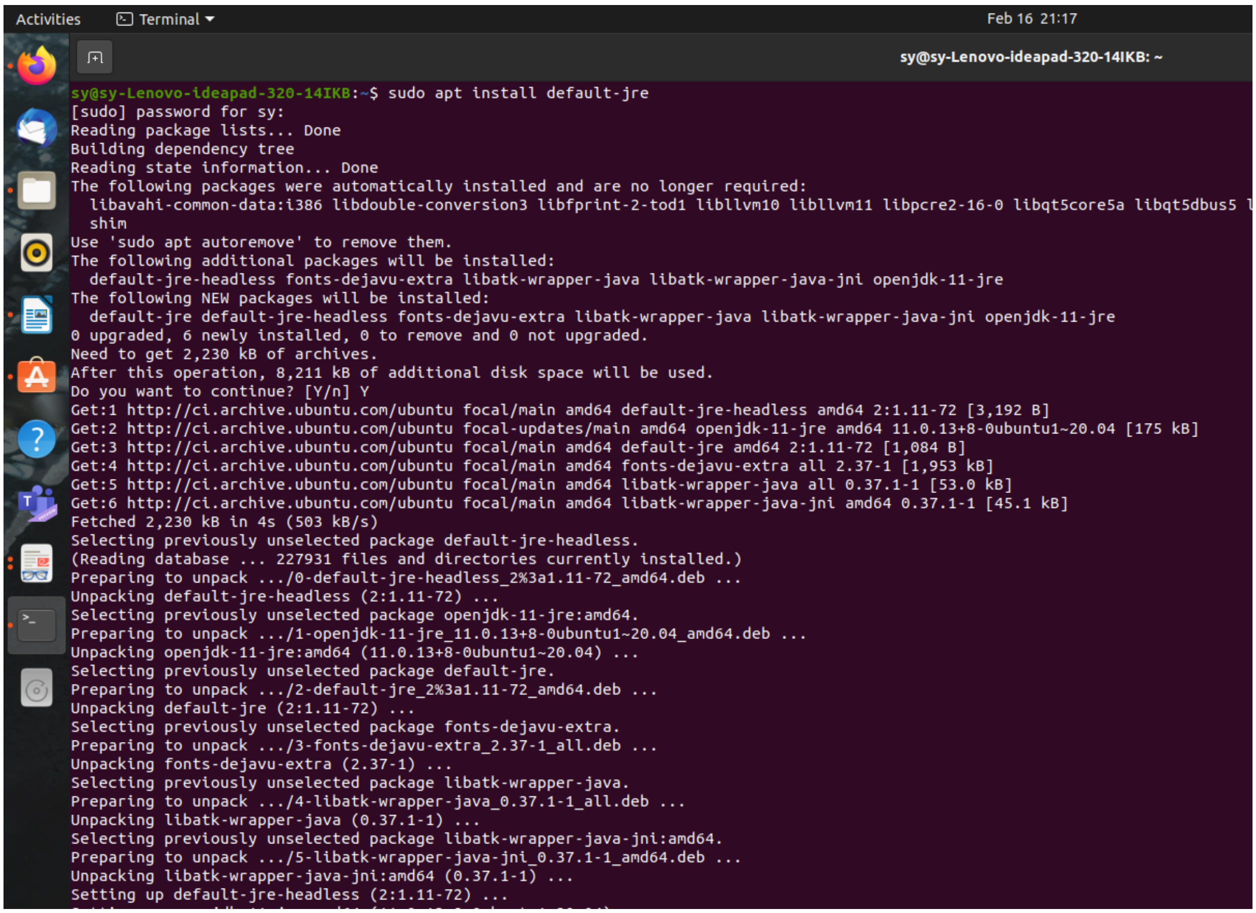


Figure 1. Installing the JRE from the terminal

After the JRE was installed, Java Development Kit (JDK) was then installed using the terminal

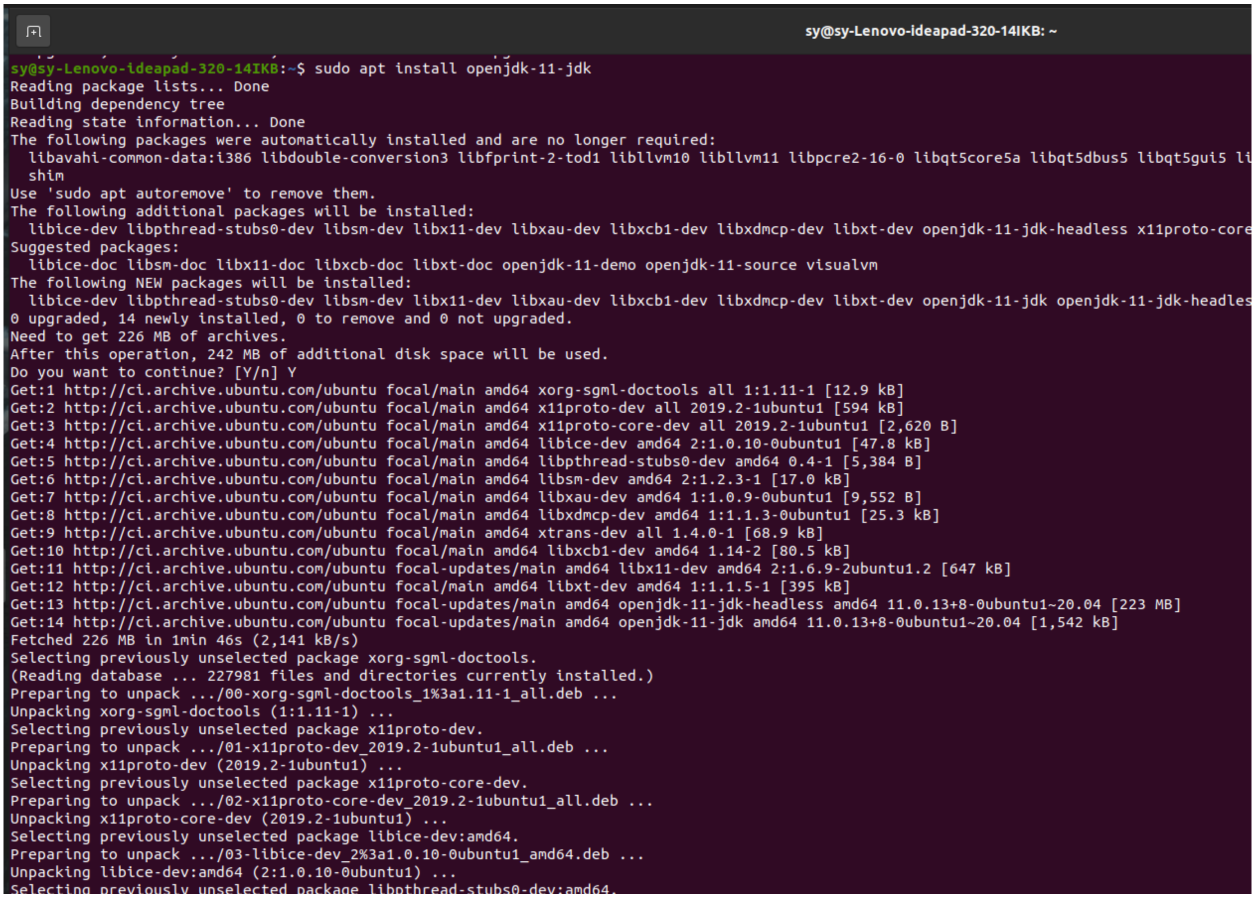


Figure 2. Installing the JDK using the terminal

The next step is the installation of Apache Spark. In this case, a slightly more complex method was used. Using Superuser privileges, a folder called Spark, was created in the opt folder of the home/sy directory. The Spark tar file was then downloaded directly into this folder and extracted. The version of Spark used is version 3.1.2.

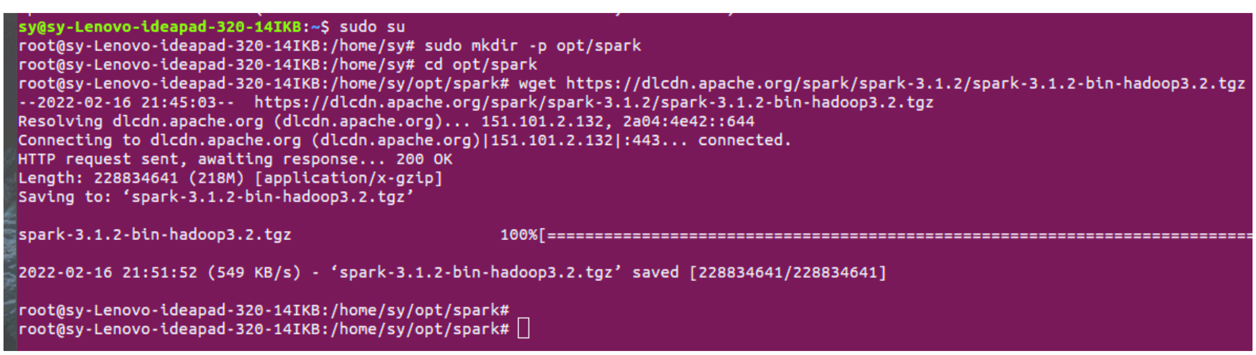


Figure 3. Downloading Apache Spark

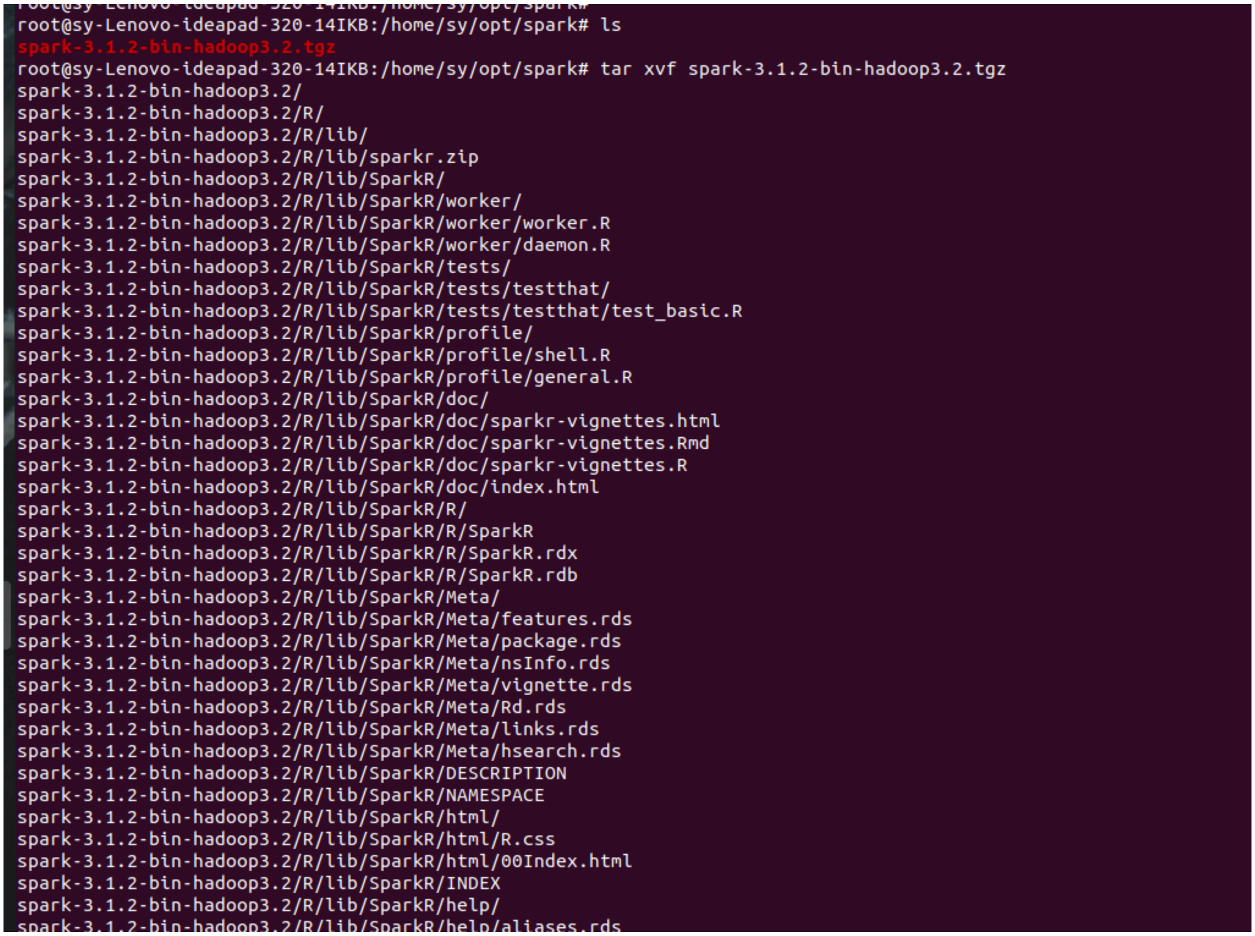


Figure 4. Extracting Apache Spark tar file into the spark directory

Next, we check to see if Apache Spark is installed and working properly. This is shown in the below figure.

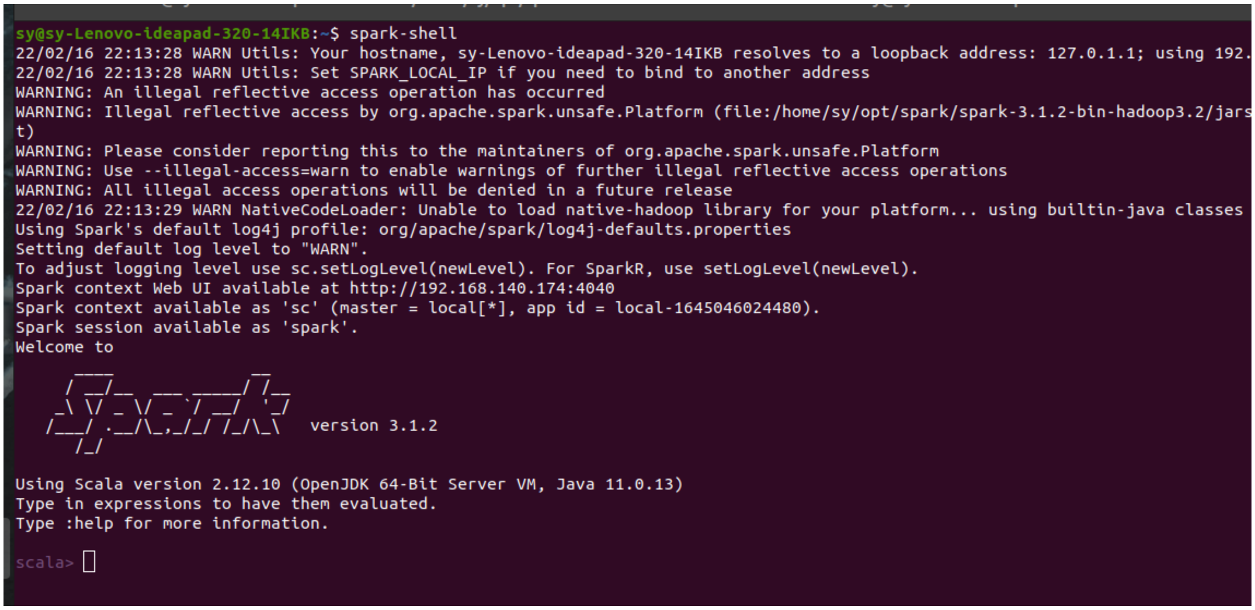


Figure 5. Running the Spark shell

Now that we have ascertained that Spark is working without incident, we need to install Python and pip.

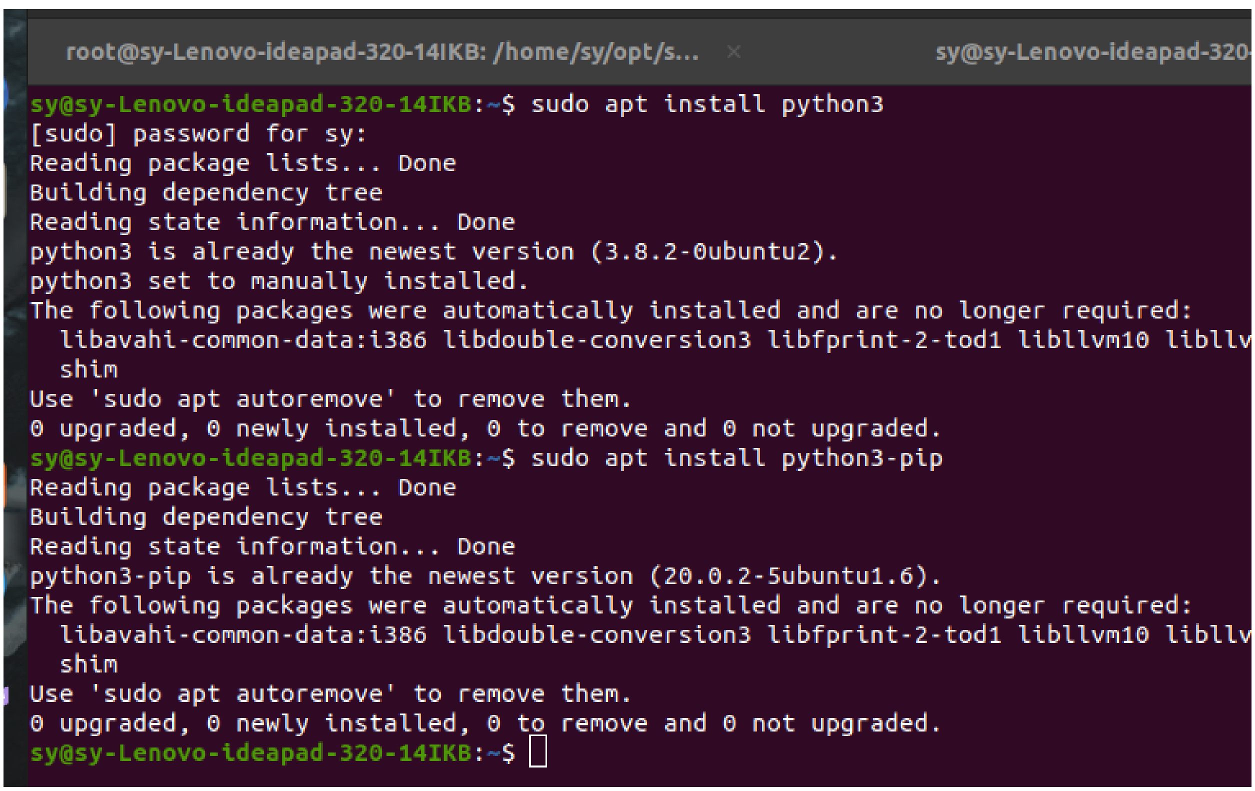


Figure 6. Installing Python and pip

As the latest versions of Python and pip were previously installed, no new files were downloaded. However, we need to add some environment variables to the *PATH*. This is done by editing the .bashrc file and adding the variables to the path there

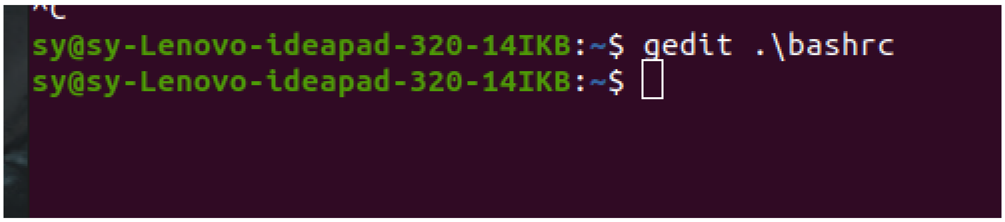


Figure 7. Editing the .bashrc file

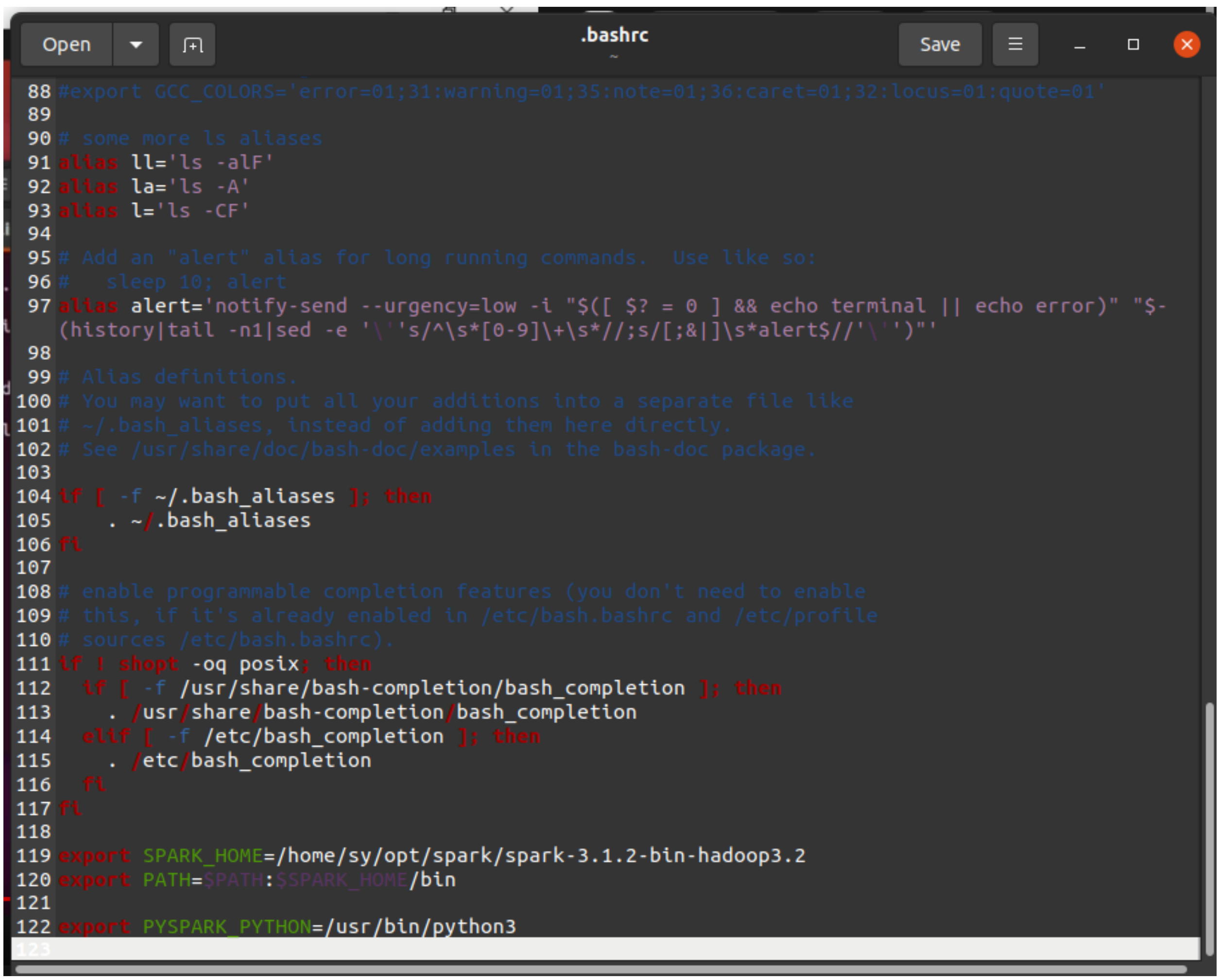


Figure 8. Adding the relevant environmental variables to PATH

The variables added to the path include the path to the downloaded and extracted Spark package (as SPARK_HOME) and instructions telling Apache Spark to use the installed Python as its default Python. This means we can then use Python to communicate with Spark in the terminal using the Pyspark command.

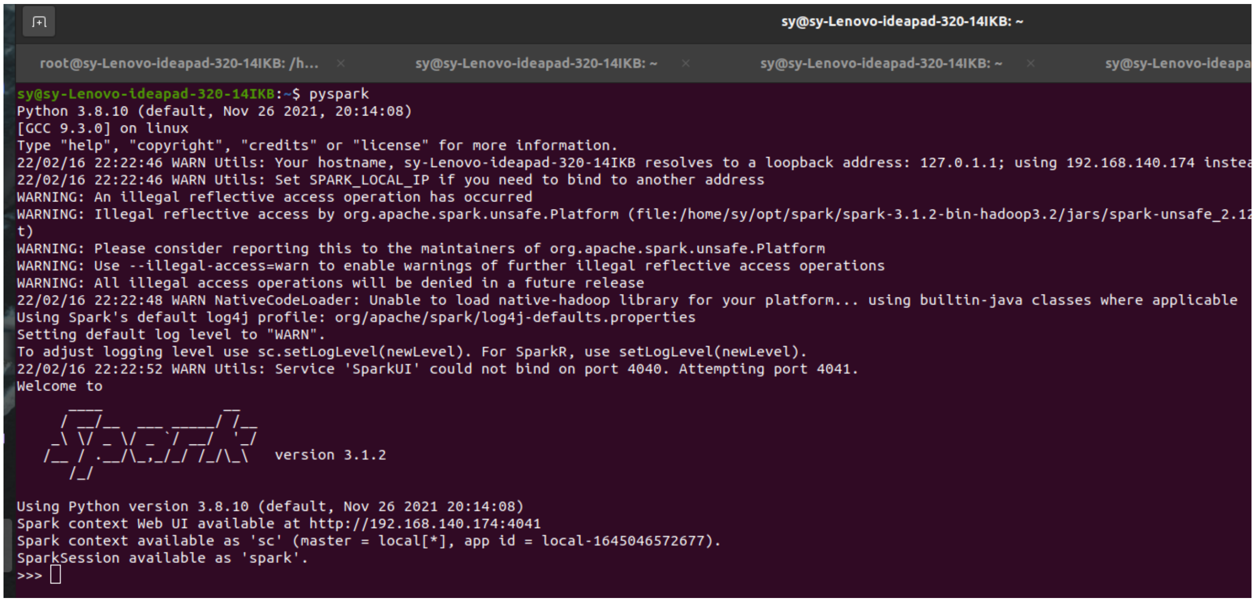


Figure 9. Running Pyspark from the terminal

2.2 Installing and Configuring Jupyter Notebooks

Now that we have Pyspark installed, what we need next is a friendlier environment than the terminal to use it. The environment selected for this report is Jupyter Notebook. This was installed by way of Anaconda which is a Python distribution used for Data Science, Machine Learning, and Data Analytics tasks (Anaconda Inc, n.d.).

Anaconda comes prepackaged with Anaconda Navigator, which provides a GUI interface to Anaconda. Most interestingly, Anaconda Navigator gives us access to Jupyter Notebook.

The Anaconda installer was downloaded from the website and installed following the instructions found there.

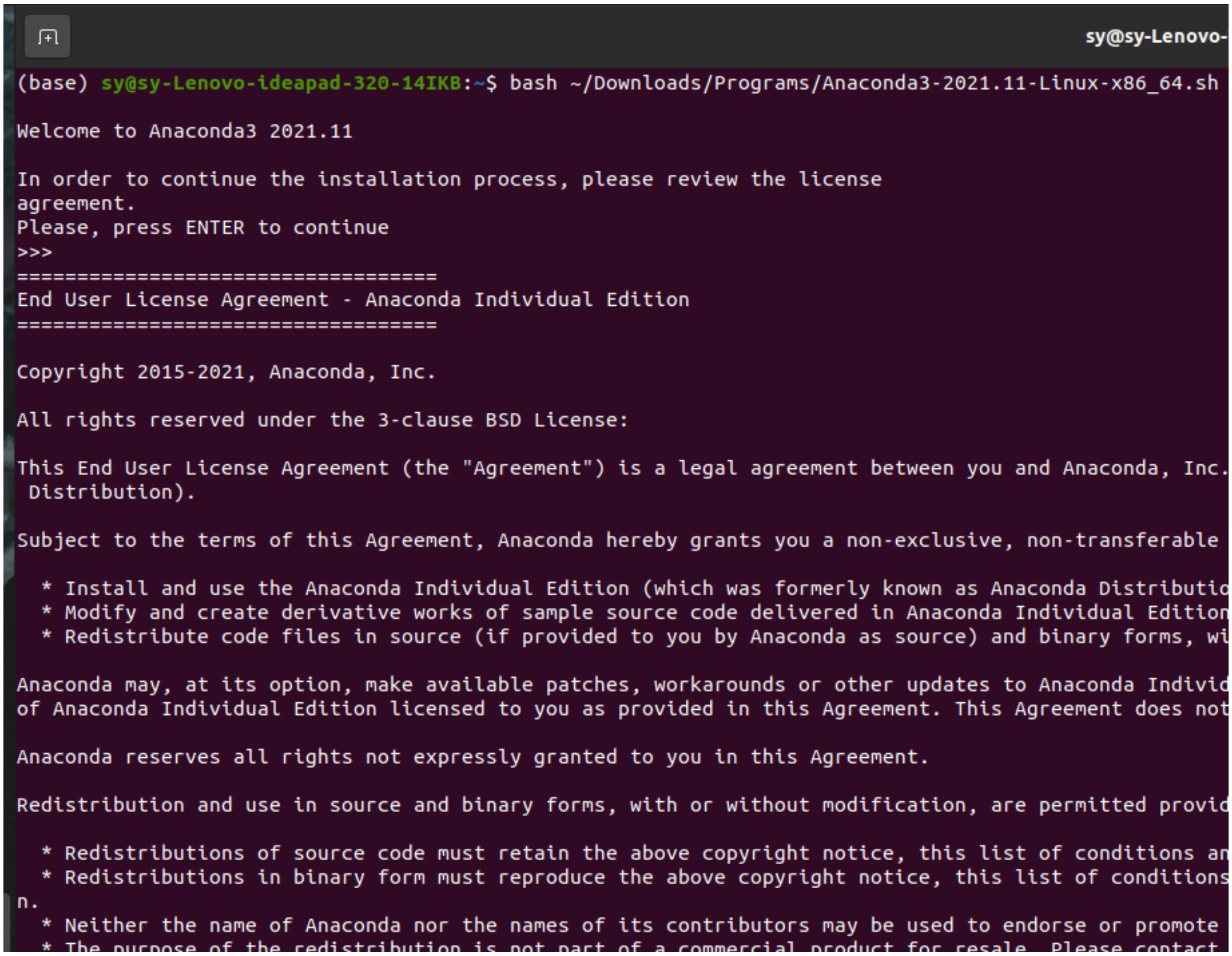


Figure 10. Installation of Anaconda

After the installation is done, the .bashrc file needs to be modified as well. This is to enable Jupyter Notebook to be able to access Pyspark. The lines added to the file are lines 123 to 125 in the below figure.

```
# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

export SPARK_HOME=/home/sy/opt/spark/spark-3.1.2-bin-hadoop3.2
export PATH=$PATH:$SPARK_HOME/bin

export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH
export PYSPARK_DRIVER_PYTHON="jupyter"
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
export PYSPARK_PYTHON=/usr/bin/python3


# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/home/sy/anaconda3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
```

Figure 11. Configuring Jupyter Notebook for Pyspark

Now, we are ready to begin the analysis. To allow Jupyter Notebook access to Pyspark, we need to open a terminal and run Jupyter Notebook from the following path: /home/sy/opt/spark/spark-3.1.2-bin-hadoop3.2/python.

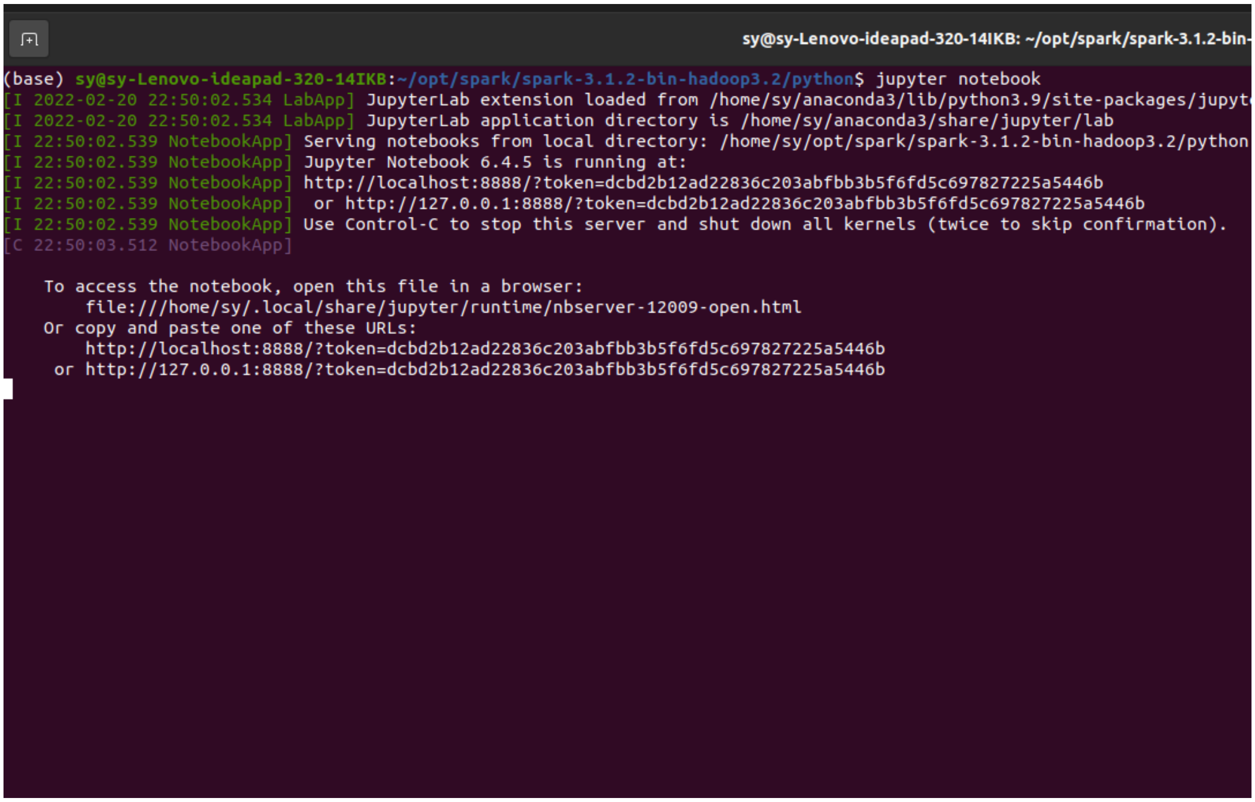


Figure 12. Starting Jupyter Notebook

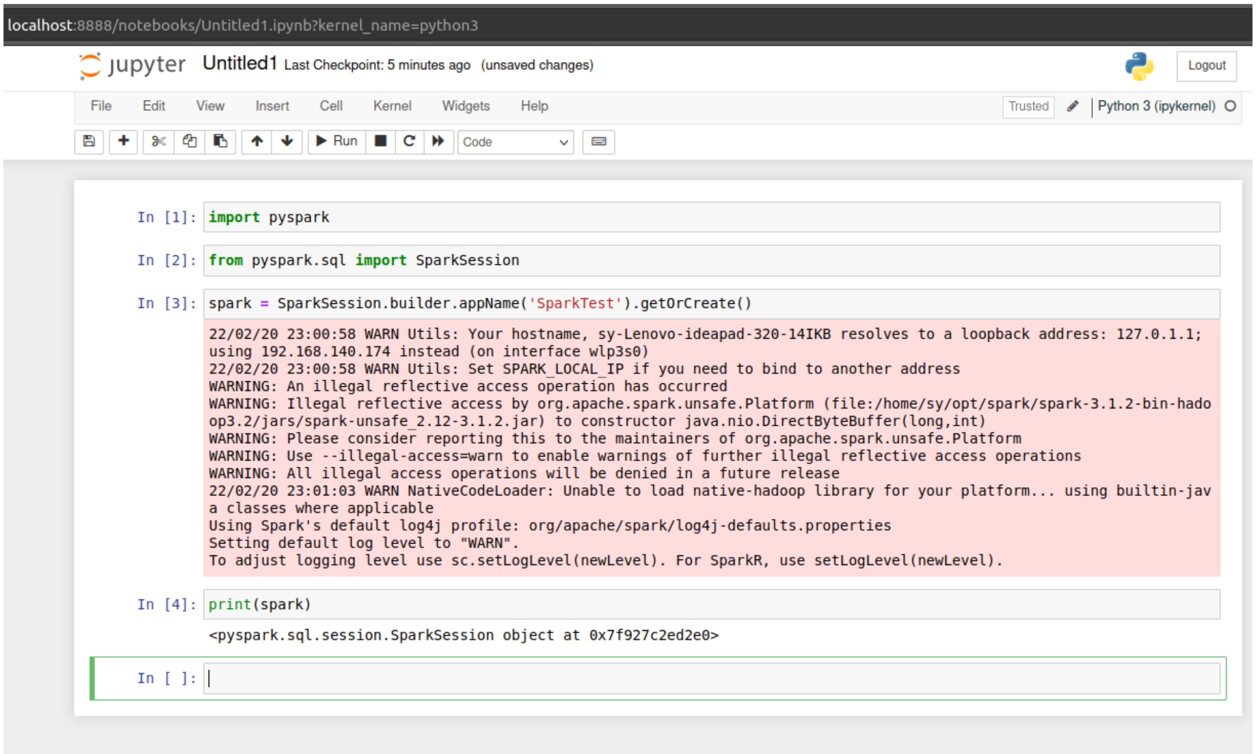


Figure 13. Creating a Spark Session in Jupyter Notebook

2.3 Installing Tableau

Tableau is a visual analytics platform that simplifies the ways that data is presented. It provides a simple interface for non-technical people to see, analyse, and gain insights from data (Perry, n.d.).

For this report, Tableau Online was used because it is less resource-intensive and more accessible. This provides all the functionality of Tableau over the cloud. Accessing it is as simple as creating an account and then logging in.

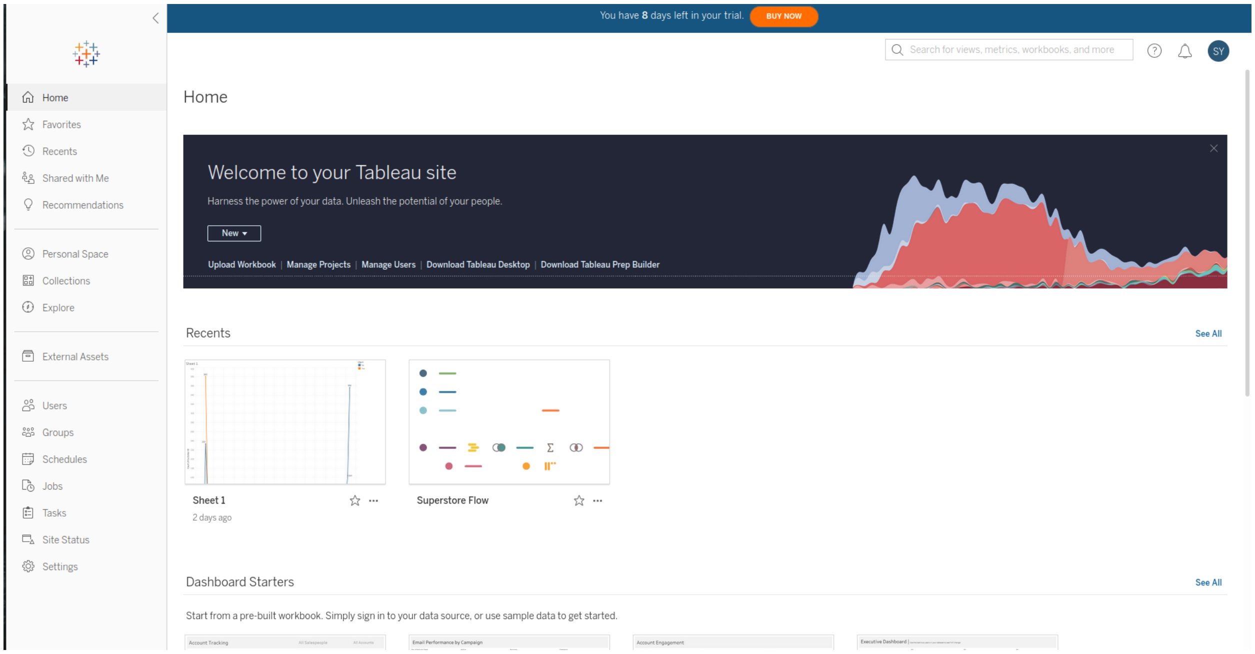


Figure 12. Tableau Online after login

3. The Dataset

The dataset used in this report is called the Telco Customer Churn dataset and is meant to help with focused customer retention programs. The dataset contains information about customer characteristics and whether that customer churned or not. The dataset is inspired by the IBM sample datasets (Blastchar, 2018).

In the dataset, which contains 7048 rows, each row represents a customer and contains information such as whether the customer left in the past month or not (churn), information on services that the customer is subscribed to, account information, and demographic information (Blastchar, 2018).

To further understand the dataset, some visualisations were carried out using Tableau Online. The data dictionary is shown below:

Attribute	**Data Type**	**Description**
CustomerID	String	A unique string that identifies the customer
Gender	String (Male or Female)	The Gender of the customer

SeniorCitizen	String (Yes or No)	Indicates if the customer is a Senior Citizen or not
Partner	String (Yes or No)	Indicates if the customer has a significant other
Dependents	String (Yes or No)	Indicates if the customer has dependents
Tenure	Integer	Indicates how many months the customer used the Telco's services
PhoneService	String (Yes or No)	Indicates if the customer has home phone service
MultipleLines	String (Yes or No, No Phone Service)	Indicates if the customer has multiple telephone line subscriptions
InternetService	String (DSL, Fibre Optic, No)	Indicates if the customer subscribed for internet with the telco
OnlineSecurity	String (Yes, No, No Internet Service)	If the customer paid for online security
OnlineBackup	String (Yes, No, No Internet Service)	If the customer paid for an online backup service
DeviceProtection	String (Yes, No, No Internet Service)	If the customer subscribed to a device protection plan
TechSupport	String (Yes, No, No Internet Service)	If the customer subscribed for technical support
StreamingTV	String (Yes, No, No Internet Service)	Does the customer stream television programs from a 3rd party?
StreamingMovies	String (Yes, No, No Internet Service)	Does the customer stream movies from a 3rd party?
Contract	String (Month-to Month, Yearly, Two Year)	Current contract type

PaperlessBilling	String (Yes or No)	Whether customer has chosen paperless billing
PaymentMethod	String (Bank Transfer, Credit card (automatic), Electronic check, mailed check)	The method of payment
MonthlyCharges	Float	Monthly charges
TotalCharges	Float	Total charges incurred
Churn	String (Yes or No)	Whether the customer churned or not

Table 1. Data dictionary for the Telco Dataset ((IBM Sample Data Team & Macko, 2019)*)*

3.1 Exploratory Data Analysis

Before using any dataset, it is necessary to explore it. This is to ensure outliers, missing values, and incorrect values and data types are taken into account. We use some modules of Pyspark to load the dataset and perform analysis on it. Exploratory Data Analysis will be incomplete without visualisations. As per the conditions of this report, that was carried out using Tableau.

3.2 Exploring the Data with Pyspark

When using Pyspark, we have to import the relevant libraries first. As we are using Spark, we have to start a Spark Session. After that, we can load the data into a dataframe. The dataframes used in this report are from the pyspark.sql.SparkSession module. The code to do that is shown below:

```
#importing pyspark and starting a SparkSession
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('EDA').getOrCreate()

#read the dataset in with the column names
churn_df = spark.read.option('header',
'True').csv('WA_Fn-UseC_-Telco-Customer-Churn.csv', inferSchema=True)
```

Thus, we have initialised a Spark Session with the name EDA and loaded the dataset into a Spark Dataframe called churn_df. To get a clearer picture of the dataset, we can look at its schema. This is achieved as shown below:

```
churn_df.printSchema()
```

From this operation, we can see that the TotalCharges column is in string form when it should be a float. We have to convert it.

```
churn_df=churn_df.withColumn('TotalCharges',
churn_df['TotalCharges'].cast('float'))
```

We can then safely work with the TotalCharges column. Next, we explore the possibility of null values in the dataset. Since null values will skew our analysis, we can count how many null values are in each column and drop them from the dataset if any are found.

```
churn_df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c)
for c in churn_df.columns]).show()
#Drop all null values in the dataset.
churn_df = churn_df.na.drop()
churn_df.count()
```

The above executed code shows that we have 11 null values in the TotalCharges column. We drop these null values and check the count of rows in the dataset. This then shows we have 7032 rows in the dataset. The data is then considered suitable for visualisation.

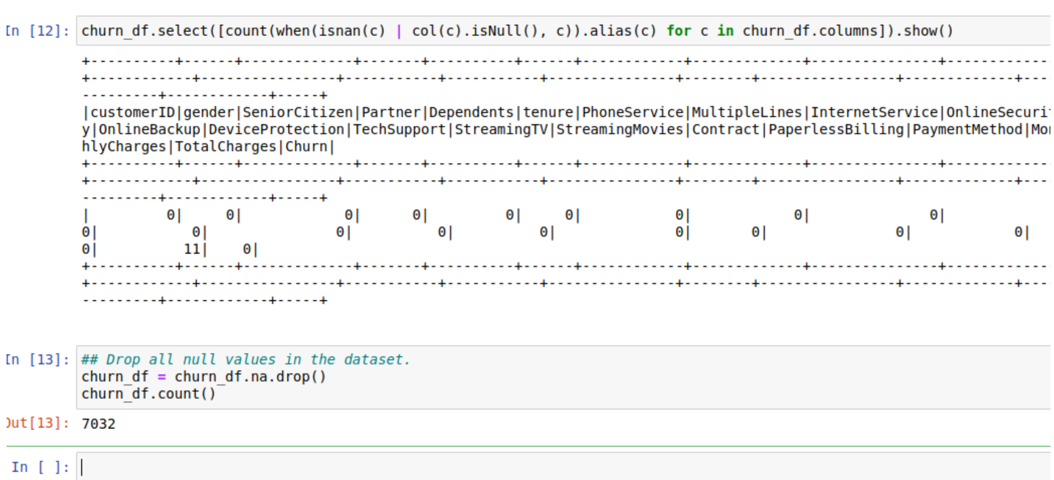


Figure 13. Checking and dropping null values in the dataset

3.3 Data Visualisation with Tableau

Data visualisation provides us the means to view data graphically and as a result glean insights that would not have been intuitively arrived at from just viewing the data in its raw form. Tableau is one of the most popular visualisation softwares on the market. It was used to visualise the data. This helped us view the relationships between attributes and even suggested which attributes were more important.

We first viewed how the categorical attributes affected churn, then we viewed the relationship between numerical data and churn.

3.4 Importing Data into Tableau

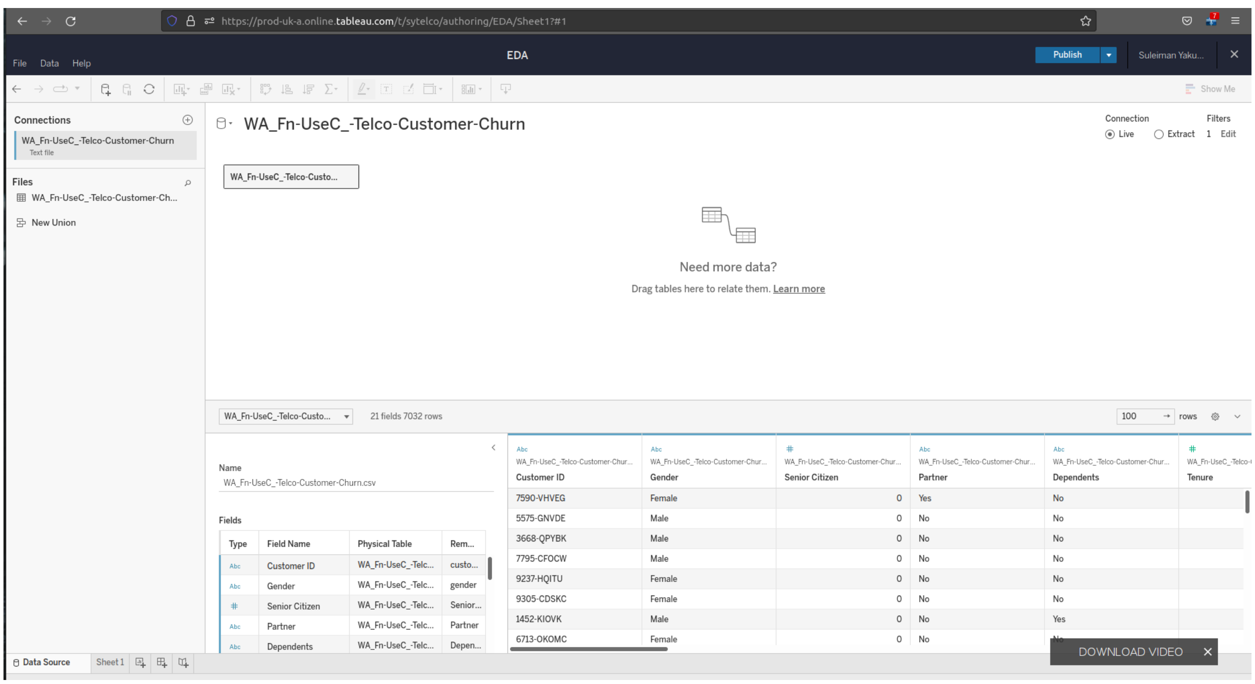


Figure 14. Importing data into Tableau

After the data is imported into Tableau, a filter was created for the TotalCharges column. This is because it was noticed that there were some null values in the column. The filter helps us to drop the null values to prevent skewing the analyses.

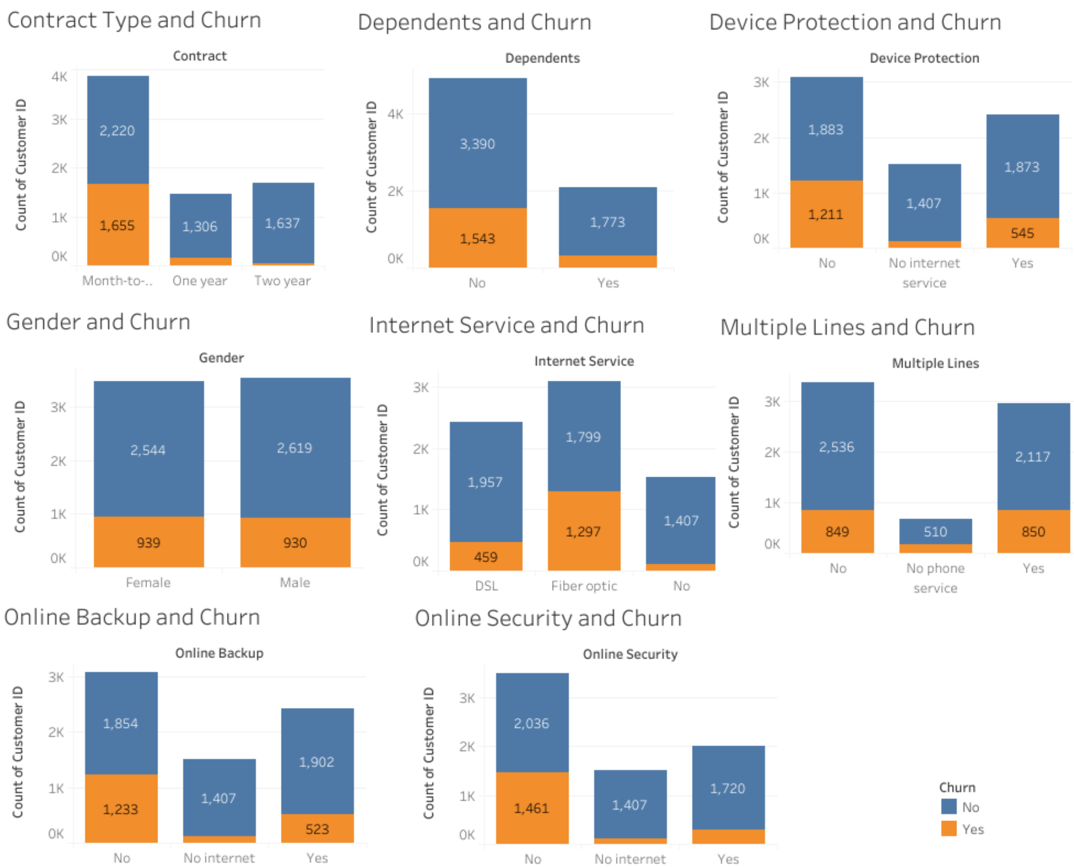


Figure 15. Some categorical attributes and their relationship with churn

In Tableau, different categorical attributes were visualised against the churn to see which of them most significantly affected it. When the contract type was modelled against the churn, we noticed most of the customers (3875) used a month-to-month contract. Out of those, 1,655 churned. This is 42.7097% of customers with month-to-month contracts as opposed to 168 from one-year contracts and 48 from two-year contracts which accounts for 11% and 2% respectively.

Overall, it was noticed that customers that did not pay for internet or internet-based services at all had the least churn out of all the categories of customers. This might imply that the company's internet service left much to be desired. However they were a noticeably smaller group compared to the others.

When gender comes into play, the difference is negligible. The company was found to be balanced between its female and male customers. The customers were not found to churn based on gender.

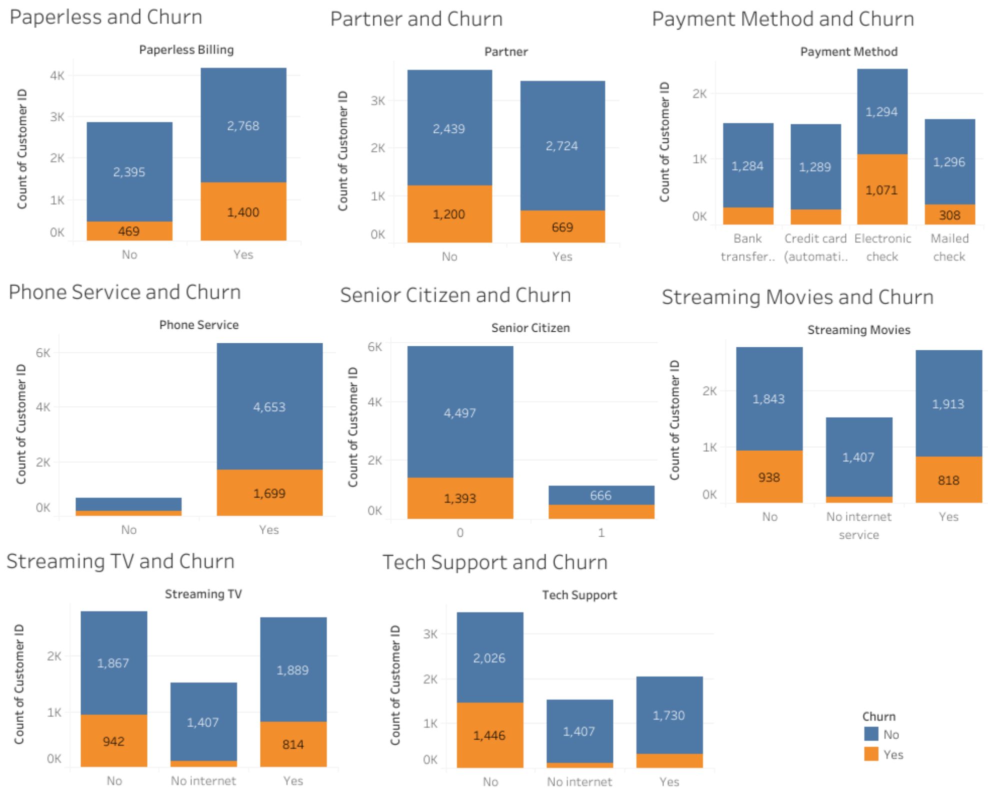


Figure 14. Some categorical attributes plotted against churn

When methods of payment were analysed, it was found that customers who subscribed for paperless billing churned significantly more those that did not. Again, this can suggest that the company's paperless billing service is not as efficient as it might be. From payment methods, the electronic check payment group had the most churn.

Senior citizens had a high percentage of churn, even though they accounted for a very small percentage of overall customers. Again the data shows that the group of customers who subscribed to services that relied on the internet like streaming movies, or tech support had a higher churn percentage than the other group.

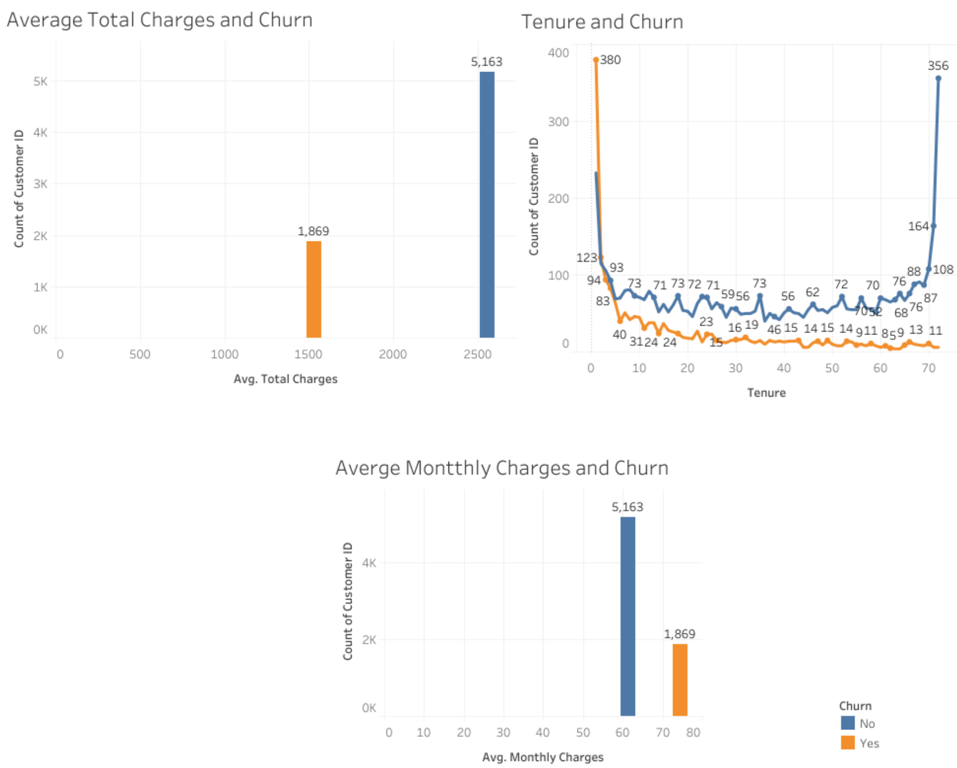


Figure 15. Some continuous attributes of against Churn

It was found that customers tended to be more likely to churn in the earlier days of their subscription to the telecommunications company's services with the churn percentage being highest within the first five (5) months of subscription. After that, the churn rate experienced a sharp drop and apart from a few fluctuations, kept dropping the longer the customer stayed with the company. This could imply the company is focusing on the older customers to the detriment of the newer ones.

When analysing the effect of monthly charges and total charges on churn, a summary of the data was used. The mean was used because using the charges of each customer was noisy and patterns were difficult to tease out. The mean implied that the customers were, on average, comfortable paying between 60 and 70 currency units monthly, Going higher increased the likelihood of churn.

The average total charges chart told a different story. On average, customers with a higher total charge did not churn. This backs up the hypothesis that customers who have stayed longer with the company are less likely to churn since the longer a customer stays, the more their charges accumulate.

4. Machine Learning

One of the goals of this report is to analyse the dataset, make inferences, predict churn, and proffer solutions if possible. Machine learning models make this possible. For this report, the models used were Decision Trees, Random Forest, and Logistic Regression.

All the machine learning algorithms selected are classification algorithms. This is because predicting customer churn is fundamentally a classification problem. The end result of the prediction is either Churn or Not Churn.

NB: The code used in this project was referenced from the Apache Spark documentation and YouTube tutorials. The code was then customised according to the needs of the machine learning models and dataset used.

4.1 Data Preprocessing

Before we can apply machine learning algorithms to data, we need to perform some operations on the data to make it suitable. For instance, most machine learning algorithms cannot handle categorical data properly. We need to convert categorical data to its numerical equivalent before the algorithms can work on them (Garg, 2021).

Because of this we performed data exploration and cleaning on the data to make it suitable for the algorithms used. Extensive use of Pyspark's Dataframes API was made to make the data up to scratch. All the code used will be available in the appendix.

The first step was to import the relevant libraries: pyspark, SparkSession, etc. This enabled a Spark session to be initialised, and then the dataset was imported as well.

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql import types
from pyspark.sql.functions import col,isnan, when, count
spark = SparkSession.builder.appName('ML').getOrCreate()
churn_df = spark.read.csv('WA_Fn-UseC_-Telco-Customer-Churn.csv',
header=True, inferSchema=True)
```

After the dataset was read in, the *TotalCharges* column was converted to a double data type. This was done because during the data exploration and visualisation stage, it was noticed to be in string form. Further cleaning was performed on the dataset by dropping the 11 null values from the *TotalCharges* column noticed in the visualisation stage.. For the *MultipleLines* column, it was noticed that there were 3 possible values; “No”, “No phone service”, and “Yes”. All “No phone service” entries were converted to “No” as it amounts to the same in the end. A customer with no phone service will of course not have multiple lines.

Similarly, the “No internet service” values in the columns *OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, and StreamingMovies* were converted to “No”. This is shown in the below code:

```
    churn_df = churn_df.withColumn('TotalCharges',
churn_df['TotalCharges'].cast('double'))
churn_df = churn_df.na.drop()
churn_df_enhanced = churn_df.withColumn('OnlineSecurity',
regexp_replace('OnlineSecurity', 'No internet service', 'No'))
churn_df_enhanced = churn_df.withColumn('MultipleLines',
regexp_replace('MultipleLines', 'No phone service', 'No'))
churn_df_enhanced = churn_df_enhanced.withColumn('OnlineBackup',
regexp_replace('OnlineBackup', 'No internet service', 'No'))
churn_df_enhanced = churn_df_enhanced.withColumn('DeviceProtection',
regexp_replace('DeviceProtection', 'No internet service', 'No'))
churn_df_enhanced = churn_df_enhanced.withColumn('TechSupport',
regexp_replace('TechSupport', 'No internet service', 'No'))
churn_df_enhanced = churn_df_enhanced.withColumn('StreamingTV',
regexp_replace('StreamingTV', 'No internet service', 'No'))
churn_df_enhanced = churn_df_enhanced.withColumn('StreamingMovies',
regexp_replace('StreamingMovies', 'No internet service', 'No'))
```

The *CustomerID* column was then dropped because it is information that is superfluous to our needs here. Next, all columns that consisted of categorical data were indexed to convert them to numerical form. The useful columns were then assembled into a single vector using the *VectorAssembler* module. For this dataset, all the independent features were assembled into a single vector and stored in a new column called *IndependentFeatures.* The label or outcome of the dataset, which is Churn, was also indexed, and saved as *Churn_indexed.* These two columns (*IndependentFeatures* and *Churn_indexed*) were then saved in a new dataframe.

The data then underwent a train-test split. A randomSplit in the ratio of 7:3 was used; 70% of the data went to training the models, and 30% was for testing.

```
#dropping the customerID column
churn_df_encoded = churn_df_enhanced.drop('customerID')
#indexing categorical data
indexer = StringIndexer(inputCols=['gender', 'Partner', 'Dependents',
'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',
'Churn'],
outputCols=['gender_indexed', 'Partner_indexed',
'Dependents_indexed', 'PhoneService_indexed',
'MultipleLines_indexed', 'InternetService_indexed',
'OnlineSecurity_indexed',  'OnlineBackup_indexed',
'DeviceProtection_indexed', 'TechSupport_indexed',
'StreamingTV_indexed', 'StreamingMovies_indexed', 'Contract_indexed',
      'PaperlessBilling_indexed', 'PaymentMethod_indexed',
'Churn_indexed'])

Churn_df_encoded =
indexer.fit(churn_df_encoded).transform(churn_df_encoded)

#using the vector assembler
assembler = VectorAssembler(inputCols=['SeniorCitizen', 'tenure',
'MonthlyCharges', 'TotalCharges', 'PhoneService_indexed',
'StreamingTV_indexed', 'Dependents_indexed',
'InternetService_indexed', 'MultipleLines_indexed',
'TechSupport_indexed', 'Contract_indexed',
'DeviceProtection_indexed','StreamingMovies_indexed',
'OnlineBackup_indexed', 'PaymentMethod_indexed', 'gender_indexed',
      'PaperlessBilling_indexed', 'OnlineSecurity_indexed',
'Partner_indexed'],
outputCol='IndependentFeatures')
```

```
churn_output = assembler.transform(churn_df_encoded)

working_df = churn_output.select('IndependentFeatures',
'Churn_indexed')

(train, test) = working_df.randomSplit([0.7, 0.3])
```

4.2 Decision Tree Model

The Decision Tree is a supervised learning machine learning algorithm that can be used both for classification and regression problems. It works, as the name suggests, creating a node for each decision (or test case) and following the node to the eventual outcome or classification. This last node is called the leaf or terminal node.

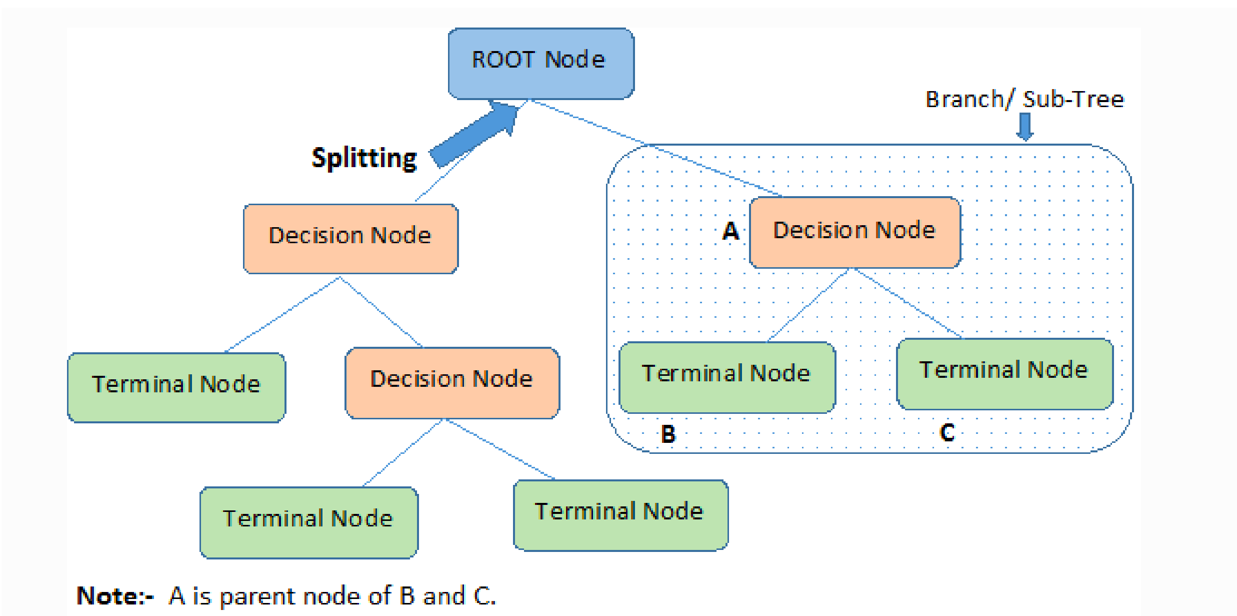


Figure 16. A high-level diagram of the Decision Tree Algorithm (Chauhan, n.d.)

The *DecisionTreeClassifier* module was used in implementing the Decision Tree. First, the model was trained using the training data part of the dataset, and then the prediction of the model was tested on the testing part of the data.

```
dtc = DecisionTreeClassifier(featuresCol='IndependentFeatures',
labelCol="Churn_indexed")
```

```
dtc = dtc.fit(train)
predict = dtc.transform(test)
```

The accuracy of the model was calculated at 0.7863571766935101 or approximately 78.64%.

```
from sklearn.metrics import confusion_matrix
evaluator =
MulticlassClassificationEvaluator(labelCol="Churn_indexed",
predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predict)
print('The accuracy of the Decision Tree Model is: ', accuracy)
y_pred = predict.select('prediction').collect()
y_orig = predict.select('Churn_indexed').collect()

conf_mat = confusion_matrix(y_orig, y_pred)
print('Confusion Matrix: ')
print(conf_mat)
```

4.3 Random Forest Classifier

The Random Forest Classifier is another supervised learning algorithm that is closely related to the Decision Tree. It simply builds several trees and takes a majority vote (hence the name; Random Forest) and is generally a more accurate machine learning model. The model can be used for both classification and regression problems as well. Although it takes the average, not the majority, when used in regression problems. (Sruthi, 2021)

The Random Forest Classifier is part of a class of algorithms that use a technique called the ensemble technique. This technique works by using multiple models to make predictions instead of just one. The Random Forest algorithm uses an ensemble technique called bagging or Bootstrap Aggregation. (Sruthi, 2021).

In this report, the Random Forest algorithm was used courtesy of the *RandomForestClassifier* of the *pyspark.ml* package. The training of the model is achieved by passing it the training set of the data, the predictions are then tested on the test set. Some performance metrics are then run on the model such as the accuracy of the model and the test error.

```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="Churn_indexed",
featuresCol="IndependentFeatures", numTrees=20)
model = rf.fit(train)
predictions = model.transform(test)

rf_evaluator =
MulticlassClassificationEvaluator(labelCol='Churn_indexed',
predictionCol='prediction', metricName='accuracy')
rf_accuracy = evaluator.evaluate(predictions)
print('Accuracy of the Random Forest Classifier Model is: ',
rf_accuracy)
print('Test error of the Random Forest Classifier Model is: ', 1.0 -
rf_accuracy)
```

The accuracy of the model was calculated as 0.7948839412600663 and the test error as 0.20511605873993366

4.4 Logistic Regression

The final algorithm used in the logistic regression algorithm. It is a supervised learning algorithm that works well for classification problems. The version of the algorithm used for this report is called binary or binomial regression. Here, the target or dependent variable can only be one of two values: 0 or 1, with 1 indicating a favourable outcome.

The algorithm was imported from the *pyspark.ml.classification* package. The model was trained on the training set of the data and tested on the testing set. The accuracy of the model is then evaluated.

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol='IndependentFeatures',
labelCol='Churn_indexed')
lr_model = lr.fit(train)
lr_predictions = lr_model.transform(test)
from pyspark.ml.evaluation import BinaryClassificationEvaluator
lr_eval = BinaryClassificationEvaluator(labelCol='Churn_indexed',
rawPredictionCol='prediction')
lr_predictions.select('Churn_indexed', 'prediction')
```

```
lr_accuracy = lr_eval.evaluate(lr_predictions)
print('Logistic Regression Model accuracy is: ', lr_accuracy)
```

The accuracy of the model was evaluated as 0.7186616492071075

4.5 Sub Conclusions

Of the 3 algorithms used, the Random Forest Classifier had the best accuracy. With hyperparameter tuning, this accuracy could have been better. The logistic regression algorithm had the poorest performance at approximately 71%.
All the variables were used in the logistic regression model. Logistic regression performs better when the variables included are meaningful. Some of the variables used here such as the gender variable could have been left out to generate better performance. However, all the algorithms were executed without any tuning.

5. Discussion of Results

After exploring the data and running the models on the data, the following has been observed.

1. Predicting customer churn can be incredibly important to companies as gaining new customers is much more expensive than holding on to existing ones
2. Predicting customer churn is fundamentally a classification problem. Traditional regression algorithms do not perform as well as traditional classification algorithms.
3. More rows of data could have led to better predictions as this will mean the machine learning models will have more data to train on.
4. Newer customers are uncertain about staying with the company. The longer the customer stayed with the company, the less they were likely to churn
5. Newer customers are more likely to churn. This could be because they are more uncertain about the services the company offers.
6. Customers with higher total bills were less likely to churn. This could be because they are more likely to have been with the company longer.

7. Customers who paid via electronic check were significantly more likely to churn.

8. The company's Fibre optic customers were far more likely to churn than DSL customers and customers with no internet.

9. The vast majority of the company's customers are not senior citizens.

10. The company's tech support is doing an admirable job as customers with tech support churned significantly less.

11. Gender has almost no bearing on whether a customer will churn or not.

6. Limitations and Advantages of Big Data for this Report

For this report, using Big Data Programs was not necessary even though the variability in the data was high. The Big Data programs were resource intensive. This means that for smaller projects like this, it does not make sense to allocate the resources that Big Data Programs need to it.

However, the advantages of Big Data would have been obvious if the dataset was large, and variable enough. In that instance, running Spark (via pyspark) on a laptop machine would not make sense. Big Data programs work in clusters so that they can parallelise the job to be done. This makes the work faster and promotes safety of data (via redundancy) and speed (as many nodes are simultaneously working on some part of the data)

7. Conclusion and Recommendations

1. Hyperparameter tuning of the algorithms could have led to better predictions as well. However, that was difficult to achieve because of time and knowledge constraints.

2. Additional information, such as the reason for churn, about the customers could have led to better predictions.

3. The telecommunication company in the dataset needs to focus more efforts on its newer customers as they are uncertain about staying, it was noticed that the highest churn percentage was within the first five months of using the service.

4. Tech support should be commended as customers with tech support were significantly less likely to churn

5. The company's electronic check payment system might be too complex for some customers. Customers with this payment method churned far more than customers in the other groups.

6. Further study can be done with more data and other algorithms. Algorithms such as the XGBoost algorithm might get better results in a shorter time.

References

Anaconda Inc. (n.d.). *Anaconda Docs*. Retrieved February 11, 2022, from https://docs.anaconda.com/anaconda/

Apache Spark. (n.d.). *MLlib: Main Guide - Spark 3.2.1 Documentation*. Apache Spark. Retrieved February 23, 2022, from https://spark.apache.org/docs/latest/ml-guide.html

Blastchar. (2018, February 23). *Telco Customer Churn*. Kaggle. Retrieved February 10, 2022, from https://www.kaggle.com/blastchar/telco-customer-churn

Blismos Academy. (2021, January 29). *How to Install Pyspark on Ubuntu with Java, Spark, and Python*. Youtube.com. Retrieved February 10, 2022, from https://www.youtube.com/watch?v=iK8IR4pPpNg

Chauhan, N. S. (n.d.). *Decision Tree Algorithm, Explained*. KDnuggets. Retrieved February 23, 2022, from https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

Data TechNotes. (2021, June 29). *PySpark Decision Tree Classification Example*. DataTechNotes. Retrieved February 23, 2022, from https://www.datatechnotes.com/2021/06/pyspark-decision-tree-classification.html

Garg, S. (2021, May 24). *How to Deal with Categorical Data for Machine Learning*. KDnuggets. Retrieved February 23, 2022, from https://www.kdnuggets.com/2021/05/deal-with-categorical-data-machine-learning.html

IBM Sample Data Team & Macko, S. (2019, July 11). *Telco customer churn (11.1.3+)*. IBM Community. Retrieved February 12, 2022, from https://community.ibm.com/community/user/businessanalytics/blogs/steven-macko/2019/07/11/telco-customer-churn-1113

The Intuitive Vibe. (2020, April 5). *Step by Step Spark Installation with Ubuntu + Python + Jupyter Notebook - From Scratch*. Youtube.com. Retrieved February 11, 2022, from https://www.youtube.com/watch?v=N8-3kCscjrU

Naik, K. (2021, May 17). *Tutorial 8- Pyspark Multiple Linear Regression Implementation In Databricks*. YouTube. Retrieved 02 14, 2022, from https://www.youtube.com/watch?v=QuWbB_cRSKk&t=1355s

Sruthi, E. R. (2021, June 17). *Random Forest | Introduction to Random Forest Algorithm*. Analytics Vidhya. Retrieved February 23, 2022, from https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Tableau Inc. (n.d.). *What Is Tableau?* Tableau. Retrieved February 12, 2022, from https://www.tableau.com/why-tableau/what-is-tableau

Tan Kah Wang, K. (2020, October 29). *Decision Trees & Random Forests in Pyspark*. Python in Plain English. Retrieved February 15, 2022, from

https://python.plainenglish.io/decision-trees-random-forests-in-pyspark-d0754
6e4fa7d

Appendix

Appendix A: Code

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql import types
from pyspark.sql.functions import col,isnan, when, count


spark = SparkSession.builder.appName('ML').getOrCreate()


churn_df = spark.read.csv('WA_Fn-UseC_-Telco-Customer-Churn.csv',
header=True, inferSchema=True)
```

```
churn_df = churn_df.withColumn('TotalCharges',
churn_df['TotalCharges'].cast('double'))
churn_df.count()


from pyspark.sql.functions import *
churn_df = churn_df.na.drop()
churn_df.count()

churn_df_enhanced = churn_df.withColumn('MultipleLines',
regexp_replace('MultipleLines', 'No phone service', 'No'))
churn_df_enhanced = churn_df_enhanced.withColumn('OnlineBackup',
regexp_replace('OnlineBackup', 'No internet service', 'No'))
churn_df_enhanced = churn_df_enhanced.withColumn('DeviceProtection',
regexp_replace('DeviceProtection', 'No internet service', 'No'))
churn_df_enhanced = churn_df_enhanced.withColumn('TechSupport',
regexp_replace('TechSupport', 'No internet service', 'No'))
```

```
churn_df_enhanced = churn_df_enhanced.withColumn('StreamingTV',
regexp_replace('StreamingTV', 'No internet service', 'No'))
churn_df_enhanced = churn_df_enhanced.withColumn('StreamingMovies',
regexp_replace('StreamingMovies', 'No internet service', 'No'))
```

```
from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import VectorAssembler

from pyspark.ml.feature import StringIndexer


churn_df_encoded = churn_df_enhanced.drop('customerID')


indexer = StringIndexer(inputCols=['gender', 'Partner', 'Dependents',
'PhoneService', 'MultipleLines', 'InternetService',
                                  'OnlineSecurity', 'OnlineBackup',
'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
'Contract',
                                  'PaperlessBilling',
'PaymentMethod', 'Churn'], outputCols=['gender_indexed',
'Partner_indexed', 'Dependents_indexed','PhoneService_indexed',

'MultipleLines_indexed', 'InternetService_indexed',
'OnlineSecurity_indexed',

'OnlineBackup_indexed', 'DeviceProtection_indexed',
'TechSupport_indexed',

'StreamingTV_indexed', 'StreamingMovies_indexed', 'Contract_indexed',
```

```
'PaperlessBilling_indexed', 'PaymentMethod_indexed',
'Churn_indexed'])
churn_df_encoded =
indexer.fit(churn_df_encoded).transform(churn_df_encoded)
```

```
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=['SeniorCitizen', 'tenure',
'MonthlyCharges', 'TotalCharges', 'PhoneService_indexed',
                                     'StreamingTV_indexed',
'Dependents_indexed', 'InternetService_indexed',
                                     'MultipleLines_indexed',
'TechSupport_indexed', 'Contract_indexed',
'DeviceProtection_indexed',
                                     'StreamingMovies_indexed',
'OnlineBackup_indexed', 'PaymentMethod_indexed', 'gender_indexed',
                                     'PaperlessBilling_indexed',
'OnlineSecurity_indexed', 'Partner_indexed'],
                           outputCol='IndependentFeatures')
churn_output = assembler.transform(churn_df_encoded)
```

```
churn_output.printSchema()


churn_output.select('IndependentFeatures').show()


working_df = churn_output.select('IndependentFeatures',
'Churn_indexed')
working_df.show()
(train, test) = working_df.randomSplit([0.7, 0.3])
```

```
dtc = DecisionTreeClassifier(featuresCol='IndependentFeatures',
labelCol="Churn_indexed")
dtc = dtc.fit(train)

predict = dtc.transform(test)
predict.show(10)


from sklearn.metrics import confusion_matrix
evaluator =
MulticlassClassificationEvaluator(labelCol="Churn_indexed",
predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predict)

print('The accuracy of the Decision Tree Model is: ', accuracy)

y_pred = predict.select('prediction').collect()
y_orig = predict.select('Churn_indexed').collect()

conf_mat = confusion_matrix(y_orig, y_pred)
print('Confusion Matrix: ')
print(conf_mat)
```

```
from pyspark.ml.classification import RandomForestClassifier



rf = RandomForestClassifier(labelCol="Churn_indexed",
featuresCol="IndependentFeatures", numTrees=20, rand)



model = rf.fit(train)

predictions = model.transform(test)
```

```
predictions.show(10)



rf_evaluator =
MulticlassClassificationEvaluator(labelCol='Churn_indexed',
predictionCol='prediction', metricName='accuracy')
rf_accuracy = evaluator.evaluate(predictions)

print('Accuracy of the Random Forest Classifier Model is: ',
rf_accuracy)
print('Test error of the Random Forest Classifier Model is: ', 1.0 -
rf_accuracy)
```

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol='IndependentFeatures',
labelCol='Churn_indexed')


lr_model = lr.fit(train)
lr_predictions = lr_model.transform(test)
lr_predictions.show()


from pyspark.ml.evaluation import BinaryClassificationEvaluator
lr_eval = BinaryClassificationEvaluator(labelCol='Churn_indexed',
rawPredictionCol='prediction')
lr_predictions.select('Churn_indexed', 'prediction')

lr_accuracy = lr_eval.evaluate(lr_predictions)
print('Logistic Regression Model accuracy is: ', lr_accuracy)
```

Appendix B: Figures and Tables

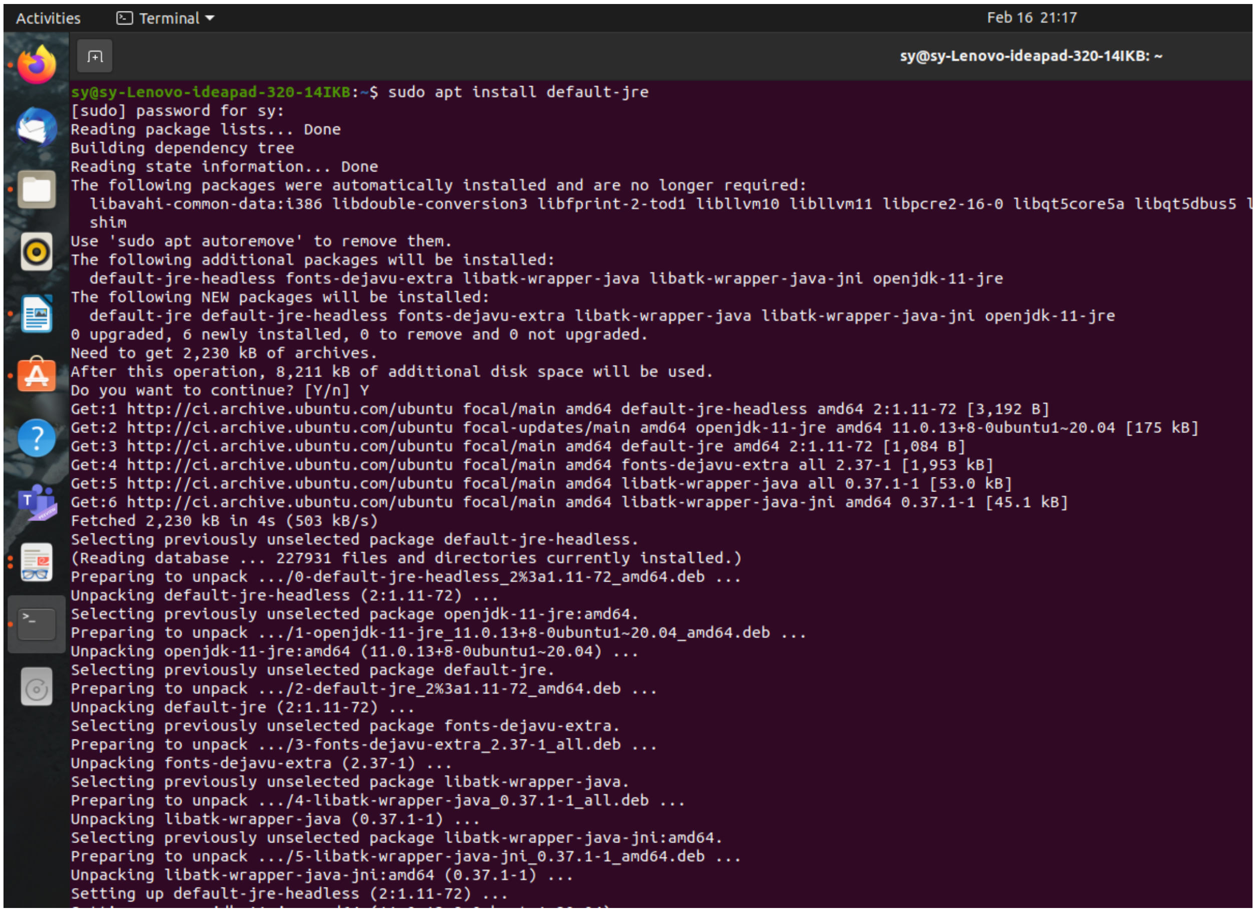
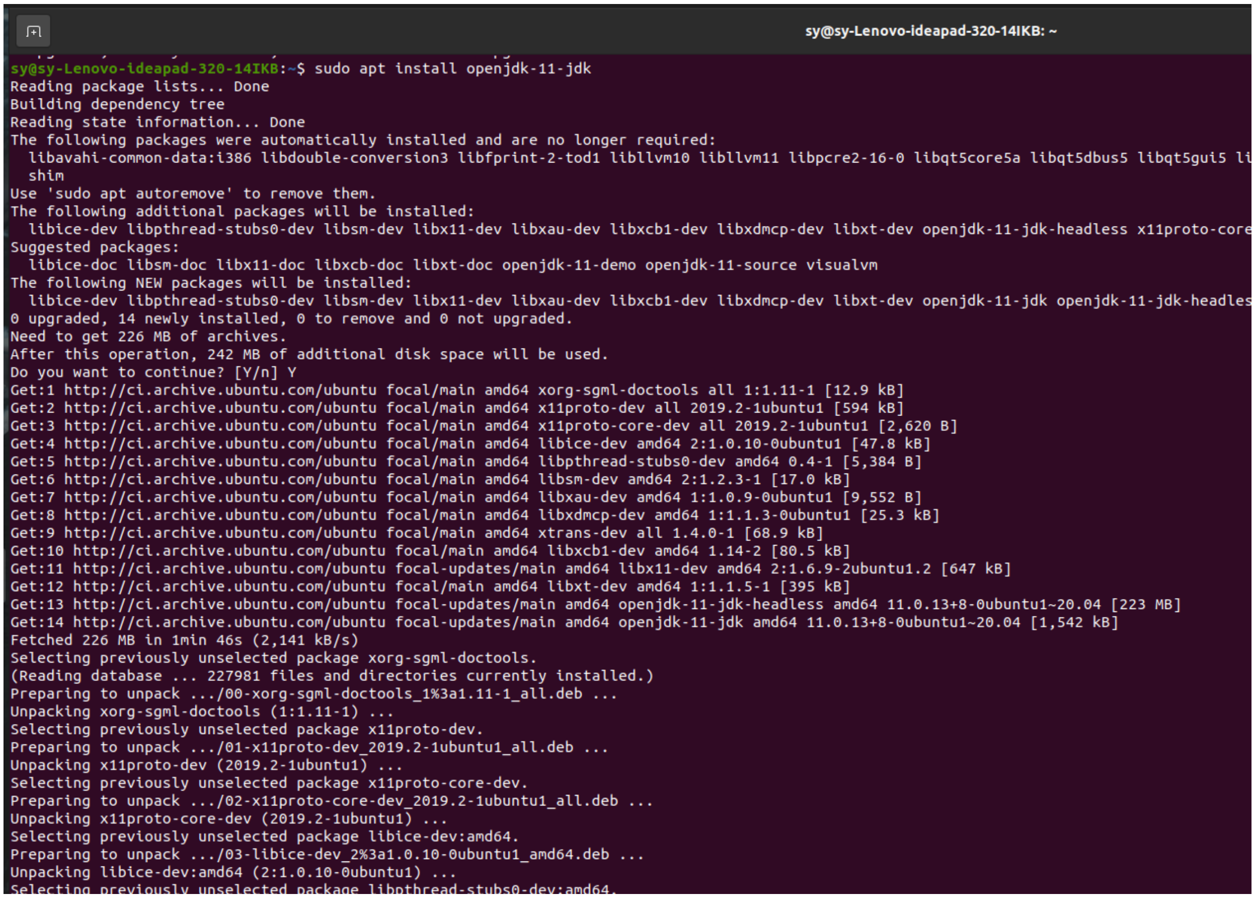


Figure 1. Installing the JRE from the terminal



```
sy@sy-Lenovo-ideapad-320-14IKB: ~
sy@sy-Lenovo-ideapad-320-14IKB:~$ sudo apt install openjdk-11-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libavahi-common-data:i386 libdouble-conversion3 libfprint-2-tod1 libllvm10 libllvm11 libpcre2-16-0 libqt5core5a libqt5dbus5 libqt5gui5 li
  shim
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libice-dev libpthread-stubs0-dev libsm-dev libx11-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-11-jdk-headless x11proto-core
Suggested packages:
  libice-doc libsm-doc libx11-doc libxcb-doc libxt-doc openjdk-11-demo openjdk-11-source visualvm
The following NEW packages will be installed:
  libice-dev libpthread-stubs0-dev libsm-dev libx11-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-11-jdk openjdk-11-jdk-headles
0 upgraded, 14 newly installed, 0 to remove and 0 not upgraded.
Need to get 226 MB of archives.
After this operation, 242 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://ci.archive.ubuntu.com/ubuntu focal/main amd64 xorg-sgml-doctools all 1:1.11-1 [12.9 kB]
Get:2 http://ci.archive.ubuntu.com/ubuntu focal/main amd64 x11proto-dev all 2019.2-1ubuntu1 [594 kB]
Get:3 http://ci.archive.ubuntu.com/ubuntu focal/main amd64 x11proto-core-dev all 2019.2-1ubuntu1 [2,620 B]
Get:4 http://ci.archive.ubuntu.com/ubuntu focal/main amd64 libice-dev amd64 2:1.0.10-0ubuntu1 [47.8 kB]
Get:5 http://ci.archive.ubuntu.com/ubuntu focal/main amd64 libpthread-stubs0-dev amd64 0.4-1 [5,384 B]
Get:6 http://ci.archive.ubuntu.com/ubuntu focal/main amd64 libsm-dev amd64 2:1.2.3-1 [17.0 kB]
Get:7 http://ci.archive.ubuntu.com/ubuntu focal/main amd64 libxau-dev amd64 1:1.0.9-0ubuntu1 [9,552 B]
Get:8 http://ci.archive.ubuntu.com/ubuntu focal/main amd64 libxdmcp-dev amd64 1:1.1.3-0ubuntu1 [25.3 kB]
Get:9 http://ci.archive.ubuntu.com/ubuntu focal/main amd64 xtrans-dev all 1.4.0-1 [68.9 kB]
Get:10 http://ci.archive.ubuntu.com/ubuntu focal/main amd64 libxcb1-dev amd64 1.14-2 [80.5 kB]
Get:11 http://ci.archive.ubuntu.com/ubuntu focal-updates/main amd64 libx11-dev amd64 2:1.6.9-2ubuntu1.2 [647 kB]
Get:12 http://ci.archive.ubuntu.com/ubuntu focal/main amd64 libxt-dev amd64 1:1.1.5-1 [395 kB]
Get:13 http://ci.archive.ubuntu.com/ubuntu focal-updates/main amd64 openjdk-11-jdk-headless amd64 11.0.13+8-0ubuntu1~20.04 [223 MB]
Get:14 http://ci.archive.ubuntu.com/ubuntu focal-updates/main amd64 openjdk-11-jdk amd64 11.0.13+8-0ubuntu1~20.04 [1,542 kB]
Fetched 226 MB in 1min 46s (2,141 kB/s)
Selecting previously unselected package xorg-sgml-doctools.
(Reading database ... 227981 files and directories currently installed.)
Preparing to unpack .../00-xorg-sgml-doctools_1%3a1.11-1_all.deb ...
Unpacking xorg-sgml-doctools (1:1.11-1) ...
Selecting previously unselected package x11proto-dev.
Preparing to unpack .../01-x11proto-dev_2019.2-1ubuntu1_all.deb ...
Unpacking x11proto-dev (2019.2-1ubuntu1) ...
Selecting previously unselected package x11proto-core-dev.
Preparing to unpack .../02-x11proto-core-dev_2019.2-1ubuntu1_all.deb ...
Unpacking x11proto-core-dev (2019.2-1ubuntu1) ...
Selecting previously unselected package libice-dev:amd64.
Preparing to unpack .../03-libice-dev_2%3a1.0.10-0ubuntu1_amd64.deb ...
Unpacking libice-dev:amd64 (2:1.0.10-0ubuntu1) ...
Selecting previously unselected package libpthread-stubs0-dev:amd64.
```

Figure 2. Installing the JDK using the terminal

```
sy@sy-Lenovo-ideapad-320-14IKB:~$ sudo su
root@sy-Lenovo-ideapad-320-14IKB:/home/sy# sudo mkdir -p opt/spark
root@sy-Lenovo-ideapad-320-14IKB:/home/sy# cd opt/spark
root@sy-Lenovo-ideapad-320-14IKB:/home/sy/opt/spark# wget https://dlcdn.apache.org/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz
--2022-02-16 21:45:03--  https://dlcdn.apache.org/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz
Resolving dlcdn.apache.org (dlcdn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 228834641 (218M) [application/x-gzip]
Saving to: ‘spark-3.1.2-bin-hadoop3.2.tgz’

spark-3.1.2-bin-hadoop3.2.tgz                100%[==========================================================================

2022-02-16 21:51:52 (549 KB/s) - ‘spark-3.1.2-bin-hadoop3.2.tgz’ saved [228834641/228834641]

root@sy-Lenovo-ideapad-320-14IKB:/home/sy/opt/spark#
root@sy-Lenovo-ideapad-320-14IKB:/home/sy/opt/spark#
```

Figure 3. Downloading Apache Spark

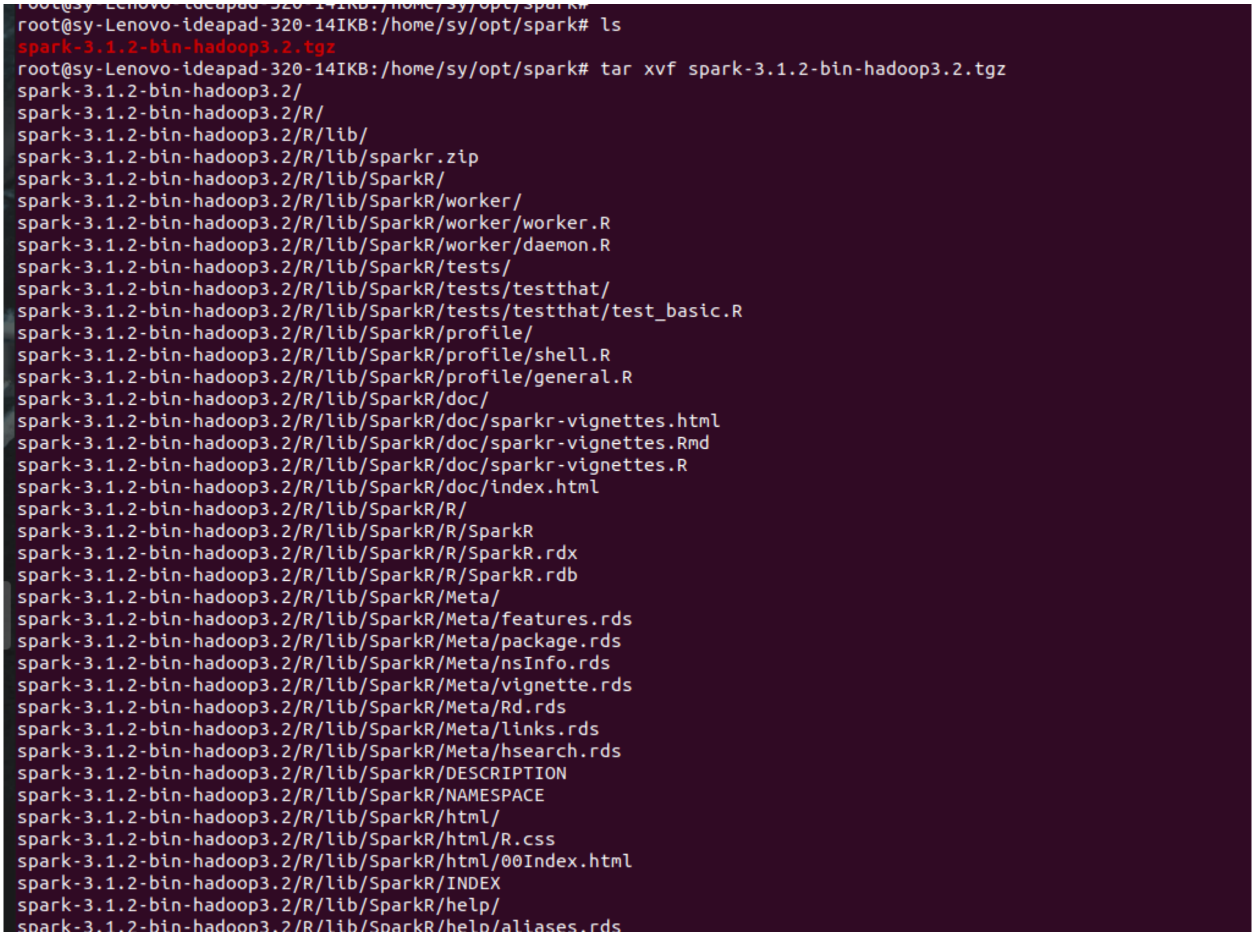


Figure 4. Extracting Apache Spark tar file into the spark directory

Figure 5. Running the Spark shell

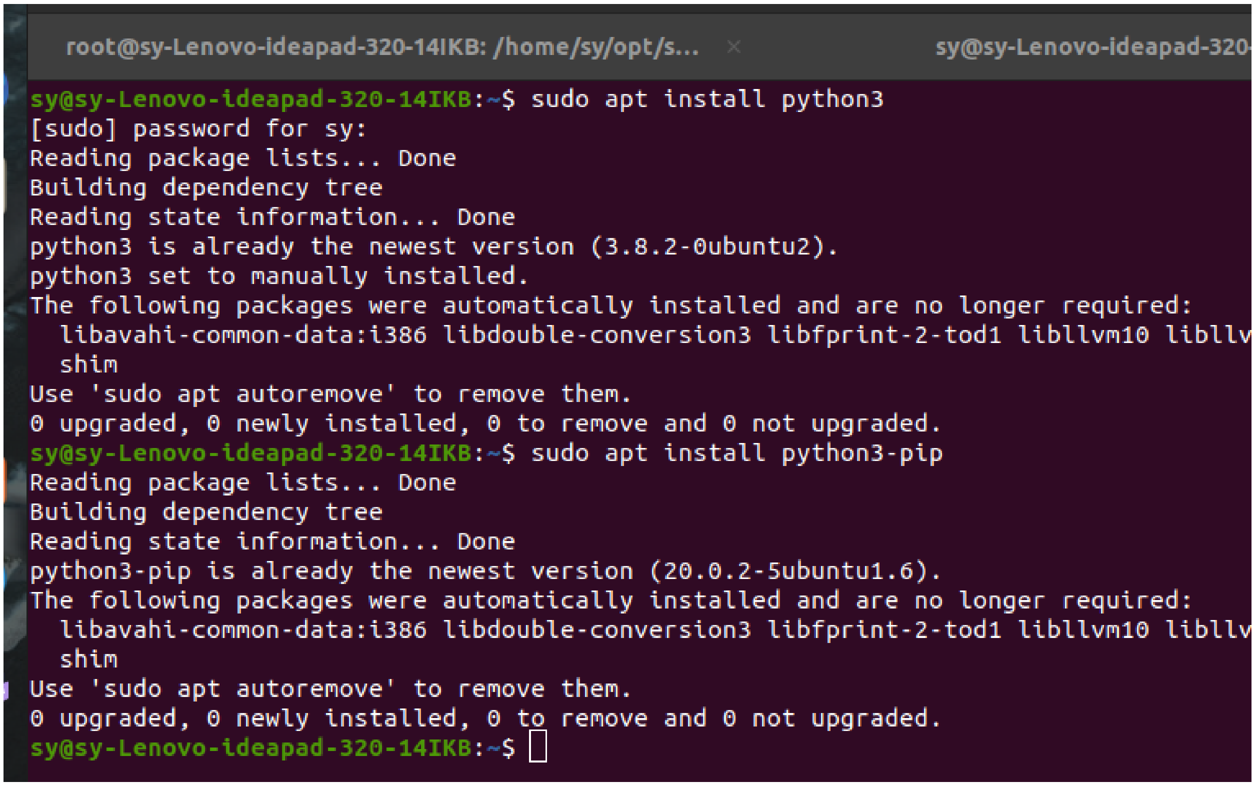


Figure 6. Installing Python and pip

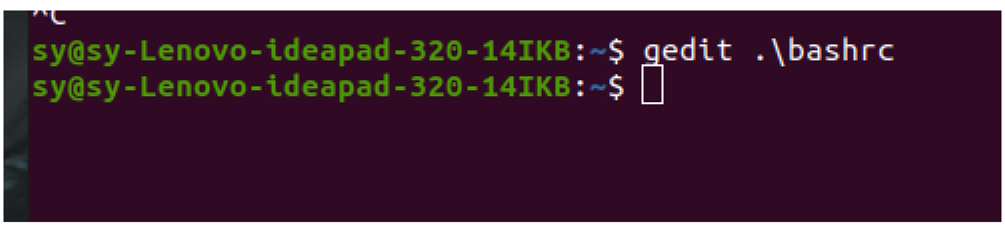
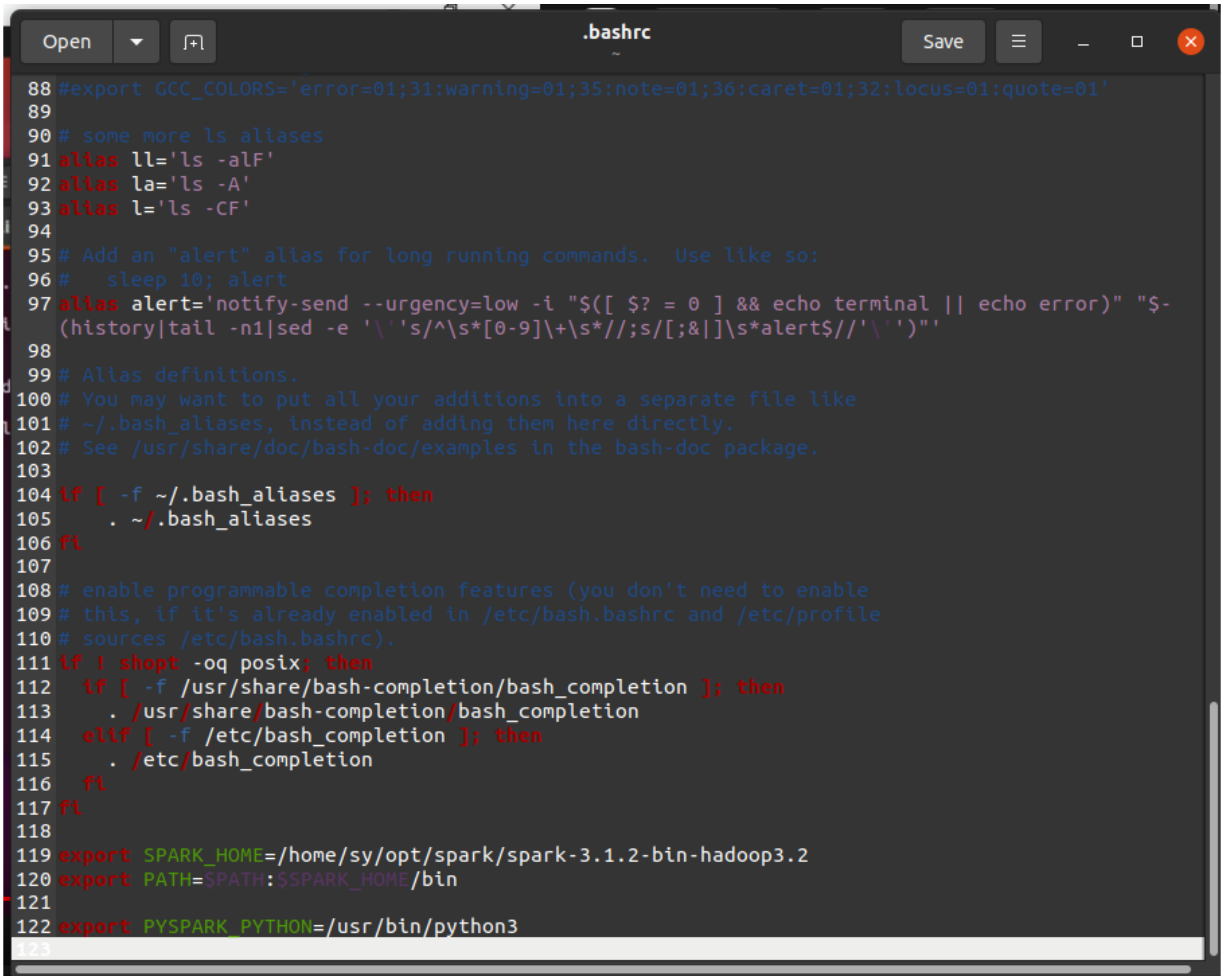


Figure 7. Editing the .bashrc file



```
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

export SPARK_HOME=/home/sy/opt/spark/spark-3.1.2-bin-hadoop3.2
export PATH=$PATH:$SPARK_HOME/bin

export PYSPARK_PYTHON=/usr/bin/python3
```

Figure 8. Adding the relevant environmental variables to PATH

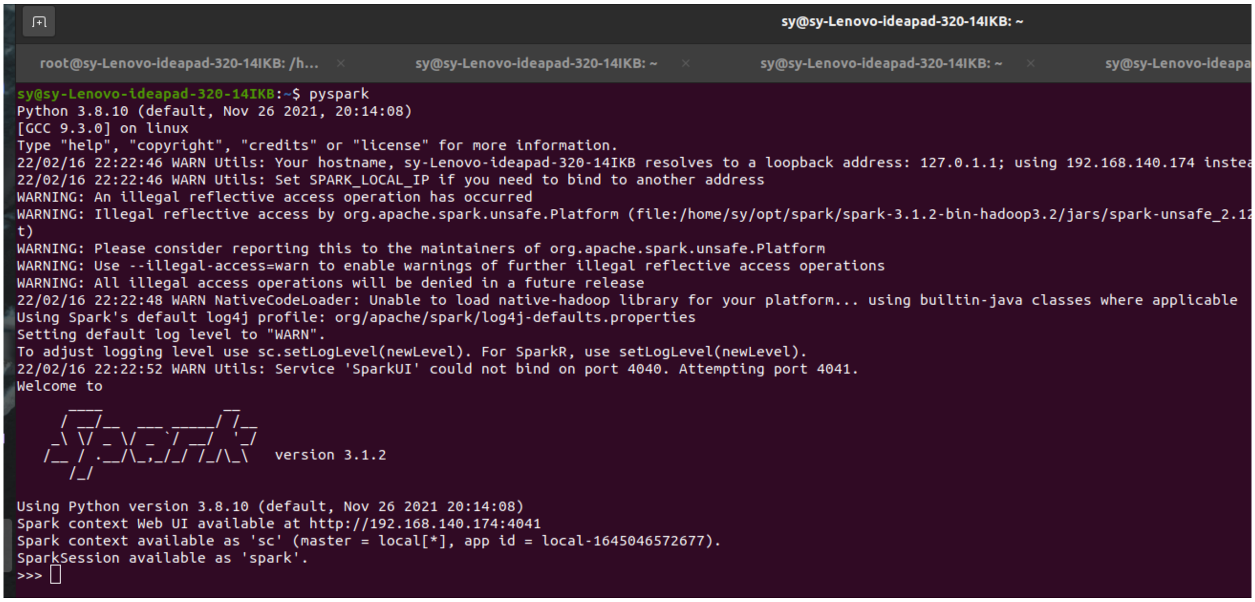


Figure 9. Running Pyspark from the terminal

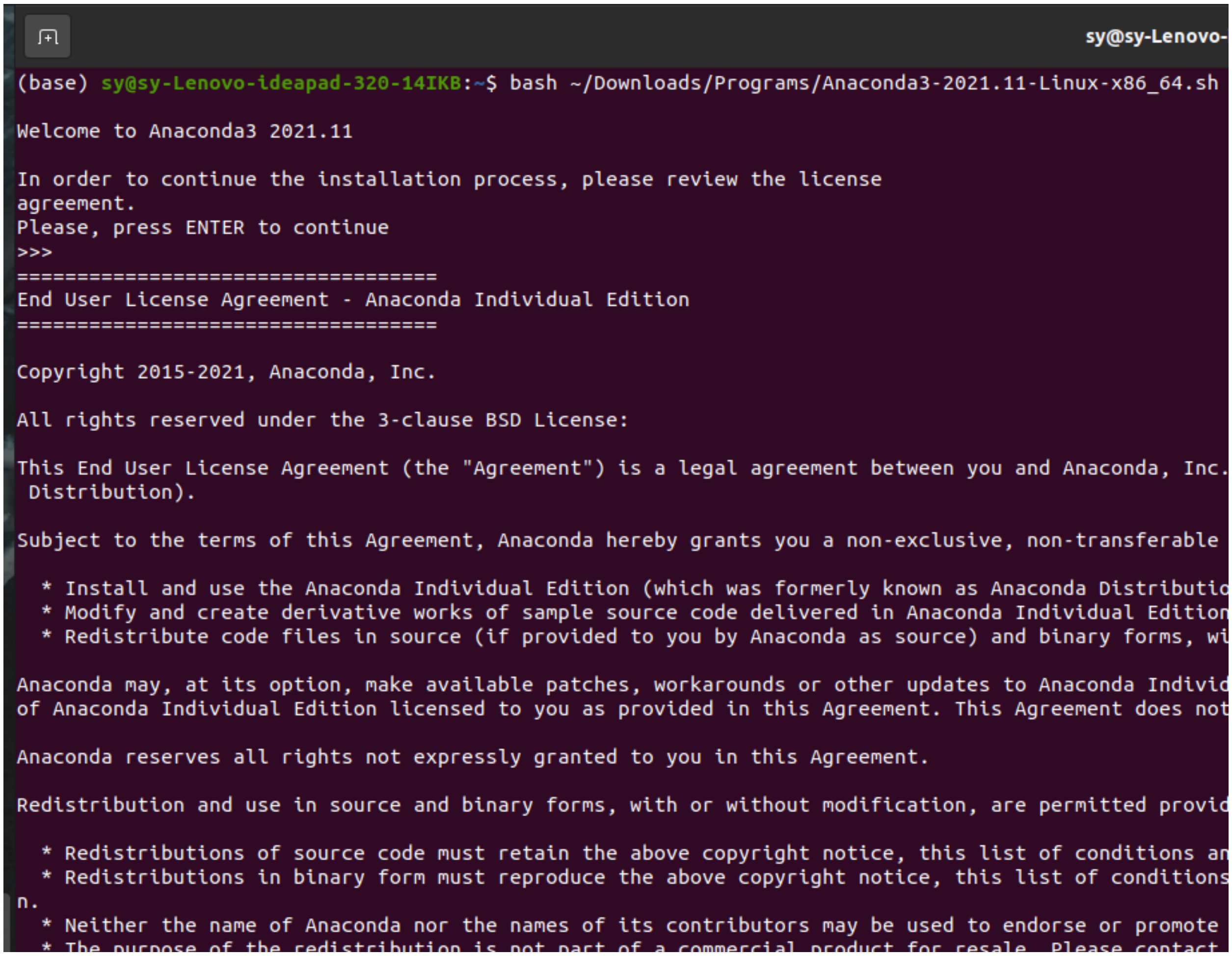


Figure 10. Installation of Anaconda

```bash
# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

export SPARK_HOME=/home/sy/opt/spark/spark-3.1.2-bin-hadoop3.2
export PATH=$PATH:$SPARK_HOME/bin

export PYTHONPATH==$SPARK_HOME/python:$PYTHONPATH
export PYSPARK_DRIVER_PYTHON="jupyter"
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
export PYSPARK_PYTHON=/usr/bin/python3


# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/home/sy/anaconda3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
```

Figure 11. Configuring Jupyter Notebook for Pyspark

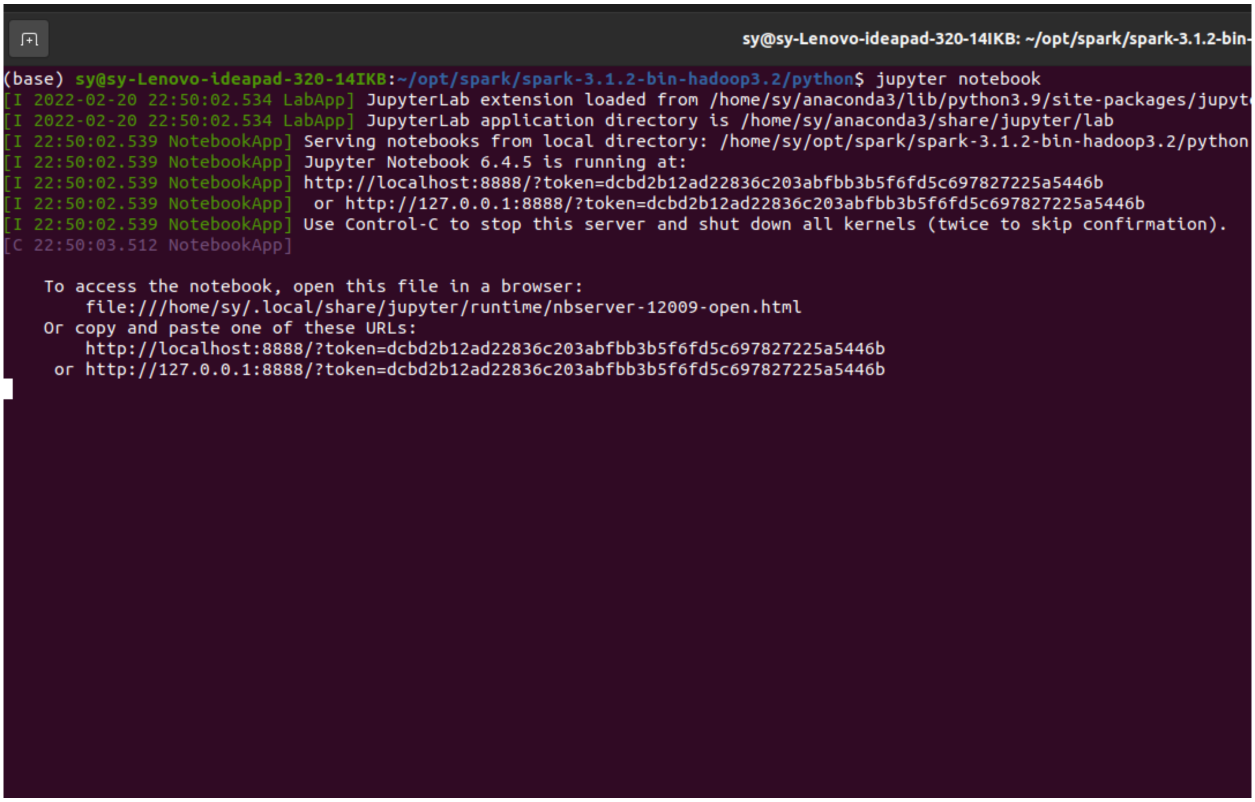


Figure 12. Starting Jupyter Notebook

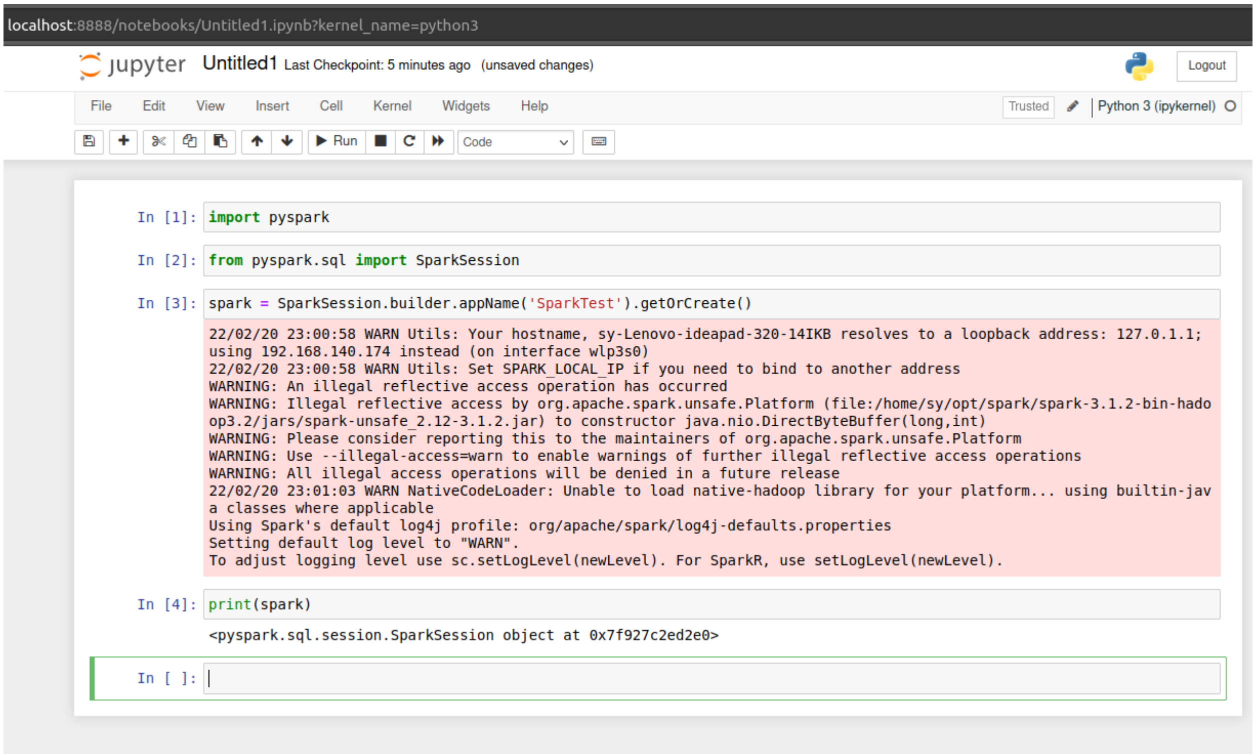


Figure 13. Creating a Spark Session in Jupyter Notebook

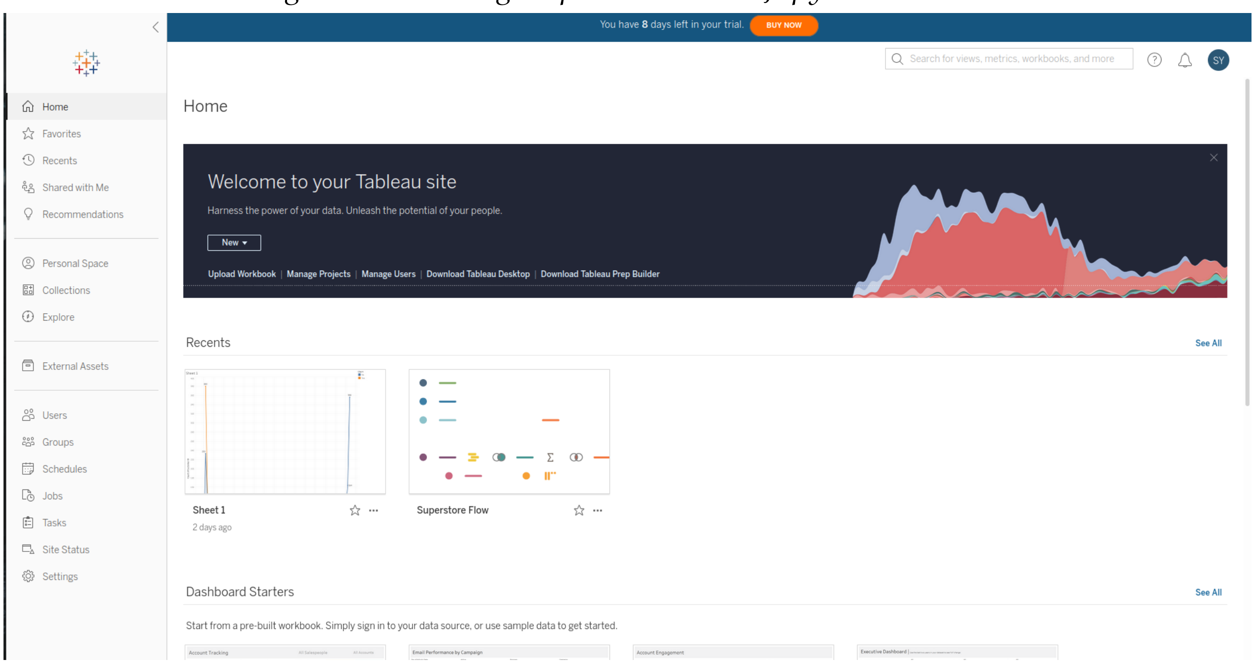


Figure 12. Tableau Online after login

```
In [12]: churn_df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in churn_df.columns]).show()

+----------+------+-------------+-------+----------+------+------------+-------------+---------------+-------------
+------------+----------------+-----------+-----------+---------------+--------+----------------+-------------+---
---------+------------+-----+
|customerID|gender|SeniorCitizen|Partner|Dependents|tenure|PhoneService|MultipleLines|InternetService|OnlineSecurit
y|OnlineBackup|DeviceProtection|TechSupport|StreamingTV|StreamingMovies|Contract|PaperlessBilling|PaymentMethod|Mon
hlyCharges|TotalCharges|Churn|
+----------+------+-------------+-------+----------+------+------------+-------------+---------------+-------------
+------------+----------------+-----------+-----------+---------------+--------+----------------+-------------+---
---------+------------+-----+
|         0|     0|            0|      0|         0|     0|           0|            0|              0|
0|           0|               0|          0|          0|              0|       0|               0|            0|
0|          11|    0|
+----------+------+-------------+-------+----------+------+------------+-------------+---------------+-------------
+------------+----------------+-----------+-----------+---------------+--------+----------------+-------------+---
---------+------------+-----+
```

```
In [13]: ## Drop all null values in the dataset.
         churn_df = churn_df.na.drop()
         churn_df.count()

Out[13]: 7032

In [ ]:
```

Figure 13. Checking and dropping null values in the dataset

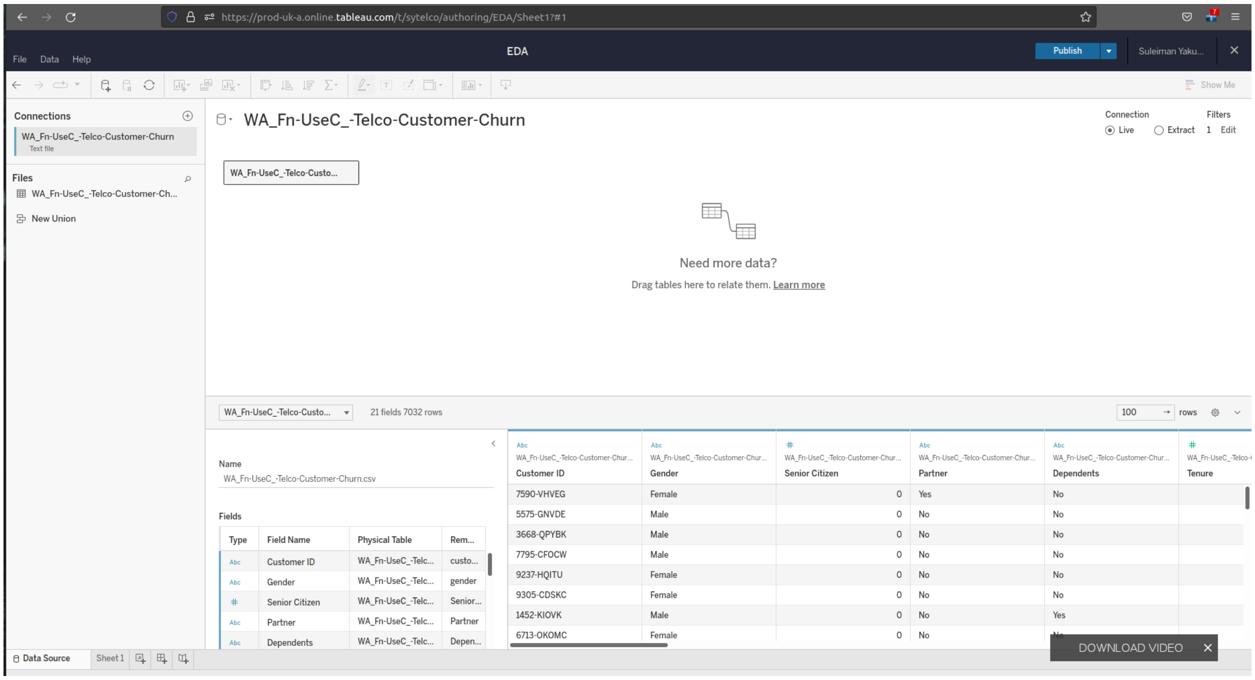


Figure 14. Importing data into Tableau

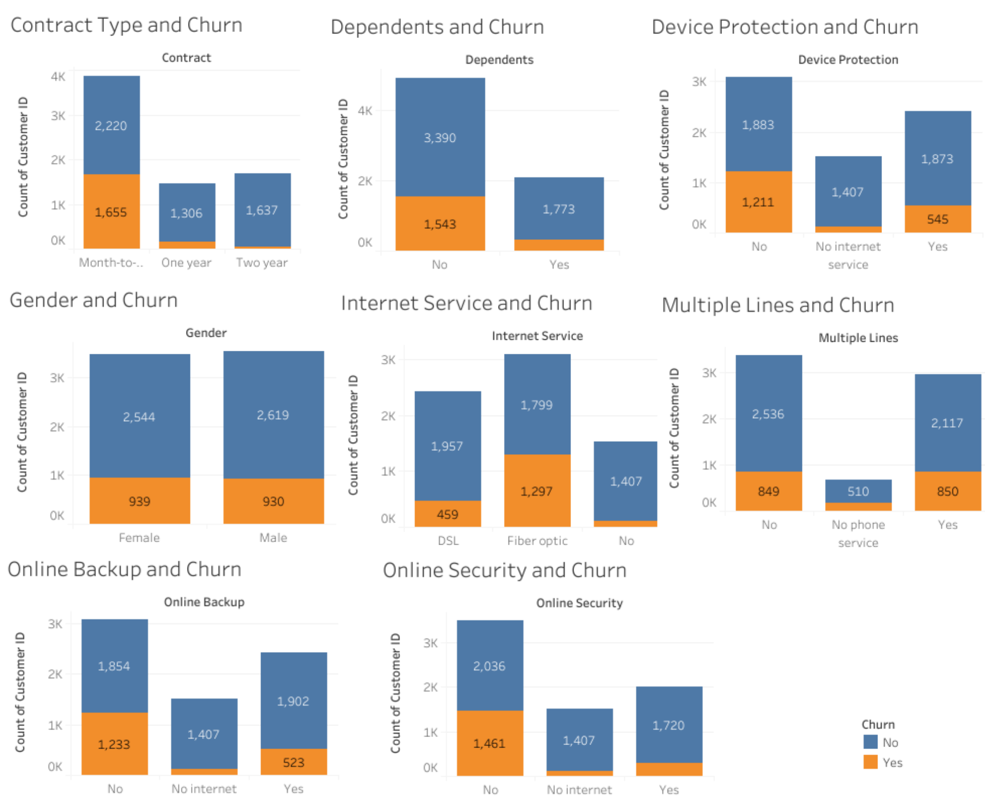


Figure 15. Some categorical attributes and their relationship with churn

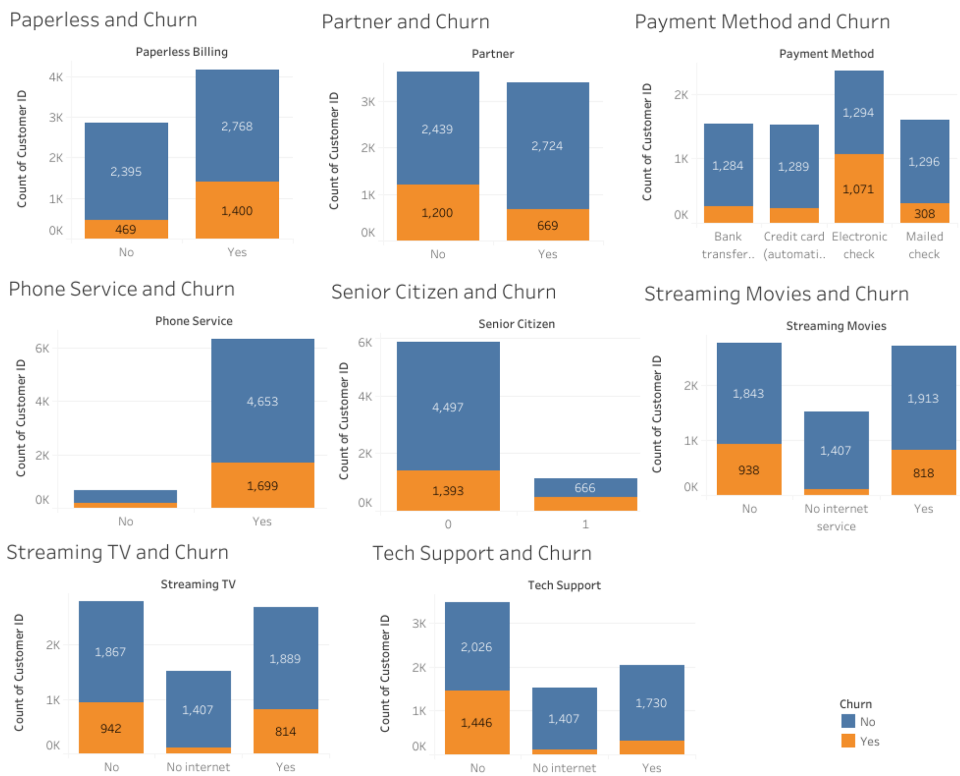


Figure 14. Some categorical attributes plotted against churn

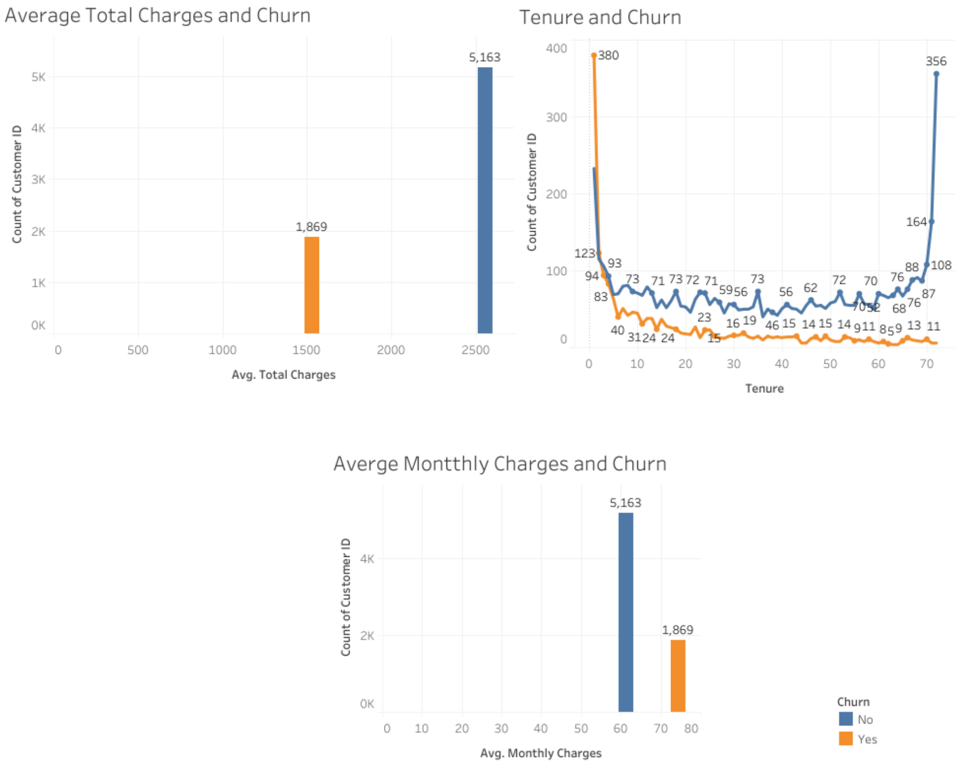


Figure 15. Some continuous attributes of against Churn

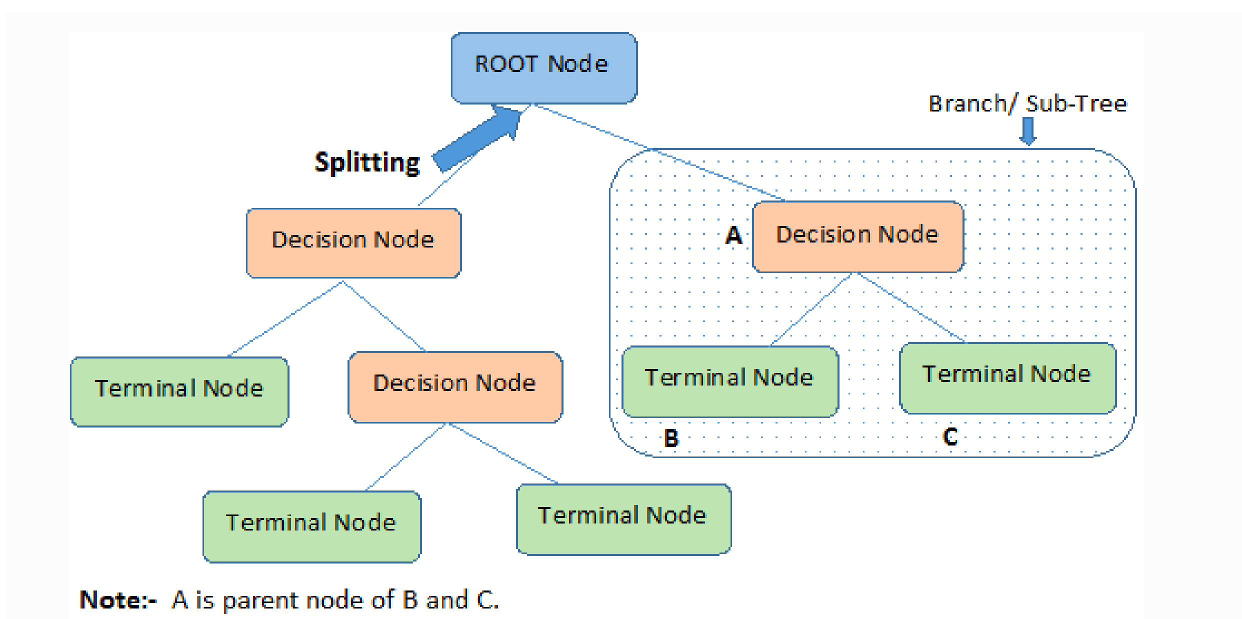


Figure 16. A high level diagram of the Decision Tree Algorithm (Chauhan, n.d.)

Attribute	Data Type	Description
CustomerID	String	A unique string that identifies the customer
Gender	String (Male or Female)	The Gender of the customer
SeniorCitizen	String (Yes or No)	Indicates if the customer is a Senior Citizen or not
Partner	String (Yes or No)	Indicates if the customer has a significant other
Dependents	String (Yes or No)	Indicates if the customer has dependents
Tenure	Integer	Indicates how many months the customer used the Telco's services
PhoneService	String (Yes or No)	Indicates if the customer has home phone service
MultipleLines	String (Yes or No, No Phone Service)	Indicates if the customer has multiple telephone line subscriptions
InternetService	String (DSL, Fibre Optic, No)	Indicates if the customer subscribed for internet with the telco
OnlineSecurity	String (Yes, No, No Internet Service)	If the customer paid for online security
OnlineBackup	String (Yes, No, No Internet Service)	If the customer paid for an online backup service
DeviceProtection	String (Yes, No, No Internet Service)	If the customer subscribed to a device protection plan
TechSupport	String (Yes, No, No	If the customer subscribed

	Internet Service)	for technical support
StreamingTV	String (Yes, No, No Internet Service)	Does the customer stream television programs from a 3rd party?
StreamingMovies	String (Yes, No, No Internet Service)	Does the customer stream movies from a 3rd party?
Contract	String (Month-to Month, Yearly, Two Year)	Current contract type
PaperlessBilling	String (Yes or No)	Whether customer has chosen paperless billing
PaymentMethod	String (Bank Transfer, Credit card (automatic), Electronic check, mailed check)	The method of payment
MonthlyCharges	Float	Monthly charges
TotalCharges	Float	Total charges incurred
Churn	String (Yes or No)	Whether the customer churned or not

Table 1. Data dictionary for the Telco Dataset ((IBM Sample Data Team & Macko, 2019)*)*