Objective:

ZETA is a private organization in Toronto that updates users on next day weather conditions with the help of mobile application. We are trying to help them predict next day weather as accurately as possible by developing Machine Learning models. The historical weather data is provided by ZETA in Kaggle.

Description on Data:

The data is given by ZETA and we have temperatures of each day from 1937 to 2018. Although, records from early 1930's to 2012 have been missing most of the days in each year, we are still considering them. We have temperatures recorded for almost every day in each year from 2013 to 2018. Below are the number of records we have from 2013 to 2018. A total of 2890 records were used for training and testing

Year	Count
2013	208
2014	365
2015	365
2016	366
2017	365
2018	318

We have the following attributes for each day in the years mentioned above.

1. Date
2. Mean Temp (C)
3. Max Temp (C)
4. Min Temp (C)
5. Total Rain (mm)
6. Total Snow (mm)
7. Total Precipitation (mm)
8. Season

We are using the below features to train our model and forecast the **max temp** of future days

1. Mean Temp (C)
2. Min Temp (C)
3. Total Rain (mm)

Overview of Models:

Linear Regression: It is one of the common machine learning models that tries to get the relationship between the independent variables (mostly inputs) and dependent variable (the predicted or output) by plotting them on a straight line. Although we can have many independent variables, we can have only one dependent or output variable. The equation is called Simple Linear Regression when only one independent variable is used, if multiple independent variables are used, it is called Multiple Linear Regression. When multiple variables that are dependent on each other are used to predict a single variable, the process or model is called Multivariate Linear Regression.

The general equation for Linear Regression using single independent and dependent variable is as below

$$Y= mX+b$$

Here Y – dependent or predictable variable, m – slope of the linear line, X- independent variable used to predict Y, b- intercept of the line

The general equation for Multiple Linear Regression using multiple independent variables is as below

$$Y = m_1X_1+m_2X_2+m_3X_3+m_4X_4+....+m_nX_n + b$$

Here Y – dependent or predictable variable, X_1- independent variable used to predict Y, b- intercept of the line, m_1 – coefficient value that measures a unit change in the dependent variable when X_1 changes

Decision Tree: A decision tree is one of machine learning algorithm that uses a tree like structure to give the final output. A decision tree is usually drawn upside down with the root on the top, which branches into possible outcomes. Each of those outcomes leads to additional nodes, which branch off into other possibilities. This gives it a treelike shape.

There are three different types of nodes: chance nodes, decision nodes, and end nodes. A chance node, represented by a circle, shows the probabilities of certain results. A decision node, represented by a square, shows a decision to be made, and an end node shows the outcome of a decision path.

Decision Tree can be used for solving both Classification and Regression problem

Support Vector Regression: Support Vector Regression (SVR) is little different from Support Vector Machine (SVM) in terms of few variables they use for dividing the space for classification and regression. SVR is used for predicting continuous values whereas SVM is used for classification. The general idea is to separate various categories by a boundary and predict the outcome. If data is a 2D model, then the separator is called a line, if data is 3D the separator is called plane and if we have more features or n-D, then separator is called Hyperplane.

The points that are closer to the hyperplane are called "Support Vectors".

Not all data is suitable for SVM to solve. So, to make the data suitable for us to understand and classify, we transform the data from lower dimension to upper dimension and pick a classifier to solve the problem. This is done with the help of kernel.

Implementation and Evaluation Procedure:

1. **Linear Regression:** The dataset was observed carefully to find correlation and covariance between data so that the variables that have less or no impact on the output can be removed and to understand to what extent a variable can impact the output. In the process of correlation and covariance, the following variables were found to have very little or no impact on the

predictable variable. Below table shows the feature name and value of its impact on the predictable variable.

Table showing Correlation of variables:

Feature Name	**Impact Value**
Mean_Temp_C	0.991714
Min_Temp_C	0.989511
Total_Rain_mm	0.120251
Total_Precip_mm	0.002725
Season	-0.296395
Total_Snow_cm	-0.395967

Table showing Covariance of variables:

Feature Name	**Impact Value**
Mean_Temp_C	122.595897
Min_Temp_C	113.316722
Total_Rain_mm	47.631358
Total_Precip_mm	5.062400
Season	-3.775693
Total_Snow_cm	-44.769164

The variables highlighted were chosen as features after removing the variables that have negative values and close to zero.

Since we have 3 features that are dependent on each other and are impacting the output variable, our linear regression model is called **Multivariate Linear Regression.** So, the equation for this model becomes,

$$Y = m_1X_1 + m_2X_2+m_3X_3+b$$

Where

Y = Max Temp

m_1, m_2, m_3 are coefficients of X_1, X_2 and X_3 and they determine to what times the input or feature must be multiplied to predict the output.

X_1 = Mean_Temp_C

X_2= Min_Temp_C

X_3= Total_Rain_mm

The below model was implemented using Machine Learning Frameworks such as Scikit-learn, Python Frameworks such as NumPy and pandas.

A total of 2890 records is used here. At first, the data was cleaned to fill any missing values and replacing NULL with the column's mean value.

```
#%% read file into Data Frame
df = pd.read_csv('Toronto_temp.csv')

#%% Data Cleaning - replacing NaN with mean value of respective columns
df.fillna(df.mean(), inplace=True)
```

Later, the correlation and covariance of all the variables was calculated so that features (denoted above) can be shortlisted.

```
#%% Finding correlation between data to drop unncessary columns
corr_matrix = df.corr()


#Finding covariance between data to drop unncessary columns
cov_matrix = df.cov()


#%% deleting the unncessary columns and viewing the dataframe using 'describe'
df=df.drop(['Date/Time','season','Total_Snow_cm','Total_Precip_mm'],axis='col-
umns')
```

Later, the data is split into training and test data in 7:3 ratio respectively using the below code.

```
#%% Spliting data into Test and Train
from sklearn.model_selection import train_test_split
y=df.Max_Temp_C # value to be predicted is generally depicted by 'y'
X=df.drop(['Max_Temp_C'],axis='columns') # values or independent variables used
#to predict 'y' are denoted by 'X' and are called as 'features'
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,ran-
dom_state=0,shuffle=False)
```

Finally, an instance of Linear Regression object is created from scikit-learn library and our linear regression model was trained on the training data and tested on the test data as well

```
#%% Create an instance of LR class and run on training data
reg = linear_model.LinearRegression()
reg.fit(X_train,y_train)

# Getting the coefficients and intercept
Reg.coef_
reg.intercept_


#%% Predicting the output using test data
y_predict=reg.predict(X_test)
```

Since the model is trained on the data, we can now see the coefficients it gave us. Below is the revised model equation.

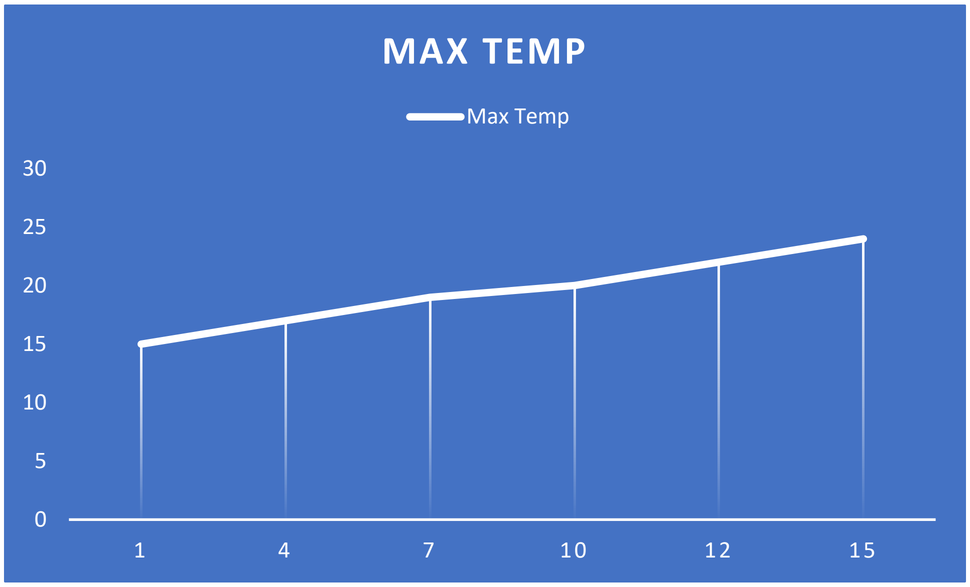
$$Y = (1.920)\ X_1 + (-0.919)\ X_2 + (0.000431)\ X_3 + 0.334$$

Since, we tested on test data as well, now let's try to understand whether our model was able to predict values close to our expected results or not. This is done by finding out Root Mean Square Error, Mean Square Error and Mean Absolute Error. Below are the results

```
#%% checking various errors
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predict))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_er-
ror(y_test, y_predict)))
```

```
Mean Absolute Error: 0.5172331958898808

Mean Squared Error: 0.48058272683048875

Root Mean Squared Error: 0.6932407423330575

# Note that for rmse, the lower that value is, the better the fit

R2 Score: 0.9964637420813469

# The closer towards 1, the better the fit
```

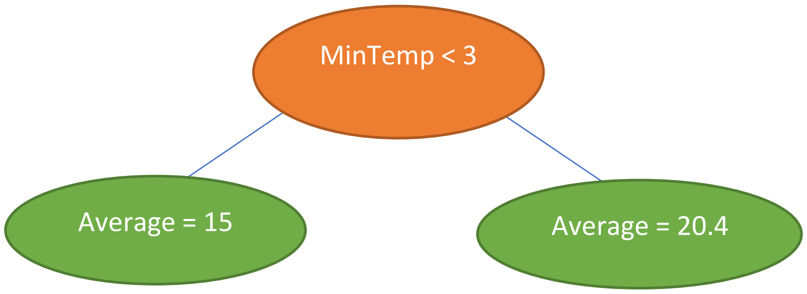
2. **Decision Tree:** When the data cannot be fitted with the help of straight line, using other machine learning algorithms to understand the data is very important. Although, we can see my previous observation that our data is very linear and can be solved accurately by Linear Regression, we are trying to understand to how far decision tree can solve this. These trees are mainly known as classifiers, but they can be used for solving continuous output (max_temp in our case). When solving such cases, these trees are called as Regression Trees. In regression Trees, each leaf (the last node) represents the numeric value (the output or decision). In



contrast, classification trees have “True” or “False” or some other discrete category in their leaf nodes.

Our data has multiple variables predicting the final output. A Decision tree has a root node and the branches travel downwards from the node. To understand how root, sub-branches and leaves are selected, let’s use only one variable that predicts one output. For example, let’s take Min Temp to predict Max Temp.

From the graph, let’s take some value from x-axis i.e. 3 to be our initial threshold value of the root node to make decisions based on the value. The root node checks if MinTemp < 3 or not and has two possible outcomes based on this. For the value of left outcome, we need to average all the actual outputs between 1 to 3. Similarly, we need to average all the outputs at each value of X for the right-side outcome. The tree should look something like below



From the graph, we have only one MaxTemp value for all “**MinTemp<3**” taken as threshold. So, the average on first left branch is **“15”** whereas we have many datapoints for **“MinTemp>=3”,** so, the average of all becomes **“20.4”.** Now, let’s test whether our initial version of regression tree works or not by taking MinTemp=15. The actual output should be **“24”** according to graph whereas our decision tree predicts as “20.4”. The difference in value between actual and predicted is called residual.

Residual = (Predicted value – Actual)

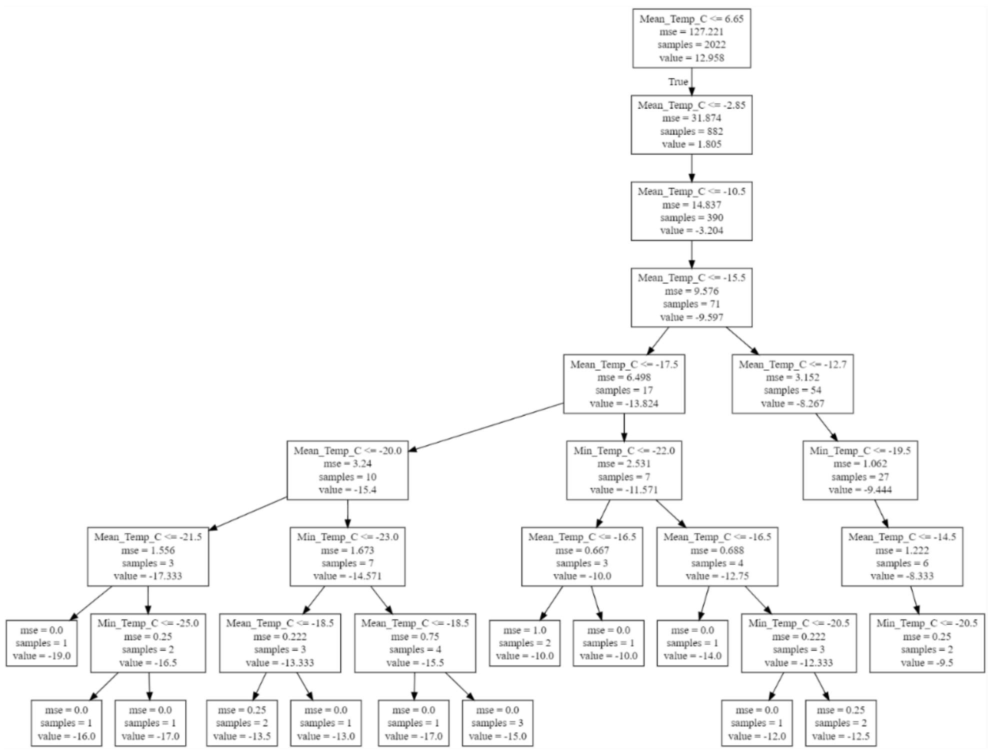
This residual effect our prediction tree very badly. To make sure our tree fits the data, we need to choose the roots and branches correctly. Initially, for the sake of understanding the tree, we chose a value from X-axis to be the threshold to calculate sub-branches and leaf values (output values). Ideally, all the inputs are chosen as roots one by one and leaves and sub-branches for the roots are calculated. Later, we need to calculate **Squared Residual** of each leaf at each root node and add them.

Squared Residual= (Predicted value – Actual)^2

Plot a graph with Threshold in x-axis and squared residual at y-axis. Select the point which has the minimal square residual, and this will be the root of our final decision tree. Now you have 2 sub-branches, follow the steps same as above, choosing a random threshold, finding the averages and square residual to decide the value of the subbranch root. If the datapoints under any subbranch are less than 20, then you can consider it as leaf node. In this way, you can figure out the root, subbranch and leaf of the tree.

The above explanation works for single input and single output but in our case, we have multiple inputs and one output. We can create the required tree by following the same old process on each input and deciding the roots and subbranch based on square residual at each input. In our case, when we take "MinTemp" as our first input, we calculate the root node condition by choosing the datapoint which has the minimum square residual value. Also, we sum all the square residual values for this root condition. Similar approach goes for each input. Finally, we pick the input variables that has minimum square residual sum as our tree root node value/condition. The first subbranch of this root node will be the input variable that has the second least square residual value. We follow similar steps of calculating the averages based on input to either create a new subbranch or end it as leaf on the other side of root node.

Below figure shows a part of our decision tree as we are dealing with huge data. Here Mean_Temp takes the root node followed by min_temp and finally followed by Total_Rain.



Mean_Temp_C <= 6.65 mse = 127.221 samples = 2022 value = 12.958
True
Mean_Temp_C <= -2.85 mse = 31.874 samples = 882 value = 1.805
Mean_Temp_C <= -10.5 mse = 14.837 samples = 390 value = -3.204
Mean_Temp_C <= -15.5 mse = 9.576 samples = 71 value = -9.597
Mean_Temp_C <= -17.5 mse = 6.498 samples = 17 value = -13.824
Mean_Temp_C <= -12.7 mse = 3.152 samples = 54 value = -8.267
Mean_Temp_C <= -20.0 mse = 3.24 samples = 10 value = -15.4
Min_Temp_C <= -22.0 mse = 2.531 samples = 7 value = -11.571
Min_Temp_C <= -19.5 mse = 1.062 samples = 27 value = -9.444
Mean_Temp_C <= -21.5 mse = 1.556 samples = 3 value = -17.333
Min_Temp_C <= -23.0 mse = 1.673 samples = 7 value = -14.571
Mean_Temp_C <= -16.5 mse = 0.667 samples = 3 value = -10.0
Mean_Temp_C <= -16.5 mse = 0.688 samples = 4 value = -12.75
Mean_Temp_C <= -14.5 mse = 1.222 samples = 6 value = -8.333
mse = 0.0 samples = 1 value = -19.0
Min_Temp_C <= -25.0 mse = 0.25 samples = 2 value = -16.5
Mean_Temp_C <= -18.5 mse = 0.222 samples = 3 value = -13.333
Mean_Temp_C <= -18.5 mse = 0.75 samples = 4 value = -15.5
mse = 1.0 samples = 2 value = -10.0
mse = 0.0 samples = 1 value = -10.0
mse = 0.0 samples = 1 value = -14.0
Min_Temp_C <= -20.5 mse = 0.222 samples = 3 value = -12.333
Min_Temp_C <= -20.5 mse = 0.25 samples = 2 value = -9.5
mse = 0.0 samples = 1 value = -16.0
mse = 0.0 samples = 1 value = -17.0
mse = 0.25 samples = 2 value = -13.5
mse = 0.0 samples = 1 value = -13.0
mse = 0.0 samples = 1 value = -17.0
mse = 0.0 samples = 3 value = -15.0
mse = 0.0 samples = 1 value = -12.0
mse = 0.25 samples = 2 value = -12.5

The code implementation is like Multivariate Linear Regression. The only difference is we need to create Decision Tree Regression object and pass our inputs to train the model.

```
#%% Create an instance of LR class and run on training data
reg =  DecisionTreeRegressor(random_state = 0)
reg.fit(X_train,y_train)

#%% Predicting the output using test data
y_predict=reg.predict(X_test)
```

Later, we calculate the errors as applicable for decision tree by using the below code

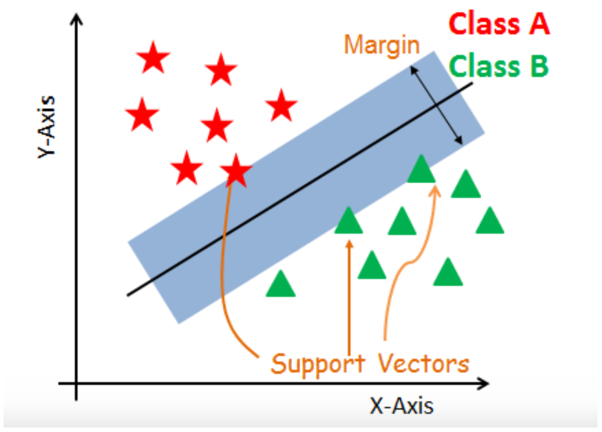
```
#%% checking various errors
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predict))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pr
edict)))
```

```
Mean Absolute Error: 0.7028571137611878

Mean Squared Error: 0.8768246439126394

Root Mean Squared Error: 0.9363891519622808
```

3. **Support Vector Regression:** Generally, SVM is considered to be a classification approach, but it can also be used for solving regression. In simple regression we try to minimize the error rate. While in SVR we try to fit the error within a certain threshold**.**

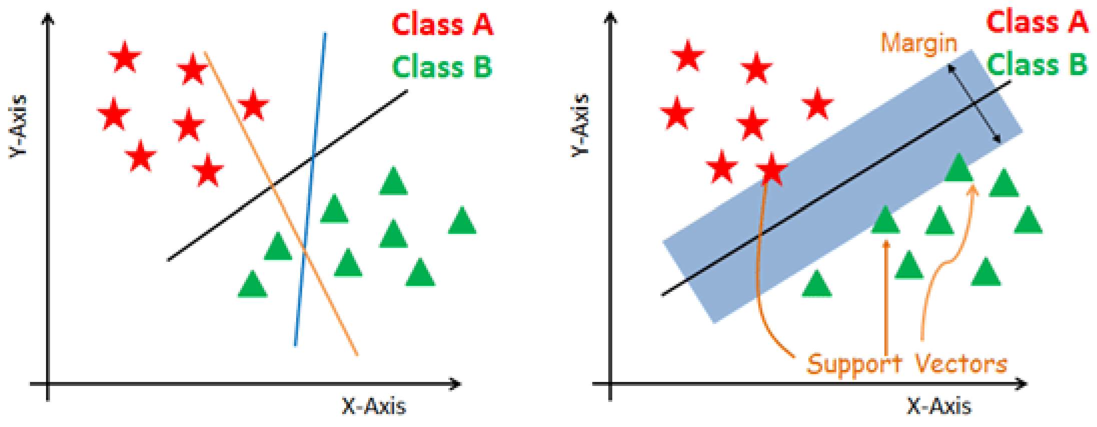
SVM constructs a hyperplane in n-dimensional space to differentiate or distinguish different categories or classes. SVM uses a trial and error approach to calculate and generate optimal hyperplane, which is used to reduce or minimize error. In the end, SVM gives us the best hyperplane possible to separate categories.

Figure below shows a hyperplane for 2-D data. The thick line inside the blue rectangle is our hyperplane



Lets redefine the terminologies in the picture above.

1. **Support Vectors**: These are the points that are closer to the hyperplane (the thick black line). These lines will help in determining how best we can separate the categories by choosing our separator plane.
2. **Hyperplane**: a hyperplane is a separation plane which separates a set of objects having different categories.
3. **Margin**: it is the distance between two lines that are parallel to our hyperplane. The parallel lines lie on the support vectors and are chosen in such a way that larger the margin, good the prediction would be. This is calculated as the perpendicular distance from the line to support vectors or closest points.
4. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. Let us understand the working of SVM with the help of below figure.



Every line of left-hand side of the figure separates the classes but orange and blue line don't separate them perfectly whereas black line separates accurately. This is what we are talking about, SVM does a trail and error method to figure out the hyperplane that separates objects.

There can be n number of lines parallel to our black line and can be considered as hyperplane but that won't do good for forecasting. The line which has the maximum distance from both sides of nearest points (support vectors) is selected to be the best line. Figure on the left shows us the separation and selection. Some problems can't be solved using linear hyperplane, so SVM provides a mechanism by which you can transform a data from one dimension to another to calculate the separator for that dimension. This feature is called kernel.

A kernel basically transforms your datapoints or equation from a lower dimension to higher dimension to draw a plane and boundaries between them to classify or find continuous values.

Importing and cleaning data remains same as done for Linear regression. The only change is creating SVR object with 'Linear' kernel. SVM has many kernels and each one is best used depending about the type of data. For example, RBF goes best with non-linear data, since our dataset is linear, we are going with linear kernel.

```
#%%
sv_reg = SVR(kernel='linear')
sv_reg.fit(X_train,y_train)

#%% Predicting the output using test data
y_predict=sv_reg.predict(X_test)
```

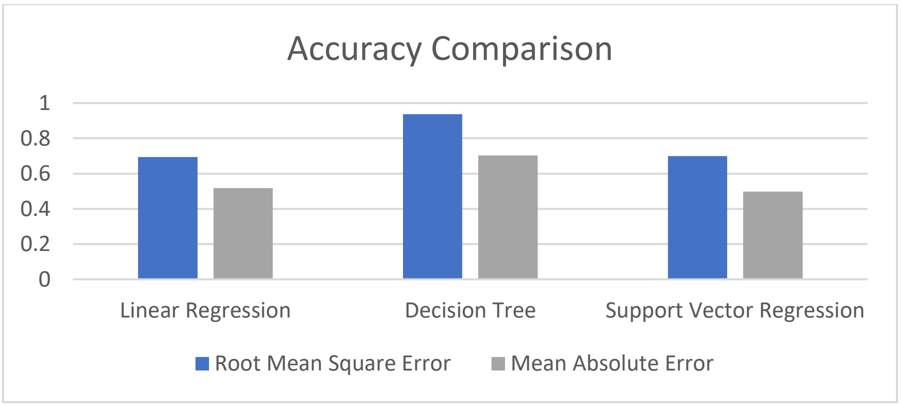
Later, we find different errors by running the following code

```
#%% checking various errors
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_predict))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pr
edict)))
from sklearn.metrics import r2_score
print('R2 Score:',r2_score(y_test,y_predict))
```

```
Mean Absolute Error: 0.4981945391801813

Mean Squared Error: 0.4885630698025133

Root Mean Squared Error: 0.6989728677155597

R2 Score: 0.9964050205554725
```

Conclusion:

Let's try to analyze Mean Absolute Error and Root Mean Square Error for all the three models to conclude which one suits our data the best.



We can see that Linear Regression model is the best fit as its RMSE is lower than others