DS203 ASSIGNMENT
E3

SOURAV SREENATH
24B1841
BTECH-ENGINEERING PHYSICS
INDIAN INSTITUTE OF TECHNOLOGY
BOMBAY

1. **Task 1:**

Section a:

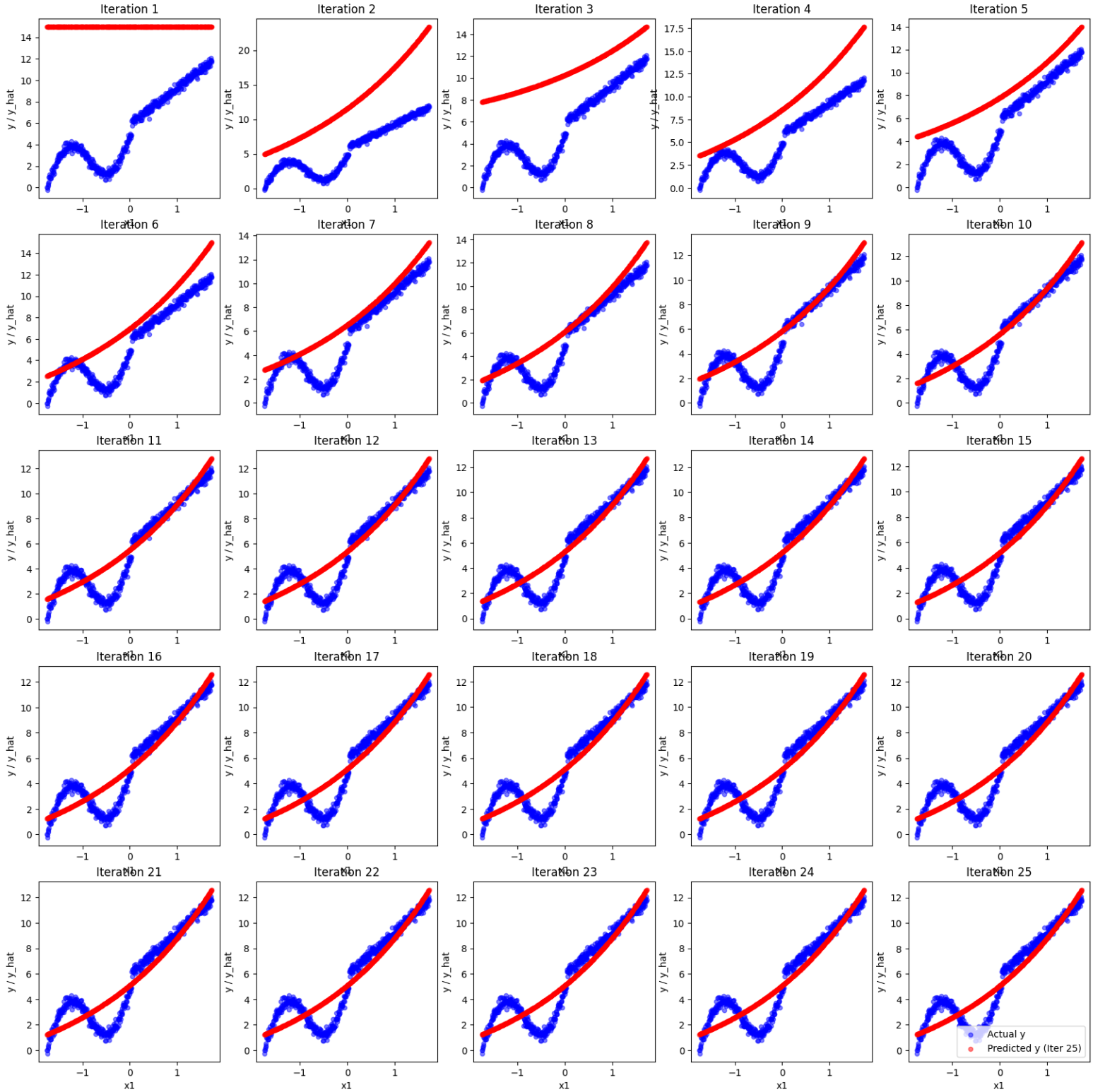
So, in this task, we have been provided with an excel sheet and an .ipynb code. The excel sheet contains the columns x1, x2… , x6 and y. The code uses the pandas library to get a hold of the data for use in the code. The code mainly focuses on the gradient descent method taught in the class during MLR.

An initial condition is set for the weights and then using the gradient descent method, similar to the Newton-Raphson method, through which the slope of the graph is followed to the minimum of the graph. The graph is a 7-D hyperplane, hence we plot the projections of y vs xi onto subsequent graphs.
That is what is done here. Around 25 iterations are done, getting better and better values. Then plots are plotted between y and the selected xi. This can be viewed to get an idea of the convergence. We see the regression line slowly coming toward the data points.

```
Starting coefficient (Beta) vector: [15, 0, 0, 0, 0, 0, 0]
Learning rate: 0.0005
Coefficient (Beta) vector after 25 iterations: [5.69085963
0.30545071 0.43321176 0.53668762 0.61408907 0.66541102
 0.69212419]
```

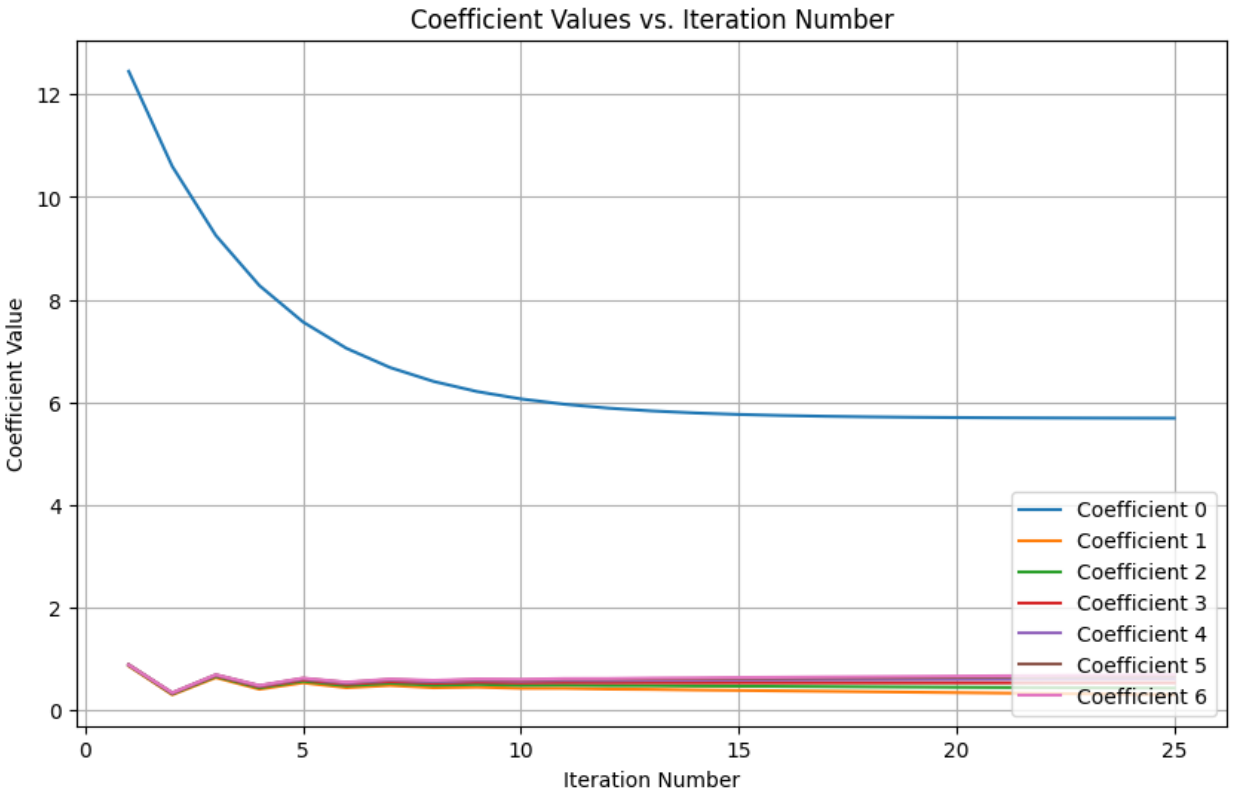
These are the starting coefficients or weights. The weights are set to these initial values and then the gradient descent process is done. The result of the process is quite evident through the plot. The plot reveals slow convergence to the final values of stability, ie: a minima. It is more evident through plots that will be provided later. For now, the below plot gives us an idea on how the graph approaches the curve in a way that the Squared Error values sum is decreased as much as possible.

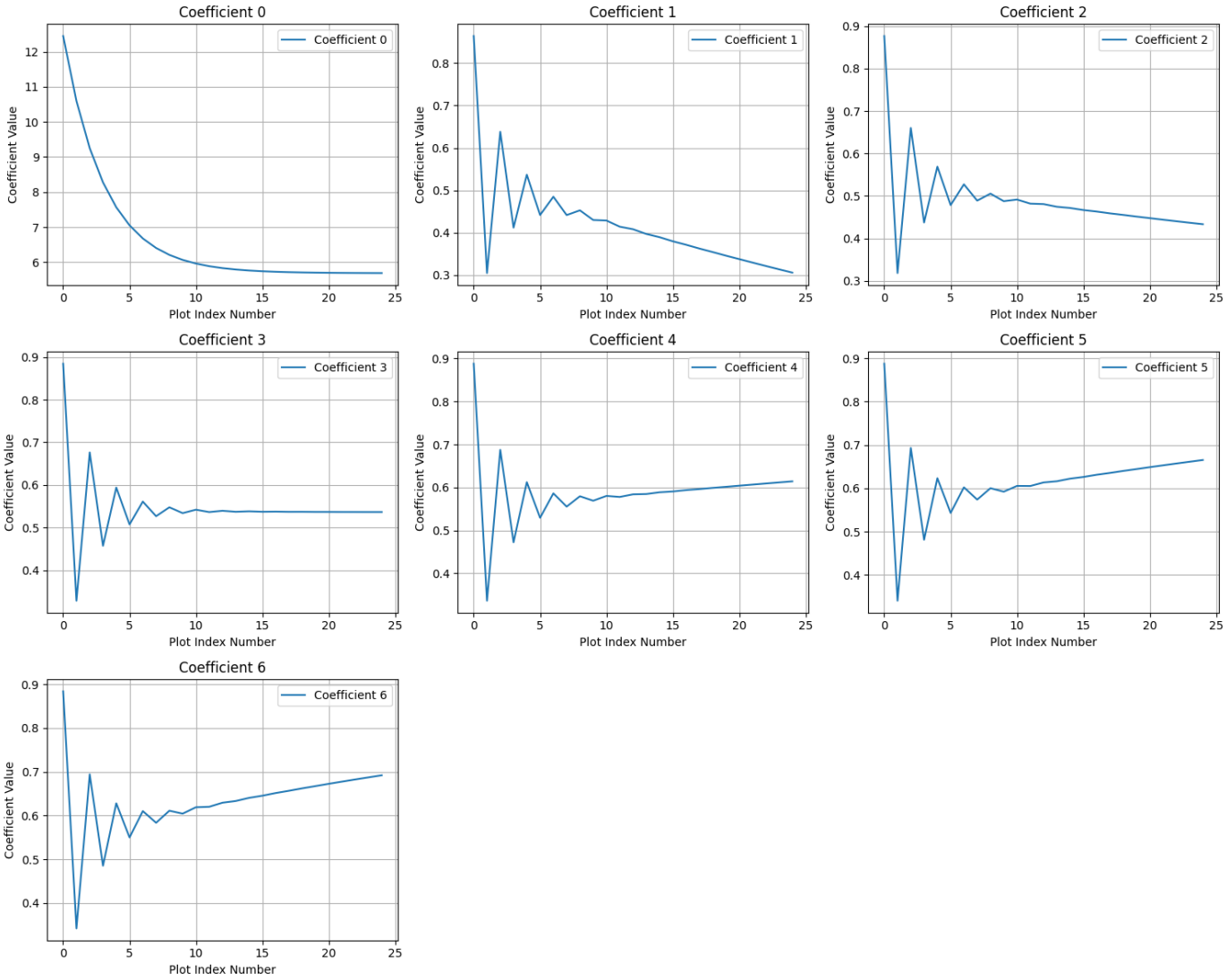
The notebook provides the ability to plot y wrt any feature of our choice. But, I have chosen to keep it at a constant x1 only as the shapes are quite similar in both and didn't seem to change at all in between different features.


MLR using Gradient Descent (y vs. x1) - Plotted Every 1 Iterations
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
Iteration 11
Iteration 12
Iteration 13
Iteration 14
Iteration 15
Iteration 16
Iteration 17
Iteration 18
Iteration 19
Iteration 20
Iteration 21
Iteration 22
Iteration 23
Iteration 24
Iteration 25
y / y_hat
x1
Actual y
Predicted y (Iter 25)

The plots verify the idea that i have given earlier. The plots comes closer and closer to the graph giving a better fit. One interesting observation is that the graph even though its just many SLR put together into one to make the MLR, the prediction plot has some curvature. This is quite interesting. It could be due to the fact that the plot is getting distorted due to the presence of the other features.

There are two more plots generated to show the convergence of the weights. They are as below. In this we can see how the values converge towards certain values.





From these two plots, its quite obvious how, more the iterations the closer the derivative term gets to 0, ie: in the sense more minimal is the error term. We see a slight linear drift towards the end of some of the graphs. I believe that is just the coefficients tending to minimize the error even more or its could be due to correlation too.

Just as an extra note, the coefficient 0 is nothing but the constant term, which gets added to y(ie: the intercept).

Section b:

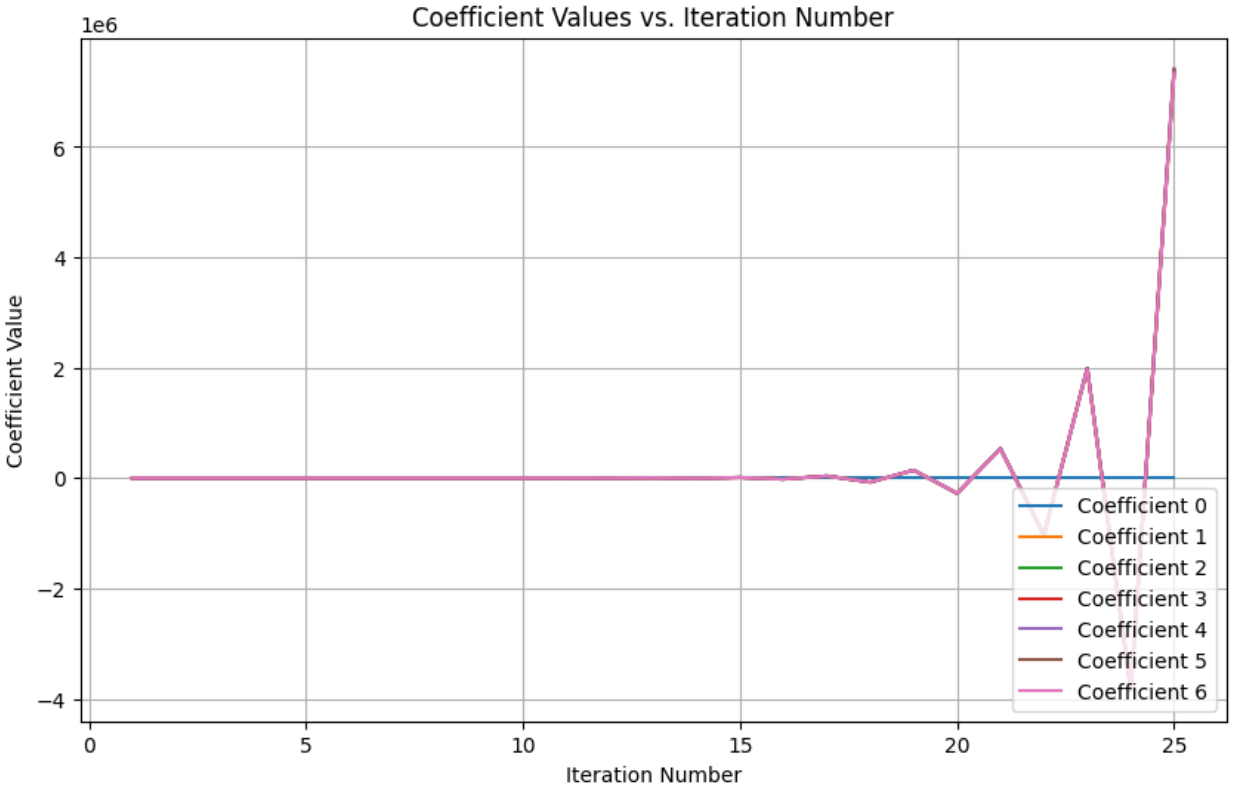
Here the exercise is to change the learning rate. That is change the step size across which the point moves along the derivative to reach the next point.

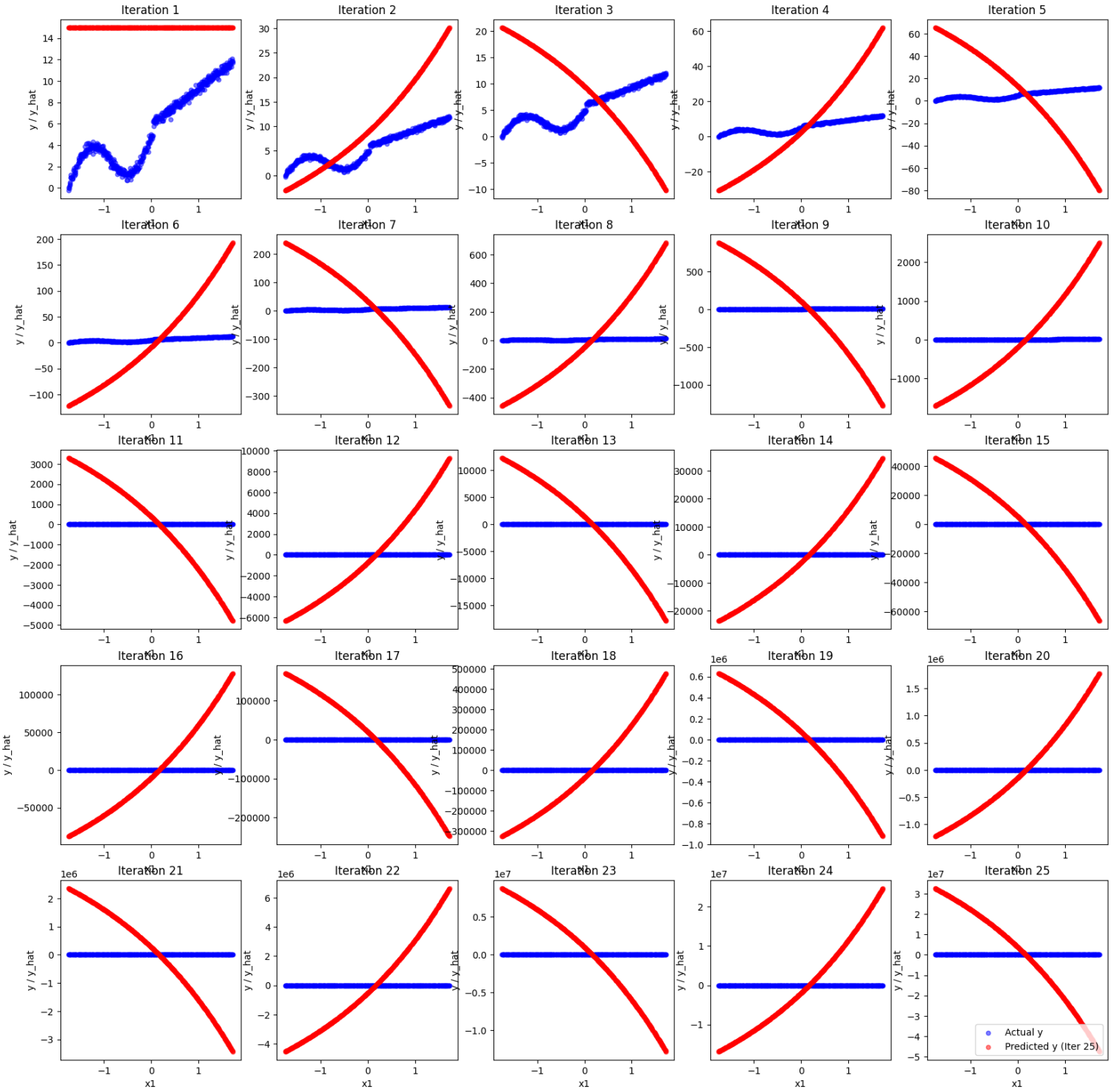
The learning rate is a very intricate quantity. Too little, then we see that the convergence takes a lot of time and iterations. Too much, then we see that the minima is always overshot. In the sense, the step size is too much along the derivative and it always overshoots the minima. So, there is a sweet spot for setting the learning rate.

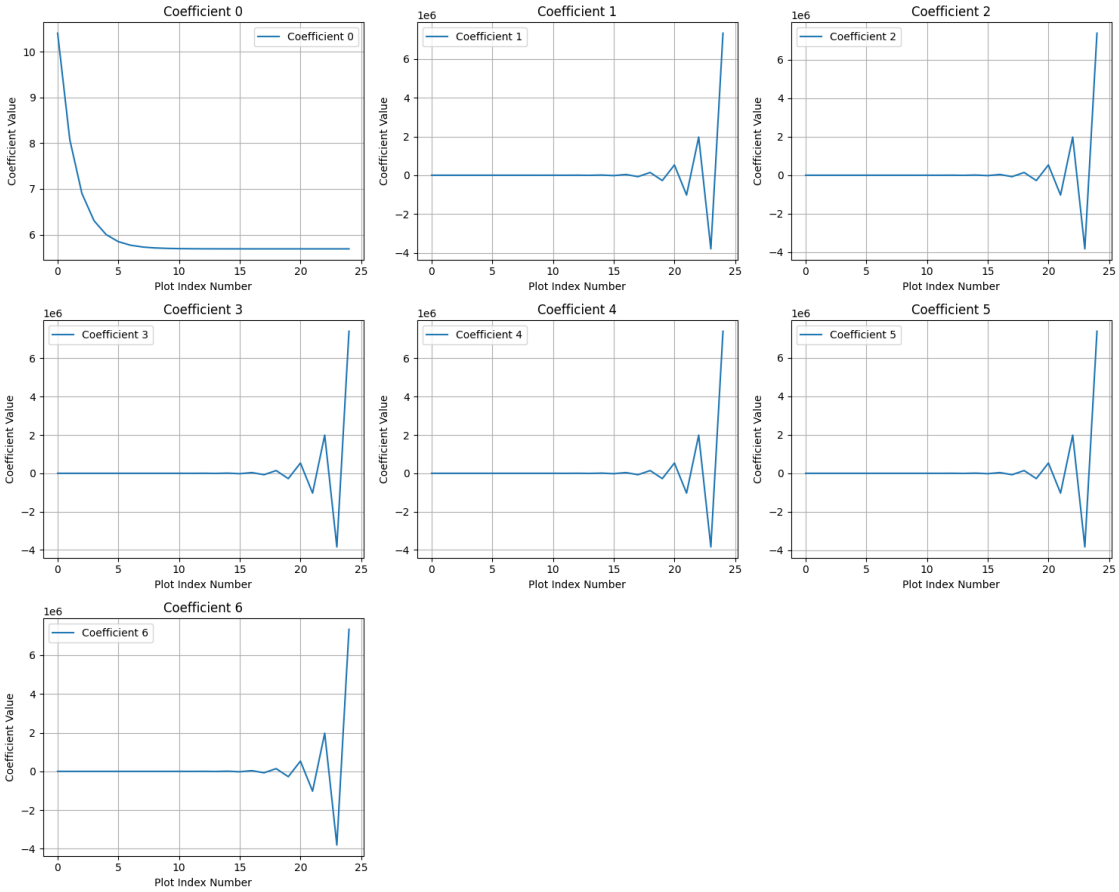
Here, to see the behaviour, as per the question's requirement, I am setting the learning rate to 0.09.

Due to whitespace presence due to the large plot, I will change the order a bit and will continue with my explanation after that.

```
Starting coefficient (Beta) vector: [15, 0, 0, 0, 0, 0, 0]
Learning rate: 0.0009
Coefficient (Beta) vector after 25 iterations: [5.68775210e+00
7.33036512e+06 7.38634316e+06 7.41246768e+06
 7.41104926e+06 7.38540295e+06 7.33937673e+06]
```




MLR using Gradient Descent (y vs. x1) - Plotted Every 1 Iterations
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
Iteration 11
Iteration 12
Iteration 13
Iteration 14
Iteration 15
Iteration 16
Iteration 17
Iteration 18
Iteration 19
Iteration 20
Iteration 21
Iteration 22
Iteration 23
Iteration 24
Iteration 25
y / y_hat
x1
Actual y
Predicted y (Iter 25)



Here, it is quite clear what happens. The learning rate increased too much and now the minima is just simply getting overshot. After the first iteration, it lands in a place with a big slope. This in combination with the larger learning rate means the overshoot only keeps on increasing from there. We see that then the coefficients get larges and larger and then they dont even come close to the graph. The coefficient 0 here converges properly here as the graph's starting point is still the same as the minimization property applies. Rest coefficients diverge due to the overshoot + minimization property. If the coeff 0 oscillated the minimization wouldnt happen, that's why coeff 0 converges.

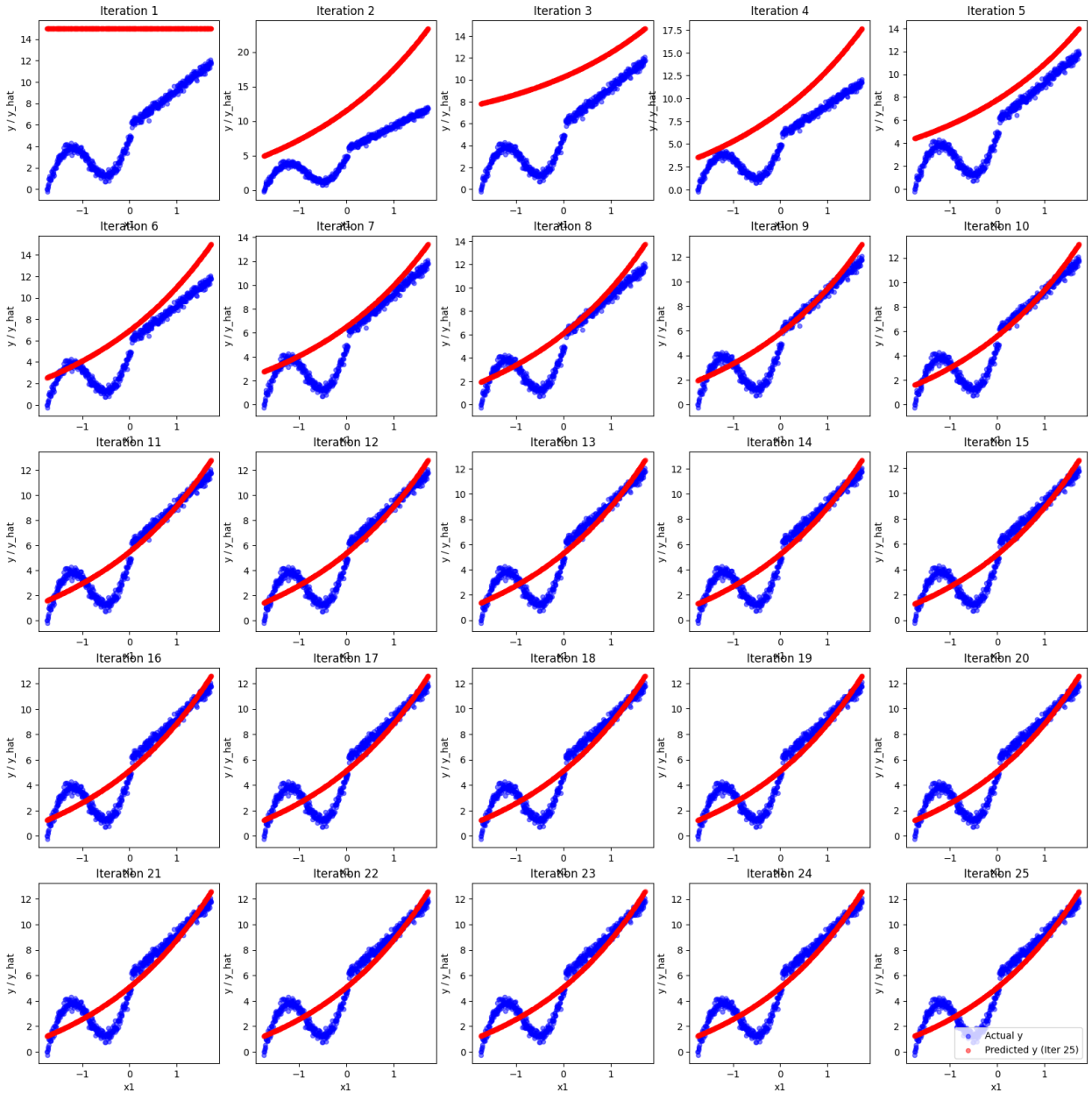
This is quite evident from the 2nd diagram with the 25 iterations, where towards the end, the y axis is set to be at 10^6 that our sample looks like a straight line on the x-axis. This is quite obvious of overshooting. This means our coefficients explode to large numbers. This is the result we get from the graph.
This means a learning rate of 0.0009 is too large a step size whereas the earlier used 0.0005 is a good balance.

Section c:

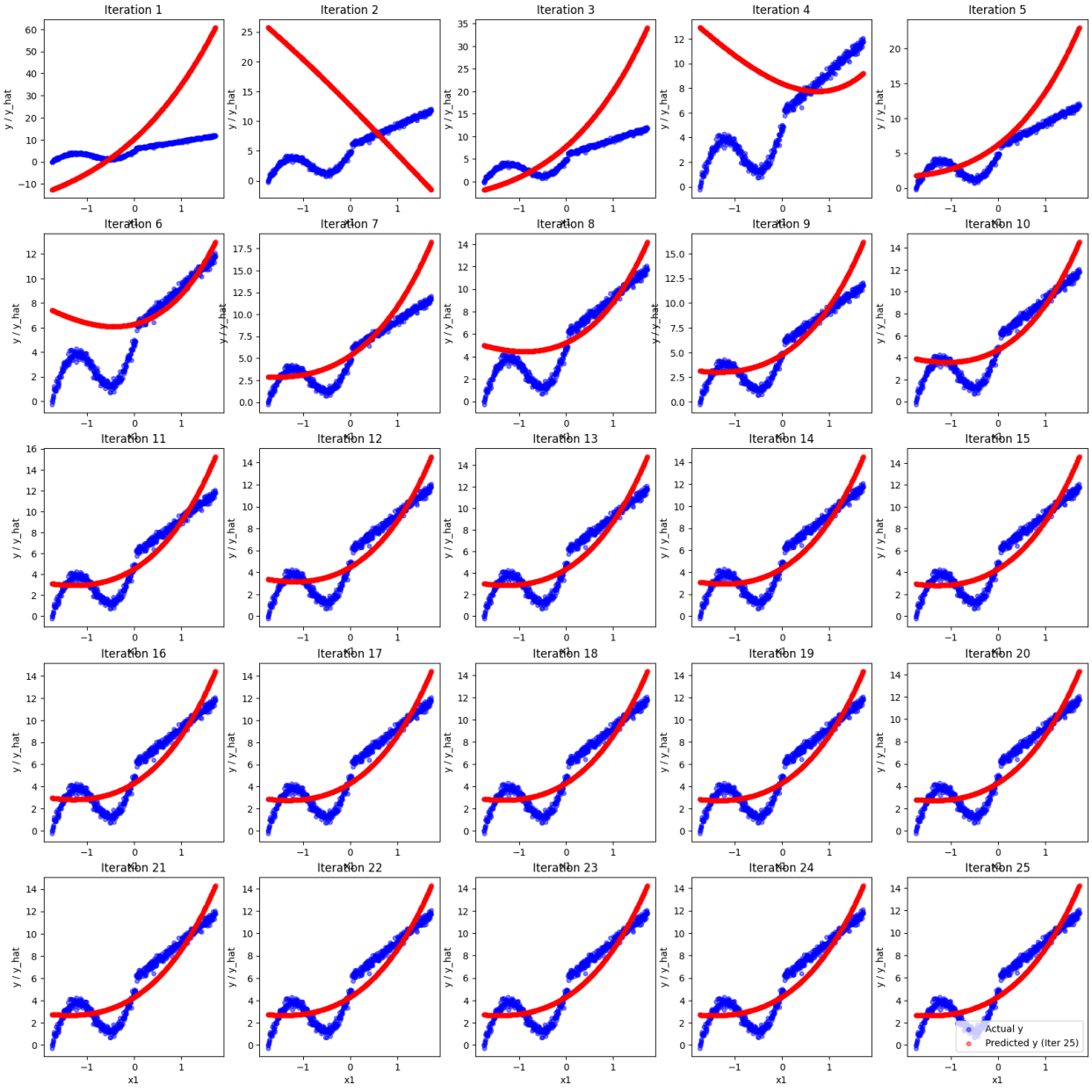
Here the experiment I am doing is choosing different initial points and looking at the y vs x1 plots and the convergence values.
The initial points have been provided in the .ipynb file itself and I am just uncommenting the necessary ones.

```
Starting coefficient (Beta) vector: [15, 0, 0, 0, 0, 0, 0]
Learning rate: 0.0005
Coefficient (Beta) vector after 25 iterations: [5.69085963
0.30545071 0.43321176 0.53668762 0.61408907 0.66541102
 0.69212419]
```

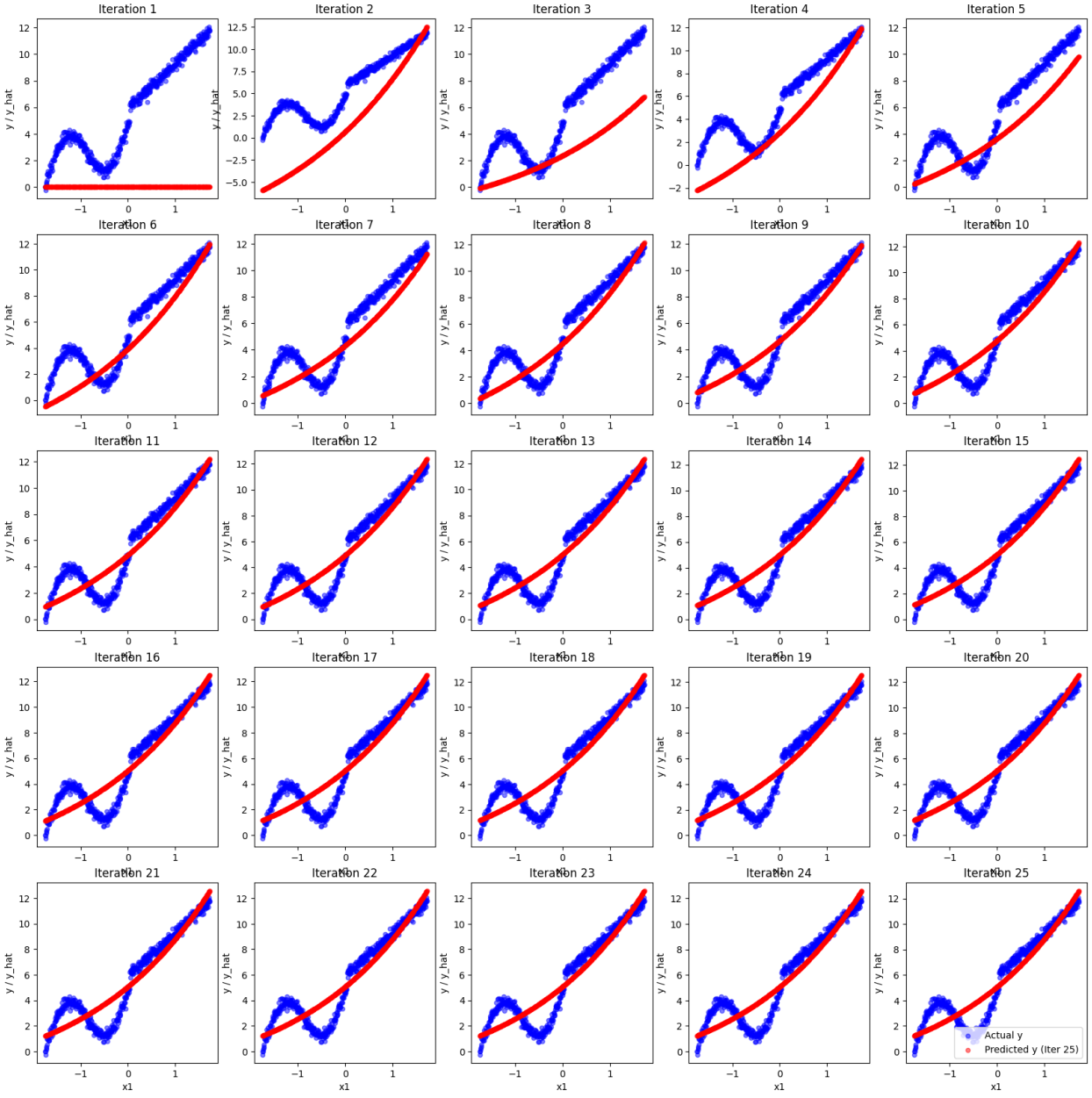
MLR using Gradient Descent (y vs. x1) - Plotted Every 1 Iterations



```
Starting coefficient (Beta) vector: [15, 1, 2, 3, 4, 5, 6]
Learning rate: 0.0005
Coefficient (Beta) vector after 25 iterations: [ 5.69085963 -
1.2982489  -0.55949582  0.17381419  0.90594332  1.64102482
  2.38278308]
```



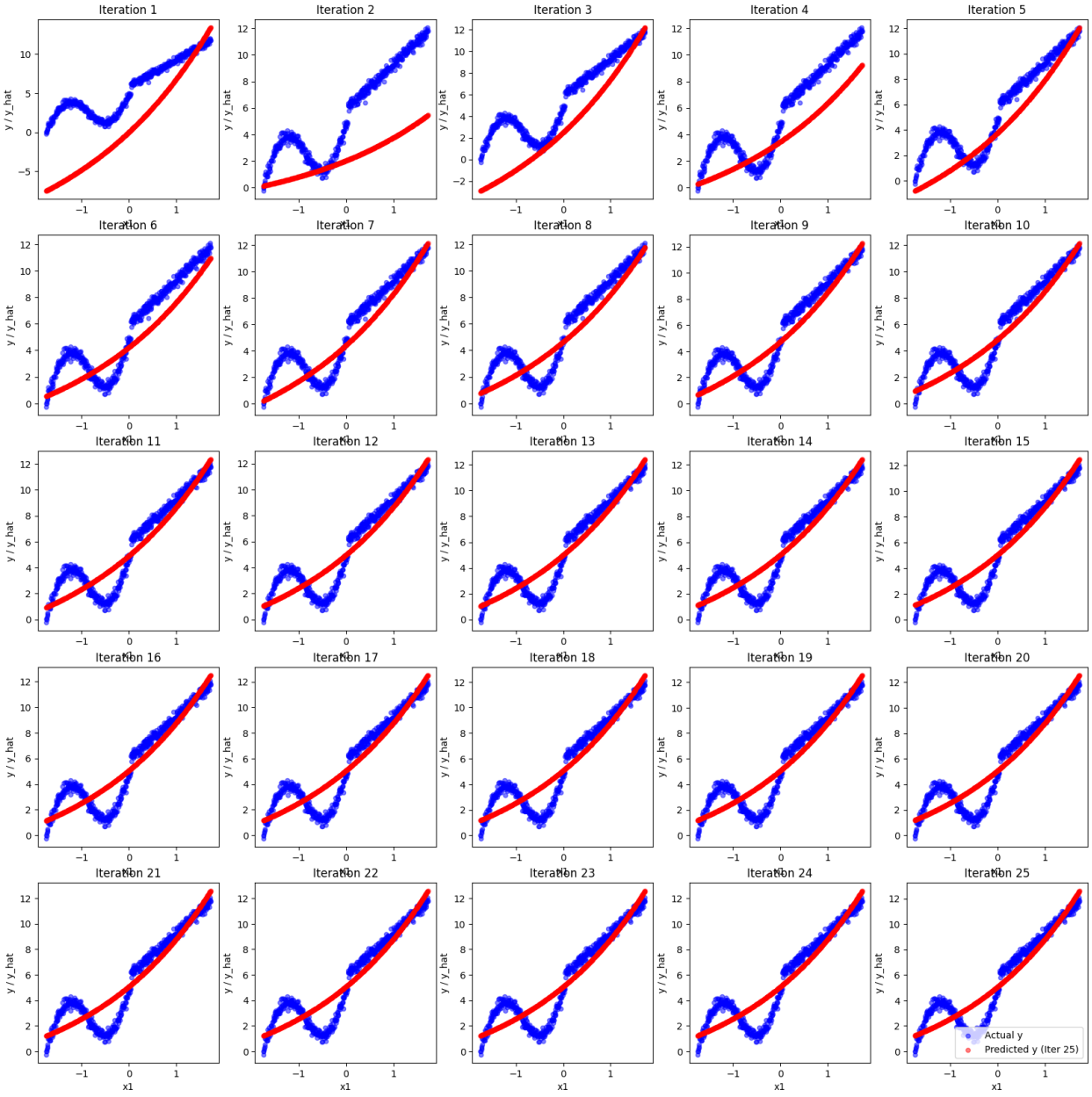
```
Starting coefficient (Beta) vector: [0. 0. 0. 0. 0. 0. 0.]
Learning rate: 0.0005
Coefficient (Beta) vector after 25 iterations: [5.68585345
0.30545071 0.43321176 0.53668762 0.61408907 0.66541102
 0.69212419]
```




```
Starting coefficient (Beta) vector: [1. 1. 1. 1. 1. 1. 1.]
```

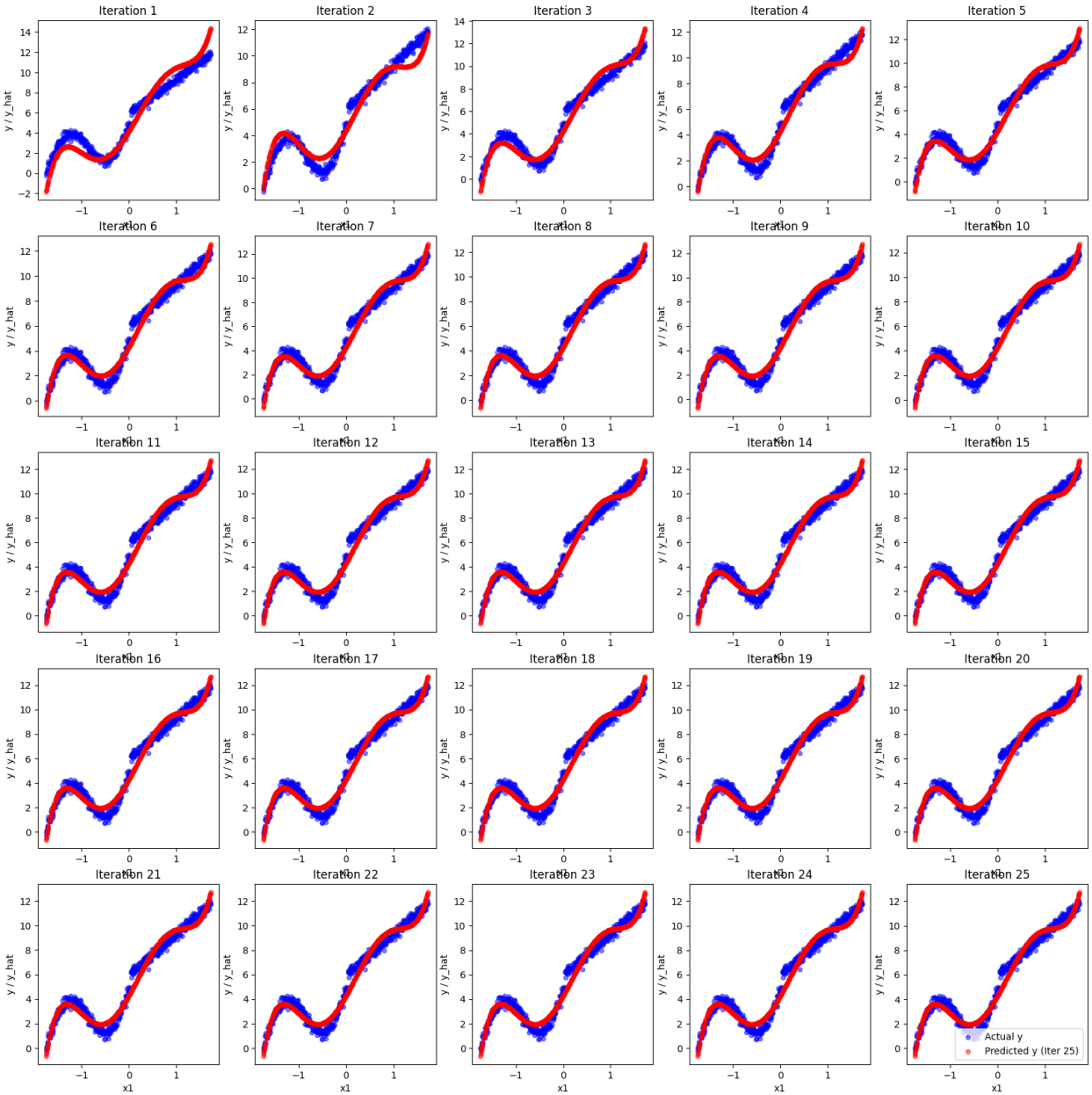
```
Learning rate: 0.0005
Coefficient (Beta) vector after 25 iterations: [5.68618719
0.31163596 0.43190554 0.53192941 0.60960375 0.66447337
 0.69748713]
```

MLR using Gradient Descent (y vs. x1) - Plotted Every 1 Iterations



```
Starting coefficient (Beta) vector: [5.6877, 9571, -32361, 33075, 0,
-17615, 7334]
Learning rate: 0.0005
Coefficient (Beta) vector after 25 iterations: [ 5.68775169e+00
9.57080992e+03 -3.23611683e+04  3.30748539e+04
 -1.24048461e-01 -1.76151025e+04  7.33391824e+03]
```

MLR using Gradient Descent (y vs. x1) - Plotted Every 1 Iterations



I have included only the iteration graphs and the convergence values. I have done this because the results from these plots and data give a good enough idea on what is happening. Here, the idea is choosing different initial points and seeing how the final graph looks like. We see that the results are quite interesting.

The first 4 choices give similar results after the first 25 iterations, whereas the last one is an outlier. It gives massively varying results.

The last one seems like a better fit than the others but in reality it feels as though it is a worser result. The errors might be better in the last case, but the large coefficients just scream overfitting. It might be attaching to the data too much or maybe even its utilizing the collinearity in the data in a very bad manner, in the sense it might over-doing the predictions with the collinearity. But, one thing is for sure, even though the model looks better, the large coefficients reveal a huge red flag in how the model might be an overfit one.

2. Task 2:

Section a:

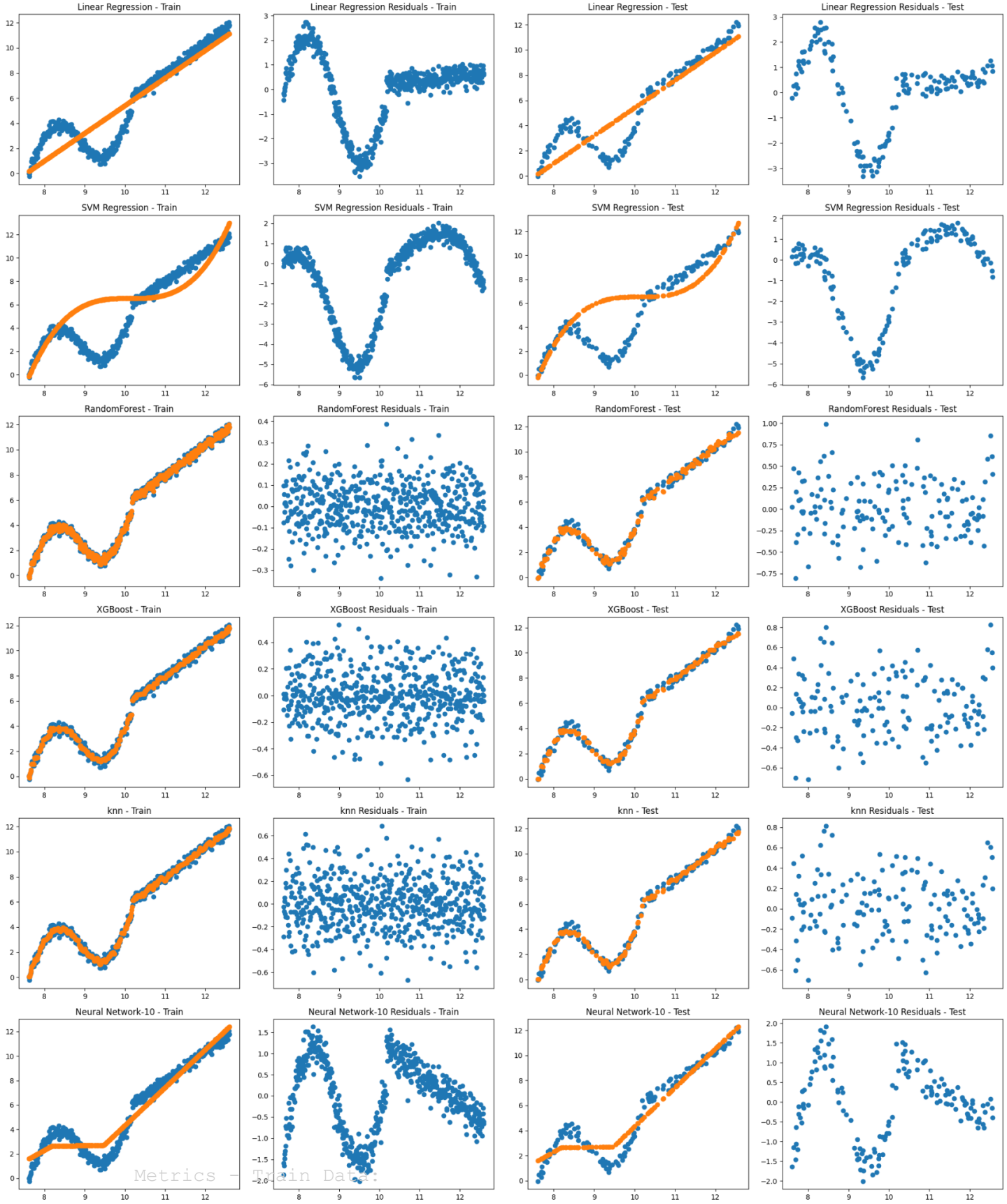
This question deals with the use of many different regression techniques to see which one is the most apt to fit onto a particular data set with only one feature and the y variable given.

The code extracts the required data and passes it through various classification and regression models. These are taken from the sklearn library.

The models used are:

- Linear regression
- SVM Regression
- Random Forest
- XGBoost
- Knn
- Neural Network-10

The results for each of these are given below for understanding the results.


Linear Regression - Train
Linear Regression Residuals - Train
Linear Regression - Test
Linear Regression Residuals - Test
SVM Regression - Train
SVM Regression Residuals - Train
SVM Regression - Test
SVM Regression Residuals - Test
RandomForest - Train
RandomForest Residuals - Train
RandomForest - Test
RandomForest Residuals - Test
XGBoost - Train
XGBoost Residuals - Train
XGBoost - Test
XGBoost Residuals - Test
knn - Train
knn Residuals - Train
knn - Test
knn Residuals - Test
Neural Network-10 - Train
Neural Network-10 Residuals - Train
Neural Network-10 - Test
Neural Network-10 Residuals - Test
Metrics - Train Data:

Metrics - Train Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.833674	1.408923	0.059761	51.466506	6.670987e-12
SVM Regression	0.545415	2.329245	0.021942	68.878362	1.104724e-15
RandomForest	0.999000	0.109241	2.931659	3.769479	1.518686e-01
XGBoost	0.997146	0.184549	2.372918	1.046859	5.924852e-01
knn	0.995929	0.220416	2.510498	0.705376	7.027965e-01
Neural Network-10	0.939374	0.850625	0.162938	22.918092	1.055357e-05

Metrics - Test Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.818068	1.501800	0.079118	12.500550	0.001930
SVM Regression	0.562991	2.327567	0.034813	23.409392	0.000008
RandomForest	0.991747	0.319861	1.834557	0.904898	0.636068
XGBoost	0.992842	0.297890	1.851431	1.003285	0.605535
knn	0.992994	0.294699	1.971390	0.972202	0.615020
Neural Network-10	0.932186	0.916890	0.203631	3.434655	0.179545

These are the results that i got from one run of the code. The above table is the table containing the metrics calculated for each of the models. The observations also includes a plot of the 6 models. From that we can get a qualitative idea of how well they perform based on how well the predicted data matches with the predicted data. In both the plot and the table, the results of the metrics on the test data and the plots with the test data are also included.

The metrics given are R^2 which is preferred to be as close to 1 as possible without overfitting issues. RMSE is the error related metric, this is a metric we need to reduce as much as possible. Closer to 0 the better.

Durbin-Watson is another metric we can use that lies in the range 0-4. Its preferable to have the DW statistic as close to 2 as possible. Even 1.5-2.5 is acceptable under standard conditions.

Jarque-Bera is another metric we use. It is a metric that when closer to 0, the better. We would like to reduce this statistic as much as possible.

The JB p-value is another metric coming out of the JB test itself. It is preferable to have its value >0.05.

A detailed value comparison will be done in section b) as the question specifically asks it.

But, just from a qualitative idea, just by looking at the graphs and the metric value tables, we see that:

Linear Regression, as its a straight line, it cant get to all the point properly, hence it cannot get a proper fit.
SVM does curve a bit, but it seems as though the curvature wasn't sharp enough, hence, many points were missed out.
Random Forest, XGBoost and knn are the best performing qualitatively speaking. Because they follow the curvature of the model very neatly.
Neural-Network-10 tries to follow the curve but the amplitude of the curve wasn't matched properly there. Hence, this model also fails to model the data properly.

Section b:

Looking at the metric table we can make our decision properly.

Linear Regression:
Here the model is meant to be a straight line and yes from the graph it is as such, but the points are not present in such a manner. The curve is like a sine wave initially and caps out as a line. The model, being a line, misses out on the sine wave like area but captures the 2nd phase: the line properly, at least with some intercept difference due to the effect of the sine wave area.
The R^2 value for the train data is 0.833 and test is 0.81. This is quite low for a model. This means the sum of errors dominate a lot compared to the sum of the y values, which is an issue. The errors must always be small compared to the y values. The RMSE is 1.4 and 1.5 respectively for the train and test data. This is not that bad, but still compared to the other models, it's worse. The DW statistic is approx 0.6 and 0.8 for the train and test respectively. This again is not that great considering the value must be closer to 2. It shows that the residuals have some correlation between them, ie: they are not random. The JB statistic is greater than 10 in both cases, which is abysmally bad. The value must be close to 0. The JB p-value is less than 0.05 in both cases which signifies the model is not that great.

SVM Regression:

Here the model is curved a bit but is not able to map the entire thing properly. Checking the values, we see a lower R^2 than SLR, higher RMSE value than SLR, lower DW than SLR, higher JB than SLR and lower JB p-value than SLR. Even though i did not mention the values specifically, comparing with SLR, we see that, all the statistics of SVM is worser than SLR. It just means it is a bad model.

Random Forest, XGBoost, knn:

Even though these 3 are different, I am including them under the same heading as their stats are very close by. The R^2 values are 0.99 or above for both testing and training, the RMSE values are closeby and their difference is too small to have any major difference. The DW statistic is 2.3-2.7 for all. This is fine for most purposes and the models are not bad. The JB statistic is the only differentiating factor ranging from 0.7-3. But for most real world scenarios, this is not a huge issue. It's close to 0 without a huge difference. The other metrics prove it. Similarly the JB p-value is greater than 0.05 for all meaning the models are good.

Neural-Network-10:

This model also has its issues. It has a good enough R^2 value of 0.93 for both the train and test data. The RMSE is also quite fine with around 0.85-0.9. The DW is 0.16 and 0.2 which are very low meaning the errors/residuals have string correlation and that the pattern has not been extracted out fully. The JB values for the train are very bad but by sheer chance, the JB values on the test data are much better. This isn't a good sign though. Such huge variability in the metrics isn't a sign of a good model. The large JB value in the train data makes it a pretty unreliable model which again can be seen in the model plot.

Based on all the above parameters, it is quite obvious that Random Forest, XGBoost and knn produce the best models and can be stated as the best algorithm. This choice is due the combination of observed metrics, ie: the R^2 value, RMSE, DW, JB and JB p-value stats as we have seen above already.

Choosing one of the above from Random Forest, XGBoost and knn is quite difficult as some lag at one metric while the other leads. But the differences are quite small so as to make a perfect choice between the 3.

Section c:

The code has been structured in a way that a manipulation is done with the dataset before passing it onto the various models.

One thing is that, the standardization of the data, ie: mean=0, SD=1 is done in the code before passing onto the model. This is a good generalization practice for comparison.

Another is that the features before passing onto the models, more features are generated. That means, using the Polynomial function, new features have been generated according to the supplied argument, degree. The degree is initially set to 1, in the polynomial function, hence no new features are generated. Even though for this exercise, I didnt change it, it is a good addition to implement polynomial regression using SLR in an easy way.

Rest structure of the code is quite common, where there are plots and value calculations, etc.

My learning from this exercise are mainly the implementations of the various types of models. We used the sklearn library for all the implementations over here. In addition, one of my major learnings was the implementation of the polynomial function which could prove useful in polynomial regression.

I also learnt the method of analyzing how well a model performs just by looking at the R^2, RMSE, DW, JB and JB p-value. I have learnt how to use them practically to determine what all i could see just by looking at the plot. This is a very important learning as for larger datasets, its just these parameters that matter more than just simple visualization.

3. Task 3:

Many sklearn functions have been used inside this code. Summaries of them are given below. The major functions used are the standardization function, polynomial function, RMSE function and the 6 models. Rest statistics have been calculated using statsmodels.

The explanations have been taken from the internet as the question demands but it is written in the form of my inferences but some wordings have been kept the same as they are quite important.

Function	Explanation
PolynomialFeatures()	Takes the input as an array of features. Then as per the given argument, degree, it creates cross terms and power terms of the features already there. If the argument interaction_only is set to true, only the cross terms of a certain degree are added, whereas power terms of the same feature is not included. The argument include_bias when set to true adds a set of 1s to the array to use intercepts. The output after all of this is simply an array containing the already known features and newly generated features.
StandardScaler()	This is a pre-processing function for data. It takes an array of data-points as the input and standardizes their value to mean=0 and SD=1. If the argument with_mean=false, then the original mean is kept and SD=1, and if with_std=false, then the mean=0 and SD is the original SD. It gives an output array of standardized values.
LinearRegression()	This function fits a straight line model to the given data using the ordinary least squares method. The input is a set of features and a target variable. The argument fit_intercept decides

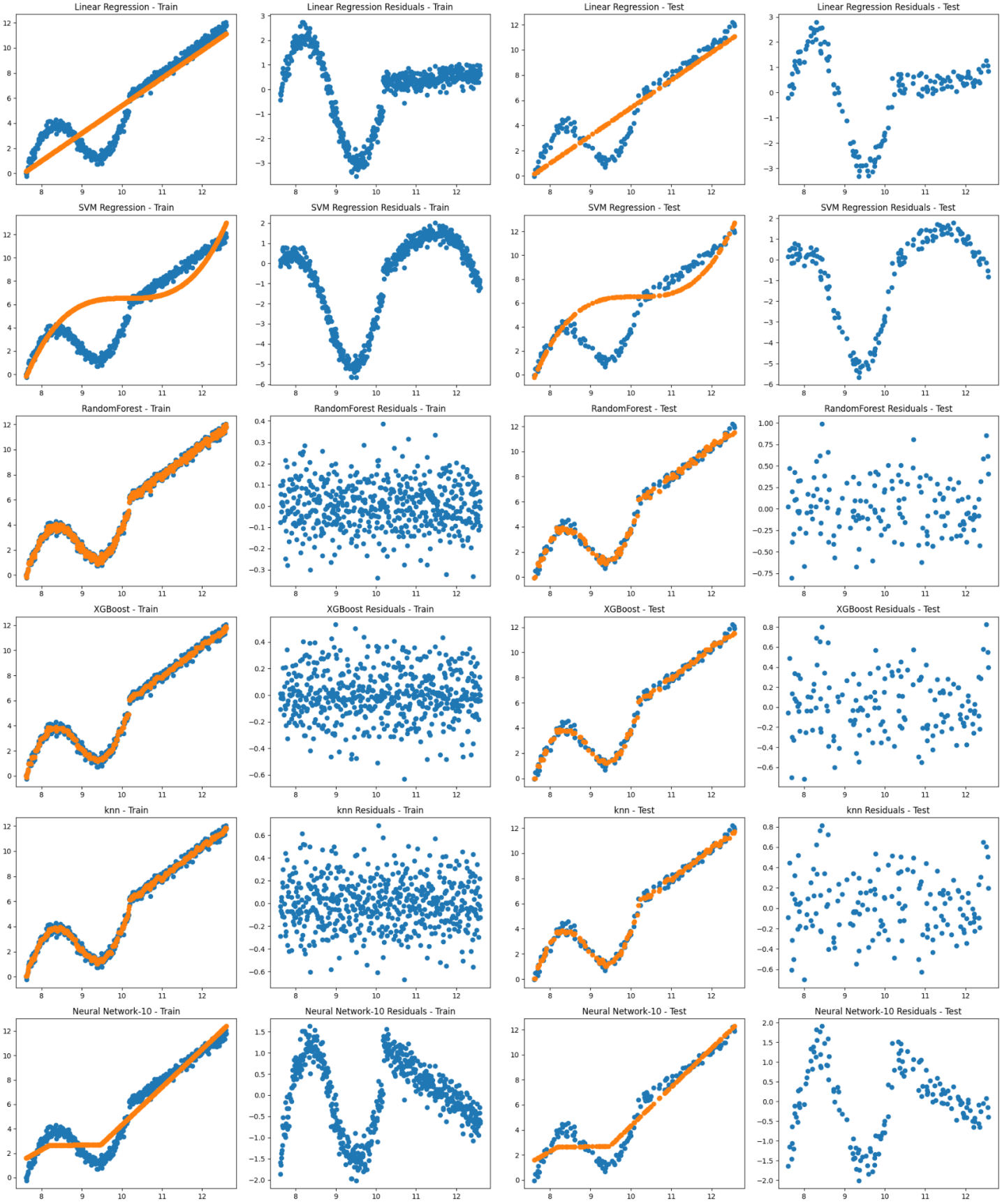
	whether to include a constant term (intercept) in the model. If it is set to false, the model is forced through the origin. The argument copy_X decides whether the original data should be copied or reused. The output is a fitted regression model that gives the feature weights (coefficients), the intercept, and can also be used to make predictions.
SVR()	This stands for Support Vector Regression. It takes the features and target as input and tries to predict continuous values using the support vector machine method. The argument kernel decides the type of mapping used to separate data, for example linear, polynomial, or radial basis function. The argument C controls how much the model tries to avoid errors, with larger values making the fit stricter. Epsilon defines a margin where small errors are ignored. For a polynomial kernel, degree sets the order of the polynomial, while gamma controls how far the influence of a single data point spreads. The output is a regression model that predicts continuous outcomes.
RandomForestRegressor()	This method builds many decision trees and combines their results. The input is the features and target values. The argument n_estimators sets how many trees are built. The argument max_depth decides how deep each tree can grow, while min_samples_split controls the minimum number of points needed to split a node. The argument max_features tells how many features are considered at each split, adding randomness. If bootstrap is true, then

	each tree is trained on a random sample of the data taken with replacement. The final output is an average of the predictions from all the trees. In short, the arguments are just the variables we can change for each an every tree. It is the exact same Random Forest we learnt in class but with control over every part of the model.
GradientBoostingRegressor()	This method builds trees one after the other, where each new tree tries to correct the mistakes made by the previous ones. The input is again the features and target values. The argument n_estimators decides how many boosting steps to perform. The learning_rate controls how much each tree contributes to the final model, with smaller values making the learning slower but more stable. The argument max_depth limits the depth of each tree, while subsample decides the fraction of data to use for each stage, which adds some randomness. The argument loss chooses the type of error measure to minimize, such as squared error. The output is a boosted regression model made up of many trees.
MLPRegressor()	This is basically a neural network that learns patterns in the data. You give it your features and the target values. Inside, it has layers of “nodes” that process the data step by step. The argument hidden_layer_sizes tells how many layers and how big they are, like how many nodes are in each layer. The model keeps adjusting these connections again and again until it learns to predict well. The argument max_iter is how many times

	it tries to learn from the data. The learning_rate decides how big each adjustment step is. In the end, the output is a trained network that can handle complicated, non-linear relationships in the data things that straight lines or simple trees can't capture.
KNeighborsRegressor()	This method predicts the value of a new point by looking at the values of the nearest data points. The input is the feature data and target values. The argument n_neighbors sets how many neighbors to look at. The argument weights decides if all neighbors are treated equally or if closer neighbors should have more influence. The metric argument controls how distances are measured, for example using Euclidean or Manhattan distance. The output is the average of the neighbors' values, giving the prediction for the new point.
root_mean_squared_error()	This is an evaluation metric that measures how far the predictions are from the actual values. It takes the true values and the predicted values as input. It calculates the squared error for each prediction, averages them, and then takes the square root. The argument multioutput decides how the error is calculated if there are multiple outputs, either as a single average or separately for each output. The result is a single number showing the average prediction error in the same units as the target variable.

4. Task 4:

Section a, b and c:



For the degree=1 case,

Metrics - Train Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.833674	1.408923	0.059761	51.466506	6.670987e-12
SVM Regression	0.545415	2.329245	0.021942	68.878362	1.104724e-15
RandomForest	0.999000	0.109241	2.931659	3.769479	1.518686e-01
XGBoost	0.997146	0.184549	2.372918	1.046859	5.924852e-01
knn	0.995929	0.220416	2.510498	0.705376	7.027965e-01
Neural Network-10	0.939374	0.850625	0.162938	22.918092	1.055357e-05

Metrics - Test Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.818068	1.501800	0.079118	12.500550	0.001930
SVM Regression	0.562991	2.327567	0.034813	23.409392	0.000008
RandomForest	0.991747	0.319861	1.834557	0.904898	0.636068
XGBoost	0.992842	0.297890	1.851431	1.003285	0.605535
knn	0.992994	0.294699	1.971390	0.972202	0.615020
Neural Network-10	0.932186	0.916890	0.203631	3.434655	0.179545

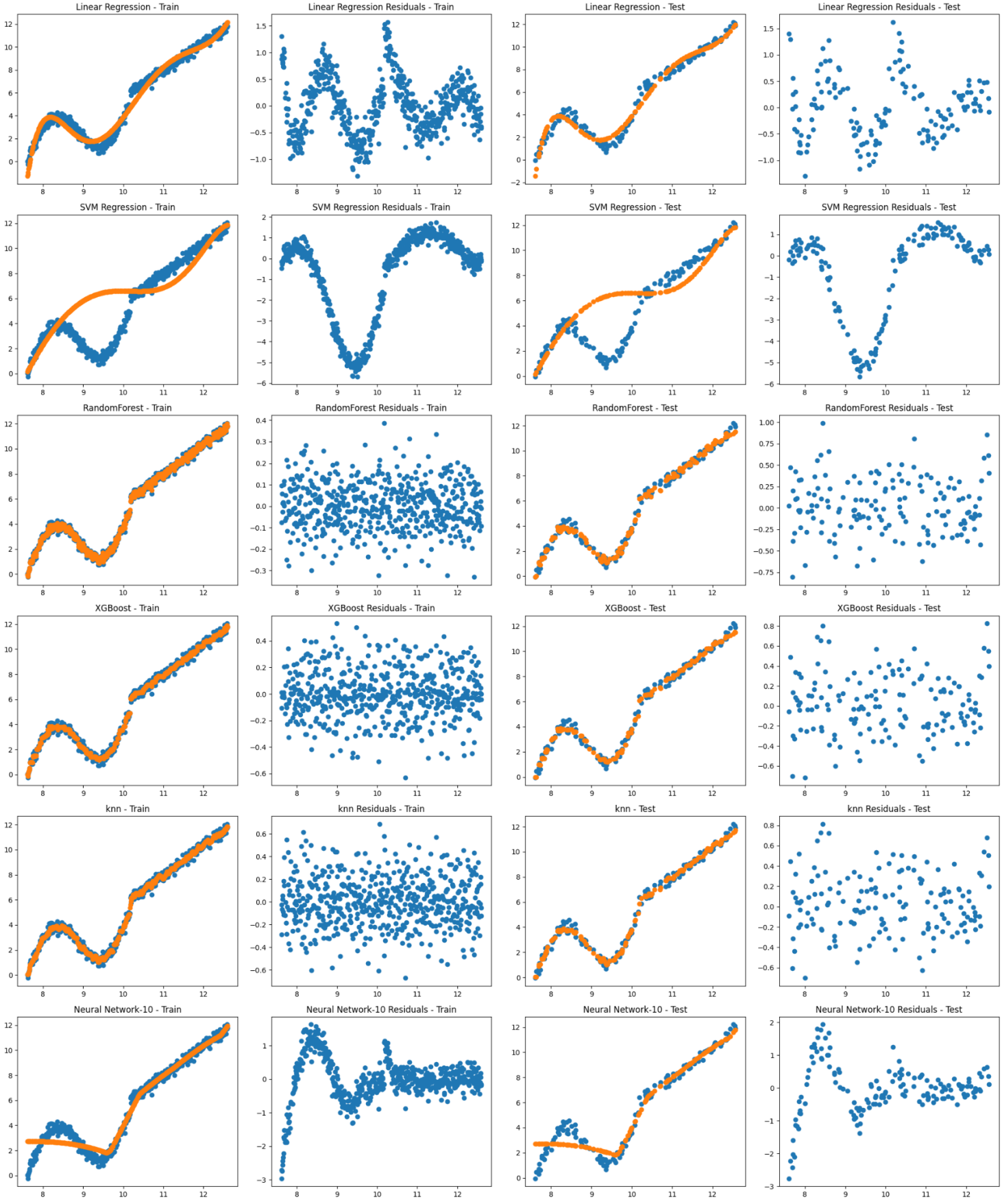
This is the original degree=1 metrics result. I'll use this as a comparison metric to show how much the degree=6 and 10 cases change the output.

I'll give my explanation on what is changing and if it for better after each curve and metric table.

I'll answer a), b) and c) together as they are very related.

I have already explained the results from degree=1 in the earlier question itself. Just as an extra point, SLR, SVM and Neural-Network-10 shows severe underfitting looking at the RMSE values whereas knn shows a good balance, and Random Forest and XGBoost look like some overfitting is there(due to higher RMSE of the test data) but it isn't too huge to give us many issues. Hence, its fine.

Below is the degree=6 data,


Linear Regression - Train
Linear Regression Residuals - Train
Linear Regression - Test
Linear Regression Residuals - Test
SVM Regression - Train
SVM Regression Residuals - Train
SVM Regression - Test
SVM Regression Residuals - Test
RandomForest - Train
RandomForest Residuals - Train
RandomForest - Test
RandomForest Residuals - Test
XGBoost - Train
XGBoost Residuals - Train
XGBoost - Test
XGBoost Residuals - Test
knn - Train
knn Residuals - Train
knn - Test
knn Residuals - Test
Neural Network-10 - Train
Neural Network-10 Residuals - Train
Neural Network-10 - Test
Neural Network-10 Residuals - Test

Metrics - Train Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.976261	0.532278	0.417086	11.050745	3.984384e-03
SVM Regression	0.566637	2.274224	0.023043	83.011284	9.426029e-19
RandomForest	0.999000	0.109255	2.927802	3.040236	2.186861e-01
XGBoost	0.997146	0.184549	2.372918	1.046859	5.924852e-01
knn	0.995895	0.221345	2.505231	0.733304	6.930509e-01
Neural Network-10	0.964262	0.653089	0.276050	198.763355	6.903761e-44

Metrics - Test Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.972925	0.579348	0.528190	3.916151	0.141130
SVM Regression	0.578190	2.286733	0.035979	27.177218	0.000001
RandomForest	0.991737	0.320048	1.832115	0.825637	0.661782
XGBoost	0.992842	0.297890	1.851431	1.003285	0.605535
knn	0.993051	0.293507	1.993466	1.266390	0.530893
Neural Network-10	0.951320	0.776844	0.279480	18.805295	0.000083

Now, looking at the above metrics, we can see how each model performs at degree=6 polynomial function argument.

Linear Regression reveals a small bit of underfitting at this point even though not that huge. This is because even though the RMSE values of train and test are close together, their values are quite large at around 0.5. Still its not too huge, hence a small underfit can be seen.

SV shows immense underfitting. The RMSE values are close to 2.3 for both train and test. This is too huge and it means the residuals still could have some pattern. This is proven from the DW factor, which is very close to 0, implying a strong correlation between residuals. The JB value is also very high implying poor normal distribution of the residuals.

Random Forest shows 0.1 RMSE for the train data and 0.3 RMSE for the test. This is a slight issue as the RMSE is almost thrice for the test data. This could be

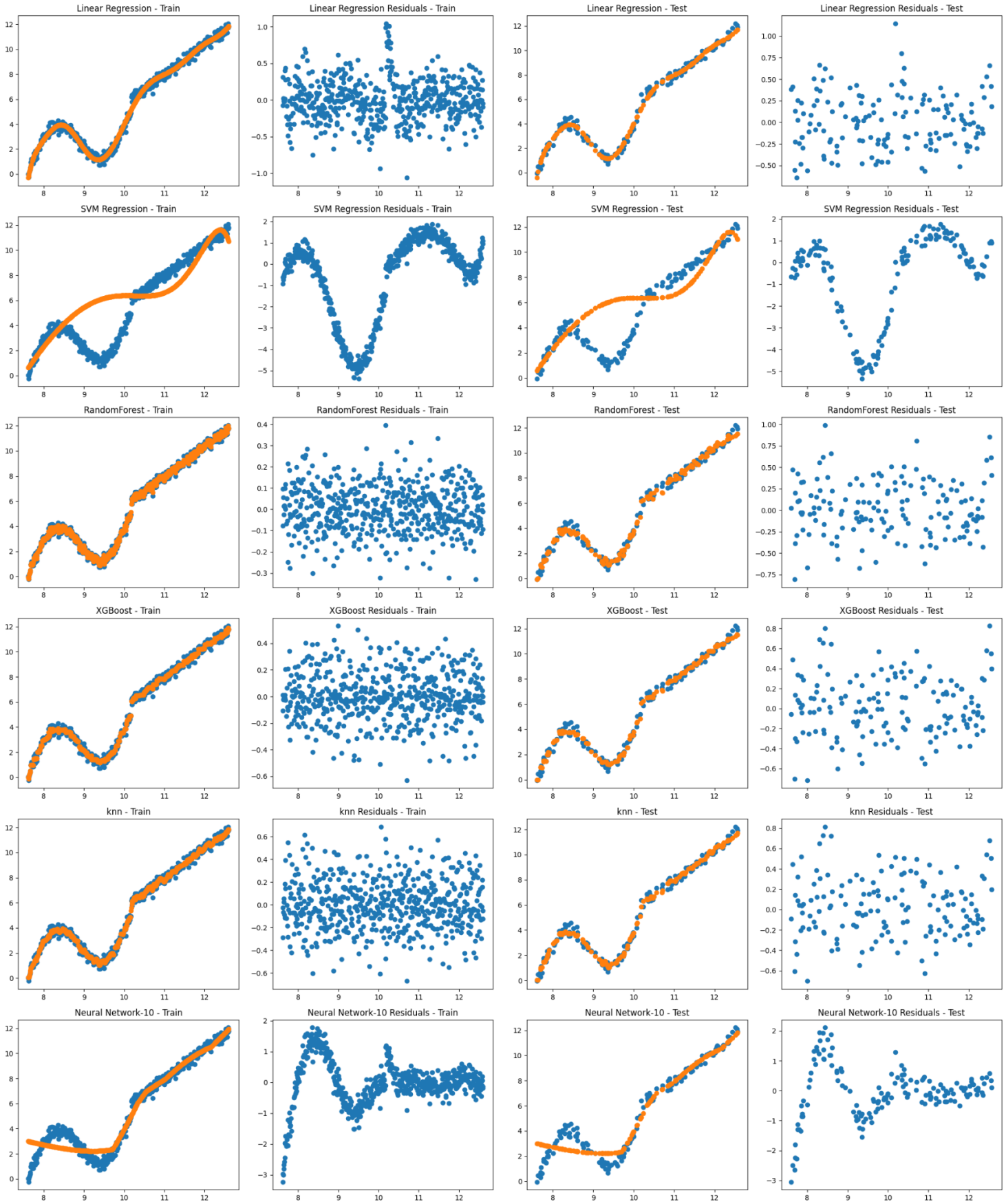
an issue with the choice of the test data, but if the data was chosen randomly, there is a good chance this is the start of overfitting. The issue is because the RMSE of the test is almost thrice the train.

XGBoost has a similar issue, but the RMSE differences are only close to 0.1, which is a much better result. Hence, it can be said that no overfit effect can be seen here, at least for a practical case.

The Neural-Network has a high RMSE, even though the difference is only 0.1, the values of the RMSE are greater than 0.6. Then means a good deal of underfitting is present in the graph. This is further concreted by the DW and JB metrics. An interesting fact to note here is that, it seems the Neural-Network went down in performance rather than an increase.

Now, depicting the plots for degree 10, we get,

(Neglect all this free space)


Linear Regression - Train
Linear Regression Residuals - Train
Linear Regression - Test
Linear Regression Residuals - Test
SVM Regression - Train
SVM Regression Residuals - Train
SVM Regression - Test
SVM Regression Residuals - Test
RandomForest - Train
RandomForest Residuals - Train
RandomForest - Test
RandomForest Residuals - Test
XGBoost - Train
XGBoost Residuals - Train
XGBoost - Test
XGBoost Residuals - Test
knn - Train
knn Residuals - Train
knn - Test
knn Residuals - Test
Neural Network-10 - Train
Neural Network-10 Residuals - Train
Neural Network-10 - Test
Neural Network-10 Residuals - Test

Metrics - Train Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.992987	0.289297	1.403683	28.852683	5.428996e-07
SVM Regression	0.617840	2.135651	0.026213	76.138305	2.929390e-17
RandomForest	0.998998	0.109336	2.926412	3.379817	1.845364e-01
XGBoost	0.997146	0.184549	2.372918	1.046859	5.924852e-01
knn	0.995895	0.221345	2.505231	0.733304	6.930509e-01
Neural Network-10	0.954087	0.740244	0.215261	146.923437	1.247329e-32

Metrics - Test Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.991865	0.317561	1.653610	3.724166	0.155349
SVM Regression	0.624045	2.158862	0.041005	23.876355	0.000007
RandomForest	0.991737	0.320059	1.830212	0.806068	0.668290
XGBoost	0.992842	0.297890	1.851431	1.003285	0.605535
knn	0.993051	0.293507	1.993466	1.266390	0.530893
Neural Network-10	0.939249	0.867829	0.223505	15.084513	0.000530

We see similar results as the table earlier. In these sense the overfitting ideas, underfit ideas, etc.

Linear Regression has improved a lot, reducing its RMSE, increasing its R^2. But, one major issue is that Linear Regression is not able to get to all the points properly. Some error spikes are there, because the JB value is quite high, which means the residuals do not form a normal curve. Overfit issues are not seen much. A bit of underfit might still be there because of the errors.

SVM Regression still is not able to model the data properly. It is still stuck at an RMSE > 2, which is terrible, and has strong residual correlation, large JB, etc. The model is doing terribly compared to the others.

Random Forst, XGBoost and knn are almost the same. They have little changes and the increase in RMSE of test and train and the ratio between them being 2 and 3 still raises alarms about overfitting. It means that these methods could be quite vulnerable to overfit. Knn seems the least affected here though and it seems quite stable too.

The Neural Network seems to be getting better after such a bad fall from degree 1->6. It still hasnt been able to get a good enough hold over the pattern of the data.

Section d:

Visually, its definitely the linear regression model that picks up pace very quickly and starts fitting better and better as the degree is increased. The Random Forest, XGBoost and knn already have fit the data properly and are just improving little by little later. The Neural Network and SVM models are quite slow changing and they still have a lot of error at the degree 10 situation too. The Linear Reg. model starts at high error and then keeps on reducing it. This is the inference from the visualization. Initially, the line was there are it didnt map all the points properly. As the degree increased to 6, the change was tremendous, and the line curved to include the sine curve like start and the end linearity. It had made a huge jump in progress which no other curve showed. There were better modelling curves but

Now, the metrics. It also reveals the same story, Linear regression changes from RMSE 1.4 -> 0.5-> 0.28 as degree changes from 1 -> 6 -> 10. This is immensely great. The model was way far back initially and then it almost caught up with the trees methods and non-parametric methods very quickly. This just proved that the visual inspection was correct and the math and metrics back that.

5. Task 5:

Section i:

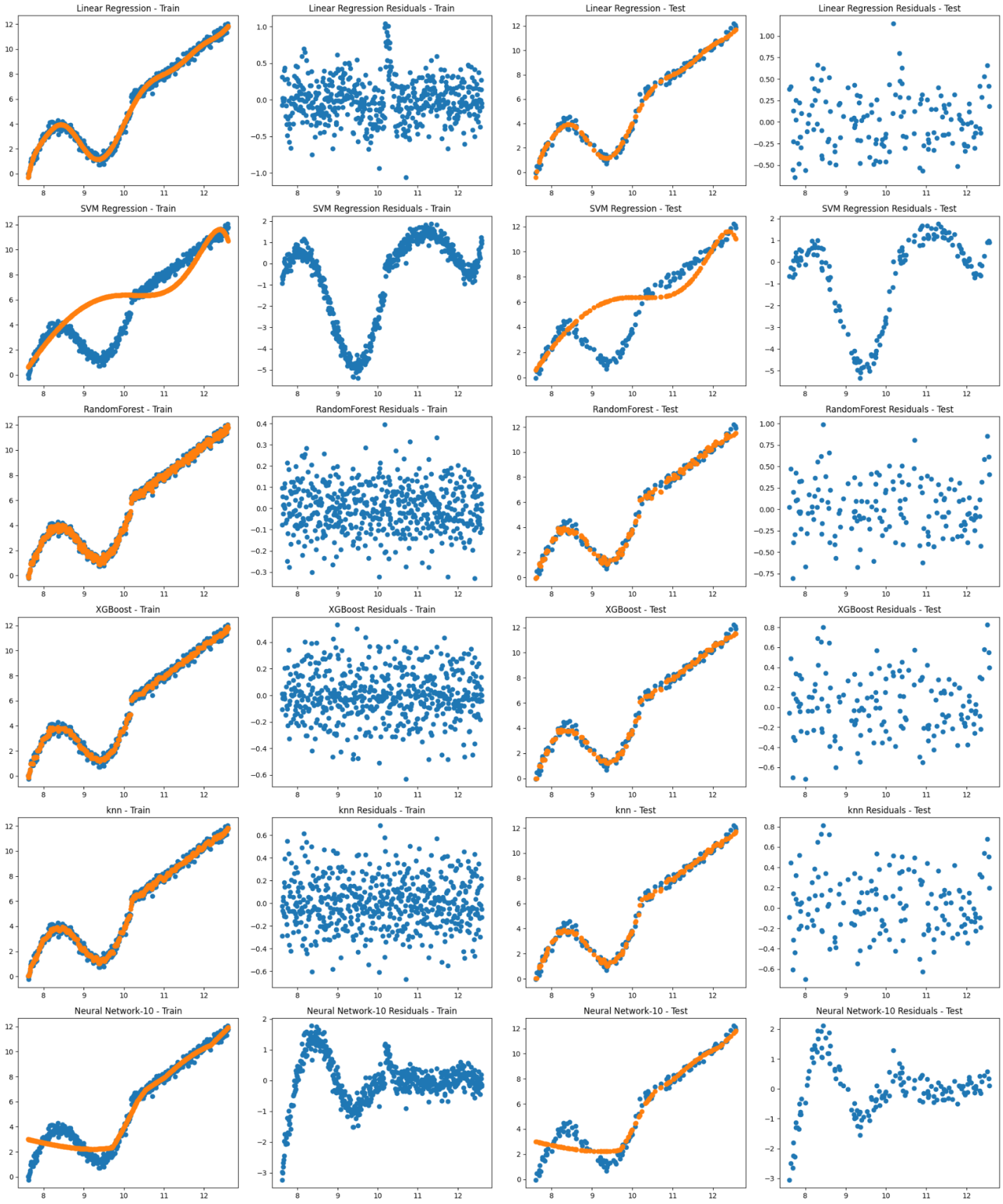
In this question we take set the degree of the polynomial function to 10. Then we check the results of the data with and without standardization.

Standardization is the function I had explained earlier. Its nothing but the change of the data values from their intrinsic mean and SD to mean=0 and SD=1. ie: the curve is brought to the middle and the SD is changed to 1. It gives a better metric to compare the models with.

But the standardization deeply depends on the values chosen, if they are very far apart, changing the SD=1 could deeply change how the data is interpreted. The standardization process is quite intricate and could be messed up easily.

Degree = 10

Standardized:



Metrics - Train Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.992987	0.289297	1.403683	28.852683	5.428996e-07
SVM Regression	0.617840	2.135651	0.026213	76.138305	2.929390e-17
RandomForest	0.998998	0.109336	2.926412	3.379817	1.845364e-01
XGBoost	0.997146	0.184549	2.372918	1.046859	5.924852e-01
knn	0.995895	0.221345	2.505231	0.733304	6.930509e-01
Neural Network-10	0.954087	0.740244	0.215261	146.923437	1.247329e-32

Metrics - Test Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.991865	0.317561	1.653610	3.724166	0.155349
SVM Regression	0.624045	2.158862	0.041005	23.876355	0.000007
RandomForest	0.991737	0.320059	1.830212	0.806068	0.668290
XGBoost	0.992842	0.297890	1.851431	1.003285	0.605535
knn	0.993051	0.293507	1.993466	1.266390	0.530893
Neural Network-10	0.939249	0.867829	0.223505	15.084513	0.000530

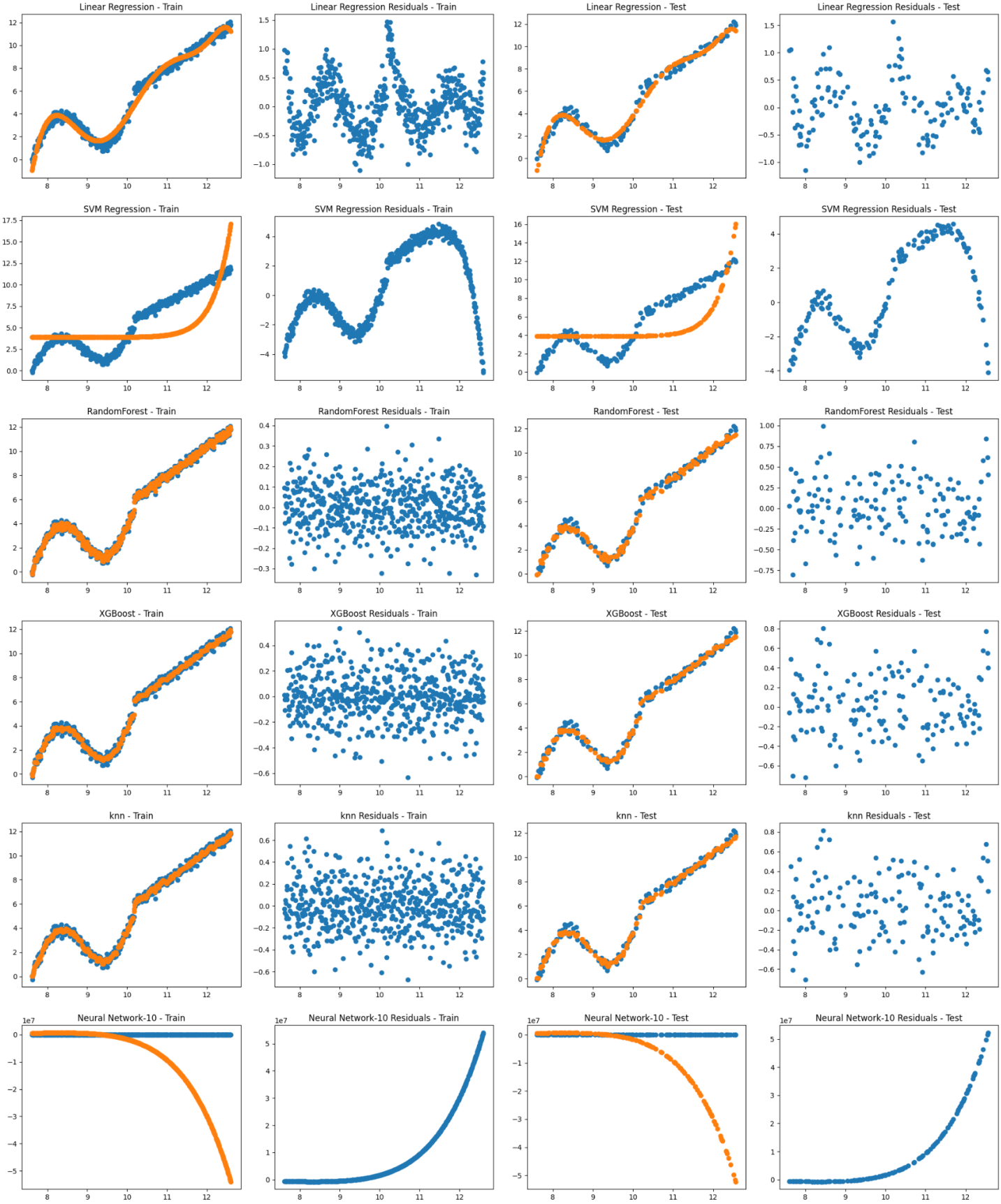
Non-Standardized:

Metrics - Train Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	9.827069e-01	4.543007e-01	0.572057	18.489218	9.663120e-05
SVM Regression	3.946295e-01	2.687931e+00	0.016682	43.824250	3.045685e-10
RandomForest	9.989994e-01	1.092800e-01	2.930430	3.250107	1.969011e-01
XGBoost	9.971463e-01	1.845492e-01	2.372918	1.046859	5.924852e-01
knn	9.958949e-01	2.213447e-01	2.505231	0.733304	6.930509e-01
Neural Network-10	-2.457190e+13	1.712485e+07	0.000101	245.456189	5.010323e-54

Metrics - Test Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	9.801501e-01	4.960630e-01	0.707796	4.669792	9.682056e-02
SVM Regression	4.252236e-01	2.669357e+00	0.030947	13.031702	1.479796e-03
RandomForest	9.917636e-01	3.195410e-01	1.835604	0.730177	6.941354e-01
XGBoost	9.928767e-01	2.971658e-01	1.864063	0.955220	6.202641e-01
knn	9.930510e-01	2.935072e-01	1.993466	1.266390	5.308928e-01
Neural Network-10	-2.187842e+13	1.646892e+07	0.001808	63.039205	2.047437e-14


Linear Regression - Train
Linear Regression Residuals - Train
Linear Regression - Test
Linear Regression Residuals - Test
SVM Regression - Train
SVM Regression Residuals - Train
SVM Regression - Test
SVM Regression Residuals - Test
RandomForest - Train
RandomForest Residuals - Train
RandomForest - Test
RandomForest Residuals - Test
XGBoost - Train
XGBoost Residuals - Train
XGBoost - Test
XGBoost Residuals - Test
knn - Train
knn Residuals - Train
knn - Test
knn Residuals - Test
Neural Network-10 - Train
Neural Network-10 Residuals - Train
Neural Network-10 - Test
Neural Network-10 Residuals - Test
1e7

From the above plots and metrics, its safe to say that the Random Forest, XGBoost and knn are the models which do not change much at all. Some do have changes but both increases and decreases are found, but the the difference is of the order 0.001, hence it can be neglected in some way. So it can be safely said they do no change.

Linear Regression has changed a bit. The RMSE has almost doubled. It seems as though the stabilization somehow kept things in order. It contained the values in some manner, but now as things are spread out again, the error has increased. But the change is only 0.2->0.4 which is big but not as huge as the upcoming issues.

SVM has changed tremendously. The metrics dont reveal much on this though apart from R^2 where the value has dropped almost 0.2. This is a huge change. The plot reveals that the entire plot has in some way changed to an exponential curve. The explanation follows in the next part.

The Neural-Network got affected in the worst manner here. It has drastically changed. The entire shape of the plot has changed and the RMSE has increased to 10^7, which is highly absurd. The explanation why this must have happened follows in part ii).

Section ii:

As earlier discussed, some models didnt change whereas some dont even look alike even a bit compared to their standardized models.

Random Forest and XG Boost rely on making different classes for the data sets. Splitting data into different classes does not rely on the scale of the data points chosen. What i mean is that, even if the data points are separated, as long as they are separated with the same scale, the classes are made in the same way. Hence, we see very little difference anywhere. For the tree based models.

knn works by averaging values with the nearest neighbours. Here, again scale does play a small role but as the features are powers of the main feature, it looks as though the C in C(Ax + …. - Bx^10), which is the scale that is getting averaged properly across all points without any issue. Small changes are occurring because the default neighbour distance changes when we change the scale, but those changes are very small.

Linear Regression gets a slight better treatment when standardized as then the points get closer together when the SD=1. This means the variance gets reduced a lot which means the variance to be explained by model in the y_pred decreases a lot. Hence the ratio of residual errors to the variance in the model reduces, leading to better RMSE and R^2 too.

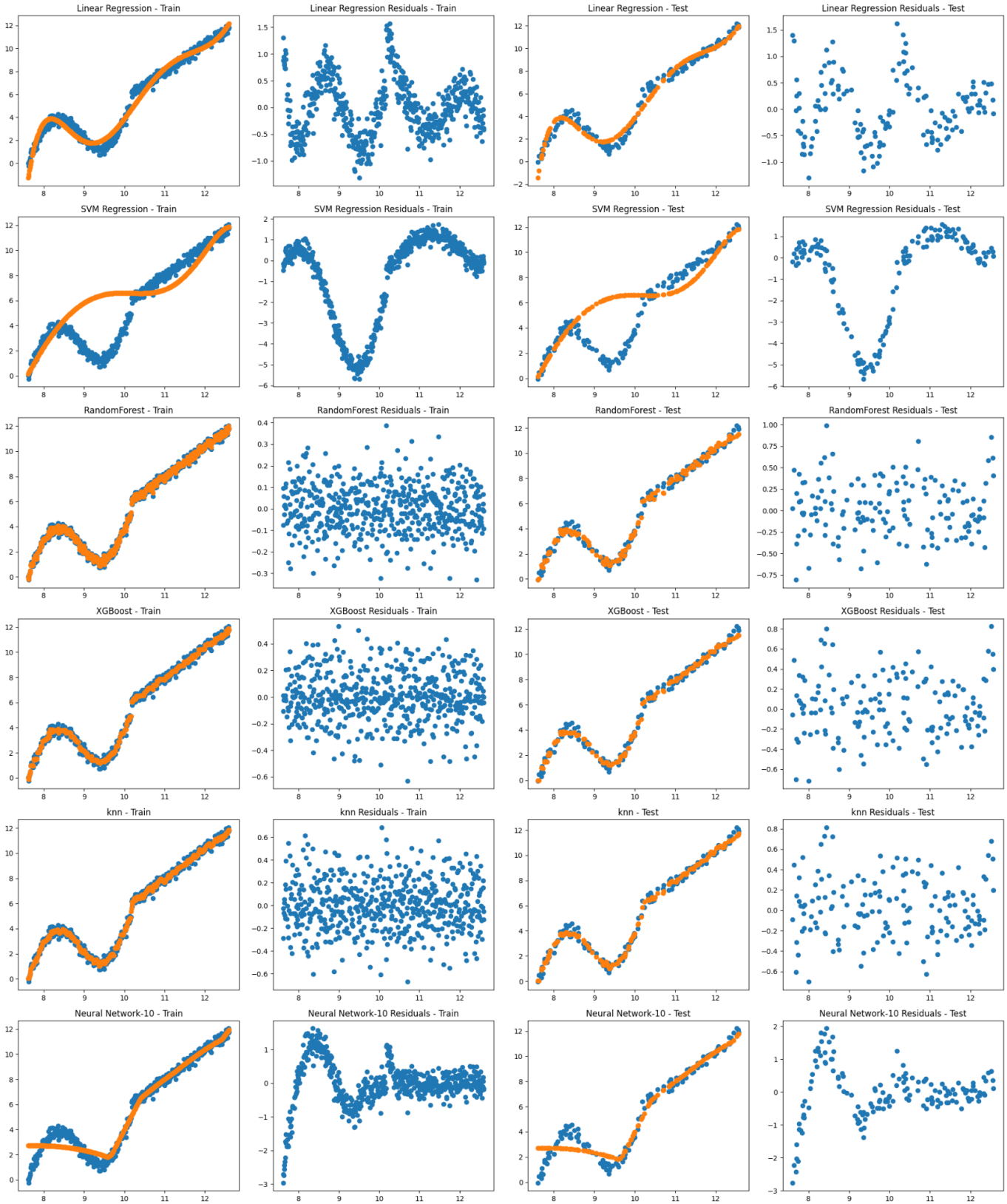
SVM gets treated very badly here when non-standardised. I believe this might be due to the intrinsic way of how SVM works. SVM works by putting forward ‘distances’ between hyperplanes and the points. When standardized, all the data points are small enough hence this distance is also fine. But when we increase, the values of the data points also increase. This is because the x^10 could explode very quickly depending on what x is. Hence, the distance could also explode quickly. This is supported by the plot too, where the divergence from the graph occurs at high x values only where the x^10 has the highest chance of becoming too large.

The Neural-Network is affected in the worst manner of all here. The RMSE shoots up to 10^7. This is very bad. It means the model was not able to even extract the pattern in any good way. This means when the degree=10 and the data was not standardized, the model just started misbehaving. This could be due to how the Neural Network works. It works by using the gradient descent method. When the values are standardised, their value is contained within a range and hence no issues occur and gradient descent occurs properly. But when the values are not standardized, the values can grow too much too fast. Finding the gradient can lead to many instabilities at large values. This means the gradient gets overshot very quickly and the pattern could just get more and more chaotic. It’s like the 0.0009 learning rate case, where the overshoot just kept on increasing again and again it reached 10^6 very quickly. Another issue could be in the way the ‘neurons’ work. They have weights associated with them. It seems as though the selection procedure selects the large x^10 values in a greater manner than the lower ones and gives them a greater weight or in another sense doesnt give them a small enough weight. This means those values explode out giving a very bad plot.

6. Task 6:

Section i:

For normal degree = 6,



Metrics - Train Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.976261	0.532278	0.417086	11.050745	3.984384e-03
SVM Regression	0.566637	2.274224	0.023043	83.011284	9.426029e-19
RandomForest	0.999000	0.109255	2.927802	3.040236	2.186861e-01
XGBoost	0.997146	0.184549	2.372918	1.046859	5.924852e-01
knn	0.995895	0.221345	2.505231	0.733304	6.930509e-01
Neural Network-10	0.964262	0.653089	0.276050	198.763355	6.903761e-44

Metrics - Test Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.972925	0.579348	0.528190	3.916151	0.141130
SVM Regression	0.578190	2.286733	0.035979	27.177218	0.000001
RandomForest	0.991737	0.320048	1.832115	0.825637	0.661782
XGBoost	0.992842	0.297890	1.851431	1.003285	0.605535
knn	0.993051	0.293507	1.993466	1.266390	0.530893
Neural Network-10	0.951320	0.776844	0.279480	18.805295	0.000083

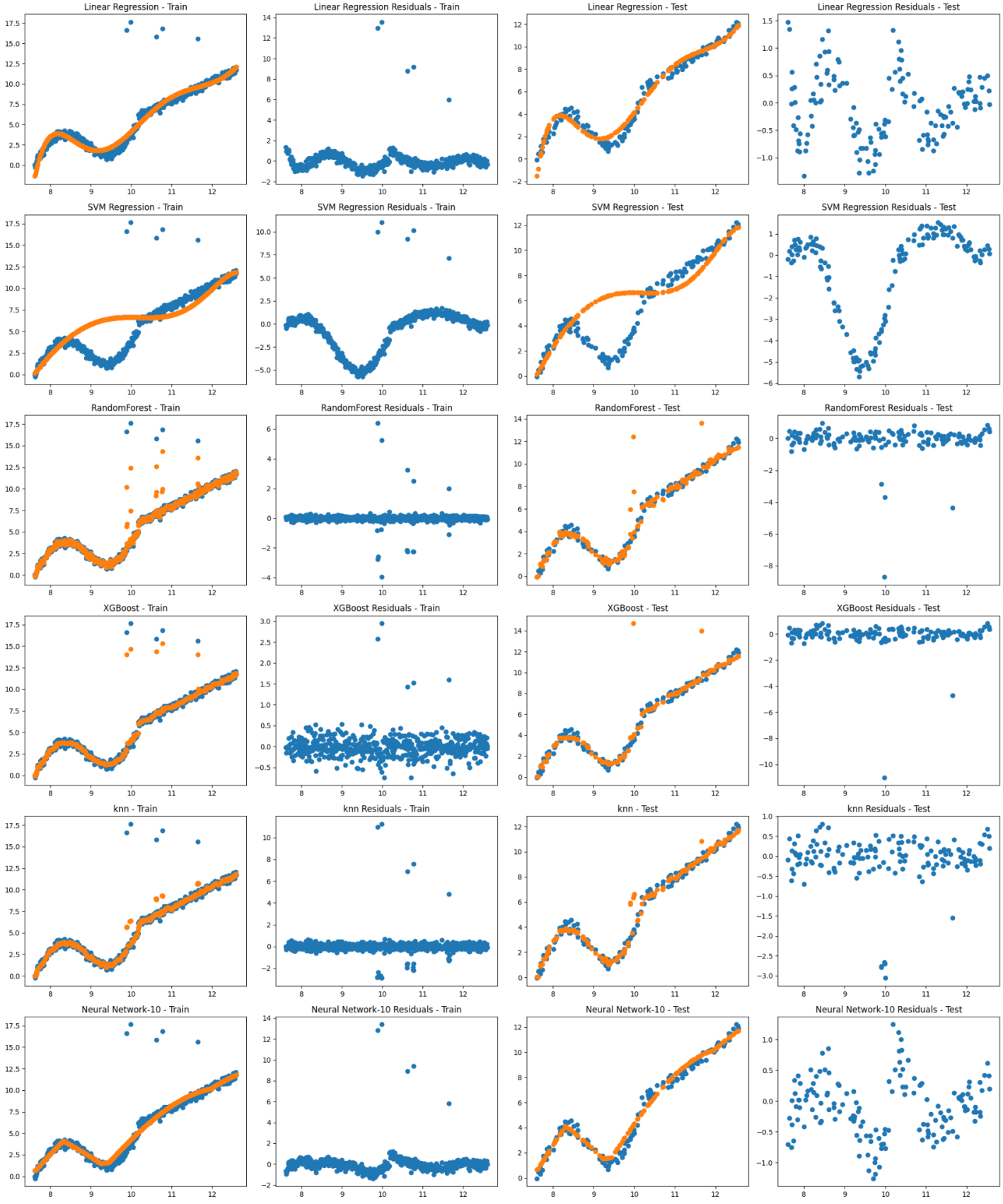
For the outlier added data,

Metrics - Train Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.899544	1.137503	0.347445	151552.426493	0.000000e+00
SVM Regression	0.533769	2.450564	0.051834	151.685745	1.153079e-33
RandomForest	0.979081	0.519082	0.774452	125542.541464	0.000000e+00
XGBoost	0.993643	0.286150	1.246634	29119.903663	0.000000e+00
knn	0.932941	0.929382	0.495940	175594.587772	0.000000e+00
Neural Network-10	0.908161	1.087626	0.369110	210810.169918	0.000000e+00

Metrics - Test Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.970786	0.601799	0.491065	2.523623	2.831407e-01
SVM Regression	0.574131	2.297710	0.035652	27.322876	1.166576e-06
RandomForest	0.929675	0.933707	1.474115	17518.681464	0.000000e+00
XGBoost	0.915877	1.021207	1.889649	50480.727064	0.000000e+00
knn	0.971129	0.598262	1.081701	1197.012762	1.180270e-260
Neural Network-10	0.981604	0.477552	0.722470	0.684184	7.102829e-01


Linear Regression - Train
Linear Regression Residuals - Train
Linear Regression - Test
Linear Regression Residuals - Test
SVM Regression - Train
SVM Regression Residuals - Train
SVM Regression - Test
SVM Regression Residuals - Test
RandomForest - Train
RandomForest Residuals - Train
RandomForest - Test
RandomForest Residuals - Test
XGBoost - Train
XGBoost Residuals - Train
XGBoost - Test
XGBoost Residuals - Test
knn - Train
knn Residuals - Train
knn - Test
knn Residuals - Test
Neural Network-10 - Train
Neural Network-10 Residuals - Train
Neural Network-10 - Test
Neural Network-10 Residuals - Test

Its quite difficult to exactly pick which models were the most resilient to outside noise, ie: outliers.

But, in my view the only model which truly stayed silent in the presence of outliers was the SVM model. It's R^2 value stayed approximately same meaning that the model was pretty unchanged. It decreased a bit and it is ofcourse going to as some error is seen but as the data size is large and the difference being small means the model withstood change. It makes sense because it puts weight for points. And if the distance is too large the loss function won't consider that point. This means lower effect of the outliers.

Almost all the others got changed due to the outliers.

The trees and the non-parametric models got a decreased R^2 value, most probably because the trees created extra classes for including those points and tried to overfit bringing the noise in the equation. This is clear with the test metrics where they differ a lot with respect to the train data, which signifies some sort of overfitting. The non-parametric model, ie: knn took the average with respect to the outlier too, decreasing its correctness.

Linear regression got affected the worst where is suffered a huge decrease in R^2 and increase in RMSE. This is because Linear Reg. is a basic model which simply minimizes the error in between the points and it doesnt care about outliers. Therefore it gives the worst results.

The only real outlier in the set of models is the Neural Network which, contrary to how all the others worked, actually fit the original data better when the outliers were added. It led to decreased RMSE actually even after adding the outliers. This seemed very weird. But it must be due to how it measures distances. The outliers must have forced the model to go past its tipping point in some way.

Section b:

SVM seems like a very good choice here but the only issue is that that even though it keeps the same R^2 value, not getting affected by the outliers, still its R^2 value is terrible. Meaning, even though the noisy data did not affect it, the model as a whole doesnt perform well compared to those which did get affected.

Linear Regression would be a very bad choice in the case of outliers on one side of the regression line, which would pull the line towards the outliers and cause

unnecessary increase in errors and mispredictions. So, Linear Regression also seems like a bad choice.

The Neural Network was the real shocker here. But the issue I face with it is that, even though it performed well totally after the outliers came, it did not before. There is no proof to whether it could be a coincidence or not, hence it could be a choice the user has to take. If the number of hidden layers was increased, it could be a good contender.

The overall best choices seem to be the tree and non-parametric models. This is because even though the model did get misled by the entries, they still are able to consistently plot the predictions properly, given the outlier numbers are small.

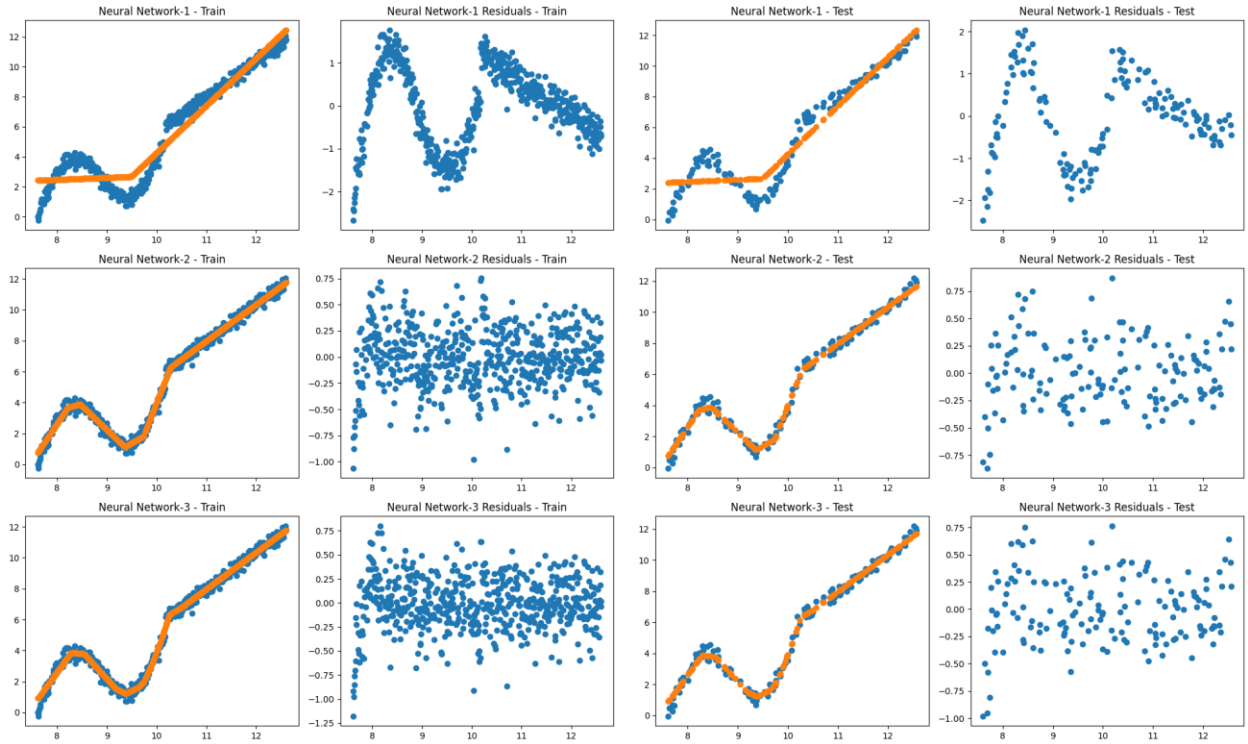
If the outliers were large in any way SVM could be the best choice as it would stay the most resilient to the outliers.

7. Task 7:

Section i:

For this particular task, I have to make 3 new Neural Networks. I have to make the Neural Networks with 1, 2 and 3 hidden layers with 10 neurons in each. I have changed the code for it as below and the data received will follow.

```
# New algorithms i added for the Q.7 for the neural network case:
algorithms = {
    'Neural Network-1': MLPRegressor(hidden_layer_sizes=[10],
max_iter=20000),
    'Neural Network-2': MLPRegressor(hidden_layer_sizes=[10, 10],
max_iter=20000),
    'Neural Network-3': MLPRegressor(hidden_layer_sizes=[10, 10,
10], max_iter=20000),
}
```



Metrics - Train Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Neural Network-1	0.932303	0.898863	0.146157	17.217504	1.825015e-04
Neural Network-2	0.993462	0.279339	1.499435	12.673009	1.770480e-03
Neural Network-3	0.993509	0.278342	1.512692	34.844785	2.713633e-08

Metrics - Test Data:

	R-squared	RMSE	Durbin-Watson	Jarque-Bera	JB P-value
Neural Network-1	0.921467	0.986697	0.177973	2.748799	0.252992
Neural Network-2	0.992171	0.311533	1.678978	0.634342	0.728206
Neural Network-3	0.992059	0.313753	1.698892	0.755333	0.685459

Neural Network-i -> i is the number of hidden layers.

Consider an input and output layer which represents the input features and the target value respectively. This is how a neural network looks. The hidden layers are the layers in between these which do all the actual processing and classification or regression. There are activation functions, etc.

So, as per logic, more the layers, more the pattern extraction.

The first Neural Network tries to extract the pattern but it cannot get through to all the points in a proper manner. It misses out on the oscillatory-like first phase and then to minimize error goes at the linear part too in a wrong way. This is quite obvious from the high value of RMSE=0.9, low value of DW=0.14 and high JB value.

The second Neural Network performs much better than the first one. It extracts the pattern much better than the first one. It fits the oscillatory first part and the linear second part in a very neat way. This is quite obvious from the metrics too. Here the R^2 value is 0.99 and the RMSE had reduced a lot and now is 0.27. The DW value is also much better now showing lesser correlation between the errors. The second Neural Network has improved a lot compared to the first.

The third Neural Network possesses values that are very close to the second Neural Network, except that the JB value is much higher than the second Neural Network. This could be due to some intrinsic property of the data itself which has some non-normal residues in some way. It could be such a weird property of the data. Overfitting is also a possibility but not a probable one. This is because the test data shows optimal level of RMSE and R^2 comparable to the train data. If overfitting were there this would not be the case at all even for interpolation.

So, one thing is clear is that more the hidden layers, the better the plot, until a limit. Then more complexity is not that useful, like the 2->3 case where very little has changed. Adding even more complexity could even bring upon learning of the noise patterns and therefore overfitting.

8. Task 8:

Non-parametric models, here the knn model are very easy to use first of all. They fit onto data with great metrics like R^2 and RMSE. The values of the metrics are comparable to the tree models too. This is great for quick regression models.

It was also observed that the knn model was not affected by the outliers that much too. It has the highest R^2 value when compared to the tree models which is pretty advantageous. So it also handles noisy data with outliers pretty well.

But there are major issues in the knn model. It doesnt have an exact training phase. It just stores the data and when needed, for eg: when evaluating test data, the points are taken one by one and compared to find the output. This translates to a much larger issue. As the data size was small here, we didnt face much of an issue. But in the case of larger data sets, the time taken could reach a large value. This is a major shortcoming.

Another major issue lies in how knn works. knn intrinsically calculates a distance factor between the points and the hyperplanes between them. For normal degree 1 polynomials this isnt an issue. But as soon as we include large degrees in the feature space, the distance parameter could be very large. This can lead to errors and if not standardized properly.

All others left aside, when working with small data sizes, the knn model is a very good and versatile model.

9. Task 9:

The use of Linear Regression or knn/Random Forest comes down to how the data looks like.

If the data is say linear, using Random Forest or knn is just adding more unnecessary complications. If the data is linear, the best solution would be to use SLR.

It is best to use Random Forest or knn when the given data set is quite complex and the relationship seems non linear. Also another major thing that can be stated here, even though it doesnt make a huge difference is standardization. If the data is pre-processed and standardized and if it is non-linear, using a general model like Random Forest or knn is the best choice. This gives us a very good approximation of how the regression curve or classification would look like.

Another good time to leave SLR and use Random Forest/knn is when the data has a lot of noise, ie: outliers. Random Forest/knn is quite good at staying the same and not focusing on the outlier data, whereas linear regression, as it has no weights or distance function, it just simply wants to reduce the SSE. This means to reduce the error from the outliers the plot deviates from the actual line. This is an issue Linear regression faces.

So, in short, just because Random Forest or knn can do what Linear regression does, in very simple cases where the data points are not noisy, use of Linear Regression gets rid of a lot of unnecessary complexity. But for general, non-linear cases, use of Random Forest or knn is preferred. So, not using Linear Regression is not an option, its just that depending on the data, one might be better than the other taking into consideration time, etc.

10. Task 10:

I have learnt a lot from this assignment.

Firstly, I have learnt how to properly use the Gradient Descent method and in class even though we did learn about how learning rates matter, doing it hands on has really taught me to appreciate the method. I saw how the overshoot occurs and also how small differences in the learning rate can trigger huge changes.

Then next, I have learnt the implementation of some of the most important example models of parametric, non-parametric and trees. I analysed their behaviour to see how well they model data.

Initially when I ran it, I got the standard models. They said that some were better than others at modelling the data.

Then as per the tasks, I changed some code bits. First I checked what happens during feature generation. This was done with the PolynomialFeatures function, which created new features which were powers of the original features and cross terms with the same total power. I analysed the data and found that models like the Linear regression became better and better, when more features were added.

When noise was added to the system in the form of outliers, some models stayed resilient. I researched into why such stabilities are there and why some models cant handle outliers at all and are misled easily.

Standardization was one of the major things I learnt from this exercise. Using data simply means that, if we use large Polynomial degree feature additions, the model cannot accurately predict the results. This was seen in SVM and the Neural Network. I learnt that it is one of the most important pre-processing steps.

I went deeper into how a Neural Network works. This, the knn and the SVM were the only blanks I found that i had to fill. I went deeper and I learnt how they basically work even though i didnt go deep into the math.

I learnt how to use the metrics properly to predict which model is better. I learnt how to understand the shortcomings in a plot from just the metrics too.

I also learnt the Functions used in python for such models and analysis and also learnt their arguments and how they can be modulated in ways according to our preference.