Model development :-

1. Simple and Multiple Linear Regression
2. Model Evaluation using Visualization
3. Polynomial Regression and Pipelines
4. R-squared and MSE for In-sample Evaluation
5. Prediction and Decision Making

Q. How can you determine a fair value for a used car ?

→ A model can be thought of as a mathematical eqn used to predict a value given one or more other values

→ Relating one or more independent variables to dependent variables.

independent variables (features) 'highway-mpg' → Model → dependent variable 'predicted price'

• Usually the more relevant data you have the more accurate your model is

Simple linear Regression
multiple linear Regression
polynomial regression

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

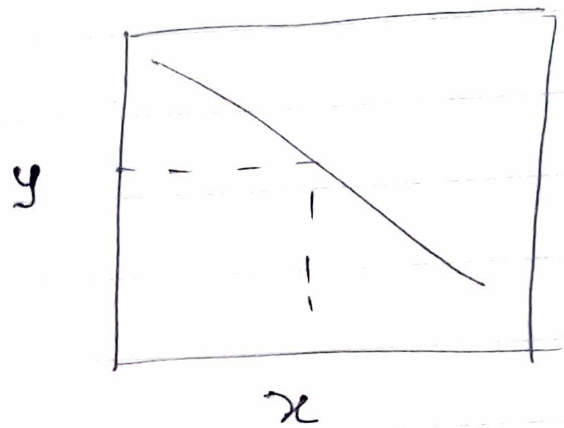
b_1 = the slope

Simple linear Regression - Prediction

$y = 38423 - 821x$

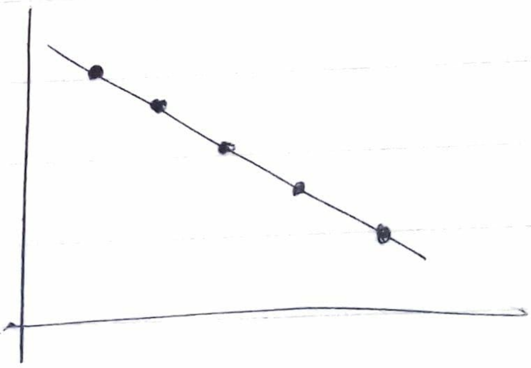
If $x = 0$

$y = 22003$



* Fit

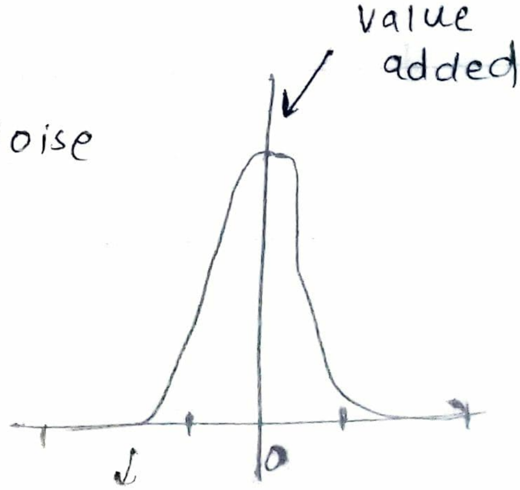
training points → Fit → (b_0, b_1)



Store points as data Frame or numpy arrays.

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

→ Noise :- distribution of Noise



probability that the

training points $\rightarrow$ $\boxed{\text{Fit}}$ $\xrightarrow{b_0 b_1}$ $\boxed{\text{Predict}}$ $\rightarrow \hat{y} = b_0 + b_1 x$

Fitting a simple linear model Estimation:-

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()
```

3. We create the predictor variable and target variable

```
x = df[['highway-mpg']]
y = df['price']
```

4. Then use lm.fit (X, Y) to fit the model, i.e. Find the parameters b_0, b_1

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

5. We can obtain a prediction

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

6. b_0 = lm.intercept = 38423.30

lm.coef = -821.733

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 b/n.

→ 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 ($x = 0$)

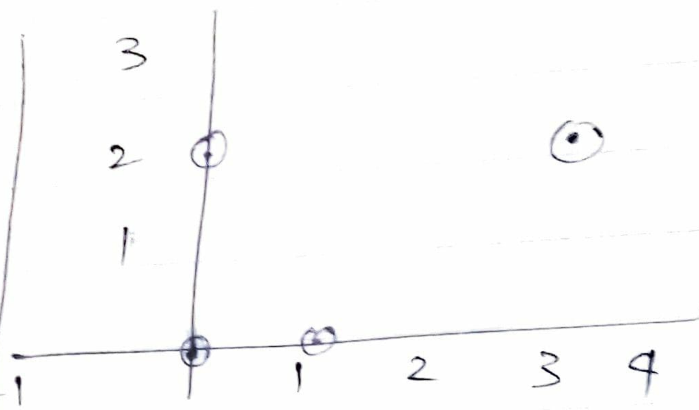
b_1 : the coefficient or parameter of x_1

b_2 : the coefficient of paramter x_2 and so on.

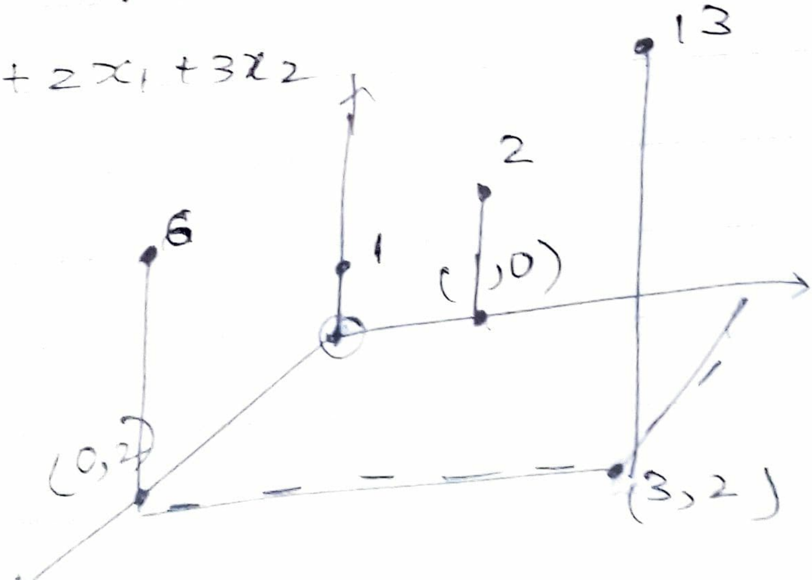
$$\hat{y} = 1 + 2x_1 + 3x_2$$

→ The variables x_1 and x_2 canbe visualized on a 2D plane,

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



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



Fitting a multiple Linear model Estimator

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

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

3. We can also obtain a prediction

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

Lect : Model Evaluation using Visualization

Why use regression plot ?

It gives us a good estimate of :

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

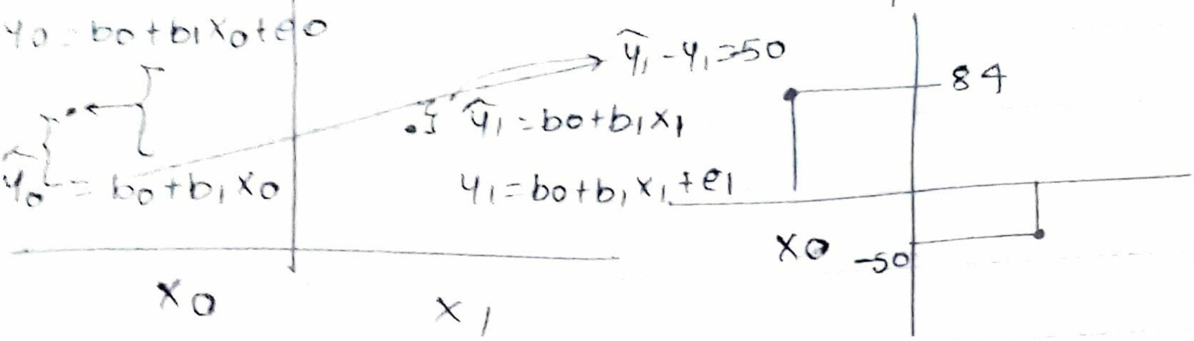
Regression plot shows us a combination of :

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

Regression plot :- To use regplot from the seaborn library.

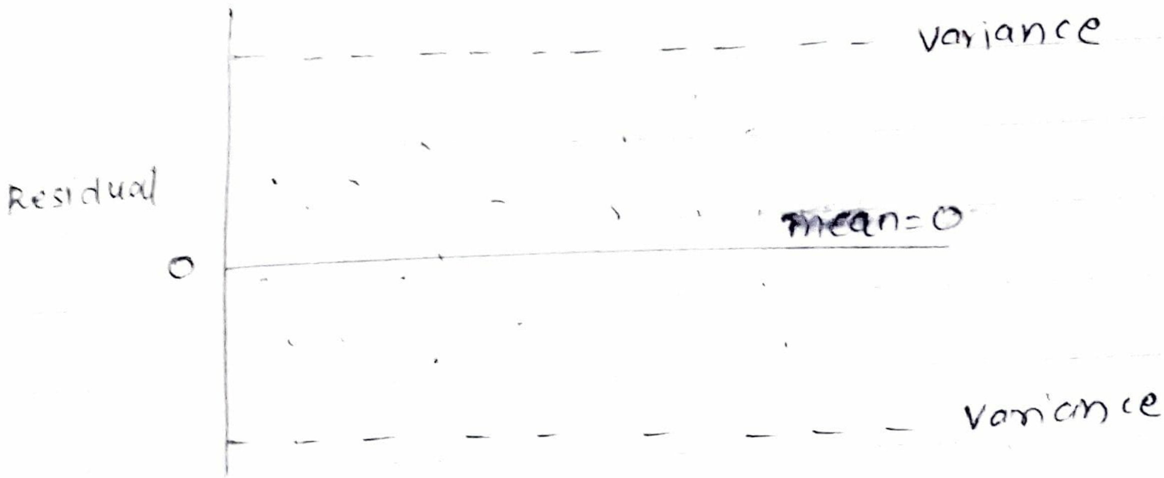
```
import seaborn as sns

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

Residual plot :- It represents the error b/w the actual value



$\hat{y}_0 - y_0 = 84$

X-axis : the predictor variable or fitted values



Spread of the residuals :-

- Randomly spread out around x-axis then a linear model is appropriate.

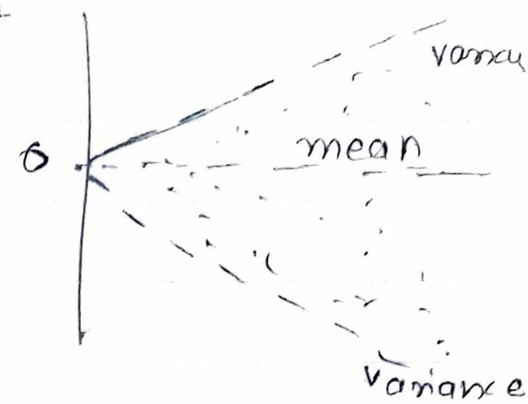
The value of error changes with X.

→ Not randomly spread out around x-axis
↳ Linear assumption is incorrect

→ Non linear model may be more appropriate

→ Not randomly spread out around the x-axis.

→ Variance appears to change with x-axis



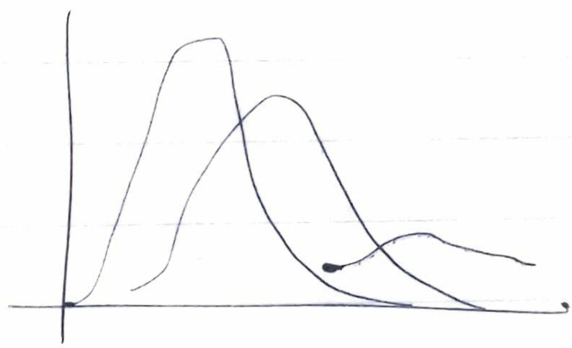
* using seaborn :-

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

* Distribution Plot :- counts the predicted value versus the actual value

compare the distribution plots :-

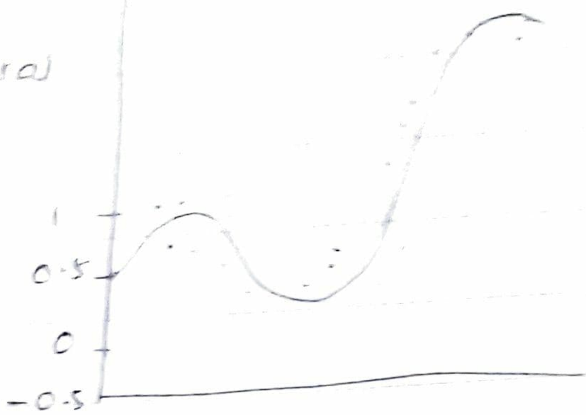
- The fitted values that result from the model
- The actual value



Polynomial Regression and Pipelines

* Polynomial Regression

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

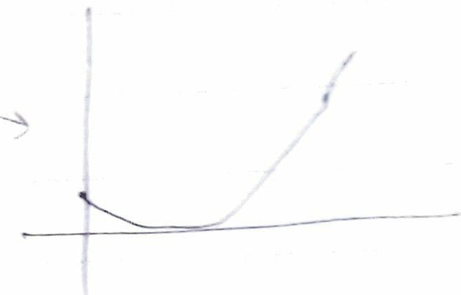


Curvilinear relationships :-

By squaring or setting higher order-terms of the predictor variables.

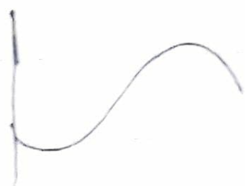
a) Quadratic - 2^{nd} order

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



b) cubic - 3^{rd} order

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



c) Higher order :-

$$\hat{y} = b_0 + b_1 x_1 + \text{- - - - - - -}$$

1. Calculate Polynomial of 3^{rd} order

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

2. We can print out of the model

```
print(p)
```

$$-1.557(x_1)^3 + 204.8(x_1)^2 + 8965x_1 + 1.37 \times 10^6$$

→ We can also have multi dimensional polynomial linear regression

$$\hat{y} = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_1 x_2 + b_4 (x_1)^2 + b_5 (x_2)^2 + \ldots$$

→ polyfit cannot perform this

- The "preprocessing" library in scikit-learn,

⇒ from sklearn.preprocessing import PolynomialFeatures

⇒ pr = PolynomialFeatures (degree = 2, include_bias = False)

pr = polymialFeatures (degree=2)

x_1	x_2
1	2

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

x_1	x_2	$x_1 x_2$	x_1^2	x_2^2
1	2	(1)(2)	1	$(2)^2$

Pre-processing :-

→ For eg we can Normalize the each feature simultaneously.

from sklearn.preprocessing import StandardScaler

SCALE = standarscaler()

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

x_scale = SCALE.transform(x_data[['horsepower', 'hig-y']]

We can simplify code by using pipeline

→ There are many steps to getting a prediction

Normalization → Polynomial transform → Linear Regression

$\underbrace{\text{Normalization} \rightarrow \text{Polynomial transform}}_{\text{transformations}} \rightarrow \underbrace{\text{Linear Regression}}_{\text{Prediction}}$

Measure for In-sample Evaluation

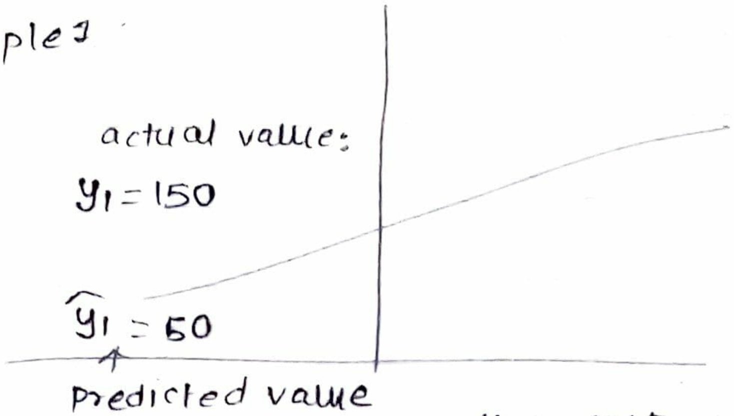
→ A way to numerically determine how good the model fit on dataset

→ Two important measures to determine the fit of a model:

- Mean Squared Error (MSE)
- R-squared (R^2)

For eg. for sample 1

150 - 50 = 100



MSE :- In python we can measure the MSE as follows :-

```
from sklearn.metrics import mean_squared_error
mean_squared_error(df['price'], Y_predict_simplefit)
```

R-squared :- The coefficient of Determination or R square (R^2).

→ Is a measure ~~of~~ to determine how close the data is to the fitted regression line.

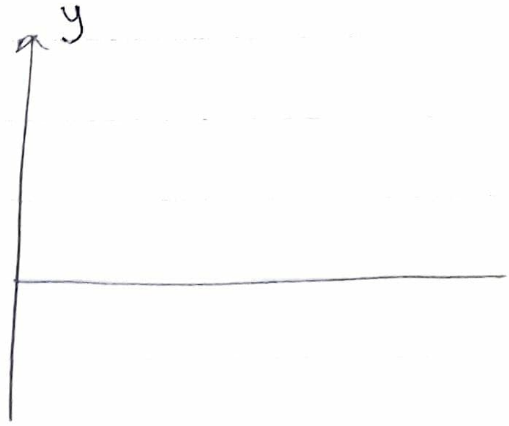
→ R^2 : the % ~~of~~ of variation of the target variable (Y) that is explained by the linear model.

→ This about as comparing a regression model to a simple model i.e the mean of the data points

Coefficient of Determination (R^2)

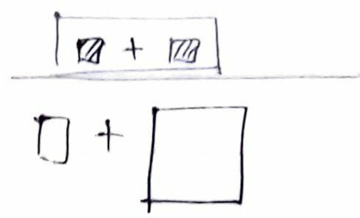
$$R^2 = \left(1 - \frac{\text{MSE of regression line}}{\text{MSE of the average of the data}}\right)$$

range b/w 0 to 1



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

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



$= 0$

```
X = df[['highway-mpg']]
Y = df['price']
lm.fit(X, Y)

lm.score(X, Y)
0.49659118 8
```

If R^2 is negative it can be due over fitting csv file

Prediction and Decision Making

→ Do the predicted values make sense

→ visualization

→ Numerical measures for evaluation

→ Comparing models

1. First we train the model

lm.fit (df['highway-mpg'], df['prices'])

2. Let's predict the price of a car with 30 highway-mpg.

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

3. Result : $ 13771.30