The University of Chicago Booth School of Business

Machine Learning

Winter 2020

HW #3

Jiyoon Chung

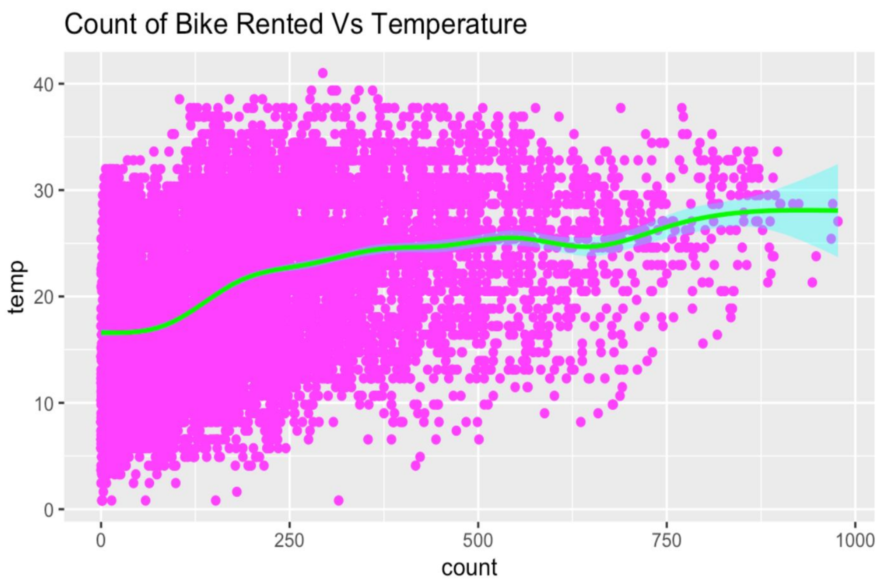
Arman Bhuiyan

Hikaru Sugimori

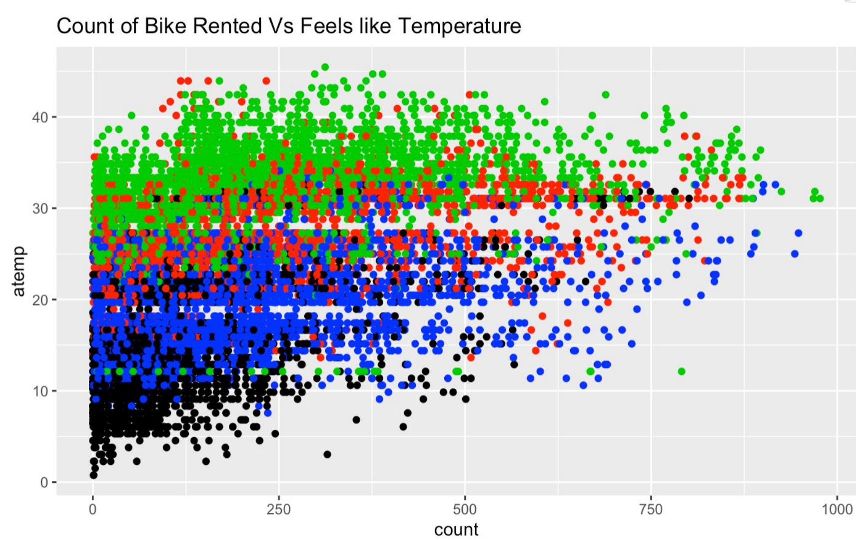
Deepak Putchakayala

In this homework assignment, we forecasted bike rental counts in tin Washington, D.C, using a dataset that contained both continuous and categorical data. Initially we used R to explore this data graphically to get a better feel for what the data contained and to recognize patterns. For your reference, the code used to arrive at our conclusions will be available at the end of the writeup.

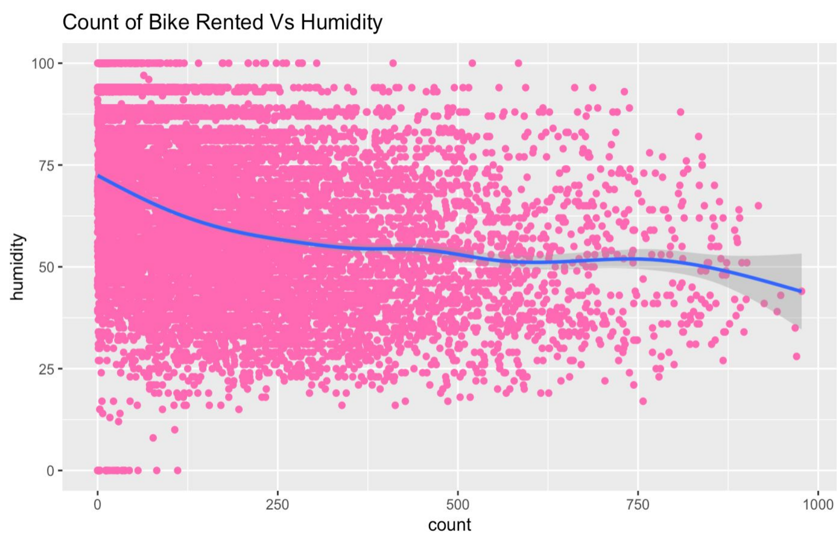
The visualization below shows that there is a positive relationship between temperature and bike rental counts, although the relationship is somewhat noisy.



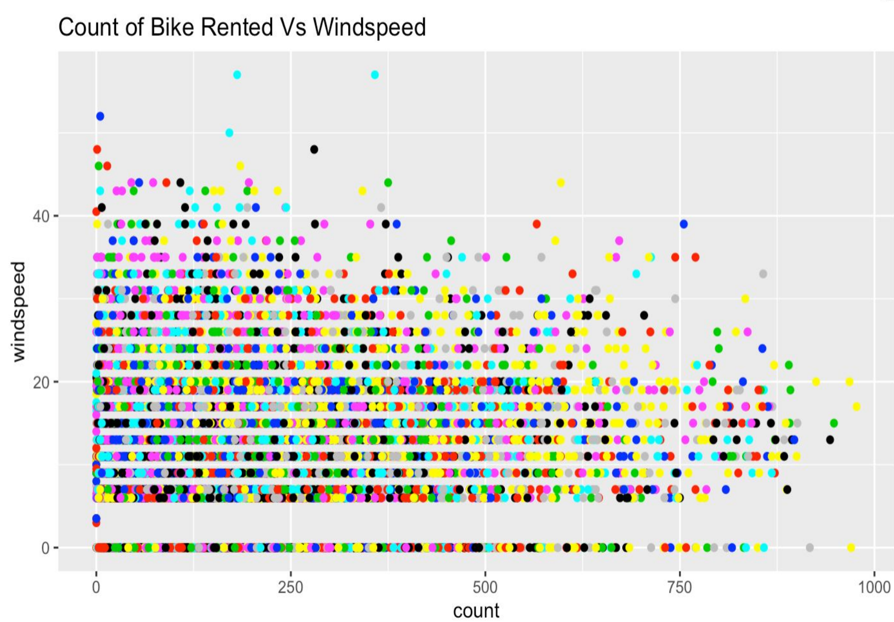
A similar relationship holds for feeling temperature as well (the data points are color coded based on the season).



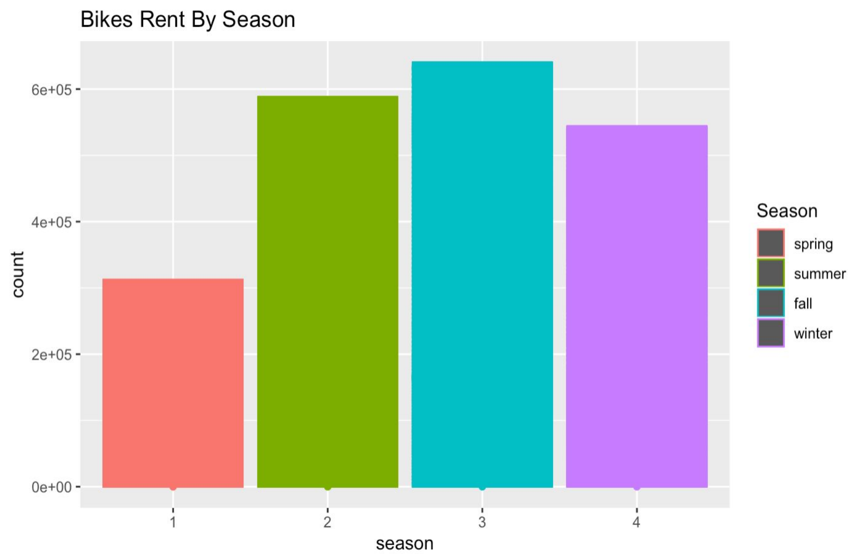
With regard to humidity, higher humidity levels are associated with lower bike rental counts.



Higher wind speeds are associated with lower bike rental counts as well.



The bar graph below shows bike rental counts by season. Bike rental counts trough in Spring and peak in fall.

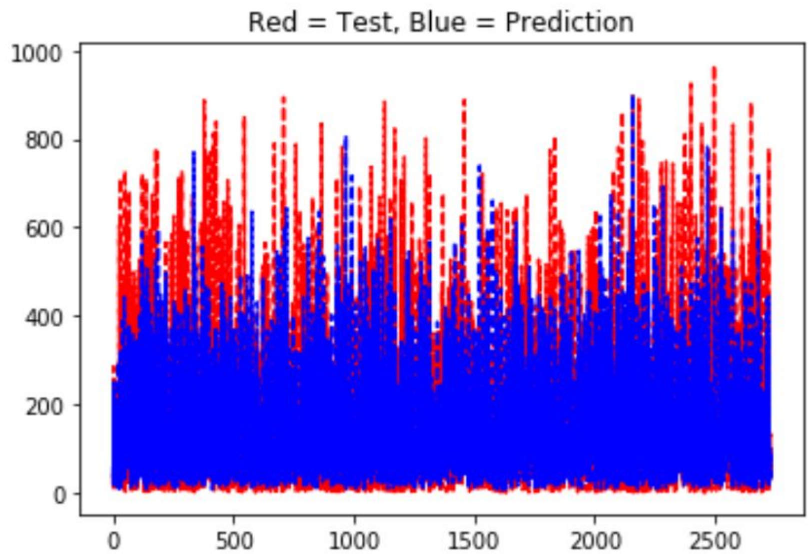


As you can see this is a relatively large data set with higher dimensionality. As such, we expected that decision trees or random forests would be better approaches to model this data than a multiple regression.

Before analyzing the data we cleaned it to make it more amenable to analytics. We removed unnecessary data such as the “daylabel” column, normalized the continuous features (temp, atemp, humidity, windspeed) so that our model does not get distorted by placing an excessive weight on those fringe data points, and we created a function that transformed rental counts into log(count+1) (we would find often times such a transformation enhanced the predictive power of our models).

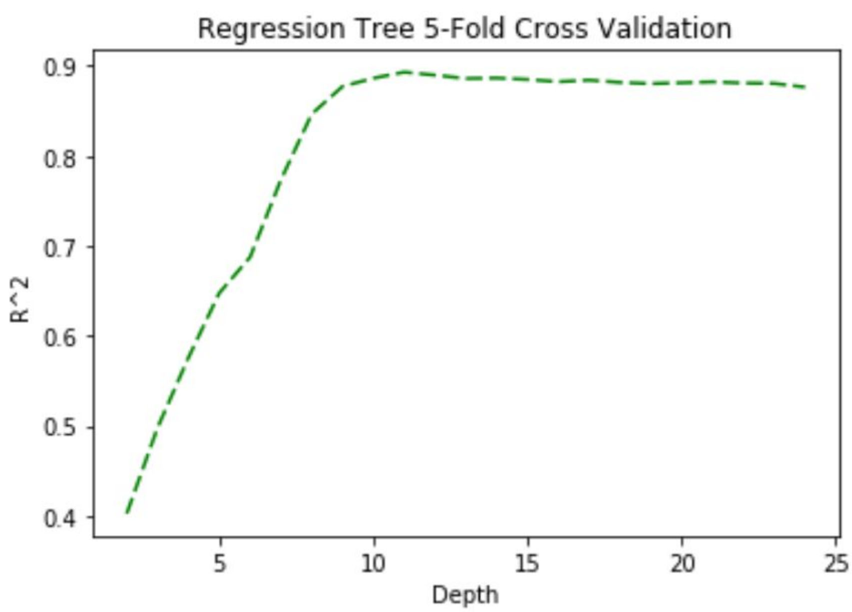
We first used a simple multiple regression to analyse our data. As suspected, the multiple regression predictions did not fit the test data very well and our R^2 was a lousy 49%.

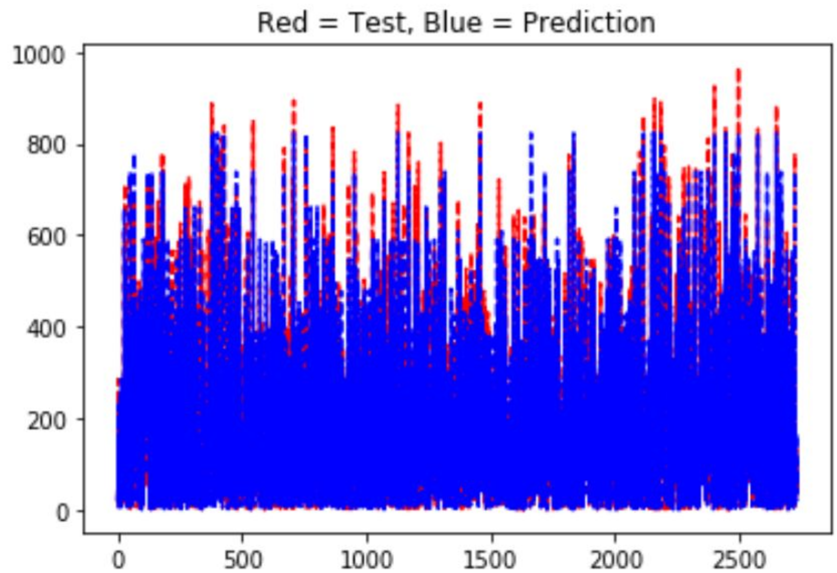
```
Performance of multiple linear regression
Mean Squared Error:  26080.679582
RMSE:                161.495138
R^2:                 0.492567
```



A decision tree model on the other hand got us a much better fit.

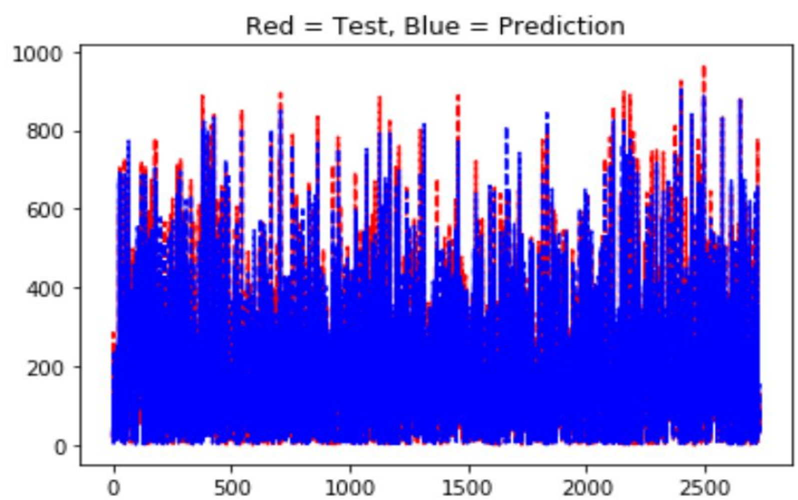
```
Optimal Depth:       11.000000
Mean Squared Error:  3310.265536
RMSE:                57.534907
R^2:                 0.901885
```





To try to improve the fit even further we created a random forest model that uses multiple decision trees. This resulted in the best fit as multiples trees makes our model much more robust.

```
Best n estimators:   35.000000
Best max features:   auto
Best max depth:      24.000000
Random Forest
Mean Squared Error:  2015.522215
RMSE:                44.894568
R^2:                 0.940261
```



R Code Used For Visualizations:

```
install.packages("corrplot")
install.packages("rpart.plot")
library(tidyverse)
library(ggplot2)
library(magrittr)
library(corrplot)
library(MASS)
library(rpart)
library(rpart.plot)
packages<-function(x){
  x<-as.character(match.call()[[2]])
  if (!require(x,character.only=TRUE)){
        install.packages(pkgs=x,repos="http://cran.r-project.org")
        require(x,character.only=TRUE)
  }
}

packages(caret)
packages(car)
packages(caTools)
packages(tree)
packages(ISLR)
packages(rpart)
packages(rpart.plot)
packages(randomForest)
packages(e1071)
packages(tidyverse)
packages(mlbench)

setwd("~/Desktop/ML")

Bike_train <- read.csv("/Users/Hikaru/Desktop/ML/Bike_train.csv")
Bike_test <- read.csv("/Users/Hikaru/Desktop/ML/Bike_test.csv")
dim(Bike_train)
Bike_train$season <- as.factor(Bike_train$season)
Bike_train$holiday <- as.factor(Bike_train$holiday)
Bike_train$workingday <- as.factor(Bike_train$workingday)
Bike_train$weather <- as.factor(Bike_train$weather)

Bike_train$daylabel <- as.Date.POSIXct(Bike_train$daylabel)
Bike_train$year <- as.Date.POSIXct(Bike_train$year)
```

```
Bike_train$month <- as.Date.POSIXct(Bike_train$month)
Bike_train$day <- as.Date.POSIXct(Bike_train$day)
Bike_train$hour <- as.Date.POSIXct(Bike_train$hour)

Bike_train <- Bike_train[,-1]
Bike_test <- Bike_test[,-1]

par(mfcol = c(2, 2))
ggplot(Bike_train, aes(x = count, y = temp)) +geom_point(,color = "magenta") + ggtitle("Count of Bike Rented Vs Temperature")+geom_smooth(color="green",fill = "cyan")

ggplot(Bike_train, aes(x = count, y = atemp)) +geom_point(,color = Bike_train$season)+ ggtitle("Count of Bike Rented Vs Feels like Temperature")

ggplot(Bike_train, aes(x = count, y = windspeed)) +geom_point(,color = Bike_train$atemp)+ ggtitle("Count of Bike Rented Vs Windspeed")

ggplot(Bike_train, aes(x = count, y = humidity)) +geom_point(color = "hotpink")+ ggtitle("Count of Bike Rented Vs Humidity")+geom_smooth()

season_summary_by_hour <- group_by(Bike_train,season, hour) %>% summarize(count = mean(count) )

ggplot(Bike_train, aes(x=season, y=count, color=season))+geom_point(data = season_summary_by_hour, aes(group = season))+geom_line(data = season_summary_by_hour, aes(group = season))+ggtitle("Bikes Rent By Season")+ scale_colour_hue('Season',breaks = levels(Bike_train$season), labels=c('spring', 'summer', 'fall', 'winter'))+geom_col()
```

R Code Used for Modeling:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import os

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor, RandomForestRegressor
```

```
os.chdir('/Users/hikaru/Desktop/ML')

#The functions below were used for cleaning/pre-processing of the data

CSV_FILE = "Bike_train.csv"
TESTING_SIZE = 0.25
SHUFFLE = True
NORMALIZE = True
CATEGORICAL = ['season', 'holiday', 'workingday', 'weather']
CONTINUOUS = ['temp', 'atemp', 'humidity', 'windspeed']

def load_data(csv=CSV_FILE):

    df = pd.read_csv(csv)
    # headers = list(df.columns)

    y_data = df['count']
    x_data = df.drop(columns=['count'])

    df_np = df.values
    y_np = y_data.values
    x_np = x_data.values

    return y_np, x_np, df


def split_data(y_np, x_np, testing_percent=TESTING_SIZE, shuffle_data=SHUFFLE):

    x_train, x_test, y_train, y_test = train_test_split(x_np, y_np,
                                    test_size=testing_percent,
                                    random_state=20,
                                    shuffle=shuffle_data)

    return x_train, x_test, y_train, y_test

def transform_count(y_np, direction):

    if direction == "forward":
    # transform into log(count + 1)
    y_np = y_np + 1
    y_np = np.log(y_np)
```

```
        return y_np

        elif direction == "backward":

        y_np = np.exp(y_np)
        y_np = y_np - 1
        return y_np

        else:
        print("ERROR")

def visualize_train_data():

        pass

def clean_data(df, normalize=NORMALIZE):

        if normalize:
        for feature in CONTINUOUS:
        df[feature] = (df[feature] - df[feature].mean()) / \
                        (df[feature].max() - df[feature].min())

        y_data = df['count']
        x_data = df.drop(columns=['count'])
        x_data = x_data.drop(columns=['daylabel'])

        df_np = df.values
        y_np = y_data.values
        x_np = x_data.values

        return y_np, x_np, df

#Linear Regression
y_np, x_np, df = load_data()
x_train, x_test, y_train, y_test = split_data(y_np, x_np)

lr_model = LinearRegression()
y_train = transform_count(y_train, "forward")
lr_model.fit(x_train, y_train)
y_predict = lr_model.predict(x_test)
y_predict = transform_count(y_predict, "backward")
mse = mean_squared_error(y_predict, y_test)
r2 = lr_model.score(x_test, transform_count(y_test, "forward"))
```

```
print("Performance of multiple linear regression")
print("Mean Squared Error:  %f" % mse)
print("RMSE:              %f" % (mse ** 0.5))
print("R^2:               %f" % r2)

plt.plot(x, y_test, color='red', linestyle='--')
plt.plot(y_predict, color='blue', linestyle='--')
plt.title("Red = Test, Blue = Prediction")
plt.show()
np.savetxt("hikarusugimori.csv", y_predict, delimiter=",")

#Regression Tree
y_np, x_np, df = load_data()
x_train, x_test, y_train, y_test = split_data(y_np, x_np)
kFold = 5
depth = np.arange(2, 25)
param_grid = {'max_depth': depth}
tree_grid = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=kFold)

y_np_c, x_np_c, df_c = clean_data(df)
x_train, x_test, y_train, y_test = split_data(y_np_c, x_np_c)

tree_grid = GridSearchCV(DecisionTreeRegressor(), param_grid, cv=kFold)
tree_grid.fit(x_train, y_train)

tree_best = tree_grid.best_params_['max_depth']
tr_model = DecisionTreeRegressor(max_depth=tree_best)
tr_model.fit(x_train, y_train)
tree_scores = tree_grid.cv_results_['mean_test_score']

y_predict = tr_model.predict(x_test)
mse = mean_squared_error(y_predict, y_test)
r2 = tr_model.score(x_test, y_test)

print("Optimal Depth:            %f" % tree_best)
print("Mean Squared Error:  %f" % mse)
print("RMSE:              %f" % (mse ** 0.5))
print("R^2:               %f" % r2)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(depth, tree_scores, color='green', linestyle='--', dashes=(5, 2))
```

```
ax.set_xlabel('Depth')
ax.set_ylabel('R^2')

plt.title("Regression Tree 5-Fold Cross Validation")
plt.show()

x = np.arange(len(y_predict))
plt.plot(x, y_test, color='red', linestyle='--')
plt.plot(y_predict, color='blue', linestyle='--')
plt.title("Red = Test, Blue = Prediction")
plt.show()

np.savetxt("hikarusugimori.csv", y_predict, delimiter=",")

# Random Forest
kFold = 5
param_grid = {'n_estimators': np.arange(5, 40, 5),
              'max_features': np.array(['auto', 'sqrt', 'log2']),
              'max_depth': np.arange(2, 30)}
forest_grid = GridSearchCV(RandomForestRegressor(), param_grid, cv=kFold)

x_np, y_np, df = load_data()
y_np_c, x_np_c, df_c = clean_data(df)
x_train, x_test, y_train, y_test = split_data(y_np_c, x_np_c)
forest_grid.fit(x_train, y_train)
best_n = forest_grid.best_params_['n_estimators']
best_f = forest_grid.best_params_['max_features']
best_d = forest_grid.best_params_['max_depth']

print("Best n estimators:   %f" % best_n)
print("Best max features:   %s" % best_f)
print("Best max depth:           %f" % best_d)

forest_model = RandomForestRegressor(n_estimators=best_n,
                                     max_features=best_f,
                                     max_depth=best_d)
forest_model.fit(x_train, y_train)

y_predict = forest_model.predict(x_test)
mse = mean_squared_error(y_predict, y_test)
r2 = forest_model.score(x_test, y_test)

print("Random Forest")
```

```
print("Mean Squared Error:  %f" % mse)
print("RMSE:                %f" % (mse ** 0.5))
print("R^2:                 %f" % r2)

plt.plot(x, y_test, color='red', linestyle='--')
plt.plot(y_predict, color='blue', linestyle='--')
plt.title("Red = Test, Blue = Prediction")
plt.show()
```