Title: Study on predicting the retail prices of onion in the local market of Bangladesh.

Author Neaz Mahmood
Department of Computer Science and Engineering
University of Information Technology and Sciences
Neazmahmood445@gmail.com

Abstract Demand and supply forces are often expected to determine the prices of a product in the market. However, sometimes these forces may lead to unprecedented price increases in the market, making it necessary to have other forces to help regulate the market prices. In Bangladesh, in October 2021 and March 2022, the demand and supply factors caused a sharp rise in onion prices, leading to a public outcry. This price hike was caused by various factors like bad harvesting and restriction on the export of onion by the neighboring country.
In this study, we tried to predict the price of onions using various machine learning models and determine which machine learning model gives a better result.

Introduction Onion (*Allium cepa*) is among the most popular vegetables in the world. Onion is a crop that is classified as a cool-season crop. However, it can be grown in a wide range of climatic conditions. It is grown mainly for its bulb, which is used in every home almost daily across Ethiopia (AgroBIG, 2016). Ethiopia, the third-biggest pinion producer in the African continent, next to Egypt and South Africa, contributes only 2.7% to the total world production between 2000–2011 (FAOSTAT, 2019). Averagedthroughoutf 2010 to 2018, the onion area harvested, production, and yield at the national level are 28,942 hectares, 33,947 tons, and 11.70 tons/ha, respectively, (CntrCentralistical Agency (CSA), 2019; Food and Agriculture Organization Statistical Division (FAOSTAT), 2019), which is far below the world average of 19.7 tons/ha (Megersa, 2017). In the major rainy season of 2018/19, onion production covered about 11.46% of the country's root crop area.[1] In the same season, it was grown by 28,682 smallholder farm households in Tigray, Ethiopia. Together, these households produced 8,223 tons of onion on 1,299.06 hectares with yields of 6.33 tons/ha, which than the national average yield of 9.32 tons/ha in the same season (CSA, 2019).

Onion is one of the major spices crops in Bangladesh. It is used as a common ingredient for its own unique flavor in preparing various types of cooking in our country. It has curative power that makes it essential to use them in the medicinal plant too.

Data set. The study area is every subdistrict market of Bangladesh. The data set consists of 7 columns and 6009 rows. The data was collected from 'www.dam.gov.bd.' website. The data set includes specific dates and the retail prices of various types of onions in the subdistrict market. Price consists of 3 values max, min, and mean price. We also added the corresponding Arabic calendar date and USD rate for every date. The 'MEAN' column is the output of this dataset, and we used every other column to find the relation with the output column. I labeled the string values 'location', 'type' 'date' & 'MEAN.'

Dataset sample(Before labeling)

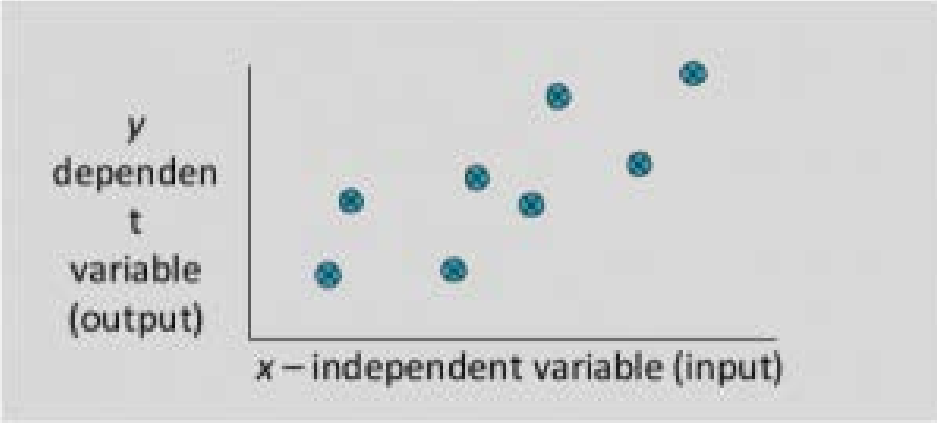
Arabic_date	Date	USD rate	Location	type	MIN	MAX	MEAN
15/5/1442	2020-12-30	84.78	Chandpur Sadar	Imported	34.0	35.0	34.5
15/5/1442	2020-12-30	84.78	Cox's Bazar Sadar	Imported	35.0	60.0	47.5
15/5/1442	2020-12-30	84.78	Khagrachhari Sadar	Imported	25.0	35.0	30.0
15/5/1442	2020-12-30	84.78	Lakshmipur Sadar	Imported	30.0	35.0	32.5
15/5/1442	2020-12-30	84.78	Rangamati Sadar	Imported	27.0	30.0	28.5

Dataset sample(After labeling)

Arabic_date	Date	USD rate	Location	type	MIN	MAX	MEAN
9/11/1443	1	92.92	1	2	50.0	60.0	2
9/11/1443	1	92.92	2	3	25.0	30.0	3
9/11/1443	1	92.92	3	1	34.0	35.0	4
9/11/1443	1	92.92	4	2	30.0	35.0	5

Supervised Machine Learning, The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (x) and an output variable (Y), and you use an algorithm to learn the mapping function from the input to the output Y = f(X) . The goal is to approximate the mapping function so well that when you have new input data (x), you can predict the output variables (Y) for that data.
Techniques of Supervised Machine Learning algorithms include linear and logistic regression, multi-class classification, Decision Trees and support vector machines.
Supervised learning requires that the data used to train the algorithm is already labelled with correct answers. For example, a classification algorithm will learn to identify animals after being trained on a dataset of images that are properly labelled with the species of the animal and some identifying characteristics.
Supervised learning problems can be further grouped into Regression and Classification problems. Both issues aim to construct a brief model that can predict the value of the dependent attribute from the attribute variables. The difference between the two tasks is the dependent attributepointical for regression and categorical for classification.

Regression Problem A regression problem is when the output variable is a natural or continuous value, such as “salary” or “weight”. Many different models can be used; the simplest is linear regression. It tries to fit data with the best hyper-plane, which goes through the points.



Multiple linear regression attempts to model the relationship between two or more features and a response by fitting a linear equation to the observed data.
It is nothing but an extension of simple linear regression.
Consider a dataset with p features(or independent variables) and one response(or dependent variable).
Also, the dataset contains n rows/observations.
We define:

X (feature matrix) = a matrix of size n X p where x_{ij} denotes the values of jth feature for ith observation.
So,

$$\begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \vdots & x_{np} \end{pmatrix}$$

and
y (response vector) = a vector of size n where y_{i} denotes the response value for ith observation.

$$y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{bmatrix}$$

The regression line for p features is represented as:

$$h(x_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}$$

Where h(x_i) is the predicted response value, the for the ith observation and b_0, b_1, …, b_p are the regression coefficients.
Also, we can write:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \varepsilon_i$$

or

$$y_i = h(x_i) + \varepsilon_i \Rightarrow \varepsilon_i = y_i - h(x_i)$$

where e_i represents a residual error in ith observation.
We can generalize our linear model a little bit more by representing feature matrix X as:

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}$$

So now, the linear model can be expressed in terms of matrices as:

$$y = X\beta + \varepsilon$$

where,

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ . \\ . \\ \beta_p \end{bmatrix}$$

And

$$\varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ . \\ . \\ \varepsilon_n \end{bmatrix}$$

Now, we determine an estimate of b, i.e., by using the Least Squares method.
As already explained, the Least Squares method tends to determine b' for which total residual error is minimized.
We present the result directly here:

$$\hat{\beta} = (X'X)^{-1}X'y$$

Where ' represents the transpose of the matrix while -1 represents the matrix inverse.
Knowing the least square estimates, b', the multiple linear regression model can now be estimated as:

$$\hat{y} = X\hat{\beta}$$

where y' is the estimated response vector.

Code for fitting the data frame to multiple linear regression model

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
 random_state = 0)

#Fitting the MLR model to the training set:
from sklearn.linear_model import LinearRegression
lr = LinearRegression(normalize=True)
lr.fit(X_train, y_train)

#Predicting the Test set result;
y_pred= lr.predict(X_test)

print('Train Score: ', lr.score(X_train, y_train))
print('Test Score: ', lr.score(X_test, y_test))
```

Result of the multiple linear regression model

Train Score: 11%
Test Score: 10%

Code for getting the RMSE

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

```
import math

MSE = nm.square(nm.subtract(y_test,y_pred)).mean()

RMSE = math.sqrt(MSE)
print("Root Mean Square Error:\n")
print(RMSE)
```

RMSE result

```
Root Mean Square Error:

22.014
```

Root Mean Squared Error or RMSE

MSE is calculated by taking the average of the square of the difference between the original and predicted values of the data.

$$\frac{1}{N}\sum_{i=1}^{n}(\text{actual values} - \text{predicted values})^2$$

Hence, MSE =

Here N is the total number of observations/rows in the dataset. The sigma symbol denotes that the difference between actual and predicted values taken on every i value ranging from **1 to n**.

RMSE is the standard deviation of the errors which occur when a prediction is made on a dataset. This is the same as MSE (Mean Squared Error) but the root of the value is considered while determining the accuracy of the model.

Classification Problem

Classification Methods Artificial intelligence methods facilitated the classification and the determination of the validity of the results. There were six different ML (supervised learning) methods in this study. These methods are k-nearest neighbors (kNN), support

vector machine (SVM), random forest (RF), Decision tree, naïve Bayes (NB), and logistic regression (LR).

k-Nearest Neighbors (kNN) This algorithm makes a clustering process based on the proximity relations between objects. It works in the coordinate plane with the linear decomposition method that obtains neighbor data using the Euclidean distance between data points.

Support Vector Machine (SVM) finds the best way to classify the data based on the position in relation to a border between positive class and negative class. This border is known as the hyperplane, which maximizes the distance between data points from different classes. Like the decision tree and random forest, a support vector machine can be used in both classification and regression, SVC; SVCort vector classifier) is for classification problems.

Random Forest (RF) random forest is a collection of decision trees. It is a common type of ensemble method which aggregates results from multiple predictors. Random forest additionally utilizes a bagging technique that allows each tree to be trained on a random sampling of the original dataset and takes the majority vote from trees. Compared to dethe cision tree, it has better generalization but is leinterpretablble because of more layers added to the model.

Decision tree builds tree branches in a hierarchical approach, and each branch can be considered an if-else statement. The branches develop by partitioning the dataset into subsets based on the most imessentialeatures. Final classification happens at the leaves of the decision tree.

Naïve Bayes (NB) The naïve Bayes re-scans the entire dataset for each new classification operation which might cause it to operate relatively slowly.

Logistic Regression (LR) LR is the iterative presentation of the powerful linear combination of variables most likely to determine the observed outcome.

Code for fitting the data frame to classification models

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
```

```
model_pipeline = []
model_pipeline.append(LogisticRegression(solver='liblinear'))
model_pipeline.append(SVC())
model_pipeline.append(KNeighborsClassifier())
model_pipeline.append(DecisionTreeClassifier())
model_pipeline.append(RandomForestClassifier())
model_pipeline.append(GaussianNB())

from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

model_list = ['Logistic Regression', 'SVM', 'KNN', 'Decison Tree', 'Random
 Forest', 'Naive Bayes']
acc_list = []
cm_list = []

for model in model_pipeline:
  model.fit(X_train, y_train)
  y_pred = model.predict(X_test)
  acc_list.append(metrics.accuracy_score(y_test, y_pred))
  cm_list.append(confusion_matrix(y_test, y_pred))

import numpy as np
result_df = pd.DataFrame({'Model':model_list, 'Accuracy': acc_list})
#result_df=result_df.round({"Accuracy":2})
result_df['Accuracy'] =  np.round(result_df['Accuracy'], decimals = 3)
#result_df.round(2)
print(result_df)
```

Code for getting Sensitivity(sn), specifity(sp), F1 Score and MCC

```
# Creating a function to report confusion metrics
def confusion_metrics (conf_matrix):
# save confusion matrix and slice into four pieces
    TP = conf_matrix[1][1]
    TN = conf_matrix[0][0]
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
```

```
    print('True Positives:', TP)
    print('True Negatives:', TN)
    print('False Positives:', FP)
    print('False Negatives:', FN)

    # calculate accuracy
    conf_accuracy = (float (TP+TN) / float(TP + TN + FP + FN))

    # calculate mis-classification
    conf_misclassification = 1- conf_accuracy

    # calculate the sensitivity
    conf_sensitivity = (TP / float(TP + FN))
    # calculate the specificity
    conf_specificity = (TN / float(TN + FP))

    # calculate precision
    conf_precision = (TN / float(TN + FP))
    # calculate f_1 score
    conf_f1 = 2 * ((conf_precision * conf_sensitivity) / (conf_precision +
 conf_sensitivity))
    print('-'*50)
    print(f'Accuracy: {round(conf_accuracy,2)}')
    print(f'Mis-Classification: {round(conf_misclassification,2)}')
    print(f'Sensitivity: {round(conf_sensitivity,2)}')
    print(f'Specificity: {round(conf_specificity,2)}')
    print(f'Precision: {round(conf_precision,2)}')
    print(f'f_1 Score: {round(conf_f1,2)}')

for cm in cm_list:
  confusion_metrics (cm)
```

Table: without feature selection

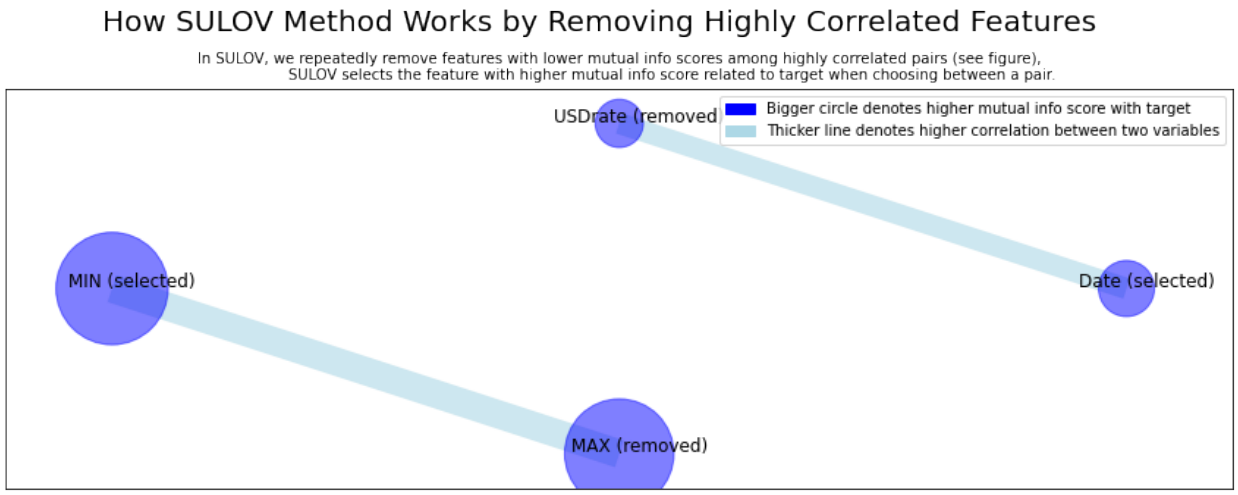
Classifier	Sn /recall	sp	acc	MCC	F1
Logistic Regression (LR)	undefined	1	21%	0	undefined

Support Vector Machine (SVM)	undefined	1	32%	0	undefined
k-Nearest Neighbors (KNN)	1	1	50%	0	1
Decision Tree	1	1	97%	0	1
Random Forest(RF)	1	1	92%	0	1
Naive Bayes(NB)	1	1	75%	0	1

Feature Selection

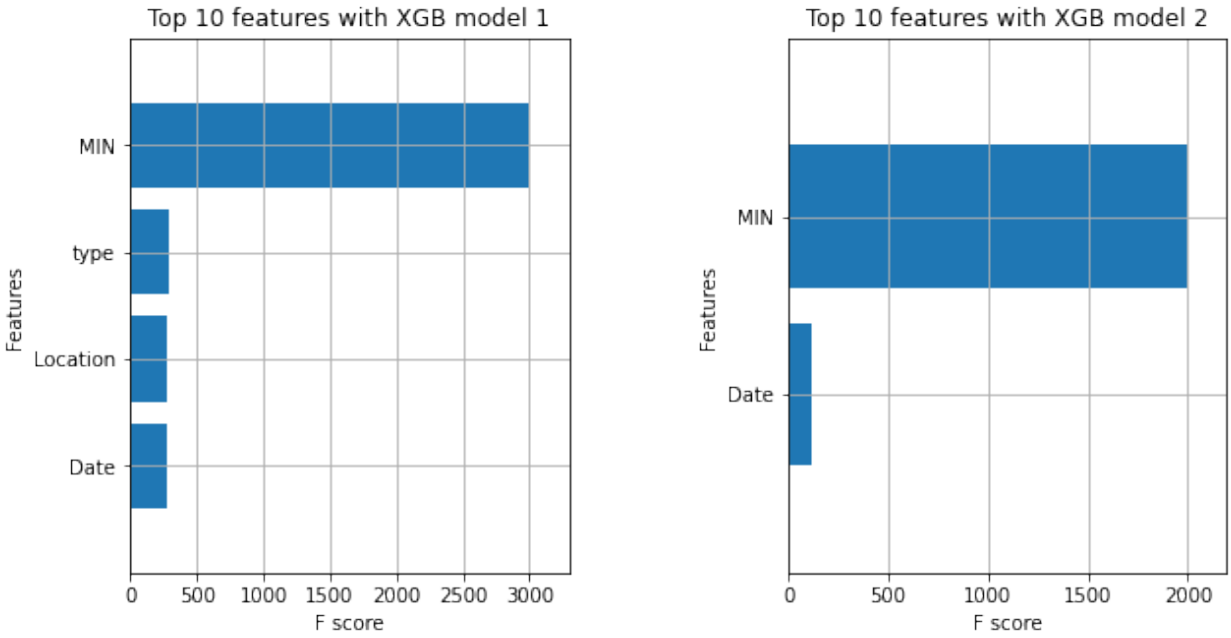
```
Imported version = 0.1.87.
from featurewiz import FeatureWiz
wiz = FeatureWiz(verbose=1)
X_train_selected = wiz.fit_transform(X_train, y_train)
X_test_selected = wiz.transform(X_test)
wiz.features  ### provides a list of selected features ###

############################################################################
###############
############      F A S T   F E A T U R E  E N G G   A N D   S E L E C T I
O N ! ########
# Be judicious with featurewiz. Don't use it to create too many un-
interpretable features! #
############################################################################
###############
Skipping feature engineering since no feature_engg input...
Skipping category encoding since no category encoders specified in input...
**INFO: featurewiz can now read feather formatted files. Loading train
data...
    Shape of your Data Set loaded: (6009, 7)
    Loaded train data. Shape = (6009, 7)
    Some column names had special characters which were removed...
No test data filename given...
############################################################################
##########
######################## C L A S S I F Y I N G  V A R I A B L E S
####################
############################################################################
##########
```

```
Classifying variables in data set...
    6 Predictors classified...
        No variables were removed since no ID or low-information variables
found in data set
GPU active on this device
    Tuning XGBoost using GPU hyper-parameters. This will take time...
    After removing redundant variables from further processing, features left
= 6
No interactions created for categorical vars since feature engg does not
specify it
#### Single_Label Regression problem ####
    target labels need to be converted...
################################################################################
##########
#####  Searching for Uncorrelated List Of Variables (SULOV) in 6 features
############
################################################################################
##########
    there are no null values in dataset...
    Removing (2) highly correlated variables:
    ['USDrate', 'MAX']
    Following (4) vars selected: ['Location', 'type', 'MIN', 'Date']
```




```
Time taken for SULOV method = 1 seconds
    Adding 0 categorical variables to reduced numeric variables  of 4
Final list of selected vars after SULOV = 4
Readying dataset for Recursive XGBoost by converting all features to
numeric...
################################################################################
##########
#####    R E C U R S I V E   X G B O O S T : F E A T U R E   S E L E C T I O
N  #######
################################################################################
##########
    using regular XGBoost
Train and Test loaded into Dask dataframes successfully after feature_engg
completed
Current number of predictors = 4
    XGBoost version: 1.6.1
Number of booster rounds = 100
        using 4 variables...
```

```
    Time taken for regular XGBoost feature selection = 1 seconds
using 2 variables...
    Time taken for regular XGBoost feature selection = 2 seconds
```




```
    Total time taken for XGBoost feature selection = 2 seconds
##############################################################################
##########
#####          F E A T U R E   S E L E C T I O N   C O M P L E T E D
#######
##############################################################################
##########
Selected 4 important features:
['MIN', 'type', 'Location', 'Date']

    Time taken for feature selection = 3 seconds
Returning 2 dataframes: dataname and test_data with 4 important features.
```

Top 3 classifiers with feature selection

Classifier	Number of Features	Sn /recall	sp	acc	MCC	F1
Logistic Regression (LR)	4	1	1	14	0	1
Support Vector	4	undefined	1	28	0	undefined

Machine (SVM)						
k-Nearest Neighbors (KNN)	4	1	1	36	0	1
Decision Tree	4	undefined		56	0	undefined
Random Forest(RF)	4	1	1	58	0	1
Naive Bayes(NB)	4	undefined	1	48	0	undefined

Result

Confusion Matrix to evaluate model performance

A confusion matrix is used to display parameters in a matrix format. It allows us to visualize true and false positives and true and false negatives.

To get the overall accuracy, we can subtract total false positives and negatives from the total number of tests and divide that by the total number of tests. We can use a confusion matrix by importing it through the *sklearn* library. Scikit-learn (sklearn) in Python contains machine learning and statistical modeling tools, including methods from classification to dimensionality reduction.

The confusion matrix in ML is used to evaluate a classification model's precision.
The following lines of code would import and implement a confusion matrix, assuming that *y_pred* and *y_test* have been initialized previously. In the following Python example, y_test and y_pred are variables that represent the tested and predicted values outputted by the machine learning model.

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print (cm)
```

What Is Sensitivity (sn)

Sensitivity measures the proportion of actual positive cases predicted as positive (or true positive). Sensitivity is also termed Recall. This implies that another proportion of actual positive cases will get predicted incorrectly as negative (and, thus, could also be termed as the false negative). This can also be represented in the form of a false negative rate. The sum of sensitivity and false negative rate would be 1. Let's try to understand this with the model used to predict whether a person is suffering from the disease. Sensitivity is a measure of the proportion of people suffering from the disease who got predicted correctly as the ones suffering from the disease. In other words, the unhealthy person actually got predicted as unhealthy.

Mathematically, sensitivity can be calculated as the following:

Sensitivity = (True Positive)/(True Positive + False Negative)

What Is Specificity? (sp)

Specificity is defined as the proportion of actual negatives predicted as the negative (or true negative). This implies that another ratio of actual negative will be predicted as positive and could be termed false positives. This proportion could also be called a false positive rate. The sum of specificity and false positive rate would always be 1. Let's try to understand this with the model used to predict whether a person is suffering from the disease. Specificity is a measure of the proportion of people not suffering from the disease who got predicted correctly as the ones who are not suffering from the disease. In other words, the person who is healthy actually got predicted as healthy is specificity.

Mathematically, specificity can be calculated as the following:

Specificity = (True Negative)/(True Negative + False Positive)

Accuracy

Accuracy is one metric for evaluating classification models. Informally, **accuracy** is the fraction of predictions our model got right. Formally, accuracy has the following definition:

Accuracy=Number of correct predictions/Total number of predictions

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

Accuracy=(TP+TN) / (TP+TN+FP+FN)

Where *TP* = True Positives, *TN* = True Negatives, *FP* = False Positives, and *FN* = False Negatives.

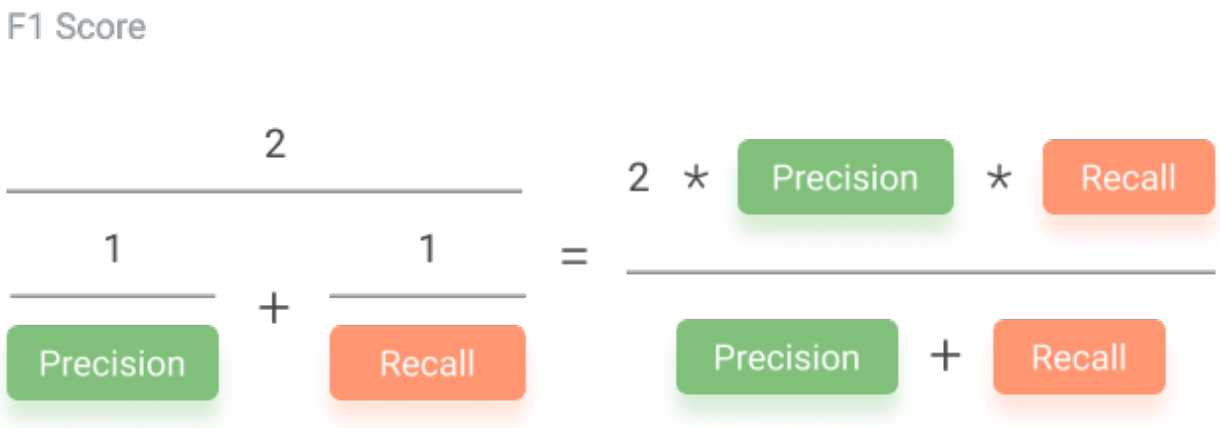
Specificity of Machine Learning Models

Specificity entails the percentage of negative instances out of the total actual negative instances. It is similar to the true positive rate method above. Here, the denominator is the sum of real numbers not generated by the model but instead verified by data. Because the numerator is the number of true negatives, this equation allows us to see how often the model is correct when it generates negative outputs based on the true negatives of the total negatives it produced in the past.

F1 Score to analyze Machine Learning Models

F1 score is when we take the mean of precision and recall. It takes the contribution of both of them, which means that the higher the F1 score, the more accurate the model. Conversely, if the product in the numerator dips down too low, the final F1 score decreases dramatically.

A model with a good F1 score has the most drastic ratio of “true: false” positives and the most drastic “true:false” negatives ratio. For example, if the number of true positives to the number of false positives is 100:1, that will play a role in producing a good F1 score. Meanwhile, having a close-ratio, say 50:51 true to false positives, will produce a low F1 score. The equation for the F1 score is below.



F1 Score is the weighted average of Precision and Recall.
An F1 score is considered perfect when it's 1, while the model is a total failure when it's 0. Thus, a low F1 score indicates both poor precision and poor recall.

Key Terms Onion, Decision Tree, Random Forest, K-Nearest Neighbor, Logistic Regression, Classification, Regression, Specificity.

Conclusion Onions comprise an essential component of the diet of Bangladeshi people. Their demand has increased considerably recently, and the growing demand has been met by domestic production. Yet, the onion prices have risen and become more volatile, shooting up periodically.

From this study, I have observed that classification models produced a better result than the regression model. It would be better if there were more attributes in the data set. But due to insufficient data, I had to use fewer attributes. If I look at the accuracy score for all the models, it is clear that the decision tree works best for this data set, producing 98% accuracy. After the decision tree, the Random forest model had good accuracy. So, I will recommend the decision tree model for this study.