ECE 9603/9063b – Data Analytics Foundations

Assignment 2: Neural Networks

BY

Srujana Bushireddy (251041014)

Forecasting Problem

Now a days having a healthy life style is a dream of everyone. In this busy world people hardly have time to concentrate on exercises to be healthy. One of the best ways to stay healthy is having a healthy diet. Healthy diet majorly contains avocado in it. Avocado is a nutrient-dense fruit which helps us in decreasing the depression and it is also used as a cure for the cancer. Due to the huge benefits of avocado its prices have been increasing drastically. So, our forecasting model helps us in **predicting the average price of the avocado.**

Overview of the network architecture(s) you have used including important parameters that affect the network operation. Make sure you include parameters you will tune for your problem.

Multilevel perceptron:

A multilayer perceptron (MLP) is are the classical type of the neural networks. They are the class of **feedforward artificial neural network**. A simple MLP network contains at least three layers they are input layer, hidden layer and output layer. We can have more than one hidden layer in MLP. Generally, the data is fed to the input layer and we can get the predicted value from the output layer which is also known as visible layer. For training MLP uses **backpropagation** which is a supervised learning technique. We have a non-linear activation in MLP which makes it different form a linear perceptron. It is used for the data which is not linearly separable. Multilevel perceptron is used for both classification and regression.

In our problem we will deal with regression, so we will be using **MLPRegressor** Class. This class implements MLP that trains using backpropagation without no activation function in the output, which can also be seen as using the identity function as activation function. That is the reason it uses the square error as the loss function and the output is set to various continuous values. It can also support multiple outputs.

The main reason for choosing this network architecture is it can be used for non-linear data, its is suitable for tabular data(CSV file) and regression prediction problems. Where as CNN is used for classification and images problems and RNN is used not suitable for tabular data(CSV files).

Parameters of MLPRegressor:

Name	Values	Description
hidden_layer_sizes	tuple, length = n_layers - 2, default (100,)	It tells the number of neurons present in the particular layer
Activation	{‘identity’, ‘logistic’, ‘tanh’, ‘relu’}, default ‘relu’	It gives us the information about the activation function of the hidden layers.
Solver	{‘lbfgs’, ‘sgd’, ‘adam’}, default ‘adam’	It is used for weights optimization
batch_size	int, optional, default ‘auto’	It tells us the size of the mini batch it is used only for stochastic optimizers
learning_rate	{‘constant’, ‘invscaling’, ‘adaptive’}, default ‘constant’	It tells us the learning rate scheduled for the weights updates
learning_rate_init	double, optional, default 0.001	It is the initial running rate and defines the step size in updating the weights.

power_t	double, optional, default 0.5	It is the exponent for the inverse scaling rate. When the learning rate is set to invscaling it is used to update the effective learning rate.
max_iter	int, optional, default 200	It tells us the maximum number of iterations. It determines the number of epochs for stochastic solvers
Shuffle	bool, optional, default True	It tells us to shuffle the samples in the each iteration or not
random_state	int, RandomState instance or None, optional, default None	Based on the type of random_state the random number is generated.
Tol	float, optional, default 1e-4	It gives the tolerance for the optimization.
warm_start	bool, optional, default False	When set to True reuse the solution of the previous call to fit initialization or else just erase the pervious data
Momentum	float, default 0.9	Momentum for gradient descent update.
nesterovs_momentum	boolean, default True	It tells us whether to use Nesterov's momentum or not.
early_stopping	bool, default False	It tells to stop early to terminate the training if the validation score is not improving
validation_fraction	float, optional, default 0.1	The proportion of the training data set which is used for validation for early stopping
beta_1	float, optional, default 0.9	Exponential decay rate for estimates of first moment vector in adam, should be in [0, 1)
beta_2	float, optional, default 0.999	Exponential decay rate for estimates of second moment vector in adam, should be in [0, 1)
Epsilon	float, optional, default 1e-8	Value for numerical stability in adam.
n_iter_no_change	int, optional, default 10	Maximum number of epochs to not meet tol improvement

The parameters we have used in this model are a below

1) Hidden_layer_sizes
2) Activation
3) Solver
4) Batch_size
5) Learning_rate
6) Learning_rate_init
7) Max_iter
8) Verbose
9) Momentum
10) Early_stopping
11) Validation_fraction
12) N_iter_no_change

Hidden_layer_sizes: In our training we have choose different combination of number of layers and number of neurons in each to find out the best model

Activation: We have four different types of activation functions. They are as follows

a) Identity : It is a no-op activation and is used to implement linear bottleneck. This function will return f(x) = x
b) Logistic : It is the logistic sigmoid function. This function will return f(x) = 1/(1+exp(-x))
c) Tanh : It is the hyperbolic tan function. This function will return f(x) = tanh(x)
d) Relu : It is the rectified linear unit function. This function will return f(x) = max(0,x)

Now a days we are using the relu function as the activation function for all the regression problems and it is the default value of the activation function in MLPRegressor. So, we are using the relu as the activation function through out our tuning.

Solver: We are having three different types of solver function. They are as below

a) Lbfgs : It is an optimizer in the family of quasi-Newton methods.
b) Sgd: It refers to stochastic gradient descent.
c) Adam : It refers to as stochastic gradient based optimizer proposed by kingma,Diederik and Jimmy Ba

Generally, adam works pretty well on the relatively large datasets in terms of both validation score and the training time. However lbfgs will converge fastly and perform better.

In this training we will be using all the three Solvers to find out which one is best for our dataset.

Batch_size : The batch_size has been thoughout the tuning process.

Learning_rate: We have three type of learning_rates. They are as follows

a) Constant : It is a constant learning rate given by learning_rate_init
b) Invscaling : IT gradually decreases the learning rate at each time step ‘t’ using an inverse scaling exponent of ‘power_t’
c) Adaptive: It keeps the learning rate constant to learning_rate_init as long as training loss keeps decreasing.

In this training we have used only the default value of learning_rate i.e. constant. This is only used when the solver is sgd.

Learning_rate_init : The learning_rate_init is default i.e. 0.001

Max_iter : The max_iter has been varying from 500 to 10000. For stochastic solvers it determines the number of epochs.

Verbose : The value of verbose is True which means it prints the progress messages.

Momentum : Generally the momentum value should be high. So, we have it as the default value i.e 0.9. This is only used when the solver is sgd.

Early_stopping: Its value is True which means it force stops the model when the validation score is nor improving. This is only used when the solver is sgd or adam.

Validation_fraction : It tells the fraction of test data which can be used for the validation. In this tranign we will be using the default value of it i.e. 0.1 which means we use 10% of the test data for validation. It is used only when the early_stopping is true.

N_iter_no_change: It tells us the maximum number of epochs to not meet tol improvement. We assigned it to the default value i.e 10 which means after the 10 epochs it will consider whether validation score is being increased or not.

Description of the process you have used including data pre-processing, feature generation, model training, and evaluation. (7 points)

Data Pre-processing is a data mining technique that transforms the raw data into readable format. In general, the real data is incomplete, inconsistent or lacks in some behaviour. Data Pre-processing helps us in resolving this issue.

We will read the data and try to get the information present in the CSV file.

	Unnamed: 0	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region
0	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	2015	Albany
1	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	2015	Albany
2	2	2015-12-13	0.93	118220.22	794.70	109149.67	130.50	8145.35	8042.21	103.14	0.0	conventional	2015	Albany
3	3	2015-12-06	1.08	78992.15	1132.00	71976.41	72.58	5811.16	5677.40	133.76	0.0	conventional	2015	Albany
4	4	2015-11-29	1.28	51039.60	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	2015	Albany

Fig 1: Information of the dataset

In the above figure we can observe that we are having a column with Unnamed: 0 which gives us the information of the number of rows. We don't have much information in this column which helps us in our model, so we will drop this column using **drop** command. Later, we will be checking whether we have any incomplete data. To get this we need to read the data from the CSV file

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 13 columns):
Date            18249 non-null object
AveragePrice    18249 non-null float64
Total Volume    18249 non-null float64
4046            18249 non-null float64
4225            18249 non-null float64
4770            18249 non-null float64
Total Bags      18249 non-null float64
Small Bags      18249 non-null float64
Large Bags      18249 non-null float64
XLarge Bags     18249 non-null float64
type            18249 non-null object
year            18249 non-null int64
region          18249 non-null object
dtypes: float64(9), int64(1), object(3)
memory usage: 1.8+ MB
```

Fig 2: Datatype and the nonnull attributes of the dataset

In the above figure we can observe that we are having total 13 columns in our data and we don't have any missing values in it.

Feature Engineering is the process of using domain knowledge to create features that make machine learning work.

In our dataset we will try to do the feature engineering on the data feature so that we can extract day and month from it and use it further in creating our model. Please note that we are already having the year column in the data, so we don't need to extract it from the date.

To get a clear idea about the relation between the columns we need to have a correlation map.

Code : sns.heatmap(data.corr(),cmap='coolwarm',annot=True)

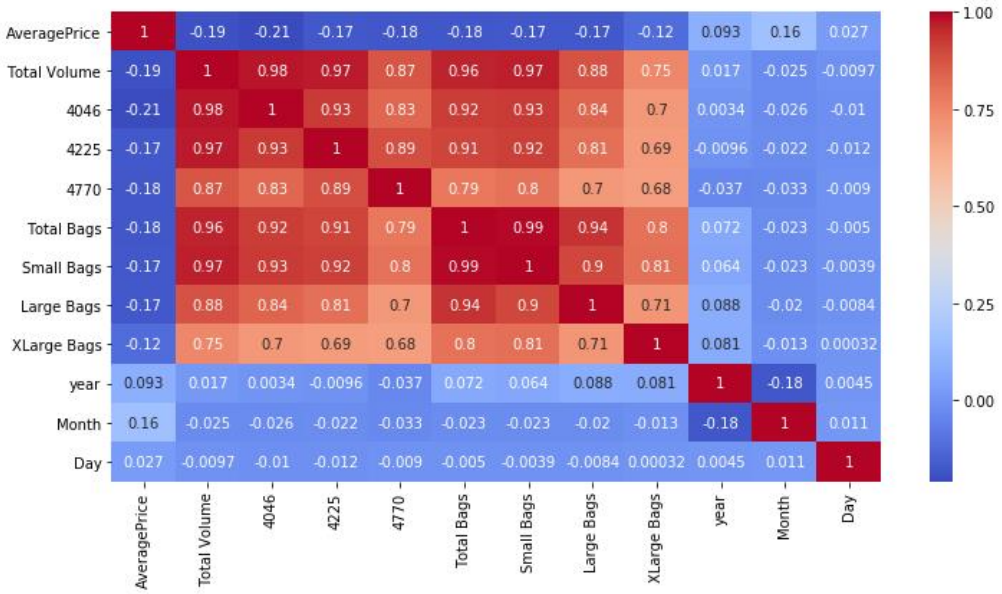

Fig 3: Correlation diagram of the attributes in the dataset

In the above figure we can see that the attributes are interdependent they are not directly dependent on the average price.

If we observe, we are having two categorical features in our data they are region and type.

We have 54 regions in the data and 2 types, so we can transform types features to dummies but transforming the regions is difficult. So, we will be dropping the region and date columns as we already have date, month and year.

The final attributes for our model are as below

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18249 entries, 0 to 18248
Data columns (total 13 columns):
AveragePrice    18249 non-null float64
Total Volume    18249 non-null float64
4046            18249 non-null float64
4225            18249 non-null float64
4770            18249 non-null float64
Total Bags      18249 non-null float64
Small Bags      18249 non-null float64
Large Bags      18249 non-null float64
XLarge Bags     18249 non-null float64
year            18249 non-null int64
Month           18249 non-null int64
Day             18249 non-null int64
type_organic    18249 non-null uint8
dtypes: float64(9), int64(3), uint8(1)
```

Fig 4: Final details of the dataset after dropping the unwanted coulmns

Model Training:

We have split the data using the train_test_split from the model_selection. The ratio of the train and test data is **75:25**. As it splits the data randomly when ever we run the model's multiple times, we may get the different outputs. I trained the model based on the following parameters

1)Number of hidden layers

2)Number of neurons

3) Type of activation function

4)Loss function/Solver

5) Batch_size

6) max_iter

7) momentum

8) early_stopping

9) validation_fraction

10) learning_rate

11)learning_rate_init

12) Verbose

13) n_iter_no_change

Training 1:

In this we are tuning the parameter as below

1) Number of hidden layers = 2
2) Number of neurons = 10,10
3) Activation function = relu
4) Loss function = squared-loss using stochastic gradient descent

Code : MLPRegressor(hidden_layer_sizes=(10,10))

We didn't mention the activation function and the solver because we are using the default values if them i.e. relu and adam respectively.After running this model we got a very high RMSE and MAE values which means the accuracy of this model is not good. So, I have changed the number of neurons to 50,50 but even then the error didn't reduce in a considerable range(Fig : 9).

Training 2 :

Increased the number of layers and the number of neurons per layer.

1) Number of hidden layers = 4
2) Number of neurons = 50,40,30,20,10
3) Activation function = relu
4) Loss function = squared-loss using stochastic gradient descent

Code : MLPRegressor(hidden_layer_sizes=(50,40,30,20,10))

We didn't mention the activation function and the solver because we are using the default values if them i.e. relu and adam respectively. After running this model for multiple time(10) the best RSME and MAE values we got are high (Fig:23).

Training 3:

Tuned the model with more parameters keeping the number of neurons and the number layers same.

1)Number of hidden layers = 4

2)Number of neurons = 50,40,30,20,10

3)Activation function = relu

4)Loss function (solver)= squared-loss using stochastic gradient descent

5) batch_size = 500

6) max_iter = 500 (This is the number of epoch when the solver is sgd/adam)

7) early_stopping = True (If the loss is not improved it will force stop the model)

8) validation_fraction = 0.1 (It means that 10 percent of the training set is used to tell the accuracy of the model)

9) momentum = 0.9

10) learning_rate = constant (This will make the model learn constantly at rate which is given by learning_rate_init)

11)learning_rate_init = 0.001

12) Verbose = True (This will print the progress message)

13) n_iter_no_change = 10

Code : MLPRegressor(hidden_layer_sizes=(50,40,30,20,10),max_iter=500,early_stopping=True,batch_size=500,verbose= True)

Here we didn't give activation_function,solver,validation_fraction , momentum, n_iter_no_change ,learning_rate and learning_rate_init values because the values which we mentioned above are the default values so we don't need to mention them explicitly in the code.

```
Iteration 1, loss = 378883064.24671811
Validation score: -748186160.062895
Iteration 2, loss = 32566247.75574348
Validation score: -219338346.368429
Iteration 3, loss = 13137416.50856579
Validation score: -133884071.054459
Iteration 4, loss = 8500549.34915648
Validation score: -93983566.544550
Iteration 5, loss = 7784381.38439046
Validation score: -65583977.998529
Iteration 6, loss = 4742667.09887741
Validation score: -57707387.293917
Iteration 7, loss = 4059028.70485441
Validation score: -36367714.626136
Iteration 8, loss = 3136882.65356106
Validation score: -35490233.290554
Iteration 9, loss = 3365671.80972416
Validation score: -40211991.134542
Iteration 10, loss = 2256359.59672505
Validation score: -57043165.795954
Iteration 11, loss = 3812369.91497568
Validation score: -36421105.613303
Validation score did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
```

Fig 5: Progress of the model while tuning

```
Validation score: -26445586.164268
Iteration 35, loss = 2448655.61319871
Validation score: -27568607.236672
Iteration 36, loss = 2190953.38298855
Validation score: -22623179.587542
Iteration 37, loss = 2023659.65868695
Validation score: -24819412.057866
Iteration 38, loss = 2034388.85850073
Validation score: -47603659.347105
Iteration 39, loss = 2233225.29310641
Validation score: -22412842.048978
Iteration 40, loss = 2064918.51836611
Validation score: -21665568.972342
Iteration 41, loss = 1883178.17267984
Validation score: -30197293.839840
Iteration 42, loss = 3341539.20772446
Validation score: -84061954.619155
Iteration 43, loss = 6558737.76907546
Validation score: -30347279.904568
Validation score did not improve more than tol=0.000100 for two consecutive epochs. Stopping.
```

Fig 6: Progress of the model while tuning

After running the code the message in the fig 5 and 6 is printed which means that the validation score is not improving so we have early stopped the model. Here we can see that after 10 iterations we started comparing the validation score improvement because n_iter_no_change over is the default value i.e. 10. We have calculated the RMSE and MAE for the model and it is very high.We have changed the values in the batch_size and max_iter but still there is no significant decrease in the error values(Fig: 36).

Training 4:

In this model we have changed the solver from the default one i.e. adam to sgd and kept all the remaining parameters as same as the above training model(didn't give the early_stopping)

Code : MLPRegressor(hidden_layer_sizes=(50,30,20,10),solver ='sgd',max_iter=1000,batch_size=500,verbose=True)

When we run this code, we are getting the convergence error which means that the model is not able to converge on a solution which is system acceptable. So, I have changed the number of neurons, number of layers and even the max_iter to 10000 even then this issue didn't resolve.

```
Iteration 987, loss = nan
Iteration 988, loss = nan
Iteration 989, loss = nan
Iteration 990, loss = nan
Iteration 991, loss = nan
Iteration 992, loss = nan
Iteration 993, loss = nan
Iteration 994, loss = nan
Iteration 995, loss = nan
Iteration 996, loss = nan
Iteration 997, loss = nan
Iteration 998, loss = nan
Iteration 999, loss = nan
Iteration 1000, loss = nan
C:\Users\sruja\Anaconda3\lib\site-packages\sklearn\neural_network\multilayer_perceptron.py:564: ConvergenceWarning: Stochasti
c Optimizer: Maximum iterations (1000) reached and the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

Fig 7 : Progress of the model while tuning when the max_iter is 1000

Fig 8 : Progress of the model while tuning when the max_iter is 10000

Training 5:

In the previous model we had a convergence issue, so we have changed the solver to lbfgs.

Please note that for this solver we don't have the following parameters

1) Batch_size
2) Learning_rate
3) Learning_rate_init
4) Momentum
5) Early_stopping
6) N_iter_no_change

We are training this model on the parameters as mentioned in the code

Code : MLPRegressor(hidden_layer_sizes=(10,10),solver='lbfgs', max_iter=500,verbose=1)

Here we can see that we have only 2 hidden layers and each have 10 neurons each. The number of iterations is 500

We have observed a drastic change in the RSME and MAE values(Fig: 49).

Training 6:

I have changed the number of neurons, the layers of the neurons and also the number of iterations to get the lowest error value.

Options which I tired are as below

1) MLPRegressor(hidden_layer_sizes=(50,30),solver='lbfgs', max_iter=500,verbose=1)
2) MLPRegressor(hidden_layer_sizes=(50,40,30),solver='lbfgs', max_iter=1000,verbose=1)
3) MLPRegressor(hidden_layer_sizes=(10,10,10,10),solver='lbfgs', max_iter=1000,verbose=1)
4) MLPRegressor(hidden_layer_sizes=(10,10,10,10,10),solver='lbfgs', max_iter=5000,verbose=1)
5) MLPRegressor(hidden_layer_sizes=(50,50,50,50,50),solver='lbfgs', max_iter=5000,verbose=1)
6) MLPRegressor(hidden_layer_sizes=(50,40,30,20,10),solver='lbfgs', max_iter=10000,verbose=1)
7) MLPRegressor(hidden_layer_sizes=(50,40,30,20,10,10),solver='lbfgs', max_iter=10000,verbose=1)

We got the lowest error for the 6th one. So, we can finalise our model with these parameters. So, we will be showing the result only for the 6th one out of all the above models(Fig: 63).

Results:

We can calculate the accuracy of the model in two ways one of them is **scale-dependent error**.

In this we can choose either Mean absolute error(MAE) or Root Mean square error(RMSE) to calculate the error.

Mean absolute error (MAE):

$$\text{MAE} = \text{mean}(|y_i - \hat{y}_i|)$$

Mean square error (MSE) :

$$\text{MSE} = mean(|y_i - \hat{y}_i|)^2$$

Root Mean square error (RMSE):

$$\text{RMSE} = \sqrt{mean(|y_i - \hat{y}_i|)^2}$$

Where y_i is the i-th observation

$\hat{y}_i$ is the forecast of y_i

In both the formulas will calculate the difference between the observed value and the forecasted value.

To get the best results I ran all the models 10 times each and took out the best value out of them.

Results while tuning the neural network:

Training 1:

MLPRegressor(hidden_layer_sizes=(10,10))

Error Values:

```
MAE:  1769.5421928204207
MSE:  23927728.05923251
RMSE:  4891.597700060023
```

Fig 9: Error value for the model 1

In the fig 9 we can see that the errors are very high. So, we cannot go with this model

Graphs:

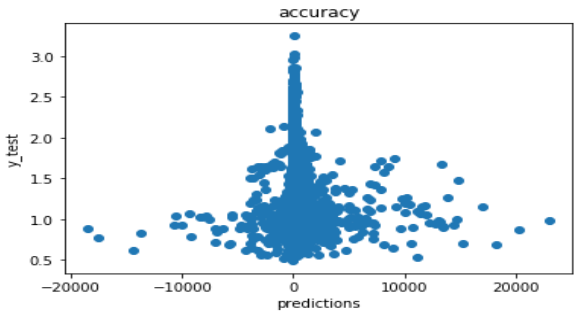

Fig 10: Predicted values vs y_test values

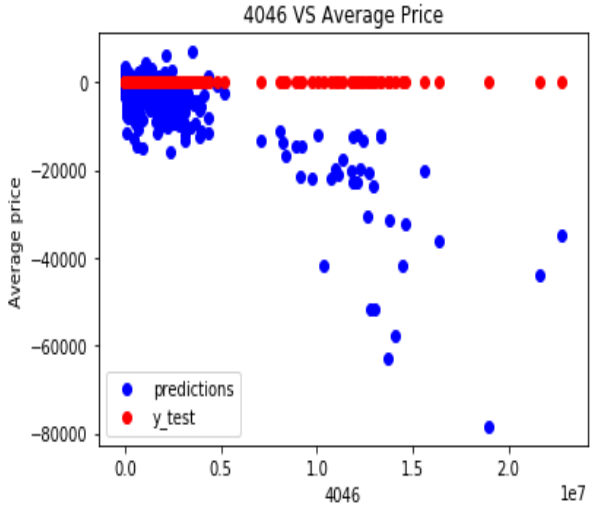

Fig 11 : Predicted and y_test Vs 4046 attribute

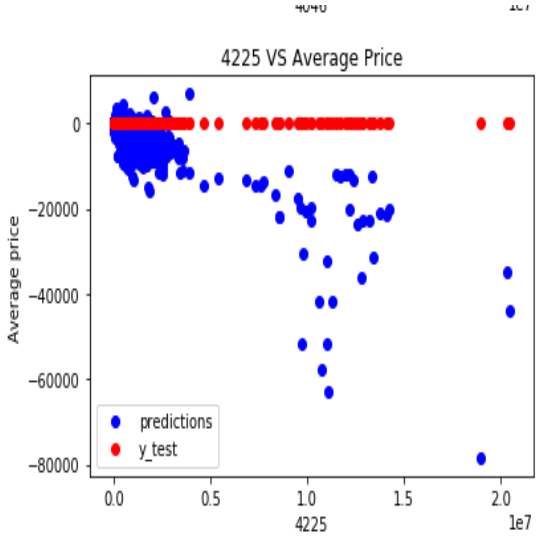

Fig 12 : Predicted and y_test Vs 4225 attribute

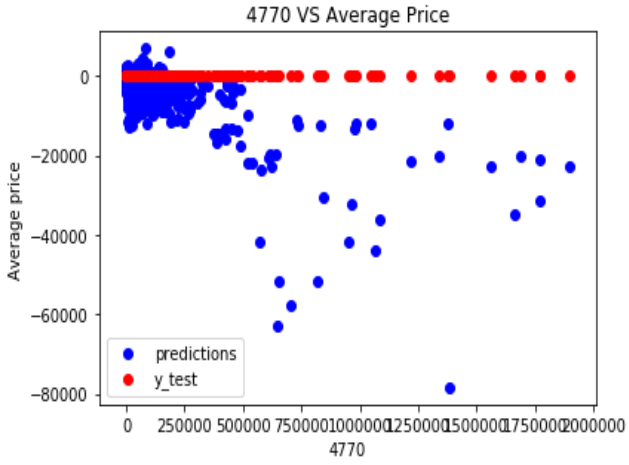

Fig 13 : Predicted and y_test Vs 4770 attribute

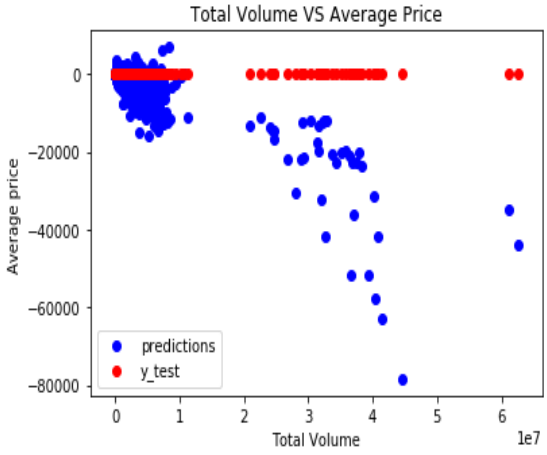

Fig 14 : Predicted and y_test Vs Total Volume attribute

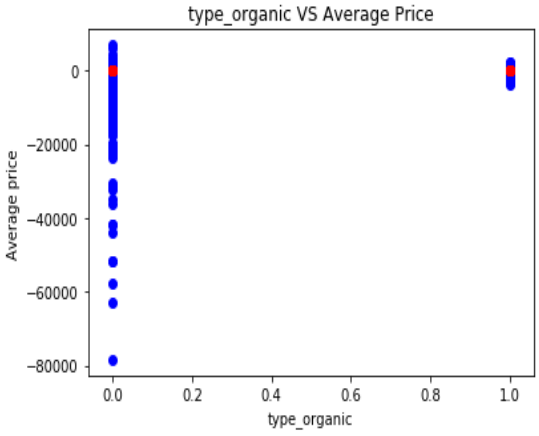

Fig 15 : Predicted and y_test Vs type_orgaing attribute

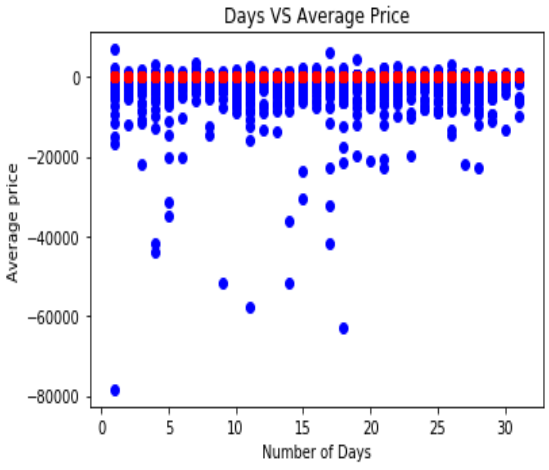

Fig 16 : Predicted and y_test Vs Days attribute

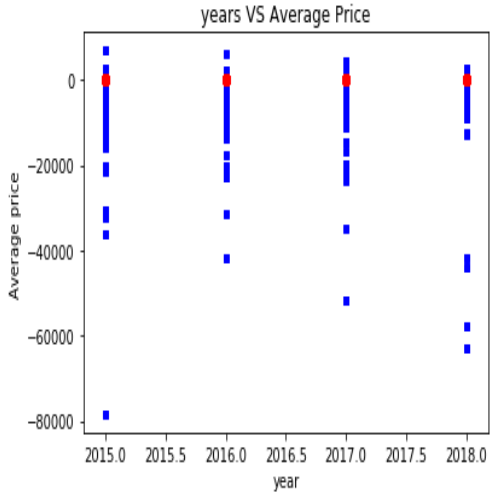

Fig 17 : Predicted and y_test Vs years attribute

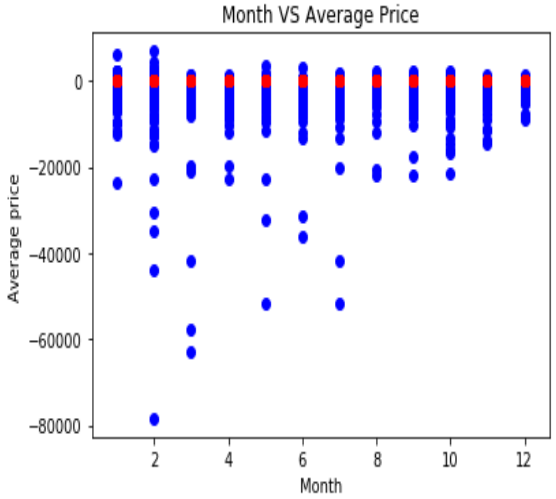

Fig 18 : Predicted and y_test Vs month attribute

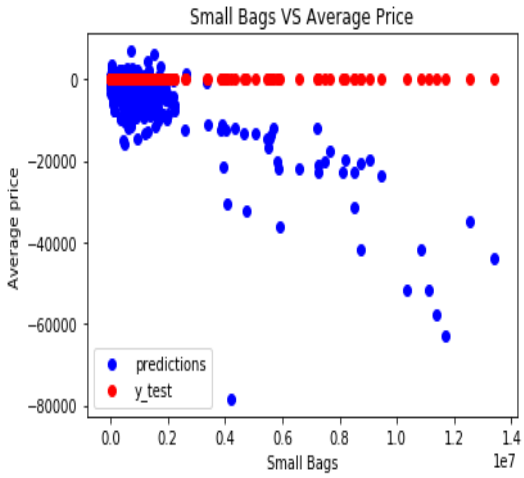

Fig 19 : Predicted and y_test Vs Small Bags attribute

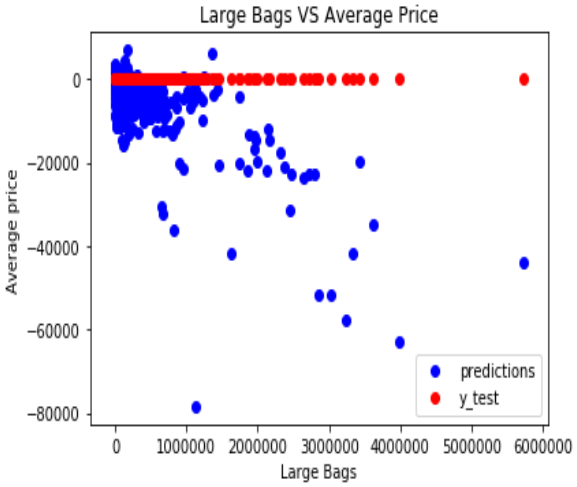

Fig 20 : Predicted and y_test Vs LargeBags attribute

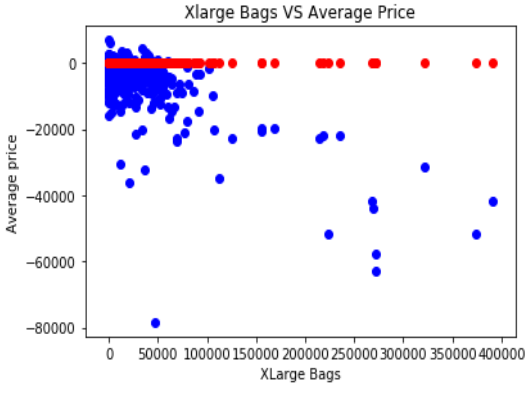

Fig 21 : Predicted and y_test Vs XlargeBags attribute

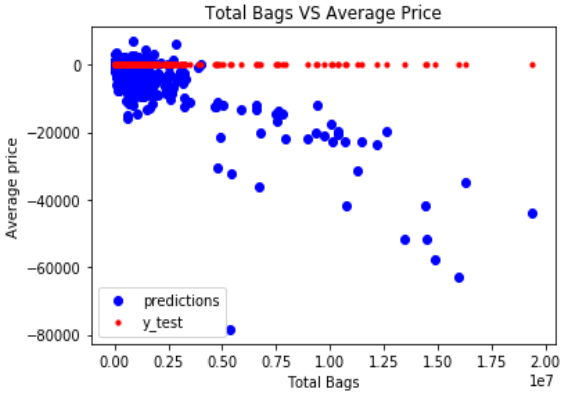

Fig 22 : Predicted and y_test Vs TotalBags attribute

In the figure 10 we can see that there is lot of difference between y_test and predicted value i.e the scale is very different. We can say the values are same if we observe any line in the graph and the scale of both axis should be same. In the above figure we have plotted both y_test and predicted values versus with all the input parameters of the dataset. We can observe that there is a lot variation between them in all the figures. The accuracy of this model is very low.

Training 2:

MLPRegressor(hidden_layer_sizes=(50,40,30,20,10))

Error:

```
MAE:   333.2483626822057
MSE:   1037111.3782332239
RMSE:  1018.3866545832304
```

Fig 23: Error for the training 2

In the fig 23 we can observe that the error values are reduced when compared to the error values in the training 1 but they are not low to consider this model.

Graphs:

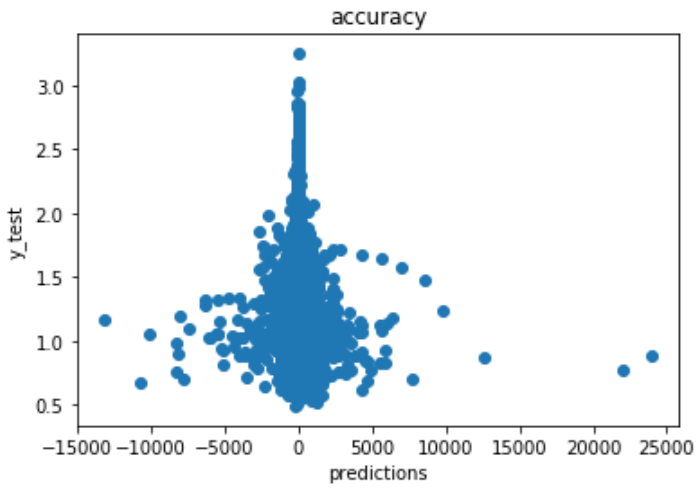

Fig 24: predicted values Vs y_test for training 2

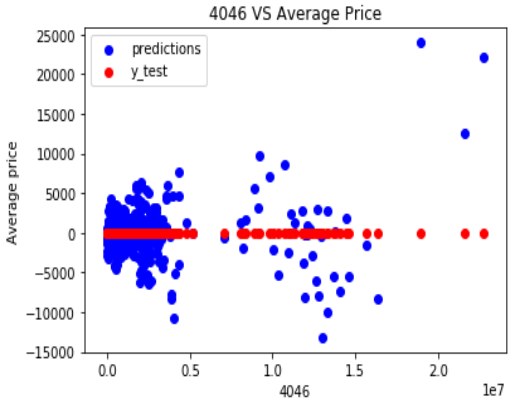

Fig 25: Predicted and y_test Vs 4046 attribute

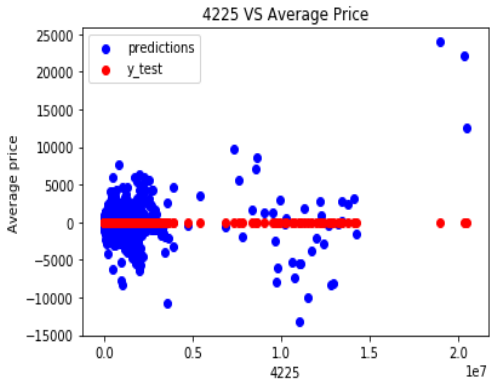

Fig 26: Predicted and y_test Vs 4225 attribute

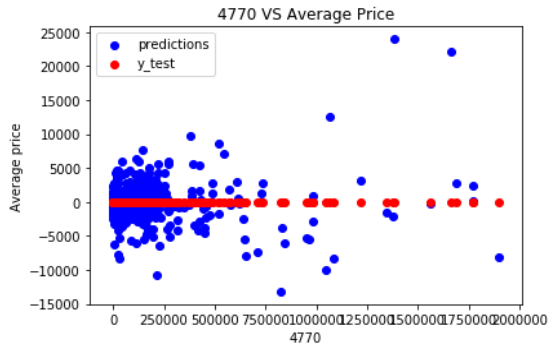

Fig 27: Predicted and y_test Vs 4770 attribute

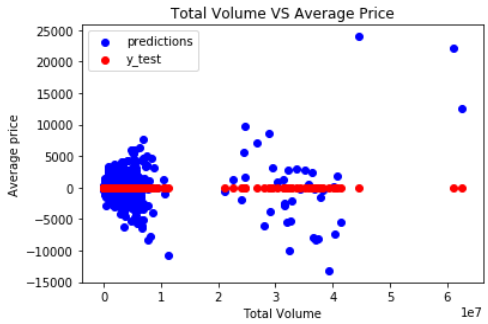

Fig 28: Predicted and y_test Vs TotalVolume attribute

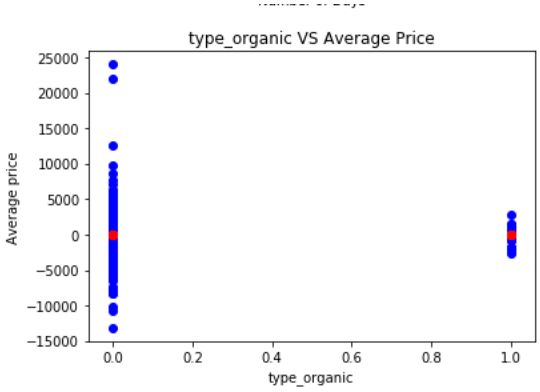

Fig 29: Predicted and y_test Vs type_organic attribute

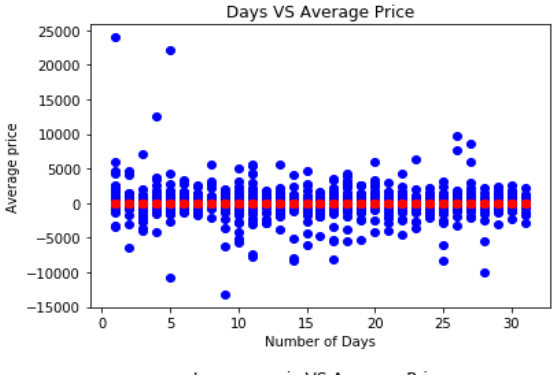

Fig 30: Predicted and y_test Vs Days attribute

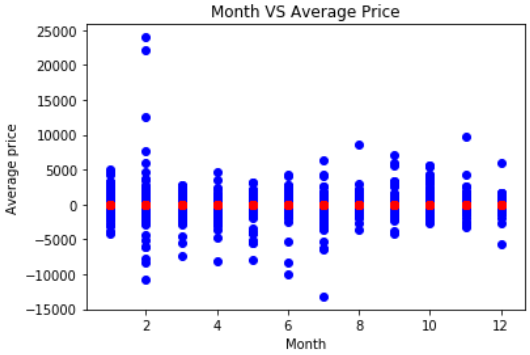

Fig 31: Predicted and y_test Vs Month attribute

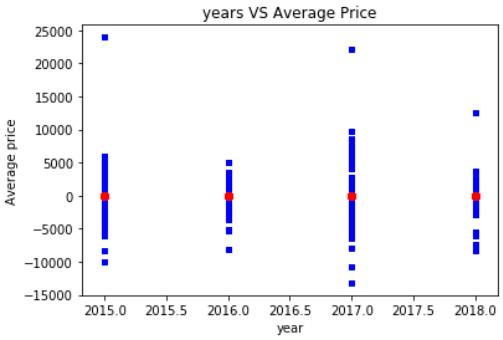

Fig 32: Predicted and y_test Vs years attribute

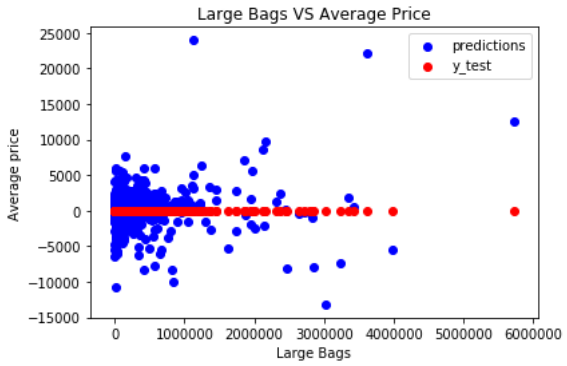

Fig 33: Predicted and y_test Vs largeBags attribute

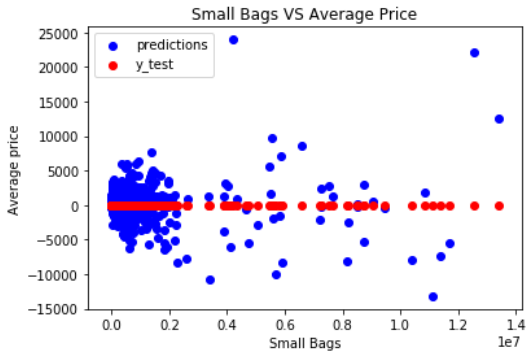

Fig 34: Predicted and y_test Vs smallBags attribute

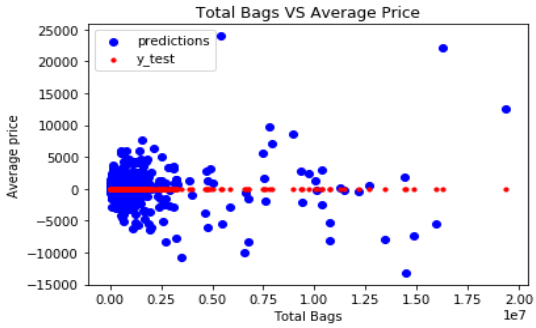

Fig 34: Predicted and y_test Vs TotalBags attribute

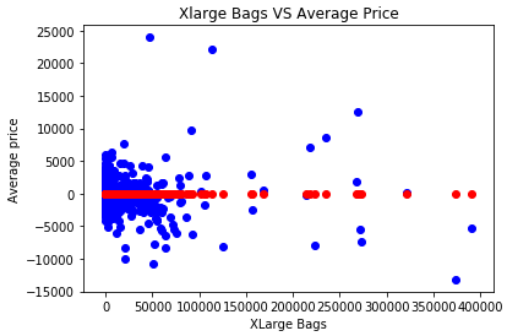

Fig 35: Predicted and y_test Vs XlargeBags attribute

In the above figures we have plotted both y_test and predicted values versus with all the input parameters of the dataset. We can observe that there is a lot variation between them in all the figures. The accuracy of this model is better than the previous model but overall the accuracy of this model is less.

Training 3:

MLPRegressor(hidden_layer_sizes=(50,40,30,20,10),max_iter=500,early_stopping=True,batch_size=500,verbose= True)

```
MAE:  621.2254416667178
MSE:  3614066.824779493
RMSE:  1901.0699158051743
```

Fig 36 Error values for training 3

In the above figure we can see that the error values are more when compared to the pervious model. Which means that the accuracy of this model is even less than the previous model.

Graphs:

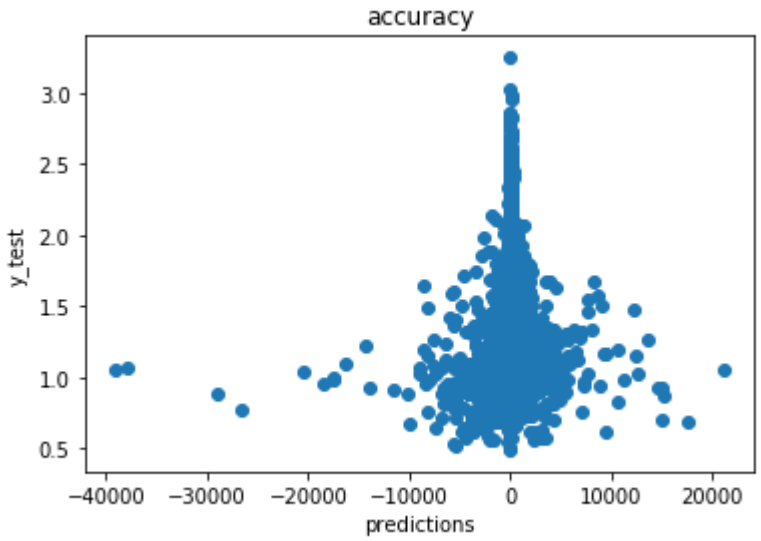

Fig 37: predicted values Vs y_test

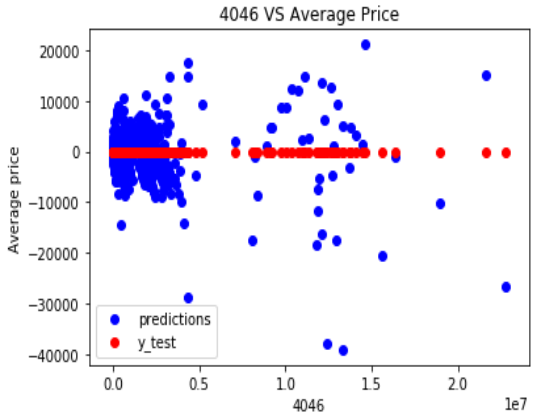

Fig 38: Predicted and y_test Vs 4046 attribute

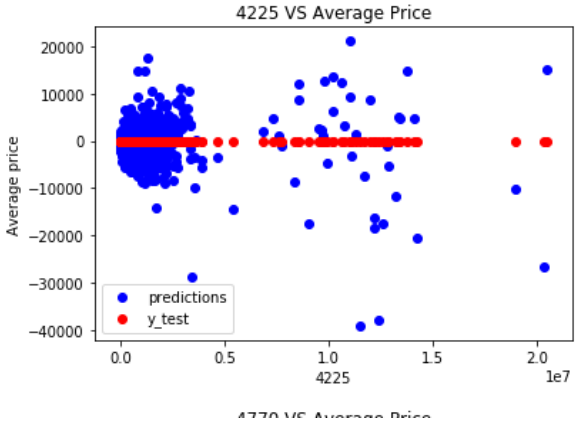

Fig 39: Predicted and y_test Vs 4225 attribute

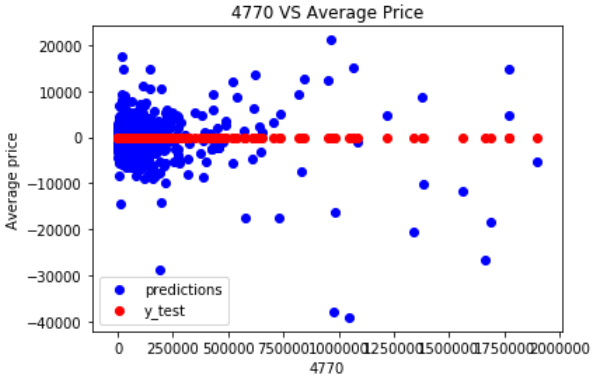

Fig 40: Predicted and y_test Vs 4770 attribute

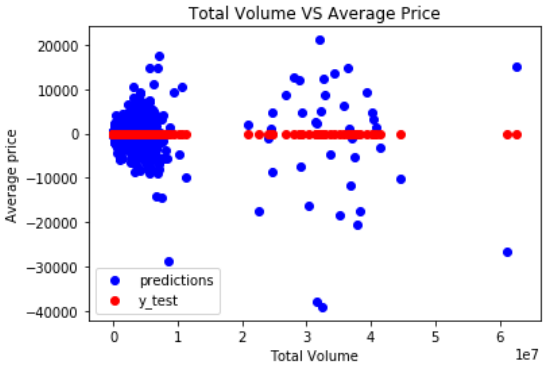

Fig 41: Predicted and y_test Vs TotalVolume attribute

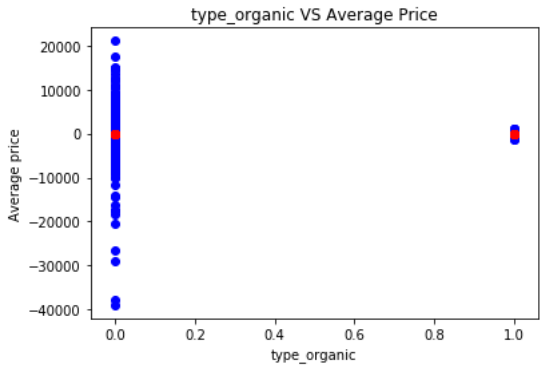

Fig 42: Predicted and y_test Vs organic attribute

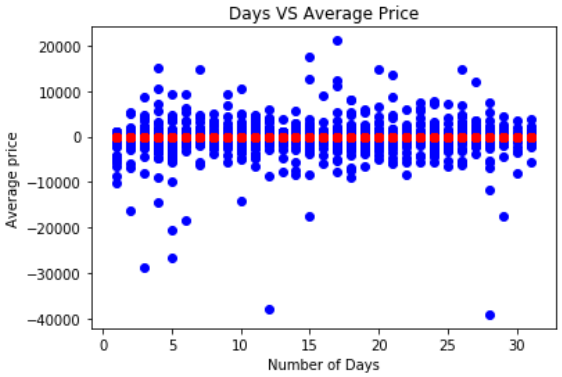

Fig 43: Predicted and y_test Vs Days attribute

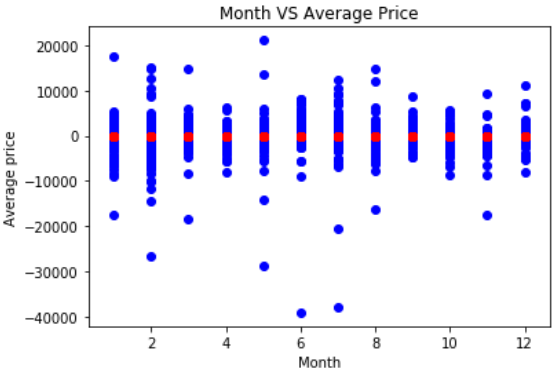

Fig 44: Predicted and y_test Vs Months attribute

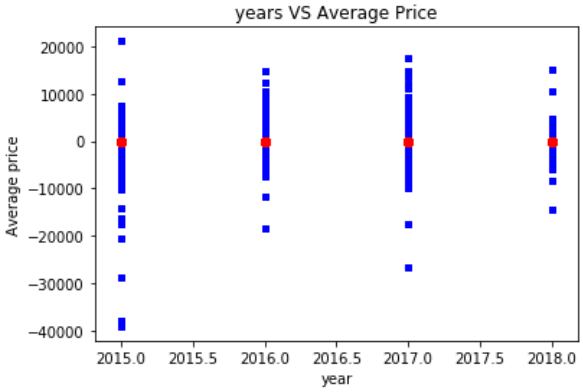

Fig 45: Predicted and y_test Vs years attribute

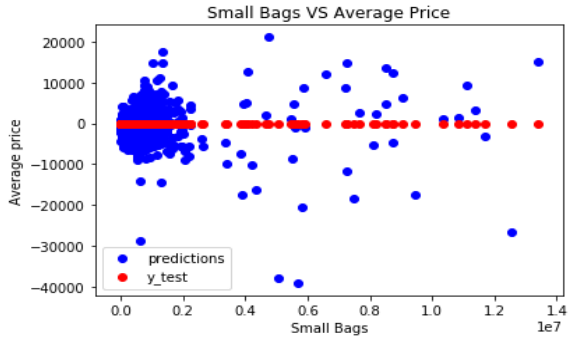

Fig 45: Predicted and y_test Vs SmallBags attribute

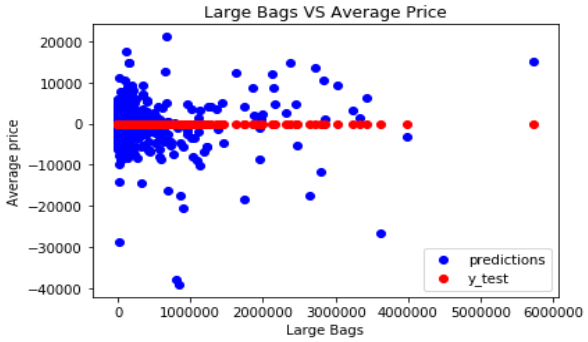

Fig 46: Predicted and y_test Vs largeBags attribute

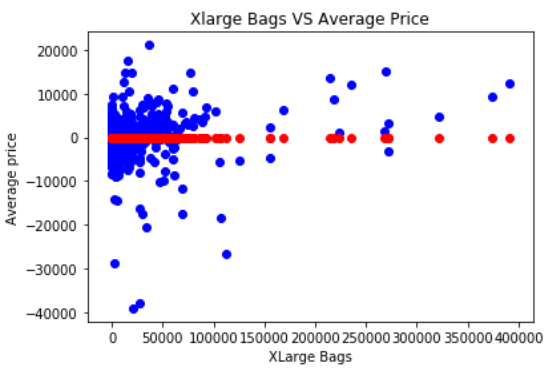

Fig 47: Predicted and y_test Vs XlargeBags attribute

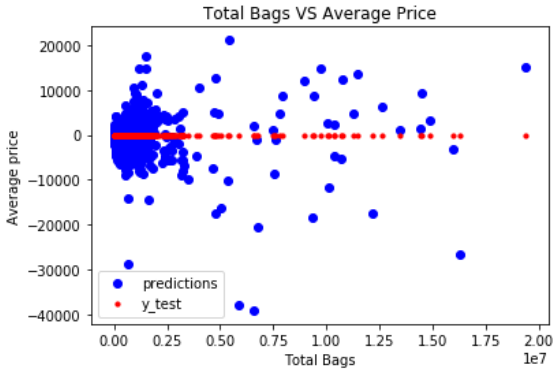

Fig 48: Predicted and y_test Vs TotalBags attribute

In the above figures we have plotted both y_test and predicted values versus with all the input parameters of the dataset. We can observe that there is a lot variation between them in all the figures. The accuracy of this model is less than the previous model so we cannot consider this model.

Training 4 :

MLPRegressor(hidden_layer_sizes=(50,30,20,10),solver ='sgd',max_iter=1000,batch_size=500,verbose=True)

In this model we got a convergence issue so we couldn't construct a model. That is the reason we don't have any accuracy calculation for this.

Training 5:

MLPRegressor(hidden_layer_sizes=(10,10),solver='lbfgs', max_iter=500,verbose=1)

Error:

```
MAE:  1.433833706019369
MSE:  2.2547651678093357
RMSE:  1.5015875491656607
```

Fig 49 : Error values for training 5

In the above figure we can see that the error is significate low when compared to all the above models.

Graph:

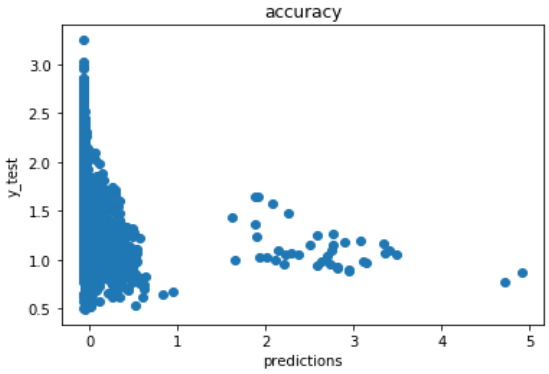

Fig 50 : predicted value Vs y_test

In the above figure we can see that the y_test and predicted values are a bit near (by seeing the scale on both the axis).

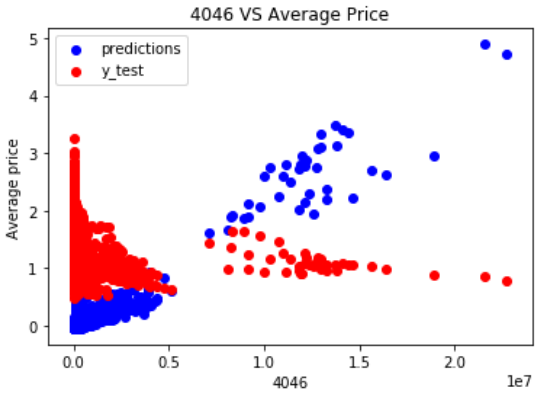

Fig 51: Predicted and y_test Vs 4046 attribute

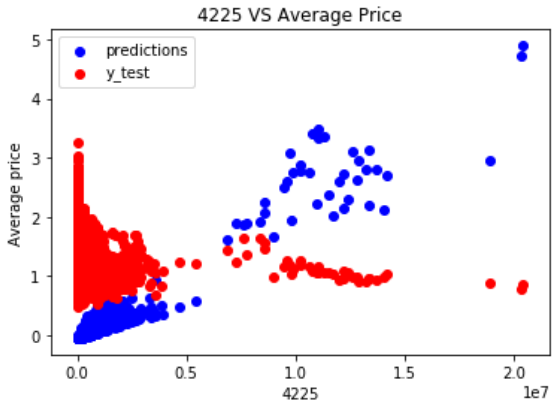

Fig52: Predicted and y_test Vs 4225 attribute

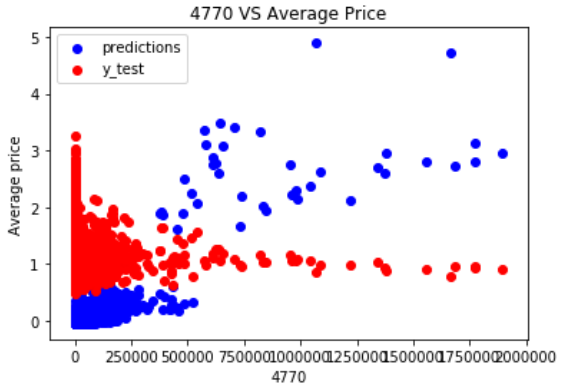

Fig 53: Predicted and y_test Vs 4770 attribute

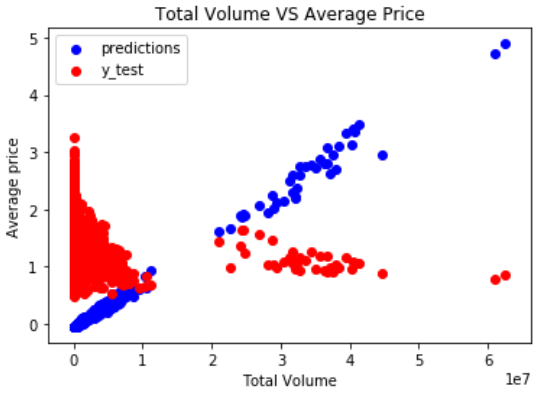

Fig 54: Predicted and y_test Vs TotalVolume attribute

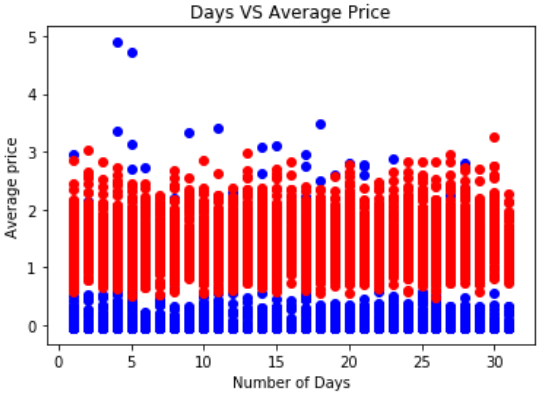

Fig 55: Predicted and y_test Vs Days attribute

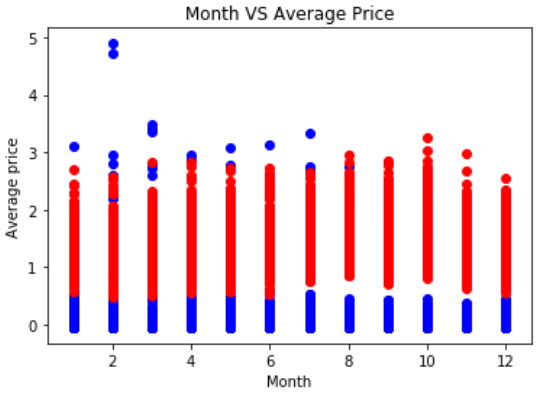

Fig 56: Predicted and y_test Vs Months attribute

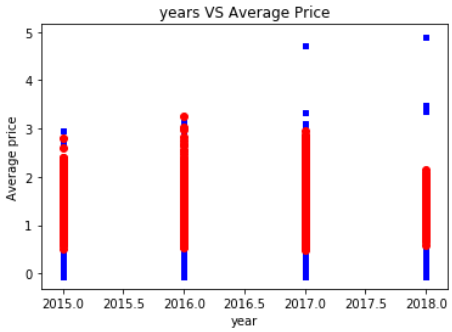

Fig 57: Predicted and y_test Vs years attribute

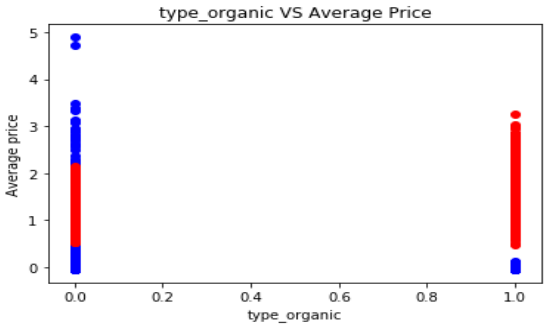

Fig 58: Predicted and y_test Vs organic attribute

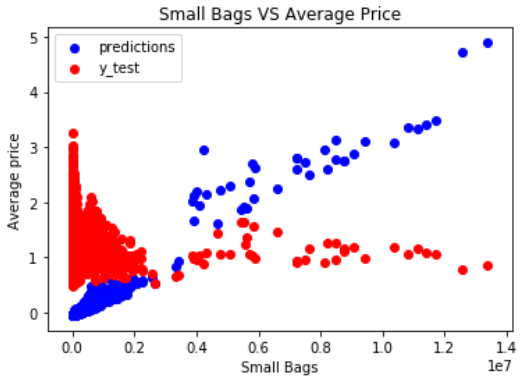

Fig 59: Predicted and y_test Vs SmallBags attribute

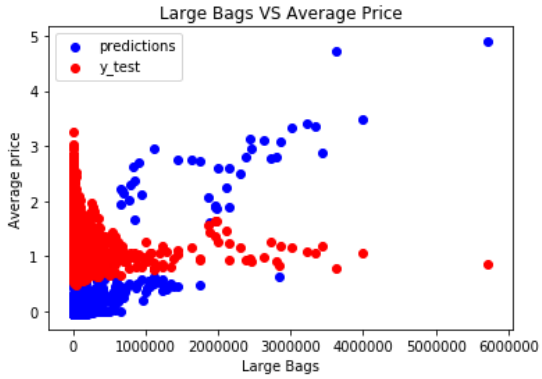

Fig 60: Predicted and y_test Vs largeBags attribute

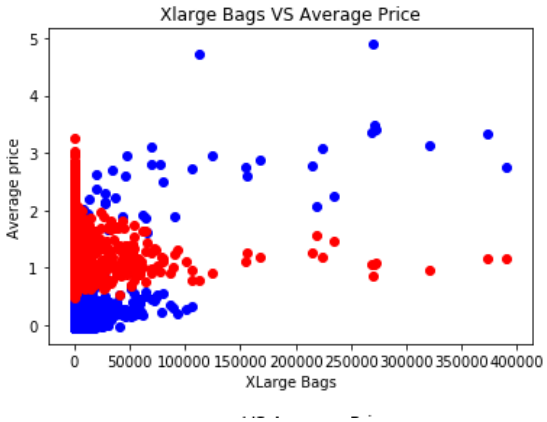

Fig 61: Predicted and y_test Vs XlargeBags attribute

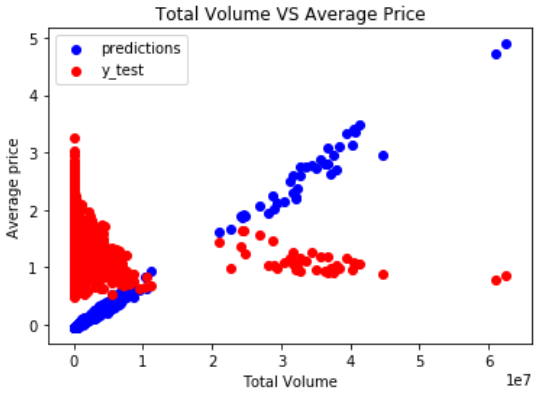

Fig 62: Predicted and y_test Vs TotalBags attribute

In the above figures we have plotted both y_test and predicted values versus with all the input parameters of the dataset. We can observe that there is a less variation between them in all the figures. The accuracy of this model is more than the all the other models we can consider this as a final model but we can check it out with more layer of neurons and compare the accuracy.

Training 6:

MLPRegressor(hidden_layer_sizes=(50,40,30,20,10),solver='lbfgs', max_iter=10000,verbose=1)

Error:

```
MAE:  0.3161620026966539
MSE:  0.15542006096798555
RMSE:  0.39423351071159024
```

Fig 63: Error values for training 6

In the above figure we can observed the error of this model is very less when compared to all the models which we have tuned above. So, we will be considering this as the final model of the neural networks.

Graphs:

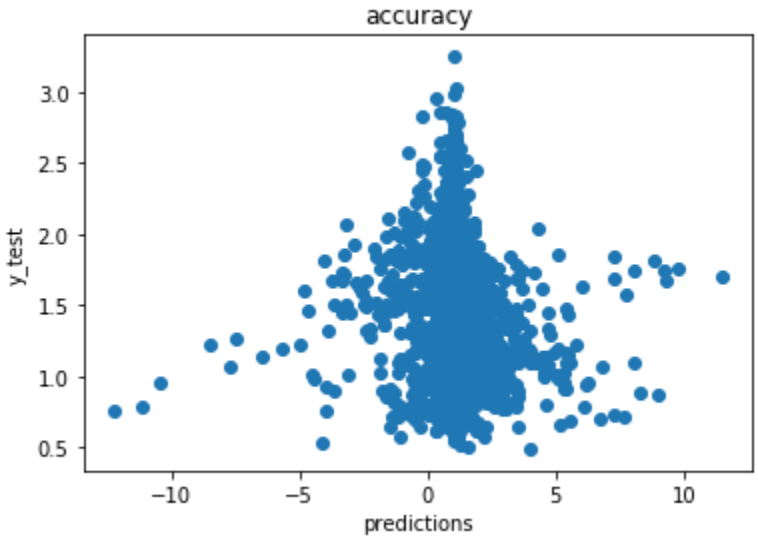

Fig 64 : predicted values Vs y_test

In the above figure we can see that the values of predicted and the y_test are almost same.

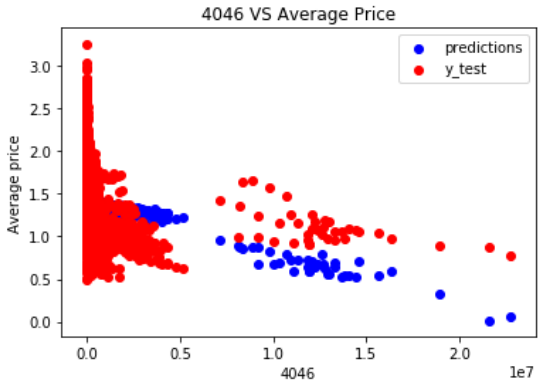

Fig 65: Predicted and y_test Vs 4046 attribute

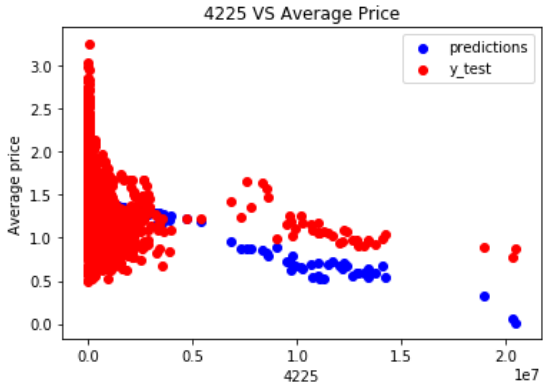

Fig 66: Predicted and y_test Vs 4225 attribute

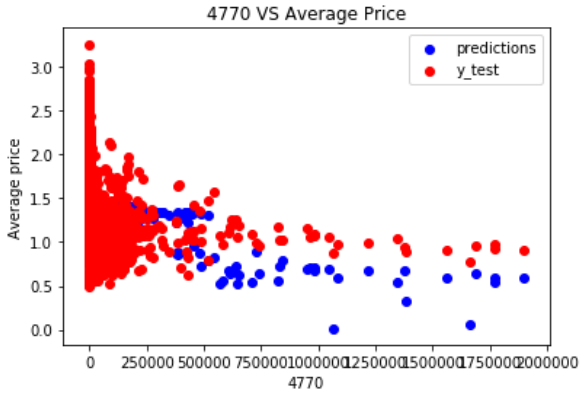

Fig 67: Predicted and y_test Vs 4770attribute

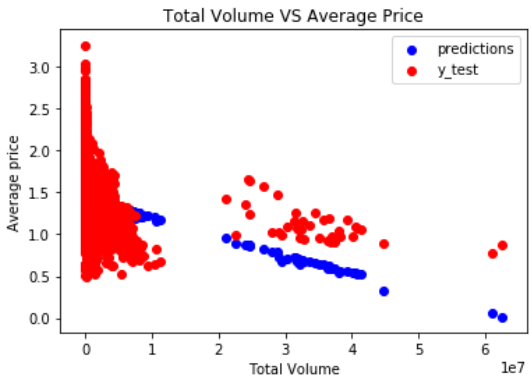

Fig 68: Predicted and y_test Vs AveragePrice attribute

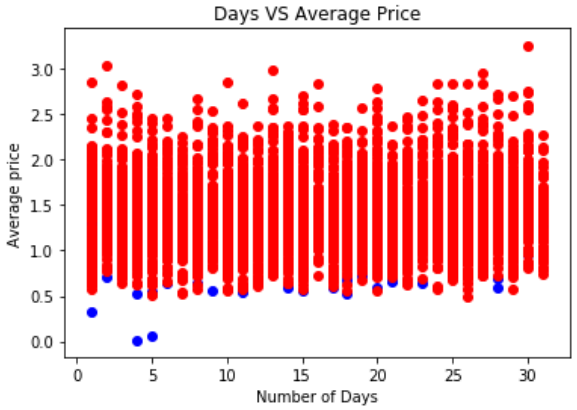

Fig 69: Predicted and y_test Vs XlargeBags attribute

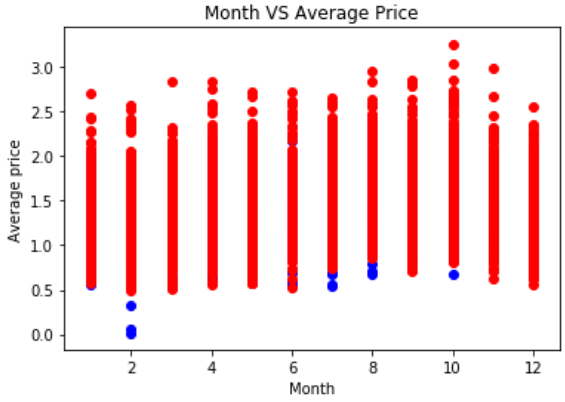

Fig 70: Predicted and y_test Vs Month attribute

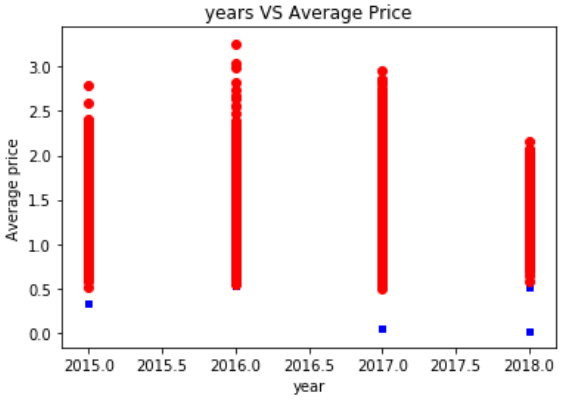

Fig 71: Predicted and y_test Vs years attribute

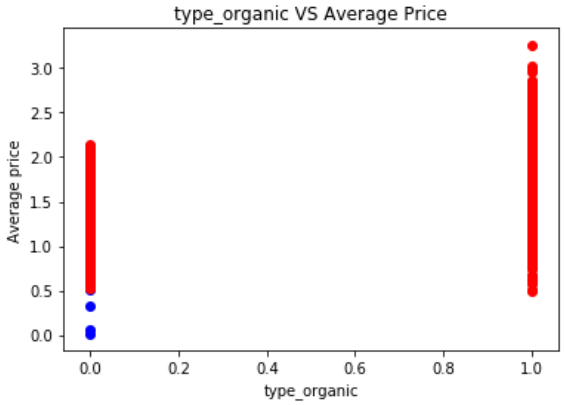

Fig 72: Predicted and y_test Vs organic attribute

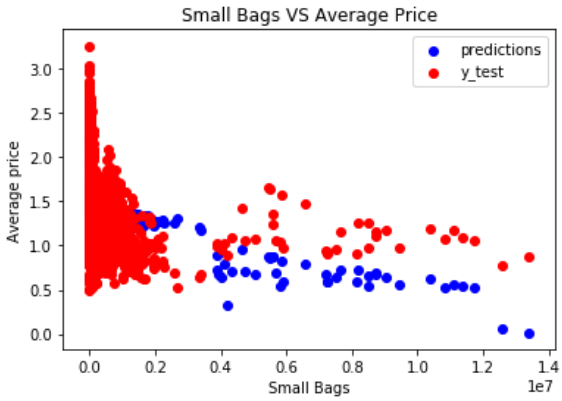

Fig 73: Predicted and y_test Vs SmallBags attribute

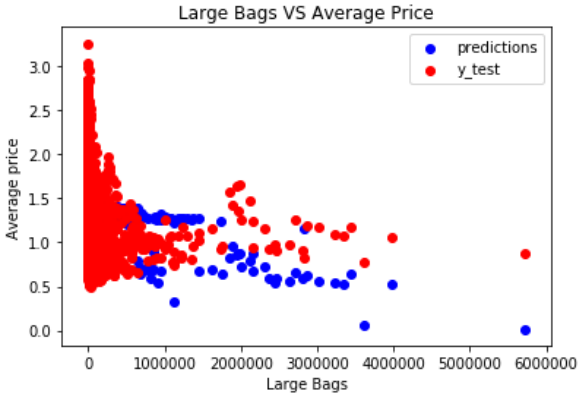

Fig 74: Predicted and y_test Vs largeBags attribute

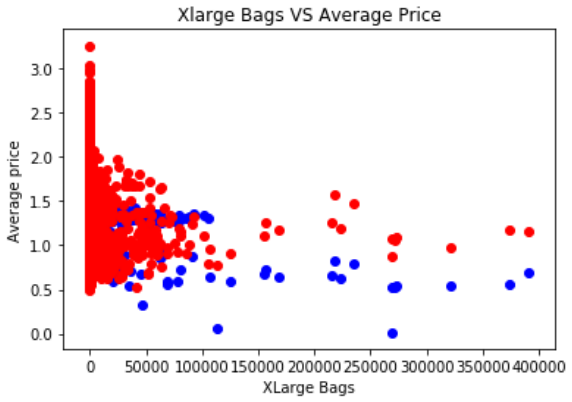

Fig 75: Predicted and y_test Vs XlargeBags attribute

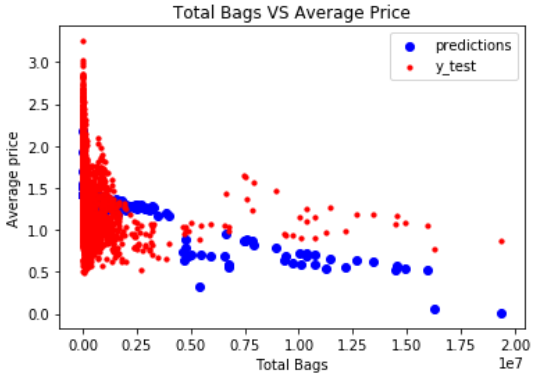

Fig 76: Predicted and y_test Vs XlargeBags attribute

In the above figures we have plotted both y_test and predicted values versus with all the input parameters of the dataset. We can observe that there is a very less variation between them in all the figures. The accuracy of this model is more than the all the other models we can consider this as a **final model for the neural networks**.

Now we can print all the attributes of this model such as weights, bias.

```
print("weights of the model: " +str(mlp.coefs_))
print("bias of the model: " +str(mlp.intercepts_))
print("loss computed using the current loss function: " +str(mlp.loss_))
print("The number of iterations the solver has ran: " +str(mlp.n_iter_))
print("Number of layers: " +str(mlp.n_layers_))
print("Number of outputs: " +str(mlp.n_outputs_))
```

```
weights of the model: [array([[-1.32893743e-01,  3.59489219e-02, -1.85271449e-01,
        -8.04547584e-02, -4.85538057e-02,  1.15586320e-01,
        -2.82011097e-01,  3.21609608e-01, -2.34834387e-01,
        -1.79506359e-01,  8.17603001e-02, -1.31578076e-01,
         2.18154319e-01, -2.24607379e-01, -7.12262097e-02,
        -1.65296488e-01,  2.79881002e-01, -1.34530203e-01,
        -2.74451682e-01, -2.13278717e-01,  1.67343367e-01,
        -2.71827271e-01, -2.93162263e-01, -2.36000969e-01,
        -1.96294777e-01, -2.18974752e-01, -2.19301948e-01,
        -1.57503275e-01,  1.96296502e-01, -2.05894997e-01,
        -7.33235338e-02, -1.25736245e-02, -4.06306045e-01,
         4.03500967e-01, -8.34739132e-02, -2.74017765e-01,
        -3.00388717e-01, -2.71654285e-01, -2.18059324e-01,
        -1.74727737e-01, -2.93963705e-01,  2.45377809e-03,
        -6.92271940e-03, -2.08168811e-01, -1.08407694e-01,
        -2.46837493e-01,  1.50512430e-02, -2.30579726e-01,
        -2.66598130e-01, -2.78403935e-01],
       [ 4.86598816e-02,  2.00537704e-01,  6.66234866e-03,
        -2.39564504e-01, -2.18635972e-01,  2.65225480e-01,
        -2.34370547e-02,  2.89388754e-01,  6.74025780e-03,
         1.68718449e-02, -1.12956260e-02,  5.62321233e-02,
         5.17912815e-02, -3.66175395e-02, -8.06497939e-02,
```

Fig 77: Some weights of the model

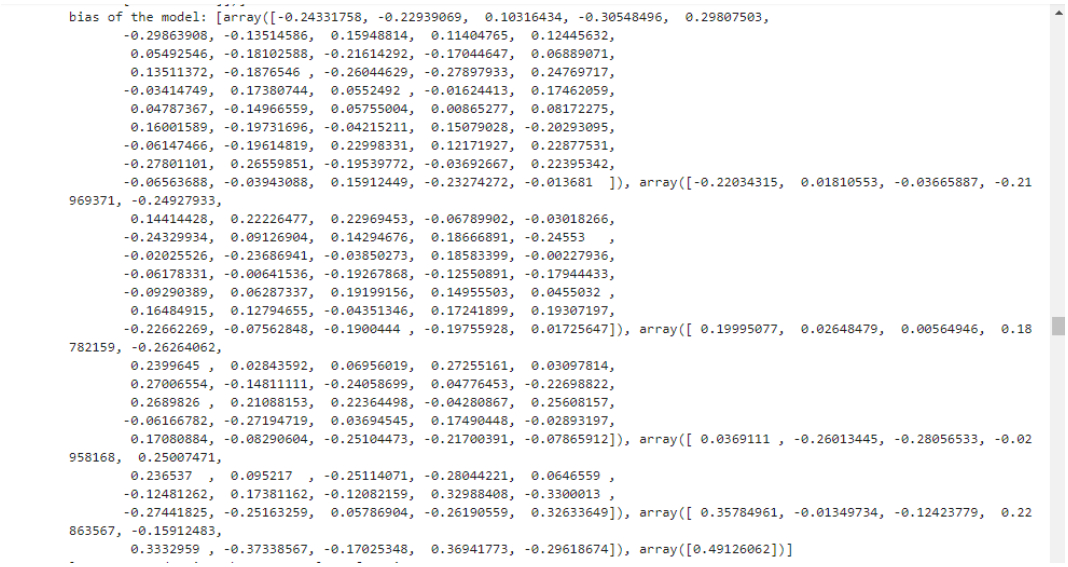

Fig 78: Bias values of the model

```
loss computed using the current loss function: 0.1269964186891202
The number of iterations the solver has ran: 8194
Number of layers: 7
Number of outputs: 1
```

Fig 79: The details of the model

Comparing this result with the models developed in assignment 1:

As I have changed the problem statement, I have developed the models again for this statement.

I have chosen linear regression, Decision Tree and Random forest.

```
print('MAE: ', metrics.mean_absolute_error(y_test,pred))
print('MSE: ', metrics.mean_squared_error(y_test,pred))
print('RMSE: ', np.sqrt(metrics.mean_squared_error(y_test,pred)))
```

```
MAE:  0.2329713329170077
MSE:  0.0910880280536491
RMSE:  0.3018079323902026
```

Fig 80: Error value for the linear regression

```
MAE:  0.13504657534246575
MSE:  0.04374438356164383
RMSE:  0.209151580346991
```

Fig 81: Error values for Decision Tree

```
MAE:  0.10867013698630136
MSE:  0.024654921643835617
RMSE:  0.1570188576058163
```

Fig 82: Error values for Random forest

```
MAE:  0.3161620026966539
MSE:  0.15542006096798555
RMSE:  0.39423351071159024
```

Fig 83: Error values for the final model of the neural network

When we compare all the error values above below is the incremental of the models.

Random Forest < Decision Tree < linear Regression < MLPRegressor(NN)

By the above comparison we can conclude that **Random Forest is the best model for our problem set**.

Code:

Please find the final code of the model attached below

MLPR.txt

References :

https://www.kaggle.com/neuromusic/avocado-prices#avocado.csv

https://en.wikipedia.org/wiki/Multilayer_perceptron

http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor

https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/

Appendix:

Description of each attribute which we have used in this assignment.

Attribute	Description
Day	The day of the observation
Month	The month of the observation
AveragePrice	The average price of a single avocado
Total Volume	Total number of avocados sold
4046	Total number of avocados with PLU 4046 sold
4225	Total number of avocados with PLU 4225 sold
4770	Total number of avocados with PLU 4770 sold
Total Bags	Total number of bags
Small Bags	Number of Small bags
Large Bags	Number of Large bags
XLarge Bags	Number of Xlarge bags
Type_Organic	If it is organic the value is 1 If it is conventional the value is 0
Year	the year of the observation