CEC 427

FUNDAMENTAL OF DATA ANALYSIS

Dr. TCHAGNA Aurelle

Retymee
repeat and teach me

open technology

inch TECHS
SPIRIT OF INNOVATION

Objectives

- The purpose of this course is to enable the student to know the principle of data analysis, data cleaning, data normalization, standardization, data preparation and visualization, data analysis to estimate and predict the insights.

Course Description

- In this course, you'll be introduced to many of the primary types of data analytics, core concepts and type of data. You'll learn about the tools and skills required to conduct data analysis. We'll go through some of the foundational math and statistics used in data analysis and workflows for conducting efficient and effective data analytics. This course covers a wide variety of topics that are critical for working in data analytics and are designed to give to the students an introduction and overview to build relevant knowledge and skills. This course provides application to how a company can use data analytics to make better business decisions and help analyze customer trends and satisfaction, which can lead to new and better products and services. The course combines theoretical knowledge with hands-on training of the data analytics techniques.

Outcomes:

- By the end of the course, students should be able to:
  - Explain the primary types of data analysis
  - Define the phases of the data analysis process
  - Identify tools and skills required to conduct data analysis

OUTLINE

Chapter 1 – General Introduction

Chapter 2 - Data Wrangling

Chapter 3 - Exploratory Data Analysis

Chapter 4 - Model Development

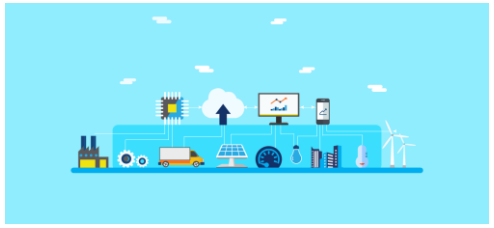
Chapter 5 - Working with Data in Python

What is Big Data Analysis?

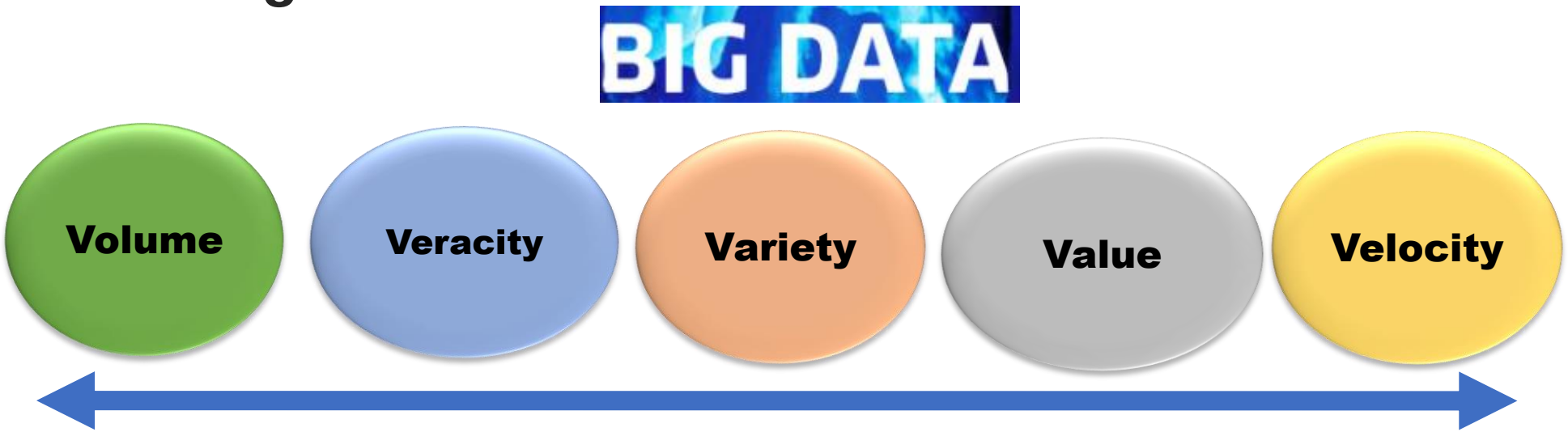
Every day we generate at least 5 trillion bytes of data

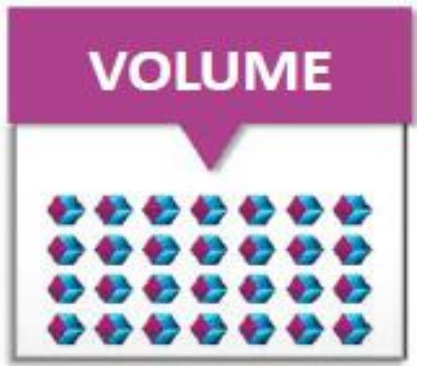
97% of the data in the world has been created in recent years (2014-2021)

- Sensors used to collect climate, traffic, consumption information;
- Smart cities, Internet des Objets (IoT);
- Social media messages
- Digital images and videos published online;
- Transactional recordings of online purchases;
- GPS signals from mobile phones.
- Ect.



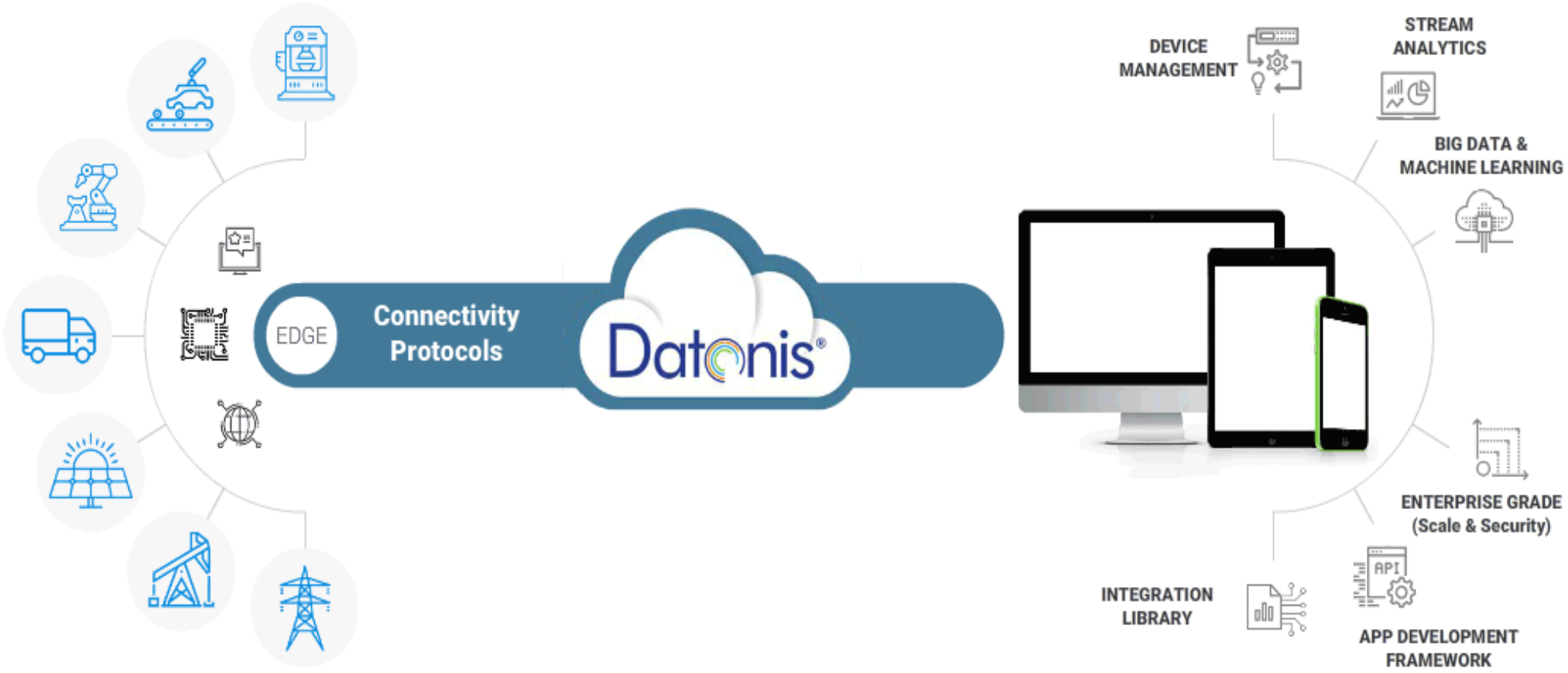
What is Big Data?





Big Data

:

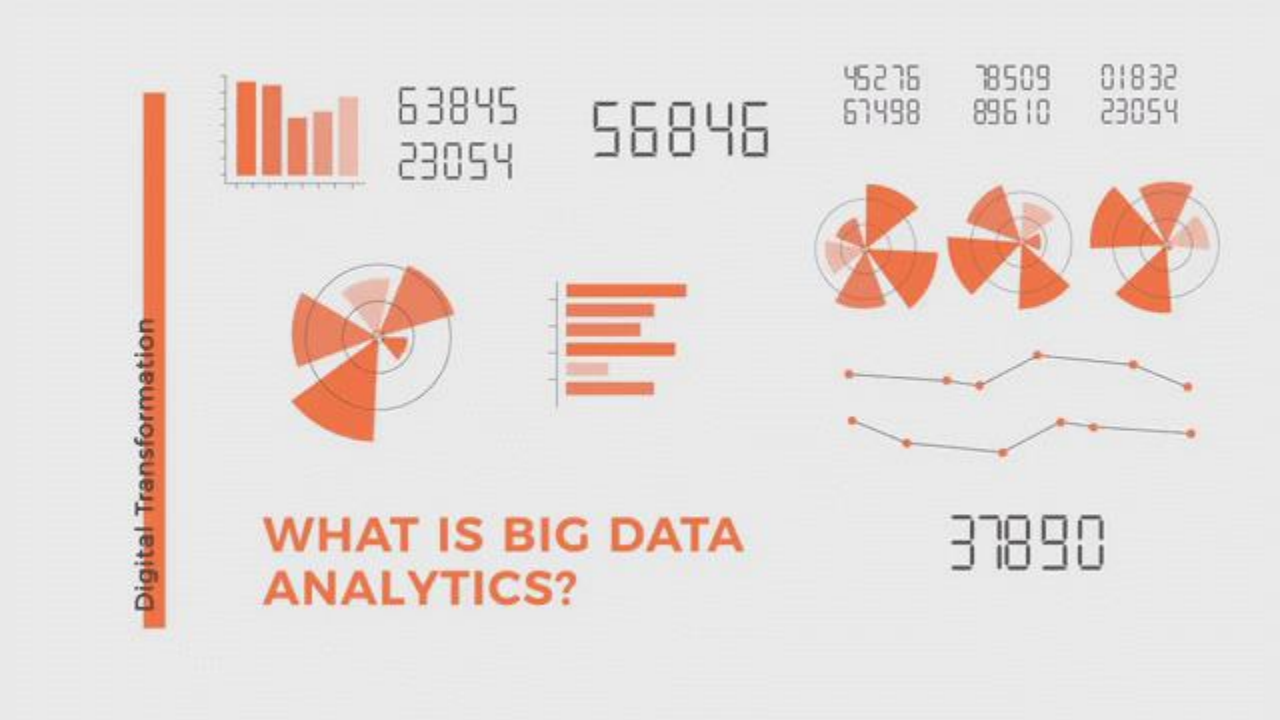


https://altizon.com/datonis-iiot-platform/

What is Data Analytic?

Data analytics is the science of analyzing **raw data** in order to make conclusions about that information.

Data analytics (DA) is the process of examining data sets in order to draw conclusions about the information they contain, increasingly with the aid of specialized systems and software. Data analytics technologies and techniques are widely used in commercial industries to enable organizations to make more-informed business decisions and by scientists and researchers to verify or disprove scientific models, theories and hypotheses.



Chapter 1 – General Introduction

1. Problem: Used car Appraisal

Can we estimate the price of used car?

2. Why Data Analysis?

- Data is everywhere.
- Data analysis/data science helps us answer questions from data.
- **Data analysis** plays an important role in:
  - Discovering useful information
  - Answering questions
  - Predicting future or the unknown

How much Money should he Sell his car for?

Aurelle wants to sell his car

The price he sets should not be too high, But not too low either.

2. Why Data Analysis?

Estimate used car prices

How can we help Tom determine the best price for his car ?

- Is there data on the prices of other cars and their characteristics?
- What features of cars affect their prices?

✓ Color? Brand? Horsepower? Something else?

- Asking the right questions in terms of data

2. Understanding the data

```
3,?,alfa-romero,gas,std,two,convertible,rwd,front,88.60,168.80,64.10,48.80,2548,dohc,four,130,mpfi,3.47,2.68,9.00,111,5000,21,27,13495
3,?,alfa-romero,gas,std,two,convertible,rwd,front,88.60,168.80,64.10,48.80,2548,dohc,four,130,mpfi,3.47,2.68,9.00,111,5000,21,27,16500
1,?,alfa-romero,gas,std,two,hatchback,rwd,front,94.50,171.20,65.50,52.40,2823,ohcv,six,152,mpfi,2.68,3.47,9.00,154,5000,19,26,16500
2,164,audi,gas,std,four,sedan,fwd,front,99.80,176.60,66.20,54.30,2337,ohc,four,109,mpfi,3.19,3.40,10.00,102,5500,24,30,13950
2,164,audi,gas,std,four,sedan,4wd,front,99.40,176.60,66.40,54.30,2824,ohc,five,136,mpfi,3.19,3.40,8.00,115,5500,18,22,17450
2,?,audi,gas,std,two,sedan,fwd,front,99.80,177.30,66.30,53.10,2507,ohc,five,136,mpfi,3.19,3.40,8.50,110,5500,19,25,15250
1,158,audi,gas,std,four,sedan,fwd,front,105.80,192.70,71.40,55.70,2844,ohc,five,136,mpfi,3.19,3.40,8.50,110,5500,19,25,17710
1,?,audi,gas,std,four,wagon,fwd,front,105.80,192.70,71.40,55.70,2954,ohc,five,136,mpfi,3.19,3.40,8.50,110,5500,19,25,18920
1,158,audi,gas,turbo,four,sedan,fwd,front,105.80,192.70,71.40,55.90,3086,ohc,five,131,mpfi,3.13,3.40,8.30,140,5500,17,20,23875
0,?,audi,gas,turbo,two,hatchback,4wd,front,99.50,178.20,67.90,52.00,3053,ohc,five,131,mpfi,3.13,3.40,7.00,160,5500,16,22,?
2,192,bmw,gas,std,two,sedan,rwd,front,101.20,176.80,64.80,54.30,2395,ohc,four,108,mpfi,3.50,2.80,8.80,101,5800,23,29,16430
0,192,bmw,gas,std,four,sedan,rwd,front,101.20,176.80,64.80,54.30,2395,ohc,four,108,mpfi,3.50,2.80,8.80,101,5800,23,29,16925
0,188,bmw,gas,std,two,sedan,rwd,front,101.20,176.80,64.80,54.30,2710,ohc,six,164,mpfi,3.31,3.19,9.00,121,4250,21,28,20970
0,188,bmw,gas,std,four,sedan,rwd,front,101.20,176.80,64.80,54.30,2765,ohc,six,164,mpfi,3.31,3.19,9.00,121,4250,21,28,21105
1,?,bmw,gas,std,four,sedan,rwd,front,103.50,189.00,66.90,55.70,3055,ohc,six,164,mpfi,3.31,3.19,9.00,121,4250,20,25,24565
0,?,bmw,gas,std,four,sedan,rwd,front,103.50,189.00,66.90,55.70,3230,ohc,six,209,mpfi,3.62,3.39,8.00,182,5400,16,22,30760
0,?,bmw,gas,std,two,sedan,rwd,front,103.50,193.80,67.90,53.70,3380,ohc,six,209,mpfi,3.62,3.39,8.00,182,5400,16,22,41315
0,?,bmw,gas,std,four,sedan,rwd,front,110.00,197.00,70.90,56.30,3505,ohc,six,209,mpfi,3.62,3.39,8.00,182,5400,15,20,36880
2,121,chevrolet,gas,std,two,hatchback,fwd,front,88.40,141.10,60.30,53.20,1488,l,three,61,2bbl,2.91,3.03,9.50,48,5100,47,53,5151
```

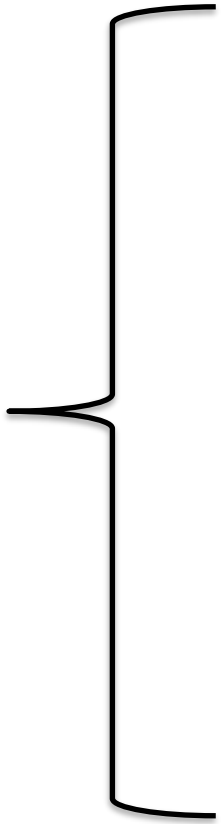
2. Understanding the data

Each of the attributes in the dataset

No.	Attribute name	attribute range	No.	Attribute name	attribute range
1	symboling	-3, -2, -1, 0, 1, 2, 3.	14	curb-weight	continuous from 1488 to 4066.
2	normalized-losses	continuous from 65 to 256.	15	engine-type	dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
3	make	audi, bmw, etc.	16	num-of-cylinders	eight, five, four, six, three, twelve, two.
4	fuel-type	diesel, gas.	17	engine-size	continuous from 61 to 326.
5	aspiration	std, turbo.	18	fuel-system	1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
6	num-of-doors	four, two.	19	bore	continuous from 2.54 to 3.94.
7	body-style	hardtop, wagon, etc.	20	stroke	continuous from 2.07 to 4.17.
8	drive-wheels	4wd, fwd, rwd.	21	compression-ratio	continuous from 7 to 23.
9	engine-location	front, rear.	22	horsepower	continuous from 48 to 288.
10	wheel-base	continuous from 86.6 120.9.	23	peak-rpm	continuous from 4150 to 6600.
11	length	continuous from 141.1 to 208.1.	24	city-mpg	continuous from 13 to 49.
12	width	continuous from 60.3 to 72.3.	25	highway-mpg	continuous from 16 to 54.
13	height	continuous from 47.8 to 59.8.	26	price	continuous from 5118 to 45400.

3. Python packages for data Analysis

1. **Scientifics Computing** Libraries



Pandas
(Data structures & tools)

NumPy
(Arrays & matrices)

SciPy
(Integrals, solving differential equations, optimization)

3. Python packages for data Analysis

2. Visualization
Libraries

Matplotlib
(plots & graphs, most popular)

Seaborn
(plots : heat maps, time series, violin plots)

3. Python packages for data Analysis

3. **Algorithmic** Libraries

Scikit-learn

(Machine Learning ; regression, Classification,…)

Statsmodels

(Explore data, estimate statistical models, and perform statistical tests.)

3. Python packages for data Analysis

NumPy:

- introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects
- provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance
- many other python libraries are built on NumPy

SciPy:

- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- part of SciPy Stack
- built on NumPy

3. Python packages for data Analysis

Pandas:

- adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)
- provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- allows handling missing data

SciKit-Learn:

- provides machine learning algorithms: classification, regression, clustering, model validation etc.
- built on NumPy, SciPy and matplotlib

3. Python packages for data Analysis

matplotlib:

- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization

Seaborn:

- based on matplotlib
- provides high level interface for drawing attractive statistical graphics
- Similar (in style) to the popular ggplot2 library in R

4. Importing Data

- Process of loading and reading data into Python from various resources.
    - **Two important properties:**
        - Format
            - Various formats: .csv, .json, xlsx, .hdf ...
        - File Path of dataset

            - Computer: /Desktop/mydata.csv

            - Internet: IBM Cloud, Google Cloud, AWS, etc,

4. Importing and Save Data

- **Import pandas as pd**
- **df=pd.cvs_read('dataset.csv')**

- **df** prints the entire dataframe (not recommended for large datasets)
- **df .head (n)** to show the first *n* rows of data frame.
- **df .tail (n)** shows the bottom *n* rows of data frame.
- Replace default header (by df.columns = headers)

Data Format	Read	Save
csv	pd.read_csv()	df.to_csv()
json	pd.read_json()	df.to_json()
Excel	pd.read_excel()	df.to_hdf()
s1l	pd.read_sql()	df.to_sql()

4. Basic insights from the data

- Understand your data before you begin any analysis
- Should check:

✓ Data Types

✓ Data Distribution

- Locate potential issues with the data

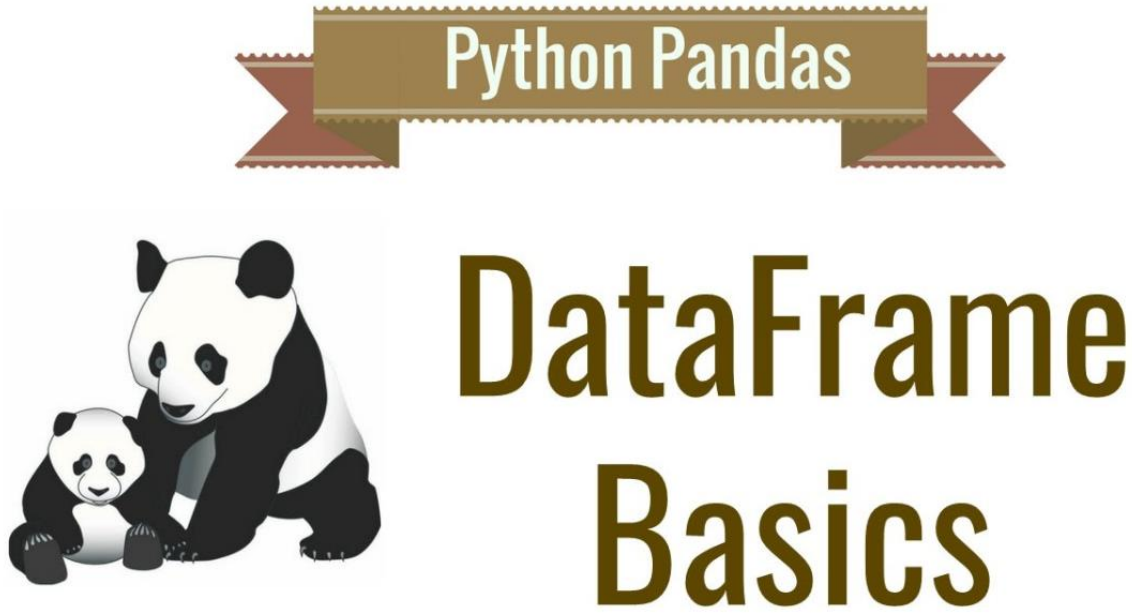
Why check data types?

- Potential info and type mismatch
- Compatibility with python methoods

Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs(see below), pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

4. Basic insights from the data

- In pandas, we use *dataframe.dtypes* to check data types : ***df.dtypes***
- Returns a statistical summary : ***df.describe()***
- Provides full summary statistics : ***df.describe (include=‘’all”)***
- *dataframe.info* provides a concise summary of your DataFrame. : ***df.info***



4. Basic insights from the data

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

5. Lab 1

- Install all dependancies (Anaconda with pandas and numpy)
- Jupyter notebook or Spyder?
- Lab 1: Introduction

By the end of this lab, you will have learned the basics of Data Analysis.

Chapter 2 - Data Wrangling

1. Data Pre-processing

Also known as:

Data Cleaning, Data Wranging

The process of converting or mapping data from the initial "raw" form into another format, in order to prepare the data for further analysis.

2. Learning Objectives

- **Identify and handle missing values**
- **Data Formating**
- **Data Normalization (centering /scaling)**
- **Data Binning**
- **Turning Categorical values to numeric variables**

Chapter 2 - Data Wrangling

3. Simple Dataframe Operations

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0
5	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5
6	1	158	audi	gas	std	four	sedan	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5
7	1	?	audi	gas	std	four	wagon	fwd	front	105.8	...	136	mpfi	3.19	3.40	8.5
8	1	158	audi	gas	turbo	four	sedan	fwd	front	105.8	...	131	mpfi	3.13	3.40	8.3

4. Dealing with Missing Values in Python

- What is missing value?
- Missing values occur when no data value is stored for a variable (feature) in an observation.
- Could be represented as "?", "N/A", 0 or just a blank cell.

	symboling	Normalized-losses	make	Fuel-type	aspiration	Num-Of-doors	Body-style	Drive-wheels	Engine-location
0	3	NaN	Alfa-romero	gas	std	two	convertible	rwd	front

5. How to deal with missing data?

Check with the data collection source

Drop the missing values

- drop the variable
- drop the data entry

Replace the missing values

- Replace it with an average (of similar datapoints)
- Replace it by frequency
- Replace it based on other functions

Replace the missing values

6. How to drop missing values in Python?

- Use dataframes.dropna () :

Highway-mpg	price
---	---
20	23875
22	NaN
29	16430
---	---

Highway-mpg	price
---	---
20	23875
22	NaN
29	16430
---	---

Highway-mpg	price
---	---
20	23875
29	16430
---	---

6. How to drop missing values in Python?

As an example, assume that we want to replace the missing values of the variable 'normalized-losses' by the mean value of the variable.

Normalized-losses	make
---	---
164	audi
164	audi
NaN	audi
158	audi

6. How to drop missing values in Python?

There are a number of methods to deal with missing values in the data frame:

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

7. Data Formatting

- Data are usually collected from different places and stored in different formats.
- Bringing data into a common standard of expression allows users to make meaningful comparison.

Non-formatted:

- confusing
- hard to aggregate
- hard to compare

City
NY
New York
N.Y
N.Y

City
New York
New York
New York
New York

Formatted:

- more clear
- easy to aggregate
- easy to compare

7. Data Formatting

- Convert ‘’mpg” to ‘’L/100km” in Car dataset.

City-mpg
21
21
19

City-L/100km
11.2
11.2
12.4

```
df["price"].tail(5)

200    16845
201    19045
202    21485
203    22470
204    22625
Name: price, dtype: object
```

df[‘’city-mpg”]= 235/df[‘’city-mpg”]
Df.rename(columns = {‘’city-mpg”: ‘’city-L/100km”}, implace=True)

- Sometimes the wrong data type is assigned to a feature.

7. Data Formatting

- **There are many data types in pandas**

- **objects: « A », « Hello »..**
- **Int64: 1,3,5**
- **Float64: 2.123,632.31,0.12**

Example: convert data type to integer in column «price »

Df[‘’price’’]= df[‘’price’’] .astype (‘’int ’’)

To *identify* data types:

- Use **dataframe, dtypes ()** to identify data type.

To *convert* data types:

- Use **dataframe, astypes ()** to convert data type.

8. Data Normalization

- Uniform the features value with different range.

Length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
171.2	65.5	52.4
176.6	66.2	54.3
176.6	66.4	54.3
177.3	66.3	53.1
192.7	71.4	55.7
192.7	71.4	55.7
192.7	71.4	55.9

scale	[150,250]	[50,100]	[50,100]
impact	large	small	small

8. Data Normalization

age	income
20	100000
30	20000
40	500000

age	income
0,2	0.2
0,3	0,04
0,4	1

Not-normalized

- ‘’age” and ‘’income” are in different range.
- Hard to compare
- ‘’income” will influence the result more

Normalized

- Similar value range.
- Similar intrinsic influence on analytical model.

8. Data Normalization

Several approaches for normalization:

① $x_{new} = \frac{x_{old}}{x_{max}}$

Simple Feature scaling

② $x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$

Min-Max

③ $x_{new} = \frac{x_{old} - \mu}{\sigma}$

Z-score

With Pandas:

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
180.0	65.5	52.4
...	...	...

→

length	width	height
0.81	64.1	48.8
0.81	64.1	48.8
0.87	65.5	52.4
...	...	...

```
df["length"] = df["length"]/df["length"].max()
```

9. Binning

- Binning: Grouping of values into "bins"
- Converts numeric into categorical variables
- Group a set of numerical values into a set of "bins".

- "price" is an feature range from 5,000 to 45,500, he wants to have a **better representation** of price.

9. Binning

price
13495
16500
18920
41315
5151
6295
...

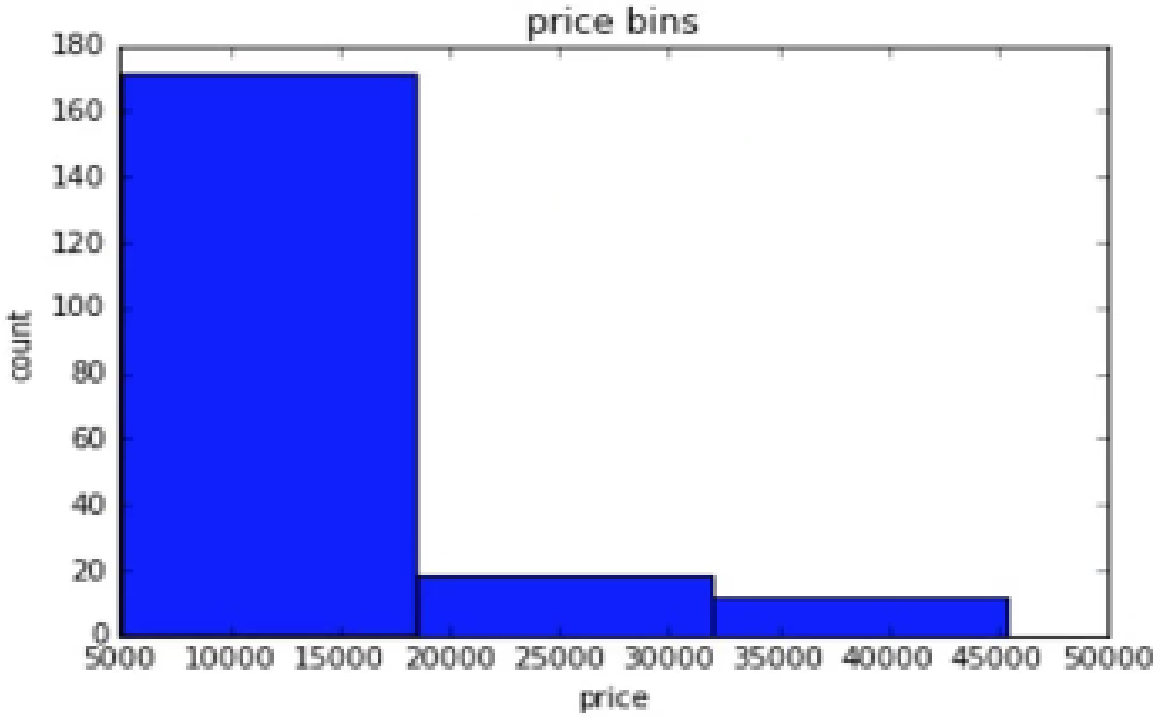
price	price-binned
13495	Low
16500	Low
18920	Medium
41315	High
5151	Low
6295	Low
...	...

```
binwidth = int((max(df["price"])-min(df["price"]))/4 )
bins = range(min(df["price"]), max(df["price"]), binwidth)

group_names = ['Low', 'Medium', 'High']

df['price-binned'] = pd.cut(df['price'], bins, labels=group_names)
```

9. Binning

Visualizing binned data

- E.g., Histograms



10. Lab 2

By the end of this Lab2, you will have learned the basics of Data Wrangling with these knowledge

- Identify and handle missing values
- Identify missing values
- Deal with missing values
- Correct data format
- Data standardization
- Data Normalization (centering/scaling)
- Binning
- Indicator variable

Chapter 3 - Exploratory Data Analysis

1. Exploratory Data Analysis (EDA)

- Preliminary step in data analysis to:
  - Summarize main characteristics of the data
  - Gain better understanding of the data set
  - Uncover relationships between variables
  - Extract important variables

- Question:

  « what are the characteristics that have the most impact on the car price? »

2. Learning Objectives)

In this module you will learn about:

- Descriptive Statistics
- GroupBy
- ANOVA
- Correlation
- Correlation - Statistics

3. Descriptive Statistics

- Describe basic features of data
- Giving short summaries about the sample and measures of the data
- Summarize statistics using pandas **describe()** method

```
df.describe()
```

	Unnamed: 0	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke
count	201.000000	201.000000	164.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000	201.000000
mean	100.000000	0.840796	122.000000	98.797015	174.200995	65.889055	53.766667	2555.666667	126.875622	3.319154	3.256766
std	58.167861	1.254802	35.442168	6.066366	12.322175	2.101471	2.447822	517.296727	41.546834	0.280130	0.316049
min	0.000000	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000
25%	50.000000	0.000000	NaN	94.500000	166.800000	64.100000	52.000000	2169.000000	98.000000	3.150000	3.110000
50%	100.000000	1.000000	NaN	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	3.310000	3.290000
75%	150.000000	2.000000	NaN	102.400000	183.500000	66.600000	55.500000	2926.000000	141.000000	3.580000	3.410000

3. Descriptive Statistics

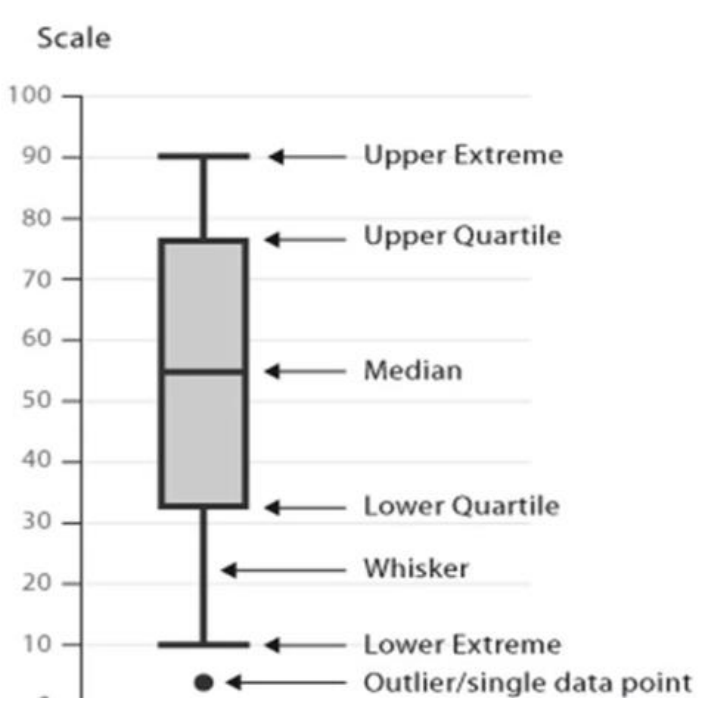
- Summarize statistics using pandas **value_counts** method

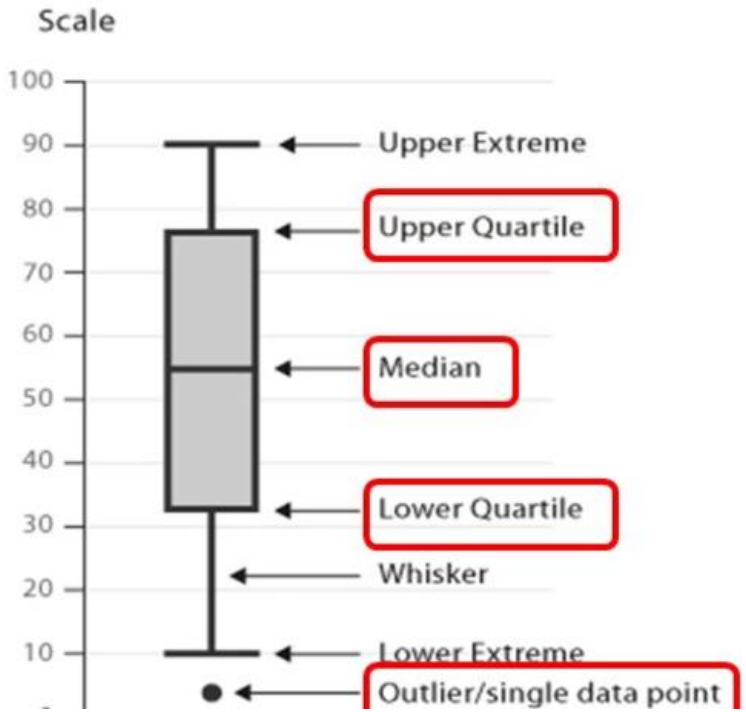
```
drive_wheels_counts=df["drive-wheels"].values_counts()

drive_wheels_counts.rename(colunms={'drive-wheels':'value_counts'  inplace=True)
drive_wheels_counts.index.name=  'drive-wheels'
```

	value_counts
drive-wheels	
fwd	118
rwd	75
4wd	8

3. Descriptive Statistics

Box Plots

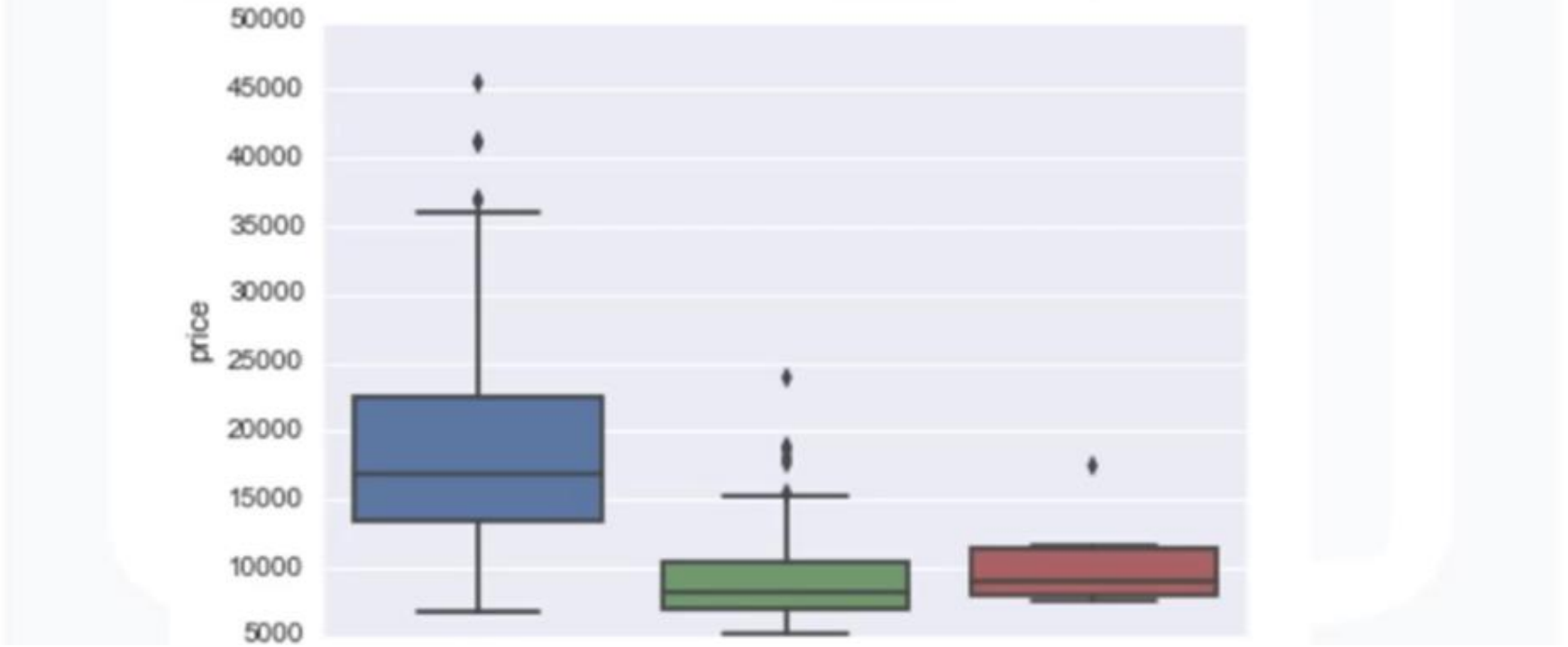




3. Descriptive Statistics

Box Plots Example

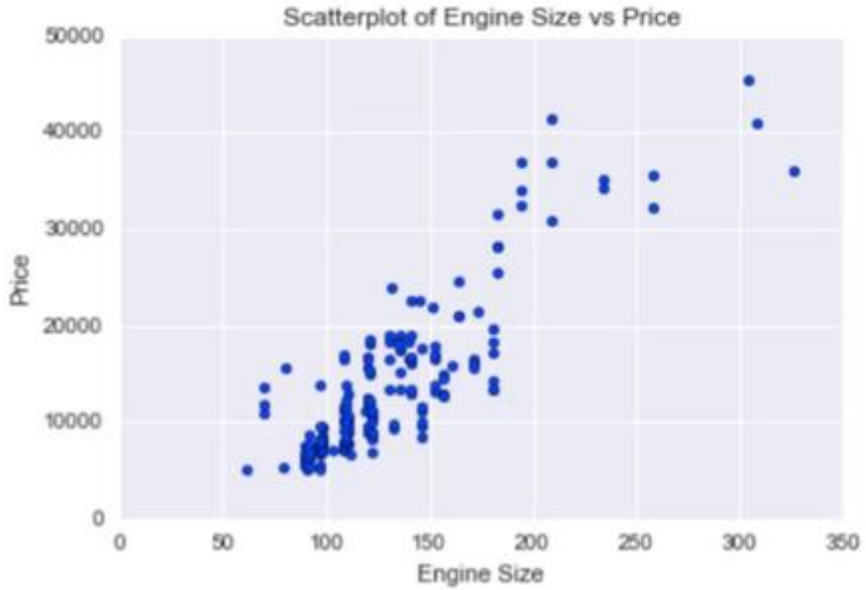
```
sns.boxplot(x= "drive-wheels", y= "prices",data=df)
```



3. Descriptive Statistics

Scatter Plot

- Each observation represented as a point
- Scatter plots show the relationship between two variables :
    1. Predictor/independent variables on x-axis
    2. Target/dependent variables on y-axis



4. GroupBy

- Use Panda **dataframe. Groupby()** method:
  - Can be applied on categorical varibles
  - Group data into categories
  - Single or multiple variables

```
df_test = df['drive-wheels', 'body-style', 'price']
```

```
df_grp = df_test.groupby(['drive-wheels', 'body-style'], as_index=False).mean()
df_grp
```

	drive-wheels	body-style	price
0	4wd	hatchback	7603.000000
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	23949.600000
9	rwd	hardtop	24202.714286

4. GroupBy

Pandas method – Pivot()

- One variable displayed along the columns and the other variable displayed along the rows

```
df_pivot = df_grp.pivot(index= 'drive-wheels', columns='body-style')
```

Just with one line of code and by using the pandas pivot method, we can pivot the "body style" variable so it is displayed along the columns and the "drive wheels" will be displayed along the rows.

	price				
body-style	**convertible**	**hardtop**	**hatchback**	**sedan**	**wagon**
drive-wheels					
4wd	20239.229524	20239.229524	7603.000000	12647.333333	9095.750000
fwd	11595.000000	8249.000000	8396.387755	9811.800000	9997.333333
rwd	23949.600000	24202.714286	14337.777778	21711.833333	16994.222222

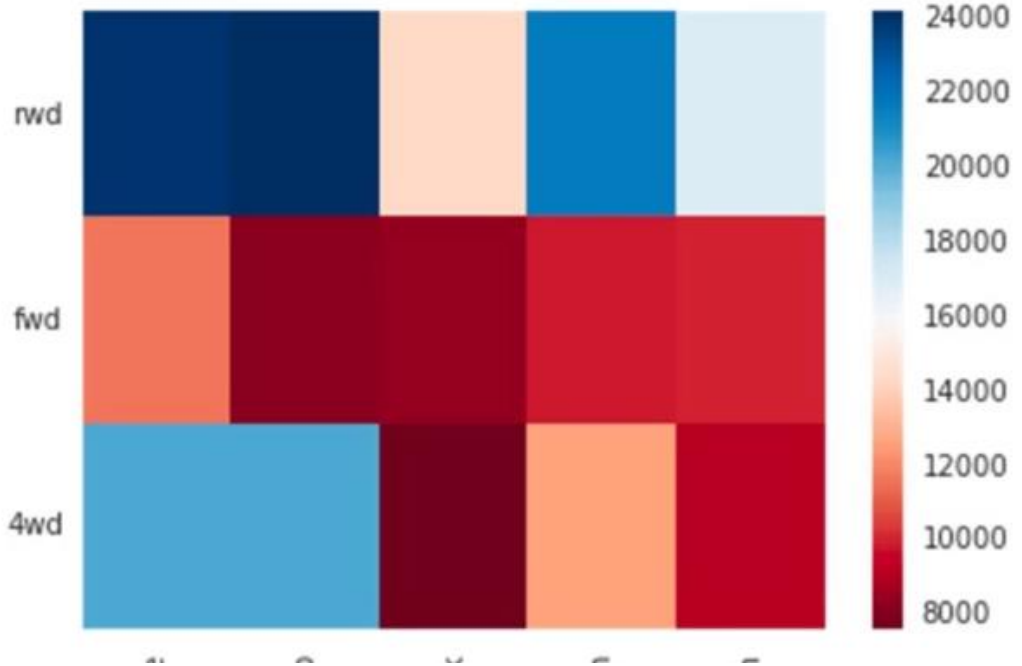
5. Heatmap

Another way to represent the pivot table is using a heatmap plot.

Heat map takes a rectangular grid of data and assigns a color intensity based on the data value at the grid points. It is a great way to plot the target variable over multiple variables and through this get visual clues of the relationship between these variables and the target.

- Plot target variable over multiple variables

```
plt.pcolor(df_pivot, cmap='RdBBu')
plt.colorbar()
plt.show()
```



6. Analysis of Variance (ANOVA)

- **Statistical comparison of groups**
- **Example: average price of different vehicle makes.**

ANOVA is a statistical test that stands for Analysis of Variance;.

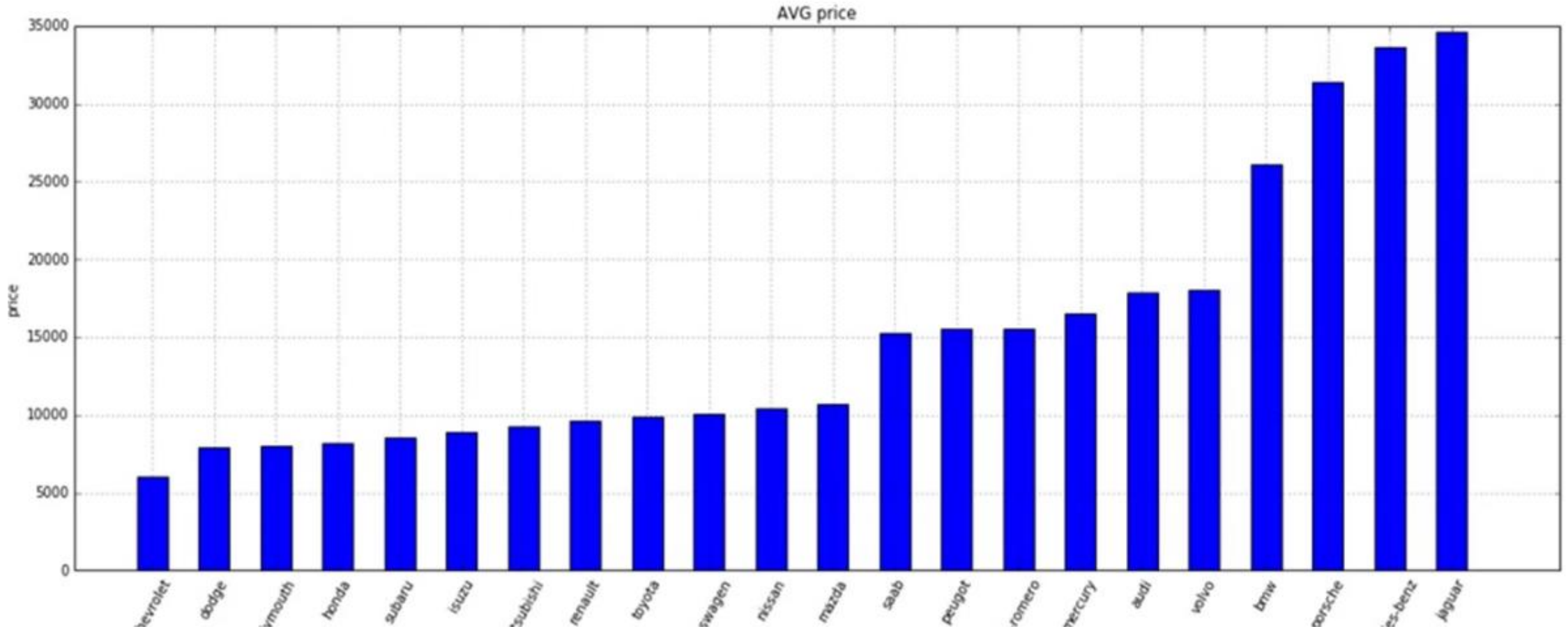
Assume that we want to analyze a categorical variable and see the correlation among different categories.

For example, consider the car dataset, the question we may ask is, how different categories of the Make feature (as a categorical variable) has impact on the price?

Chapter 3 - Exploratory Data Analysis

6. Analysis of Variance (ANOVA)

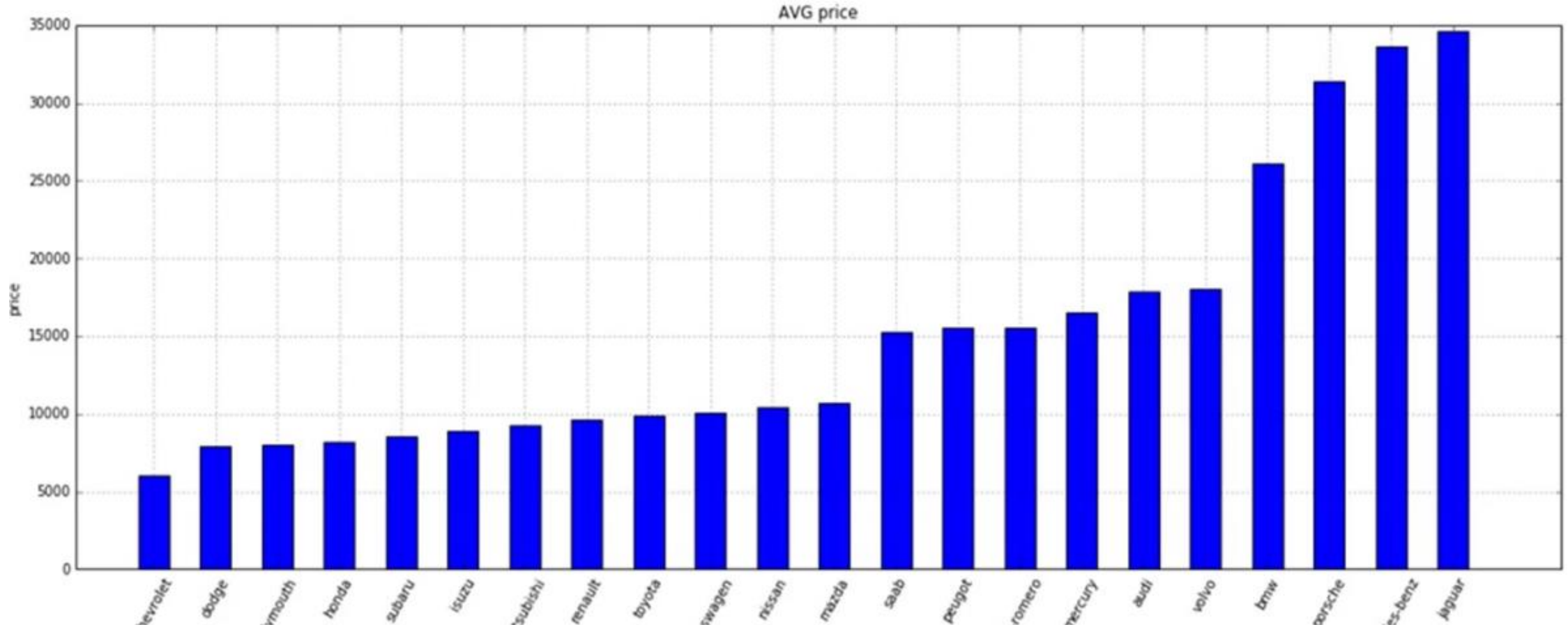
- Statistical comparison of groups



6. Analysis of Variance (ANOVA)

- **An**alysis **O**f **Va**riance (ANOVA)
- Why do we perform ANOVA?
  - Finding correlation between different groups of a categorical variable.
- What we obtain from ANOVA?
  - F-test score: variation between sample group means divided by variation within sample group
  - p-value: confidence degree

6. Analysis of Variance (ANOVA)



Getting back to our example, the bar chart shows the average price for different categories

7. Correlation

❑ Pearson Correlation

- Measure the strength of the correlation between tow features.
  - Correlation coefficient
  - P-value

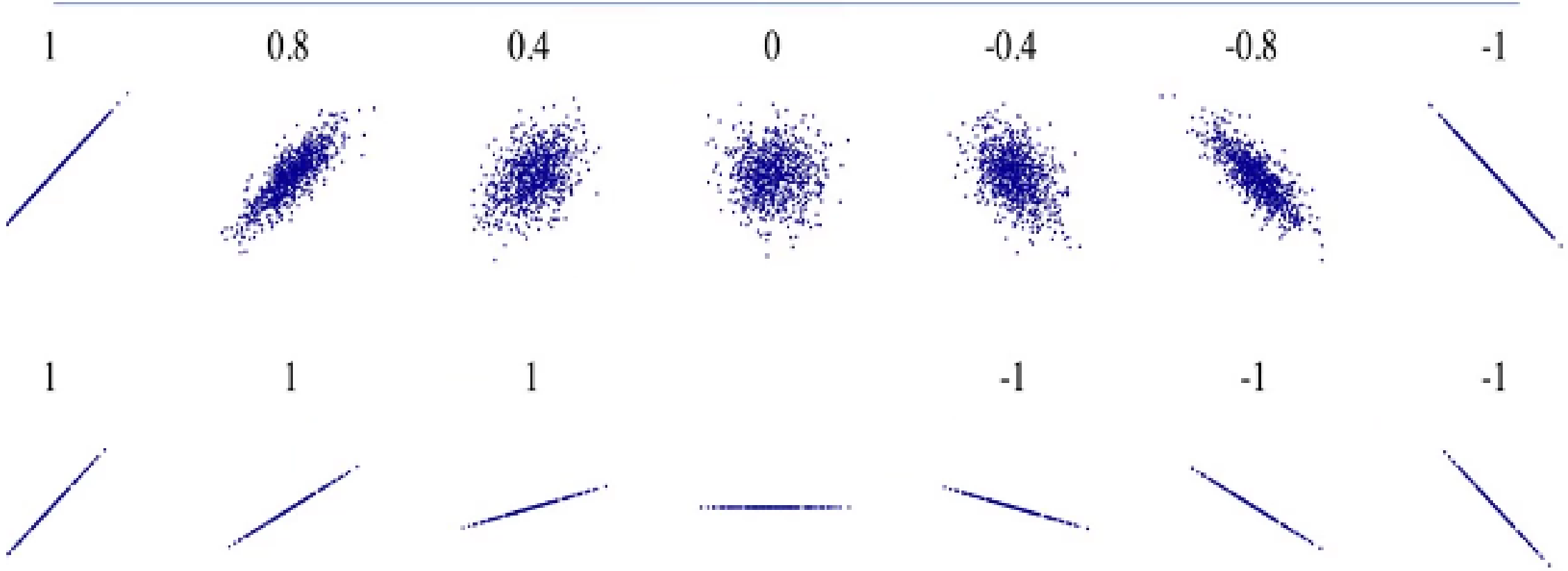
- Correlation coefficient
  - Close to +1: Large Positive relationship
  - Close to -1: Large Negative relationship
  - Close to 0: No relationship

- P-value
  - P-value<0,001 **Strong** certainty in the result
  - P-value<0,05 **Moderate** certainty in the result
  - P-value<0,1 **Weak** certainty in the result
  - P-value>0,1 **No** certainty in the result

- **Strong Correlation:**
  - **Correlation coefficient close to 1 or -1**
  - **P value less than 0.001**

7. Correlation

❑ Pearson Correlation



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

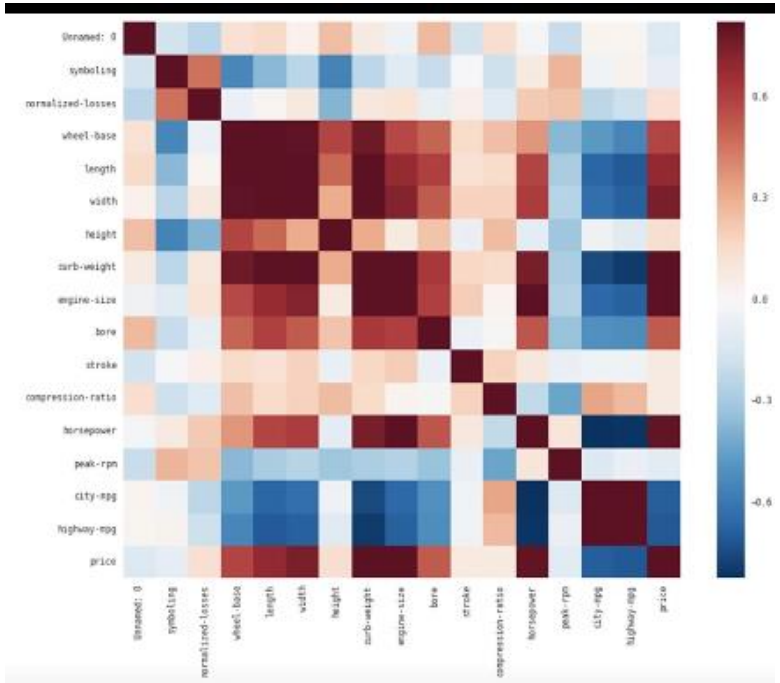
7. Correlation

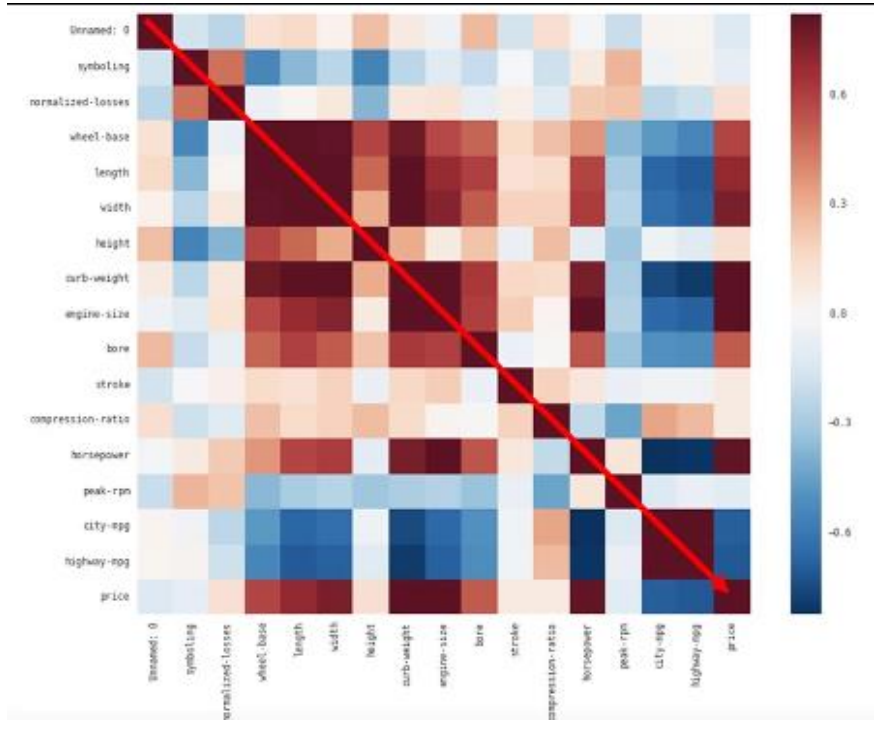
❑ Pearson Correlation- Example

```
Pearson_coef,p_value=stats.personr[['horsepower'],df['price']]
```

- Pearson correlation: 0.81
- P-value : 9.35 e-48

❑ Correlation-Heatmap





Chapter 4 - Model Development

1. Introduction

- Linear regression will refer to one independent variable to make a prediction

- Multiple Linear Regression will refer to multiple independent variables to make a prediction

Chapter 4 - Model Development

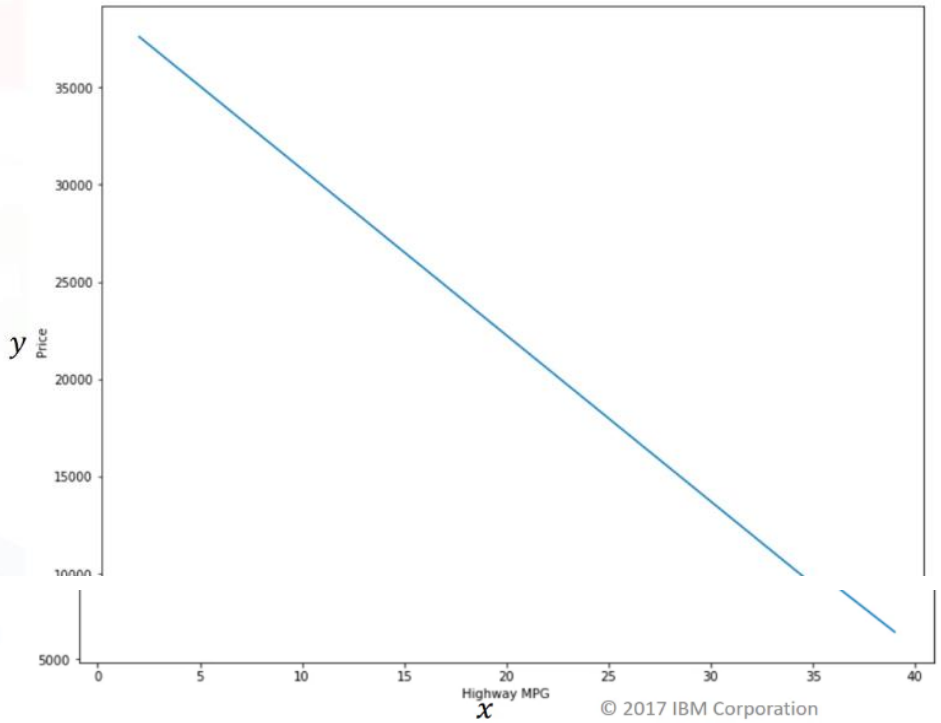
2. Simple Linear Regression

1. The predictor (independent) variable - x
2. The target (dependent) variable - y

$$y = b_0 + b_1 x$$

- b_0: the intercept
- b_1 : the slope

$$y = 38423 - 821x$$

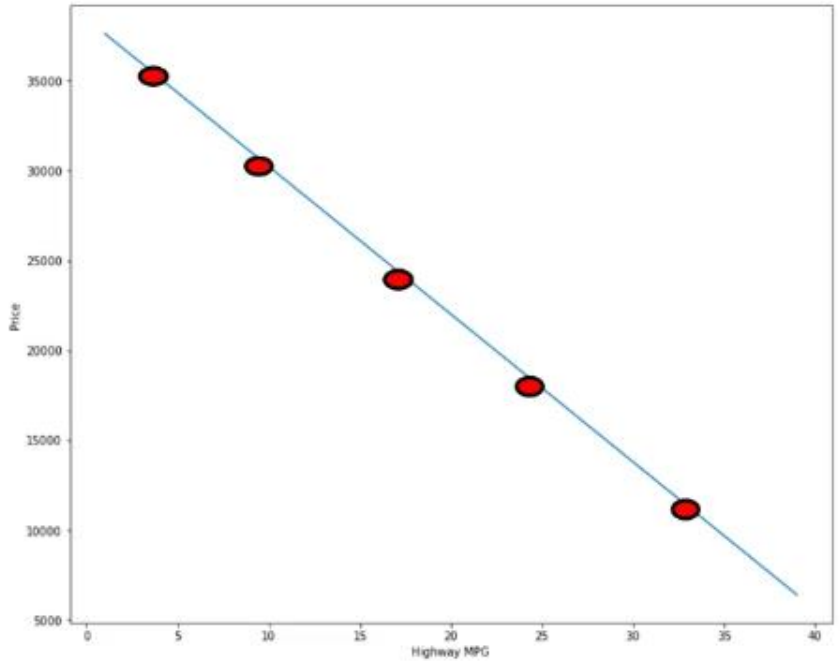


- The parameter b one is the slope When we fit or train the model, we will come up with these parameters. This step requires lots of math

Chapter 4 - Model Development

2. Simple Linear Regression

❑Fit



In order to determine the line, we take data points from our data set marked in red here.

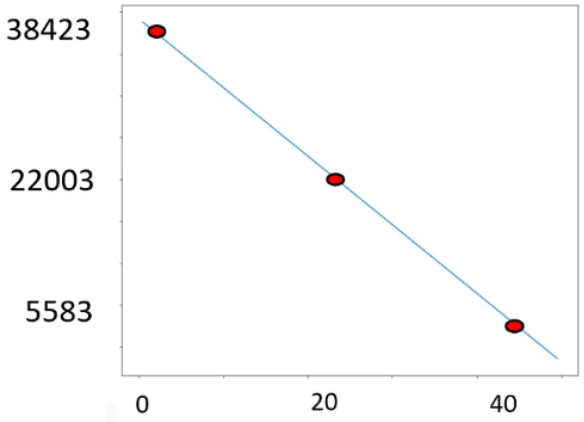
(b_0, b_1)

We then use these training points to fit our model; the results of the training points
We then use these training points to fit our model; the results of the training points
are the parameters. We usually store the data points in two dataframe or numpy arrays

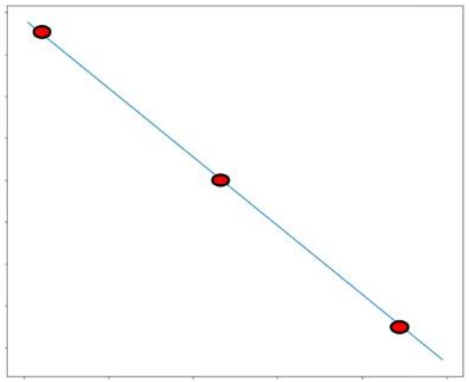
Chapter 4 - Model Development

2. Simple Linear Regression

❑ Fit



$$X = \begin{bmatrix} \end{bmatrix} \quad Y = \begin{bmatrix} \end{bmatrix}$$



$$X = \begin{bmatrix} 0 \\ 20 \\ 40 \end{bmatrix} \quad Y = \begin{bmatrix} 38423 \\ 22003 \\ 5583 \end{bmatrix}$$

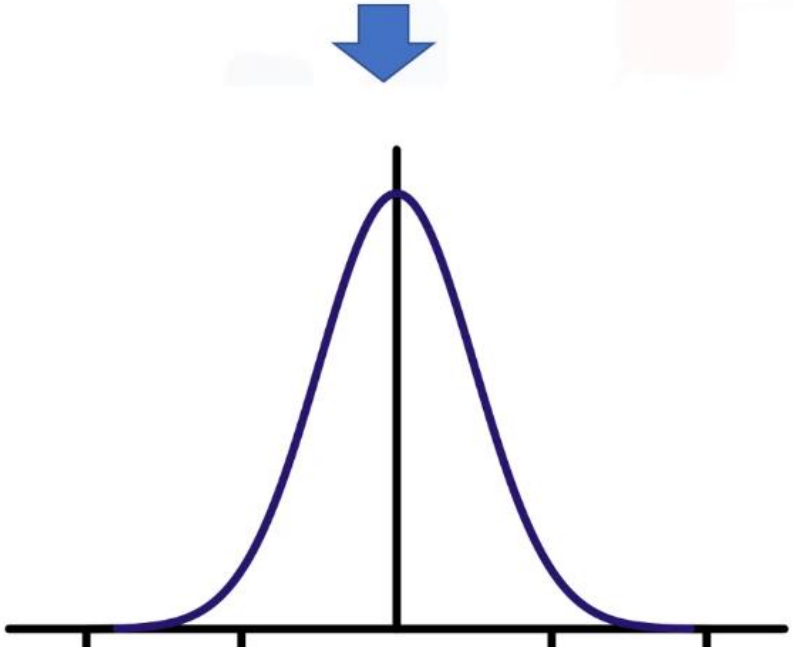
The value we would like to predict is called the target that we store in the array Y, we store the dependent variable in the dataframe or array X. Each sample corresponds to a different row in each dataframe or array.

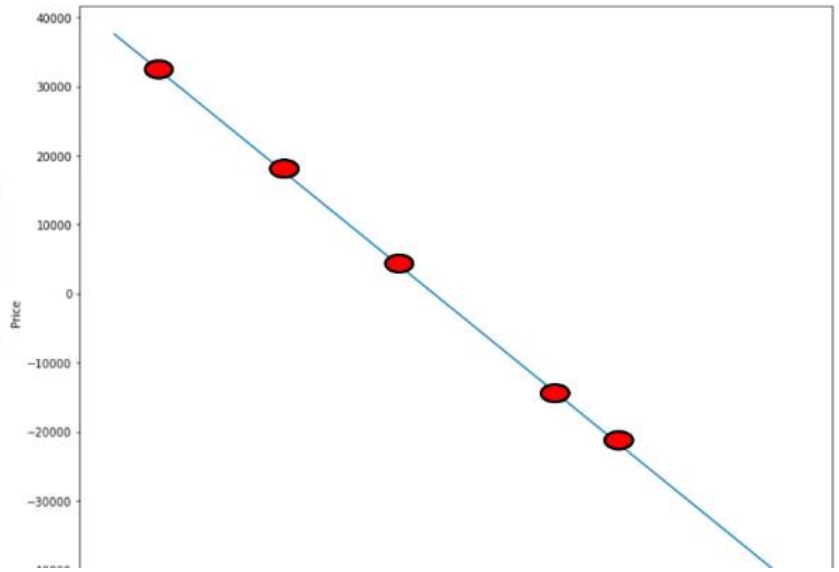
Chapter 4 - Model Development

2. Simple Linear Regression

❑ Fit

Much people pay for a car, for example, make or how old the car is. In this model, this uncertainty is taken into account by assuming a small random value is added to the point on the line; this is called noise. The figure on the left shows the distribution

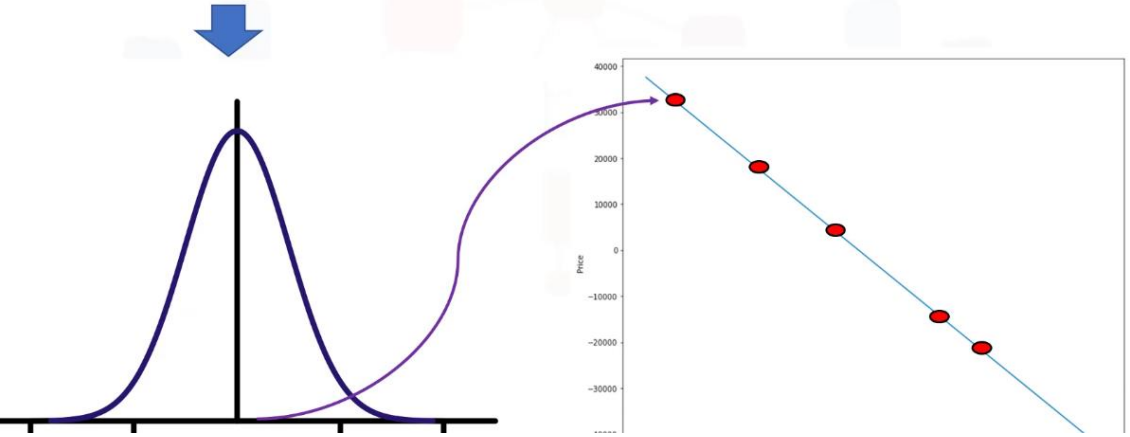


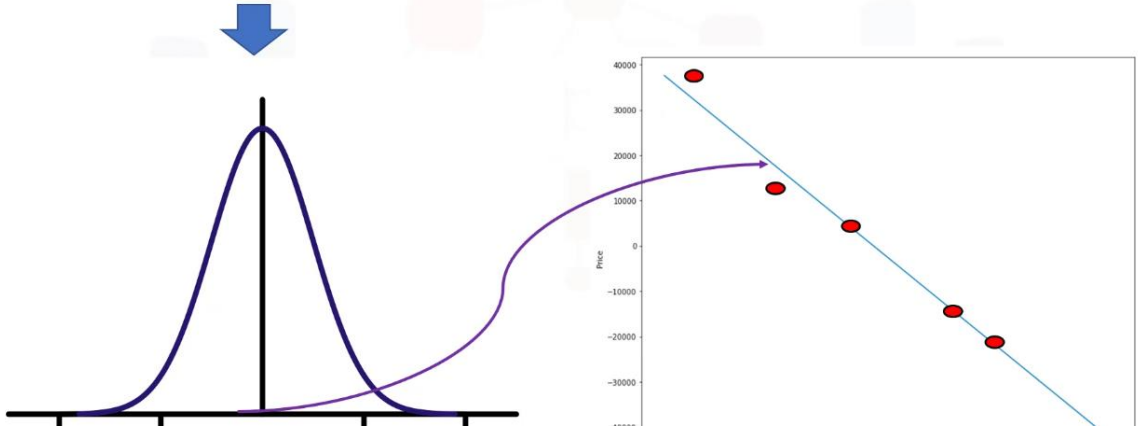


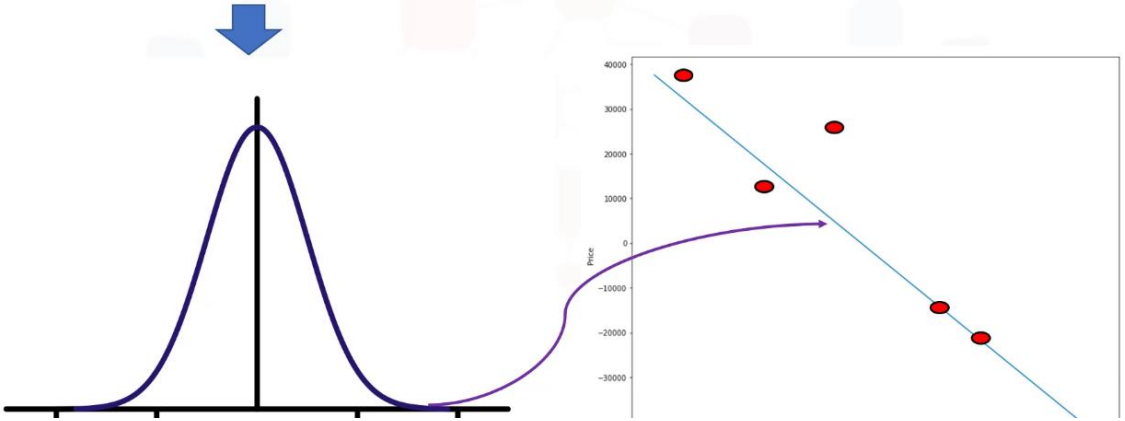
Chapter 4 - Model Development

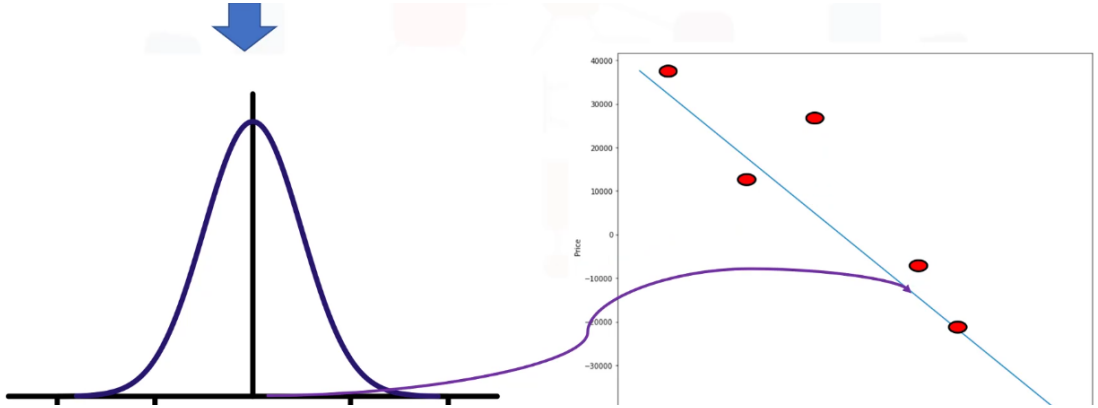
2. Simple Linear Regression

❑ Fit









values added are near zero. We can summarize the process like this:

Chapter 4 - Model Development

2. Simple Linear Regression

❑ Fit

We have a set of training points - We use these training points to fit or train the model and get parameters - We then use these parameters in the model

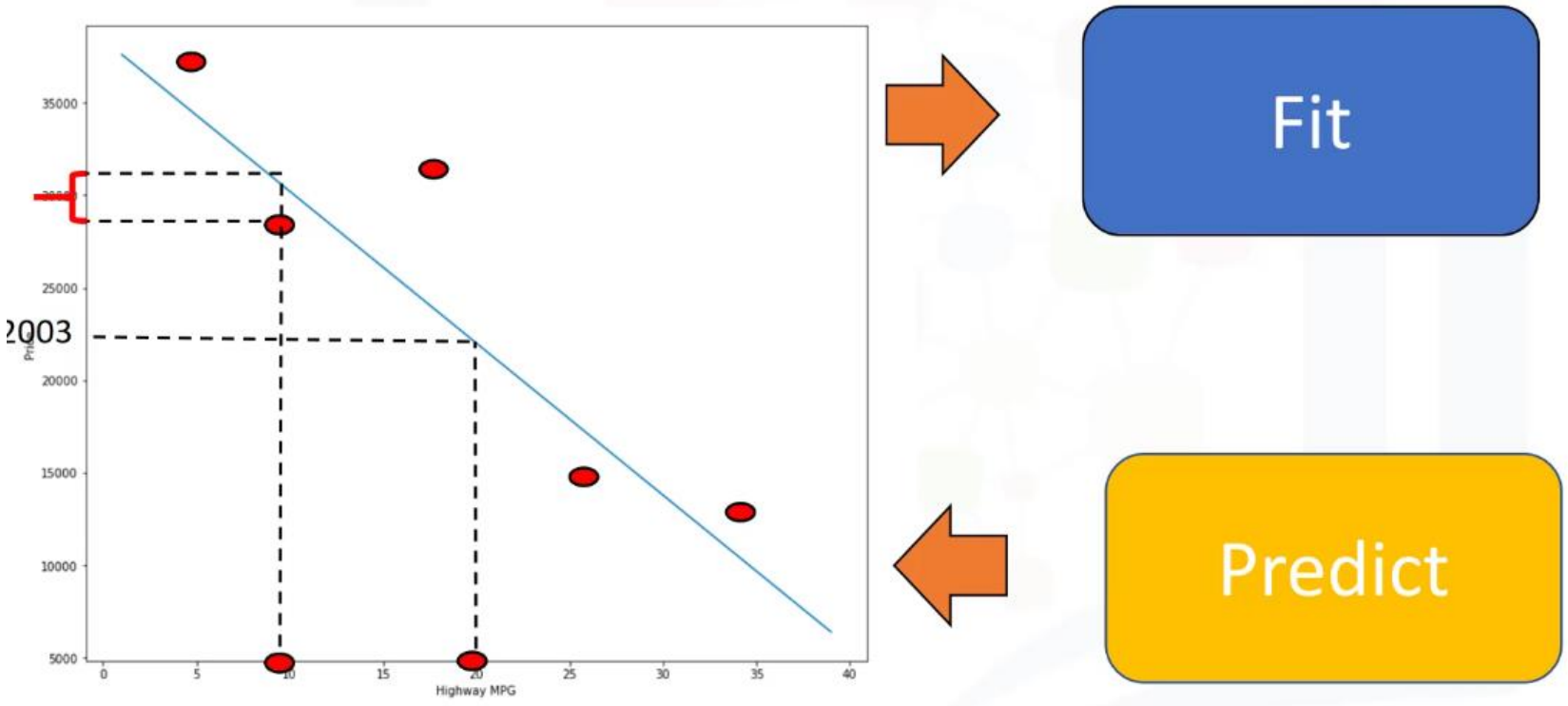
- We now have a model; we use the hat on the y to denote the model is an estimate
- We can use this model to predict values that we haven't seen.

For example, we have no car with 20 Highway Miles per Gallon, we can use our model to make a prediction for the price of this car. But don't forget our model is not always correct. We can see this by comparing the predicted value to the actual value.

Chapter 4 - Model Development

2. Simple Linear Regression

❑ **Fit**



We have a sample for 10 Highway Miles per Gallon, but the predicted value does not match the actual value. If the linear assumption is correct this error is due to the noise but there can be other reasons.

Chapter 4 - Model Development

3. Fitting a simple Linear Model Estimate

- X :Predictor variable
- Y: Target variable

1. Import linear_model from scikit-learn

```
from sklearn.linear_model import LinearRegression
```

2. Create a Linear Regression Object using the constructor :

```
lm=LinearRegression()
```

- We define the predictor variable and target variable

```
X = df[['highway-mpg']]
Y = df['price']
```

- Then use lm.fit (X, Y) to fit the model , i.e fine the parameters b_0 and b_1

```
lm.fit(X, Y)
```

- We can obtain a prediction

```
Yhat=lm.predict(X)
```

To fit the model in Python, first we import linear model from scikit-learn; then Create a Linear Regression Object using the constructor. We define the predictor variable and target variable. Then use the method fit to fit the model and find the parameters b0 and b1.

We can obtain a prediction using the method predict. The output is an array, the array has the same number of samples as the input X.

Chapter 4 - Model Development

4. Multiple Linear Regression (MLR)

This method is used to explain the relationship between:

- One continuous target (Y) variable
- Two or more predictor (X) variables

$$\hat{Y} = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4$$

- b_0 : **intercept (X=0)**
- b_1: the **coefficient** or **parameter** of x_1
- b_2: the **coefficient** of **parameter** x_2 and so on..

$$\hat{Y} = 1 + 2x_1 + 3x_2$$

- The variables x_1 and x_2 can be visualized on a 2D plane, lets do an example on the next slide

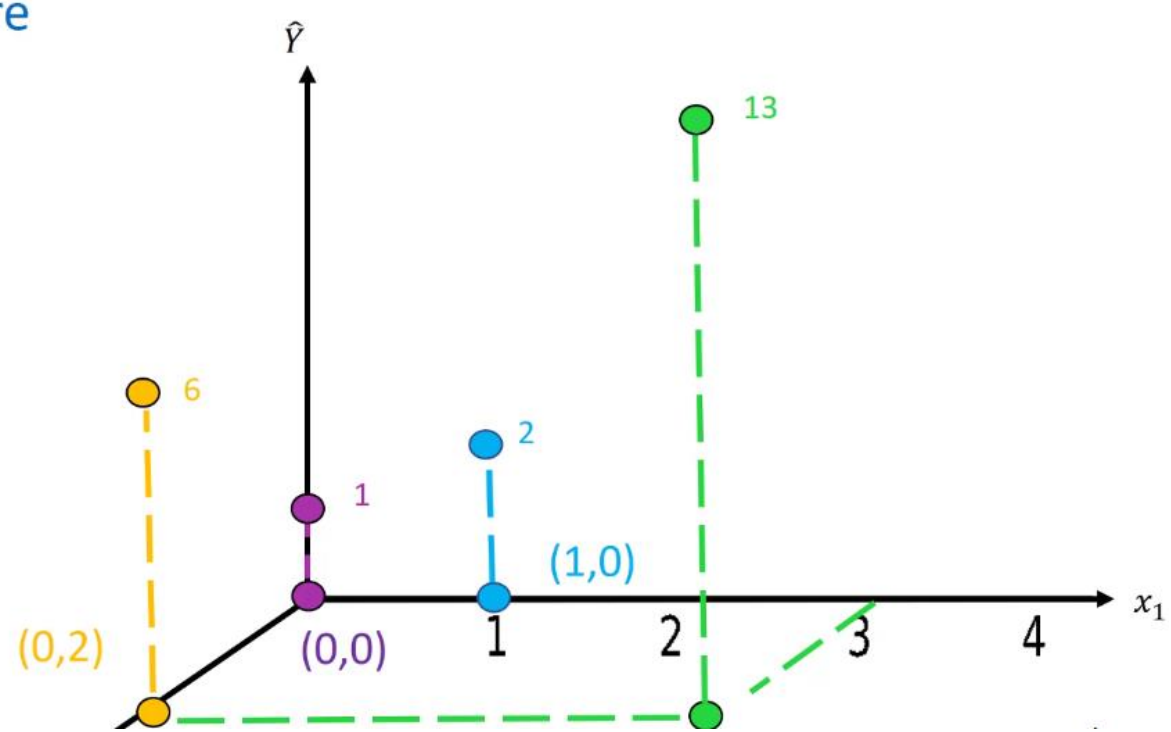
Chapter 4 - Model Development

4. Multiple Linear Regression (MLR)

- This is shown below where

$$\hat{Y} = 1 + 2x_1 + 3x_2$$

n	x_1	x_2	$\hat{Y}$
1	0	0	1
2	0	2	6
3	1	0	2
4	3	2	13



Predictor variables are X1 and X2. The position of each point is placed on the 2D plane, color coded accordingly. Each value of the predictor variables X1 and X2 will be mapped to a new value Y (y hat) the new values of Y (y hat) are mapped in the vertical direction, with height proportional to the value that yhat takes.

Chapter 4 - Model Development

5. Fitting a Multiple Linear Model Estimator

We can fit the Multiple linear regression as follows:

- We can extract the 4 predictor variables and store them in the variable Z.
- Then train the model as before using the method train, with the features or depended variables and the targets : (colon) We can also obtain a prediction using the method predict. In this case, the input is an array or dataframe with 4 columns, the number of rows correspond to the number of samples.

The output is an array with the same number of elements as number of samples.

1. We can extract the for 4 predictor variables and store them in the variable Z

```
Z = df[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']]
```

2. Then train the model as before:

```
lm.fit(Z, df['price'])
```

3. We can also obtain a prediction

```
Yhat=lm.predict(X)
```

x_1	x_2	x_3	x_4
3	5	-4	3
:	:	:	:
2	4	2	-4

Chapter 4 - Model Development

6. MLR – Estimated Linear Model

1. Find the intercept (b_0)

2. Find the coefficients (b_1, b_2, b_3, b_4)

The Estimated Linear Model:

The intercept is an attribute of the object. And the coefficients are also attributes.
It is helpful to visualize the equation, replacing the dependent variable names with actual names.

Chapter 4 - Model Development

6. MLR – Estimated Linear Model

1. Find the intercept (b_0)

```
lm.intercept_
-15678.742628061467
```

2. Find the coefficients (b_1, b_2, b_3, b_4)

```
lm.coef_
array([52.65851272 ,4.69878948,81.95906216 , 33.58258185])
```

The Estimated Linear Model:

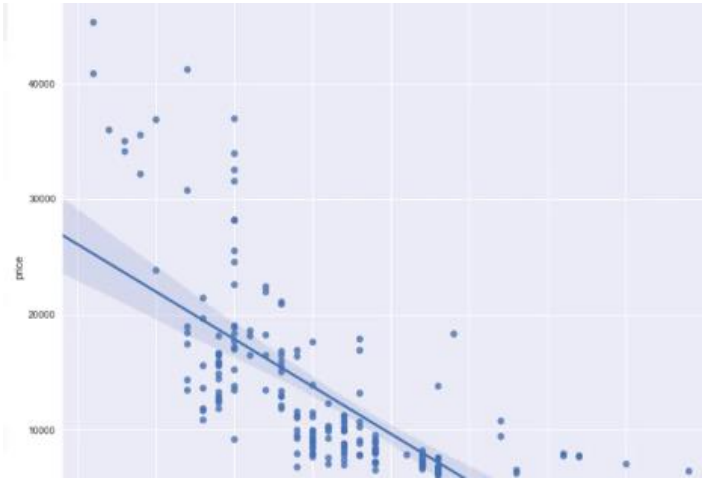
- **Price** = -15678.74 + (52.66) * **horsepower** + (4.70) * **curb-weight** + (81.96) * **engine-size** + (33.58) * **highway-mpg**

$$\hat{Y} = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + b_4 x_4$$

Chapter 4 - Model Development

7. Model Evaluation using Visualization

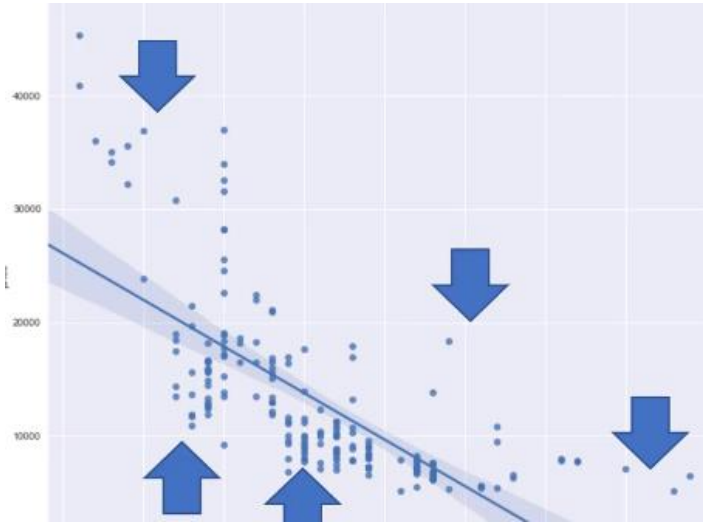
❑ Regression Plot



Why use regression plot?

It gives us a good estimate of:

1. The relationship between two kvariables
2. The strength of the correlation
3. The direction of the relationship (positive or negative)



Regression plot shows us a combination of:

- The scatterplot: where each point represents a different y
- The fitted linear regression line (y).

Chapter 4 - Model Development

7. Model Evaluation using Visualization

❑ Regression Plot

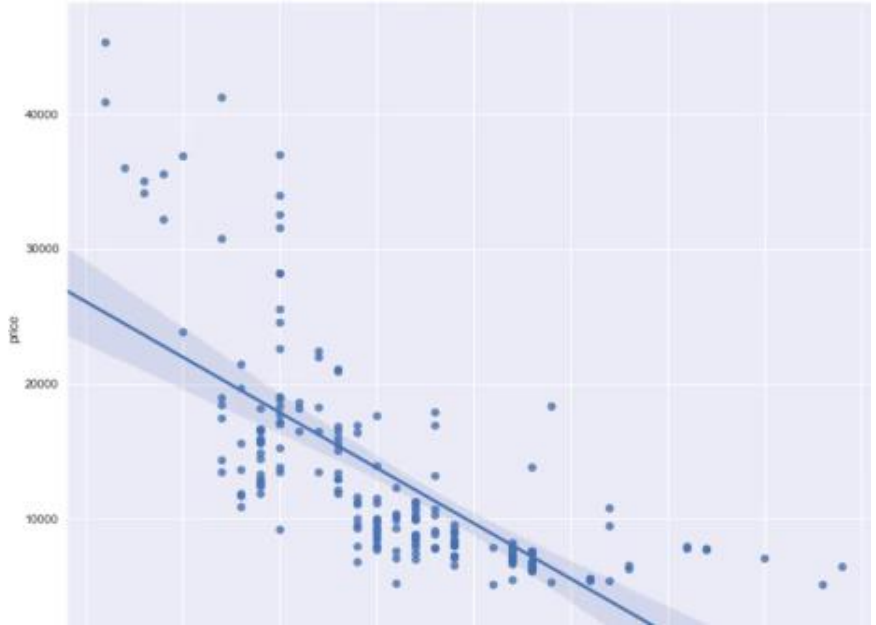
```
Import seaborn as sns
```

```
Sns.regplot(x="highway-mpg", y="pride", data=df)
Pl0.ylim(0,)
```

The parameter x is the name of the column that contains the dependent variable of feature.

The parameter y contains the name of the column that contains the name of the dependent variable.

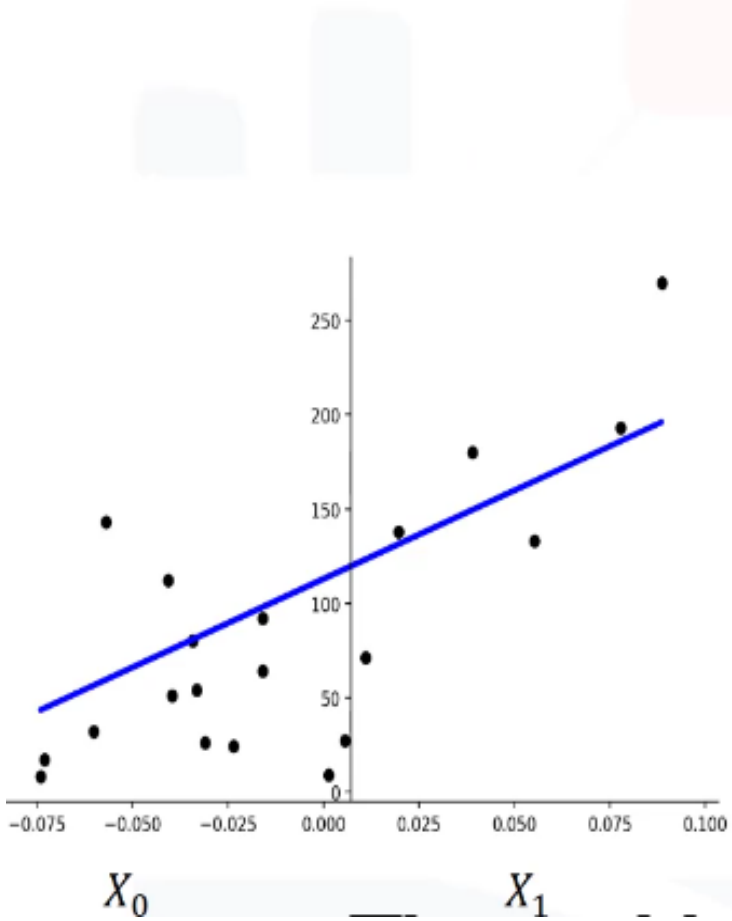
The parameter data is the name of the dataframe.

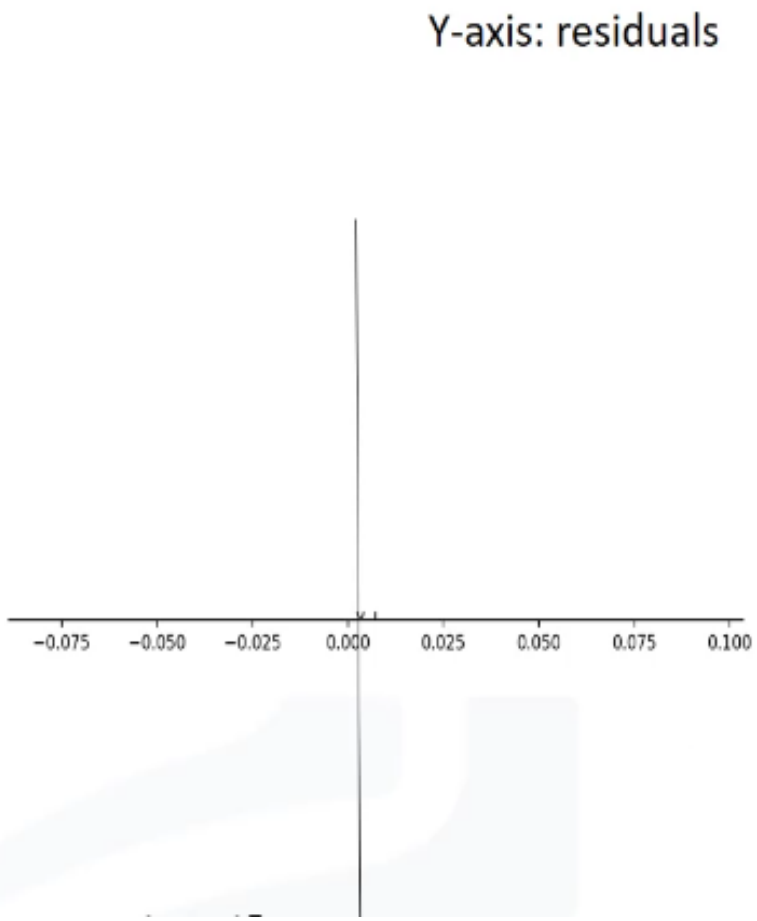


Chapter 4 - Model Development

7. Model Evaluation using Visualization

❑ Residual plot



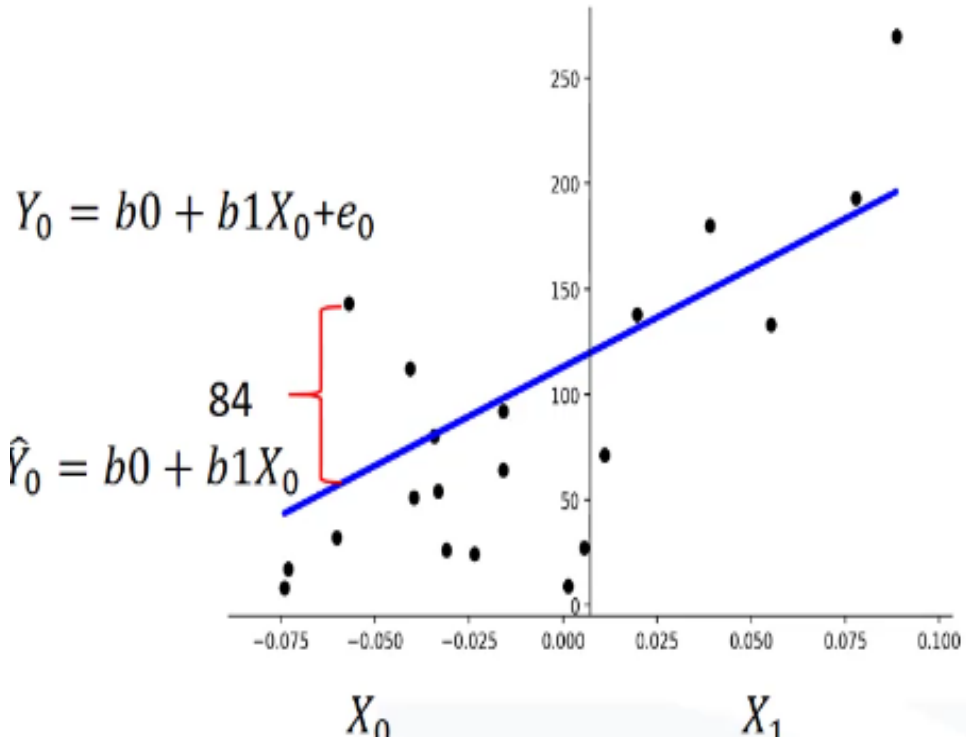


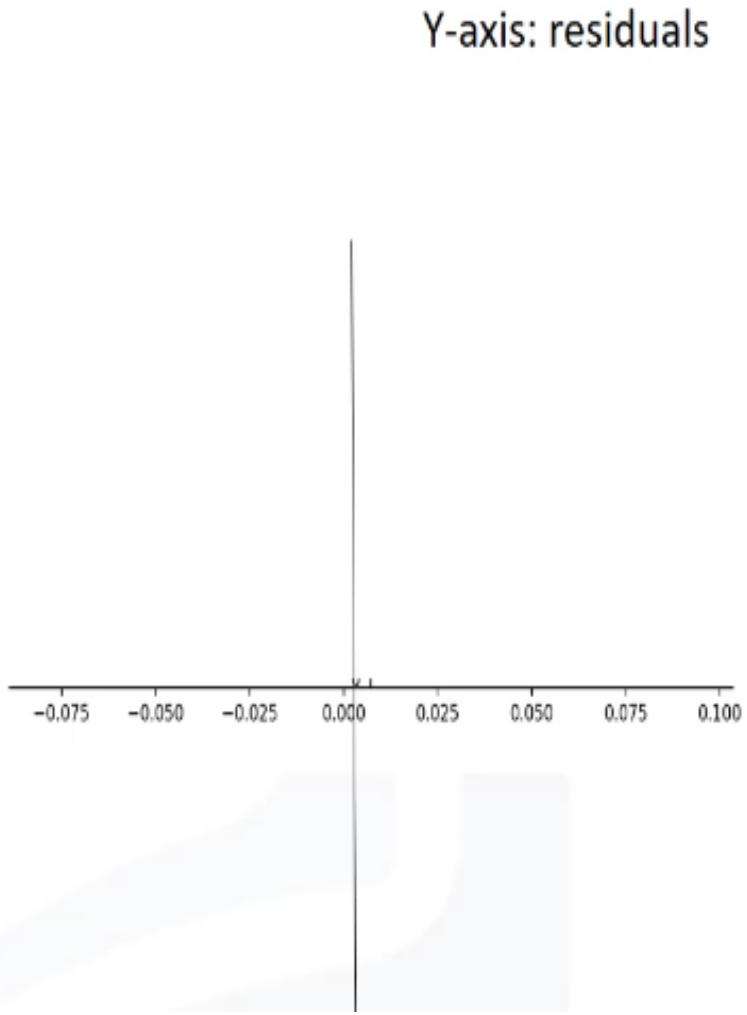
The residual plot represents the error between the actual values.

Chapter 4 - Model Development

7. Model Evaluation using Visualization

❑ Residual plot



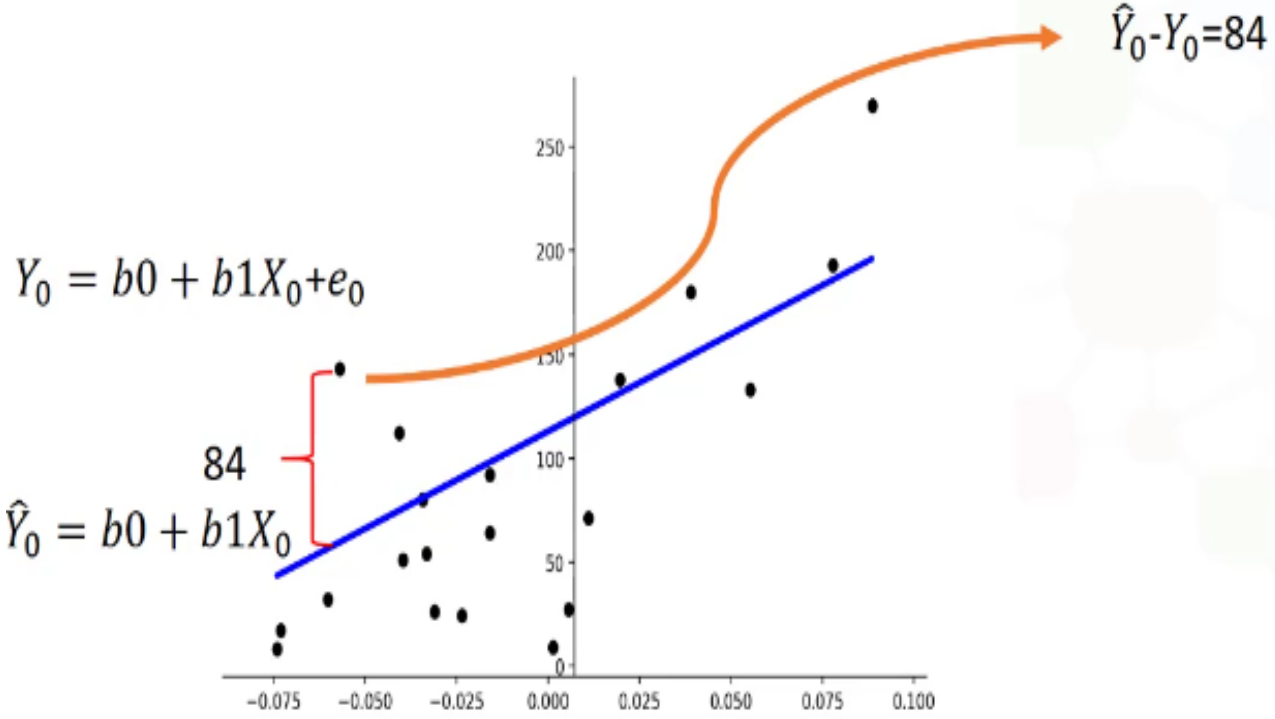


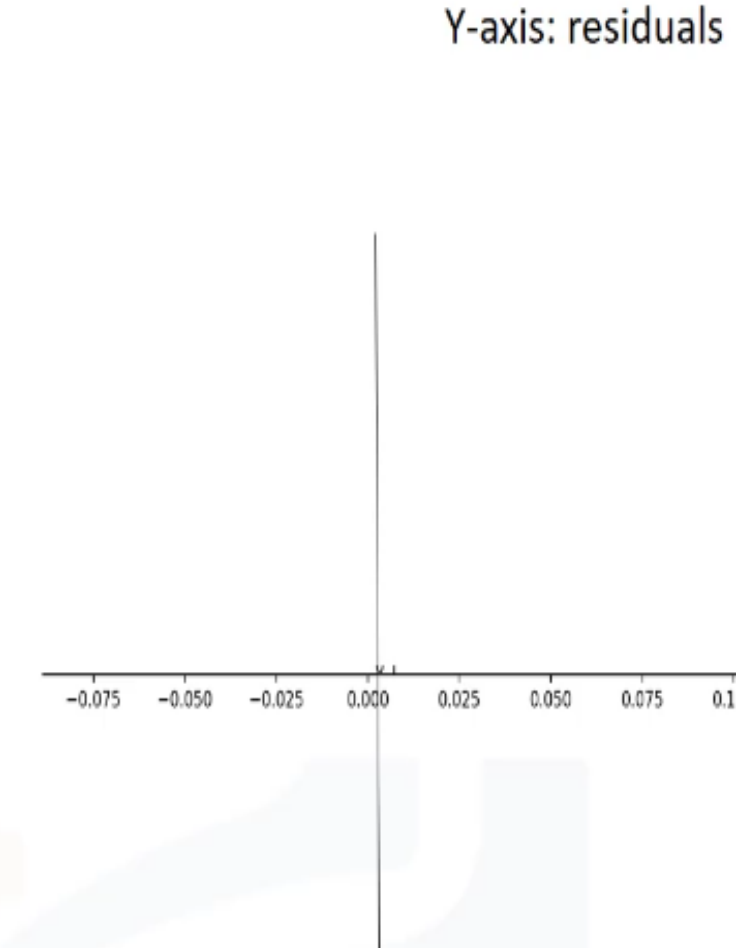
Examining the predict value and actual value we see a difference.

Chapter 4 - Model Development

7. Model Evaluation using Visualization

❑ Residual plot



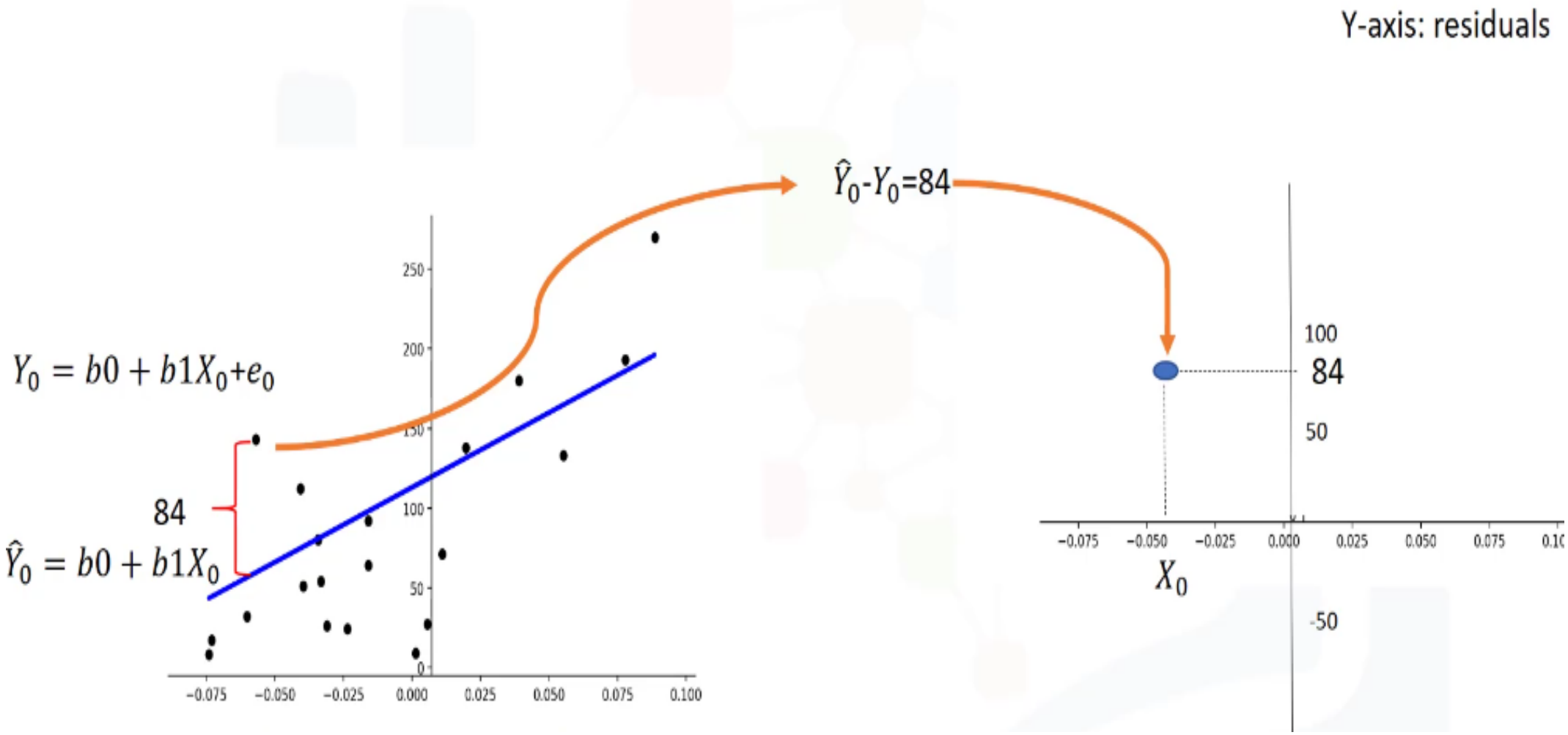


We obtain that value by subtracting the predicted value and the actual target value.

Chapter 4 - Model Development

7. Model Evaluation using Visualization

❑ Residual plot

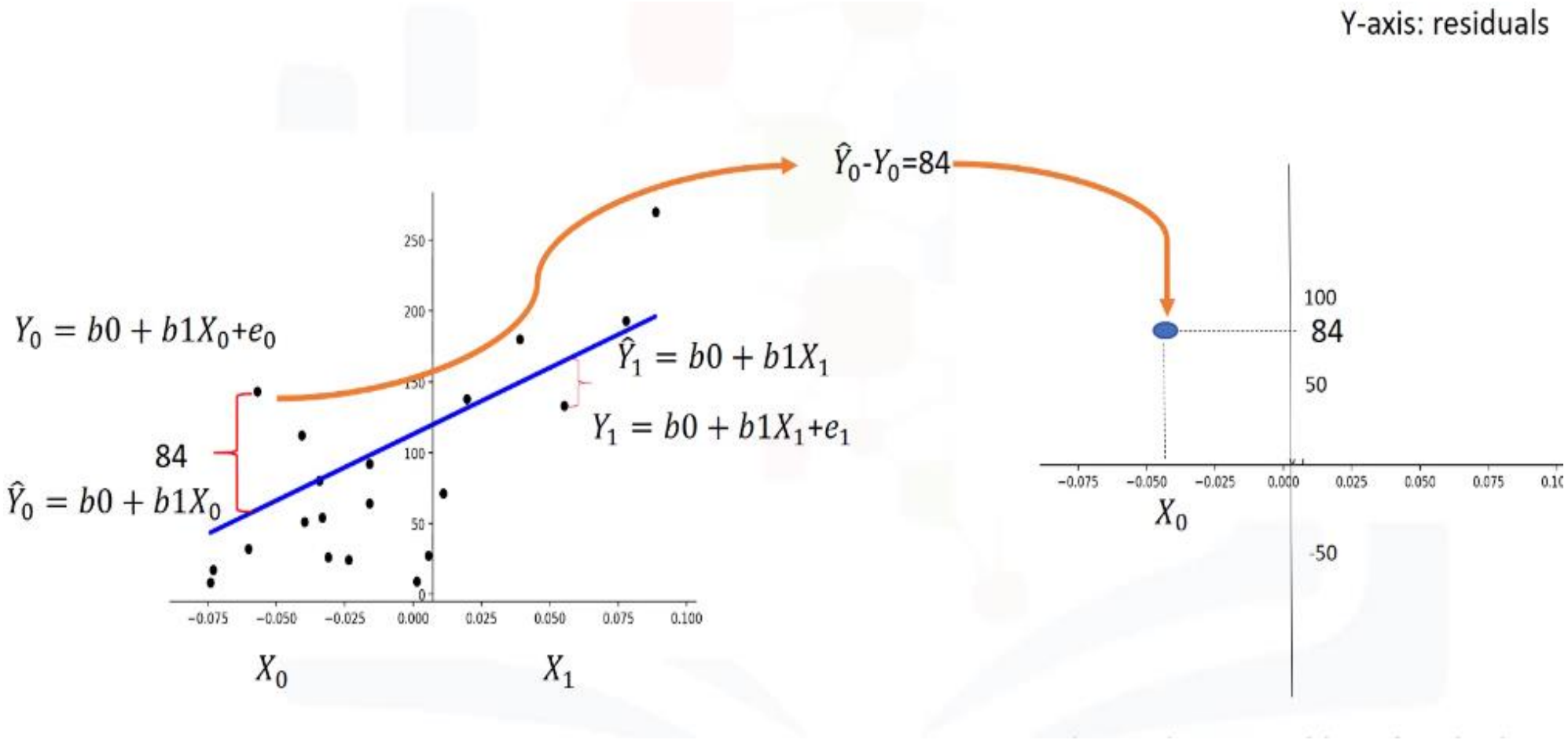


We then plot that value on the vertical axis, with the dependent variable as the horizontal.

Chapter 4 - Model Development

7. Model Evaluation using Visualization

❑ Residual plot

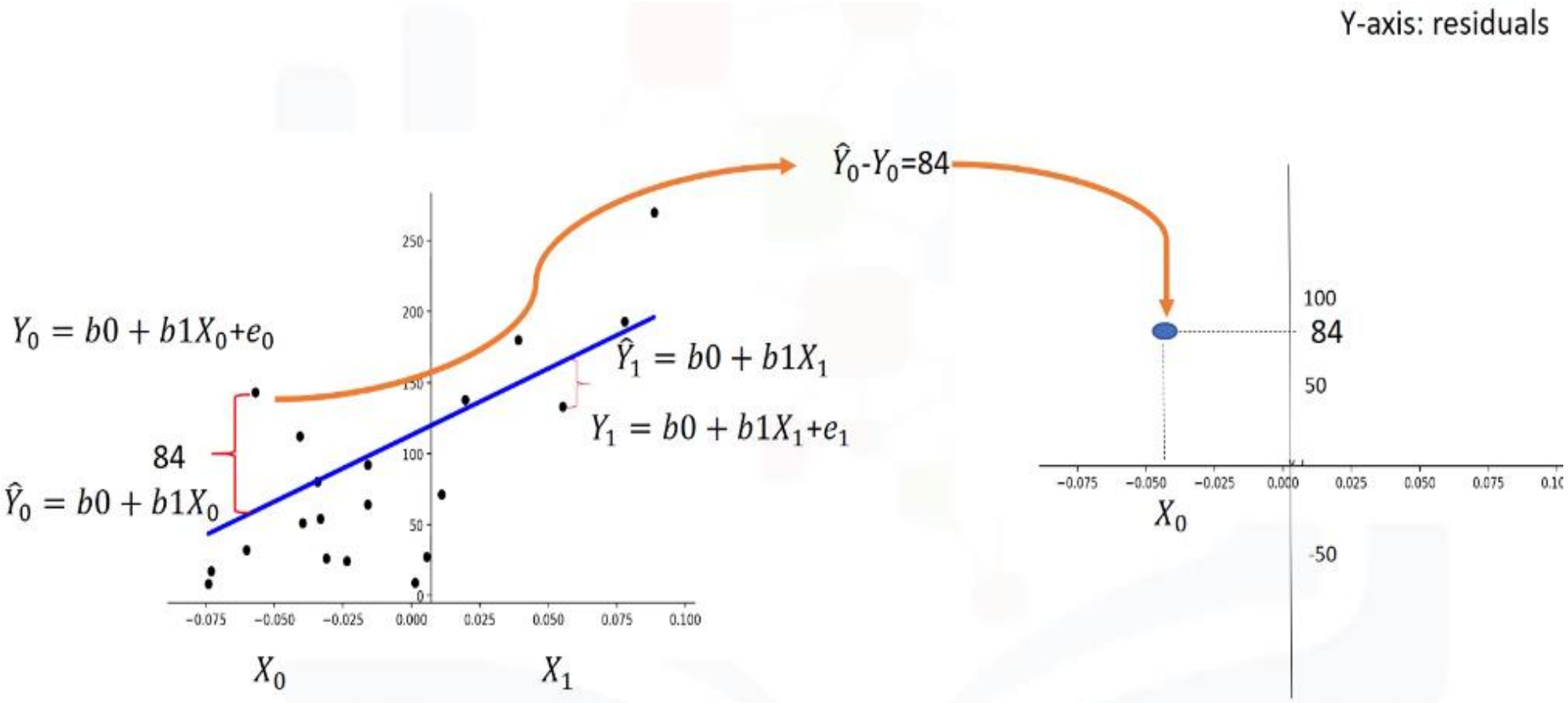


Similarly, for the second sample, we repeat the process.

Chapter 4 - Model Development

7. Model Evaluation using Visualization

❑ Residual plot



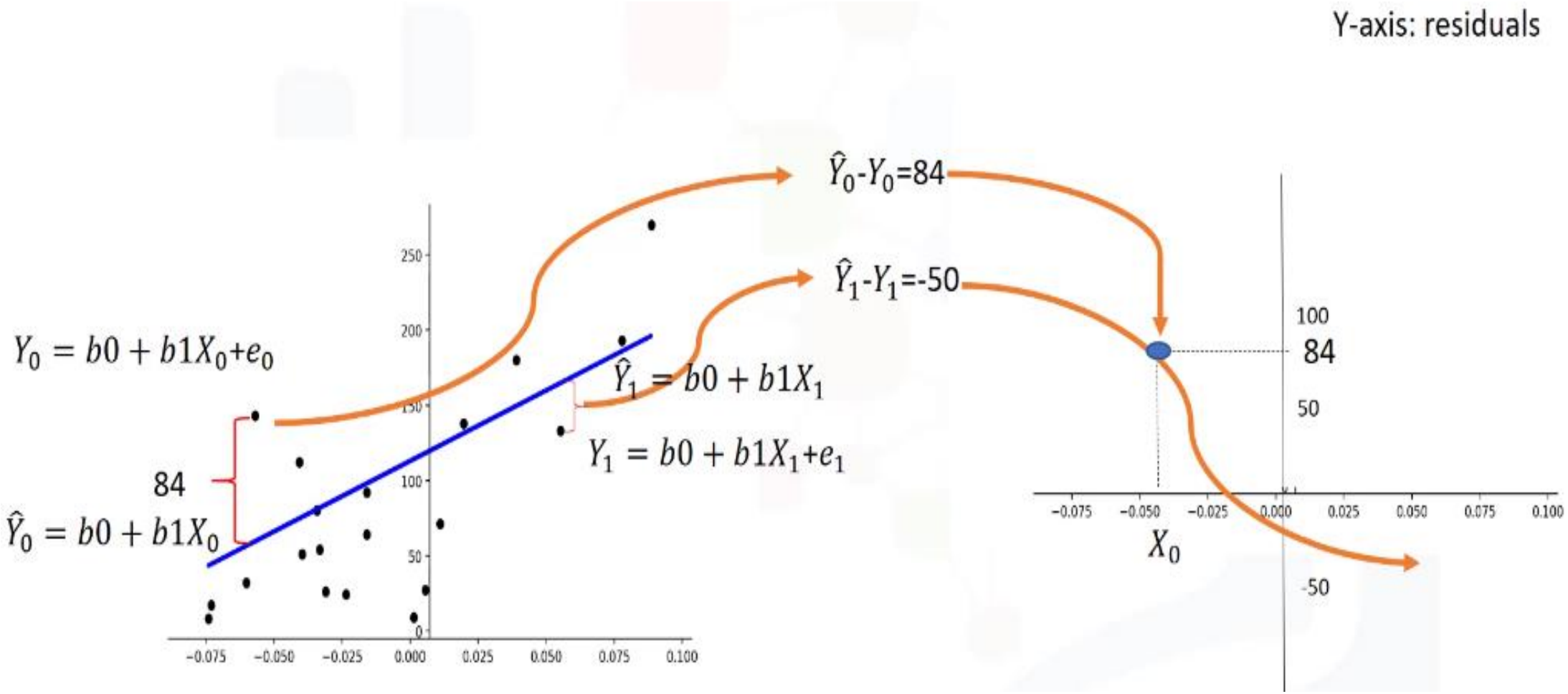
Similarly, for the second sample, we repeat the process.

Subtracting the target value from the predicted value; then plotting the value accordingly.

Chapter 4 - Model Development

7. Model Evaluation using Visualization

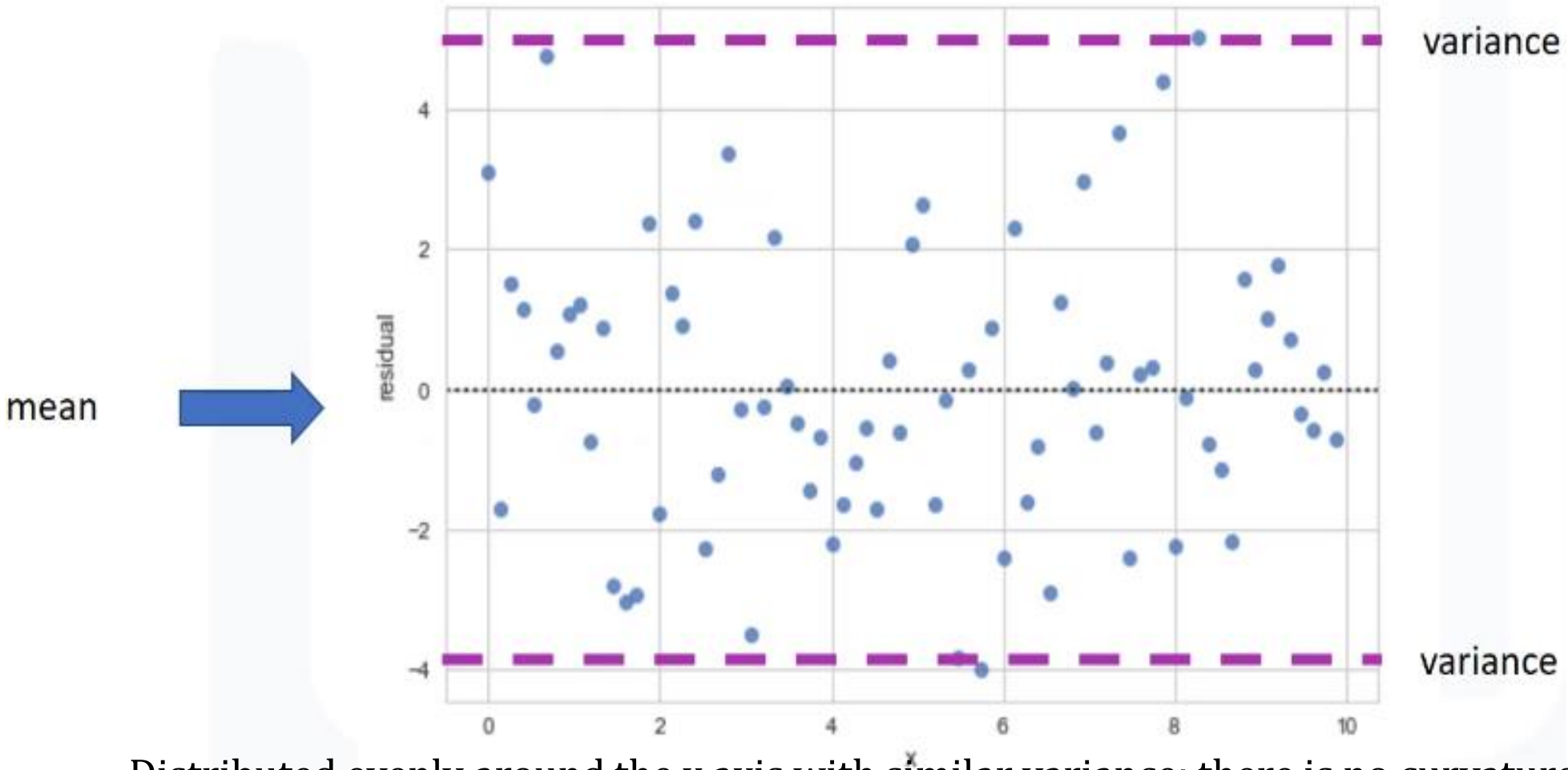
❑ Residual plot



Chapter 4 - Model Development

7. Model Evaluation using Visualization

❑ Residual plot

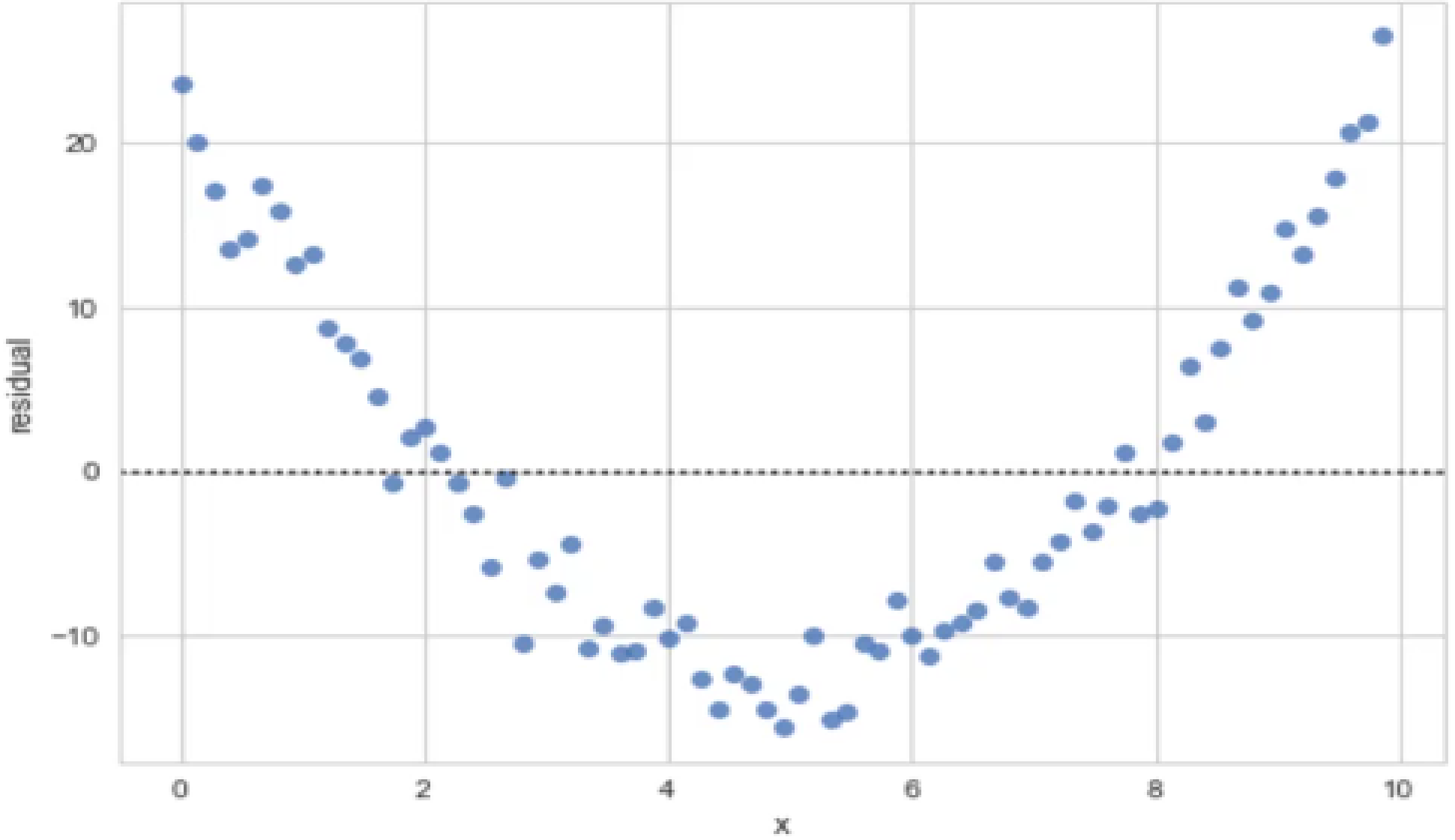


Distributed evenly around the x axis with similar variance; there is no curvature.

This type of residual plot suggests a linear plot is appropriate.

Chapter 4 - Model Development

7. Model Evaluation using Visualization

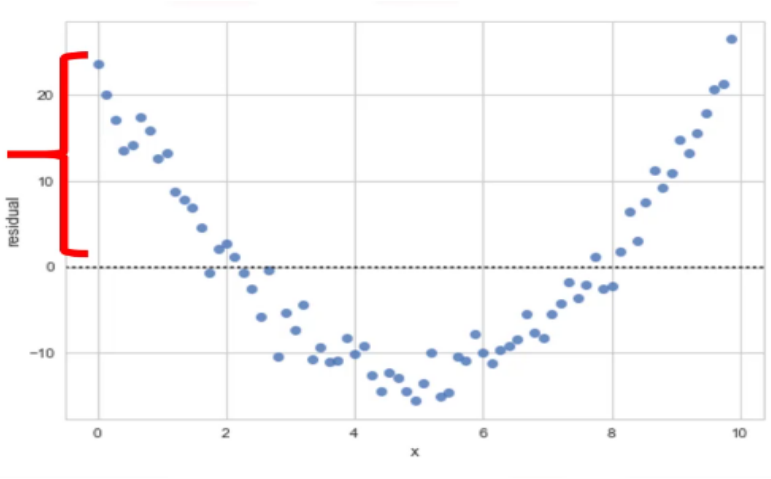


This type of residual plot suggests a linear plot is appropriate. In this residual plot there is curvature, the values of the error change with x.

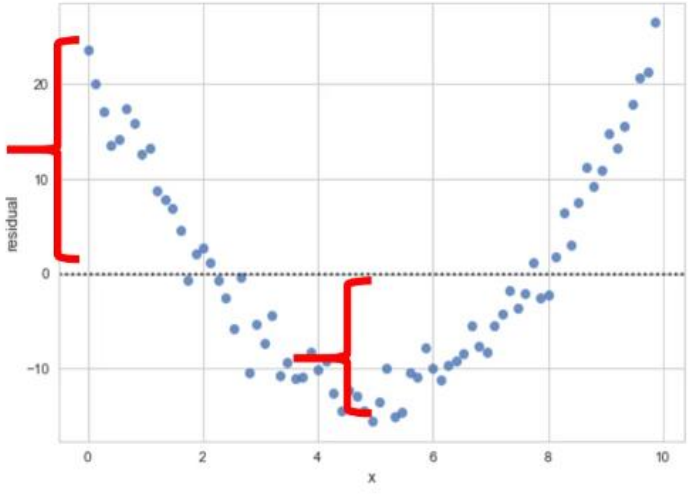
Chapter 4 - Model Development

7. Model Evaluation using Visualization

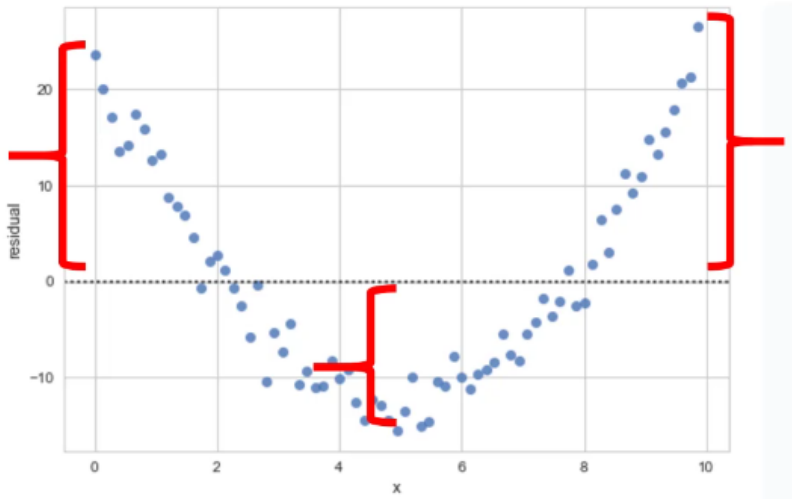
❑ Residual plot



For example, in the region, all the residual errors are positive.



In this area, the residuals are negative.

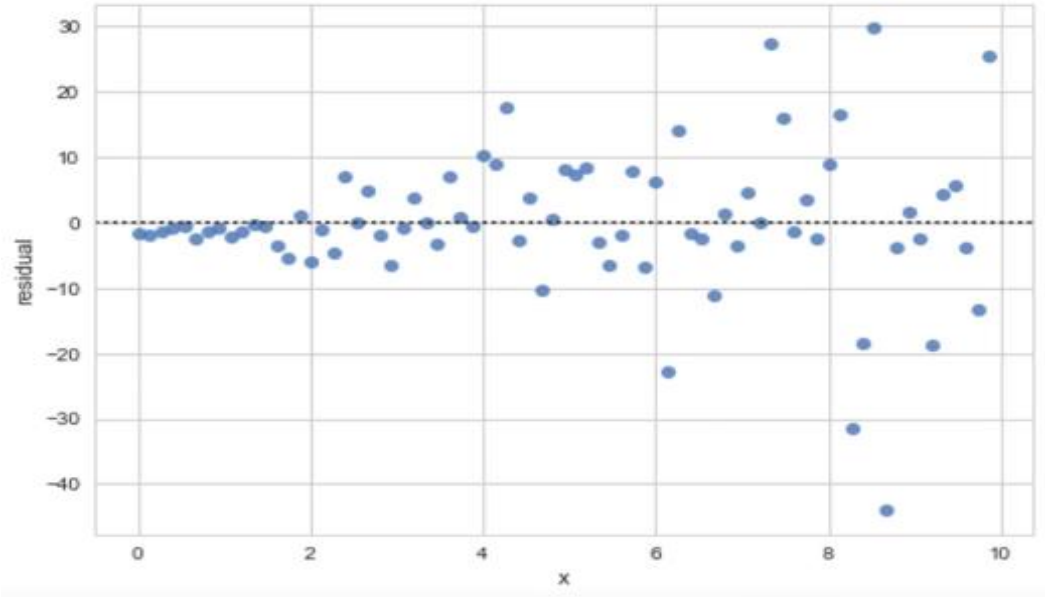


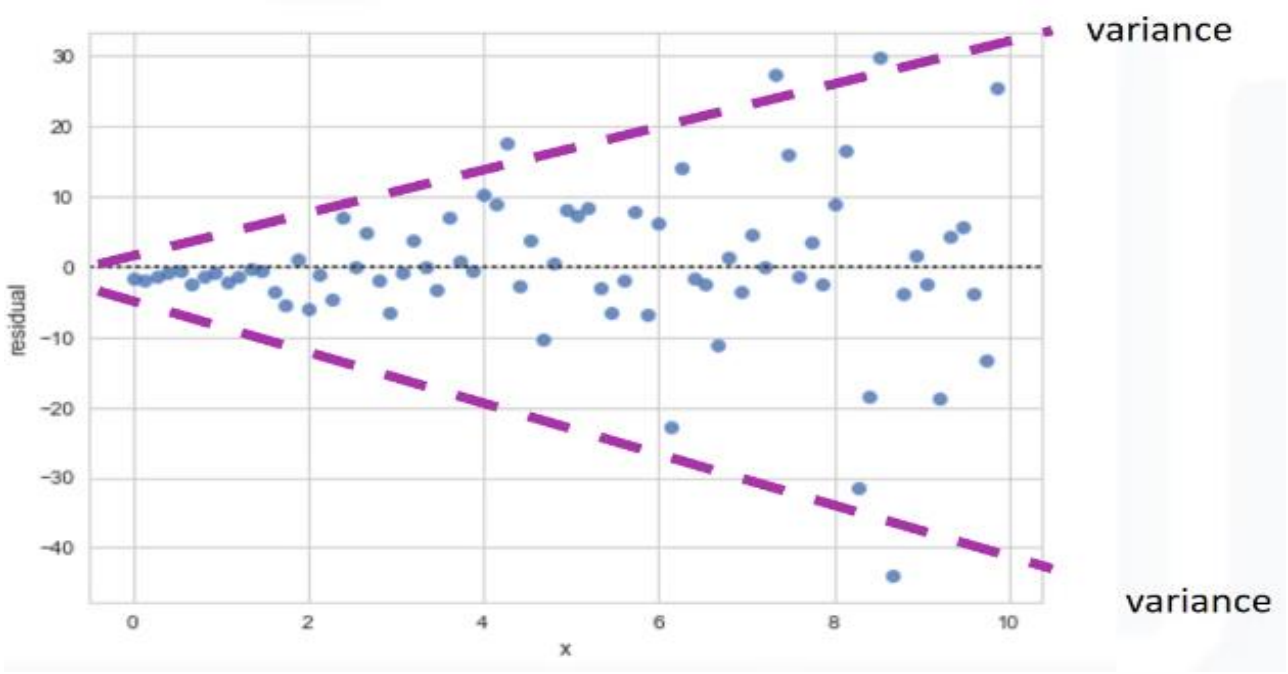
In the final location, the error is large. The residuals are not randomly separated; this suggests the linear assumption is incorrect. This plot suggests a non-linear function, we will deal with this in the next section.

Chapter 4 - Model Development

7. Model Evaluation using Visualization

❑ Residual plot





In this plot, we see that variance of the residuals increases with x, therefore our model is incorrect.

We can use seaborn to create a Residual plot.

Chapter 4 - Model Development

7. Model Evaluation using Visualization

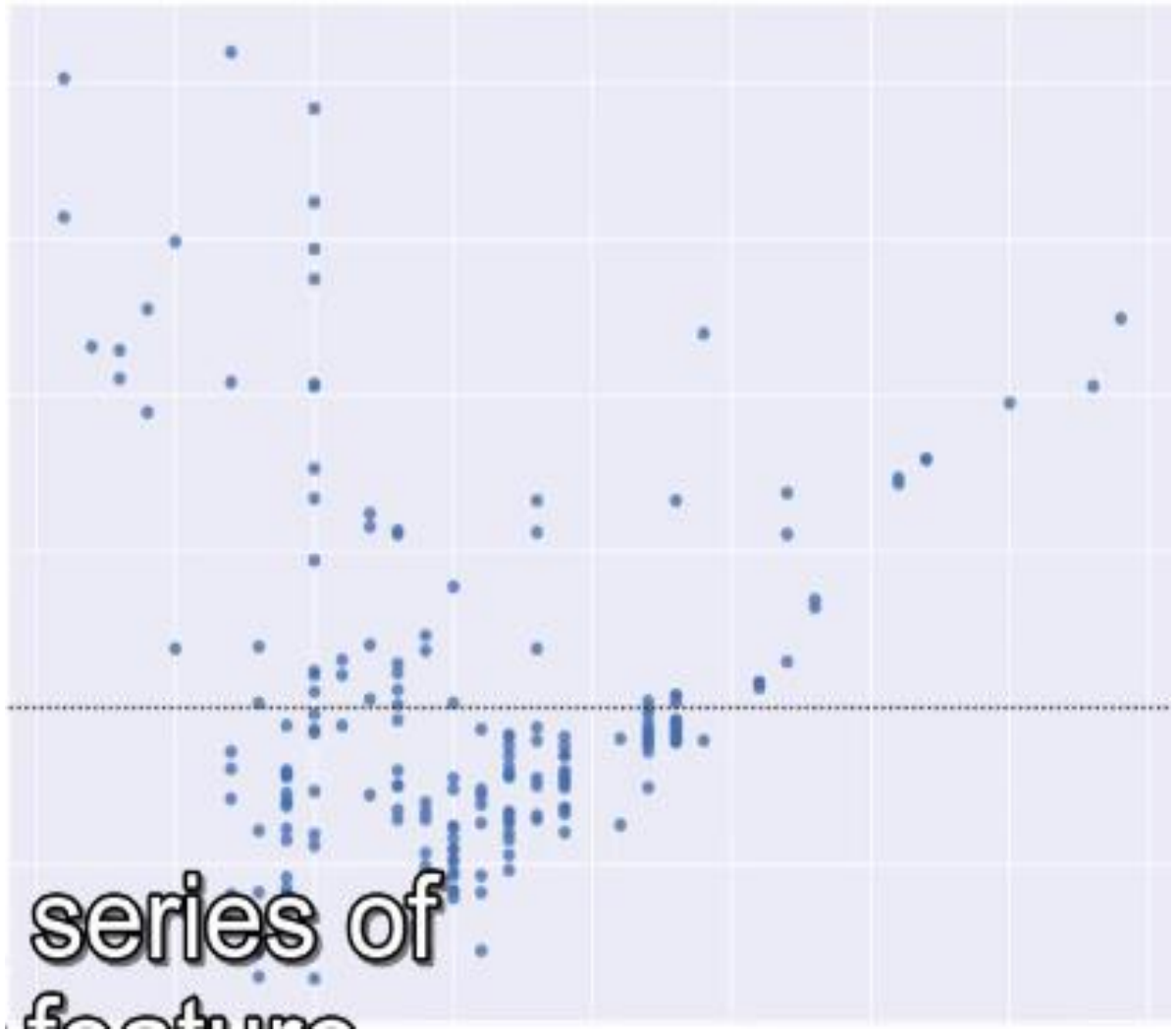
❑ Residual plot

```
Import seaborn as sns

Sns.residplot(df['highway-mpg'], df['price'])
```

The first parameter is a series of dependent variable or feature.

The second parameter is a series of dependent variable or target.

We see in this case the Residuals have a curvature.

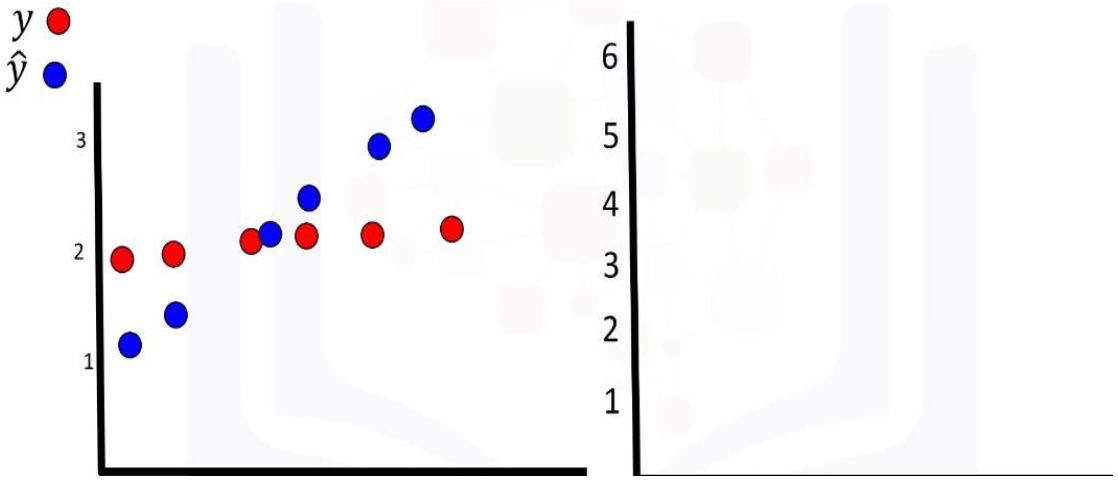


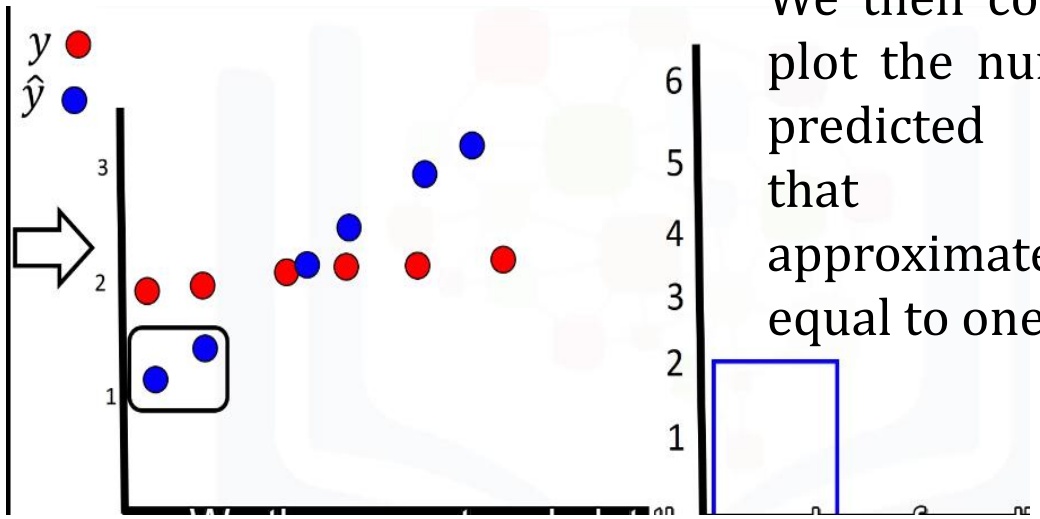
Chapter 4 - Model Development

7. Model Evaluation using Visualization

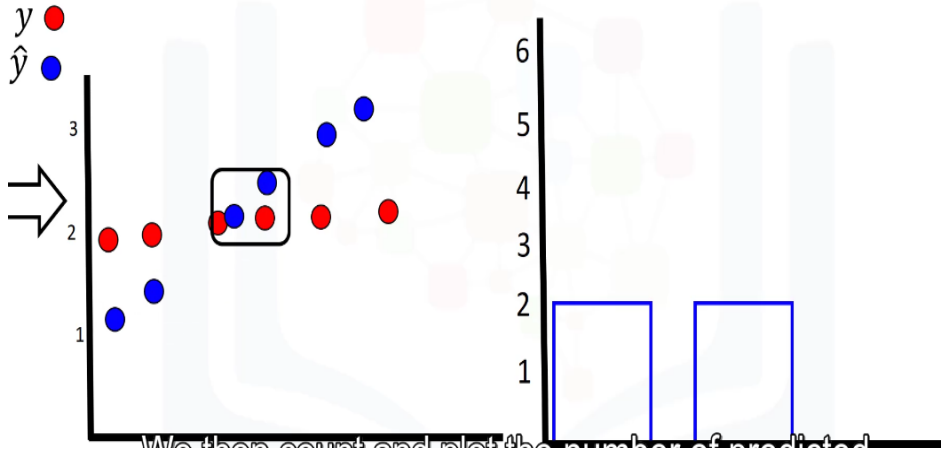
❑ Distribution plots

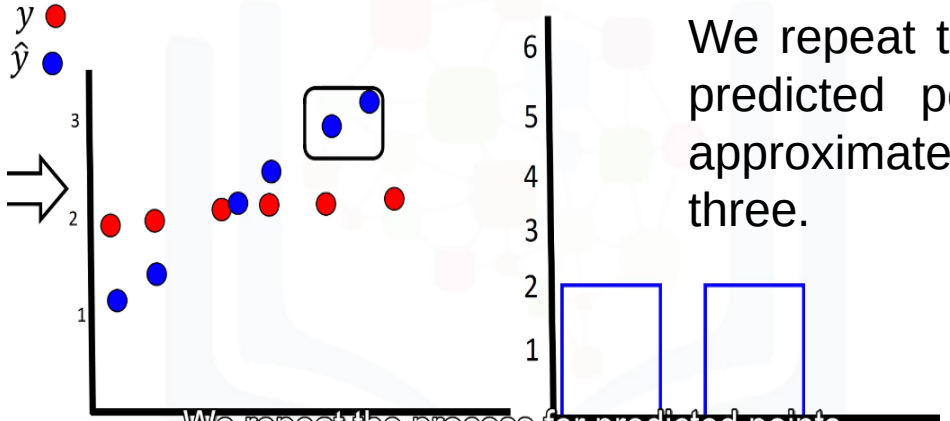
A distribution plot counts the predicted value versus the actual value. These plots are extremely useful for visualizing models with more than one independent variable or feature.





We then count and plot the number of predicted points that are approximately equal to one.



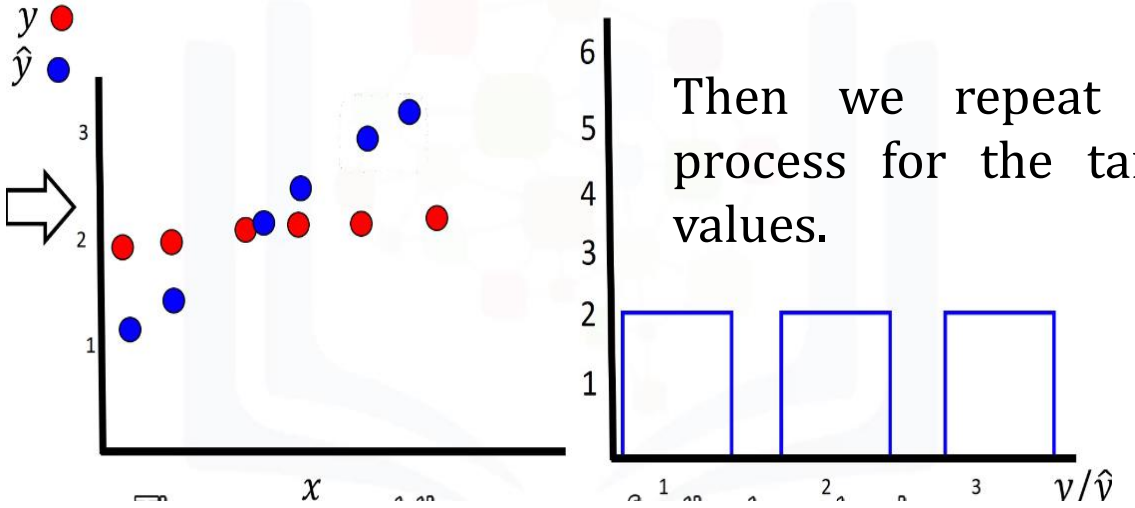


We repeat the process for predicted points that are approximately equal to three.

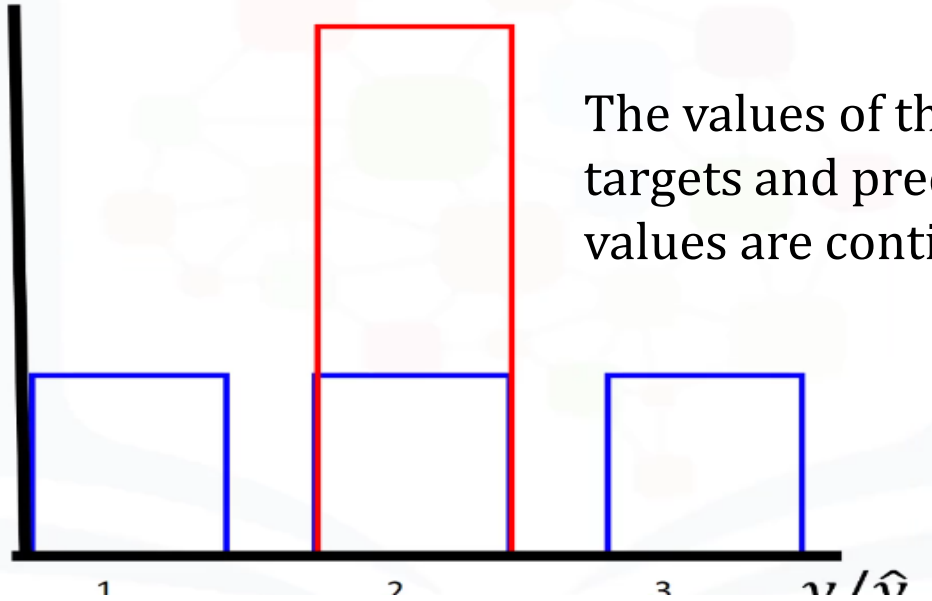
Chapter 4 - Model Development

7. Model Evaluation using Visualization

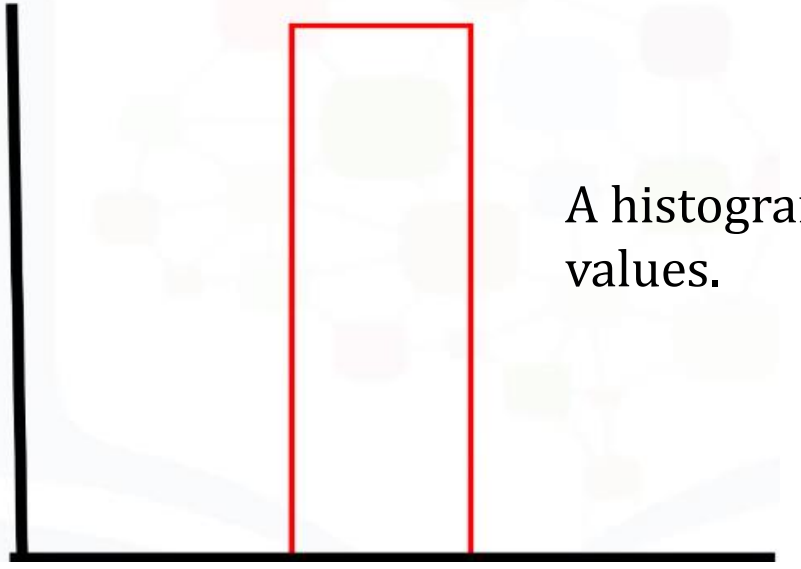
❑ Distribution plots



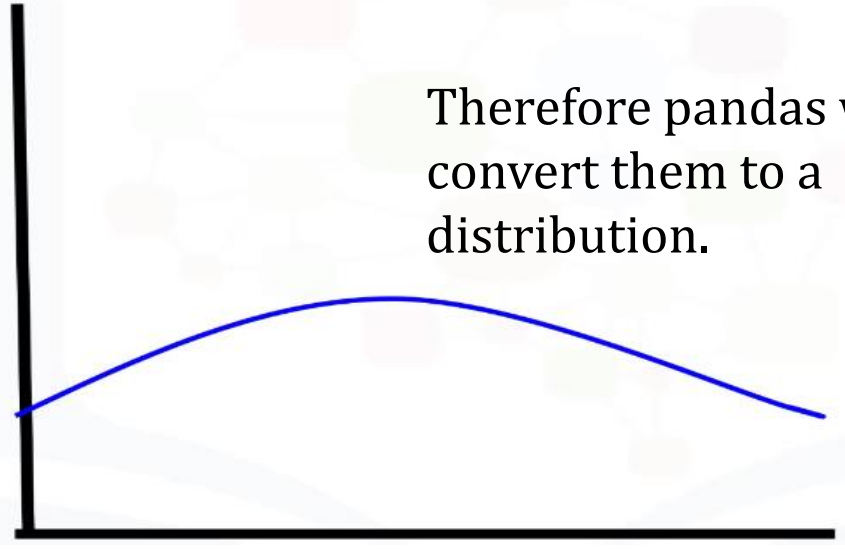
Then we repeat the process for the target values.



The values of the targets and predicted values are continuous.



A histogram is for discrete values.

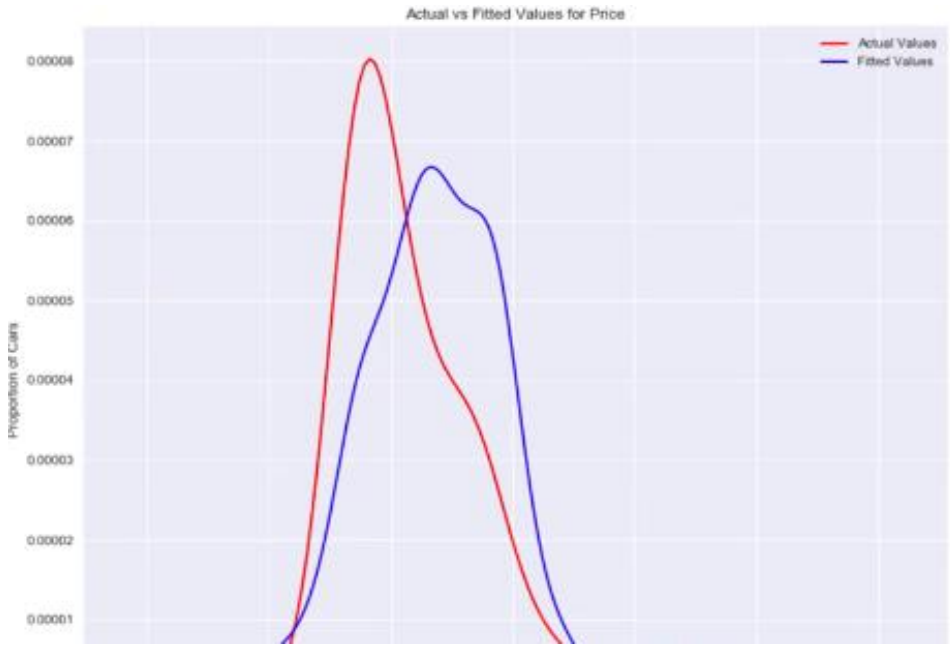


Therefore pandas will convert them to a distribution.

Chapter 4 - Model Development

7. Model Evaluation using Visualization

❑ Distribution plots



Compare the distribution plots:

- The fitted values that result from the model
- The actual values

```
Import seaborn as sns
```

```
Axl – sns.distplot (df [‘price’ ], hist-False, color-”r”, label-”Actuel value” )

Sns.distplot(yhat, hist-False, color-”b”, label- “Filtted values, ax-axl)
```

Chapter 4 - Model Development

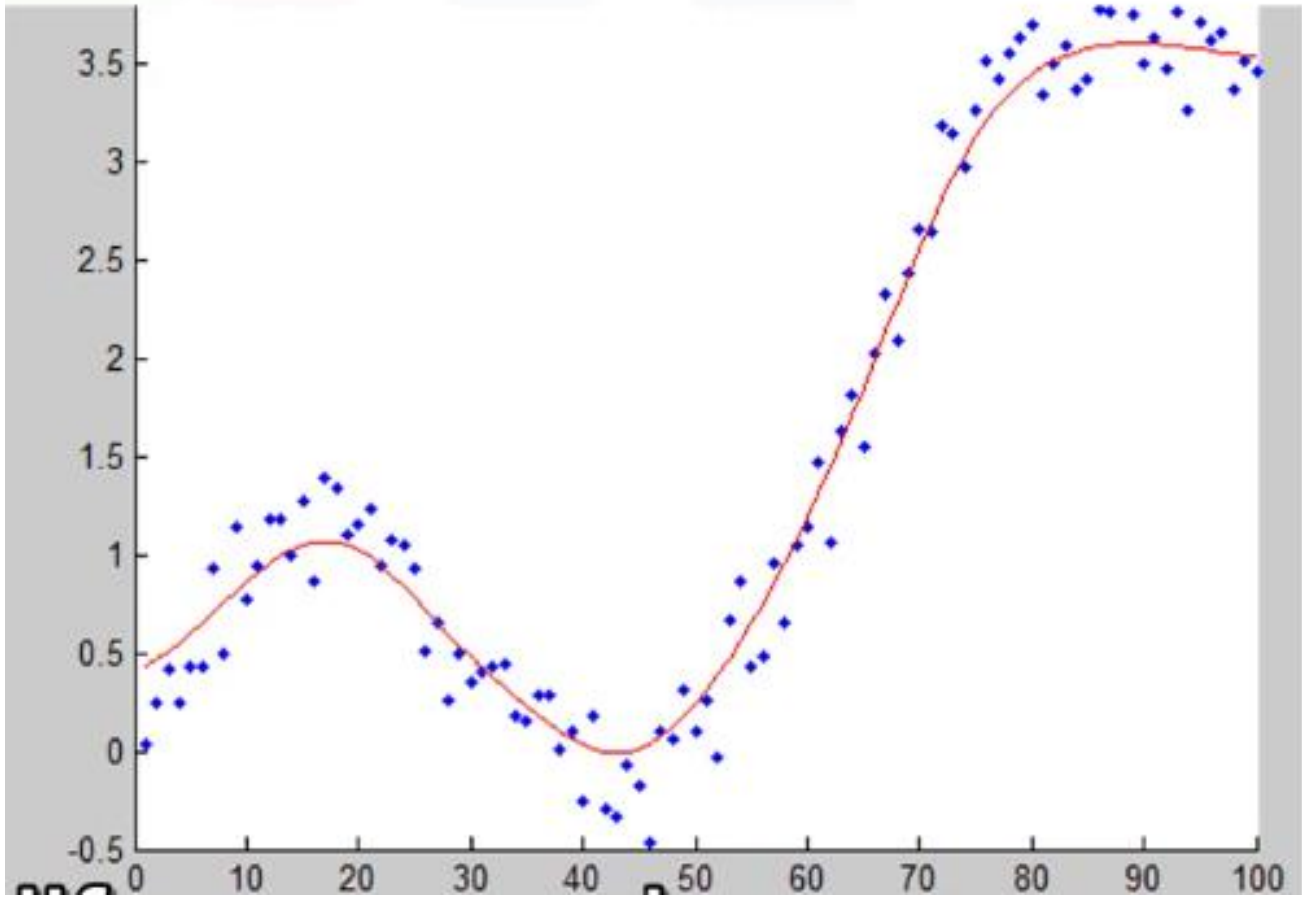
8. Polynomial Regression and Pipelines

- A special case of the general linear regression model

Polynomial Regression

- A special case of the general linear regression model
- Useful for describing curvilinear relationships

Curvilinear relationships:
By squaring or setting higher-order
Terms of the preditor variables



Polynomial Regression

- Quadratic – 2nd order

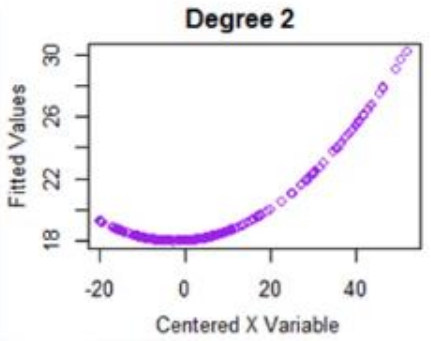
$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2$$

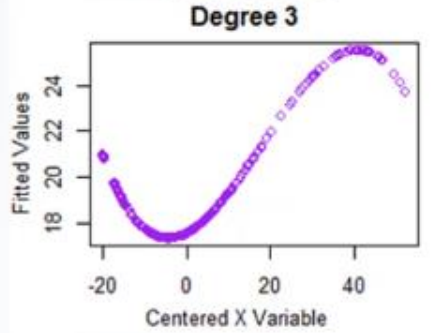
- Cubic – 3rd order

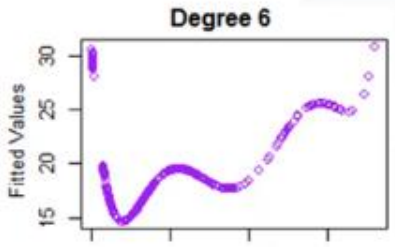
$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2 + b_3 (x_1)^3$$

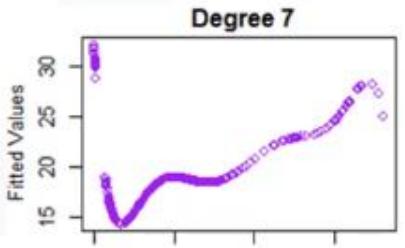
- Higher order

$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2 + b_3 (x_1)^3 + ..$$









Polynomial Regression

1. Calculate Polynomial of 3rd order

```
f=np.polyfit(x,y,3)

p=np.polydl(f)
```

2. We can print out the model

```
print (p)
```

$$-1.557(x_1)^3 + 204.8(x_1)^2 + 8965\,x_1 + 1.37\text{x}10^5$$

Polynomial Regression with More than One Dimension

- We can also have multi dimensional polynomial linear regression

$$\hat{Y} = b0 + b1\,X_1 + b2\,X_2 + b3\,X_1\,X_2 + b4(X_1)^2 + b5(X_2)^2 + .$$

- The "preprocessing" library in scikit-learn

```
from sklearn.preprocessing import PolynomialFeatures
pr=PolynomialFeatures(degree=2)
x_polly=pr.fit_transform(x[['horsepower', 'curb-weight']], include_bias=False)
```

Polynomial Regression with More than One Dimension

```
pr=PolynomialFeatures(degree=2)
```

X_1	X_2
1	2

```
pr.fit_transform([1,2], include_bias=False)
```

X_1	X_2	X_1X_2	X_1^2	X_2^2
1	2	(1) 2	1	$(2)^2$

1	2	2	1	4

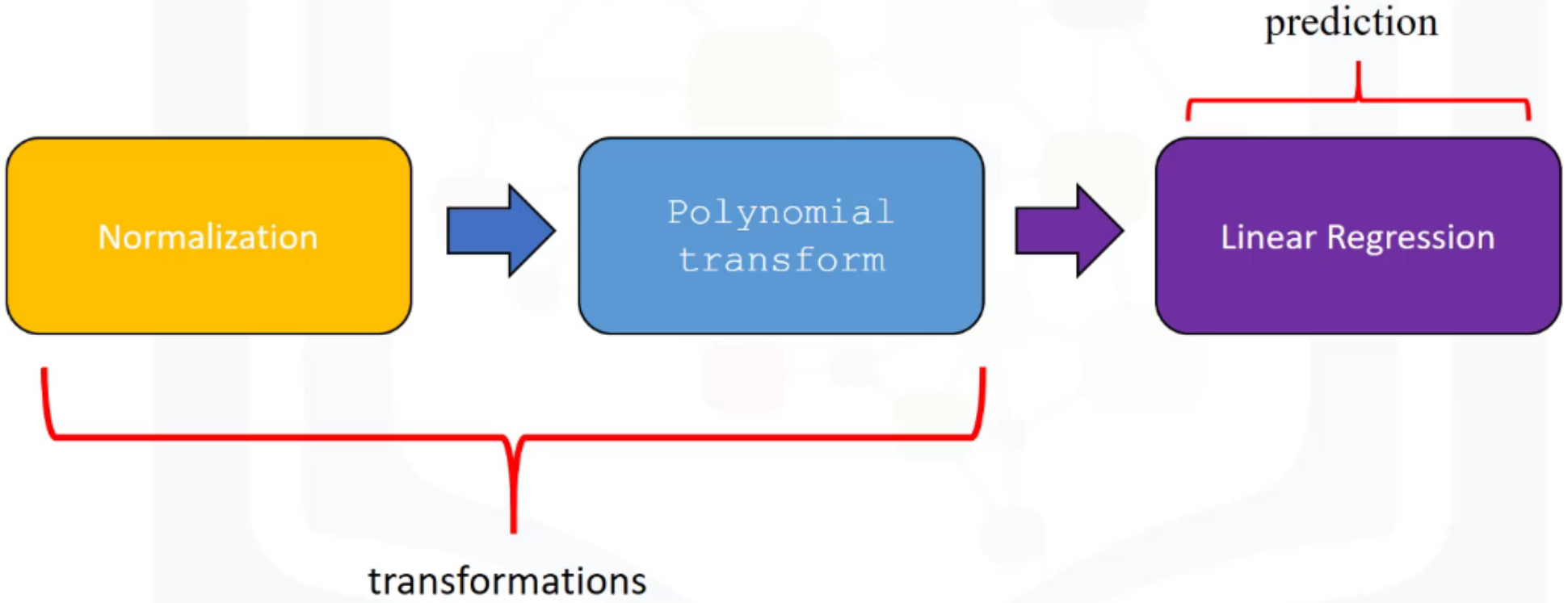
Pre-processing

- For example we can Normalize the each feature simultaneously

```
from sklearn.preprocessing import StandardScaler

SCALE=StandardScaler()

SCALE.fit(x_data[['horsepower', 'highway-mpg']])

x_scale=SCALE.transform(x_data[['horsepower', 'highway-mpg']])
```

Pipelines

- There are many steps to getting a prediction



Pipelines

```
from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline
```

Pipeline Constructor

```
Input=[('scale',StandardScaler()),('polynomial',PolynomialFeatures(degree=2),…
('mode',LinearRegression())]
```

Pipeline Constructor

- Pipeline constructor

```
pipe=Pipeline(Input)
```

pipeline object

Pipeline Constructor

- We can train the pipeline object

```
Pipe.train(X['horsepower', 'curb-weight', 'engine-size', 'highway-mpg'],y)

yhat=Pipe.predict(X[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']])
```

Prediction and Decision Making

Decision Making: Determining a Good Model Fit

To determine final best fit, we look at a cmbinatin of:

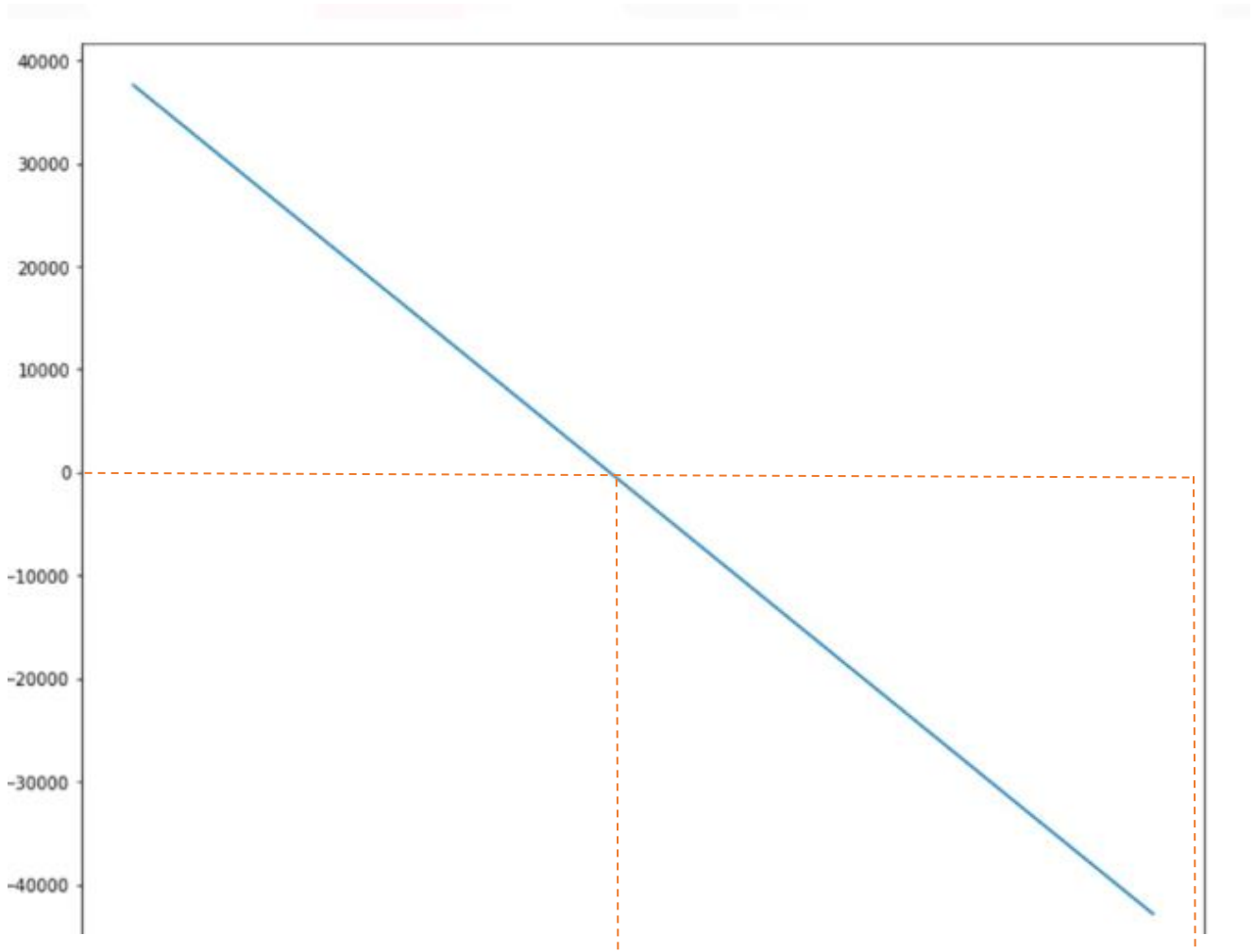
- Do the predicted values make sense
- Visualization
- Numerical measures for evaluation
- Comparing Moodels

Do the predicted values make sense

- First we train the model
  lm.fit(df[‘highway-mpg’],df[‘prices’]))
- Let’s predict the price of a car with 30 highway-mpg.
  lm.predict(30)
- Result: $ 13771.30

  lm.coef_
  -821.73337832
  - Price = 38423.31 – 821.73 * highway-mpg

Do the predicted values make sense



Do the predicted values make sense

- First we import numpy

```
import numpy as np
```

- We use the numpy function arrange to generate a sequence from 1 to 100

```
new_input=np.arange(1,101,1).reshape(-1,1)
```

Do the predicted values make sense

- First we import numpy

```
import numpy as np
```

- We use the numpy function arrange to generate a sequence from 1 to 100

```
new_input=np.arange(1,101,1).reshape(-1,1)
```

1	2	…	99	100

Do the predicted values make sense

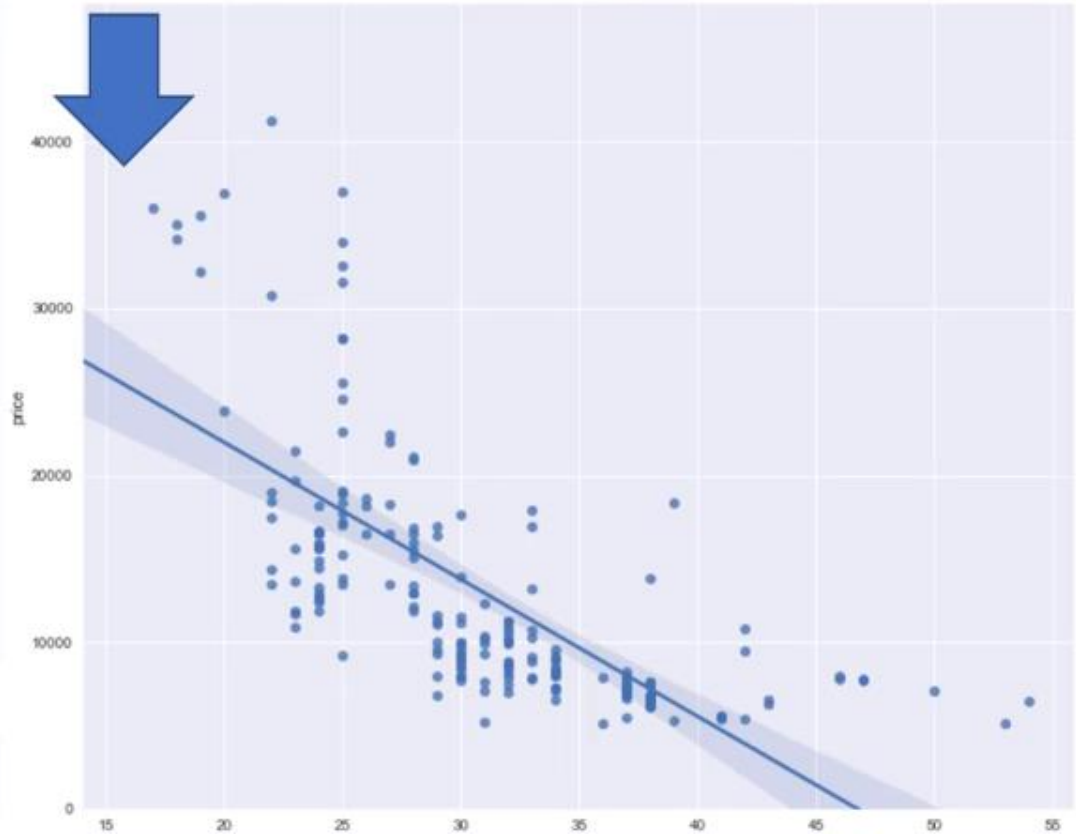
- We can predict new values

```
yhat=lm.predict(new_input)
```

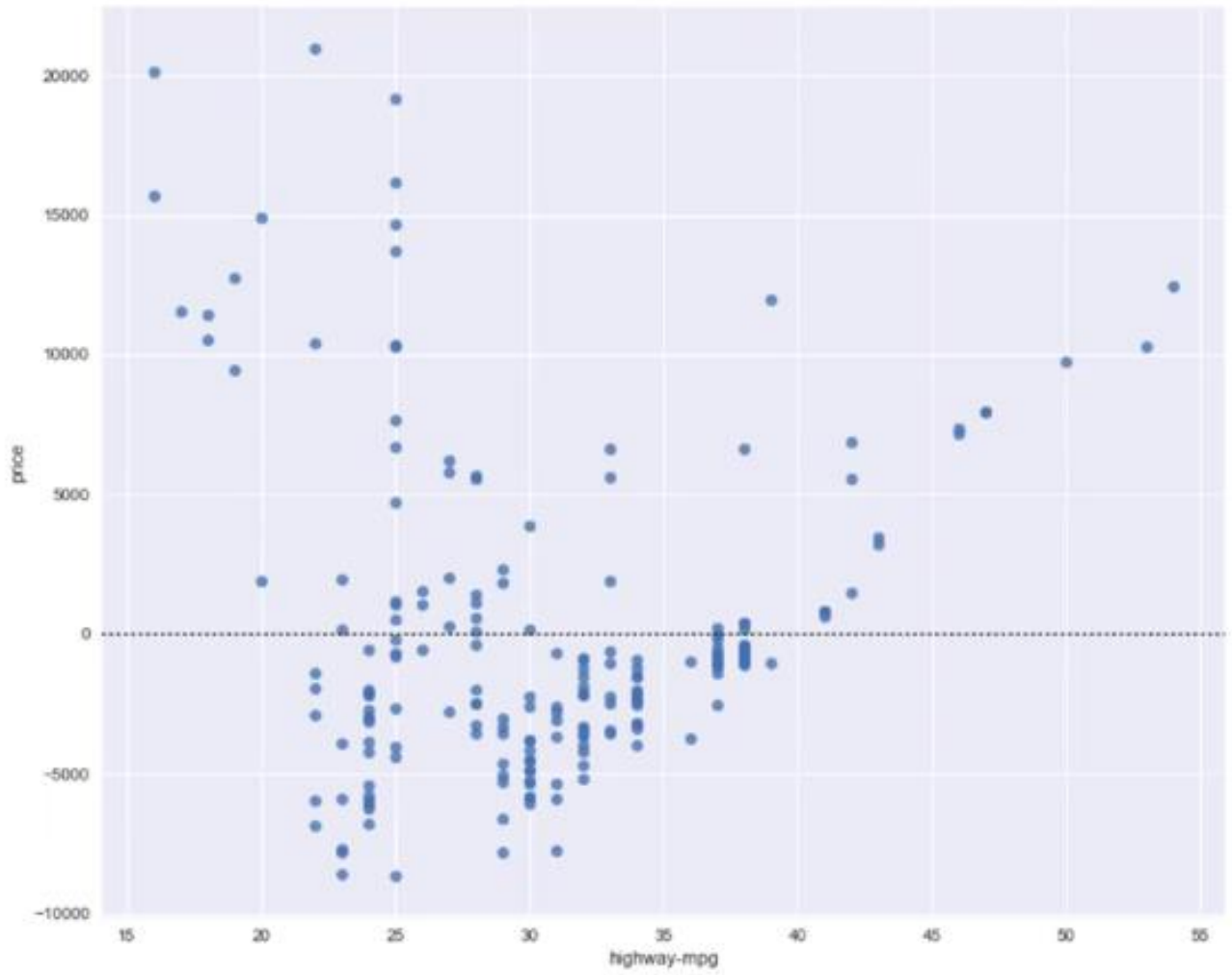
```
array([ 37601.57247984,  36779.83910151,  35958.10572319,  35136.37234487,
        34314.63896655,  33492.90558823,  32671.17220989 ,  31849.43883158,
        31027.70545326,  30205.97207494,  29384.23869662,  28562.50531829,
        27740.77193997,  26919.03856165,  26097.30518333,  25275.57180501,
        24453.83842668,  23632.10504836,  22810.37167004,  21988.63829172,
        21166.9049134 ,  20345.17153508,  19523.43815675,  18701.70477843,
        17879.97140011,  17058.23802179,  16236.50464347,  15414.77126514,
        14593.03788682,  13771.3045085 ,  12949.57113018,  12127.83775186,
        11306.10437353,  10484.37099521,   9662.63761689,   8840.90423857,
         8019.17086025,   7197.43748192,   6375.7041036 ,   5553.97072528,
         4732.23734696,   3910.50396864,   3088.77059031,   2267.03721199,
         1445.30383367,    623.57045535,   -198.16292297,  -1019.8963013 ,
        -1841.62967962,  -2663.36305794,  -3485.09643626,  -4306.82981458,
        -5128.5631929 ,  -5950.29657123,  -6772.02994955,  -7593.76332787,
        -8415.49670619,  -9237.23008451, -10058.96346284, -10880.69684116,
       -11702.43021948, -12524.1635978 , -13345.89697612, -14167.63035445,
       -14989.36373277, -15811.09711108, -16632.83048941, -17454.56386773,
       -18276.29724606, -19098.03062438, -19919.7640027 , -20741.49738102,
```

Visualization

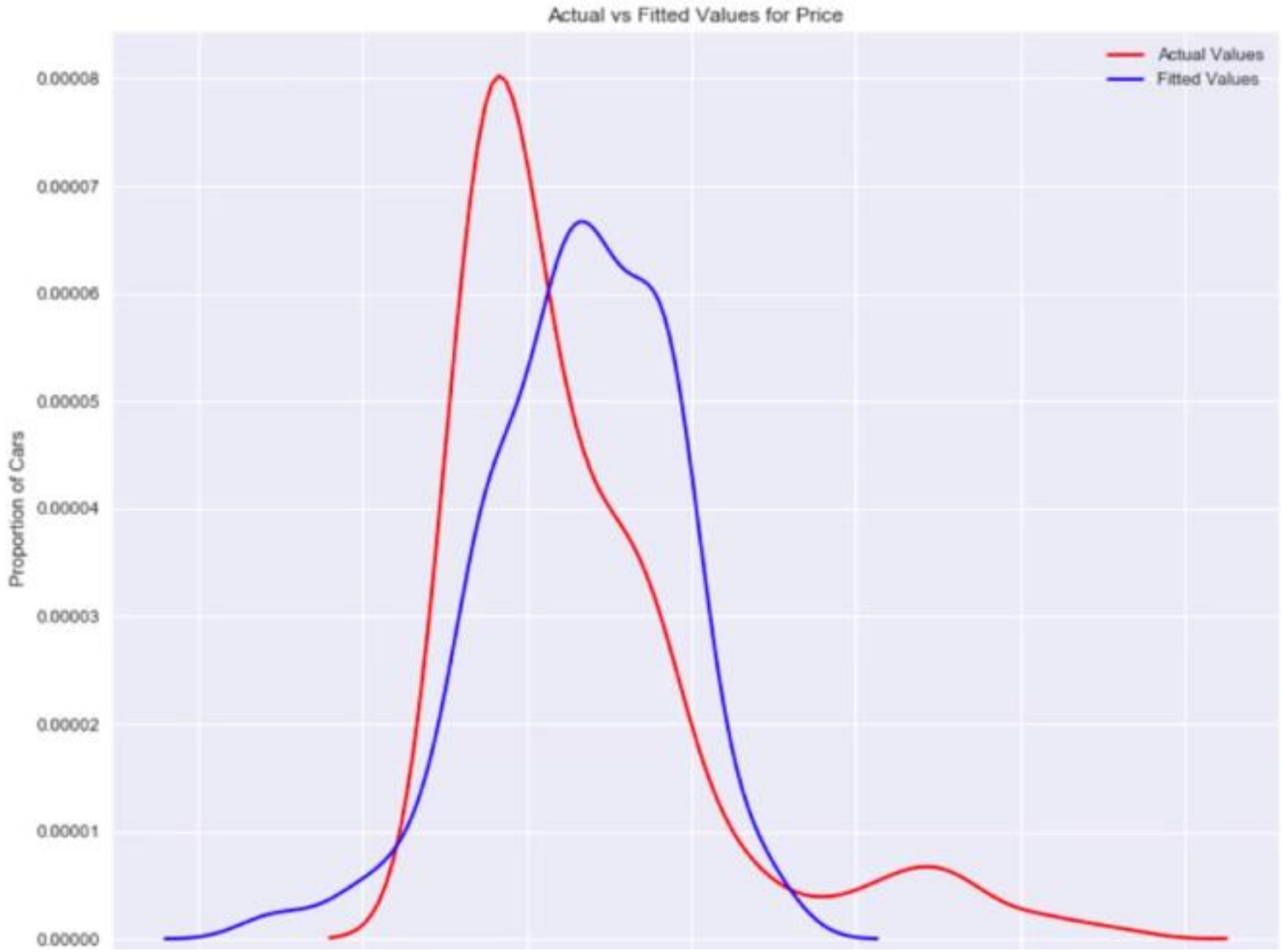
- Simply visualizing your data with a regression



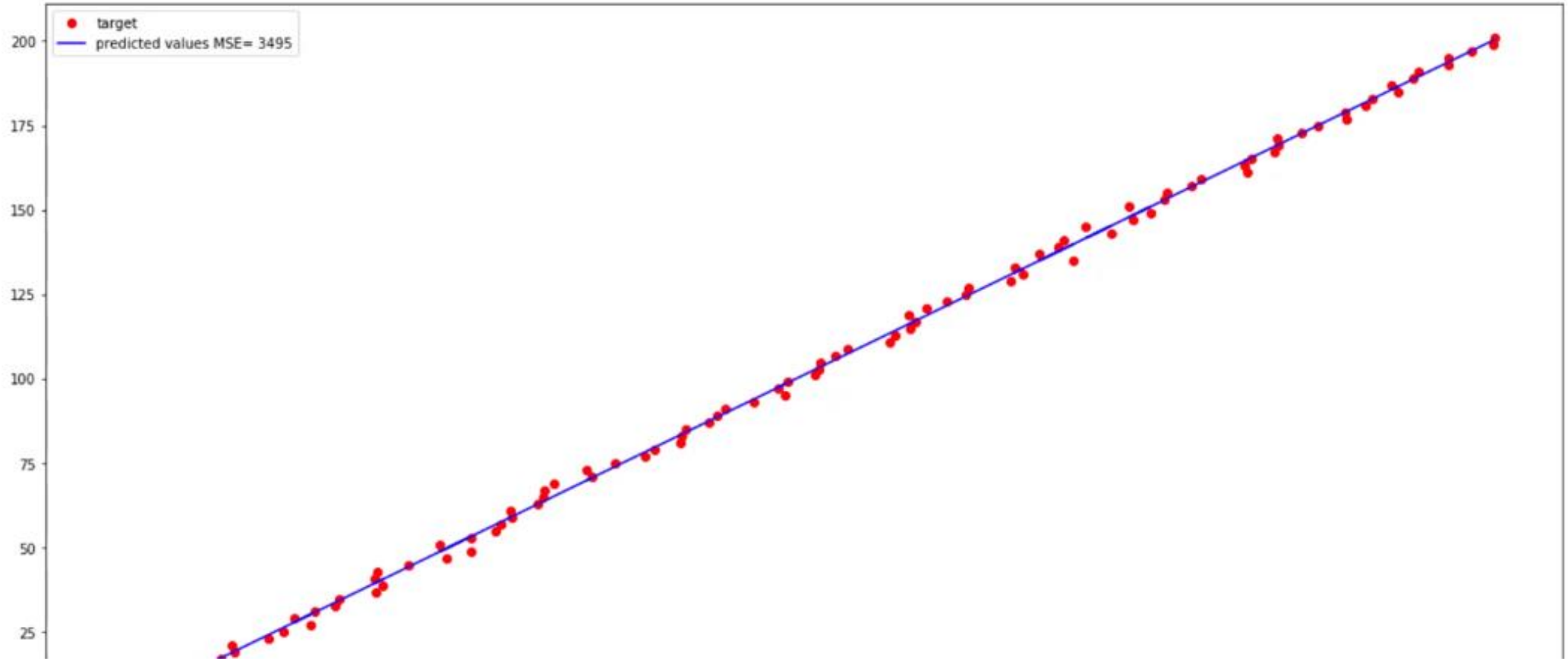
Residual Plot



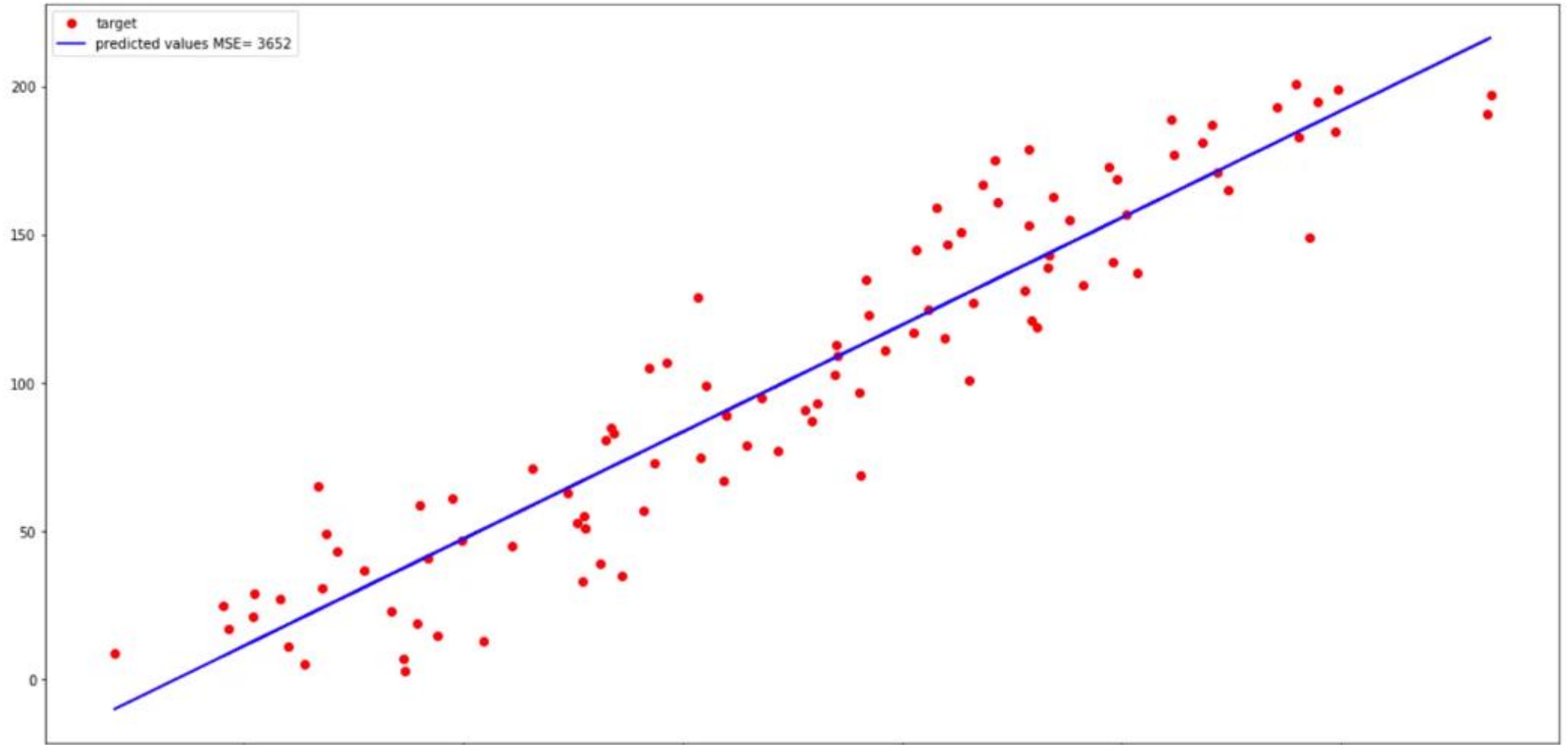
Visualization



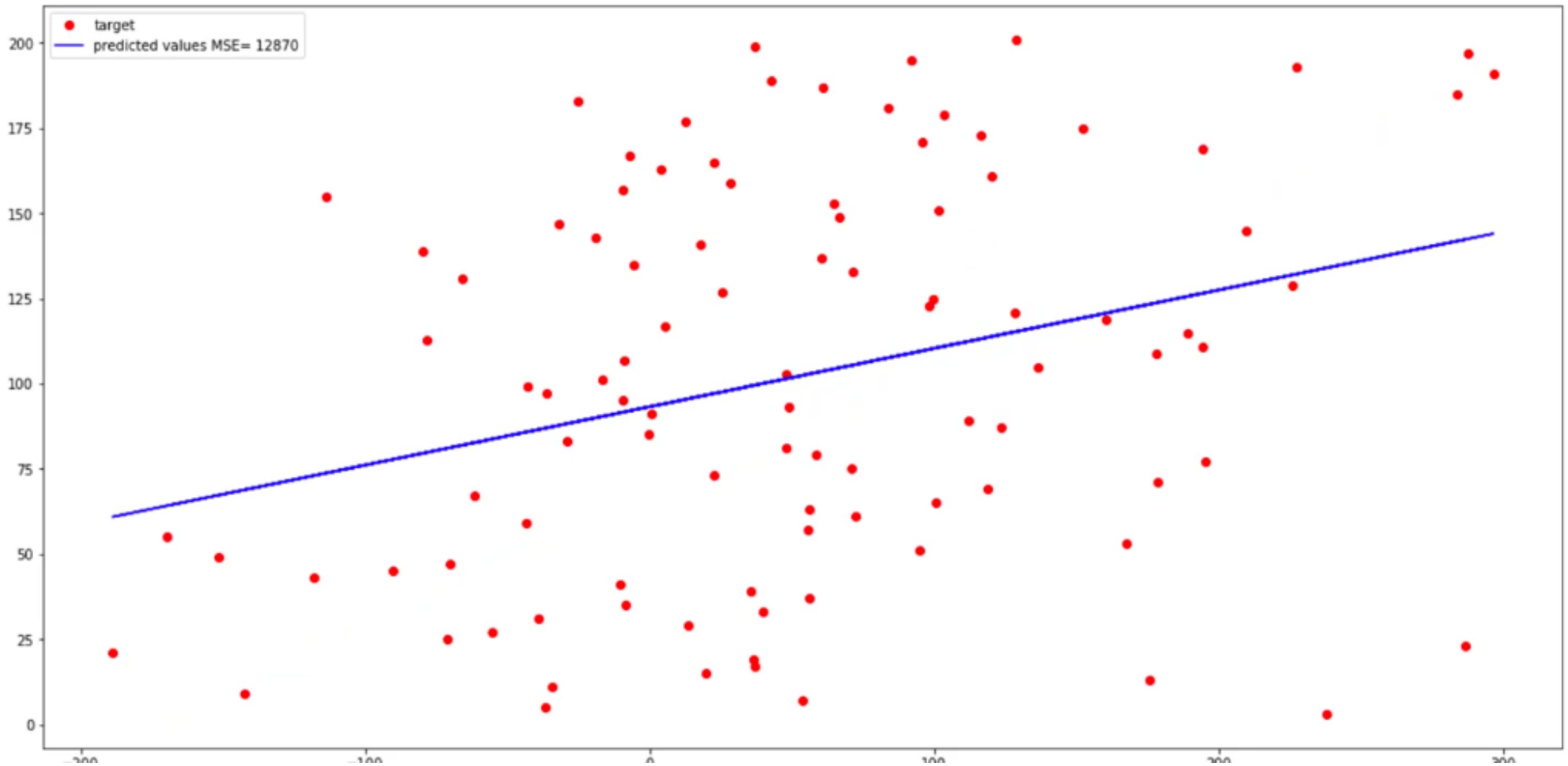
Numerical measures for Evaluation



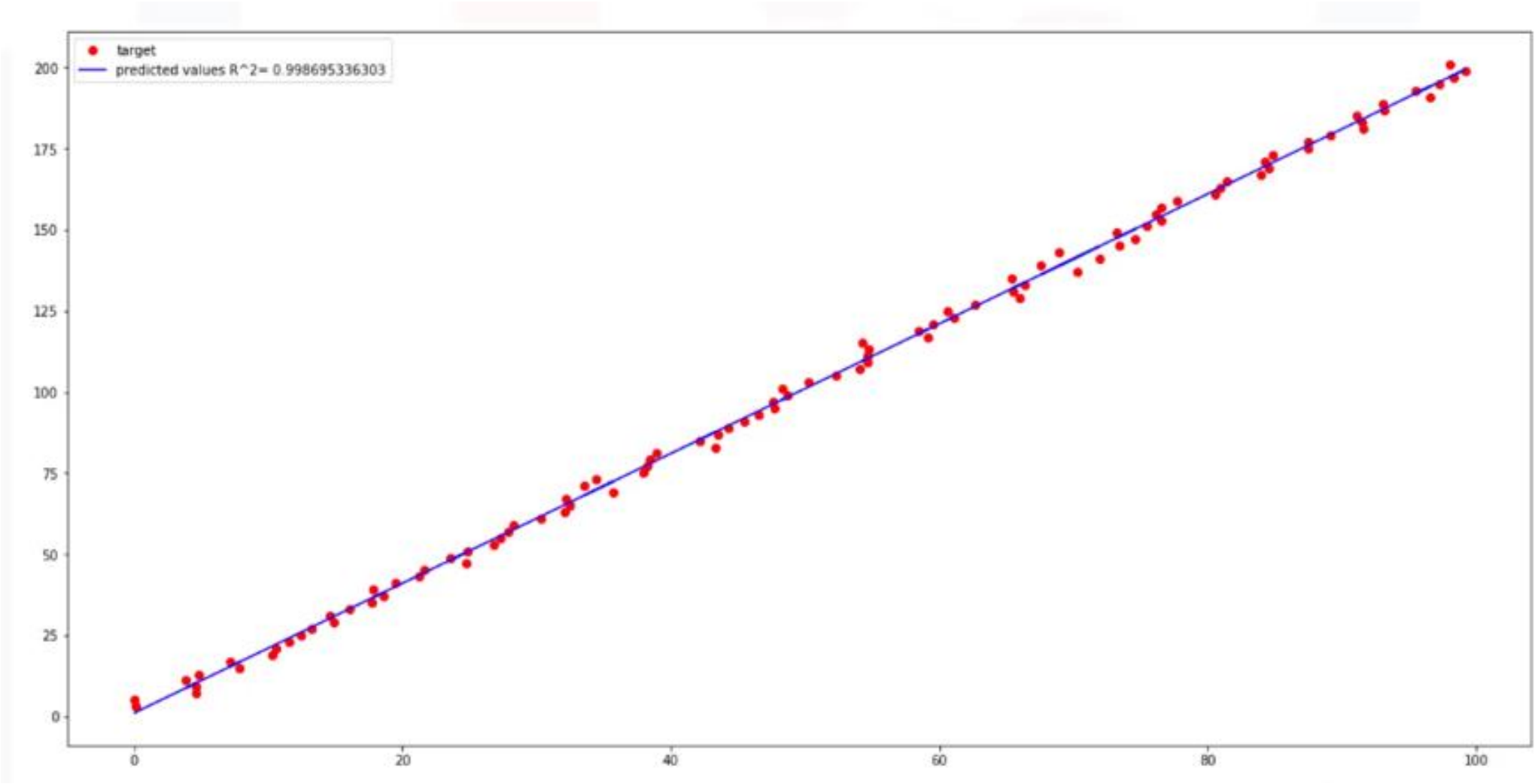
Numerical measures for Evaluation



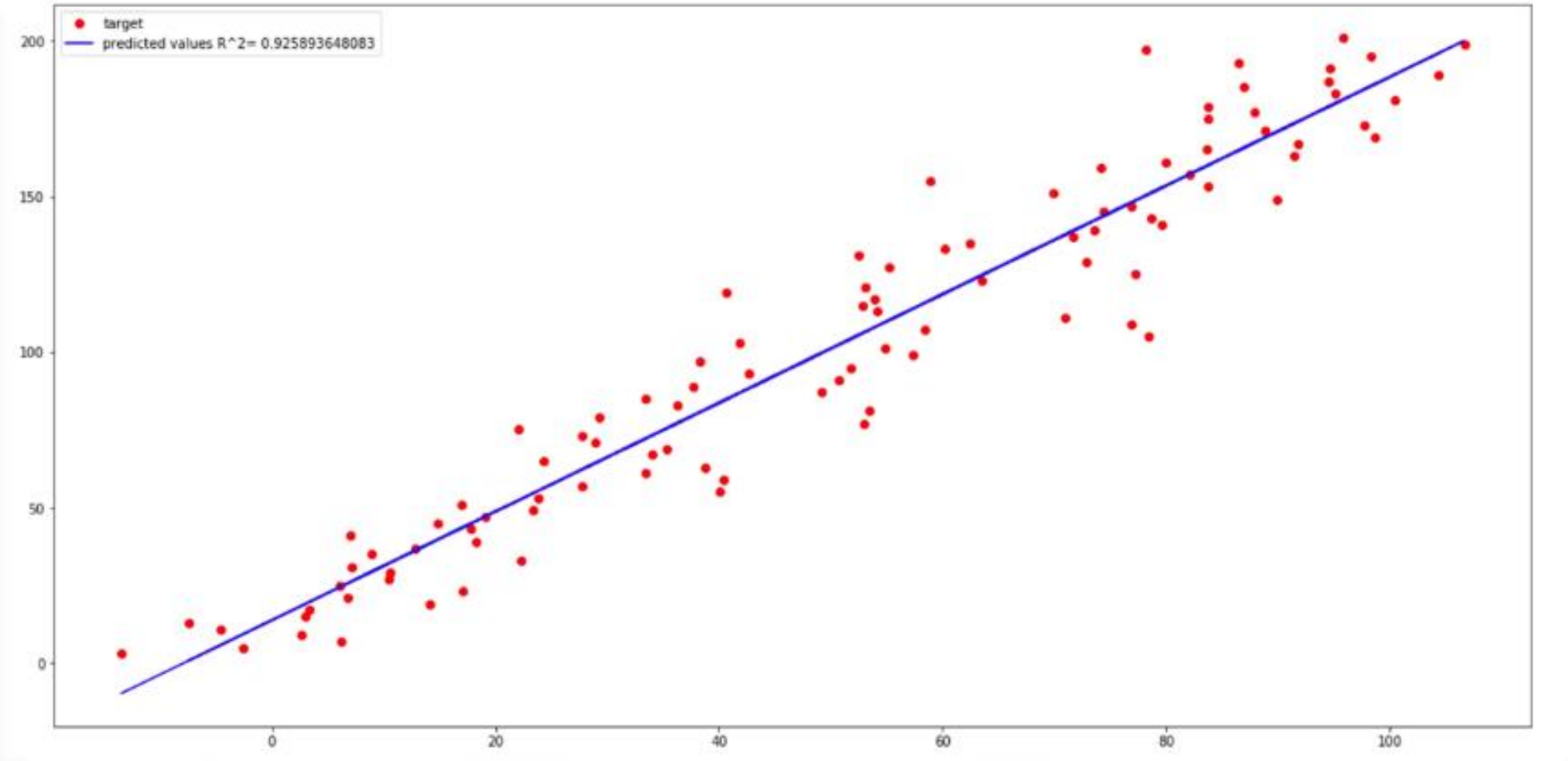
Numerical measures for Evaluation



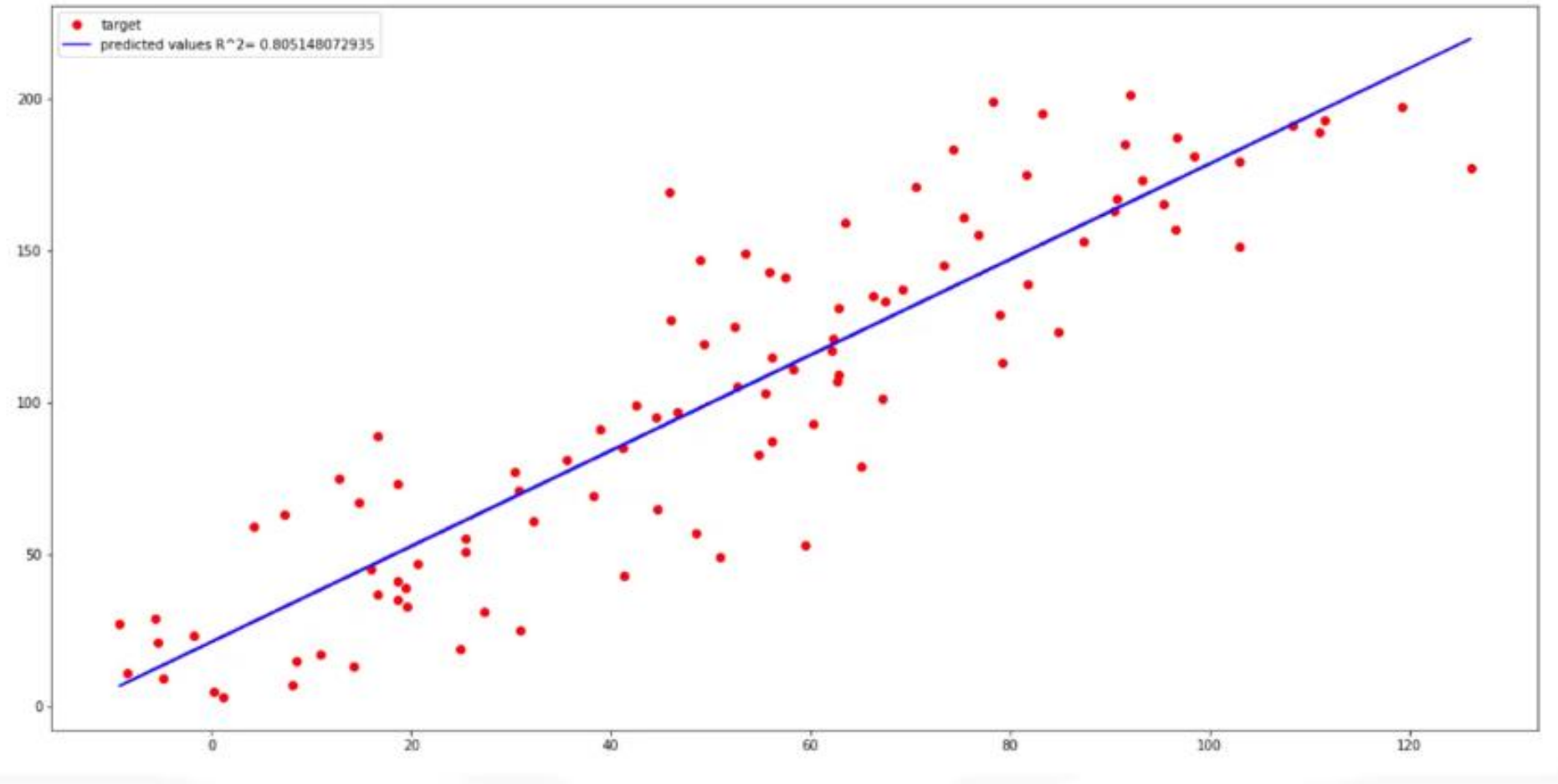
Numerical measures for Evaluation



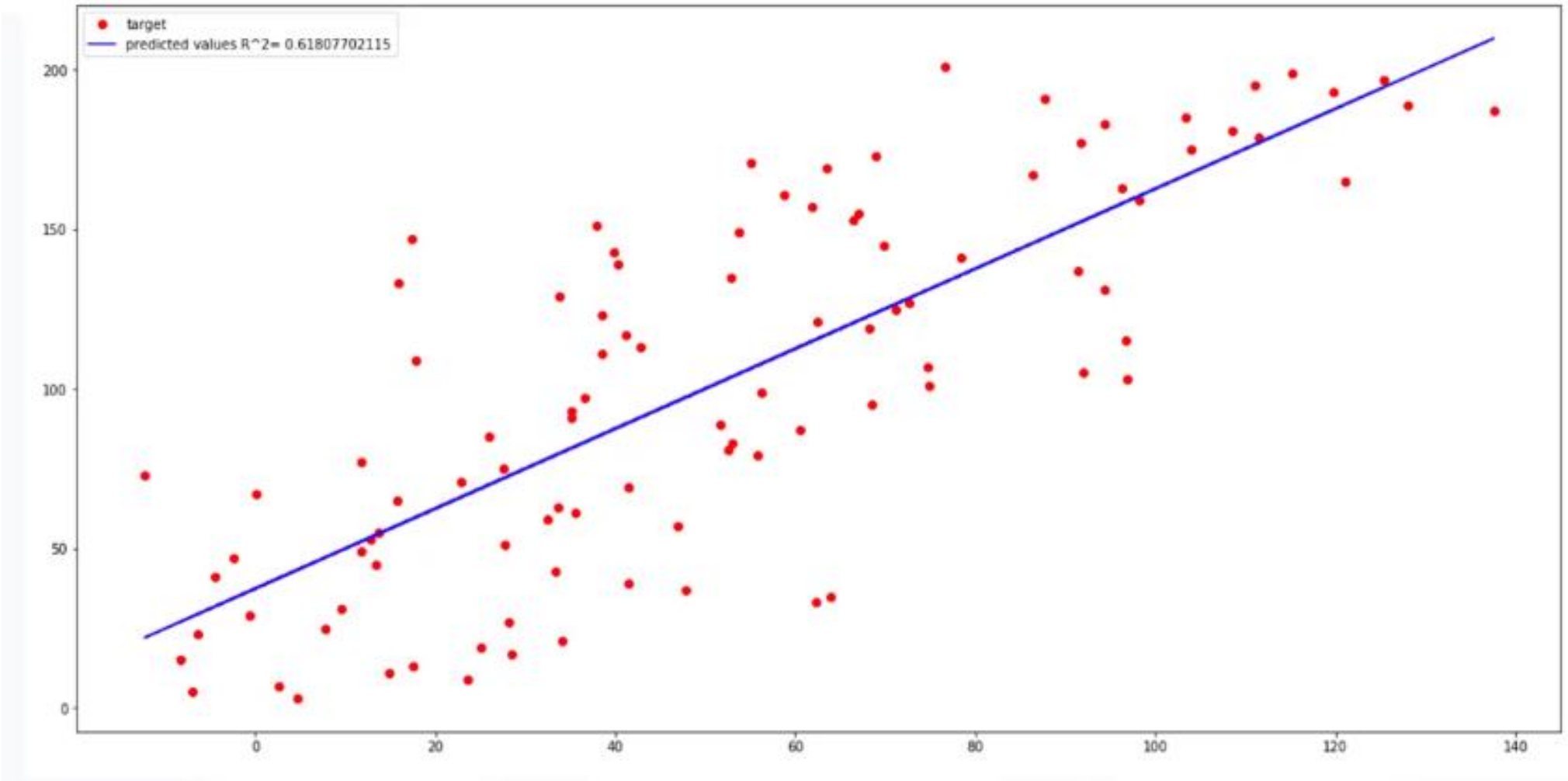
Numerical measures for Evaluation



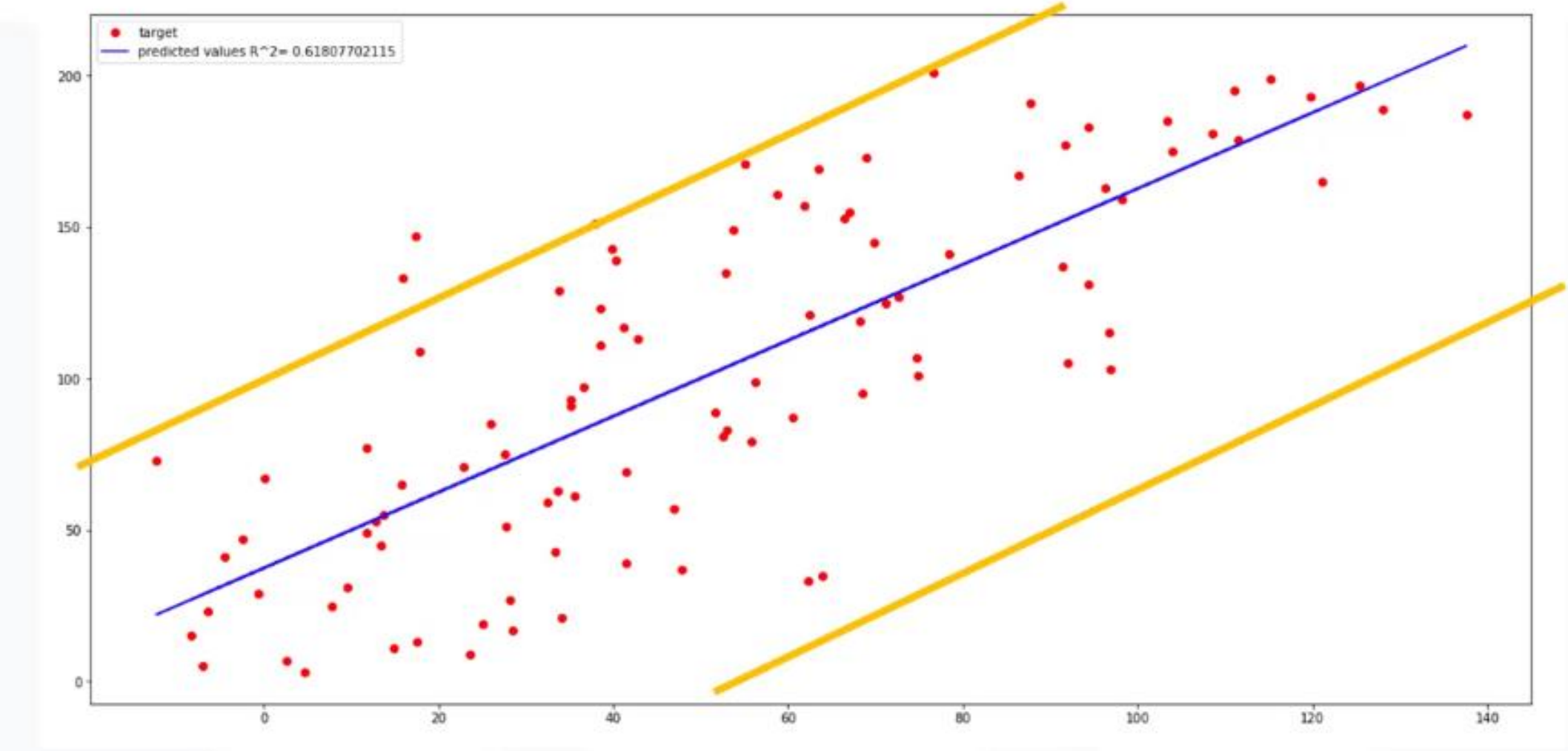
Numerical measures for Evaluation



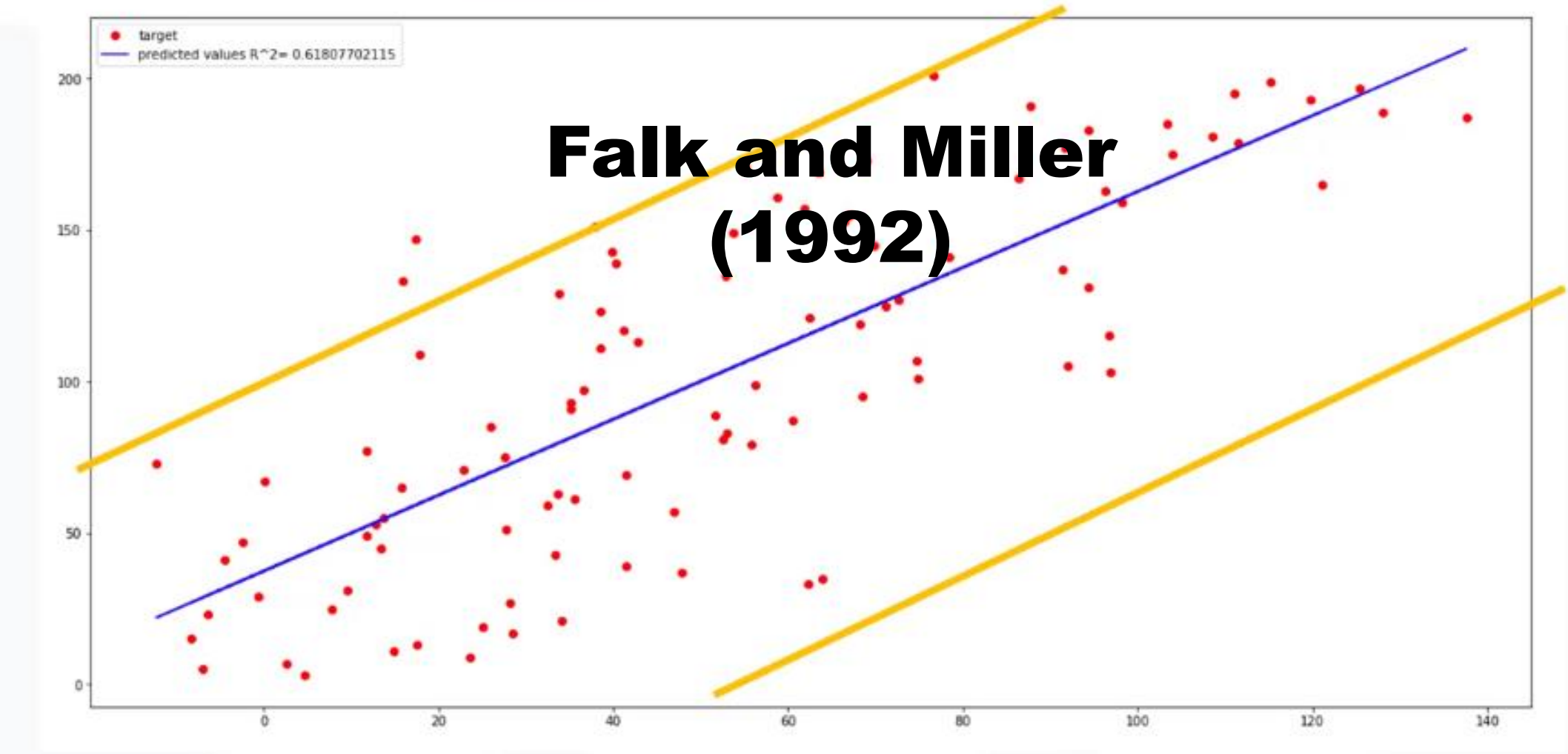
Numerical measures for Evaluation



Numerical measures for Evaluation



Numerical measures for Evaluation



Comparing MLR and SLR

1. Is a lower MSE always implying a better fit?
   - Not necessarily.
2. MSE for an MLR model will be smaller than the MSE for an SLR model, since the errors of the data will decrease when more variables are included in the model
3. Polynomial regression will also have a smaller MSE then regular regression
4. A similar inverse relationship holds for R^2
5. In the next section we will look at better ways to evaluate the model

Chapter 5: Model Evaluation and Refinement

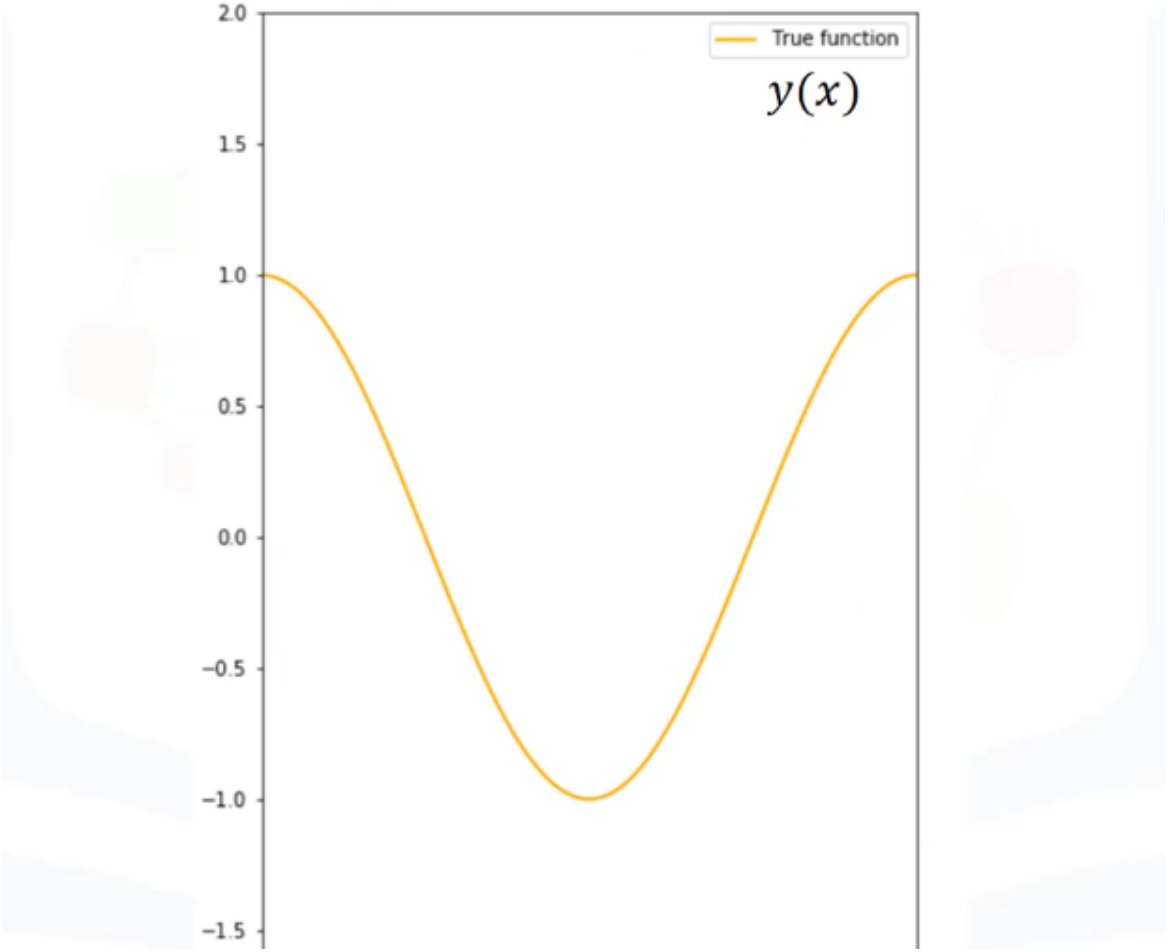
Chapter Objectives

In this chapter, you will learn about:

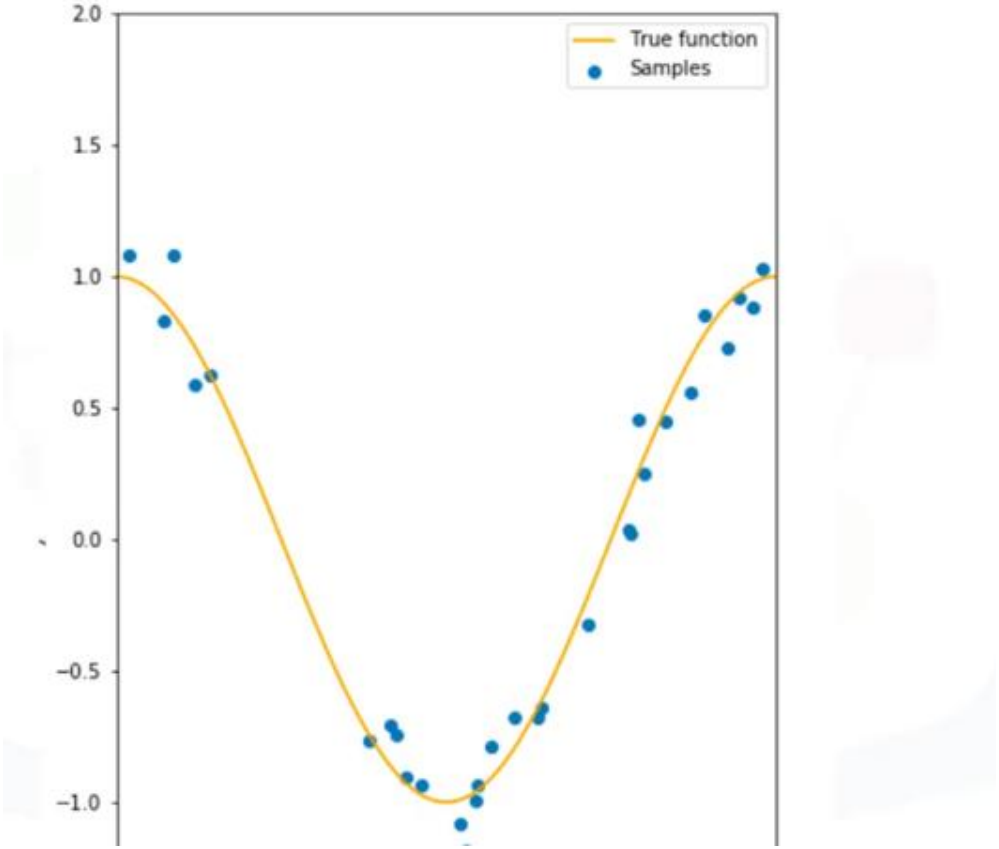
- Model Evaluation
- Over-fitting, Under-fitting and Model Selection
- Ridge Regression
- Grid Search
- Question:
  - ***How can you be certain your model works in the real world and performs optimally***

Over-fitting, Under-fitting and Model Selection

Model Selection

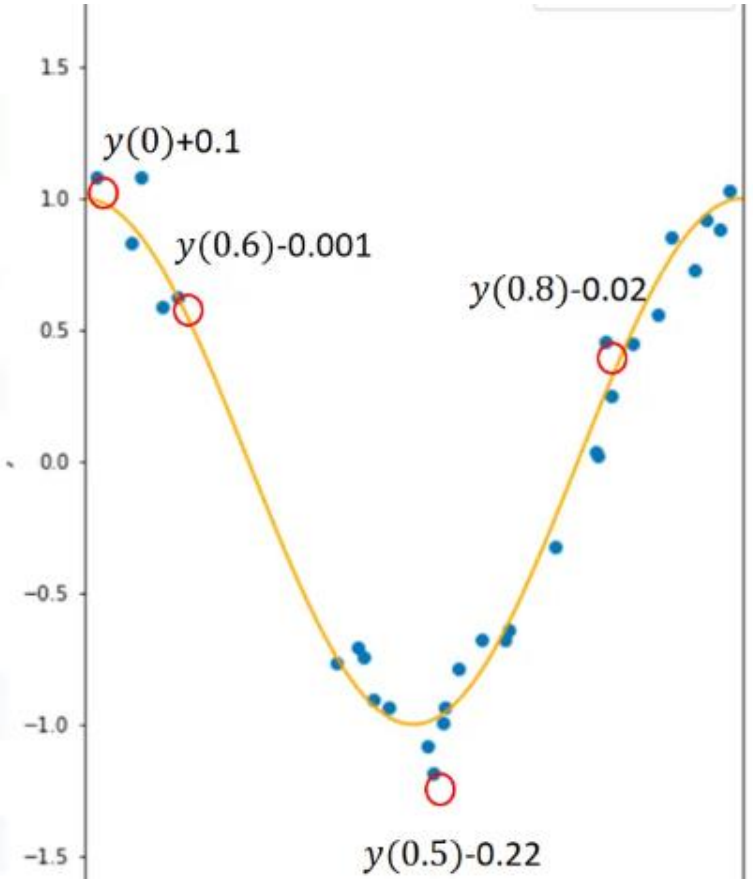


Model Selection



Model Selection

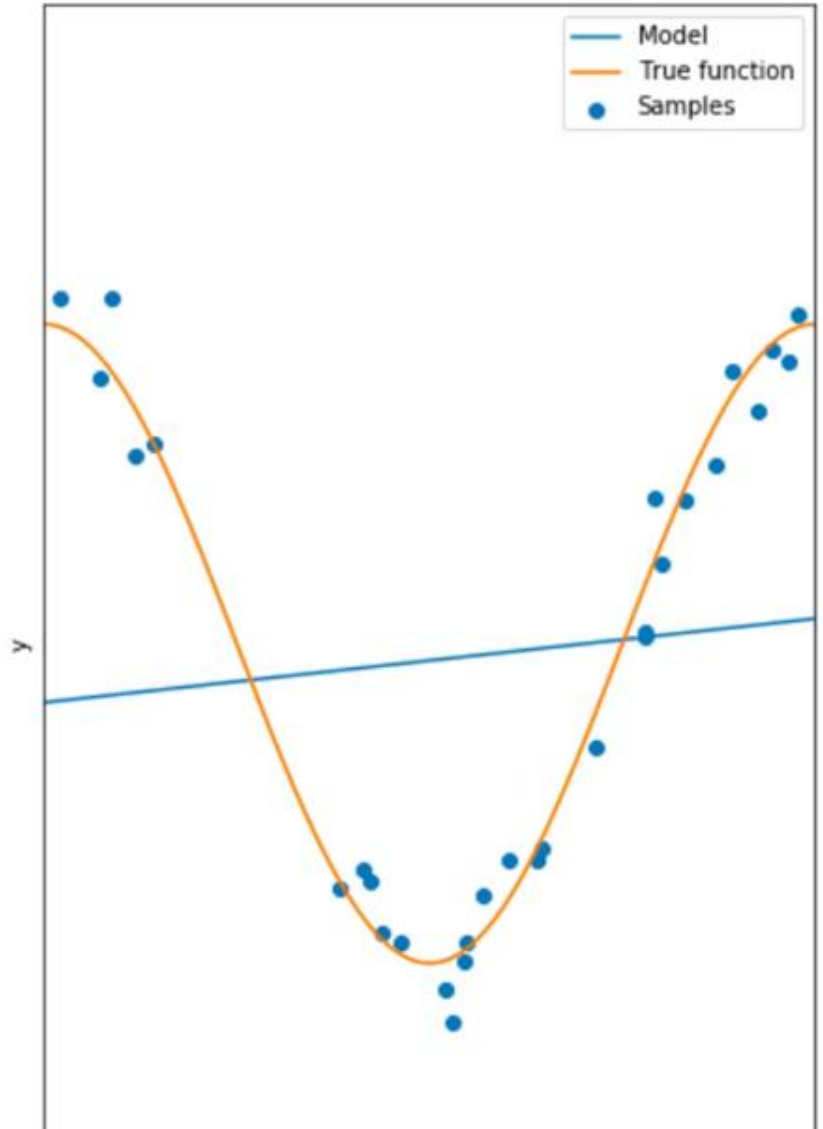
$y(x)$+noise



purpose s to determine the other of polynomial

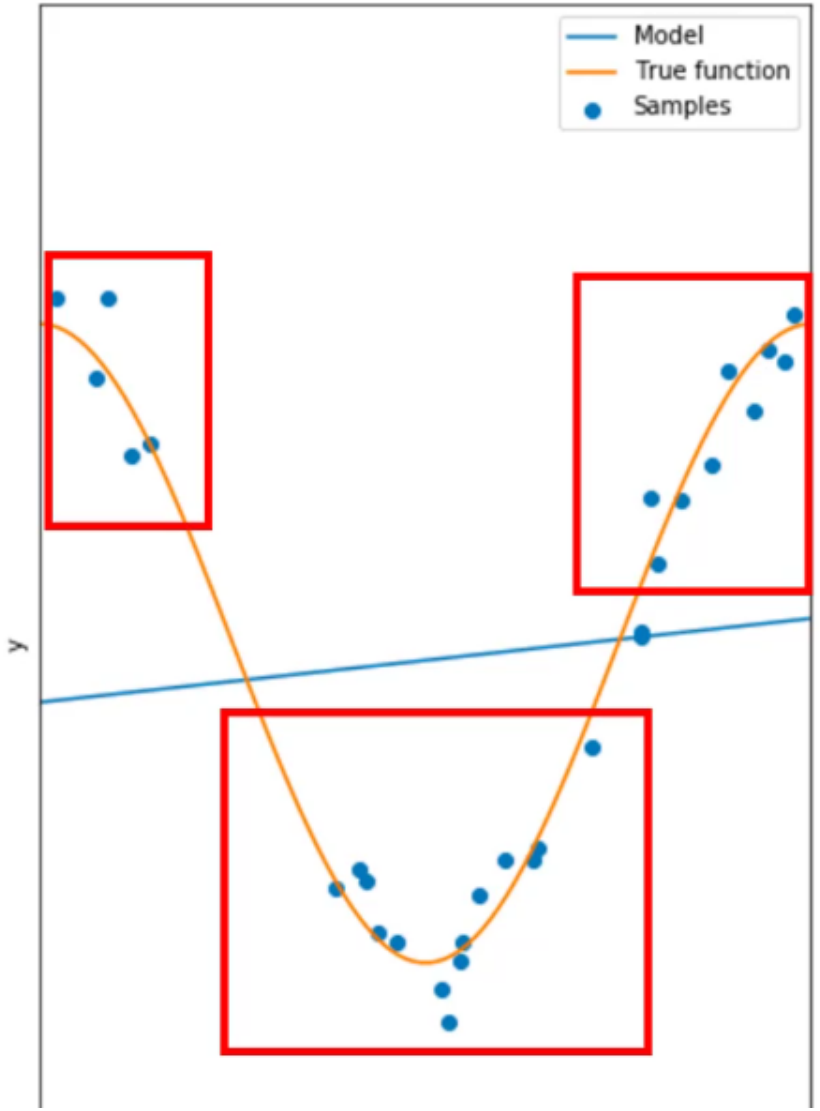
Model Selection

$$y = b_0 + b_1\, x$$



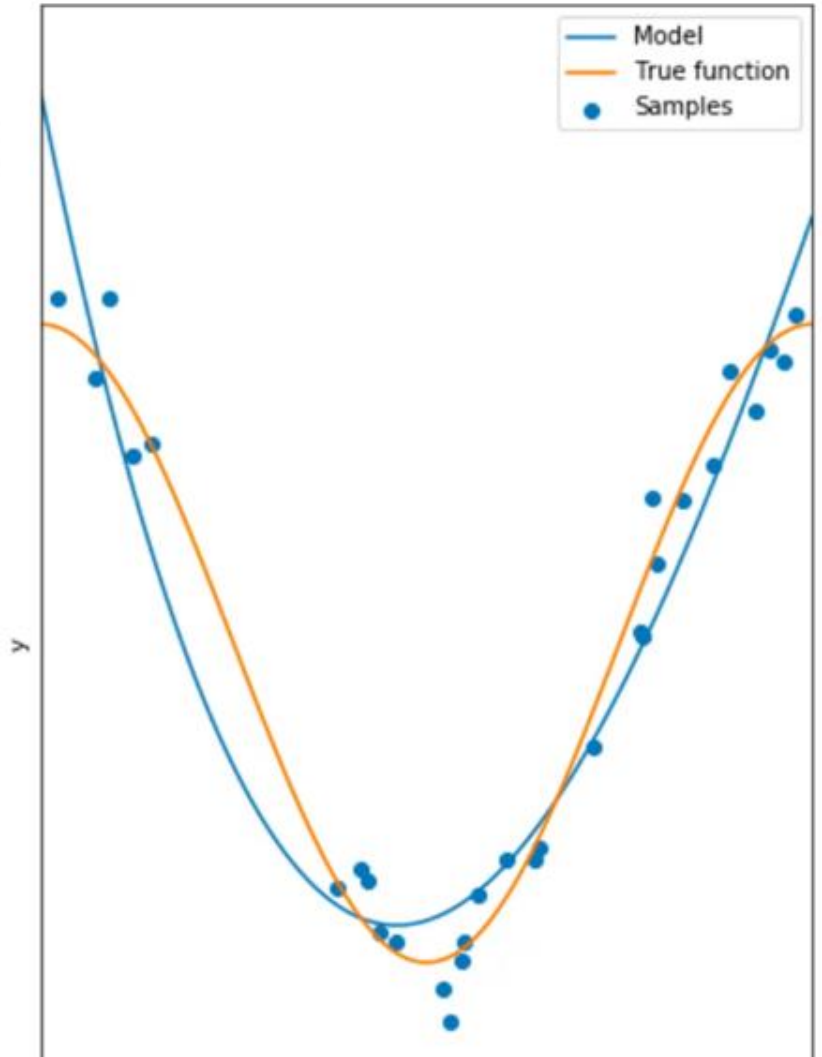
Model Selection

$$y = b_0 + b_1 x$$



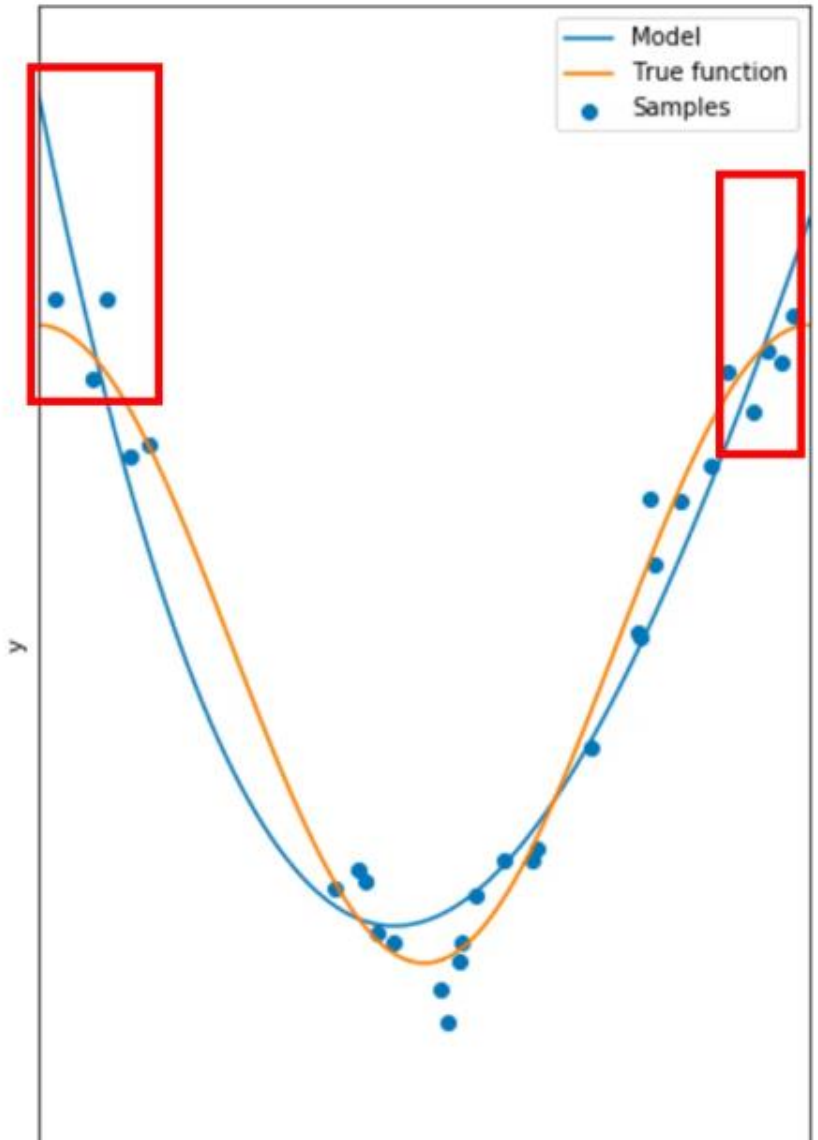
Model Selection

$$y = b_0 + b_1\, x + b_2\, x^2$$



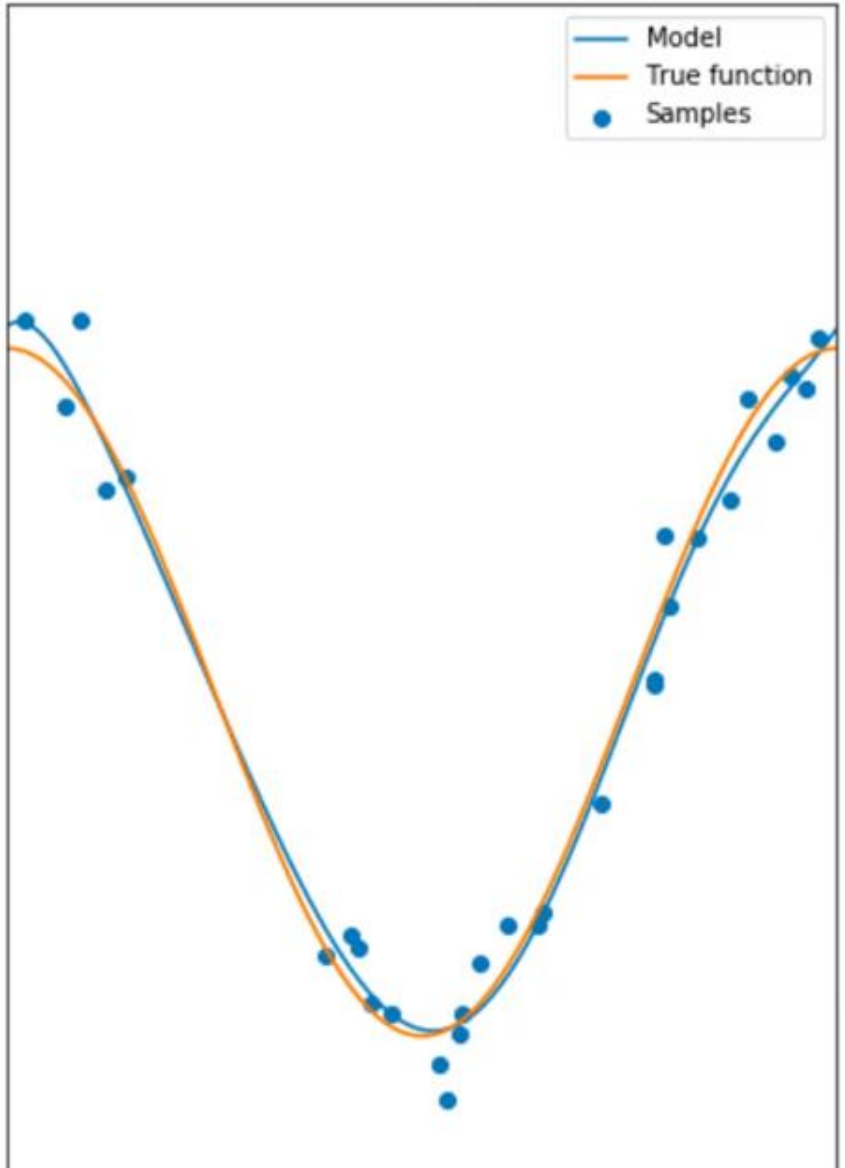
Model Selection

$$y = b_0 + b_1\, x + b_2\, x^2$$



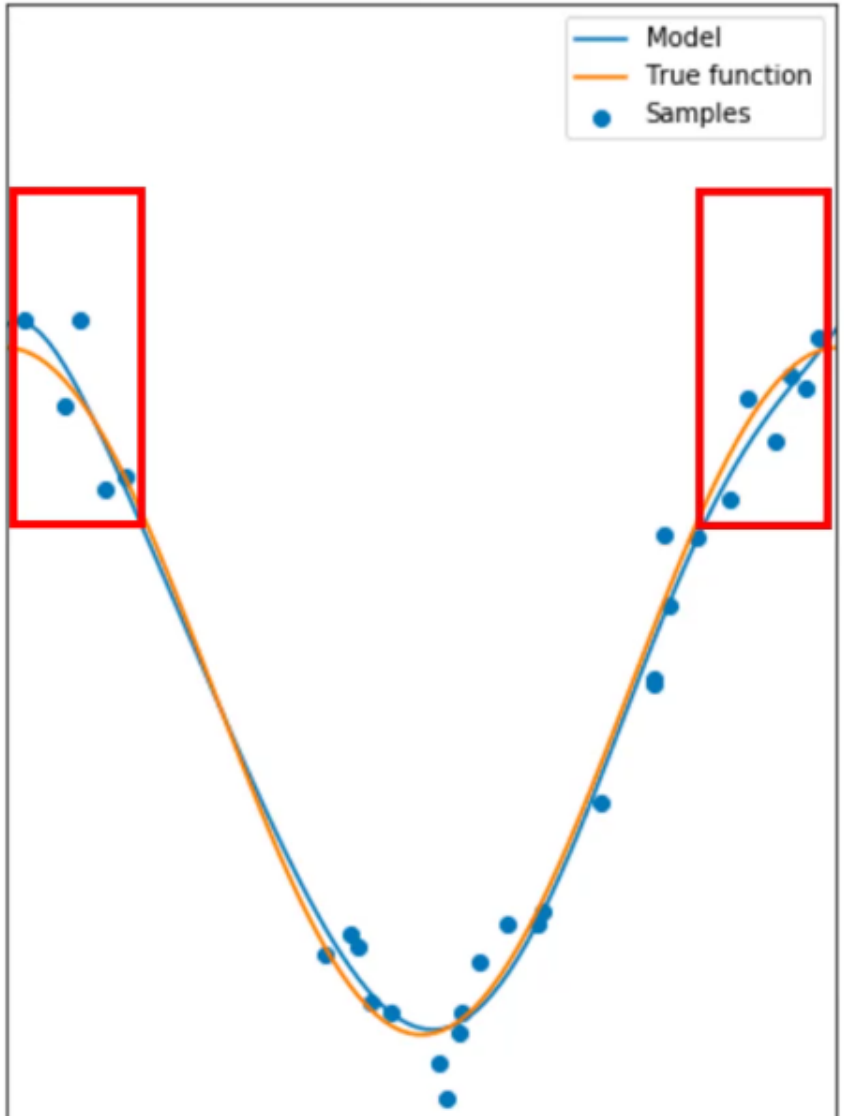
Model Selection

$$\hat{y} = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4 + b_5 x^5 + b_6 x^6 + b_7 x^7 + b_8 x^8$$



Model Selection

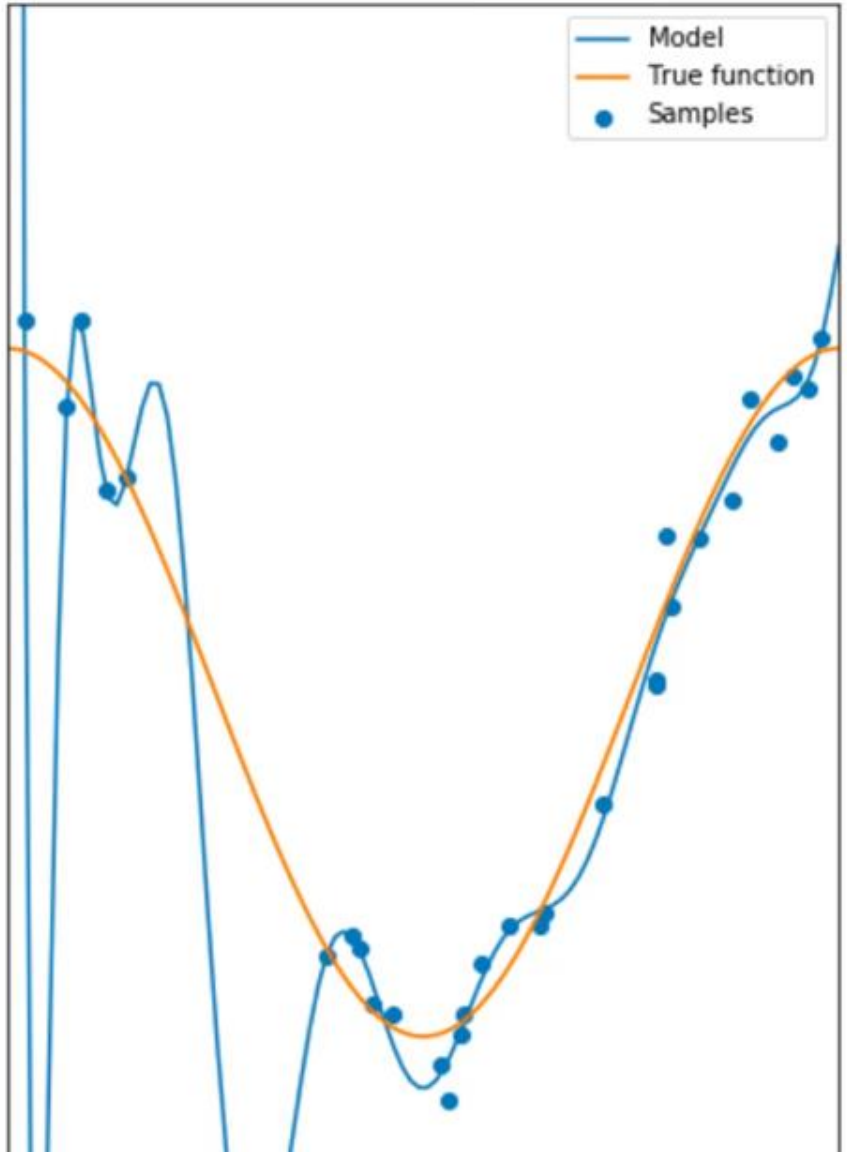
$$\hat{y} = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4 + b_5 x^5 + b_6 x^6 + b_7 x^7 + b_8 x^8$$



Model Selection

$$\hat{y} = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4 + b_5 x^5 + b_6 x^6 + b_7 x^7 + b_8 x^8 + ..$$

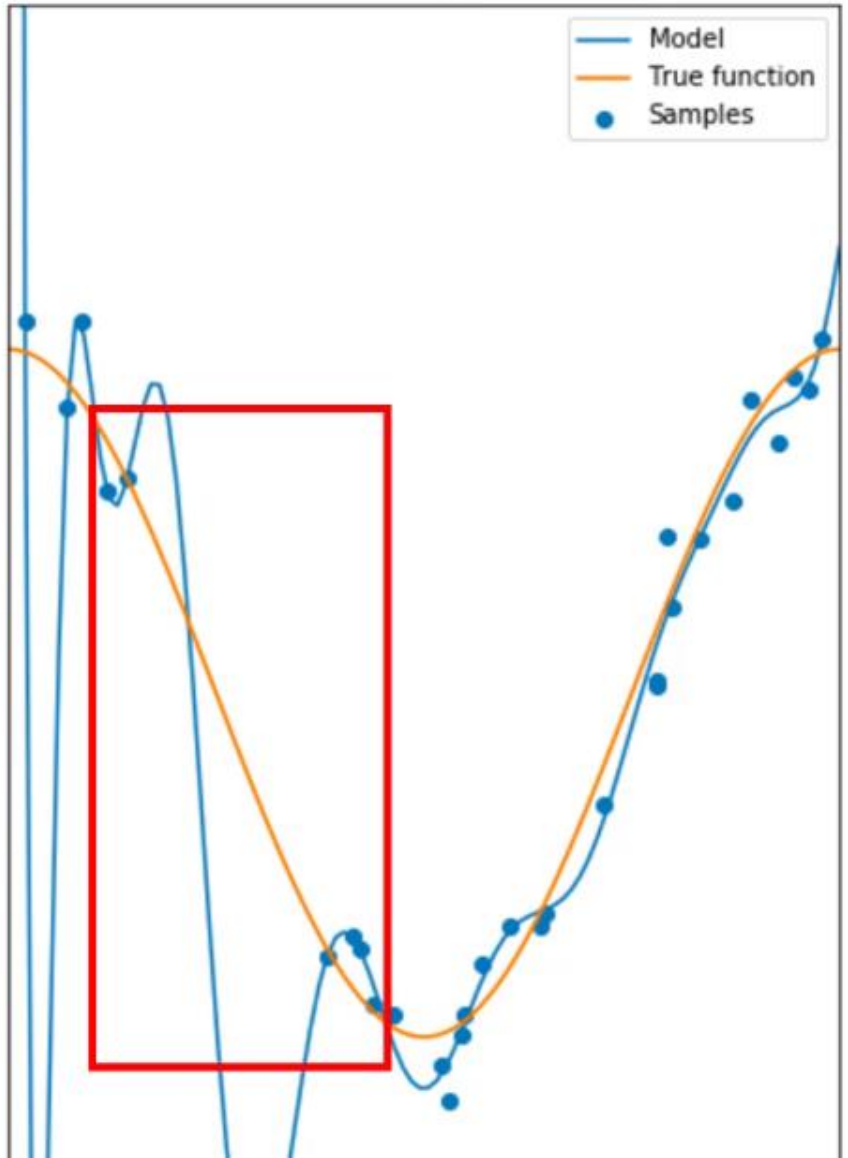
$$+ b_9 x^9 + b_{10} x^{10} + b_{11} x^{11} + b_{12} x^{12} + b_{13} x^{13} + b_{14} x^{14} + b_{15} x^{15} + b_{16} x^{16}$$



Model Selection

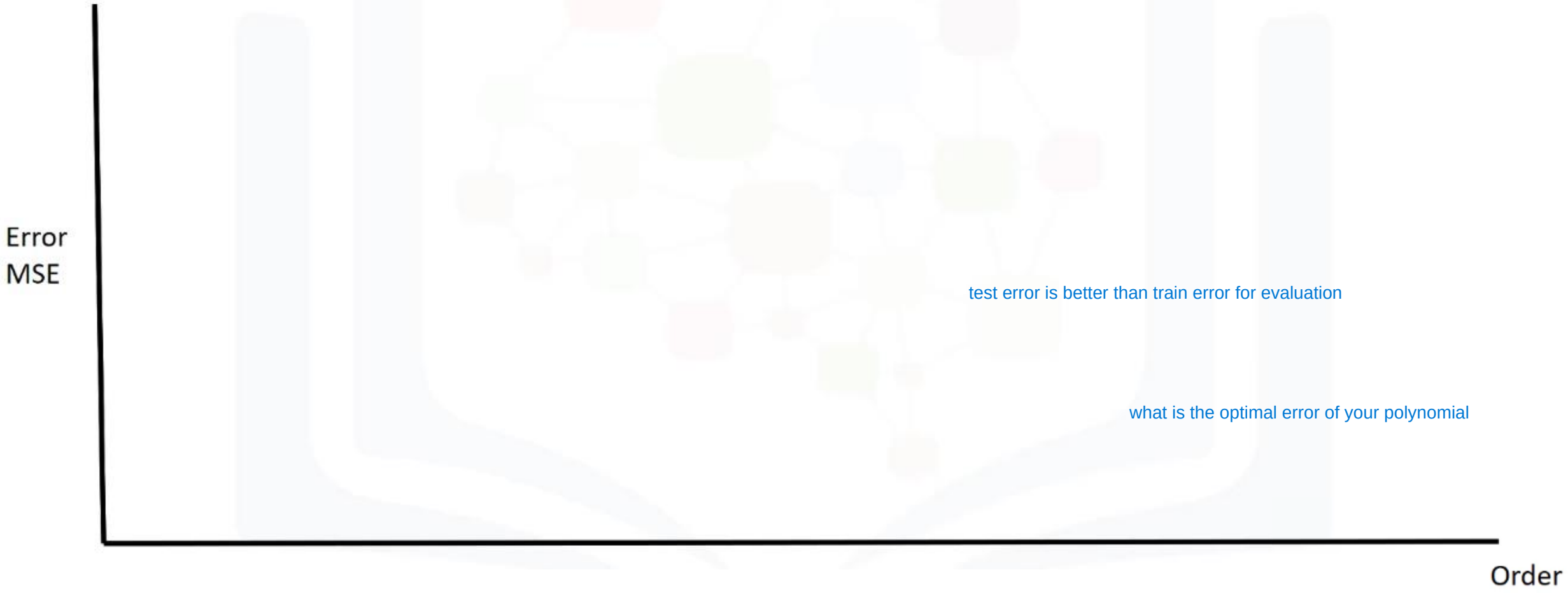
$$\hat{y} = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4 + b_5 x^5 + b_6 x^6 + b_7 x^7 + b_8 x^8 + ..$$

$$+ b_9 x^9 + b_{10} x^{10} + b_{11} x^{11} + b_{12} x^{12} + b_{13} x^{13} + b_{14} x^{14} + b_{15} x^{15} + b_{16} x^{16}$$



overfitting is when the model generalize to the unseen data

Model Selection



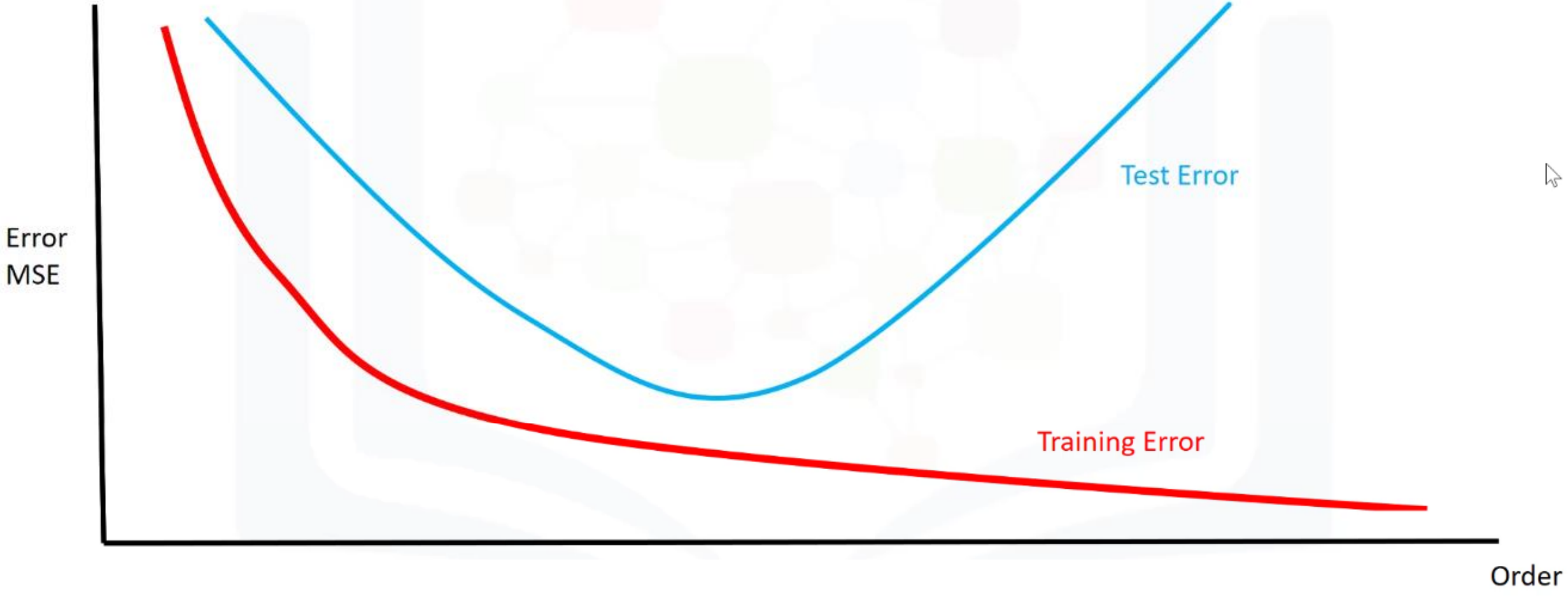
test error is better than train error for evaluation

what is the optimal error of your polynomial

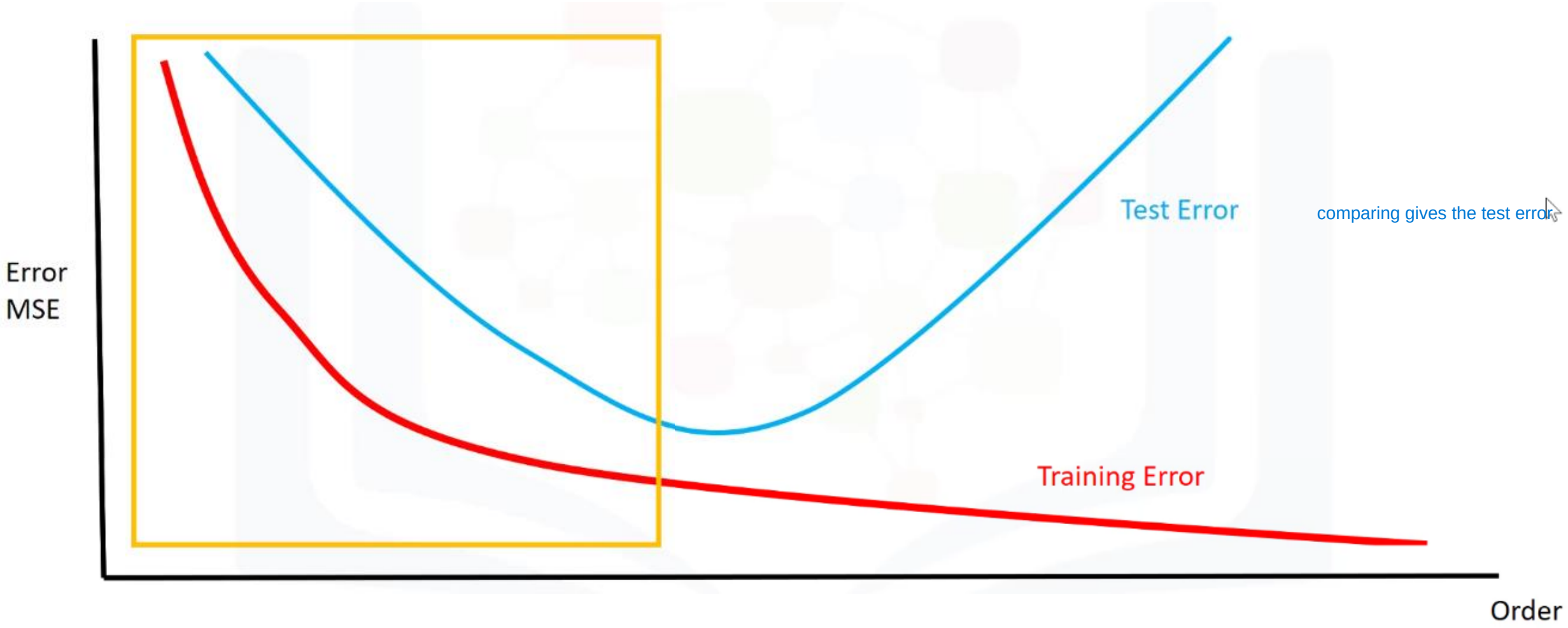
Model Selection

Model Selection

Model Selection

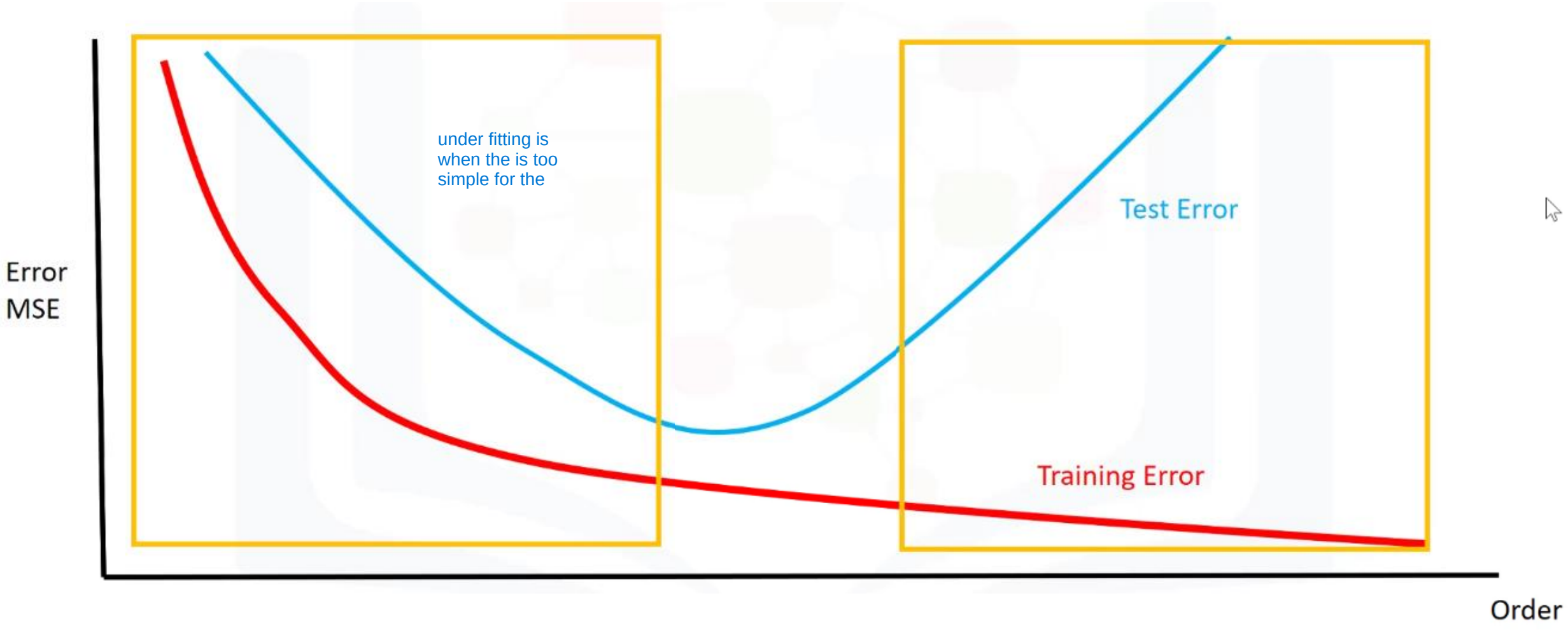


Model Selection

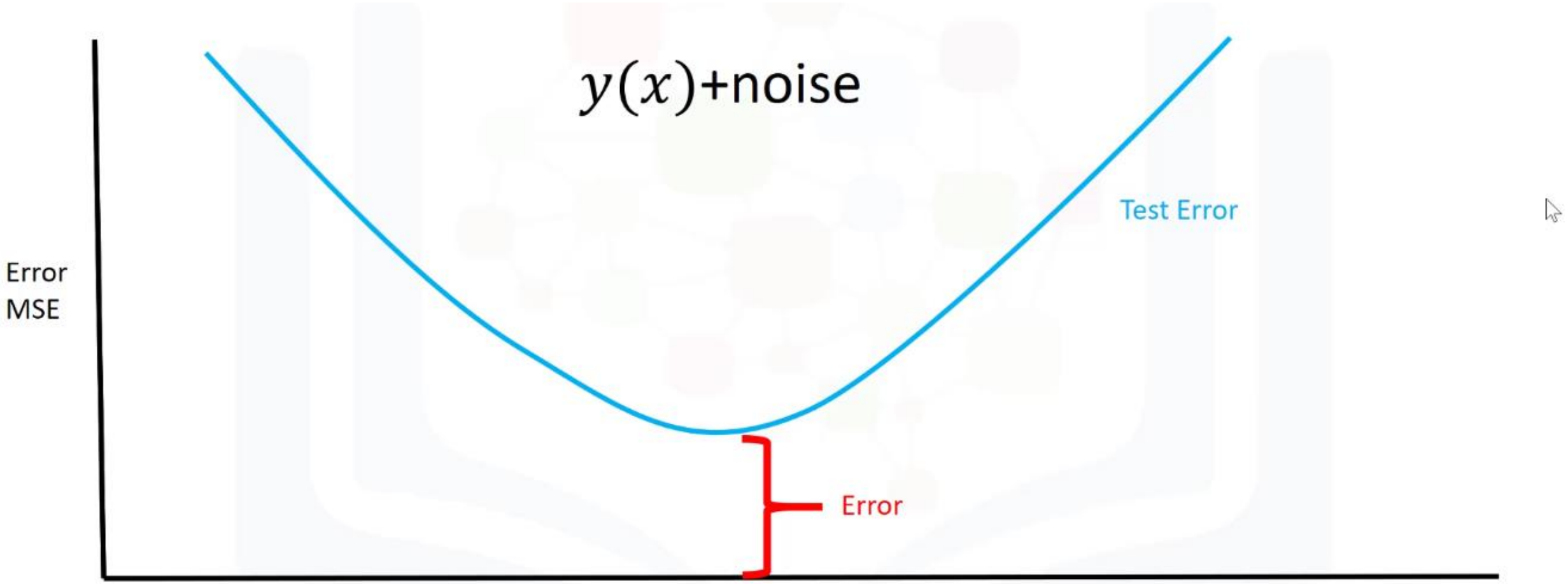


comparing gives the test error

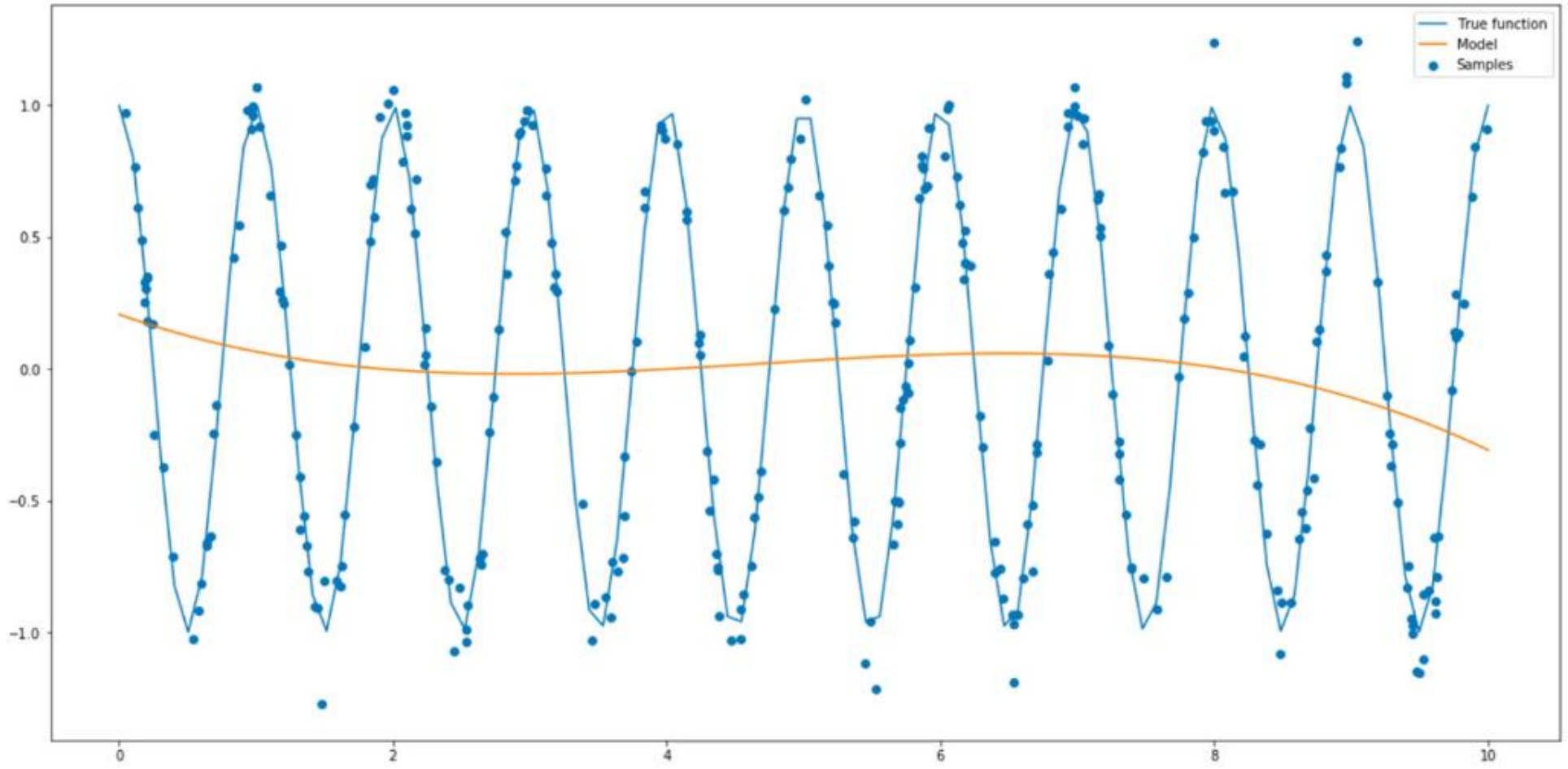
Model Selection



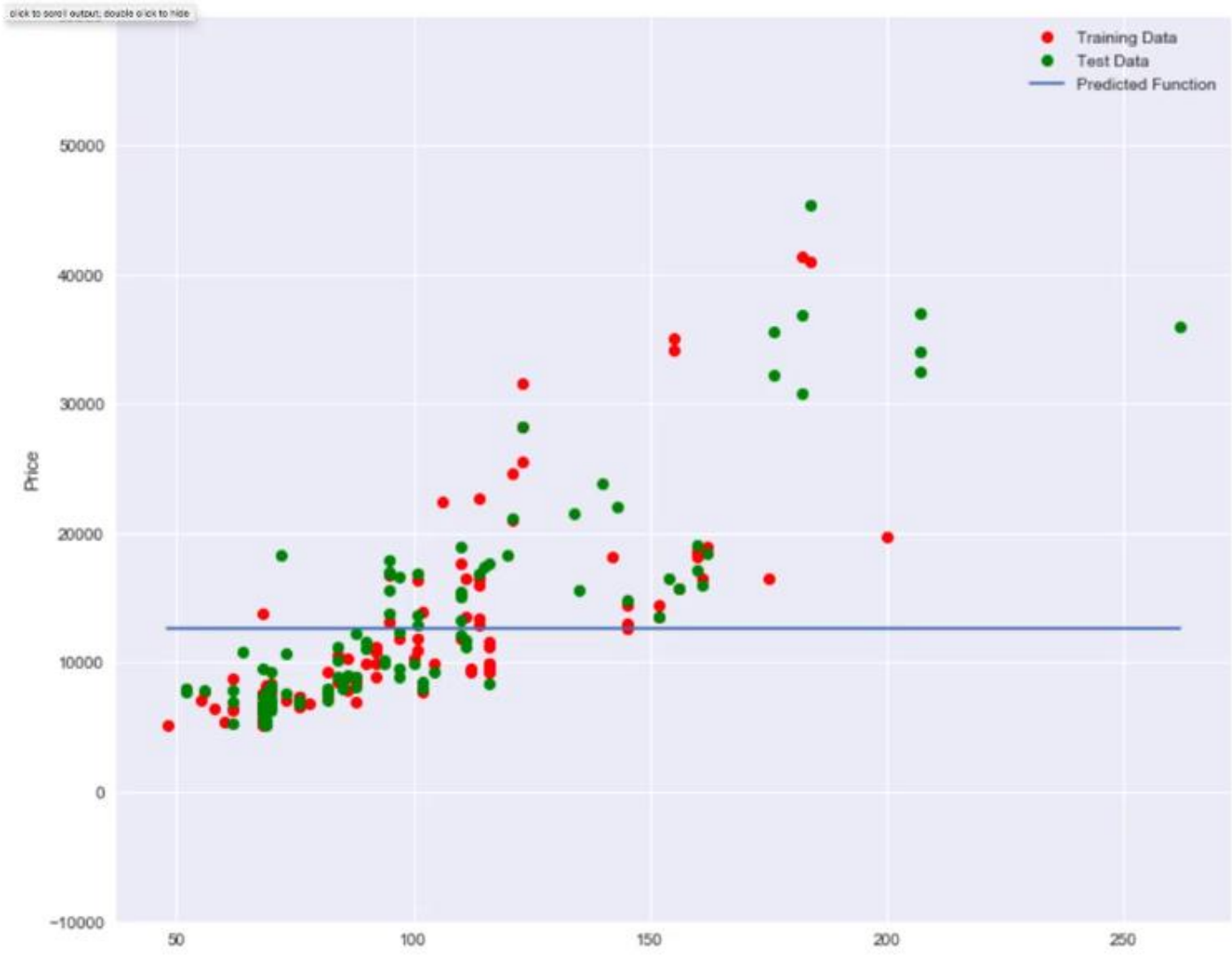
Model Selection



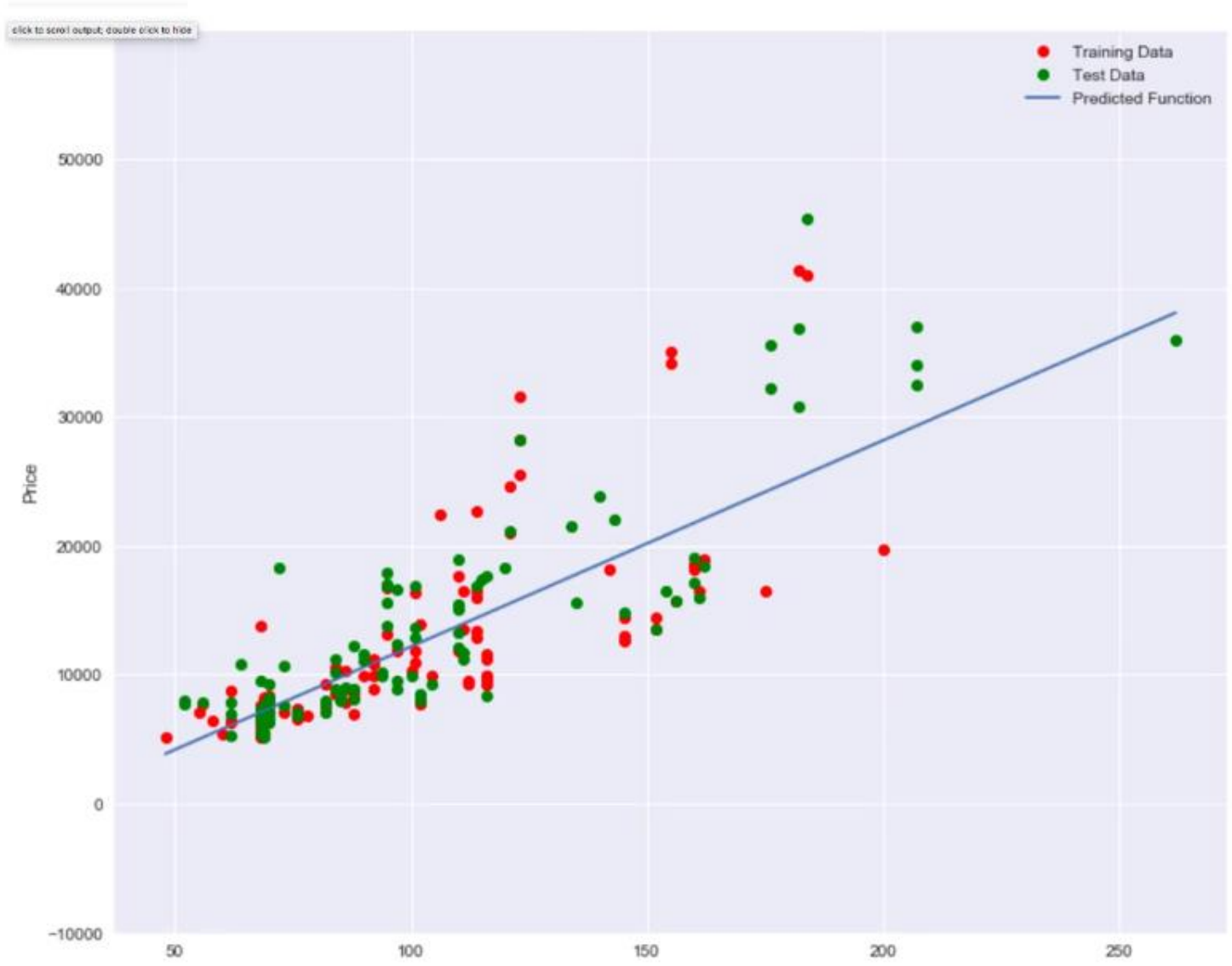
Model Selection



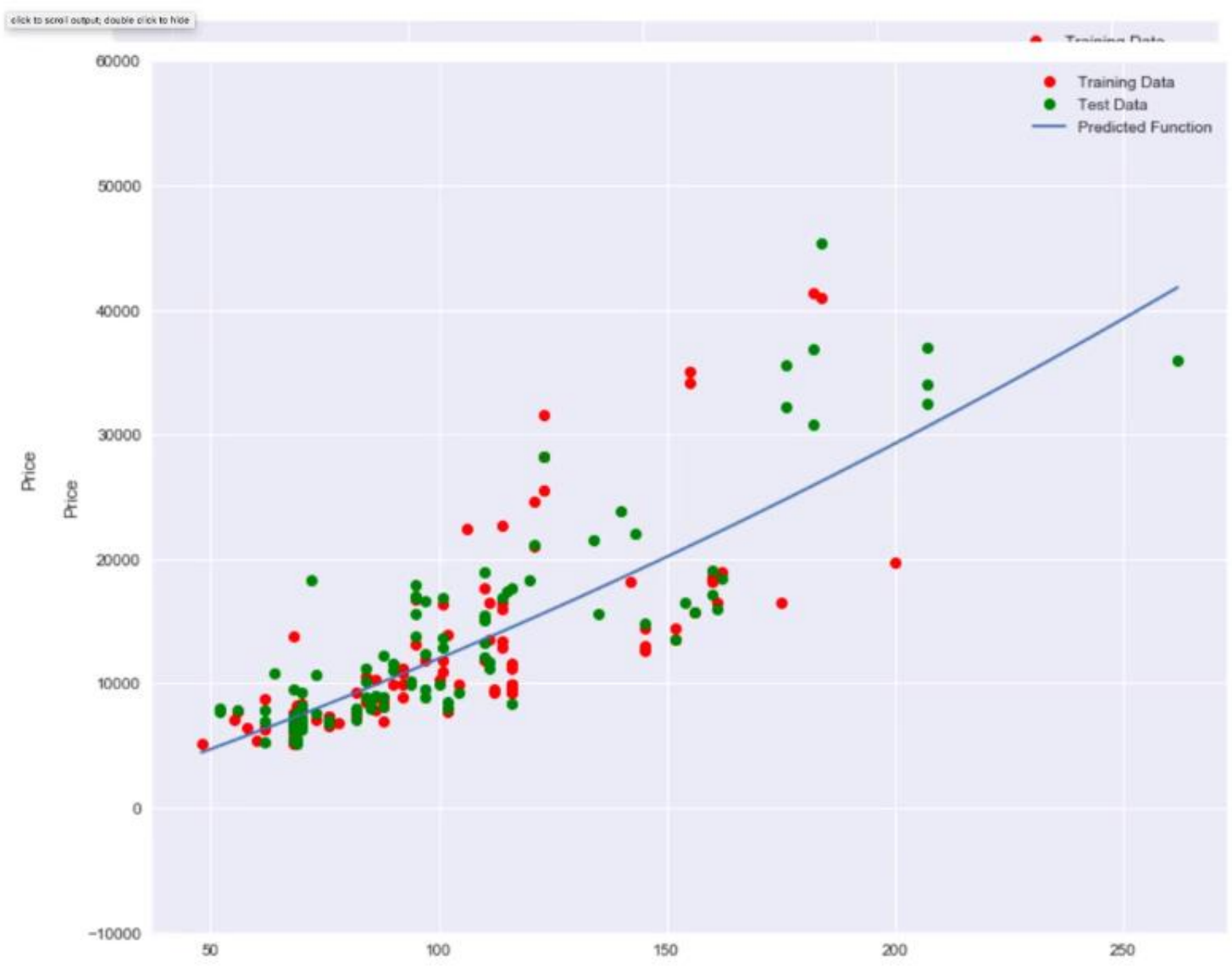
Model Selection



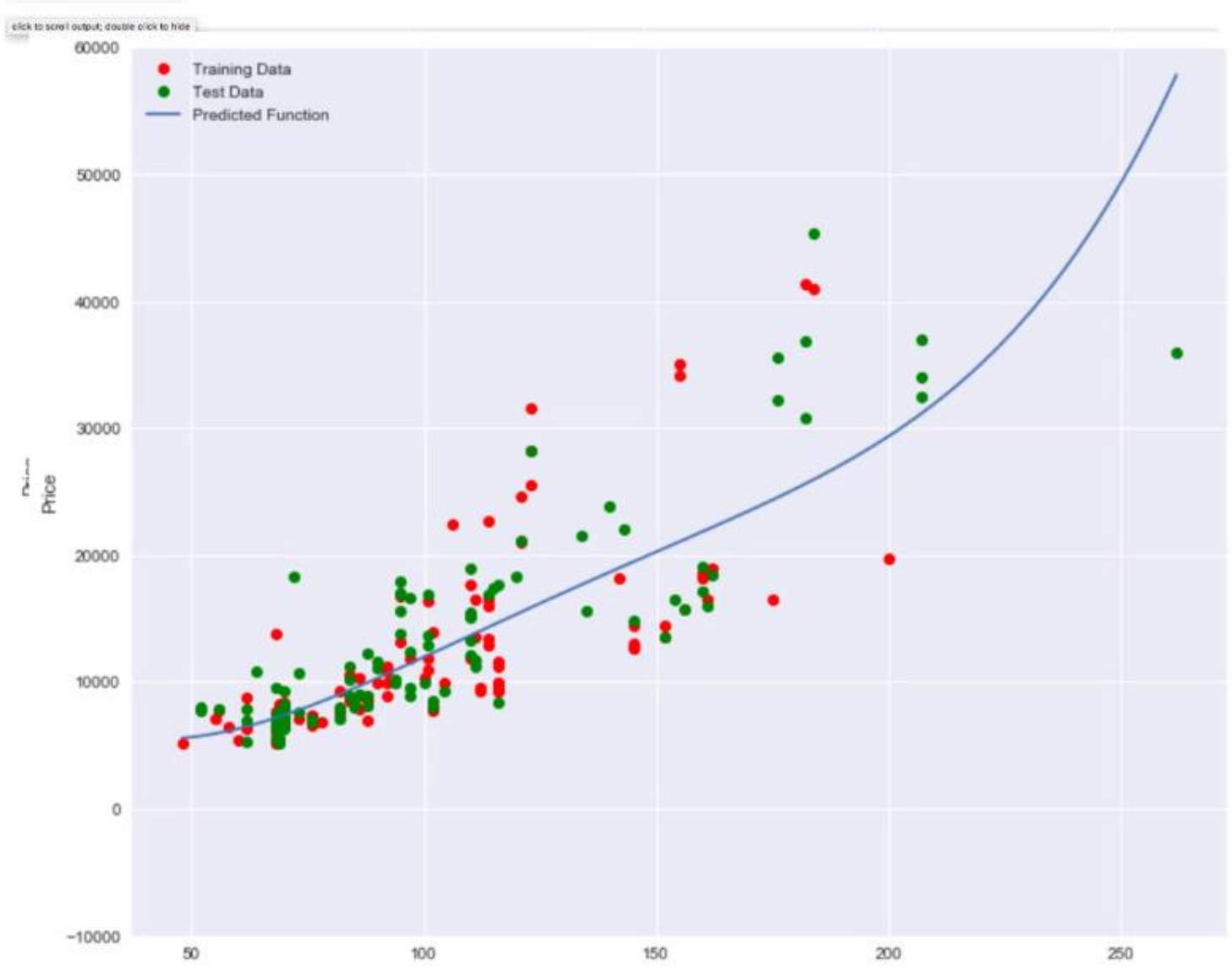
Model Selection



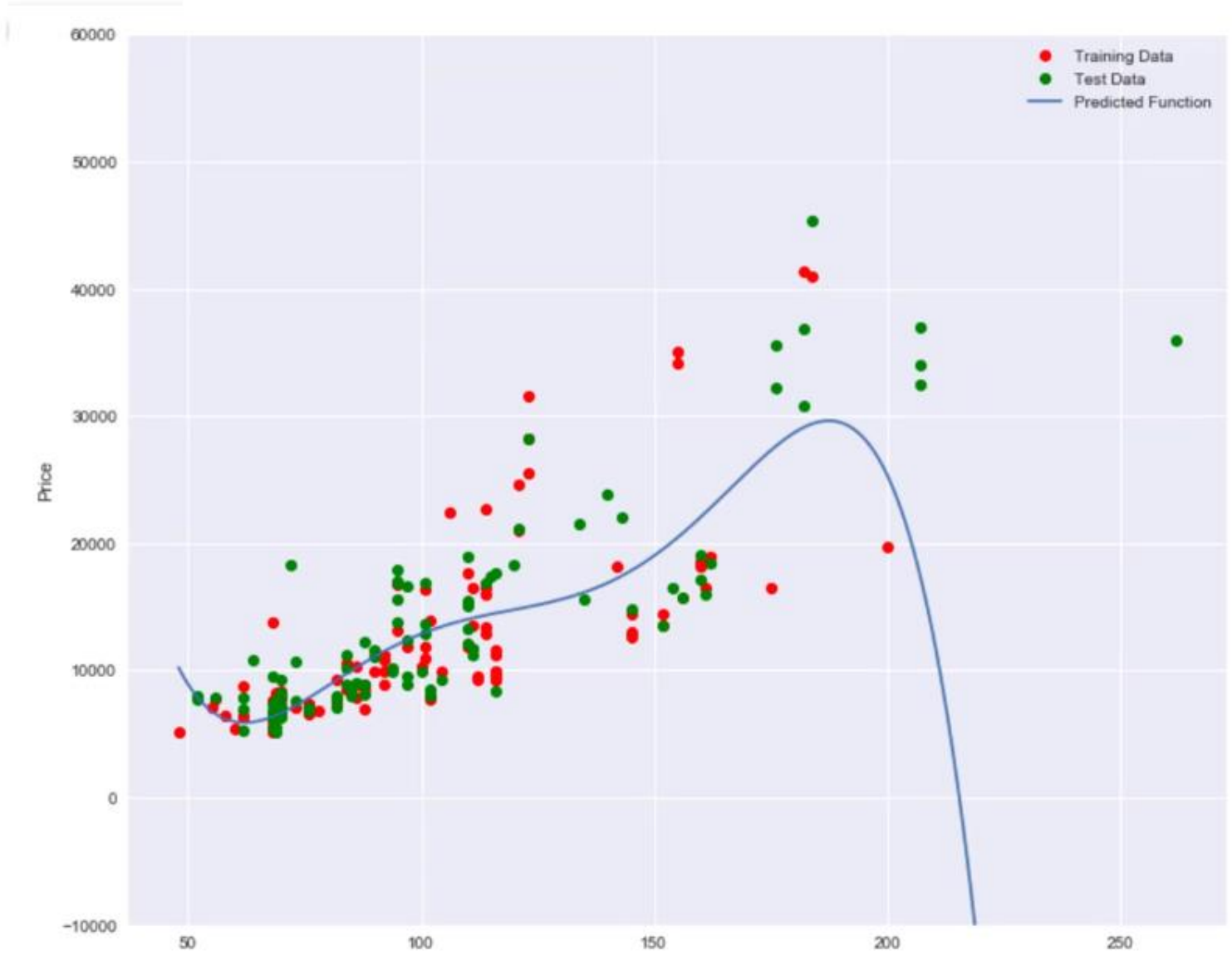
Model Selection



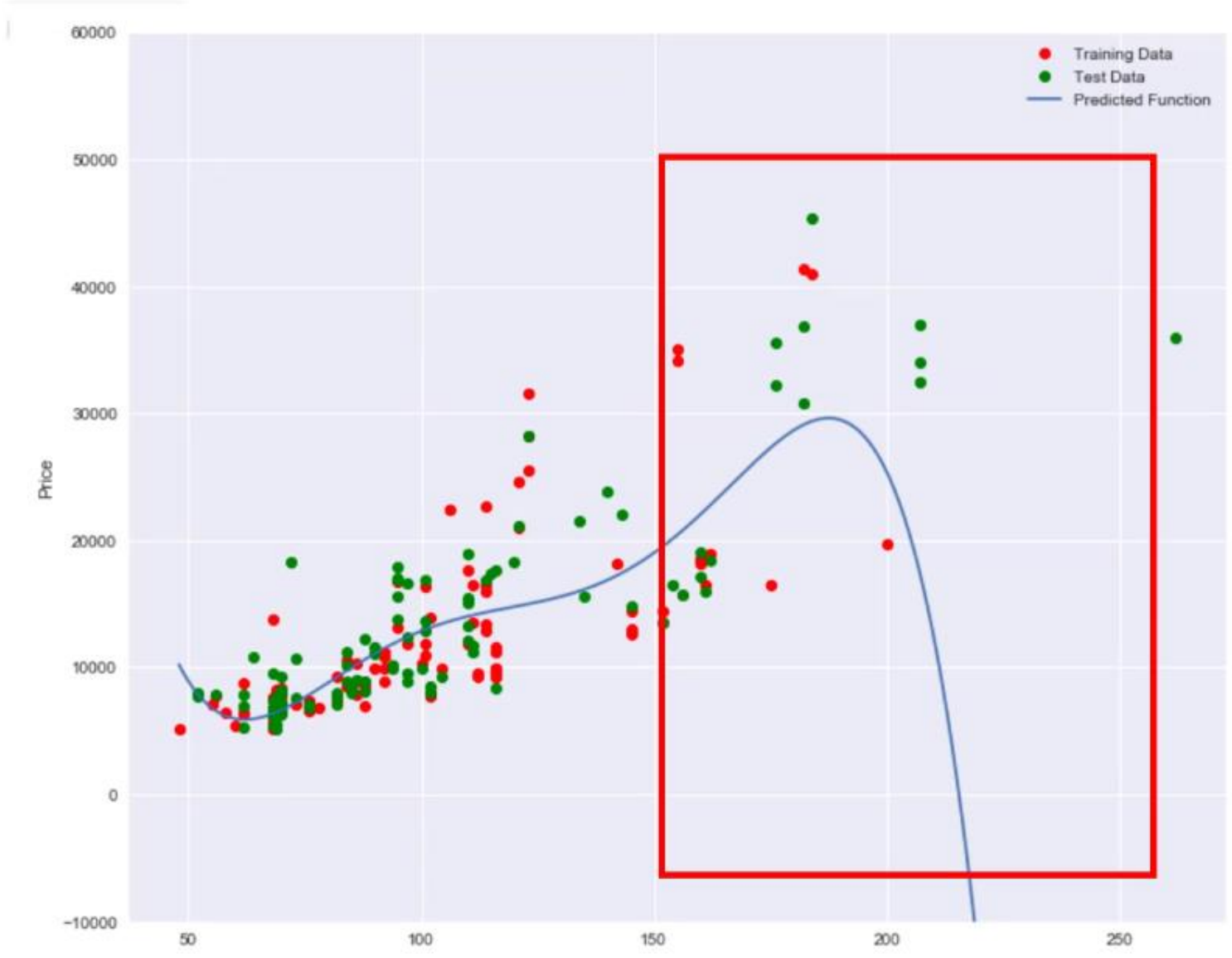
Model Selection



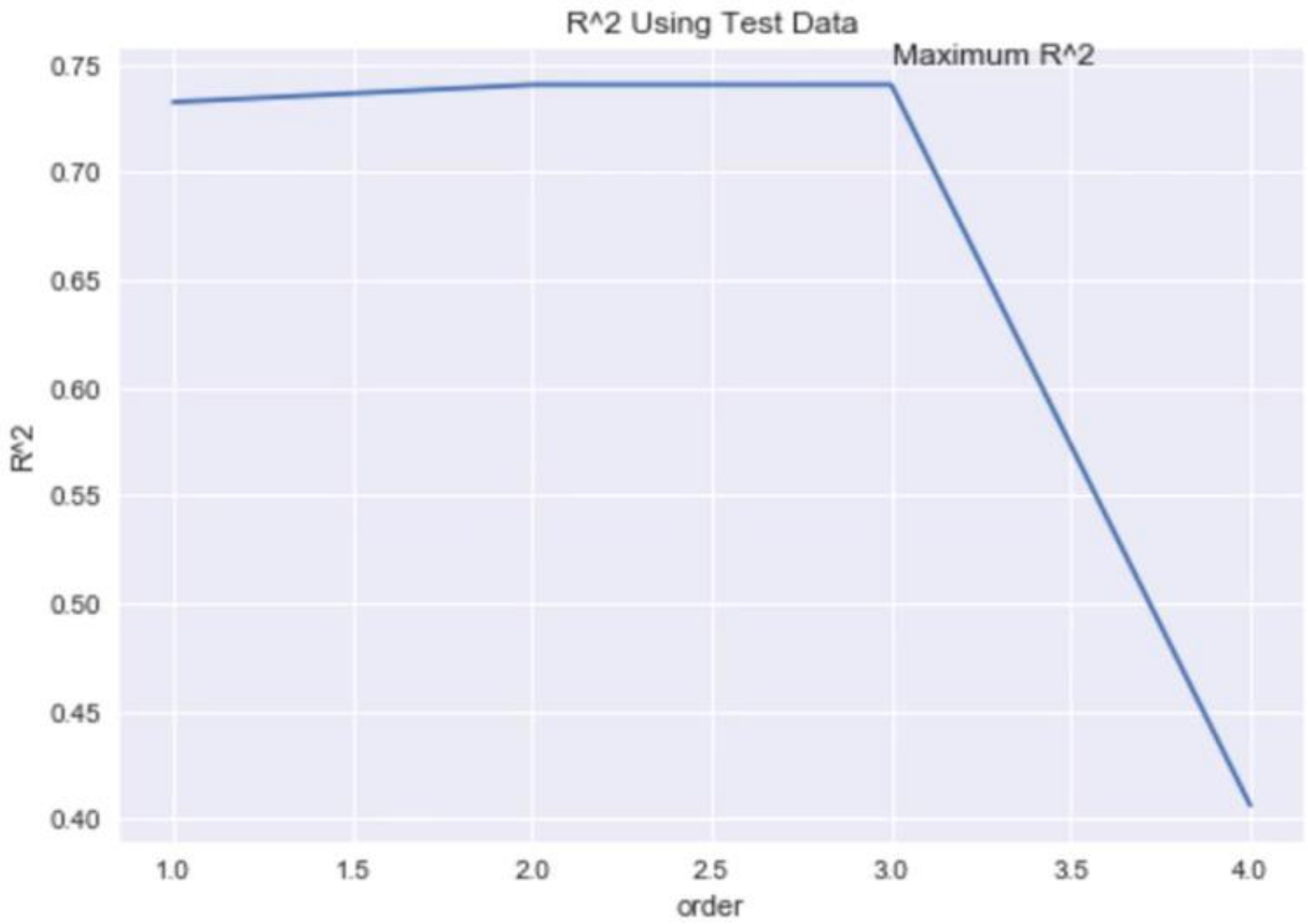
Model Selection



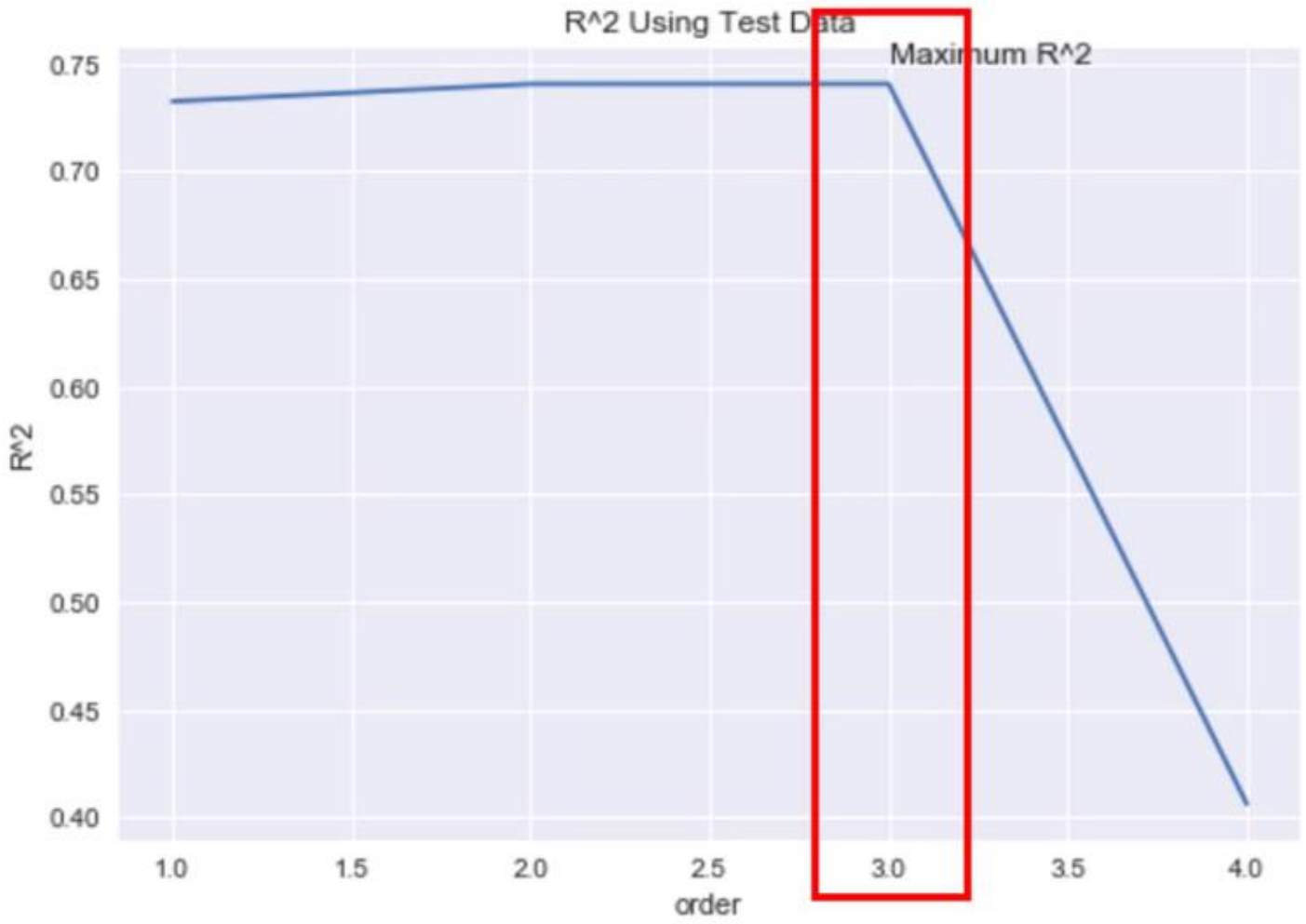
Model Selection



Model Selection



Model Selection



Model Selection

```
Rsqu_test=[]

order=[1,2,3,4]

for n in order:

    pr=PolynomialFeatures(degree=n)

    x_train_pr=pr.fit_transform(x_train[['horsepower']])

    x_test_pr=pr.fit_transform(x_test[['horsepower']])

    lr.fit(x_train_pr,y_train)

    Rsqu_test.append(lr.score(x_test_pr,y_test))
```

```
Rsqu_test=[]
order=[1,2,3,4]
for n in order:
    pr=PolynomialFeatures(degree=n)
    x_train_pr=pr.fit_transform(x_train[['horsepower']])
    x_test_pr=pr.fit_transform(x_test[['horsepower']])
    lr.fit(x_train_pr,y_train)
    Rsqu_test.append(lr.score(x_test_pr,y_test))
```

```
Rsqu_test=[]
order=[1,2,3,4]
for n in order:
    pr=PolynomialFeatures(degree=n)
    x_train_pr=pr.fit_transform(x_train[['horsepower']])
    x_test_pr=pr.fit_transform(x_test[['horsepower']])
    lr.fit(x_train_pr,y_train)
    Rsqu_test.append(lr.score(x_test_pr,y_test))
```

```
Rsqu_test=[]

order=[1,2,3,4]

for n in order:

    pr=PolynomialFeatures(degree=n)

    x_train_pr=pr.fit_transform(x_train[['horsepower']])

    x_test_pr=pr.fit_transform(x_test[['horsepower']])

    lr.fit(x_train_pr,y_train)

    Rsqu_test.append(lr.score(x_test_pr,y_test))
```

```
Rsqu_test=[]

order=[1,2,3,4]

for n in order:

    pr=PolynomialFeatures(degree=n)

    x_train_pr=pr.fit_transform(x_train[['horsepower']])

    x_test_pr=pr.fit_transform(x_test[['horsepower']])

    lr.fit(x_train_pr,y_train)

    Rsqu_test.append(lr.score(x_test_pr,y_test))
```

```
Rsqu_test=[]

order=[1,2,3,4]

for n in order:

    pr=PolynomialFeatures(degree=n)

    x_train_pr=pr.fit_transform(x_train[['horsepower']])

    x_test_pr=pr.fit_transform(x_test[['horsepower']])

    lr.fit(x_train_pr,y_train)

    Rsqu_test.append(lr.score(x_test_pr,y_test))
```

```
Rsqu_test=[]

order=[1,2,3,4]

for n in order:

    pr=PolynomialFeatures(degree=n)

    x_train_pr=pr.fit_transform(x_train[['horsepower']])

    x_test_pr=pr.fit_transform(x_test[['horsepower']])

    lr.fit(x_train_pr,y_train)

    Rsqu_test.append(lr.score(x_test_pr,y_test))
```

```
Rsqu_test=[]

order=[1,2,3,4]

for n in order:

    pr=PolynomialFeatures(degree=n)

    x_train_pr=pr.fit_transform(x_train[['horsepower']])

    x_test_pr=pr.fit_transform(x_test[['horsepower']])

    lr.fit(x_train_pr,y_train)

    Rsqu_test.append(lr.score(x_test_pr,y_test))
```

```
Rsqu_test=[]

order=[1,2,3,4]

for n in order:

    pr=PolynomialFeatures(degree=n)

    x_train_pr=pr.fit_transform(x_train[['horsepower']])

    x_test_pr=pr.fit_transform(x_test[['horsepower']])

    lr.fit(x_train_pr,y_train)

    Rsqu_test.append(lr.score(x_test_pr,y_test))
```