

Table of Contents

Dataset description.....	2
Section 0: Setting up to begin	2
0.1: Install and import packages	2
0.2 Create an account for MongoDB Create free MongoDB account	3
0.3 Create a new database by clicking the button below.....	3
0.4 Get to the network access tab and click on Add IP address and provide IP – 0.0.0.0/0	3
0.5 To access the MongoDB from Python, we would require the driver	4
0.6 Accessing the MongoDB from the Python	6
Section 1: Reading the excel file: Train and test files, Removing the rows where weekly sales < 0	6
Section 2: Cleaning the data	6
2.1: Number of rows and columns in each dataset	6
2.2: Columns of the dataset	7
2.3: What are the datatypes that we have in our data?	7
2.4: How many missing values in each column?	7
2.5: Boxplot to detect outliers for each column	8
2.6: Handling the boxplots for Unemployment	9
2.7: Correlation plot	9
2.8: Removing the Markdown4 variable	10
2.9 date, month, year, Quarters, Is it march month - tax month or not?.....	11
Section 3: Columns relationship and dummy variable handling	11
3.1: Finding relationship between the columns and weekly sales	11
3.2: Weekly sales - Date and store	11
Section 4: Creating Dummy variables and splitting to train-validation data.....	13
4.1 Creating dummy variables from train and test data	13
4.2 Drop one of the dummy variables to avoid perfect collinearity	13
4.3: x and y variable split.....	14
4.4: Preparing Min-max scaling data	14
4.5: Change y variable to log value	14
4.6: Finding out the skewed variables	14
4.7: Skewed variables are 'Markdown1', 'Markdown2', 'Markdown3', 'Markdown5' - Let us apply log transformation on them.....	14
4.8: PCA Data for x base scaled	15
4.9: PCA Data for x base scaled log	15
Section 5: Defining dataset model functions.....	15
5.1: Linear Regression	15
5.2: KNN Regression - Finding neighbors.....	16
5.3: Best KNN Regressor	16
5.4: Ridge elastic nets model.....	17
5.5 Bagging	18
5.6 Boosting.....	18
Summary: Section 6 to Section 15.....	19
Section 16: Finding out the best algorithm and dataset	20
16.1: Retrieve data from MongoDB	20
16.2: Bar Chart for Time taken to run	20
16.3: Bar Chart for final R ²	21
16.4: Best Algorithm.....	21
Section 17: Feature Selection	22
Section 18: Bagging Regressor	22
Section 19: Boosting Regressor	22

Dataset description

The dataset contains Walmart sales details of 45 Walmart stores located in different regions in USA – region and store wise. The train.csv and test.csv are the 2 excel files available - <https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/data> . train.csv will be used for building the model and the built model will be used for predicting sales given in test.csv.

Section 0: Setting up to begin

0.1: Install and import packages

The packages unavailable were installed and others imported.

0.1 Install and import packages

```
In [36]: ► #!pip install sklearn
          #!pip install statsmodels
          #!pip install patsy
          #!pip install mplcursors
          #!pip install dnspython==2.0.0
          #!pip install holidays
          #!pip install pymongo
          #pip install mongo[srv] dnspython
          #!pip install xgboost
```

```
In [3]: ► import pandas as pd
import seaborn as sbn
import datetime
import holidays
import matplotlib.pyplot as plt
import numpy as np
import math
import statistics
import pymongo
import time
import pickle
import xgboost as xgb
```

```
from sklearn.svm import SVR
from sklearn.ensemble import BaggingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn import metrics
from sklearn.metrics import r2_score
from itertools import combinations
from statsmodels import api as sm
from sklearn.neural_network import MLPRegressor
from datetime import date, timedelta
from statsmodels.api import OLS
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from statsmodels.formula.api import ols
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from datetime import datetime, timedelta
from scipy.stats import skew
from sklearn.decomposition import PCA
from sklearn.decomposition import FastICA
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectFromModel
```

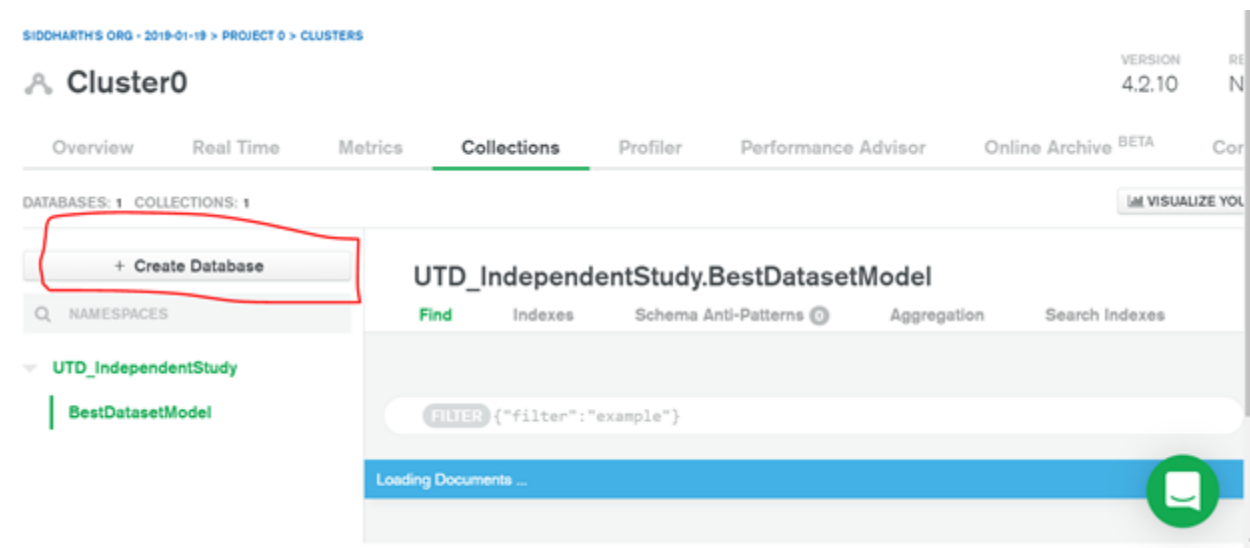
0.2 Create an account for MongoDB Create free MongoDB account

The model results will be stored in MongoDB and so a free MongoDB account is created at this location:

<https://cloud.mongodb.com/>

0.3 Create a new database by clicking the button below

After logging in, the database to store the collections is to be created. The database is created using the screenshot below.



0.4 Get to the network access tab and click on Add IP address and provide IP – 0.0.0.0/0

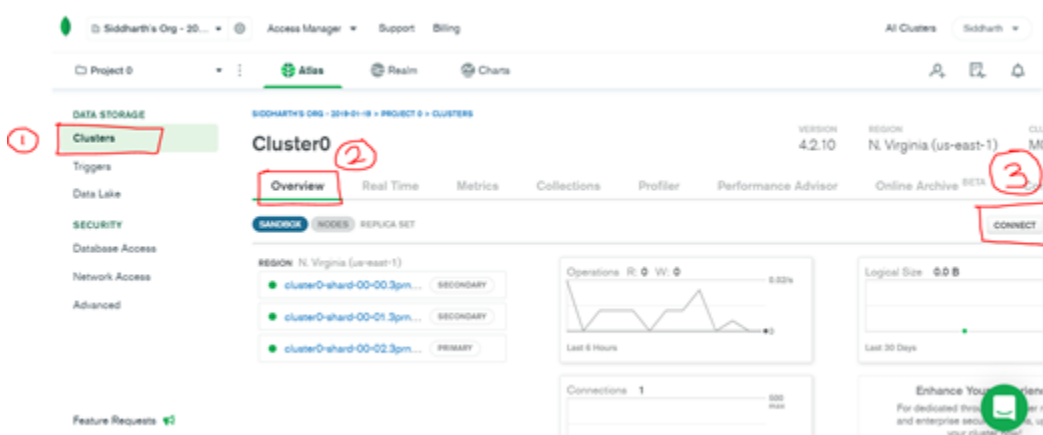
To provide access for the python to access the MongoDB, the IP address is added as mentioned in the screenshot below.



0.5 To access the mongoDB from Python, we would require the driver

- Step 1: Click the cluster tab
- Step 2: Get to the overview section
- Step 3: Click on the connect button located on RHS.

(3 steps shown in the screenshot below)




- Step 4: To connect MONGODB application with python, the driver URL will be obtained after clicking “Connect your application”


Connect to Cluster0


✓ Setup connection security Choose a connection method Connect

Choose a connection method [View documentation](#)

Get your pre-formatted connection string by selecting your tool below:

**Connect with the mongo shell**
Interact with your cluster using MongoDB's interactive Javascript interface

**Connect your application**
Connect your application to your cluster using MongoDB's native drivers

**Connect using MongoDB Compass**
Explore, modify, and visualize your data with MongoDB's GUI

Go Back Close

- Step 5: The driver programming language and the version is selected.

Connect to Cluster0

✓ Setup connection security

✓ Choose a connection method

Connect

1

Select your driver and version

DRIVER

Python

Node.js

Perl

PHP

Python

Ruby

Scala

VERSION

3.11 or later

2

Copy into your application code

Sample

`mongodb://<password>@cluster0.3prnb.mongodb.net/<dbname>`

Copy

Replace `<password>` with the password for the `siddharth` user. Replace `<dbname>` with the name of the database that connections will use by default. Ensure any option params are [URL encoded](#).

0.6 Accessing the MongoDB from the Python

The URL is copied from the above step and connected with MongoDB. The database name is “UTD_IndependentStudy” and the table/collection name is “BestDatasetModel”

```
In [44]: myclient = pymongo.MongoClient("mongodbsrv://siddharth:siddharth@cluster0.3prnb.mongodb.net/UTD_IndependentStudy?"\
    "retryWrites=true&w=majority")
    db = myclient["UTD_IndependentStudy"]
    collection = db["BestDatasetModel"]
```

Section 1: Reading the excel file: Train and test files, Removing the rows where weekly sales<0

The train and the test data csv files are read using pandas pd, stored in train_data and test_data respectively and rows where weekly sales < 0 are dropped.

```
In [5]: train_data = pd.read_csv("train.csv")
    test_data = pd.read_csv("test.csv")

    train_data = train_data[train_data['Weekly_Sales']>0]
    train_data = train_data.reset_index(drop=True)
```

Section 2: Cleaning the data

2.1: Number of rows and columns in each dataset

- The training set has 281,551 rows and 16 columns
- The test set has 139,119 rows and 15 columns

```
In [6]: ▶ def dataShape(data, nameofDataset):  
          print("Shape of ", nameofDataset, " is", data.shape)  
          dataShape(train_data, "train")  
          dataShape(test_data, "test")  
  
          Shape of train is (281551, 16)  
          Shape of test is (139119, 15)
```

2.2: Columns of the dataset

(i) What are the column names in training and test data?

The columns in the train and test data is printed below.

```
In [7]: ▶ def ColumnNames(data, nameofDataset):  
          print("Columns in ", nameofDataset, " dataset is", data.columns)  
  
          ColumnNames(train_data, "train")  
          ColumnNames(test_data, "test")  
  
          Columns in train dataset is Index(['Store', 'Dept', 'Date', 'Weekly_Sales', 'IsHoliday', 'Temperature',  
          'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4',  
          'MarkDown5', 'CPI', 'Unemployment', 'Type', 'Size'],  
          dtype='object')  
          Columns in test dataset is Index(['Store', 'Dept', 'Date', 'IsHoliday', 'Temperature', 'Fuel_Price',  
          'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4', 'MarkDown5', 'CPI',  
          'Unemployment', 'Type', 'Size'],  
          dtype='object')
```

(ii) What's that one column that is present in training data but not in test data?

The column weekly_sales is missing in training data but not in test data which is the variable that we are trying to predict.

2.3: What are the datatypes that we have in our data?

```
In [8]: ▶ train_data.dtypes  
  
Out[8]: Store          int64  
         Dept          int64  
         Date          object  
         Weekly_Sales  float64  
         IsHoliday     bool  
         Temperature   float64  
         Fuel_Price    float64  
         MarkDown1     float64  
         MarkDown2     float64  
         MarkDown3     float64  
         MarkDown4     float64  
         MarkDown5     float64  
         CPI           float64  
         Unemployment  float64  
         Type          object  
         Size          int64  
         dtype: object
```

2.4: How many missing values in each column?

BUAN 6V99.002 - Special Topics in Business Analytics - F20
Siddharth Govindarajan – SXG180066

```
def handleMissingValue(data,nameofDataset):
    print("Percentage missing values (Before imputing) in ",nameofDataset," dataset:",(data.isna().sum() / len(data)) * 100)
    data = data.fillna(0)
    print("Percentage missing values (After imputing) :", (data.isna().sum() / len(data)) * 100)
    return data

train_data = handleMissingValue(train_data,"train")
test_data = handleMissingValue(test_data,"test")
```

	Train data (Percentage)		Test data (Percentage)	
	Before imputation	After imputation	Before imputation	After imputation
Store	0	0	0	0
Dept	0	0	0	0
Date	0	0	0	0
Weekly_Sales	0	0		
IsHoliday	0	0	0	0
Temperature	0	0	0	0
Fuel_Price	0	0	0	0
MarkDown1	64.419945	0	63.94382	0
MarkDown2	73.717373	0	73.39256	0
MarkDown3	67.601607	0	67.24387	0
MarkDown4	68.13366	0	67.6996	0
MarkDown5	64.239871	0	63.76987	0
CPI	0	0	0	0
Unemployment	0	0	0	0
Type	0	0	0	0
Size	0	0	0	0

2.5: Boxplot to detect outliers for each columns

```
def detectOutliers(data,nameofDataset):
    print("Outliers in ",nameofDataset," dataset:")
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    print(((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).sum())
    print("=====")

detectOutliers(train_data,"train")
detectOutliers(test_data,"test")
```

Number of Outliers	Train dataset	Test dataset
CPI	0	0
Date	0	0
Dept	0	0
Fuel_Price	0	0

<i>IsHoliday</i>	19744	9842
<i>MarkDown1</i>	38054	17812
<i>MarkDown2</i>	68847	33473
<i>MarkDown3</i>	56763	28115
<i>MarkDown4</i>	52824	25992
<i>MarkDown5</i>	27297	12659
<i>Size</i>	0	0
<i>Store</i>	0	0
<i>Temperature</i>	50	17
<i>Type</i>	0	0
<i>Unemployment</i>	21658	10412

2.6: Handling the boxplots for Unemployment

```
def handling_outliers(data):
    columns_with_outliers = ["Temperature", "Unemployment"]
    for i in columns_with_outliers:
        boxplot_unemployment = data[[i]]
        boxplot_unemployment.boxplot()
        Q1 = boxplot_unemployment.quantile(0.25)
        Q3 = boxplot_unemployment.quantile(0.75)
        IQR = Q3 - Q1
        upper_hinge_unemployment = (Q3 + 1.5 * IQR)[0]
        lower_hinge_unemployment = (Q1 - 1.5 * IQR)[0]

        for j in range(0, len(data)):
            currentValue = data[i][j]
            if (currentValue >= lower_hinge_unemployment and currentValue <= upper_hinge_unemployment):
                data[i][j] = currentValue
            elif (abs(upper_hinge_unemployment - currentValue) <= abs(currentValue - lower_hinge_unemployment)):
                data[i][j] = upper_hinge_unemployment
            else:
                data[i][j] = lower_hinge_unemployment

    return data

train_data = handling_outliers(train_data)
test_data = handling_outliers(test_data)
```

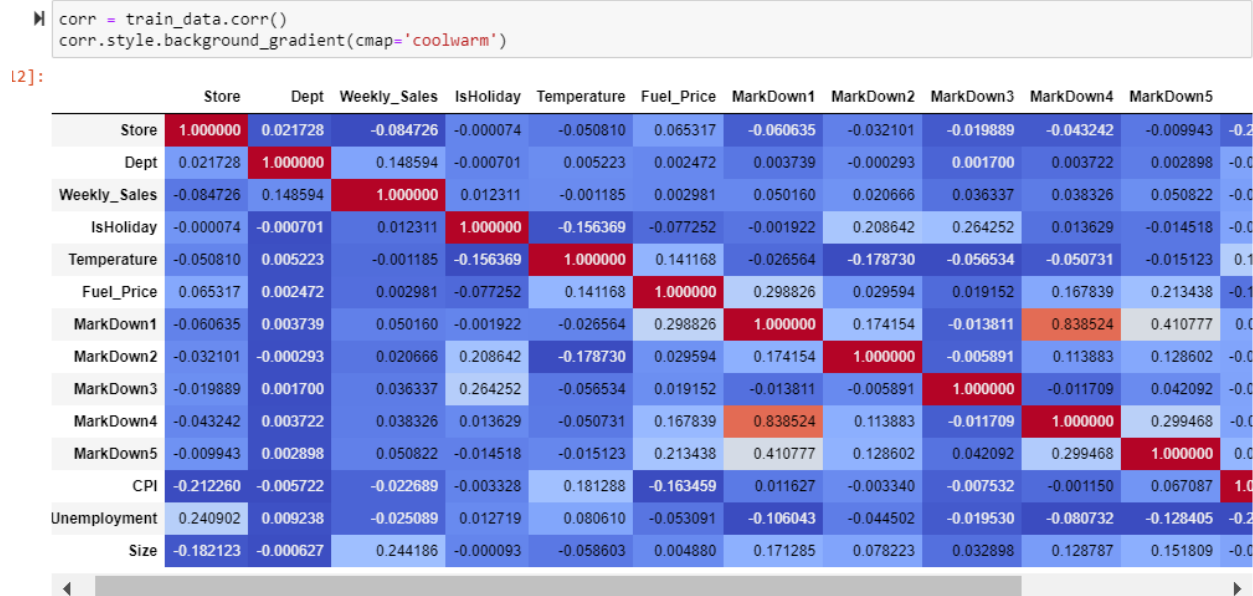
The outliers beyond the interquartile range are substituted with upper and lower hinge values.

2.7: Correlation plot

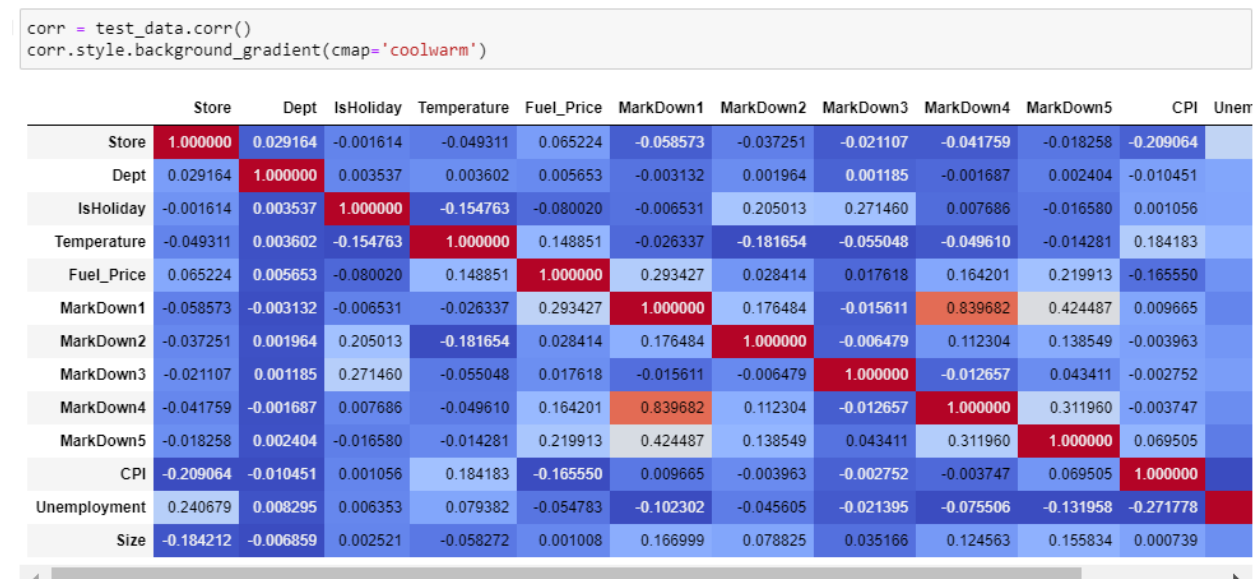
Train data correlation plot

BUAN 6V99.002 - Special Topics in Business Analytics - F20
Siddharth Govindarajan – SXG180066

2.7 : Correlation plot



Test data correlation plot



In both, train and test dataset, there is high correlation between the Markdown 1 and markdown 4 columns and one of them is removed.

2.8: Removing the Mardown4 variable

The markdown 4 column is removed from train and test dataset.

```
def removingOutlierColumn(data):  
    return data.drop(columns=['MarkDown4'])  
train_data = removingOutlierColumn(train_data)  
test_data = removingOutlierColumn(test_data)
```

2.9 date, month, year, Quarters, Is it march month - tax month or not?

Based on the date column, 5 more columns are created.

- Date
- Month
- Year
- Quarter
- Tax season or not

```
def dismantle_date_column(data):  
    data["Date"] = pd.to_datetime(data["Date"])  
    data = data.sort_values(by=['Store', 'Dept', 'Date'])  
    data["DateVal"] = data["Date"].dt.day  
    data["MonthVal"] = data["Date"].dt.month  
    data["YearVal"] = data["Date"].dt.year  
    data["quarterVal"] = data["Date"].dt.quarter  
    data["taxSeason"] = 0  
    totalRows = len(data)  
  
    for i in range(0, totalRows):  
        if (data["MonthVal"][i] <= 3) or (data["MonthVal"][i] == 4 and data["DateVal"][i] <= 15):  
            data["taxSeason"][i] = 1  
        else:  
            data["taxSeason"][i] = 0  
    return data  
  
train_data = dismantle_date_column(train_data)  
test_data = dismantle_date_column(test_data)
```

Section 3: Columns relationship and dummy variable handling

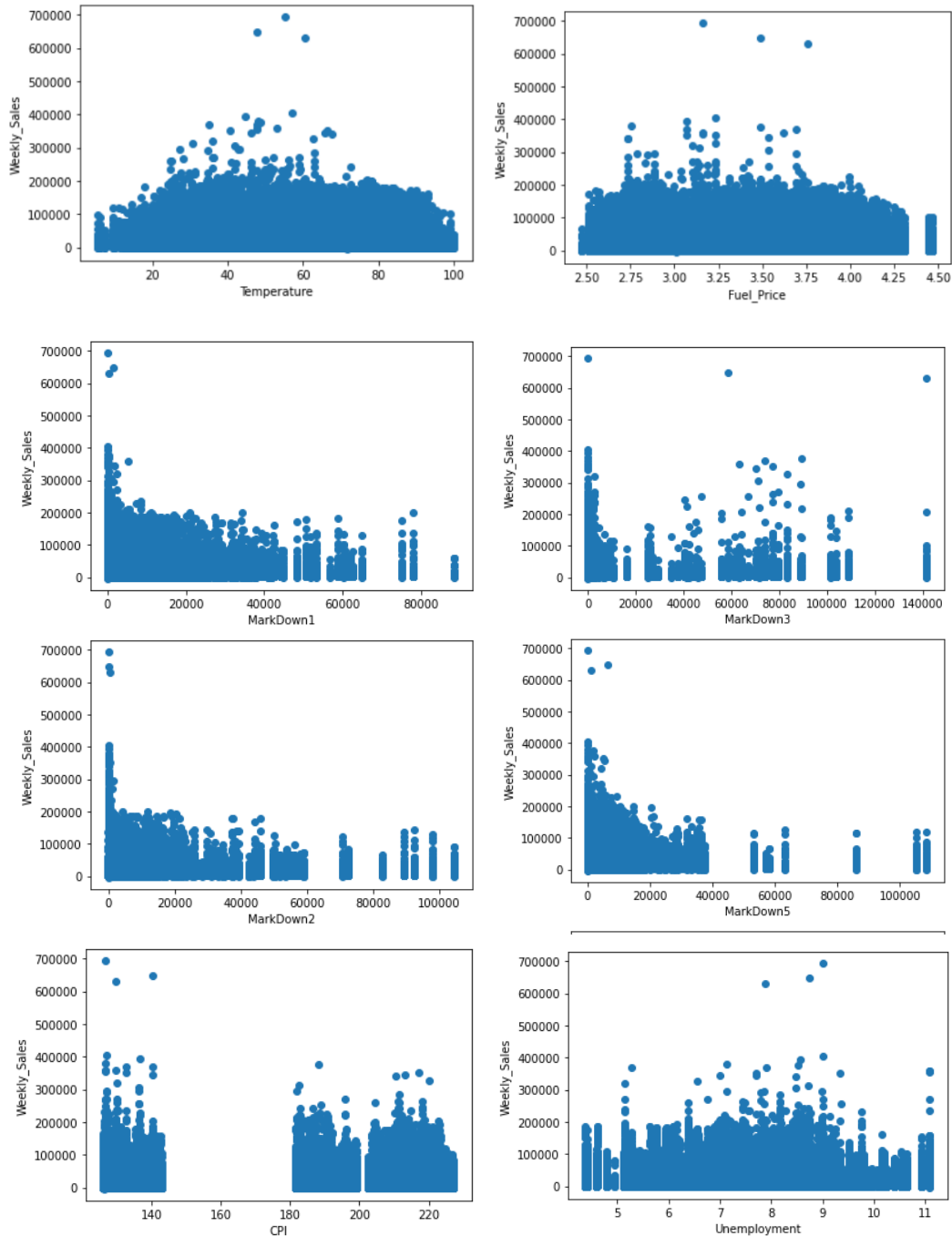
3.1: Finding relationship between the columns and weekly sales

3.2: Weekly sales - Date and store

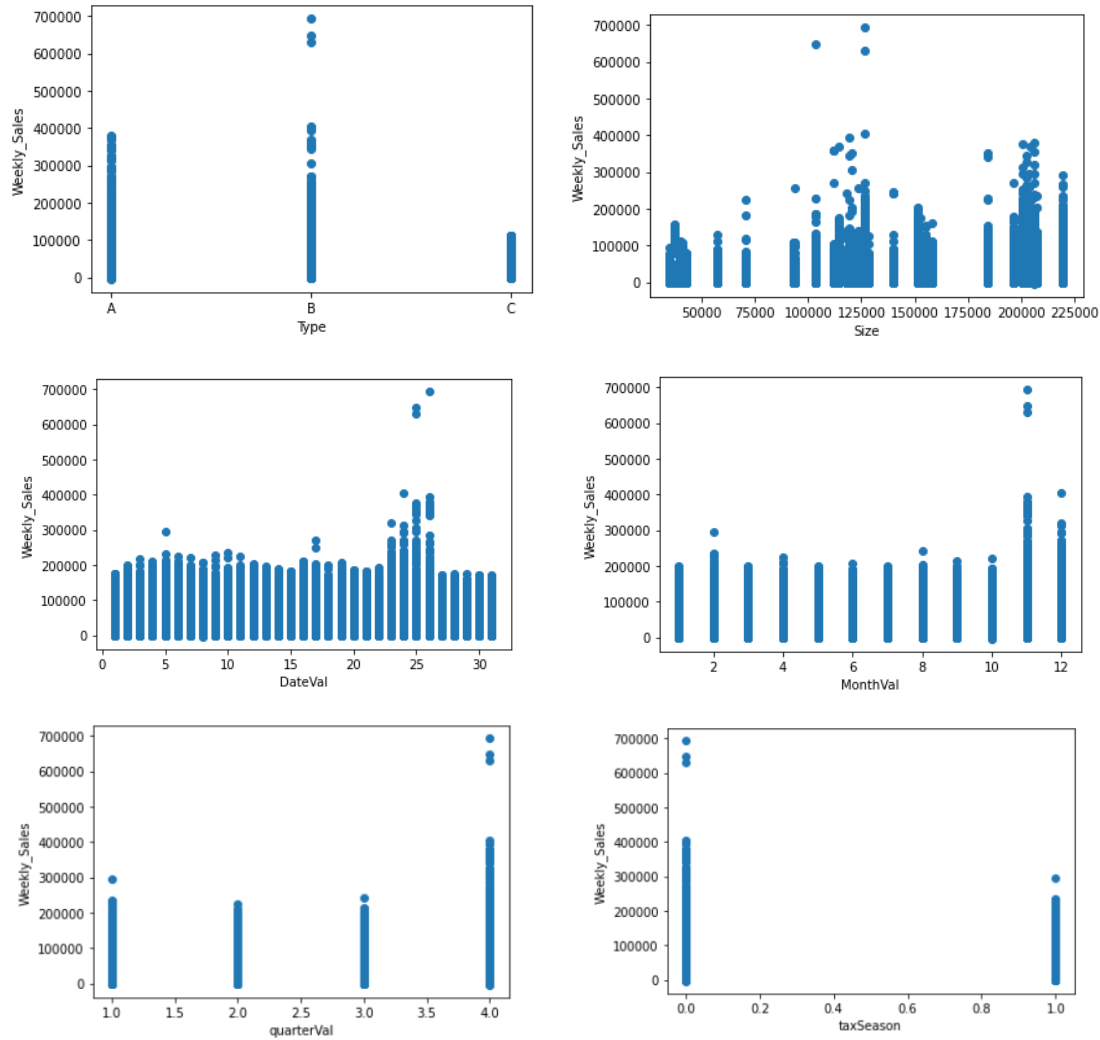
- The Department 72 from store 10 had the largest weekly sales. It happened on the date 26th November 2010.
- Largest weekly sales happened during holidays.
- The top 5 weekly sales happened when the average temperature of the day lie in between 40F and 60F
- The highest sales happened when the fuel price was between \$3 and \$3.75
- The weekly sales were high when there were no markdowns. (Markdown 1, Markdown 2, Markdown 3, Markdown 5)
- The weekly sales were the highest when the CPI was 0
- Highest sales occurred when the unemployment rate lied in between 8 and 9.
- Amongst Type A, Type B and Type C; Type B had the highest weekly sales.
- The highest weekly sales were when the store size lied in between 100,000 and 125,000.

BUAN 6V99.002 - Special Topics in Business Analytics - F20
Siddharth Govindarajan – SXG180066

- The highest weekly sales occurred between 23rd and 25th of a month
- The highest weekly sales occurred between 11th and 12th month of a year
- 4th Quarter had the highest weekly sales
- The highest weekly sales occurred on non-tax seasons.



BUAN 6V99.002 - Special Topics in Business Analytics - F20
Siddharth Govindarajan – SXG180066



Section 4: Creating Dummy variables and splitting to train-validation data

4.1 Creating dummy variables from train and test data

The dummy/indicator variables are created based on these columns - IsHoliday, Type, Store, Dept, DateVal, MonthVal, YearVal, quarterVal.

```
train_data_dummy_same_column = train_data

def createDummyVariable(data):
    data = pd.get_dummies(data, columns=['IsHoliday', 'Type', 'Store', 'Dept', 'DateVal', 'MonthVal', 'YearVal', 'quarterVal'])
    return data

train_data = createDummyVariable(train_data)
test_data = createDummyVariable(test_data)
```

4.2 Drop one of the dummy variables to avoid perfect collinearity

Amongst the dummy variables created, one of them is deleted to avoid perfect collinearity.

```
def removeOneDummyVariable(data):  
    data = data.drop(['IsHoliday_False', 'Type_A', 'Store_1', 'Dept_1', 'DateVal_1', 'MonthVal_1', 'YearVal_2010', 'quarterVal_1'], axis=1)  
    return data  
  
train_data = removeOneDummyVariable(train_data)  
test_data = removeOneDummyVariable(test_data)
```

4.3: x and y variable split

The train dataset is split such that – x variable with all columns except Weekly_Sales, y variable with only Weekly_Sales.

```
X_base = train_data.drop(['Weekly_Sales', 'Date'], axis=1)  
y_base = train_data[['Weekly_Sales']]
```

4.4: Preparing Min-max scaling data

The x_base_scaled dataset is created based on MinMaxScaler()

```
mms = MinMaxScaler()  
X_base_scaled = X_base.copy()  
columnsToScale = ['Temperature', 'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown5', 'CPI', 'Unemployment', 'Size']  
train_data_dummy_same_column_scaled = train_data_dummy_same_column  
  
for i in columnsToScale:  
    X_base_scaled[i] = mms.fit_transform(X_base_scaled[[i]]) #This is for X columns scaling  
    train_data_dummy_same_column_scaled[i] = mms.fit_transform(train_data_dummy_same_column[[i]]) #This is for OLS - finding si
```

4.5: Change y variable to log value

```
y_base_log = np.log(y_base)
```

The log function is applied to y variable.

4.6: Finding out the skewed variables

```
X_base_histogram = X_base[['Temperature', 'Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown5', 'CPI', 'Unemployment']  
skew(X_base_histogram)  
  
array([-0.32374356, -0.10297004,  4.71082111, 10.79168867, 15.07589677,  
       10.31874528,  0.08415372,  0.14198152, -0.3267488 ])
```

The variables having skewness below -1 or above +1 are considered skewed. The skewed variables are - 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown5'

4.7: Skewed variables are 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown5'. Let us apply log transformation on them

When the variables are skewed, applying log function would help capturing the exponential variation and so log is applied on these x columns below.

```
columnsToLog = ['MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown5']

X_base_log = X_base.copy()
X_base_scaled_log = X_base_scaled.copy()
train_data_dummy_same_column_scaled_log = train_data_dummy_same_column_scaled.copy()
train_data_dummy_same_column_log = train_data_dummy_same_column.copy()
for i in columnsToLog:

    X_base_log[i] = np.log(X_base_log[[i]], where=0<X_base_log[[i]])
    X_base_scaled_log[i] = np.log(X_base_scaled_log[[i]], where = 0<X_base_scaled_log[[i]])

    train_data_dummy_same_column_log[i] = np.log(train_data_dummy_same_column_log[[i]],
                                                where=0<train_data_dummy_same_column_log[[i]])
    train_data_dummy_same_column_scaled_log[i] = np.log(train_data_dummy_same_column_scaled_log[[i]],
                                                        where=0<train_data_dummy_same_column_scaled_log[[i]])
```

4.8: PCA Data for x base scaled

The PCA is applied to x base scaled to see the percentage of variance explained by each component.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=20)
pca.fit_transform(X_base_scaled)
print(pca.explained_variance_ratio_)

[0.06962292 0.0664059  0.05573721 0.05166675 0.05006957 0.03103677
 0.02920666 0.02676883 0.01994708 0.01840151 0.01710702 0.0163995
 0.01615843 0.0153916  0.01238336 0.01103008 0.00963034 0.00645792
 0.0061661  0.00608657]
```

4.9: PCA Data for x base scaled log

The PCA is applied to x base scaled log to see the percentage of variance explained by each component.

```
pca = PCA(n_components=3)
X_base_scaled_log_pca = pca.fit_transform(X_base_scaled_log)
print(pca.explained_variance_ratio_)

[0.47251884 0.0627518  0.05319979]
```

Section 5: Defining dataset model functions

5.1: Linear Regression

The function linear_regression_model with 5 KFold cross validation is written.

```
def linear_regression_model(X_base_linear_lm, y_base_linear_lm, methodName):
    lm = LinearRegression()
    kf = KFold(shuffle=True, n_splits=5)
    cv_results = cross_validate(lm, X_base_linear_lm, y_base_linear_lm, cv=kf, scoring=('r2', 'neg_mean_squared_error'))

    #Compute R^2
    test_r2 = cv_results["test_r2"]

    #Compute AdjustedR^2
    total_rows = X_base_linear_lm.shape[0]
    total_columns = X_base_linear_lm.shape[1]
    adj_r2 = 1-(1-test_r2)*(total_rows-1)/(total_rows-total_columns-1)

    #RMSE
    test_neg_mean_squared_error = cv_results["test_neg_mean_squared_error"]
    test_neg_mean_squared_error = np.mean((abs(test_neg_mean_squared_error))*(1/2))

    #Building Linear model with the entire data
    start = time.process_time()
    modelResult = lm.fit(X_base_linear_lm, y_base_linear_lm)
    picked_model = pickle.dumps(modelResult)

    #Inserting to Mongo DB
    query = { "Method": methodName}
    d = datetime.today()
    new_values = { "$set" : { "Method": methodName, "RMSE": test_neg_mean_squared_error,
                             "R^2": np.mean(test_r2), "Adj R^2": np.mean(adj_r2),
                             "Time Taken": (time.process_time() - start),
                             "currentDateTime" : d.strftime('%H:%M:%S %m/%d/%Y'),
                             "Model" : picked_model
                           }}
    doc = collection.find_one_and_update(query, new_values, upsert=True)
```

5.2: KNN Regression - Finding neighbors

The function knnBestRegressor with 5 KFold cross validation is written to figure out the best KNN neighbor – elbow point.

```
def knnBestRegressor(X_knn, y_knn):
    knnRMSEListLocal = pd.DataFrame(columns = ['Neighbors_count', 'RMSE'])
    for i in range(4,11):
        print(i)
        neigh = KNeighborsRegressor(n_neighbors=i)
        kf = KFold(shuffle=True, n_splits=5)
        cv_results = cross_validate(neigh, X_knn, y_knn, cv=kf, scoring=('r2', 'neg_mean_squared_error'))

        rmse_val = cv_results["test_neg_mean_squared_error"]
        rmse_val = np.mean((abs(rmse_val))*(1/2))

        test_r2 = np.mean(cv_results["test_r2"])

        knnRMSEListLocal = knnRMSEListLocal.append(
            { "Neighbors_count":i,
              "RMSE" : rmse_val,
              "R^2" : test_r2
            },ignore_index=True)
    return knnRMSEListLocal
```

5.3: Best KNN Regressor

The function bestKNNRegressor with 5 KFold cross validation is written.


```
def bestKNNRegressor(methodName, Neighbors_count, X_knn, y_knn, rmseVal, r2Val):  
    #Compute AdjustedR^2  
    total_rows = X_knn.shape[0]  
    total_columns = X_knn.shape[1]  
    adj_r2 = 1-(1-r2Val)*(total_rows-1)/(total_rows-total_columns-1)  
  
    #Building KNN model with the entire data  
    start = time.process_time()  
    neigh = KNeighborsRegressor(n_neighbors=Neighbors_count)  
    modelResult = neigh.fit(X_knn, y_knn)  
  
    picked_model = pickle.dumps(modelResult)  
    d = datetime.today()  
    query = { "Method": methodName}  
    new_values = { "$set" : { "Method": methodName, "RMSE": rmseVal,  
                             "R^2": r2Val, "Adj R^2": adj_r2,  
                             "Time Taken": (time.process_time() - start),  
                             "currentDateTime" : d.strftime('%H:%M:%S %m/%d/%Y'),  
                             "Neighbors" : Neighbors_count  
                           }}  
  
    collection.find_one_and_update(query, new_values, upsert=True)
```

5.4: Ridge elastic nets model

The function ridge_elasticnets_regression_model with 5 KFold cross validation is written.

```
def ridge_elasticnets_regression_model(ridgeOrElasticNets, methodName, x_lasso_ride, y_lasso_ride):
    clf = linear_model.Ridge() if ridgeOrElasticNets == "Ridge" else linear_model.ElasticNet()
    kf = KFold(shuffle=True, n_splits=5)
    cv_results = cross_validate(clf, x_lasso_ride, y_lasso_ride, cv=kf, scoring=('r2', 'neg_mean_squared_error'))

    #Compute R^2
    test_r2 = cv_results["test_r2"]

    #Compute AdjustedR^2
    total_rows = x_lasso_ride.shape[0]
    total_columns = x_lasso_ride.shape[1]
    adj_r2 = 1-(1-test_r2)*(total_rows-1)/(total_rows-total_columns-1)

    #RMSE
    test_neg_mean_squared_error = cv_results["test_neg_mean_squared_error"]
    test_neg_mean_squared_error = np.mean((abs(test_neg_mean_squared_error))**(1/2))

    d = datetime.today()

    #Building KNN model with the entire data
    start = time.process_time()
    modelResult = clf.fit(x_lasso_ride, y_lasso_ride)
    picked_model = pickle.dumps(modelResult)
    query = { "Method": methodName}
    new_values = {"$set" : {"Method": methodName, "RMSE": test_neg_mean_squared_error,
                           "R^2": np.mean(test_r2), "Adj R^2": np.mean(adj_r2),
                           "Time Taken": (time.process_time() - start),
                           "currentDateTime" : d.strftime('%H:%M:%S %m/%d/%Y'),
                           "Model" : picked_model
                          }}
    collection.find_one_and_update(query, new_values, upsert=True)
```

5.5 Bagging

The function BaggingRegressorFunction with 5 KFold cross validation is written.

```
def BaggingRegressorFunction(X_base_regressor, y_base_regressor, methodName):
    regr = BaggingRegressor()
    kf = KFold(shuffle=True, n_splits=5)
    cv_results = cross_validate(regr, X_base_regressor, y_base_regressor, cv=kf, scoring=('r2', 'neg_mean_squared_error'))

    print(cv_results)
    #Compute R^2
    test_r2 = cv_results["test_r2"]

    #Compute AdjustedR^2
    total_rows = X_base_regressor.shape[0]
    total_columns = X_base_regressor.shape[1]
    adj_r2 = 1-(1-test_r2)*(total_rows-1)/(total_rows-total_columns-1)

    #RMSE
    test_neg_mean_squared_error = cv_results["test_neg_mean_squared_error"]
    test_neg_mean_squared_error = np.mean((abs(test_neg_mean_squared_error))**(1/2))

    #Building Linear model with the entire data
    start = time.process_time()
    modelResult = regr.fit(X_base_regressor, y_base_regressor)

    #Inserting to Mongo DB
    query = { "Method": methodName}
    d = datetime.today()
    new_values = {"$set" : {"Method": methodName, "RMSE": test_neg_mean_squared_error,
                           "R^2": np.mean(test_r2), "Adj R^2": np.mean(adj_r2),
                           "Time Taken": (time.process_time() - start),
                           "currentDateTime" : d.strftime('%H:%M:%S %m/%d/%Y')
                          }}
    doc = collection.find_one_and_update(query, new_values, upsert=True)
```

5.6 Boosting

The function BoostingRegressorFunction with 5 KFold cross validation is written.

```
def BoostingRegressorFunction(X_base_regressor, y_base_regressor, methodName):
    regr = xgb.XGBRegressor(objective='reg:linear')
    kf = KFold(shuffle=True, n_splits=5)
    cv_results = cross_validate(regr, X_base_regressor, y_base_regressor, cv=kf, scoring=('r2', 'neg_mean_squared_error'))

    print(cv_results)
    #Compute R^2
    test_r2 = cv_results["test_r2"]

    #Compute AdjustedR^2
    total_rows = X_base_regressor.shape[0]
    total_columns = X_base_regressor.shape[1]
    adj_r2 = 1-(1-test_r2)*(total_rows-1)/(total_rows-total_columns-1)

    #RMSE
    test_neg_mean_squared_error = cv_results["test_neg_mean_squared_error"]
    test_neg_mean_squared_error = np.mean((abs(test_neg_mean_squared_error))**(1/2))

    #Building Linear model with the entire data
    start = time.process_time()
    modelResult = regr.fit(X_base_regressor, y_base_regressor)

    #Inserting to Mongo DB
    query = { "Method": methodName}
    d = datetime.today()
    new_values = {"$set" : {"Method": methodName, "RMSE": test_neg_mean_squared_error,
                            "R^2": np.mean(test_r2), "Adj R^2": np.mean(adj_r2),
                            "Time Taken": (time.process_time() - start),
                            "currentDateTime" : d.strftime('%H:%M:%S %m/%d/%Y')}
    }
    doc = collection.find_one_and_update(query, new_values, upsert=True)
```

Summary: Section 6 to Section 15

From Section 6 to Section 15, the combination of the following algorithms and dataset is experimented to find the best algorithm and dataset.

Algorithms

1. Linear Regression
2. Linear regression with significant coefficients
3. KNN Regressor
4. Ridge Regressor
5. Lasso Regressor

	X base	x scaled	PCA	x log	y base	y log
Section 6: Running dataset as it is	✓				✓	
Section 7: Features scaled		✓			✓	
Section 8: Log linear regression without variables scaled	✓					✓
Section 9: Log Regression, x scaled		✓				
Section 10: Log linear Regression - log x and y				✓	✓	
Section 11: Log Regression - log x scaled and y		✓		✓	✓	
Section 12: Log Regression - log x and log y				✓		✓
Section 13: Log Regression - log x scaled and log y		✓		✓		✓
Section 14: PCA: Log Regression - log x scaled and y		✓	✓	✓	✓	
Section 15: PCA: Log Regression - log x scaled and log y		✓	✓	✓		✓

Section 16: Finding out the best algorithm and dataset

16.1: Retrieve data from MongoDB

Based on section 6 to section 15 and 49 experiments; the R^2 , Adjusted R^2 values is retrieved from MongoDB. If the difference between R^2 and Adjusted R^2 is less than 5% then we can go with R^2 values or we have to go with Adjusted R^2 value.

```
mydoc = collection.find({"_id" : {"$exists":"true"}},{"Method", "Adj R^2", "R^2","Time Taken"})
dataModelsMongo = pd.DataFrame(columns = ["_id", "Method", "Adj R^2", "R^2", "Time Taken", "final R^2"])
for x in mydoc:
    dataModelsMongo = dataModelsMongo.append(
        {"_id":x["_id"],
         "Method" : x["Method"],
         "Adj R^2" : x["Adj R^2"],
         "R^2" : x["R^2"],
         "Time Taken" : x["Time Taken"]
        },ignore_index=True
    )

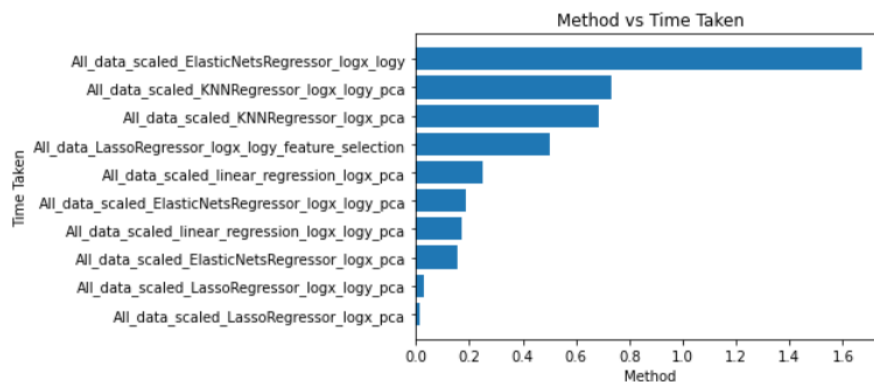
#Choose between R^2 and adjusted R^2
for i in range(0,len(dataModelsMongo)):
    diff_adj_r2 = round( (dataModelsMongo["R^2"][i] - dataModelsMongo["Adj R^2"][i]),4)
    dataModelsMongo["final R^2"][i] = diff_adj_r2

    if(diff_adj_r2 <= 0.05):
        dataModelsMongo["final R^2"][i] = dataModelsMongo["R^2"][i]
    else:
        dataModelsMongo["final R^2"][i] = dataModelsMongo["Adj R^2"][i]
```

16.2: Bar Chart for Time taken to run

The algorithm which runs at an optimal time is elastic nets algorithm with log x and log y data.

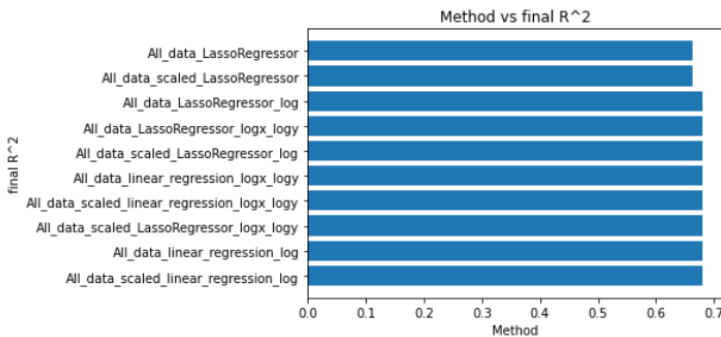
```
dataModelsMongoTime = dataModelsMongo.copy()
dataModelsMongoTime = dataModelsMongoTime.sort_values(by='Time Taken',ascending=True)
dataModelsMongoTime = dataModelsMongoTime.head(10)
plt.barh(dataModelsMongoTime["Method"],dataModelsMongoTime["Time Taken"])
plt.title('Method vs Time Taken')
plt.xlabel('Method')
plt.ylabel('Time Taken')
plt.show()
```



16.3: Bar Chart for final R^2

The best R^2 is from linear regression with log y variable.

```
dataModelsMongoTime = dataModelsMongo.copy()
dataModelsMongoTime = dataModelsMongoTime.sort_values(by='final R^2',ascending=False)
dataModelsMongoTime = dataModelsMongoTime.head(10)
plt.barh(dataModelsMongoTime["Method"],dataModelsMongoTime["final R^2"])
plt.title('Method vs final R^2')
plt.xlabel('Method')
plt.ylabel('final R^2')
plt.show()
```



16.4: Best Algorithm

After 49 experiments, top 10 R^2 and less time taken to run are plotted Based on the 2 graphs above, it is evident that All_data_scaled_linear_regression log is good for R^2 and All_data_scaled_ElasticNetsRegressor_logx_logy is good for time taken.

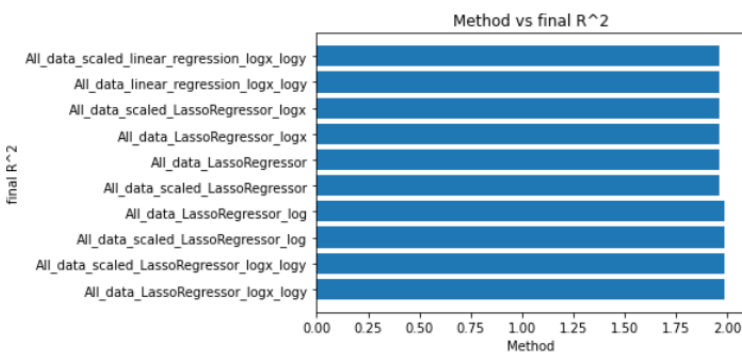
Let us build a parameter after scaling these values to figure out the best one - r^2 and Time taken

```
dataModelsMongoEfficient = dataModelsMongo.copy()

dataModelsMongoEfficient["final R^2"] = mms.fit_transform(dataModelsMongoEfficient[["final R^2"]])
dataModelsMongoEfficient["Time Taken"] = (1 - mms.fit_transform(dataModelsMongoEfficient[["Time Taken"]]))
dataModelsMongoEfficient["final_r2_timetaken"] = dataModelsMongoEfficient["final R^2"] + dataModelsMongoEfficient["Time Taken"]

dataModelsMongoEfficient = dataModelsMongoEfficient.sort_values(by='final_r2_timetaken',ascending=False)
dataModelsMongoEfficient = dataModelsMongoEfficient.head(10)

plt.barh(dataModelsMongoEfficient["Method"], dataModelsMongoEfficient["final_r2_timetaken"])
plt.title('Method vs final R^2')
plt.xlabel('Method')
plt.ylabel('final R^2')
plt.show()
```



The best algorithm is lasso regressor with log x and log y dataset.

Section 17: Feature Selection

The feature selection is performed for log x, log y dataset using random forest and lasso regression is applied on the selected features only.

```
selector = SelectFromModel(RandomForestRegressor(n_estimators=100, random_state = 10)).fit(X_base_log, y_base_log)
colNames = X_base_log.columns[selector.get_support()]
print(colNames)
X_base_log_feature_selection = X_base_log[colNames]
ridge_elasticnets_regression_model("Ridge", "All_data_LassoRegressor_logx_logy_feature_selection", X_base_log_feature_selection, y_base_log)
```

Section 18: Bagging Regressor

The bagging regressor is applied on log x and log y dataset. The 5 KF cross validation R^2 results are these :-

- 0.96283071
- 0.9636206
- 0.96367833
- 0.96387069
- 0.96512824

The difference between the R^2 and adjusted R^2 also is less than 5%. The best algorithm is bagging regressor with the average R^2 of 0.96.

Section 19: Boosting Regressor

The boosting Regressor is applied on log x, log y dataset and the average R^2 value is 0.9037077714357187.