— Module 1

1. Scientific Computing
   - Pandas
   - Numpy
   - Scipy
2. Visualization libraries
   - Matplotlib
   - Seaborn (based on Matplotlib)
3. Algorithmic libraries.
   - Scikit-learn (Regression, classification, so on)
   - statsmodels

---

① Import and Export

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

⊙ Pandas Datatype

Pandas Type	Native Python Type	Description
object	string	numbers & strings
int64	int	Numeric characters
float64	float	Numeric characters with decimals
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	time data

⊙ Why check datatypes ?

- potential info and type mismatch
- compatibility with python methods

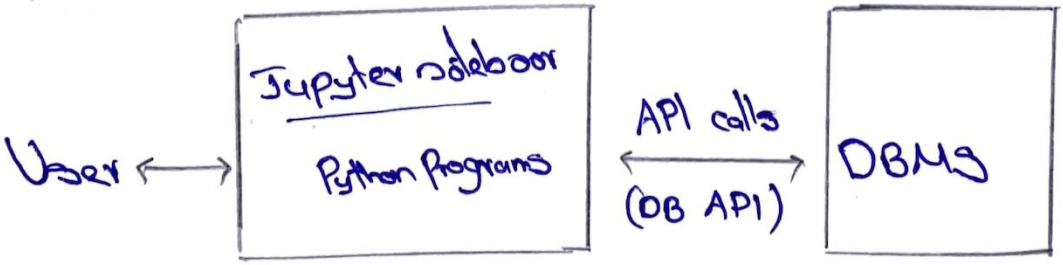
.df.dtypes (to check datatypes in pandas)

.df.describe() (for statistical summary

.df.describe(include="all") (full summary statistics)

.df.info() (to provide concise summary of the DataFrame)

⊙ Accessing databases



⊙ Concepts of the Python DB API

Connection Objects

- Database connections
- Manage transactions

Cursor Object

- Database Queries

⊙ Connection methods

- cursor()
- commit()
- rollback()
- close()

– Module 2

⊙ Data Preprocessing

– 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

Leave it as missing data

⊙ Drop missing values

Ex

```
df.dropna(subset=["Price"], axis=0, inplace=True)
```

axis=0 drops the entire row

axis=1 drops the entire column

⊙ Replace missing values

replace the ?, NaN with mean

Ex:

```
mean=df["normalized-losses"].mean()
df["normalized-losses"].replace(np.nan, mean)
```

⊙ Data Formatting

Non-formatted
- confusing
- hard to aggregate
- hard to compare

Formatted:
- more clear
- easy to aggregate
- easy to compare

City
N.Y.
NY
NY
New York

→

City
New York
New York
New York
New York

Example:

Convert "mpg" to "L/100km" in Car dataset.

city-mpg
21
21
19
...

→

city-L/100km
11.2
11.2
12.4
...

```
df["city-mpg"] = 235 / df["city-mpg"]
df.rename(columns={"city-mpg":"city-L/100km"},
          inplace=True)
```

To create a new column with value

```
df["city-L/100km"] = 235 / df["city-mpg"]
```

- Correcting data types
  - Identify data types:

    ```
    dataframe.dtypes()
    ```

  - Convert data types:

    ```
    dataframe.astype()
    ```

Imagine "price" of the column is object and you need to convert in int

```
df["Price"] = df["Price"].astype("int")
```

⊙ 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.

Methods of Normalizing data

① simple Feature scaling $x_{new} = \frac{x_{old}}{x_{max}}$

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

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

⊙ Binning

- Grouping of values to bins
- converts numeric into categorical variables.

Example:

If Price is an feature range from 5,000 to 45,500

Price: 5000, 10000, 12000, 12000, 30000, 31000, 39000, 44000, 44500

bins
low
Mid
High

⊙ Binning in Python Pandas

Price
13495
16500
18920
41315
5151
6295
...

→

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

```
bins = np.linspace(min(df["Price"]), max(df["Price"]),4)
group_names = ["Low", "Medium", "High"]
df["Price-binned"] = pd.cut(df["Price"], bins, labels=
                    group_names, include_lowest=True)
```

⊙ Visualized binned data

- Histograms

⊙ Categorical variables

Problem:

· Most statistical models cannot take in the objects/strings as input.

Car	Fuel	...
A	gas	...
B	diesel	...
C	gas	...
D	gas	...

⊙ Categorical → Numeric

Solutions:

· Add dummy variable for each unique category

· Assign 0 or 1 in each category

Car	Fuel	...	gas	diesel
A	gas	...	1	0
B	diesel	...	0	1
C	gas	...	1	0
D	gas	...	1	0

"One hot encoding"

⊙ Dummy variable in Python pandas

· Use Pandas.get_dummies() method

· Convert the categorical variables to dummy variable (0 or 1)

pd.get_dummies(df['fuel'])

fuel
gas
diesel
gas
gas

→

gas	diesel
1	0
0	1
1	0
1	0

How to turn Categorical Variable into Quantitative Variable in Python.

Module 3.

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
- Questions:

"What are the characteristics which have the most impact on the car price?"

Module Objectives:

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

Descriptive Statistics

- Giving short summaries about the sample and measure of the data.
- Summarize statistics using Pandas describe() method.

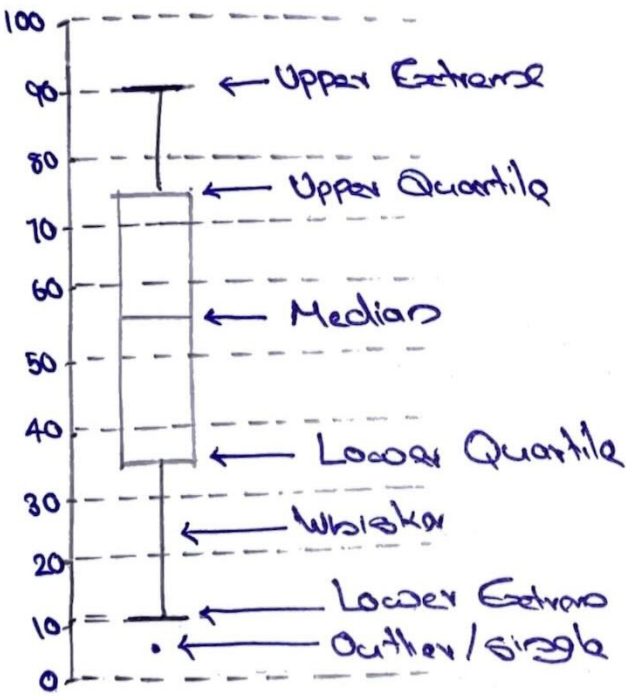
```
df.describe()
```

- Summarize the categorical data is by using the value_counts() method.

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

	value_counts
drive-wheels	
fwd	118
rwd	75
4wd	8

⊙ Descriptive Statistics: Box Plots



Example: Box plot

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

⊙ Descriptive statistics: Scatter Plot

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

Example: Scatterplot

```
y=df["Price"]
x=df["engine-size"]
plt.scatter(x,y)

plt.title("Scatterplot of Engine Size vs Price")
plt.xlabel("Engine Size")
plt.ylabel("Price")
```

⊙ GroupBy

- Pandas, dataframe.groupby() method:
  - can be applied to categorical variables
  - Group data into categories
  - Single or multiple variables

Example groupby:

```
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	rwd	convertible	23949.600000
8	rwd	hardtop	24202.714286
9	rwd	hatchback	14337.777778
10	rwd	sedan	21711.833333
11	rwd	wagon	16994.222222

⊙ Pandas method: Pivot()

. One variable is displayed along the columns, and the other variable is displayed along the rows

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

	Price				
body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels					
4wd	0.0	0.000000	7603.000000	12647.333333	9095.75000
fwd	11595.000000	8429.000000	8396.387755	9811.800000	0.000000
rwd	23949.6000000	24202.714286	14337.777778	21711.833333	16994.222222

⊙ Heatmap

. Plot target variable against multiple variables

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

⊙ Plot libraries:

1. matplotlib.

```
import matplotlib.pyplot as plt
```

or

```
from matplotlib import pyplot as plt
```

. %matplotlib inline is a magic command used to display Matplotlib plots directly inside the notebook instead of opening them in a separate window.

- matplotlib functions

a. Standard Line Plot

```
plt.plot(x, y)
```

b. Scatter plot

```
plt.scatter(x, y)
```

c. Histogram

```
plt.hist(x, bins)
```

d. Bar plot

```
plt.bar(x, height)
```

e. Pseudo Color Plot

```
plt.pcolor(C)
```

2. Seaborn

```
import seaborn as sns
```

a. Regression plot

```
sns.regplot(x='header-1', y='header-2',
            data=df)
```

b. Box and whisker plot

c. Residual plot

```
sns.residplot(data=df, x='header_1', y='header-2')
```

or

```
sns.residplot(x=df['header-1'], y=df['header-2'])
```

d. KDE plot

```
sns.kdeplot(x)
```

e. Distribution Plot

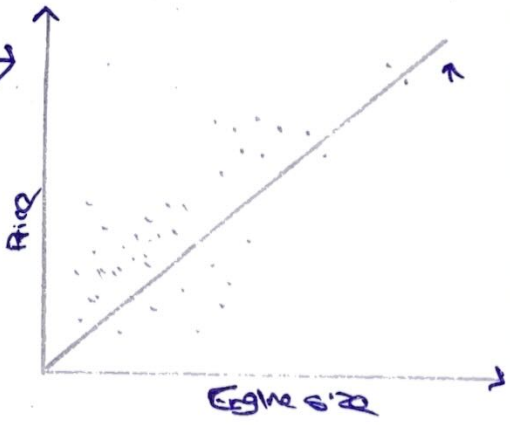
```
sns.distplot(x, hist=False)
```

⊙ Correlation

- Measure to what extent different variable are interdependent
- For example:
  - Lung cancer → Smoking
  - Rain → Umbrella
- Correlation doesn't imply causation

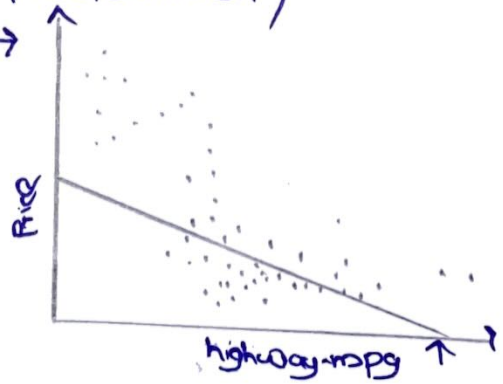
- Correlation: Positive linear relationship
  - Correlation between two features (engine-size & Price)

```
sns.regplot(x="engine-size", y="Price", data=df)
plt.ylim(0,)
```

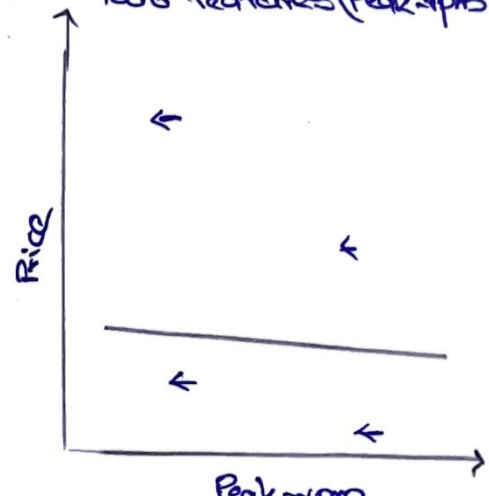


- Correlation: Negative linear relationship
  - Correlation between two features (highway-mpg & Price)

```
sns.regplot(x="highway-mpg", y="Price", data=df)
plt.ylim(0,)
```



- Correlation: Negative linear relationship
  - Weak correlation between two features (Peak-rpm & Price)




```
sns.regplot(x="peak-rpm", y="Price",
            data=df)
plt.ylim(0,)
```

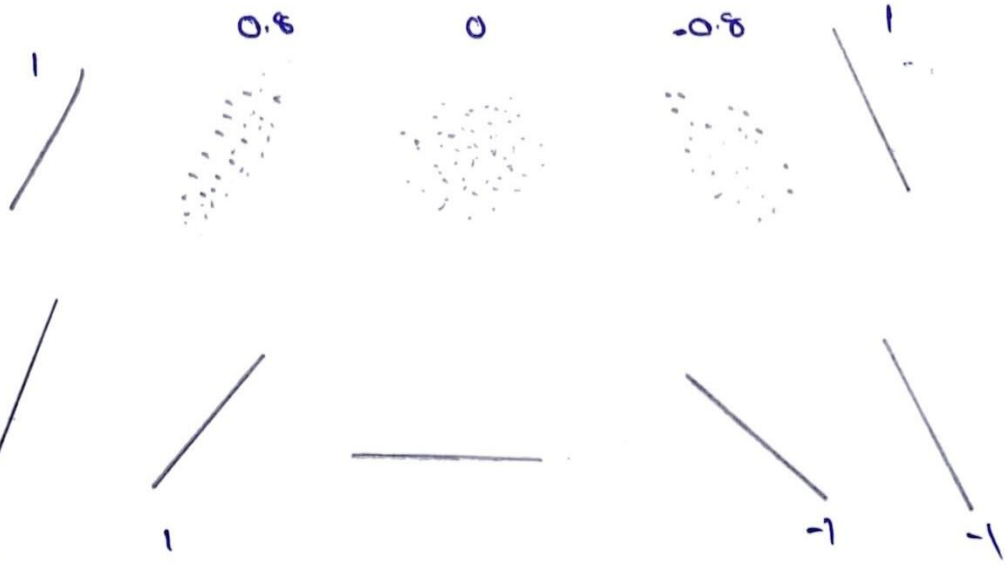
⊙ Correlation - Statistics

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

- Correlation coefficient: ~~P-value~~
  - 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



Example:

Pearson_coef, P_value = stats.pearsonr(df['horsepower'], df['price'])

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

⊙ Chi-Square Test for Categorical Variable

- It is a statistical method used to determine if there is a significant association between two categorical variables.
- It compares observed and expected frequencies to check if the variable are independent or associated
- The Test relies on the chi-square distribution and helps determine whether deviations occur randomly

- Two hypothesis
  - Null Hypothesis (H_0)

    Assumes that there is no association between the categorical variables, implying that any observed differences are due to random chance.
  - Alternative Hypothesis (H_1)

    Assumes that there is a significant association between the variables, indicating that the observed differences are not due to chance alone.

- Formula
  - chi-square statistic is calculated using the formula

$$X^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

$$X^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

where

O_i is the observed frequency for category i.

E_i is the expected frequency for category i,

calculated as;

$$E_i = \frac{(\text{row total} \times \text{column total})}{\text{grand total}}$$

- The sum is taken over all cells in the contingency table
- The calculate the chi-square statistic is then compared to a critical value from the chi-square distribution table. This table provides critical values for different degrees of freedom (df) & significance levels (α).
- The degree of freedom for the test are calculated as

$$df = (r-1) \times (c-1)$$

where r is the number of rows and c is the number of columns in the table

- Applications

1. Market Research
2. Healthcare
3. Social science
4. Education
5. Quality control

Example - Weak Correlation (Practical)

Category	Like	Dislike	Total
Male	20	30	50
Female	25	25	50
Total	45	55	100

Step 1 · Calculate Excepted Frequencies

$$E_{Male,Like} = \frac{(50 \times 45)}{100} = 22.5$$

$$E_{Male,Dislike} = \frac{(50 \times 55)}{100} = 27.5$$

$$E_{Female,Like} = \frac{(50 \times 45)}{100} = 22.5$$

$$E_{Female,Dislke} = \frac{(50 \times 55)}{100} = 27.5$$

Step 2: Compute Chi-square statistic

$$X^2 = \frac{(20-22.5)^2}{22.5} + \frac{(30-27.5)^2}{27.5} + \frac{(25-22.5)^2}{22.5} + \frac{(25-27.5)^2}{27.5}$$

$$= \frac{(2.5)^2}{22.5} + \frac{(2.5)^2}{27.5} + \frac{(2.5)^2}{22.5} + \frac{(2.5)^2}{27.5}$$

$$= \frac{6.25}{22.5} + \frac{6.25}{27.5} + \frac{6.25}{22.5} + \frac{6.25}{27.5}$$

$$x^2 = 0.277 + 0.227 + 0.277 + 0.227$$

$$x^2 = \underline{1.008}$$

Step 3: Determine Degree of Freedom

$$df = (2-1) \times (2-1) = 1$$

Step 4: Interpret the result

- Critical value: For df = 1 at $\alpha = 0.05$ is approximately 3.841.
- Comparison: $1.008 < 3.841$
- Conclusion: Fail to reject the null hypothesis
  → No significant association between gender and product preference

Practical Example - Strong Association

Category	Disease	No Disease	Total
Smoker	50	30	80
Non-Smoker	20	100	120
Total	70	130	200

Step 1: Calculate Expected frequencies

Using the formula for expected frequencies

$$E_{Smoker, Disease} = \frac{(80 \times 70)}{200} = 28$$

$$E_{Smoker, No\ Disease} = \frac{(80 \times 130)}{200} = 52$$

$$E_{Non\text{-}Smoker, Disease} = \frac{(120 \times 70)}{200} = 42$$

$$E_{Non\text{-}smoker, No\ Disease} = \frac{(120 \times 130)}{200} = 78$$

- step 2: Computer Chi-Square Statistic

$$X^2 = \frac{(50-28)^2}{28} + \frac{(30-52)^2}{52} + \frac{(20-42)^2}{42} + \frac{(100-78)^2}{78}$$

$$= \frac{(22)^2}{28} + \frac{(22)^2}{52} + \frac{(22)^2}{42} + \frac{(22)^2}{78}$$

$$= \frac{484}{28} + \frac{484}{52} + \frac{484}{42} + \frac{484}{78}$$

$$= 17.29 + 9.31 + 11.52 + 6.21$$

$$x^2 = 44.33$$

- step 3: Determine Degree of Freedom

$$df = (2-1) \times (2-1) = 1$$

- step 4: Interpretation

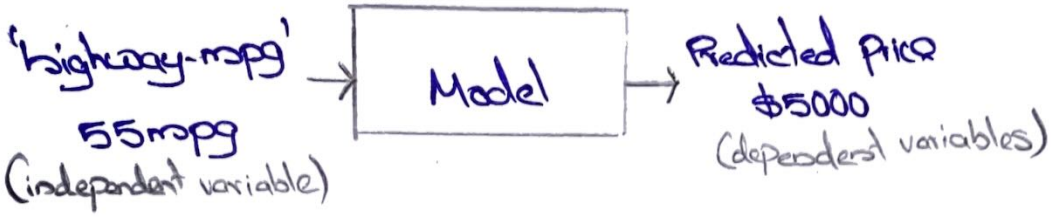
- Critical Value: For df=1 at a significance level of 0.05 the critical chi-square value is approx 3.841

- Since $44.33 > 3.841$, we reject the null hypothesis

Module 4

⊙ Model Development

- Relating one or more independent variables to dependent variables

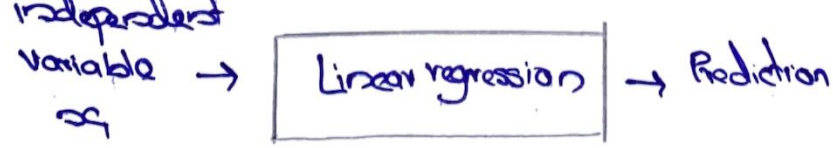


- more relevant data → more accurate model

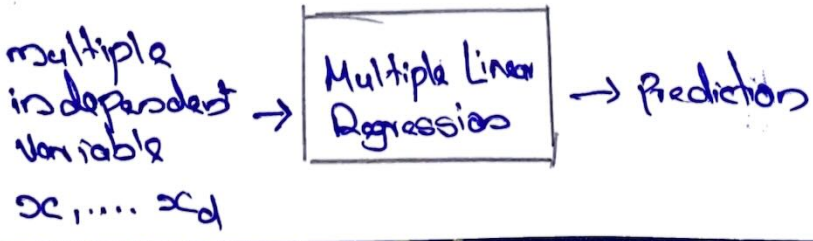
⊙ In this module

1. Simple Linear Regression
2. Multiple Linear Regression
3. Polynomial Regression

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



- Multiple linear regression will refer to multiple independent variable to make a Prediction



⊙ 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

Example:

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

find the parameters b_0 and b_1

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

We can obtain a prediction

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

Yhat	X
2	5
:	
3	4

- intercept (b_0): lm.intercept_ (38423.305858)
- slope (b_1): lm.coef_ (-821.73337832)

The relationship between Price and Highway MPG is given by:

Price = 38423.31 - 821.73 * highway-mpg

$\hat{y} = b_0 + b_1 x$

⊙ Multiple Linear Regression (MLR)

This method is used to explain the relationship between:

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

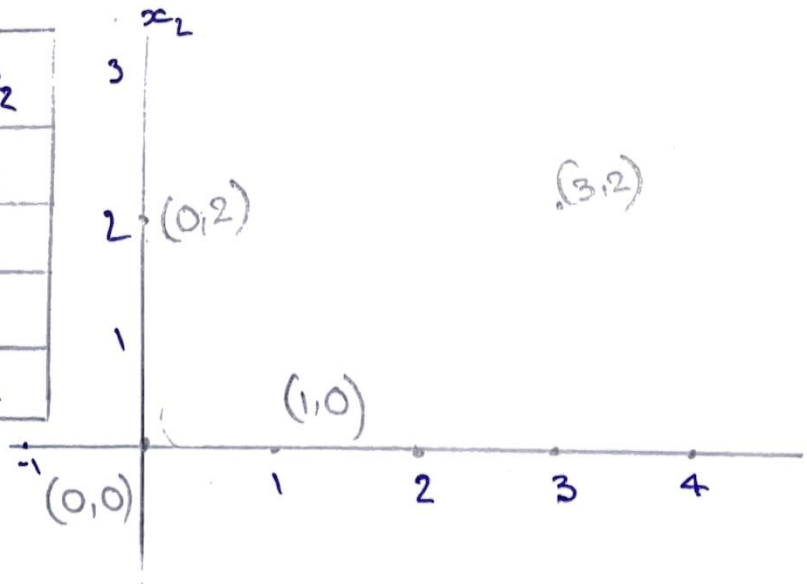
$$\hat{y} = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + b_4 x_4$$

- 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...

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

x_1 & x_2 can be visualized on a 2D plane

n	x_1	x_2
1	0	0
2	0	2
3	1	0
4	3	2

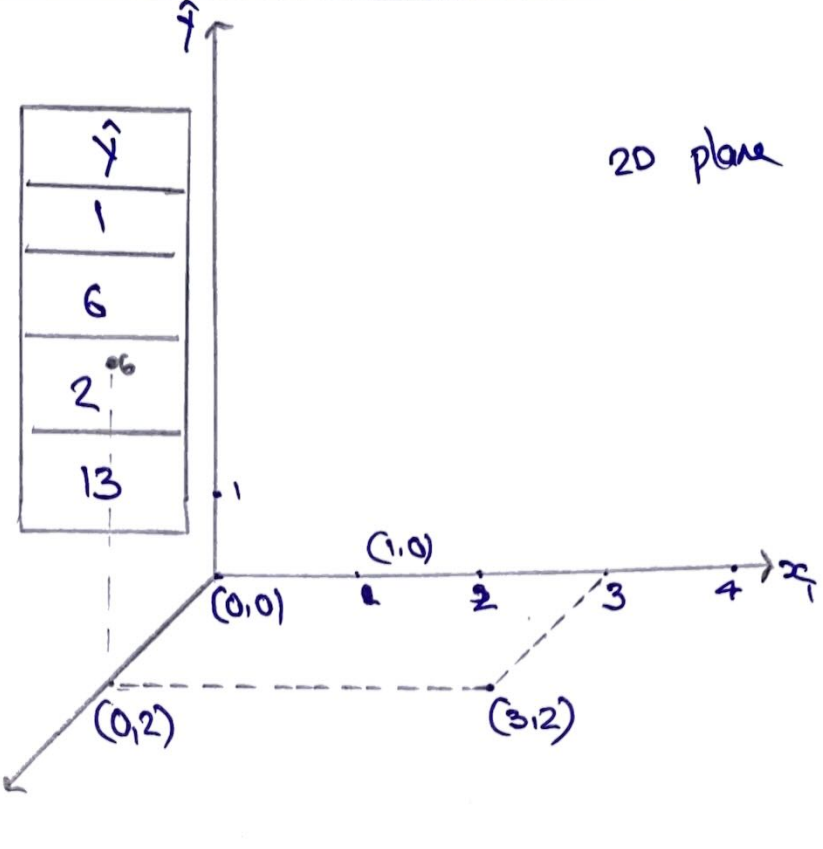


- This is shown below where

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

n	x_1	x_2
1	0	0
2	0	2
3	1	0
4	3	2

$\hat{y}$
1
6
2
13



2D plane

1. extract 4 predictor variable & 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

Yhat
2
:
3

1. Intercept (b_0)

```
lm.intercept_
-15678.742628061467
```

2. 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

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

---

① Regression plot

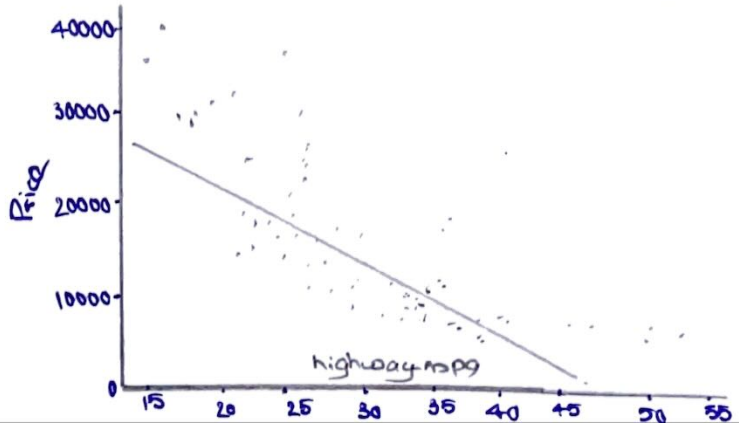
It gives us a good estimate of:

- The relationship between two variables
- The strength of the correlation
- The direction of the relationship (positive or negative)

The regression plot shows us a combination of:

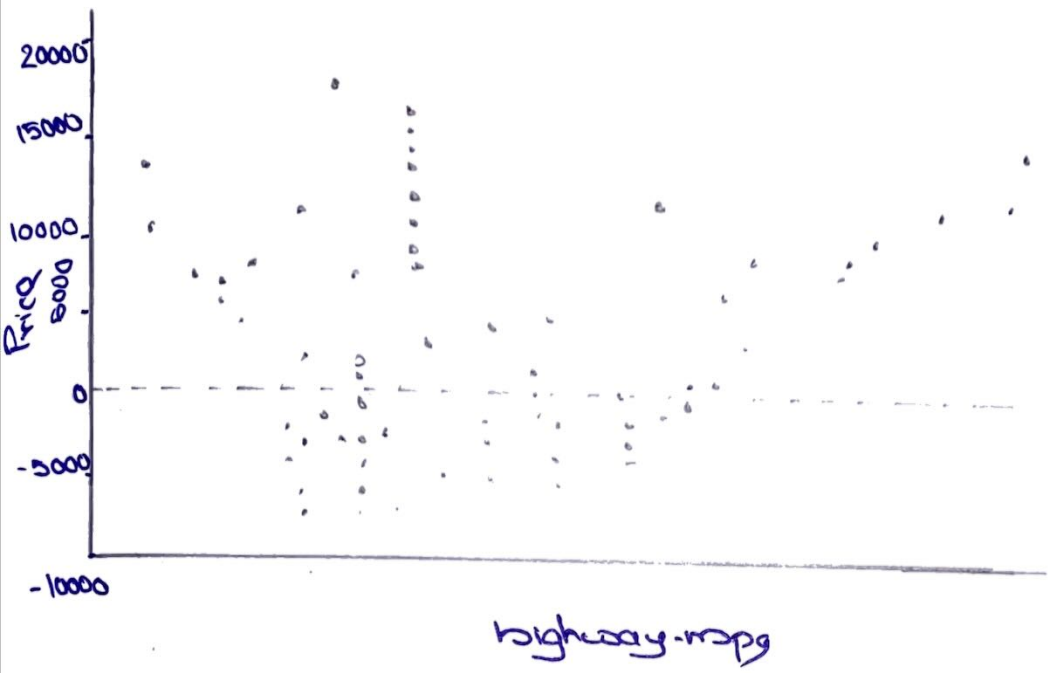
- The scatterplot: where each point represents a different
- The fitted linear regression line ($\hat{y}$)

```
import seaborn as sns

sns.regplot(x="highway-mpg", y="Price", data=df)
plt.ylim(0,)
```

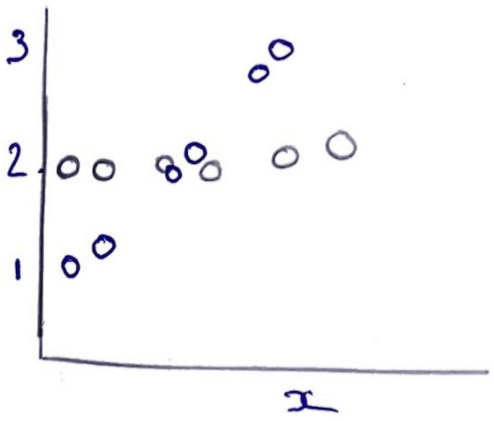


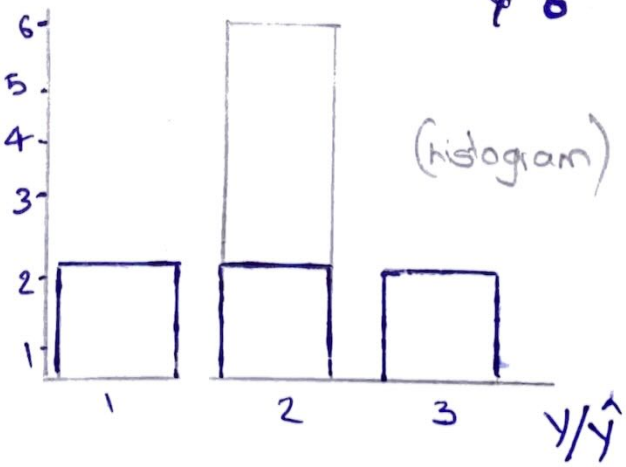
⊙ Residual plot

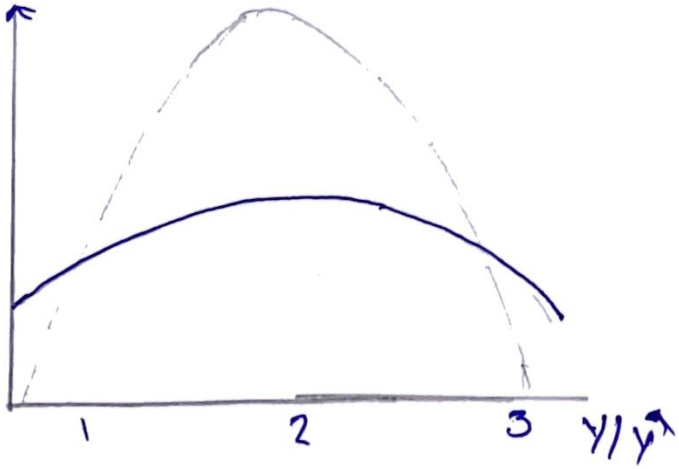
```
import seaborn as sns
sns.residplot(df['highway-mpg'], df['Price'])
```



⊙ Distribution plots







Compare the distribution plots

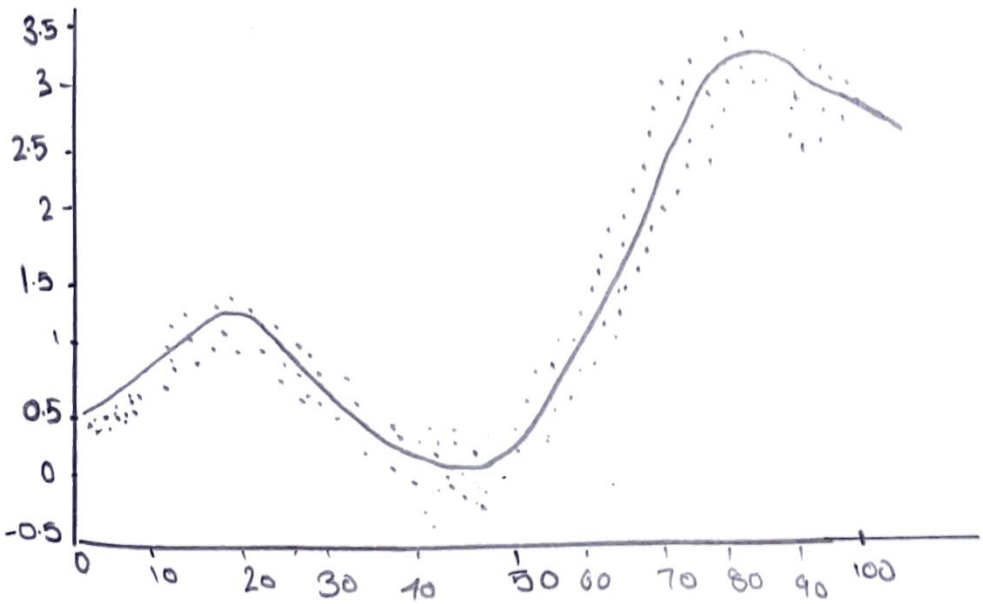
- The fitted values that result from the model
- The actual values

```
import seaborn as sns

ax1 = sns.distplot(df['Price'], hist=False, color="r",
                   label="Actual Value")
sns.distplot(Yhat, hist=False, color="b", label="Fitted
             Values", ax=ax1)
```

⊙ Polynomial Regression

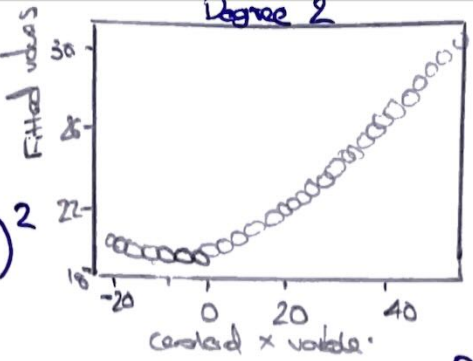
- A special case of the general linear regression model
- Useful for describing curvilinear relationships

- Curvilinear relationships:
  - By squaring or setting higher-order terms of the predictor variables



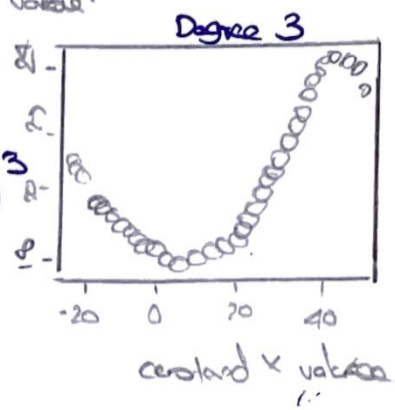
- Quadratic - 2nd order

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



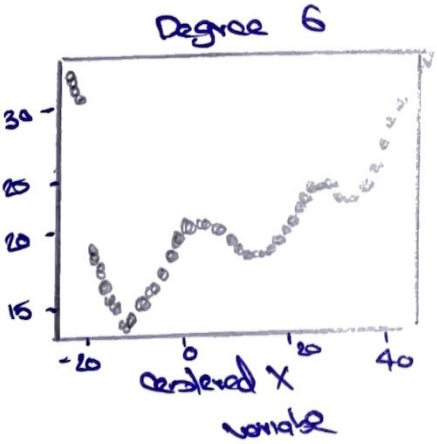
- Cubic - 3rd order

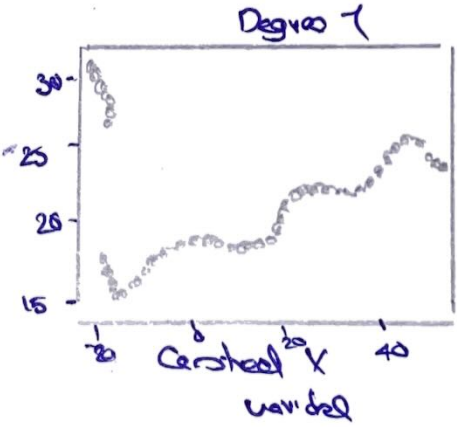
$$\hat{y} = b_0 + b_1 x_1 + b_2 (x_1)^2 + b_3 (x_1)^3$$



- Higher order

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





1. Calculate Polynomial of 3rd order

```
f = np.polyfit(x, y, 3)
p = np.poly1d(f)
```

2. We can print out the model

```
Print(p)
```

$$-1.557 (x_1)^3 + 204.8 (x_1)^2 + 8965 x + 1.37 \times 10^5$$

- Polynomial regression with more than one dimension

  - We can also have multi dimensional polynomial linear regression

$$\hat{y} = b0 + b1X_1 + b2X_2 + b3X_1X_2 + b4(X_1)^2 + b5(X_2)^2 + \dots$$

- The "Preprocessing" library in scikit-learn

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

- Example

```
Pr = PolynomialFeatures(degree=2)
Pr.fit_transform([1,2], include_bias=False
```

X_1	X_2
1	2

↓

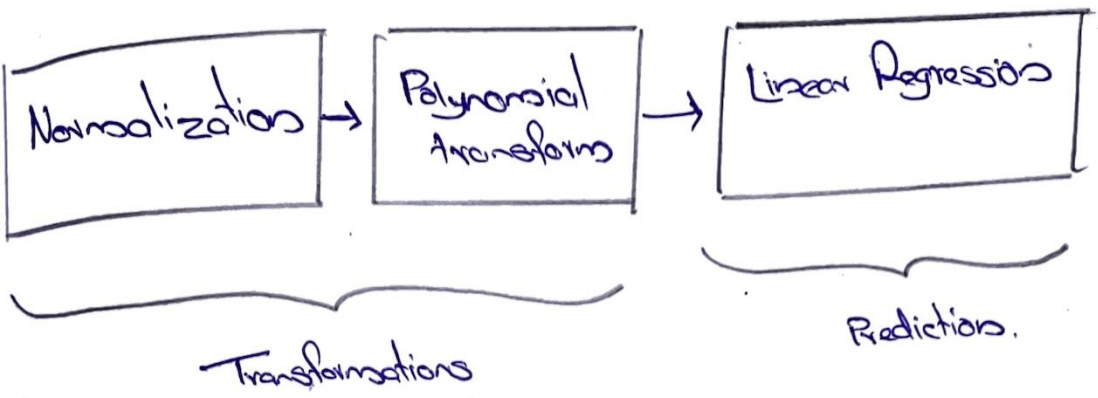
X_1	X_2	X_1X_2	X_1^2	X_2^2
1	2	(1)2	1	$(2)^2$
1	2	2	1	4

- 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




```
from sklearn.PreProcessing import polynomialFeatures
from sklearn.linear_model import Linear Regression
from sklearn.Preprocessing import standardScaler
from sklearn.pipeline import Pipeline
```

⊙ Pipeline constructor

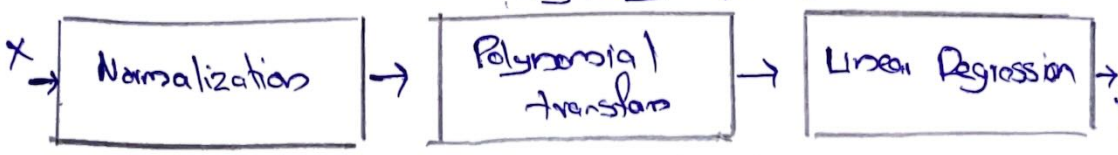
```
Input =[('polynomial', PolynomialFeature(degree=2)),
        ('scale', StandScaler()), ....
('Model', LinearRegression())]
```

. Pipeline constructor

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

. We can train the pipeline object

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



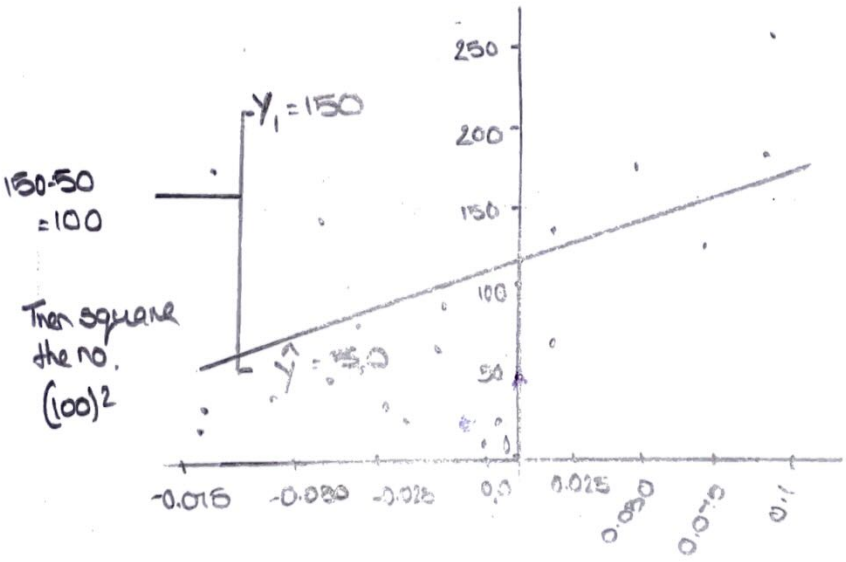
⊙ Measures for In-Sample Evaluation

- A way to numerically determine how good the model fits on dataset.

- Two important measures to determine the fit of a model:
  - Mean Squared Error (MSE)
  - R-squared (R^2)

1. Mean Squared Error (MSE)

- For example for sample 1:



$$\frac{\square + \square + \square + \square}{\text{Number of Samples}}$$

• In Python we can measure the MSE as follows

```
from sklearn.metrics import mean_squared_error
mean_squared_error(df['Price'], Y_predict_simple_fit)
```

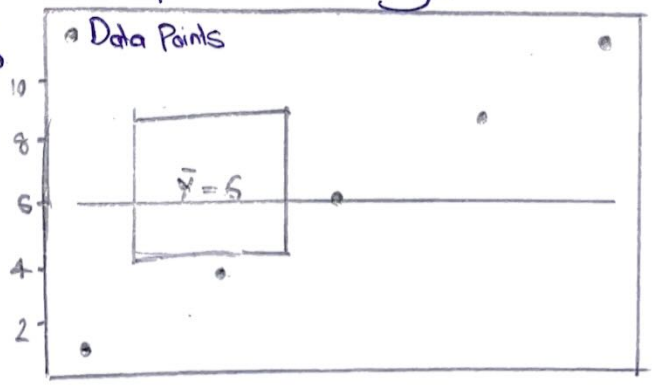
```
3163502.944639888
```

⊙ R-squared / R^2

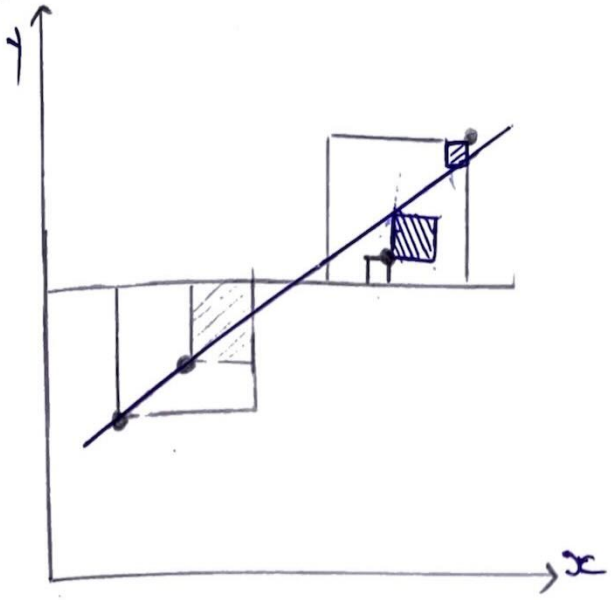
- The coefficient of Determination or R squared (R^2)
- Is a measure to determine how close the data is to the fitted regression line
- R^2: the percentage of variation of the target variable (y) that is explained by the linear model.
- Think about as comparing a regression model to a simple model
  i.e. the mean of the data points

• Coefficient of Determination (R^2)

- In this example the average of the data points $\bar{y}$ is 6



$$R^2 = \left(1 - \frac{\text{MSE of regression line}}{\text{MSE of the average of the data}}\right)$$



↙ figure. 1

- The blue line represents the regression line
- The blue squares represents the MSE of the regression line
- The Grey line represents the average value of the data points
- The Grey square represent the MSE of Grey line
- We see the area of the blue square is much smaller than the area of the Grey square

- In this case ratio of the areas of MSE is close to zero

$$\frac{\text{MSE of regression line}}{\text{MSE of } \bar{y}} = \frac{\square + \square}{\square + \square} = 0$$

from figure 1

$$R^2 = \left(1 - \frac{\text{MSE of regression line}}{\text{MSE of } \bar{y}}\right)$$

$$= (1-0)$$

$$= 1$$

⊙ R-squared / R^2

- Generally the values of the MSE are between 0 & 1.
- We can calculate the R^2 as follows

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

lm.fit(X, Y)

lm.score(x, y)
0.496591188
```

⊙ Prediction and Decision Making

- Decision making: Determining a good model fit

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

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

- 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

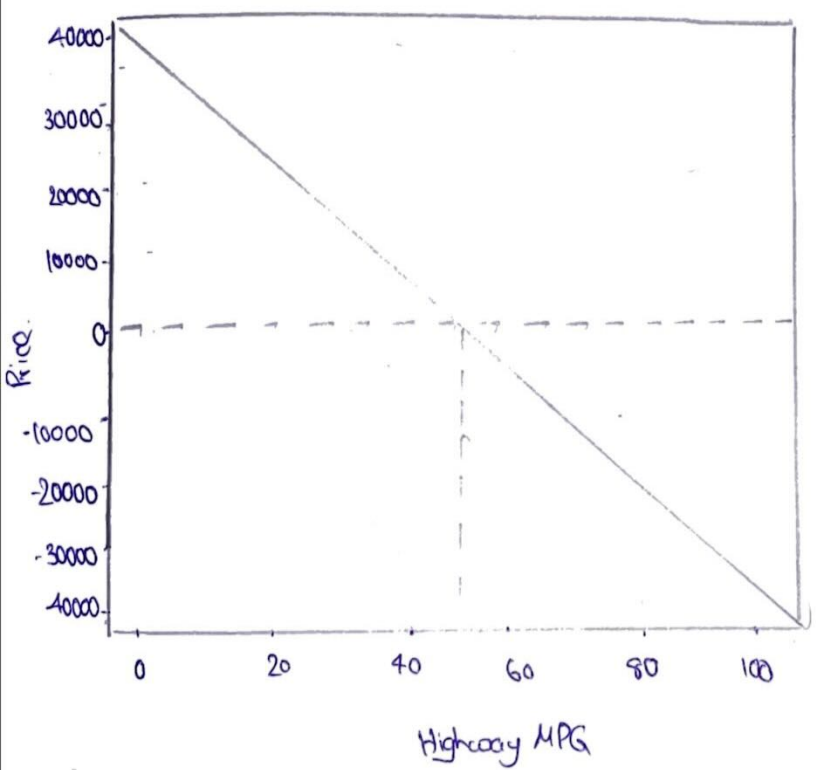
  ```
  lm.predict(np.array(30.0).reshape(-1,1))
  ```

  . Result: $ 13771.30

  lm.coef_

  ```
  -821.73337832
  ```

  . Price = 38423.31 - 821.73 * highway-mpg



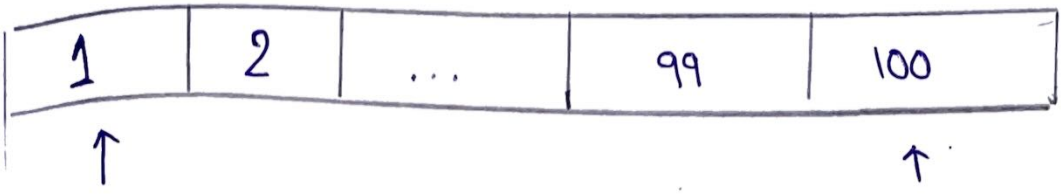
. First, we import numpy

```
import numpy as np
```

. We use the numpy function arange to generate a sequence from 1 to 100

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

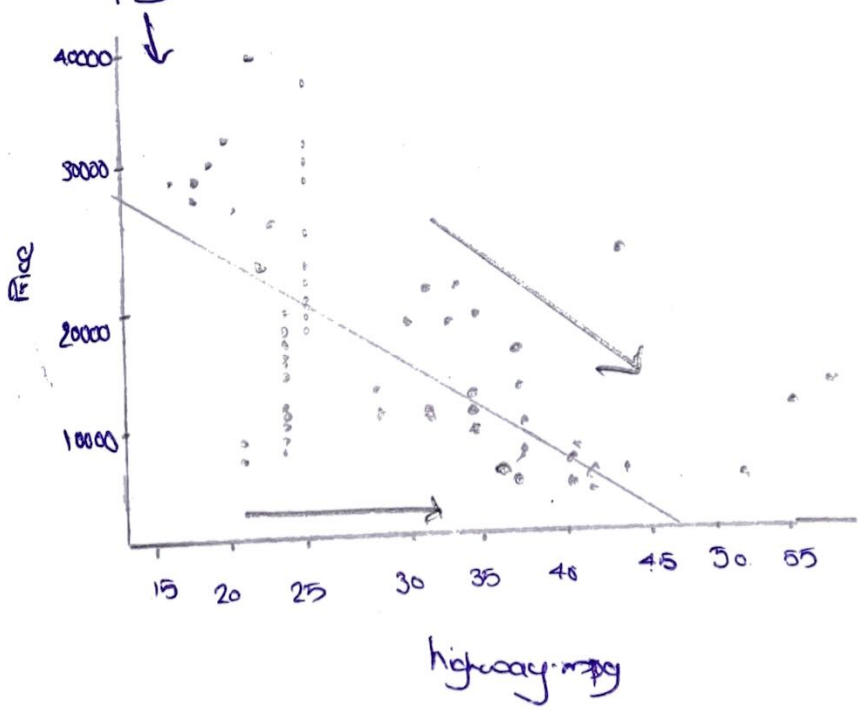
```
neco_input = np.arange(1, 101, 1).reshape(-1, 1)
```

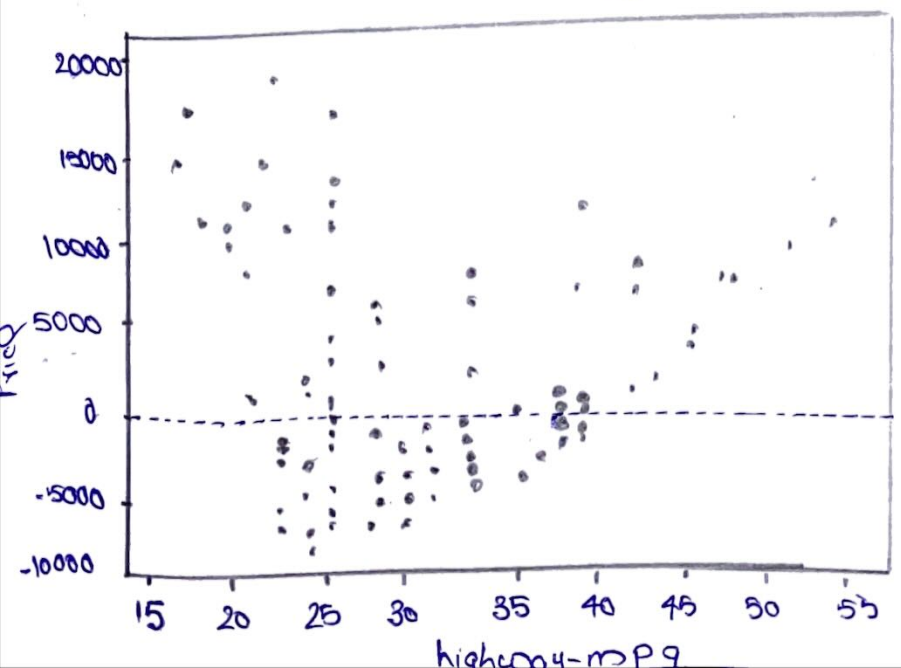


- We can Predict neo values

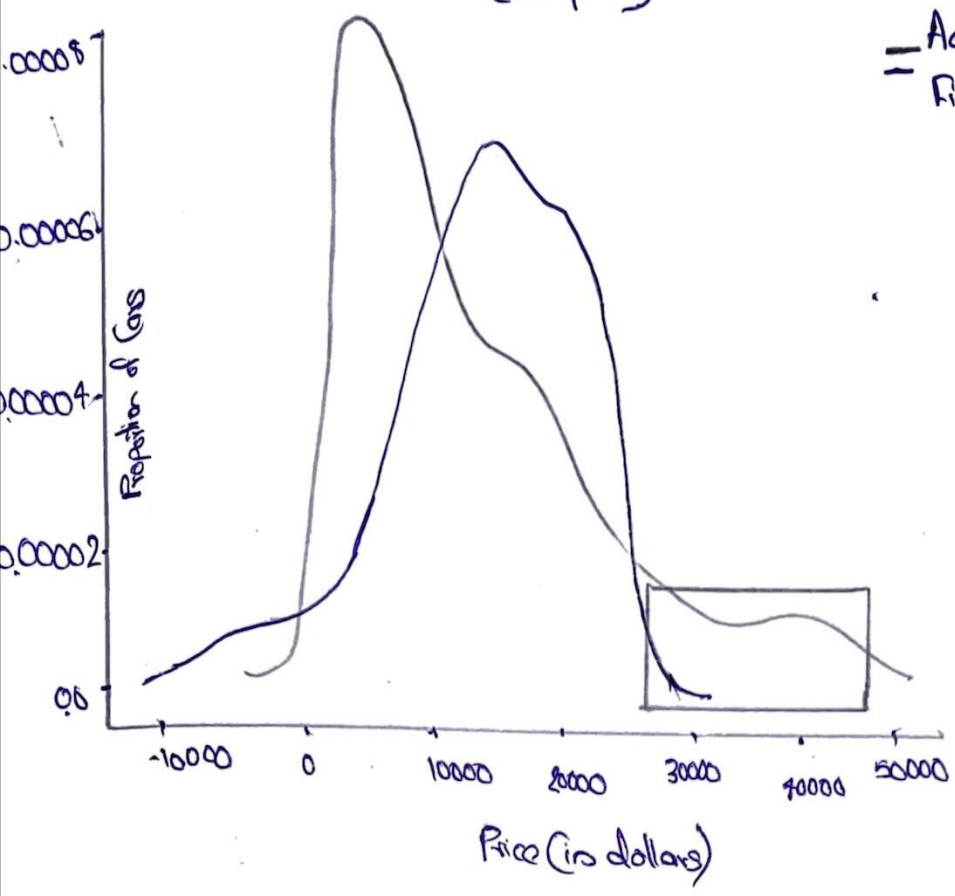
```
Yhat = lm.Predict(neco_input)
```

- Visualization
  - Simply visualizing your data with a regression

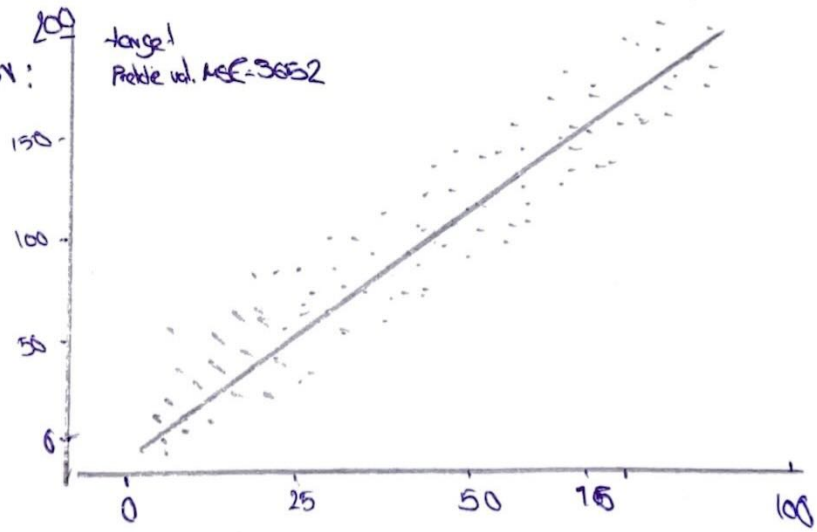




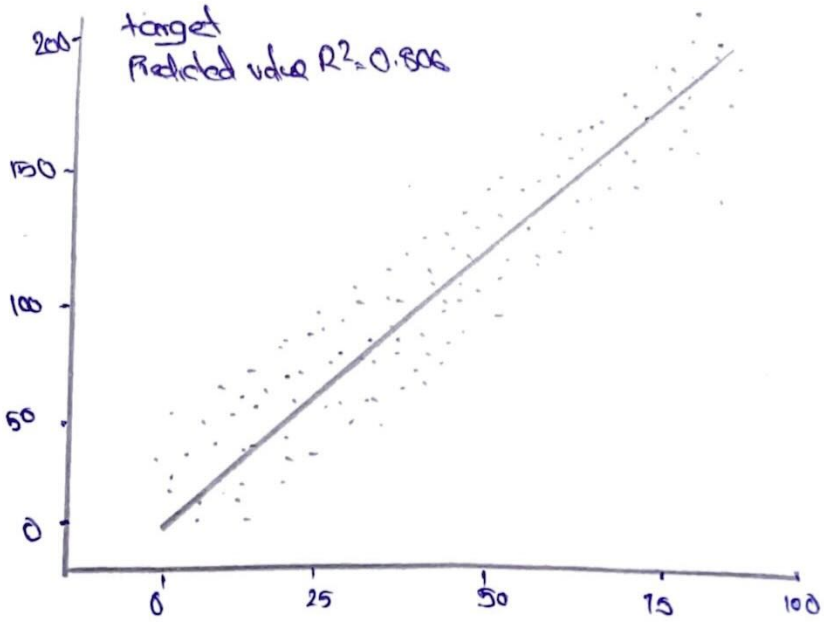
⊙ Visualization [dist plot]



• Mean Square Error:



• R-square :



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 than regular regression

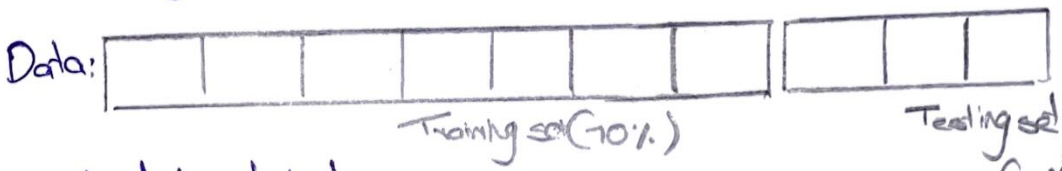
4. A similar inverse relationship holds for R^2

Module 5

Model Evaluation.

- In-sample evaluation tells us how well our model will fit the data used to train it
- Problem?
  - It does not tell us how well the trained model can be used to predict new data
- Solution?
  - In-sample data or training data
  - Out-of-sample evaluation or test set

Training/Testing sets



- Split dataset into:
  - Training set (70%)
  - Testing set (30%)
- Build and Train the model with a training set
- Use testing set to assess the performance of a predictive model
- When we have complete testing our model we should use all the data to train the model to get the best performance

- Function training_test_split()

  .Split data into random train and test subsets

  from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.3, random_state=0)

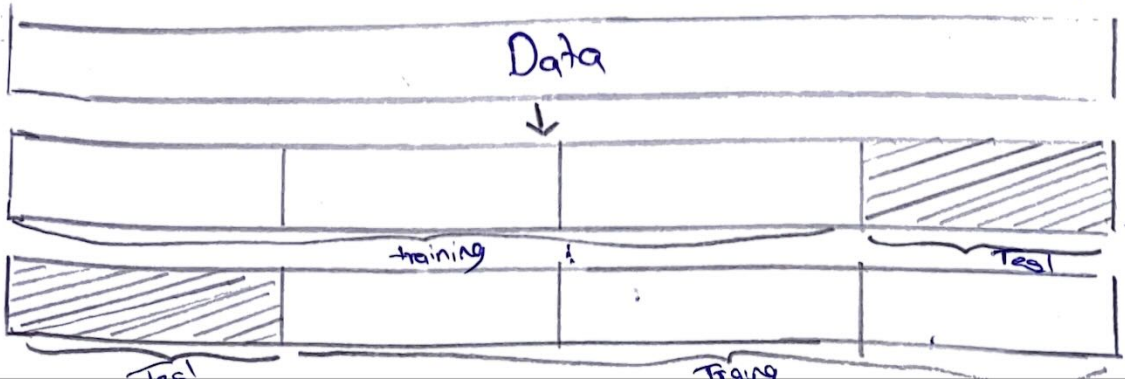
- x_data: features or independent variables
- y_data: dataset target: df['Price']
- x_train, y_train: parts of available data as training set
- x_test, y_test: Parts of available data as testing set
- test_size: percentage of the data for testing (here 30%)
- random_state: number generator used for random sampling

- Generalization Performance

  - Generalization error is a measure of how well our data does at predicting previously unseen data.
  - The error we obtain using our testing data is an approximation of this error

- Cross Validation

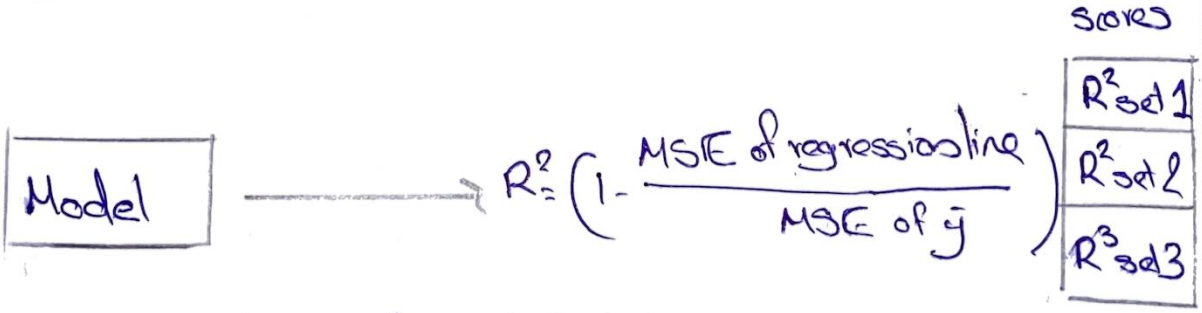
  - Most common out-of-sample evaluation metrics
  - More effective use of data (each observation is used for both training & testing)



- Function cross_val_score()

```
from sklearn.model_selection import cross_val_score

score = cross_val_score(lr, x_data, y_data, cv = 3)

np.mean(scores)
```

- Evaluating cross-validation score

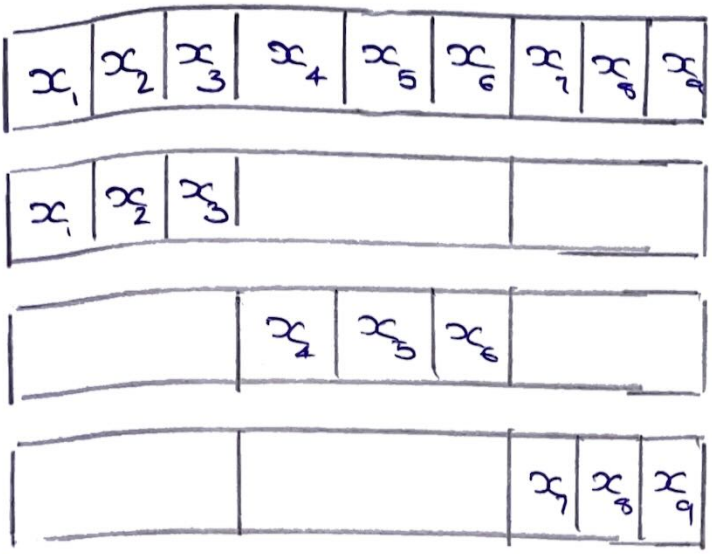


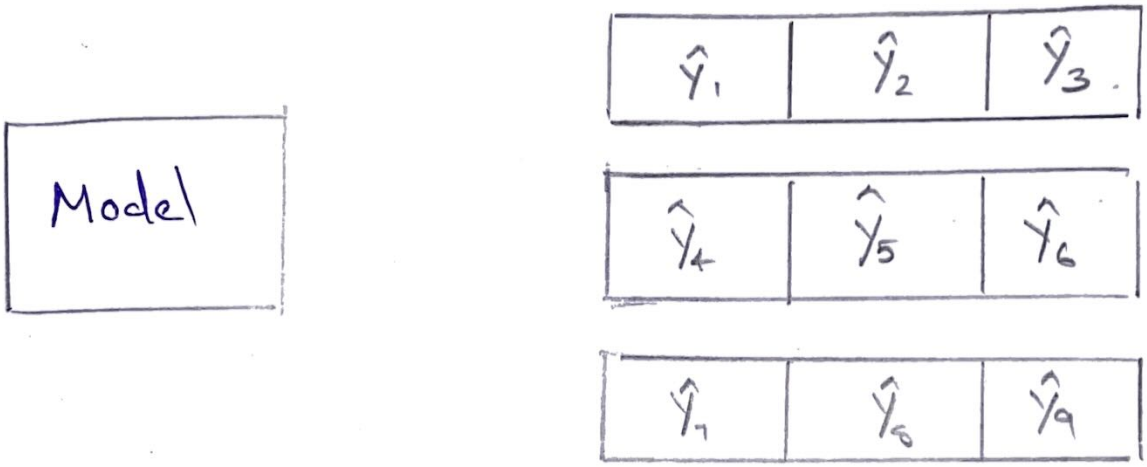
- Function cross_val_predict()

It returns the prediction that was obtained for each element when it was in the test set.

- Has a similar interface to cross_val_score()

```
from sklearn.model_selection import cross_val_predict

yhat = cross_val_predict(lr2e, x_data, y_data, cv = 3)
```



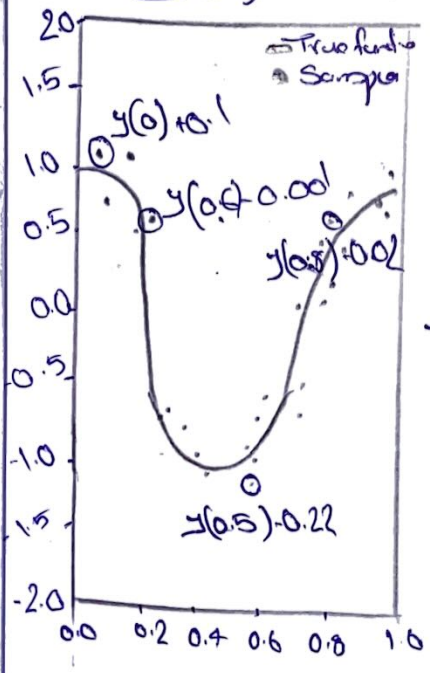


⊙ Overfitting, Underfitting and Model Selection

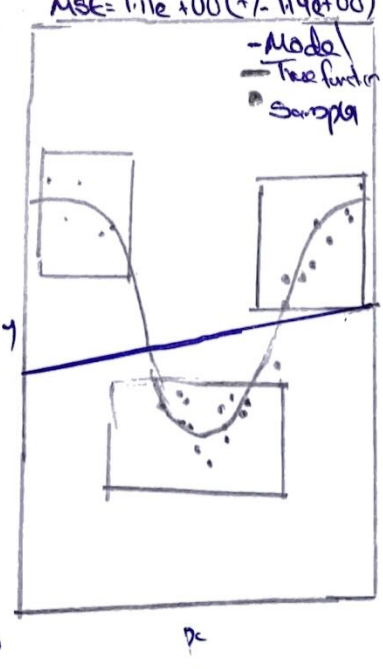
• Model Selection.

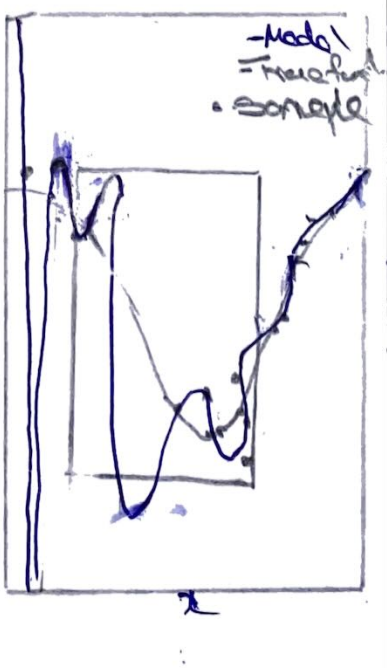
$\hat{y} = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4$
$b_5 x^5 + \ldots + b_8 x^8 + \ldots$
$\ldots\ldots b_{16} x^{16}$

· $y(x) + \text{noise}$



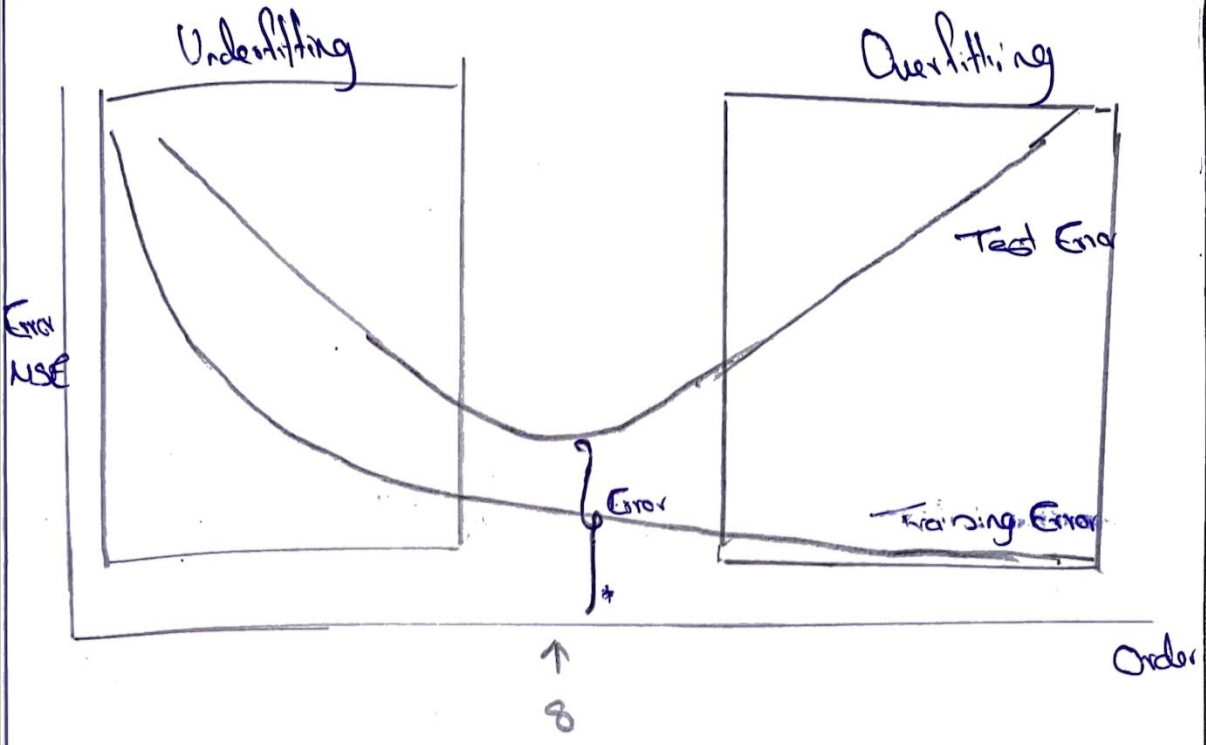
$y = b_0 + b_1 x$
Degree 1
MSE = 1.11e+00 (+/- 1.19e+00)



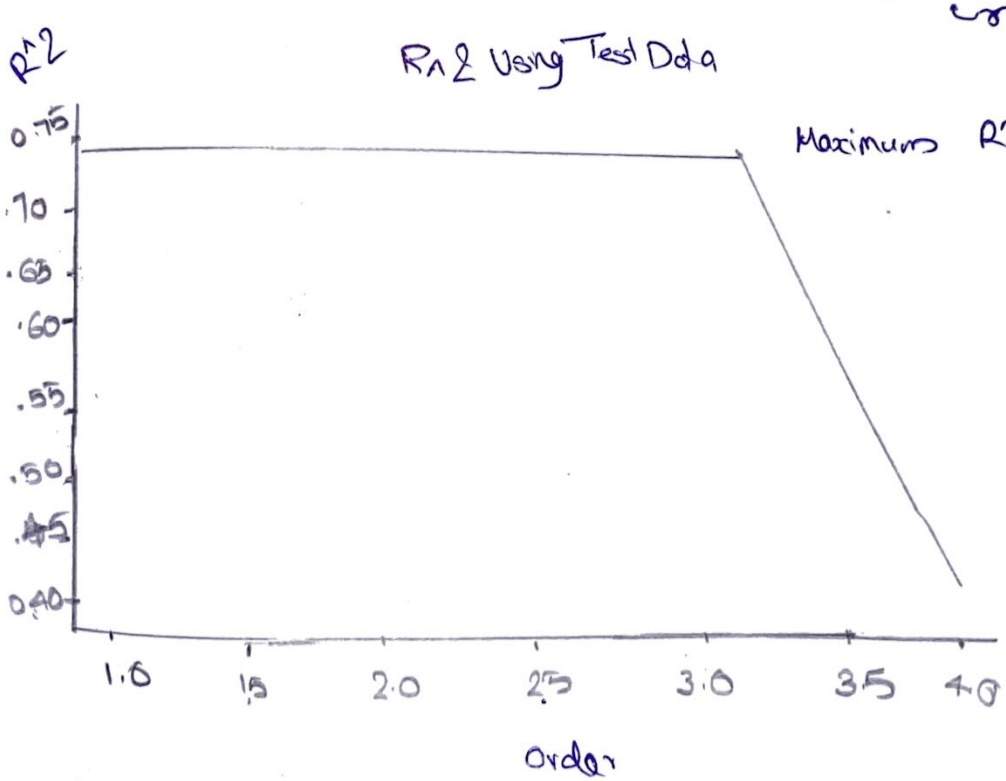


↑ Underfitting

↑ Overfitting



* y(x) noise



Less Bias | Underfitting: High Bias
Overfitting: High Variance | High Bias

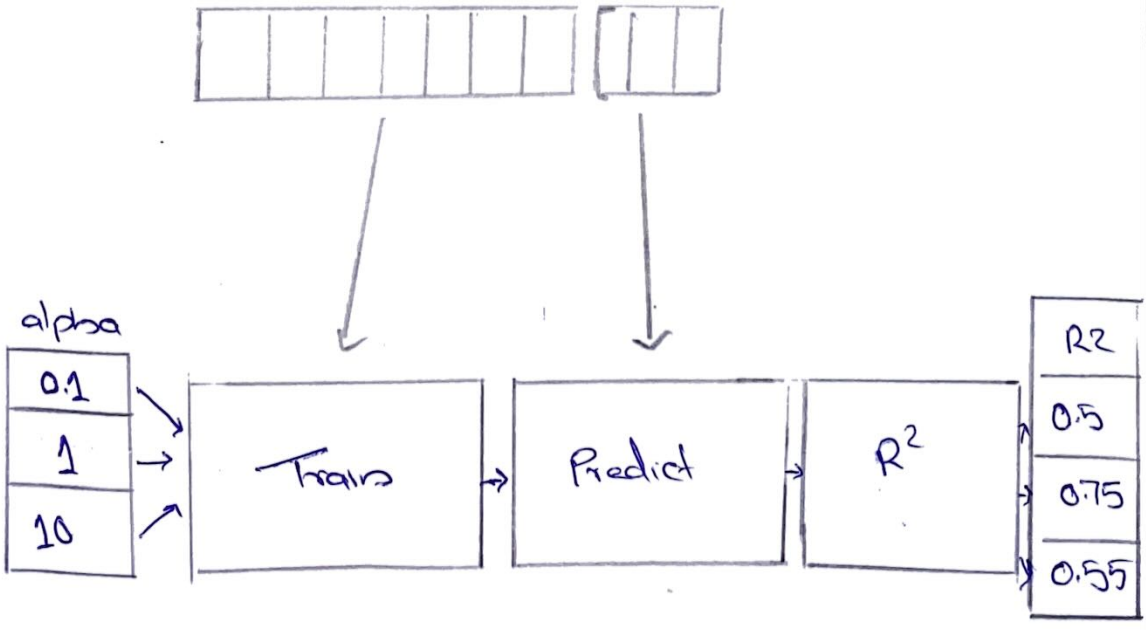
```
Requ_test=[]
order=[1,2,3,4]
for n in order:
    pr=PolynomialFeatures(degree=n)
    x_train_pr=pr.fit_transform(x_train[['horsepower']])
    x_test_pr=pr.fit_transform(x_test[['horsepower']])
    lr.fit(x_train_pr,y_train)
    Requ_test.append(lr.score(x_test_pr,y_test))
```

Ridge Regression

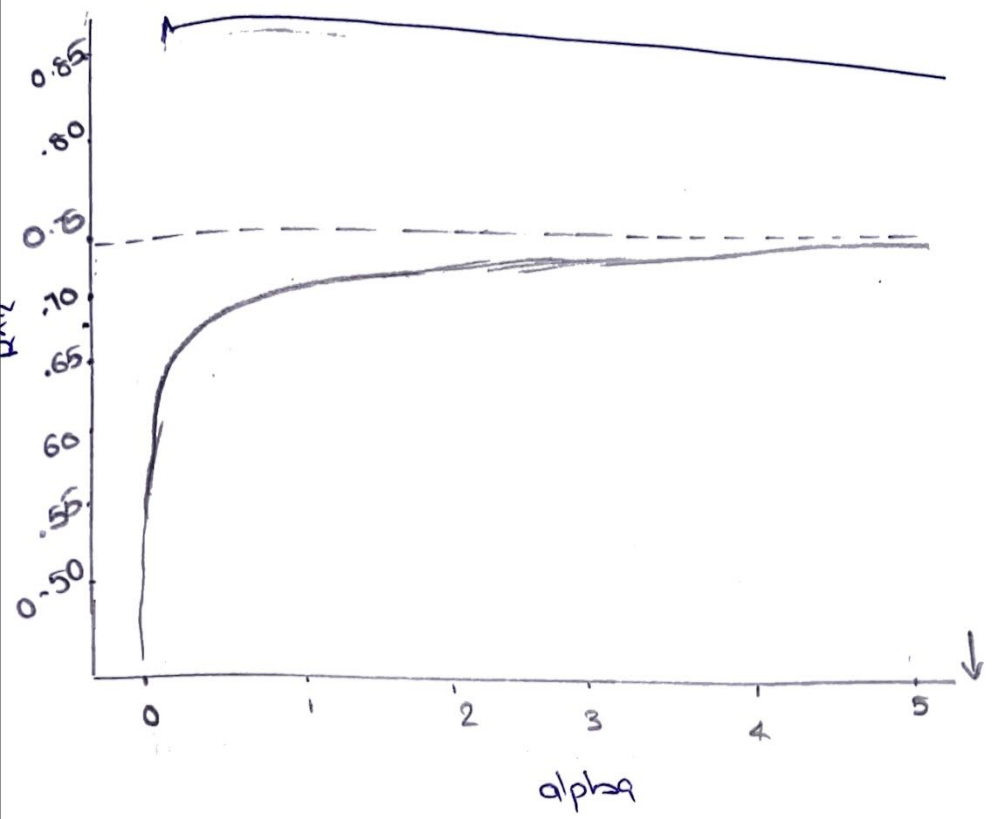
$$\hat{y} = 1 + 2x - 3x^2 - 2x^3 - 12x^4 - 40x^5 + 80x^6 + 71x^7 - 141x^8 - 38x^9 + 75x^{10}$$

Alpha	x	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9	x^{10}
0	2	-3	-2	-12	-40	80	71	-141	-38	75
0.001	2	-3	-7	5	4	-6	4	-4	4	6
0.01	1	-2	-5	-0.04	0.15	-1	1	-0.5	0.3	1
1	0.5	-1	-1	-0.614	0.70	-0.38	-0.56	-0.21	-0.5	-0.1
10	0	-0.5	-0.3	-0.37	-0.30	-0.30	-0.22	-0.22	-0.22	-0.17

```
from sklearn.linear_model import Ridge
RidgeModel = Ridge(alpha = 0.1)
RidgeModel.fit(X, y)
Yhat = RidgeModel.predict(X)
```



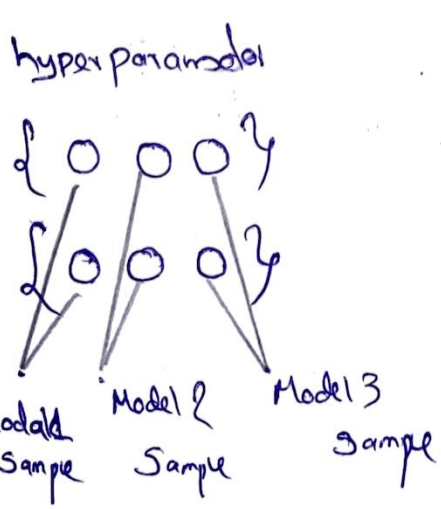
- Validation data
- training data

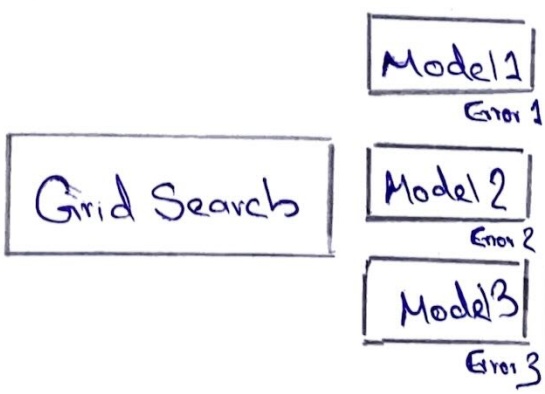


⊙ Grid Search.

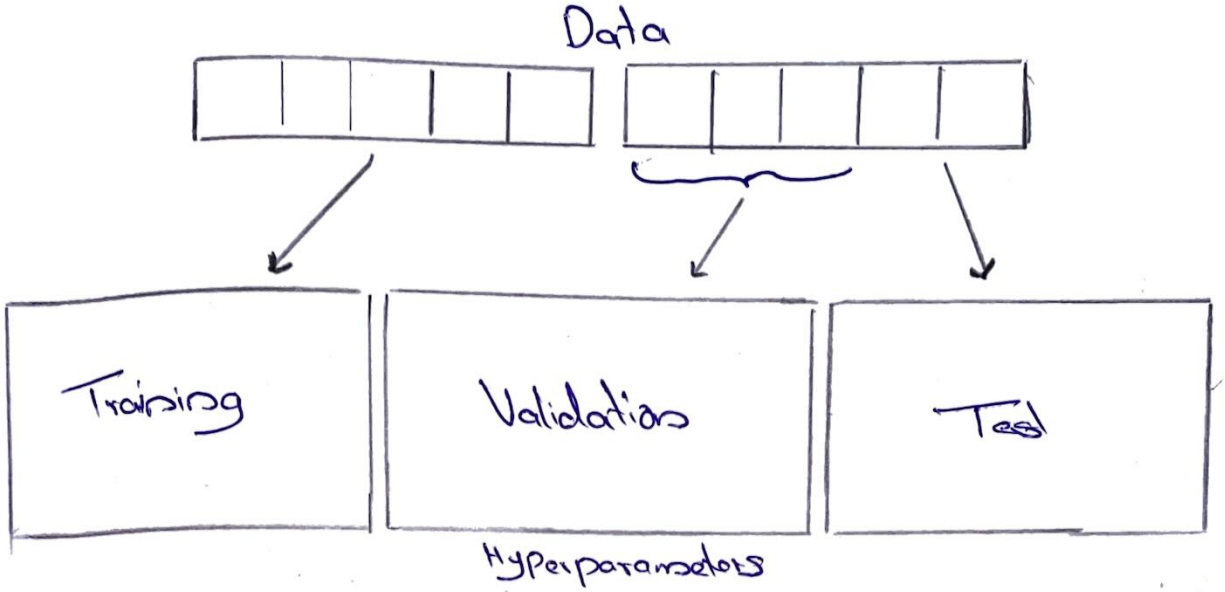
• Hyperparameters

- Alpha in Ridge regression is called a hyperparameter
- Scikit-learn has a means of automatically iterating over these hyperparameters using Cross-validation called Grid Search





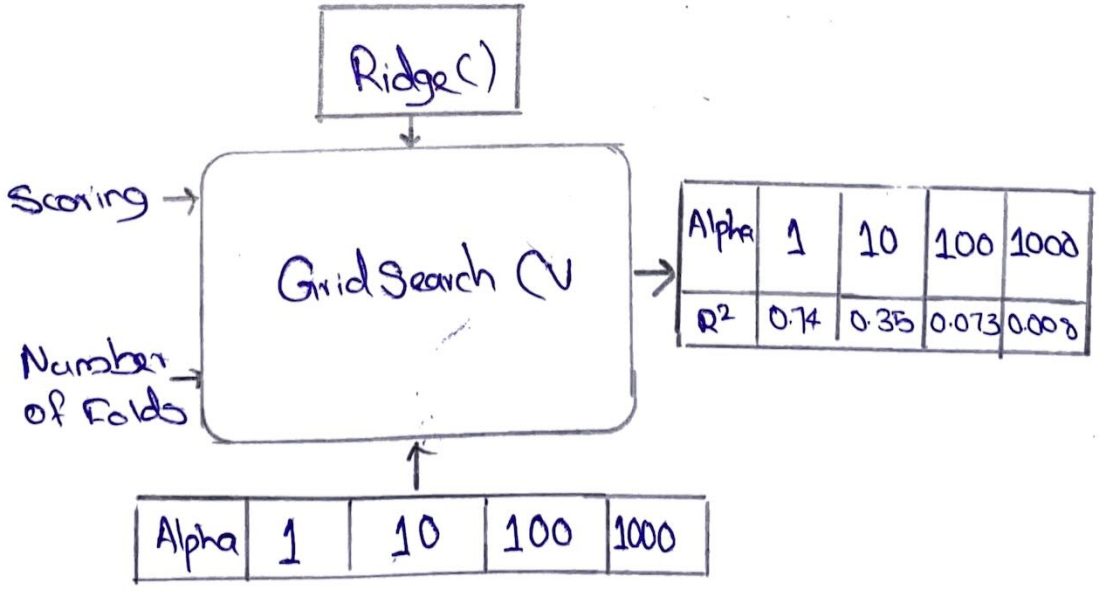
- Grid Search



Parameters: [{'alpha': [1, 10, 100, 1000]}]

Alpha	1	10	100	1000

Ridge()

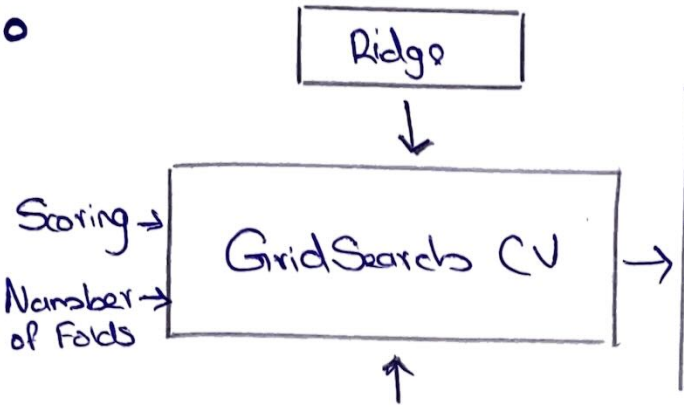


```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

Parameters1 = [{'alpha': [0.001, 0.1, 1, 10, 100, 1000, 10000
                          100000, 100000]}]

RR = Ridge()

Grid1 = GridSearchCV(RR, Parameters1, cv=4)

Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size',
                  'highway-mpg']], y_data)

Grid1.best_estimator_

scores = Grid1.cv_results_
scores['mean_test_score']
```

```
Parameters = [{'alpha': [1, 10, 100, 1000], 'normalize': [True, False]}]
```

Alpha	1	10	100	1000
Normalize	True	True	True	True
	False	False	False	False

```
Ridge()
```



Alpha	1	10	100	1000
True	0.69	0.32	0.17	0.17
False	0.67	0.66	0.66	0.64

Alpha	1	10	100	1000
Normalize	True	True	True	True
	False	False	False	False

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

Parameters1 = [{'alpha': [0.001, 0.1, 1, 10, 100, 1000, 10000, 100000
                          100000]}]

RR = Ridge()

Grid1 = GridSearchCV(RR, parameters1, cv=4)

Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine
          size', 'highway-mpg']], y_data)

Grid1.best_estimator_

Scores = Grid1.cv_results_
```