**Team 2’s Analysis of the E-Commerce Shipping Data**

**Introduction**

Customer loyalty and satisfaction are thought to be two of the most important metrics that companies can be judged by. Generating return business through increasing brand value to a customer is a strategic goal that most companies share. To create loyalty and satisfaction, a company must look at their operations from all angles to find what key performance metrics drive this repeat business. While products and companies can be judged for multiple criteria, such as quality, performance, or reliability, we decided to focus on a factor that typically affects a customer’s first impression: on-time delivery.

The dataset used for this assignment is from an international E-Commerce company. We plan to use this dataset in hopes of learning valuable information about their customers; along with figuring out how customer support plays a role. Many factors affect how shipments are prioritized to customers. Such factors accounted for in this dataset include mode of transportation, shipment point, and customer care calls. The key target variable in this dataset is on-time delivery which will inform the analyst whether the product arrived at the customer in the expected period provided.

This set was picked because we know the impact of logistical challenges that companies face and we were seeking to determine whether there were commonalities that drove more efficient delivery speeds. We also believed it would be interesting to research and analyze data about online shopping as this form of consumerism has grown due to both the pandemic and improved technology. We wanted to explore how customers react when buying items at different price points and if their product ratings were affected based on receiving those products promptly.

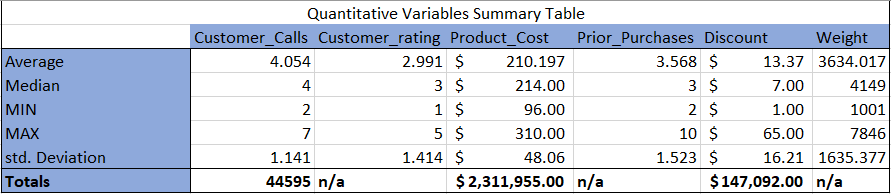
Our first questions when exploring this dataset included finding what factors most affected an “On-time” delivery. Throughout our research we were able to find clues to answer these questions as well as data which created additional queries.

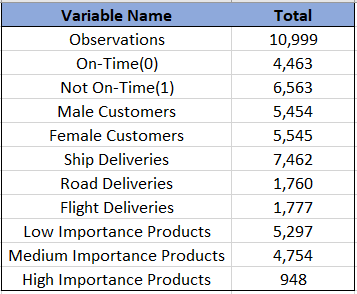
**Data Exploration and Visualization**

First for data exploration, we define the variables in the dataset below:

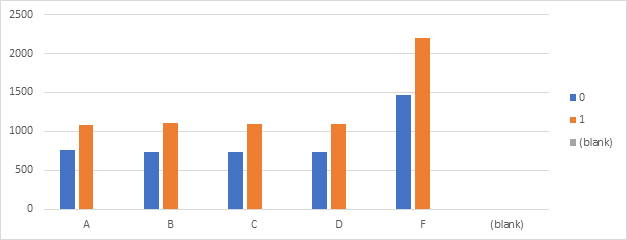
* **ID** – the numeric primary key which has been assigned to each observation in the set and all but useless for predicting.
* **Warehouse\_block** – A categorical string variable that is split into A-E, each being a block of the main warehouse that stores the product before shipment.
* **Mode\_of\_Shipment** – A categorical string variable with 3 types: Ship, Flight, and Road.
* **Customer\_Care\_Calls** – A quantitative numeric variable that is a count of the number of calls made by a customer for the individual observation for an inquiry on the purchase/delivery of the product.
* **Customer\_Rating** – A quantitative numeric variable for the customer’s rating of the product purchased for that individual observation.
* **Cost\_of\_the\_Product** – A quantitative numeric variable for the cost of the product in the observation.
* **Prior\_Purchases** – A quantitative numeric variable that details the amount of purchases a customer has made prior to the current purchase. No customer information is given in the dataset, so we cannot link observations together for repeat customers.
* **Product\_Importance** – A categorical string variable set by the company for various parameters undetailed in the set description.
* **Gender** – A categorical string variable detailing the gender of the customer for the individual observation. 0 for female, 1 for male.
* **Discount\_Offered** – A quantitative numeric variable to detail the discount given on the given observation instance.
* **Weight\_in\_gms** – a quantitative numeric variable for the weight of the product in grams.
* **Reached.on.Time\_Y.N.** – The categorical numeric target variable of the dataset to determine if the observation purchase arrived on time to the customer or not. 0 is for on time, 1 is for not on time.

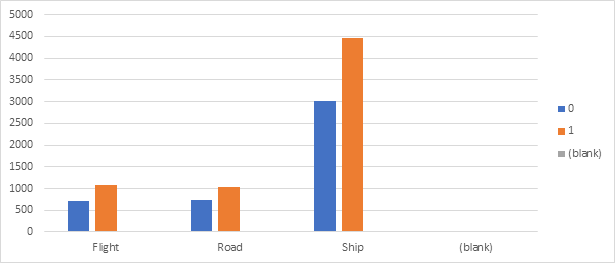
Additionally, to the descriptions of the variables, below is a summary table for the various quantitative variables in the dataset. For the total row, the variables that made sense to total are the total Customer calls made, total amount spent by all customers (total Product\_Cost), and total amount of Discounts offered in the set.

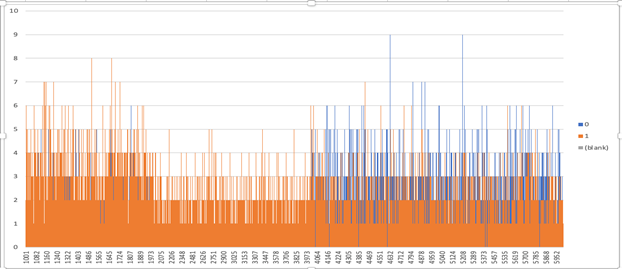


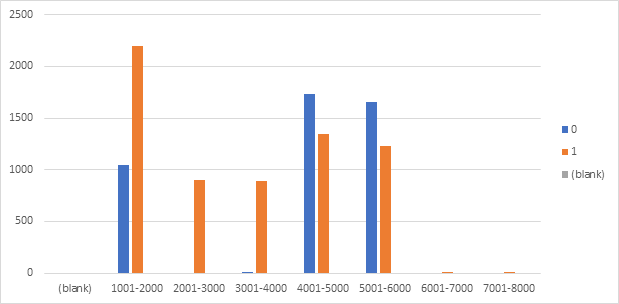
The total values for the categorical variables can be seen in the table on the right. In these totals, Ship Deliveries is the most preferred way to transfer a product at 67.8% of all observations. Road Deliveries and Flight Deliveries only make up 16% and 16.2% respectively. Additionally, High Importance Products make up only 8.6%, while Low Importance Products make up 48% and Medium Importance Products make up the remaining 43.4%. 

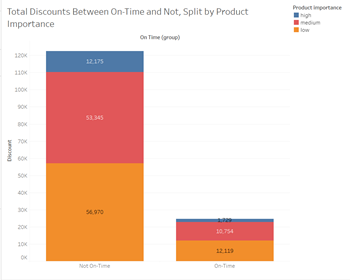
To better visualize the data beyond summarizing it, and to understand how each of the variables affect the delivery time, we created a pivot table to count the number of instances a delivery was made on time. Then with that table data we created bar graphs with the variables that we felt affected the delivery time the most. The variables include Warehouse\_block, Mode\_of\_Shipment, and Weight\_in\_gms. These were chosen before the data mining process.

The first graph on the left depicts the different warehouses and whether the goods were delivered on time from each of them, 0 for on time and 1 for not on time. The greatest number of shipments were made from Warehouse F. The number of shipments not delivered on time were greater than on time shipments regardless of the warehouse they originated from.

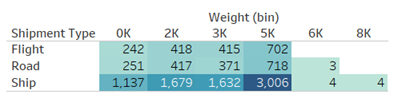
This next bar graph is for the different modes of transport used and whether the goods arrived on time or not. Same as above, transport via Ship is used about double the other options combined.

With Weight being the final important predictor, we found there was an issue with graphing the values seen here. The graph is messy and hard to read, with thousands of data points being displayed all at once.

Our solution to this was to display weight in grouped intervals of 1,000’s to better show weight groups in an easier to consume view. Interestingly, very few products weighing between 2,000-4,000 grams were delivered on time. And, as the weight of the product increased beyond that, the number of on time deliveries increased significantly.

Outside of the 3 predictors we detailed above, we had a few questions to confirm looking through the data. One of these is shown in the graph here on the right. This details the total amount or cost to the company of Discounts offered for either On Time or not. The bars are broken down per Product Importance. This shows that product shipments that were not on time were offered significantly more Discounts totaling to $122,490 at 83% of observations. For On Time totals in at $24,602 with only taking up the remaining 17%. As expected, Discounts were offered more for Late Deliveries, but what we did not expect was that High Importance Products have the smallest share of Discounts offered to them.

Our other question to consider was how are heavier products transported? With Weight and Mode of Shipment key predictors early on, we created a table detailing the relationship below.

As found above, transport via Ship is the most common, and here has the most instances of heavier products. All weights were again put into bins for easier viewability of the table.

A variable to be added could be an arrival value. 0 being on time, <0 being how many days early, and >0 being how many days late. This would have to be dropped in data mining processes due to being able to predict On Time perfectly, but its real value lies in the visualizations and exploration and how it relates to predictors like weight, shipment, etc.

**Preparation for Data Mining (Data Clean Up)**

With all that we found in our data exploration above about important predictors and correlations between each, the next step is to clean and prepare the data for the data mining. We first checked for null/blank’s and 0 values by using Excel and Power BI’s pivot table and slicers, respectively. This dataset has no null or 0 values, which makes our job easier for preparation. If there were null values, potentially we could have gone in to decide whether to drop the observations entirely or to replace them with similar data. Similarly for 0 values, we would have gone in to decide whether the 0 value was for missing data or if it was correct and an outlier and gone on from there to either keep or remove the observation.

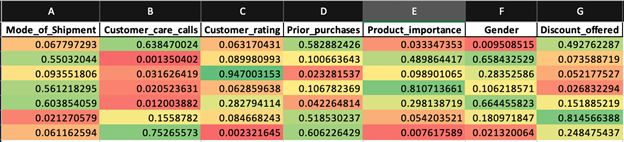
**Data Mining Techniques**

After data clean up and validation, we move on to the data mining techniques. The 4 techniques used on the dataset include Principal Component Analysis (PCA), Decision Tree Model, KNN and for the technique not mentioned in class, Random Forest Analysis. Each of these and the process of doing so will be detailed below in their corresponding section, starting with the PCA process.

**Principal Component Analysis (PCA)**

The PCA technique was done as the first technique as it can be used for later techniques in determining predictors to use. During the first run of the PCA the ID column was dropped. The target variable was ‘Reached on time’, this was dropped too. The categorical variables were changed into numbers enabling us to run a PCA. The next step was to normalize the data so that PCA didn’t end up skewed. Luckily, our data set didn’t have any missing values. In this first PCA run, 8 components were chosen with a cum. Explained variance ratio of 88.7%. We used conditional formatting in Excel to easily show the highest value in each row. By going through each row, we selected the predictor that had the highest value. The predictors selected were Customer\_care\_calls, Discount\_offered, Gender, Customer\_rating, Mode\_of\_Shipment, Product\_importance, and Prior\_purchase.

Additionally, a second PCA was run with these selected predictors:



The first 6 components were chosen with a cumulative Explained Variance Ratio of 88.4%, with the following predictors selected: Customer\_care\_calls, Gender, Customer\_rating, Product\_importance, and Discount\_offered.

The predictor that we would like to add to get a more rounded up PCA would be Month of delivery. Months like December would see a high volume in orders, and this may cause a greater number of orders to be delivered late. Winter months can also be a cause for late deliveries.

**Decision Tree Model**

Decision trees are non-parametric supervised learning used for classification and regression. The goal is to create a decision tree model to predict the target variable. In this project, we will find the predictors that have a major effect on the target variable i.e., on-time delivery, with the help of importance rate for each predictor.

As a step to clean the data and deal with missing values we use the imputation method which is filling missing values with mean column values in the data set.

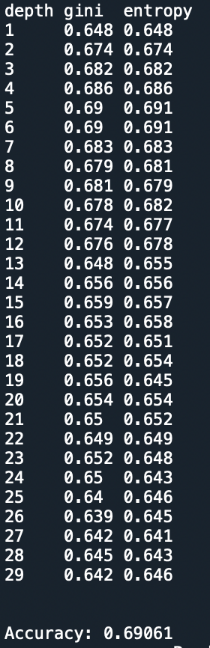
The second step is to drop unnecessary columns. This process not only helps the model in learning better but also speeds it up, making the model efficient. For this dataset, we will be dropping the **ID column** which is only a unique identifier. It is useless for prediction.

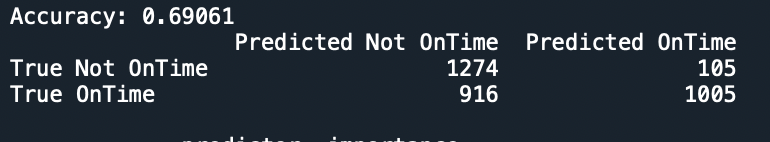
Machine learning algorithms understand numerical values more than text. Therefore, as the third step we convert categorical predictors into numbers, using LabelEncoder in python. For this reason, Warehouse\_block, Mode\_of\_Shipment, Product\_importance and gender are converted into numbers.

The above processes are part of data cleaning, which is similar for the two models which we build in the next steps. Next few processes include determining predictors, partitioning the data for training and testing and to determine target values. These steps are coded slightly differently for the two models.

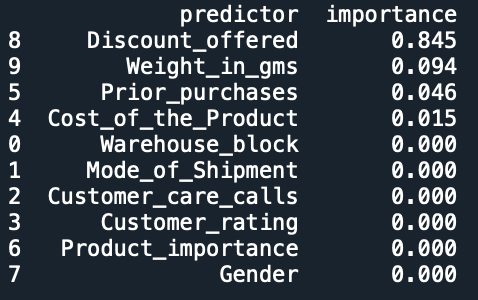
MODEL 1:

Initially to understand the importance of each predictor available in the dataset, we include all of them for the initial run. Here X = predictors and Y = Target Variables. Regarding partitioning the data, training is 70% and testing is 30%. Blocks of code are written to create the decision tree and to find the best accuracy rate we create a table of 29 different trees, using the Gini and entropy methods of purity checks for i in range (1,30):

In this model, the best depth is 5 with accuracy rate of 69.06%. We came to this conclusion with the fact that the lowest depth with highest accuracy rate gives the best Decision Tree. Now we create a decision tree with the suggested depth and test the model against the new model to determine the accuracy rate.



We also create a confusion matrix to understand the performance of the algorithm. This matrix compares the actual targeted values with the predicted values built by the model. From this matrix, we calculate the accuracy, misclassification, precision and sensitivity rate.

The next set of codes create a list of features and order them. This is a crucial step to understand the importance percentage of each predictor in the model. With this we can easily remove the unwanted predictors, which makes the model robust.

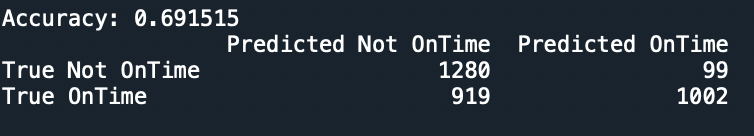
This clearly indicates to remove all other predictors than

* Discount offered – 84.5%
* Weight in grams – 9.4%
* Prior purchases – 4.6%
* Cost of the product – 1.5%

MODEL 2

With the chosen 4 predictors and ideal depth set to 6, we run model 2 to check for improvements in the accuracy rate against the first model. The accuracy increases to 69.15%. While looking at the decision tree, the first predictor is discount offered, the classifier is 10.5. When the discount rate is less than 10.5% the chance of on-time delivery is low. And when the discount rate is higher than 10.5% the chance of on-time delivery is high.

In this step, we prune the model further, to obtain a better decision tree by combining a few of the important predictors we inferred from PCA. I.e., by adding customer care calls, product importance etc. But such an addition does not increase the accuracy of the model. We prune the model by removing the cost of product variable with importance rate of 1.5% to see if the model improves, but it results in reducing the accuracy rate to 68.06%

To conclude, the best model will ideally have depth = 6 with 4 predictors and have an accuracy rate of 69.15%

**KNN**

To find clues about how data interacts with each other, data exploration must be done. Once complete, this exploration can lead to finding the best strategies to explore the data further through data mining techniques. To accomplish this, the following modules were utilized: Pandas, MatPlotLib, and Seaborn.

Utilizing Pandas, one can bring in the CSV file and assign the contents to a DataFrame, this DataFrame can then be further analyzed through the unique tools available to users in the Pandas module. The first visual exploration of this data was done by importing matplotlib and plotting the series of unique values of both objects and numbers within the dataset.   
Chart, waterfall chart

Description automatically generated

Following this, we can use matplotlib to lay out all numerical datapoints within the DataFrame to see the distribution of each value over time and whether there are any trends.

Calendar

Description automatically generated

Since the previous plot only showed numerical data, our analysis is incomplete as there is also categorical data which must be viewed and analyzed. To do this, a secondary plot can be generated using non numerical columns to plot out whether the input data skews toward any characteristics. We can see that there are some over-represented data points in each field, such as Warehouse\_block F, Mode\_of\_Shipment: ship, Product\_importance: Low, and the F Gender. Additionally, histograms of the numerical data can also be created to show if there are datapoints overrepresented in these categories.

Diagram

Description automatically generatedChart, box and whisker chart

Description automatically generated

Lastly, we can use Seaborn to visualize all our variables, how they directly interact with each other, and whether there are any outliers in our data.

Diagram, schematic

Description automatically generated

Utilizing all these visualization tools allows us to see which predictors are less uniform and therefore able to tell us more about whether our outcome target will be easily predicted. For the K-Nearest Neighbor method, we can see the variety and non-uniform structure of a few variables. The ones that we focused on were Cost of the Product, Discount Offered, and Weight in Grams. These three variables were the ones which seemed most random and able to make some determination of whether they will impact on time delivery or not.

Table

Description automatically generated with medium confidence To run the K-Nearest Neighbor method, one must normalize their data so that data is more comparable to other values within the DataFrame. Initially, the thought was to underfit the data to some of the categories which had values that were skewed higher in the histograms. Categories like Gender, Warehouse Block, and Method of Shipment. Once these three groups were scaled to be even, the result was an accuracy rating of 53.8%, showing that these fields did not have much of an impact on the on-time delivery even when considering their lop-sided nature.

Table

Description automatically generatedTable

Description automatically generatedTable

Description automatically generated with medium confidenceFinding that the accuracy of the model did not depend on scaling the high rate of repeat variables to an equal footing with the low rate of repeated variables, we then looked at other factors that could influence this model. Taking note of previous models, we passed through each models’ predictors to test which would have the highest accuracy: From our Principal Component Analysis, we tested our model against the five variables proposed which resulted in an accuracy of 60.8%. Next, from our decision tree model, we ran a KNN against the predictors: Discount Offered, Weight in Grams, Prior Purchases, and Cost of Product which resulted in an accuracy of 66%. Lastly, we used the predictors from the Random Forest Classifier. This resulted in nearly the worst accuracy rating we had when coparing models of 54.2% which came as a surprise since this model had performed so well (68.2%) on its own. In the end, we could not find a combination of predictors that resulted in a higher accuracy than that of the Decision Tree model.

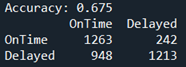
**Random Forest Classifier**

The Random Forest Classifier (RFC) is similar to a Tree Model Classifier in that both use trees; however, the RFC uses multiple uncorrelated trees for each prediction as an ensemble to more accurately predict the correct outcome. This is useful for instances where one tree may have a false classification, the rest of the trees in the forest or group have a chance to cover for the error of that tree and make the correct prediction. The technique is a supervised classifier able to be used for both classification and regression and will help predict whether a delivery will be “On Time” or “Delayed”.

Already cleaned, the data was then prepped for the code used by first splitting the dataset into a training set of 67%, and the other 33% into the test set. This amounted to 7,333 training observations and 3,666 testing observations. To do this split, the observations were first randomized in Excel and then the 33%test set was separated into the file “EcommerceTest.csv”, leaving the other 67% for the “EcommerceTrain.csv” file. The testing set includes 1,505 On Time deliveries and 2,161 Delayed deliveries. Once done, the script was modified to use our dataset while also adding the code block from the in-class script for the confusion matrix and the accuracy calculation.

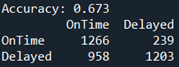
In this process, the number of trees and max depth will be changed multiple times, along with adding and dropping predictors. The number of trees is represented in the code by ‘n\_estimators = value’, with value being the tree amount. Similarly, max depth is represented by ‘max\_depth = value’. Both values can be found and manipulated in the parameters for the RandomForestClassifier function. The names of certain predictors have also been changed in this process: “Mode\_of\_Shipment” to “Shipment\_Type”, “Cost\_of\_Product” to “Product\_Cost”, “Discount\_Offered” to “Discount”, “Weight\_in\_gms” to “Weight”, and “Reached.on.Time\_Y.N.” to “On\_Time”. The files used in this process are: ‘randomforest.py’, ‘EcommerceTest.csv’, ‘EcommerceTrain.csv’, ‘InclassRFCver.py’, and ‘EcommerceData.csv’. The ‘InclassRFC.py’ has the data split and train as part of the code, unlike the manual split for the ‘randomforest.py’.

*First Run:*

 The initial accuracy is quite low compared to what was initially expected. This run used the values of 100 trees per forest and a max depth of 5. Before changing predictors, I experimented on the number of trees and max depth to increase the accuracy. In this experimentation, 100 was still the best, while a max depth of 8 gave an increased accuracy of 0.683, making 8 the new max depth.

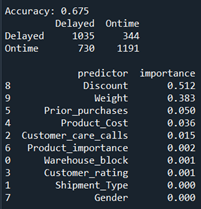
*Second Run:*

In this run predictors were removed to improve the accuracy of the set, while keeping the number of trees and max depth the same. The predictor variables removed were based on the PCA run earlier, leaving these variables to be the predictors: Warehouse\_block, Product\_Cost, and Weight.

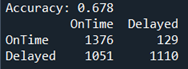
 This produced an accuracy of 0.673, lower than the first run only by an insignificant difference. Keeping this in mind, I experimented again with the number of trees and max depth to find a more ideal accuracy. While the number of trees never affected the outcome, changing the max depth to 5 increased the accuracy to 0.676. This run was not as ideal as the first run with all predictors included.

*Preparation for Third Run:*

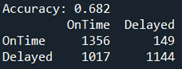
While still in the attempt to removing predictors for a better accuracy, and similarly to using the earlier PCA as a basis to remove predictors, a version of the in-class script was used that changed the Decision Tree Classifier functions to Random Forest Classifier functions. This was mainly done for the list of feature importance at the end of the script, while also used to have a comparison between available script options.

The range for the script code block that creates the Gini and Entropy table was set to 1,30 and a max depth of 3 was set in the RandomForestClassifier function parameters below as both were default. The accuracy for this run is relatively the same as the previous runs. The Gini and entropy table shows a best depth of 5 at the values of 0.688 for Gini and entropy. Using this, the script ran again with the new max depth of 5, and an improved result with an accuracy of 0.686. Based on these findings, the new choice of predictors were Discount, Weight, Prior\_purchases, and Product\_Cost. Also considering for the run the best max depth of 4 with a Gini and entropy of 0.690.

*Third Run:*

 Initially, a value of 100 trees and max depth of 8 were used as from all previous runs, returning the results to the left. The accuracy improved from previous attempts, but still not from the original with all predictors at an accuracy of 0.683. After performing more experimentation on the number of trees and max depth again, the final best accuracy for this run was 0.685 with 100 trees and a max depth of 4.

*Fourth Run:*

Before concluding, I wanted to test the predictors kept based off which I thought would be capable of predicting On Time or Delayed the best. The predictors for this run are Discount, Weight, Product\_importance, Product\_Cost, Customer\_care\_calls, and Shipment\_Type. As seen in the results to the left, the accuracy for this run at its best of 100 trees and a max depth of 8, was at 0.682. This was higher than expected for picking predictors based on their concept, but not entirely unsurprising.

After the process of performing these runs, I feel I have a good grasp on which predictors perform better than others and are closely correlated to On\_Time, with Discount being the most important. It was used in every run except the PCA related run, and for good reason because, while only a theory, it is most likely that a delivery was “delayed” and thus offered a discount to the customer. The other important predictor being Weight, could mean that it travelled slower by ship or that fewer products could be included in an already heavy shipment; however, this is all speculation.

Overall, the best possible accuracy found for this Random Forest Classifier script is 0.683 with 100 trees and a max depth of 8, while the adjusted in-class script was able to achieve an accuracy of 0.686 at a max depth of 5, and potentially could be higher with enough time and exploration. The average accuracy of this technique was lower than I anticipated with no run going over 0.690 into 0.700 or higher. I feel a better accuracy could be achieved by meticulously combing each combination of predictors; however, this would take a tremendous amount of time to accomplish and preparing the code to work at the start was challenging and time consuming enough.

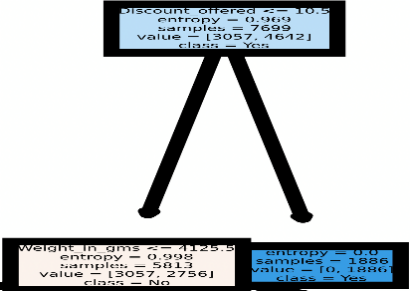
**Data Mining Technique Results**

PCA Results:

After running the first PCA we found that the first 8 components had a cumulative explained variance ratio of 88.7%. This also mean the first 8 components explained 88.7%of our model which we feel is good enough. This helped us to find those predictors that affected the model the most. Them being, Customer\_care\_calls, Discount\_offered, Gender, Customer\_rating, Mode\_of\_Shipment, Product\_importance, and Prior\_purchase. This helped us in reducing the number of predictors in our model and making our analysis easier.

We ran a second PCA to further slim down our model to make it more efficient. We chose the first 6 components that had a cumulative explained variance ratio of 88.4%. This helped us in further narrowing down our predictors. Them being, Customer\_care\_calls, Gender, Customer\_rating, Product\_importance, and Discount\_offered. PCA helped us in reducing the number of predictors from our initial 10 to just 6.

Decision Tree Results:

The tree has Discount offered as the root node with 6 leave nodes. If class= yes, the delivery is on time, class = no, the delivery is delayed. When value is (3057,2756) 3057 is the number of samples that were delayed. 2756 is the number of samples that were delivered on time. The first predictor is the discount offered with classifier as 10.5. When the discount rate is less than 10.5% the chance of on-time delivery is low. And when the discount rate is higher than 10.5% the chance of on-time delivery is high. Weight in grams is the next predictor with 4125.5 as the classifier. When the weight is less than 4125.5 grams the delivery is on time. This implies that heavier goods take more time to deliver on time. The same approach flows down to all other leaves in the decision tree. In the process of improvement, we also added predictors chosen from the PCA aiming to increase the accuracy rate. After various attempts to prune the data to make the model robust we improved the accuracy rate from 69.06% to 69.15%. Though the increase in accuracy was only 0.09% we could considerably cut the number of predictors to 4.

KNN Results:

Having run the KNN model four times, results varied from a low of 53.8% accuracy to a high of 66%. The most accurate results happened when Discount Offered, Weight in Grams, Prior Purchases, and Cost of Product were used as predictors. The highest accuracy level was still lower than all other models. Based on these results, we could evaluate these predictors against other models and see how they align. If we were able to do our own data collection, I feel like distance traveled would be an important variable that would influence on-time delivery.

Random Forest Classifier Results:

The final fourth run for the RFC had an accuracy of 68.2%, compared to the initial run having an accuracy of 68.3%; a minimal difference between both. Both used 100 trees and a max depth of 8 but used different predictors with the initial run using all and the final run using Discount, Weight, Product\_importance, Product\_Cost, Customer\_care\_calls, and Shipment\_Type. The best accuracy obtained in the RFC process was in the modified in-class script with 68.6% using a default of 100 trees and a max depth of 5. This adjusted script used all predictors except ID and the target On\_Time.

**Conclusion**

When we set out to determine what factors could impact an on-time delivery from an E-Commerce business, our group started by exploring the data we had gathered. There were some factors that were well represented, and we believed that those could have had a larger impact on on-time deliveries. Warehouse block, mode of shipment, and weight in grams were among the first predictors that we thought may result in a more accurate model. We then started looking at each group to see if there were any factors within those groups that related to having a better on-time rate than others. We found that products that were not delivered on time resulted in larger discounts being offered to the customers. Finally, before we ran our models, we needed to clean the data. Fortunately for us, we did not have any variables that contained NULL values.

Shanthosh Sivashanmugham ran a Principal Component Analysis. Jemimah Jesu ran a Decision Tree Analysis. Nicholas Ceparski ran the KNN Analysis. Dillon Drown ran a Random Forest Classifier Analysis.

The common predictor between our analysis was that of discount offered. This was the only predictor that everyone used in their models when they wanted to achieve the highest rate of accuracy. With products not delivered on time resulting in discounts, seeing that an on-time delivery was tied to discount price began to make sense. In the end, the Decision Tree resulted in the highest accuracy of all of our models at 69.15%.

**Appendix**

Citations:

* Gopalani, P. (2021). E-Commerce Shipping Data, [datafile]. <https://www.kaggle.com/datasets/prachi13/customer-analytics>

**Code for Principal Component Analysis**

‘BIM PCA.py’

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Thu Apr 21 18:53:32 2022

@author: shanthoshshanmugham

"""

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.preprocessing import LabelEncoder

#if you don't do this, your values will be in scientific notation

np.set\_printoptions(precision=3,suppress=True)

#read data into data frame, data

data = pd.read\_csv("Train.csv")

# Fill missing values with mean column values in the data set

# In this case, Age had some holes

data.fillna(data.mean(), inplace=True)

#drop the columns that we don't need!

data = data.drop(['ID'], axis=1)

data = data.drop(['Reached.on.Time\_Y.N'], axis=1)

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Warehouse\_block'])

data['Warehouse\_block'] = labelEncoder.transform(data['Warehouse\_block'])

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Mode\_of\_Shipment'])

data['Mode\_of\_Shipment'] = labelEncoder.transform(data['Mode\_of\_Shipment'])

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Mode\_of\_Shipment'])

data['Mode\_of\_Shipment'] = labelEncoder.transform(data['Mode\_of\_Shipment'])

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Product\_importance'])

data['Product\_importance'] = labelEncoder.transform(data['Product\_importance'])

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Gender'])

data['Gender'] = labelEncoder.transform(data['Gender'])

scaler = StandardScaler()

scaler.fit(data)

normalData = scaler.transform(data)

pca = PCA()

transformedPCA = pca.fit\_transform(normalData)

print("Explained variance:", pca.explained\_variance\_)

print("Explained variance ratio:", pca.explained\_variance\_ratio\_)

print("Cumulative explained variance ratio:", pca.explained\_variance\_ratio\_.cumsum())

print()

print("Feature Weights")

for i in data.columns:

print(i, end=' ')

print()

print(abs( pca.components\_ ))

weights = pd.DataFrame(abs(pca.components\_))

weights.columns = data.columns

writer = pd.ExcelWriter('output.xlsx')

weights.to\_excel(writer,index=False)

writer.save()

print()

print('###############################################')

print()

data = data.drop(['Warehouse\_block','Cost\_of\_the\_Product','Weight\_in\_gms'], axis=1)

scaler = StandardScaler()

scaler.fit(data)

normalData = scaler.transform(data)

pca = PCA()

transformedPCA = pca.fit\_transform(normalData)

print("Explained variance:", pca.explained\_variance\_)

print("Explained variance ratio:", pca.explained\_variance\_ratio\_)

print("Cumulative explained variance ratio:", pca.explained\_variance\_ratio\_.cumsum())

print()

print("Feature Weights")

for i in data.columns:

print(i, end=' ')

print()

print(abs( pca.components\_ ))

weights2 = pd.DataFrame(abs(pca.components\_))

weights2.columns = data.columns

writer2 = pd.ExcelWriter('output2.xlsx')

weights2.to\_excel(writer2,index=False)

writer2.save()

**Code for Tree Model Classifier**

"""

Created on Thu Apr 14 19:33:16 2022

@author: jemimahjesu

"""

import numpy as np

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score, confusion\_matrix

import matplotlib.pyplot as plt

#read data into data frame, data

data = pd.read\_csv("Train.csv")

# Fill missing values with mean column values in the data set

data.fillna(data.mean(), inplace=True)

#drop the columns that we don't need!

data = data.drop(['ID'], axis=1)

#convert category, Warehouse\_block, into a number

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Warehouse\_block'])

data['Warehouse\_block'] = labelEncoder.transform(data['Warehouse\_block'])

#convert category, into a number

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Mode\_of\_Shipment'])

data['Mode\_of\_Shipment'] = labelEncoder.transform(data['Mode\_of\_Shipment'])

#convert category, into a number

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Product\_importance'])

data['Product\_importance'] = labelEncoder.transform(data['Product\_importance'])

#convert category, into a number

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Gender'])

data['Gender'] = labelEncoder.transform(data['Gender'])

#list all of the predictors that are left

predictors = ['Warehouse\_block','Mode\_of\_Shipment','Customer\_care\_calls','Customer\_rating','Cost\_of\_the\_Product','Prior\_purchases','Product\_importance','Gender','Discount\_offered','Weight\_in\_gms']

#set up target, predictors, and split the training/testing partitions

X = data[predictors]

y = data['Reached.on.Time\_Y.N']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

#create the decision tree

estimator = DecisionTreeClassifier()

estimator.fit(X\_train, y\_train)

#This block of code creates a table of 29 different trees, using the Gini and

# entropy methods of purity checks, prints the accuracy of each model

print('depth','gini ','entropy')

for i in range(1,30):

dtree = DecisionTreeClassifier(criterion='gini', max\_depth=i)

dtree.fit(X\_train, y\_train)

pred = dtree.predict(X\_test)

gini\_score = accuracy\_score(y\_test, pred)

####

dtree = DecisionTreeClassifier(criterion='entropy', max\_depth=i)

dtree.fit(X\_train, y\_train)

pred = dtree.predict(X\_test)

entropy\_score = accuracy\_score(y\_test, pred)

#print(i,round(gini\_score,3),round(entropy\_score,3))

print(f'{i:<6}{round(gini\_score,3):<6}{round(entropy\_score,3)}')

####

print()

print()

#create the final tree with the suggested depth from the above code block

dtree = DecisionTreeClassifier(criterion='entropy', max\_depth=5)

dtree.fit(X\_train, y\_train)

#test the model against our new model and calculate the accuracy

pred = dtree.predict(X\_test)

print("Accuracy:", round(accuracy\_score(y\_test, pred),5))

#create confusion matrix and print it

confusionMatrix = pd.DataFrame(

confusion\_matrix(y\_test, pred),

columns=['Predicted Not OnTime', 'Predicted OnTime'],

index=['True Not OnTime', 'True OnTime']

)

print(confusionMatrix)

print()

#create a list of features and orders them

importances = pd.DataFrame({'predictor':X\_train.columns,'importance':np.round(dtree.feature\_importances\_,3)})

importances = importances.sort\_values('importance',ascending=False)

print(importances)

print()

#######################################################################

#list all of the predictors that are left

predictors = ['Prior\_purchases','Discount\_offered','Weight\_in\_gms','Cost\_of\_the\_Product']

#set up target, predictors, and split the training/testing partitions

X = data[predictors]

y = data['Reached.on.Time\_Y.N']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

#create the decision tree

estimator = DecisionTreeClassifier()

estimator.fit(X\_train, y\_train)

#This block of code creates a table of 29 different trees, using the Gini and

# entropy methods of purity checks, prints the accuracy of each model

print('depth','gini ','entropy')

for i in range(1,30):

dtree = DecisionTreeClassifier(criterion='gini', max\_depth=i)

dtree.fit(X\_train, y\_train)

pred = dtree.predict(X\_test)

gini\_score = accuracy\_score(y\_test, pred)

####

dtree = DecisionTreeClassifier(criterion='entropy', max\_depth=i)

dtree.fit(X\_train, y\_train)

pred = dtree.predict(X\_test)

entropy\_score = accuracy\_score(y\_test, pred)

#print(i,round(gini\_score,3),round(entropy\_score,3))

print(f'{i:<6}{round(gini\_score,3):<6}{round(entropy\_score,3)}')

####

print()

print()

#create the final tree with the suggested depth from the above code block

dtree = DecisionTreeClassifier(criterion='entropy', max\_depth=6)

dtree.fit(X\_train, y\_train)

#test the model against our new model and calculate the accuracy

pred = dtree.predict(X\_test)

print("Accuracy:", round(accuracy\_score(y\_test, pred),6))

#create confusion matrix and print it

confusionMatrix = pd.DataFrame(

confusion\_matrix(y\_test, pred),

columns=['Predicted Not OnTime', 'Predicted OnTime'],

index=['True Not OnTime', 'True OnTime']

)

print(confusionMatrix)

print()

#create a list of features and orders them

importances = pd.DataFrame({'predictor':X\_train.columns,'importance':np.round(dtree.feature\_importances\_,3)})

importances = importances.sort\_values('importance',ascending=False)

print(importances)

print()

plt.figure()

plot\_tree(dtree, filled=True, feature\_names=predictors, class\_names=['No','Yes'])

plt.savefig('treePlot.pdf')

plt.show()

**Code for K-Nearest Neighbor**

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

dataAll = **pd**.**read\_csv**("Train.csv")  
  
import **matplotlib**.**pyplot** as **plt**

%matplotlib widget

dataAll.shape

**pd**.**value\_counts**(dataAll.dtypes)

dataAll.**select\_dtypes**(include='object').**head**()

dataAll.**describe**(datetime\_is\_numeric=False)

unique\_values = dataAll.**select\_dtypes**(include='object').**nunique**().**sort\_values**()

unique\_values.**plot**.bar(logy=False)

**plt**.**gcf**().subplots\_adjust(bottom=0.5)

**plt**.**show**()

dataAll.shape

**pd**.**value\_counts**(dataAll.dtypes)

dataAll.**select\_dtypes**(include='number').**head**()

dataAll.**describe**(datetime\_is\_numeric=True)

unique\_values = dataAll.**select\_dtypes**(include='number').**nunique**().**sort\_values**()

unique\_values.**plot**.bar(logy=True)

**plt**.**gcf**().subplots\_adjust(bottom=0.5)

**plt**.**show**()

**plt**.**figure**(figsize=(10,8))

**plt**.**imshow**(dataAll.**isna**(), aspect="auto", interpolation="nearest",cmap="gray")

**plt**.**xlabel**("Column Number")

**plt**.**ylabel**("Sample Number")

import **missingno** as **msno**

**msno**.**matrix**(dataAll,labels=True, sort="descending")

dataAll.**plot**(lw=0,marker=".",subplots=True,layout=(-1,4),figsize=(15,30),markersize=1)

dataAll.**describe**(exclude=['number','datetime'])

fig, axes = **plt**.**subplots**(ncols=1,nrows=4,figsize=(6,8))

df\_non\_numerical = dataAll.**select\_dtypes**(exclude=['number','datetime'])

for col, ax in **zip**(df\_non\_numerical.columns, axes.ravel()):

    df\_non\_numerical[col].**value\_counts**().**plot**(

        logy=False, title=col, lw=0, marker=".", ax=ax)

**plt**.**tight\_layout**()

dataAll.**hist**(bins=25, figsize=(15,25),layout=(-1,1),edgecolor="black")

**plt**.**table**

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

**sns**.**pairplot**(dataAll)

**plt**.**savefig**('scatter.png')

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

from **sklearn**.**neighbors** import **KNeighborsClassifier**

from **sklearn**.**preprocessing** import **StandardScaler**

from **sklearn**.**preprocessing** import **MinMaxScaler**

from **sklearn**.**preprocessing** import **LabelEncoder**

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

*#read data into data frame, data*

*#drop the columns that we don't need!*

*#dataAll = dataAll.drop(['ID'], axis=1)*

labelEncoder = **LabelEncoder**()

labelEncoder.**fit**(dataAll['Warehouse\_block'])

dataAll['Warehouse\_block'] = labelEncoder.**transform**(dataAll['Warehouse\_block'])

labelEncoder = **LabelEncoder**()

labelEncoder.**fit**(dataAll['Mode\_of\_Shipment'])

dataAll['Mode\_of\_Shipment'] = labelEncoder.**transform**(dataAll['Mode\_of\_Shipment'])

labelEncoder = **LabelEncoder**()

labelEncoder.**fit**(dataAll['Product\_importance'])

dataAll['Product\_importance'] = labelEncoder.**transform**(dataAll['Product\_importance'])

labelEncoder = **LabelEncoder**()

labelEncoder.**fit**(dataAll['Gender'])

dataAll['Gender'] = labelEncoder.**transform**(dataAll['Gender'])

*#List all of the predictors needed - DO NOT include the target*

predictors = [

*#'Warehouse\_block',*

*#'Mode\_of\_Shipment',*

*#'Customer\_care\_calls',*

*#'Customer\_rating',*

*#'Product\_importance',*

*#'Gender',*

*#'Cost\_of\_the\_Product',*

    'Prior\_purchases',

    'Discount\_offered',

    'Weight\_in\_gms'

    ]

*#set up target, predictors, and split the training/testing partitions*

X = dataAll[predictors]

y = dataAll['Reached\_on\_Time\_Y\_N']

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size=0.3, random\_state=0)

*#Normalize your data Here!*

'''

scaler = MinMaxScaler()

#scaler.fit(X\_train)

scaler.fit\_transform(X\_train)

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.fit\_transform(X\_test)

'''

scaler = **StandardScaler**()

*# Fit only to the training data*

scaler.**fit**(X\_train)

**StandardScaler**(copy=True, with\_mean=True, with\_std=True)

*# Now apply the transformations to the data:*

X\_train = scaler.**transform**(X\_train)

X\_test = scaler.**transform**(X\_test)

*#knn = KNeighborsClassifier(n\_neighbors=7, metric='euclidean'*

*#Calculate the optimal k based on test partition accuracy*

k\_range = **range**(1,40)

for k in k\_range:

    knn = **KNeighborsClassifier**(n\_neighbors=k, weights='uniform')

    knn.**fit**(X\_train,y\_train)

    pred=knn.**predict**(X\_test)

    acc=**accuracy\_score**(y\_test,pred)

**print**(k, **round**(acc,3))

*#Create a kNN model with recommended k (7)*

knn = **KNeighborsClassifier**(n\_neighbors=7)

knn.**fit**(X\_train,y\_train)

*#test model against test partition*

pred = knn.**predict**(X\_test)

**print**()

confusionMatrix = **pd**.**DataFrame**(

**confusion\_matrix**(y\_test, pred),

    columns=['Predicted Delayed', 'Predicted On Time'],

    index=['True Delayed', 'True On Time'])

**print**(confusionMatrix)

**print**("Accuracy:", **round**(**accuracy\_score**(y\_test, pred),3))

**Code for Random Forest Classifier**

‘randomforest.py’

# -\*- coding: utf-8 -\*-

"""

Created on Tue Apr 22 16:23:51 2022

@author/modified for use by: ddrown

Code obtained from:

https://www.kaggle.com/code/alexisbcook/titanic-tutorial/notebook

"Titanic Tutorial"

author: Alexis Cook

September 2021

Note: Parts have been picked and copy pasted from site as this was a tutorial for another dataset.

"""

#Needed to force where I wanted code to look for file

import os

os.chdir(r'C:\Users\ddrown\OneDrive - University of Toledo\Desktop\Data exploration')

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import RandomizedSearchCV

train\_data = pd.read\_csv("EcommerceTrain.csv")

#print(train\_data.head(5))

test\_data = pd.read\_csv("EcommerceTest.csv")

#print(test\_data.head(5))

Y = train\_data["On\_Time"]

#run 1, all features

features = ["Warehouse\_block", "Product\_Cost","Weight","Shipment\_Type","Customer\_care\_calls","Customer\_rating", "Prior\_purchases","Product\_importance","Gender","Discount"]

#run 2

#features = ["Warehouse\_block", "Product\_Cost","Weight"]

#run 3

#features = ["Discount", "Weight", "Prior\_purchases", "Product\_Cost"]

#run 4

#features = ["Discount", "Weight", "Product\_importance", "Product\_Cost", "Customer\_care\_calls", "Shipment\_Type"]

X = pd.get\_dummies(train\_data[features])

X\_test = pd.get\_dummies(test\_data[features])

y\_test = test\_data["On\_Time"]

#%% Run model

RFmodel = RandomForestClassifier(n\_estimators=100, max\_depth=8, random\_state=1)

RFmodel.fit(X,Y)

predictions = RFmodel.predict(X\_test)

output = pd.DataFrame({'ID': test\_data.ID, 'On\_Time': predictions})

output.to\_csv('EcommerceOutput.csv',index=False)

#print(output)

#code below is from homework assignments due to wanting to see a confusion matrix and accuracy

print("Accuracy:", round(accuracy\_score(y\_test, predictions),3))

#create confusion matrix and print it

confusionMatrix = pd.DataFrame(

confusion\_matrix(y\_test, predictions),

columns=['OnTime', 'Delayed'],

index=['OnTime', 'Delayed']

)

print(confusionMatrix)

print()

‘InclassRFCver.py’

# -\*- coding: utf-8 -\*-

"""

Created on Tue Apr 22 17:04:21 2022

@author: Steve Wallace

modified for use by: ddrown

"""

#Needed to force where I wanted code to look for file

import os

os.chdir(r'C:\Users\ddrown\OneDrive - University of Toledo\Desktop\Data exploration')

import numpy as np

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import RandomizedSearchCV

import matplotlib.pyplot as plt

#read data into data frame, data

data = pd.read\_csv("EcommerceCSV.csv")

# Fill missing values with mean column values in the data set

data.fillna(data.mean(), inplace=True)

#drop the columns that we don't need!

#data = data.drop([''], axis=1)

#convert category into a number value

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Warehouse\_block'])

data['Warehouse\_block'] = labelEncoder.transform(data['Warehouse\_block'])

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Shipment\_Type'])

data['Shipment\_Type'] = labelEncoder.transform(data['Shipment\_Type'])

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Product\_importance'])

data['Product\_importance'] = labelEncoder.transform(data['Product\_importance'])

labelEncoder = LabelEncoder()

labelEncoder.fit(data['Gender'])

data['Gender'] = labelEncoder.transform(data['Gender'])

#list all of the predictors that are left

predictors = ['Warehouse\_block', 'Shipment\_Type', 'Customer\_care\_calls', 'Customer\_rating', 'Product\_Cost', 'Prior\_purchases', 'Product\_importance', 'Gender', 'Discount', 'Weight']

#set up target, predictors, and split the training/testing partitions

X = data[predictors]

y = data['On\_Time']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

#create the decision tree

estimator = RandomForestClassifier(random\_state=1)

estimator.fit(X\_train, y\_train)

#This block of code creates a table of different trees, using the Gini and

# entropy methods of purity checks, prints the accuracy of each model

print('depth','gini ','entropy')

for i in range(1,30):

forest = RandomForestClassifier(criterion='gini', max\_depth=i)

forest.fit(X\_train, y\_train)

pred = forest.predict(X\_test)

gini\_score = accuracy\_score(y\_test, pred)

####

forest = RandomForestClassifier(criterion='entropy', max\_depth=i)

forest.fit(X\_train, y\_train)

pred = forest.predict(X\_test)

entropy\_score = accuracy\_score(y\_test, pred)

#print(i,round(gini\_score,3),round(entropy\_score,3))

print(f'{i:<6}{round(gini\_score,3):<6}{round(entropy\_score,3)}')

####

print()

print()

#create the final tree with the suggested depth from the above code block

forest = RandomForestClassifier(criterion='entropy', max\_depth=3)

forest.fit(X\_train, y\_train)

#test the model against our new model and calculate the accuracy

pred = forest.predict(X\_test)

print("Accuracy:", round(accuracy\_score(y\_test, pred),3))

#create confusion matrix and print it

confusionMatrix = pd.DataFrame(

confusion\_matrix(y\_test, pred),

columns=['Delayed', 'Ontime'],

index=['Delayed', 'Ontime']

)

print(confusionMatrix)

print()

#create a list of features and orders them

importances = pd.DataFrame({'predictor':X\_train.columns,'importance':np.round(forest.feature\_importances\_,3)})

importances = importances.sort\_values('importance',ascending=False)

print(importances)

print()