Machine, Data and Learning

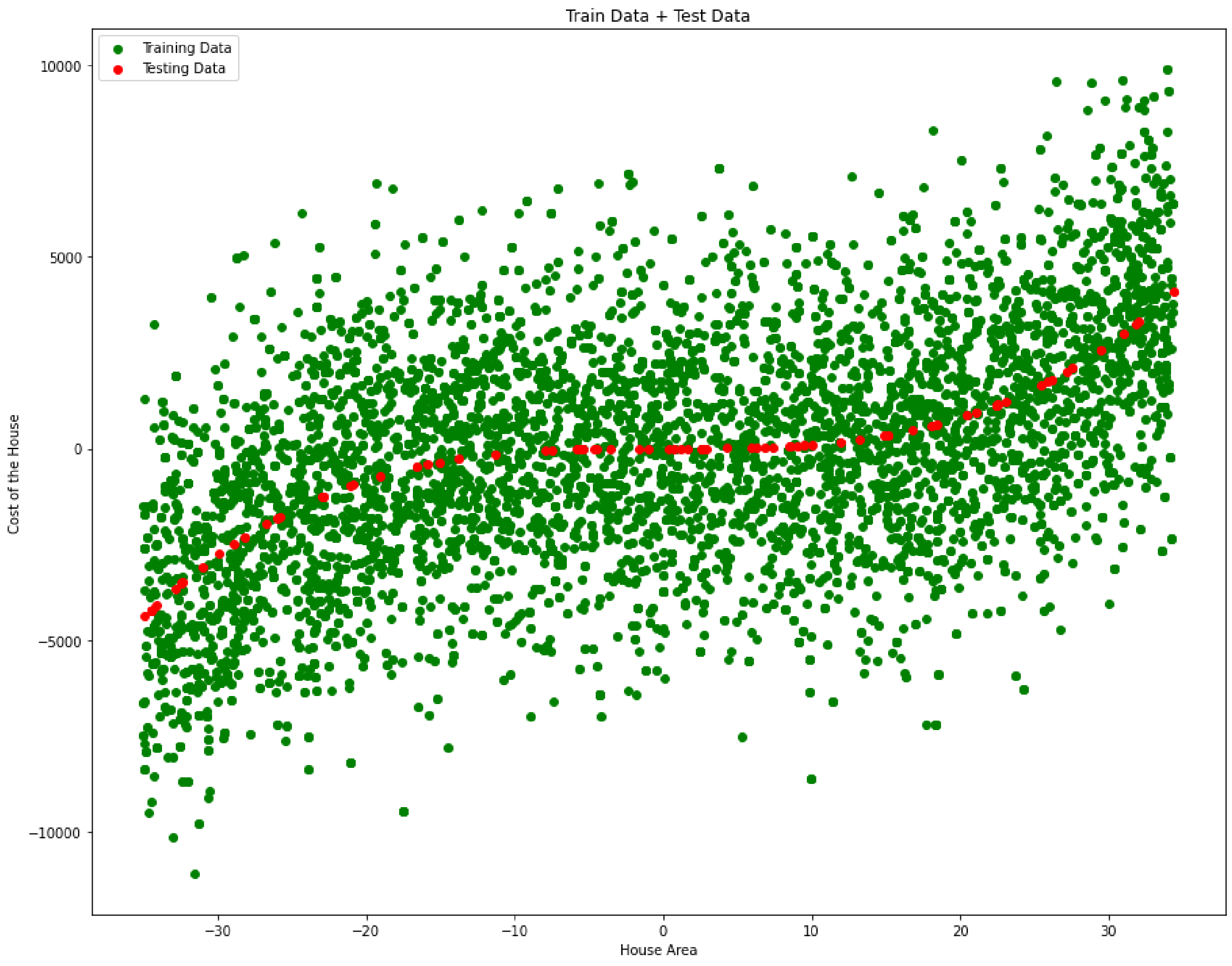
Assignment 1

> **Team number: 102**
>
> **Team Name: Underfit**
>
> **Members:**
>
> 1. Harshit Gupta - 2020114017
> 2. Vamshi Krishna Bonagiri - 2020114011

The Problem

Radwait wants to buy the best possible house at the lowest possible cost. In order to do this, he looks at the data of houses which shows how the house area is related to the cost of the house. In order to do this, he needs a model which is trained on the given data, as it can show the relation between the house area and cost.

The data given is shown in the scatterplot below:



Let's start by assuming a relation between house area(x) and house cost(y), let's assume the function $f(x)$ along with some noise(ϵ) in the data account to this relation. We denote it as

$$y = f(x) + \epsilon$$

Our task is to predict the function $f(x)$, let's say we have a model which predicted the function $\hat{f(x)}$. For this, the `Mean Squared Error(MSE)` is the average squared difference of a prediction $\hat{f(x)}$ from its true value y. It basically denotes how far the predicted model is, with respect to original relation. It is given as:

$$MSE = E[(y - \hat{f(x)})^2]$$

Our task is to minimize this error, as it would mean to have the best possible model.

Task 1: Linear Regression

For this task, we will be using a **Linear Regression** model which would allow Radwait to select the best possible house. Linear Regression is the process of generating a straight or a curved line that best fits a set of scattered data points. This line can then be used by projecting it to find new data points within a continuous range.

The `LinearRegression()` class comes from the `scikit-learn` package to aid us with the above.
We use `LinearRegression().fit()` to create a model and train our datasets on it.

```
model = LinearRegression()
```

This statement creates the variable model as the instance of `LinearRegression()`.

In order to understand what sort of model it creates, let us take the equation:

$y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_n x_n + b$

The equation above is the mathematical representation of multiple linear regression.

Where,

1. y - Dependent Variable (output)
2. $x_1, x_2, \ldots, x_n$ - Independent Variables (input)
3. w_0 - Intercept on the axis
4. $w_1, w_2, \ldots, w_n$ - Slopes
5. b - Bias

Based on all above conditions, we have the values for the Dependent and Independent Variables. The only unknown values are those of w_0, $w_1, w_2, \ldots, w_n$ and b.
The `.fit()` predicts these unknown values as optimally as possible to get the predicted values (later on) as close to the actual values.

In order to use the model, we now call `.fit()`.

```
model.fit(x, y)
```

In simple words, `.fit()` fits the model. Using the existing input(x) and output(y) as the arguments, it calculated the optimal weights of the model.

In our dataset, the arguments are only a 1D array. However, you can provide 2D arrays as arguments as well.

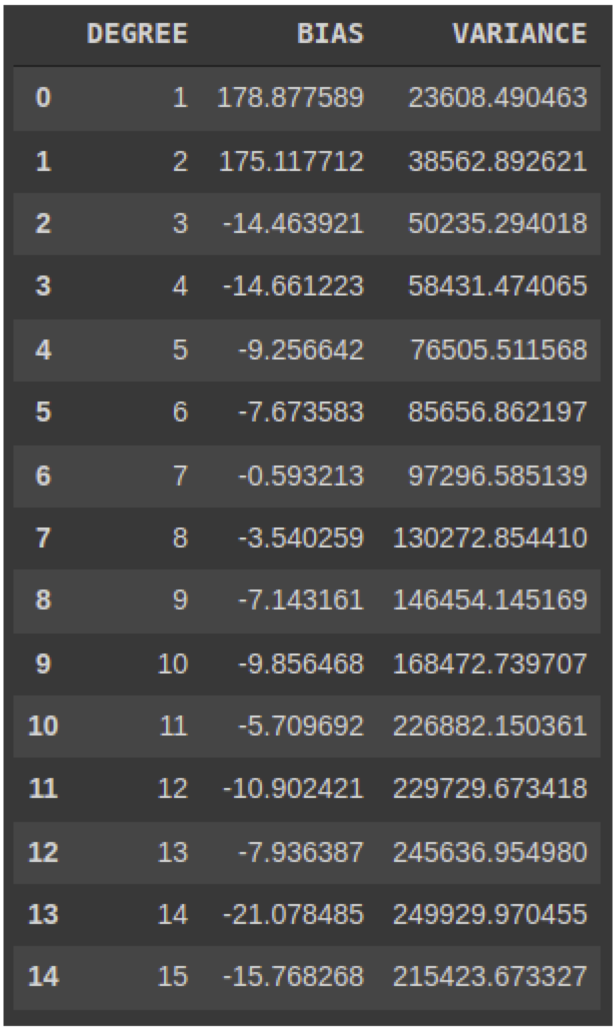
To see what parameters `.fit()` generates, you can run:

```
model.intercept_ #1D array with single element.
model.coef_ #2D array with multiple elements.
```

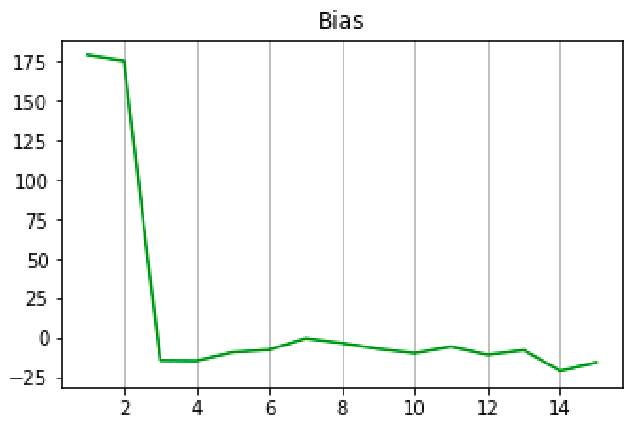
These contain the values for the slope and coefficients for the above equation.

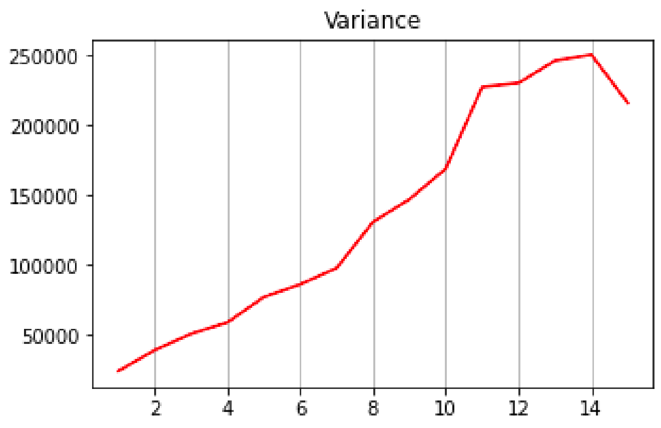
Task 2: Calculating Bias and Variance

Given are the values of bias and variance from each of the function classes(each corresponding to a polynomial degree model).



	DEGREE	BIAS	VARIANCE
0	1	178.877589	23608.490463
1	2	175.117712	38562.892621
2	3	-14.463921	50235.294018
3	4	-14.661223	58431.474065
4	5	-9.256642	76505.511568
5	6	-7.673583	85656.862197
6	7	-0.593213	97296.585139
7	8	-3.540259	130272.854410
8	9	-7.143161	146454.145169
9	10	-9.856468	168472.739707
10	11	-5.709692	226882.150361
11	12	-10.902421	229729.673418
12	13	-7.936387	245636.954980
13	14	-21.078485	249929.970455
14	15	-15.768268	215423.673327



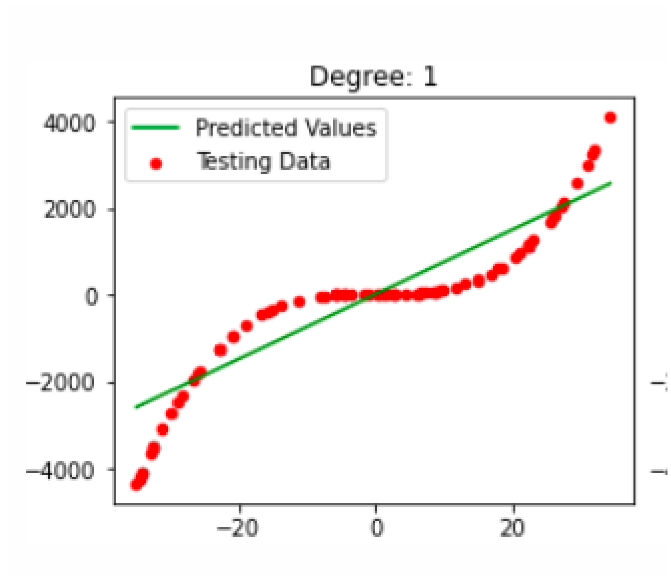


Bias

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. A model with high bias does not generalise the data well and oversimplifies the model. It always leads to a high error on training and test data.

$$Bias = E[f(\hat{x})] - f(x)$$

Clearly the simplest model in our function classes is the one with highest degree = 0 ($y = mx + c$) since it only depends on one feature/variable. We see that the bias of this model(178.9) is the highest among our class functions. This is how the linear model's(degree = 1) predictions with respect to the test dataset would look like:



Clearly the high bias is due to the model being too simple to be able to predict correct values. Similarly, the model with polynomial degree 2 also has a very high bias(175) compared to the other degrees, this is again because of the same reason of being too simple. It's noteworthy to see that the model is not simpler compared to the linear model, which makes its bias a bit lesser than the linear model.

Now, we see a sudden drop in bias from 175(degree = 2) to -14.5(degree = 3)(since bias is just a measure of difference, we dont care about its sign, this is why we usually square it before further analysis). This means that there's a huge variation in bias just by adding one feature, which implies that the new model fits the data far better than the previous one. After the sudden drop of the bias value, there are minor fluctuations of it, with degree 6 polynomial holding the least bias value of 0.6.

Therefore, the bias values of any model with degree >= 3 are very low compared to degree = 1 or 2 which would lead to high errors due to their simplicity. Another important observation to make from the plot above is that it has atleast two points of inflection, one around x = -15 and the other around x = 15, this means that the original function $f(x)$ could have been a polynomial of degree = 3 or more, this aligns with our bias observations. Another important factor to consider when trying to find the best model, is the Variance.

Variance

Variance is the variability of a model prediction for a given data point. Again, imagine you can repeat the entire model building process multiple times. The variance is how much the predictions for a given point vary between different realisations of the model.

$$E[(f(\hat{x}) - E[\hat{}f(x)])^2)]$$

Since we do not want the predictions to vary much between different realisations of the model, we try to minimize the variance. High variance is usually due to overcomplex assumptions which would lead to high error.

The variance values for our function classes are proportional to the degree of the polynomial, with the only exception being the 15th degree model. Among our function classes, we know that as the degree increases, the complexity of the model increases. This is because we increase the number of features with each degree, but as the number of features increase, the number of assumptions the model makes, also increases.

In our function classes, as the number of features increases, the model becomes more susceptible to changes in data, which would lead to higher variance. For example, assume there are two linear models $y1$ and $y2$ with weights $w11$ and $w12$, the difference between these model predictions at one point would be $y1 - y2 = (w_{11} - w_{12})x$. If we were to calculate a similar difference between models of degree two, we would get $y1 - y2 = (w_{21} - w_{22})x + (w_{23} - w_{24})x^2$, clearly the mean of the first value would be lesser than the second value, since it has lesser features. This is exactly why the variance is increasing with respect to the degree of the polynomial among our function classes.

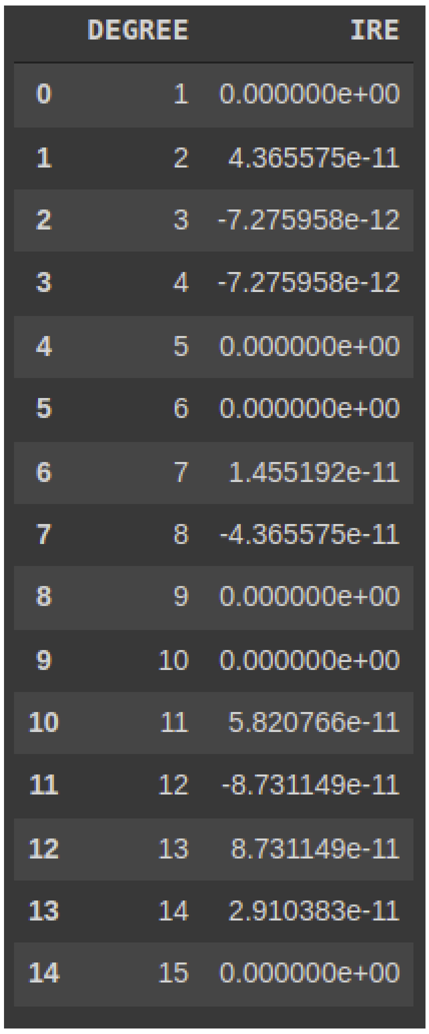
Task 3: Calculating Irreducible Error

Irreducible error is the error that cannot be reduced by creating good models. It is a measure of the amount of noise in the data. Here, it is important to understand that no matter how good we make our model, our data will have certain amount of noise or irreducible error that cannot be removed.

$$E[(f(x) - \hat{}\,f(x))2] = Bias^2 + \sigma^2 + Variance$$
$$\sigma^2 = E[(f(x) - \hat{}\,f(x))2] - (Bias^2 + Variance)$$

where f(x) represents the true value, ˆf(x) represents the predicted value, [E[(f(x) − ˆf(x))2] is the mean squared error and σ2 represents the irreducible error

Calculating Irreducible error using the above formula for each class gives us the following data:



	DEGREE	IRE
0	1	0.000000e+00
1	2	4.365575e-11
2	3	-7.275958e-12
3	4	-7.275958e-12
4	5	0.000000e+00
5	6	0.000000e+00
6	7	1.455192e-11
7	8	-4.365575e-11
8	9	0.000000e+00
9	10	0.000000e+00
10	11	5.820766e-11
11	12	-8.731149e-11
12	13	8.731149e-11
13	14	2.910383e-11
14	15	0.000000e+00

Irreducible error, as the name suggests, is irrelevant of the underlying model and has to do with the inherent noise in the problem. All of our function classes share the same dataset which also means that they share the same noise. The only thing changing across the function classes is the polynomial degree i.e, the model, since irreducible error is not supposed to change with respect to the model by definition, it is not supposed to vary across our function classes as well.

But we find different IRE values for each classes, however, the difference is very minute as its in the order of exp^{-11} while the order of dataset is in tens or thousands. We also see a lot of ire values being zero among the classes, this means that the ire value is indeed zero for all the function classes, the very minute fluctuations only occur due to the systems inefficiency to work with very small values and hence, it should be ignored.

Note that this irreducible error value need not be zero, it can be any constant value which does not change with respect to the model(in this case, function class).

Task 4: Plotting the $Bias^2 - Variance$ graph

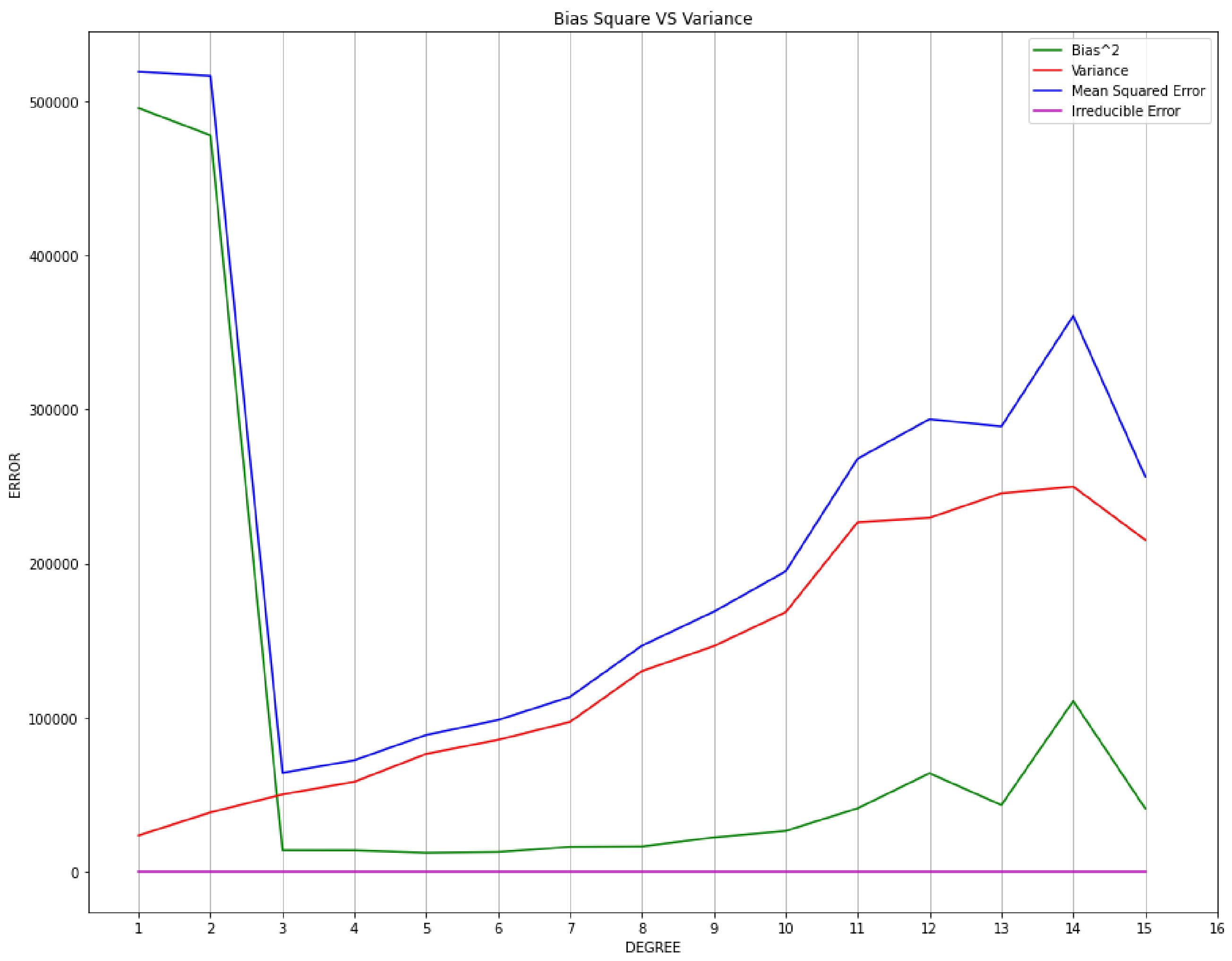
From the above analysis, we saw that a very simple model such as the linear model could not fit in the data, this occurs due to high bias values, this is called **underfitting**. Underfitting obviously leads to a lot of errors in the predictions. To get the best model, we need to avoid underfitting, we do this by making sure that the model is not to simple, which can be achieved by choosing models with low bias.

Similarly, we also need to make sure that the model's variance is low, since we need the model to predict similar values in different realisations(with different datasets). Essentially, we should make sure that a model does not depend on a wrong feature(like a man's BMI being dependent on his color). When the model starts making overcomplicated assumptions, it is called **overfitting**, such a model would perform well with the train dataset, but would lead to a lot of errors on general data. Hence, it is important to keep the model as simple as possible by keeping the variance low.

However, from the values of bias and variance collected and analyzed above, we see that the bias decreases with respect to the model complexity(or degree in this case), whereas the variance increases with respect to the model complexity. Since the bias and variance are inversely proportional, how do we make sure both of them are as low as possible?

If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand, if our model has a large number of parameters then it is going to have high variance and low bias. So we need to find the right (or good) balance without overfitting and underfitting the data. We do this by considering the intersection of $variance$ and $bias^2$, which would be the value where the error is minimum, we can confirm this hypothesis by checking the MSE at this point.

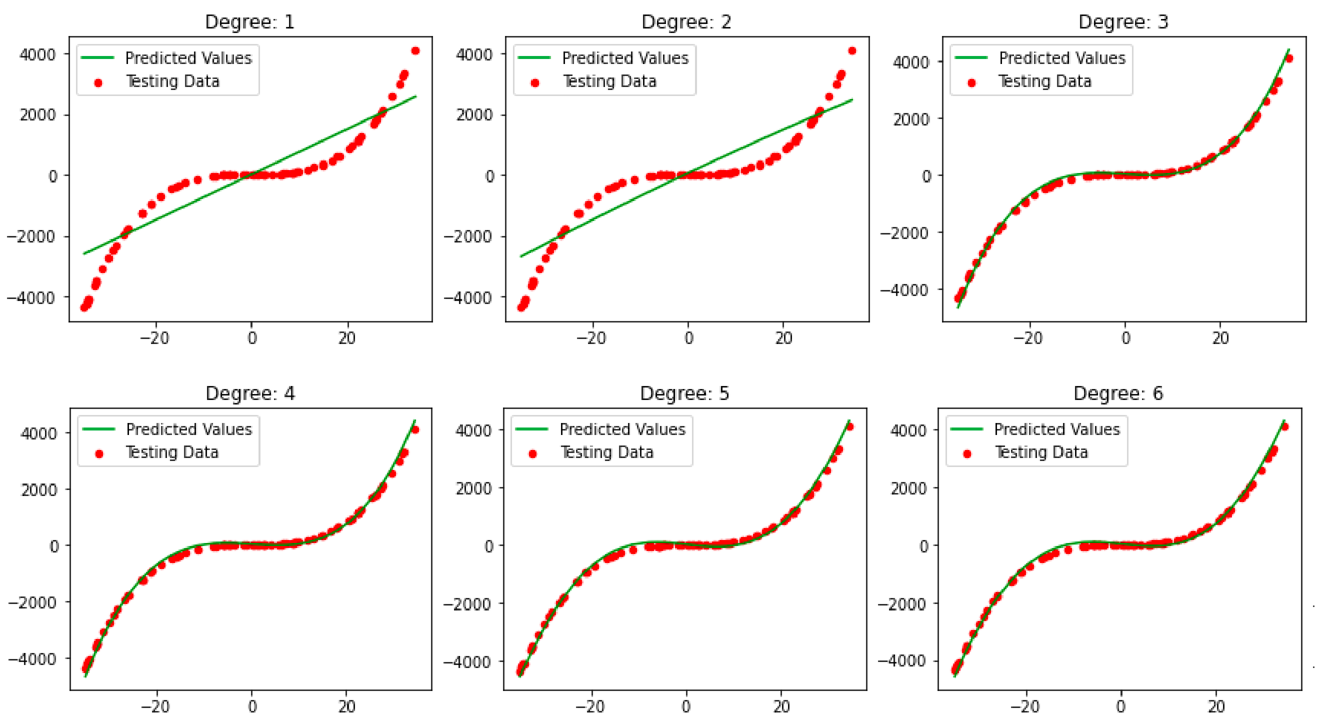
Plotting the data from above along with the MSE, we get:

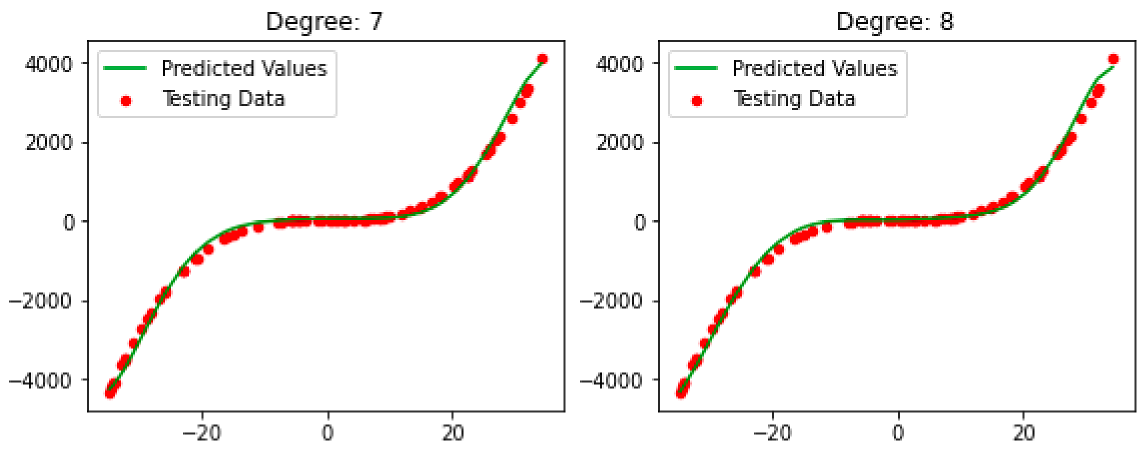


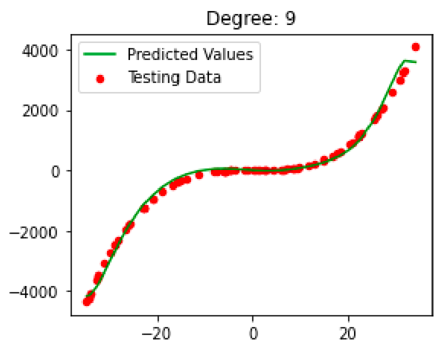
From the above graph, we see that the model behaves just as described above(more complexity leading to more bias and less variance) with an exception at the 15th degree polynomial. When the value of the degree is 1 or 2, the model is clearly **underfitting** as it has very high bias values compared to the other degree polynomials(We can visualize this in the plots below). At degree = 3, we see that there is an intersection of $bias^2$ and $variance$, this is also the place where the MSE attains its minimum value. after degree=3, as the degree increases, the variance keeps increasing along with the MSE, meaning the model is **overfitting** for values of degree > 3.

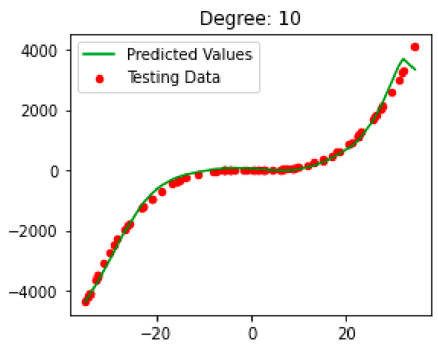
Since we attain the least error at the 3rd degree, our **best model** is the polynomial model with the **degree = 3**. This would mean that the data given is closest to how a 3 degree polynomial behaves, i.e, close to $y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3$. Meaning that it has 2 points of inflection, as observed before. The plots below give a clearer picture of the type of data

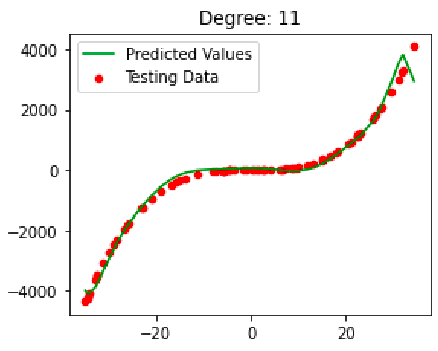
Plotting Actual VS Predicted Values Across The Degrees

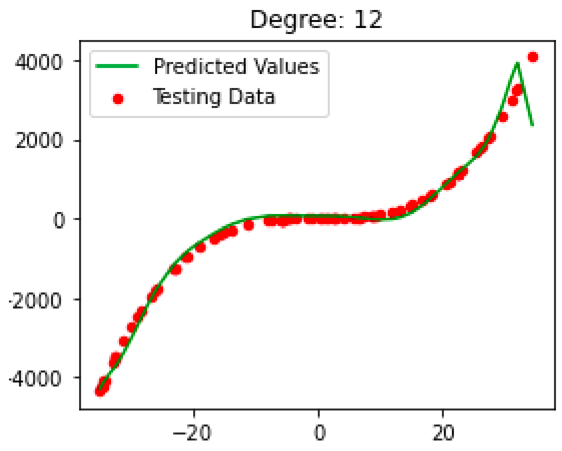


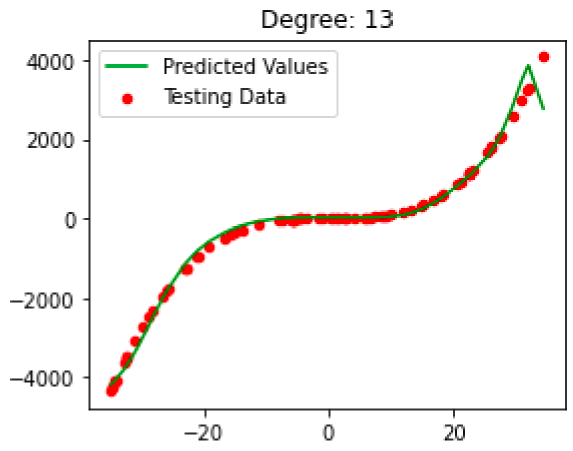


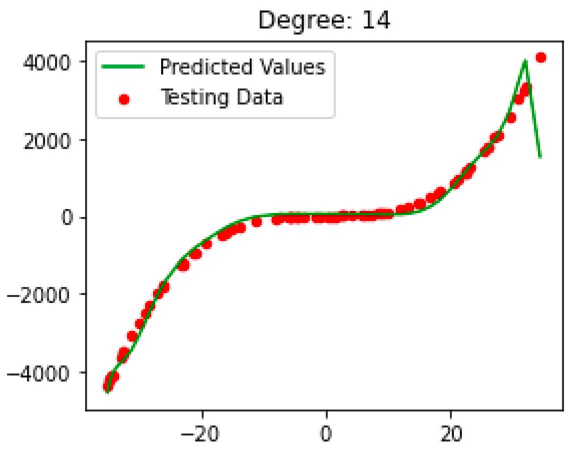


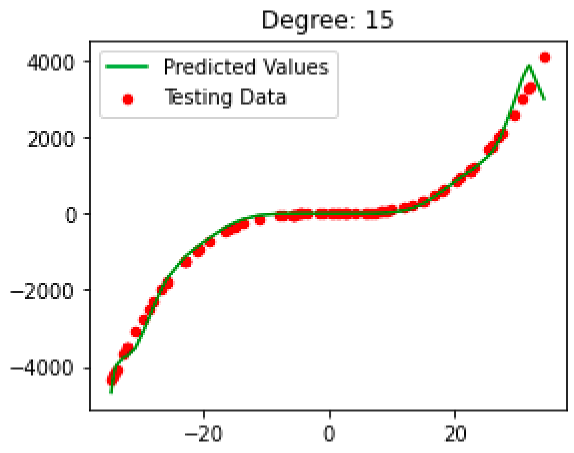












Based on the above figures, we clearly see that:

1. At degree = 1 or 2, the model underfits.
2. From degree=3 , the predicted values start to align themselves with the actual values.
3. After degree=3, the model starts to overfit.

THE END