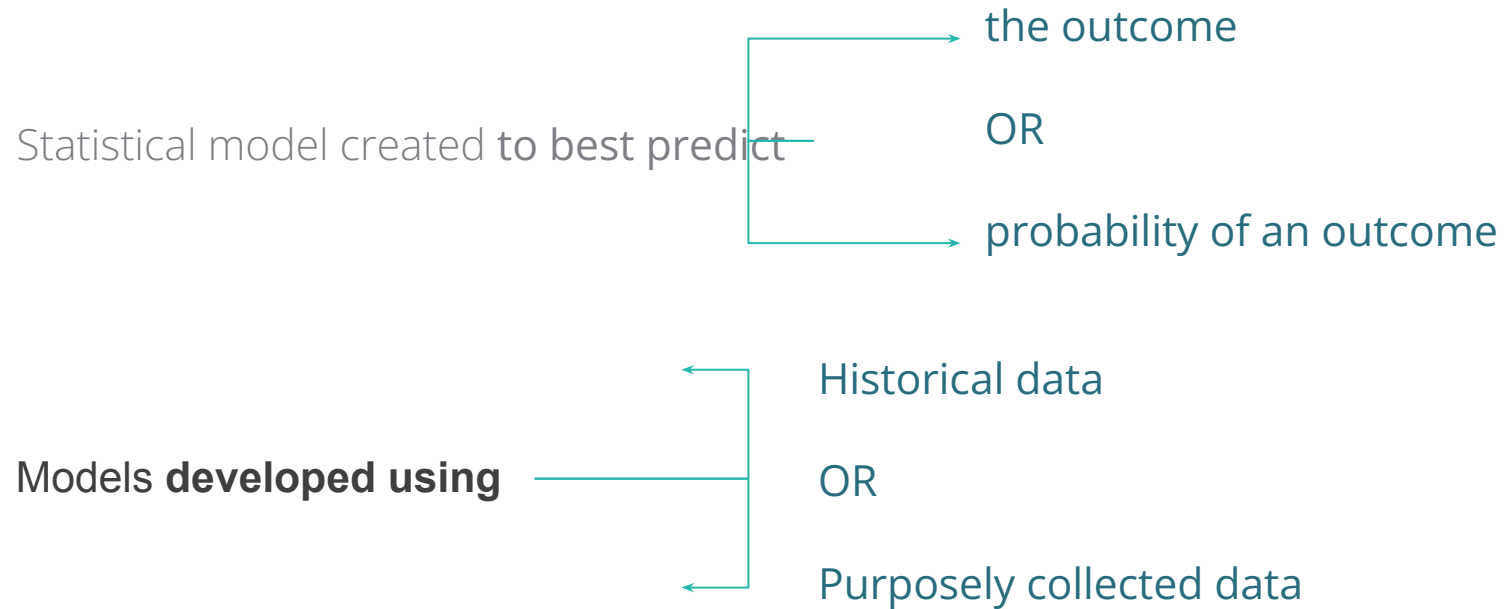


# Introduction to Predictive Modelling

# Contents

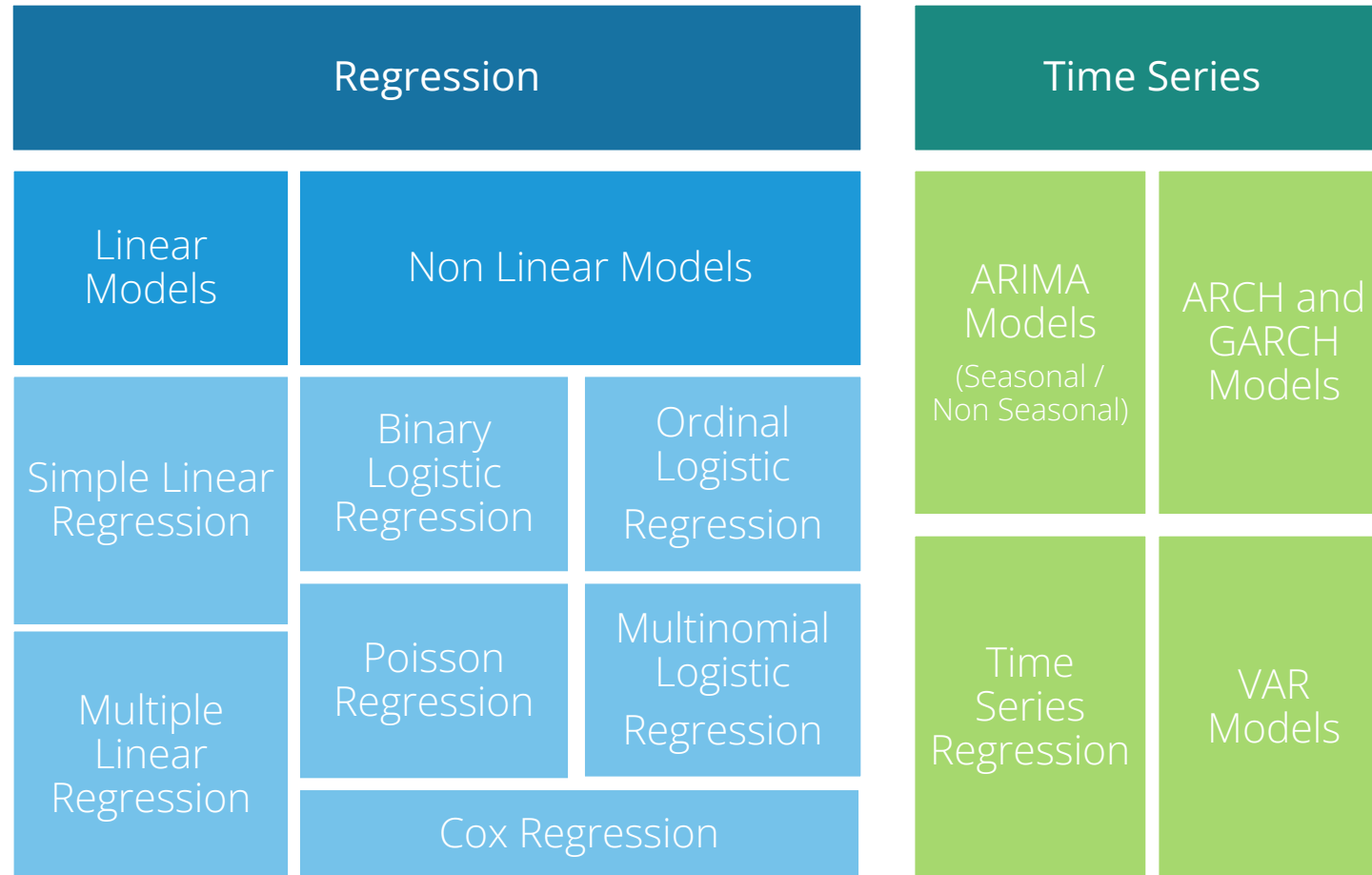
1. Introduction to Predictive Modelling
2. Important Statistical Models
3. General Approach
4. Key Steps in Model Building

# What is Predictive modelling?

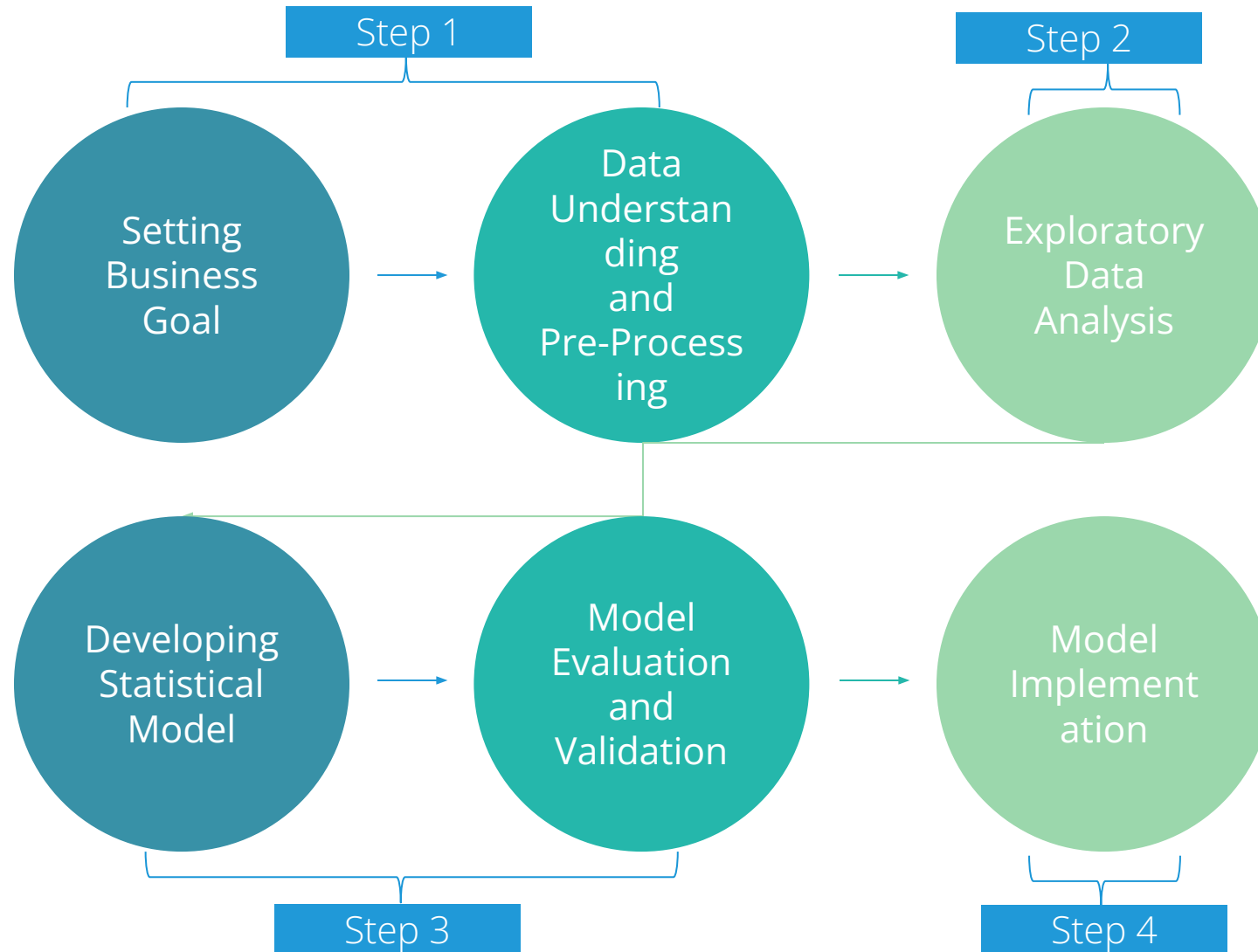


Predictive analytics is used in **financial services, insurance, telecommunications, retail, travel, healthcare, pharmaceuticals, sports and several other fields**

# Predictive modelling – Important Techniques



# Predictive modelling – General Approach



# Step 1 – Data Understanding and Pre-processing

## Data Understanding and Pre Processing

- Data Understanding
  - Understanding data dimension, variable types, variable relationships
- Converting raw data to usable data
  - Data cleaning by checking for and handling:
    - Missing values
    - Inconsistencies
- Transforming variables
- Feature engineering
  - Using domain knowledge to create new features or variables which can be used in the model

### Pre-Processing:

- Grouping / Factoring / Segmentation / Reduction

# Step 2 – Exploratory Data Analysis

## Exploratory Data Analysis

- Performing exploratory data analysis using:
  - Frequency tables
  - Cross tables
  - Descriptive statistics
  - Visualizations
  - Correlation matrix

# Step 3 – Model Identification , Selection and Validation

## Model Selection and Validation

- **Model identification and selection** is based on:
  - Study objective
  - Type of dependent variable
  - Checking different statistics / decision criteria based on models (Eg. p-value,  $R^2$ , AIC, etc.) i.e diagnostic checks
  - Automatic search procedures
- **Cross Validation**
  - Splitting data into training data and test data
  - Checking predictive ability of model on new data
  - Comparison of results with theoretical expectations, empirical results and simulation results



# Step 4 – Model Implementation

## Model Implementation

- Drawing inferences from the model results by :
  - Building equations using only the coefficients of significant variables
  - Mapping the model chosen with the existing system
- Fitting the model on new data and generating predictions
- Observing values of Predictors :
  - In a spreadsheet or
  - Web application or any other user interface or
  - Integrating the current systems

# Get an Edge!

Any predictive model is developed on historical data. Sample size and data dimension are key determinants for a good model

- If sample size is too small, model may not give good insight about the relationship among the variables.
- Also, if the data has large number of variables (columns) but few observations (rows), we are essentially trying to learn too much from a small sample. Results from models developed using such data will be erratic. **The rule of thumb for appropriate sample dimension is that observations should be 10 times the number of variables.** For instance, if we wish to study the relationship of 8 variables, then we must have more than 80 observations.

# Quick Recap

In this session, we gained knowledge on the concept of **predictive modelling** :

## Predictive modelling

- Used to predict the outcome or the probability of an outcome
- Models developed using historical data or purposely collected data

## Important Statistical models

- Regression Models- Linear and non-linear
- Time Series Models

## General Approach in Predictive modelling

- Data understanding and pre-processing
- Exploratory data analysis
- Model selection and validation
- Model Implementation

# Introduction to Multiple Linear Regression - I

# Content

1. Introduction to MLR
2. Statistical Model of MLR
3. Case Study
4. Model Fitting
5. Parameter Estimation – Ordinary Least Squares Method
6. Interpretation of Partial Regression Coefficients

# Multiple Linear Regression

- Multiple linear regression is used to explain the relationship between one continuous dependent variable and two or more independent variables.
- The independent variables can be continuous or categorical.
- Multiple Linear Regression is used when we want to predict the value of a variable based on the values of two or more other variables.
- The variable we want to predict is called the dependent variable
- The variables used to predict the value of dependent variable are called independent variables (or explanatory variables/predictors).
- Multiple linear regression requires the model to be linear in the parameters.
- Example: The price house in USD can be a dependent variable and size of house, location of house, air quality index in the area, distance from airport etc. can be independent variables.

# Statistical Model

$$Y = b_0 + b_1X_1 + b_2X_2 + \dots + b_pX_p + e$$

where,

$Y$  : Dependent Variable

$X_1, X_2, \dots, X_p$  : Independent Variables

$b_0, b_1, \dots, b_p$  : Parameters of Model

$e$  : Random Error Component

- Independent variables can either be **Continuous or Categorical**
- Multiple linear regression **requires the model to be linear in the parameters**
- Parameters of the model are estimated by Least Square Method.
- The **least squares (LS)** criterion states that the **sum of the squares of errors** (or residuals) **is minimum**.
- Mathematically, the following quantity is minimized to estimate parameters using the least square method.

^

- Error ss=  $\sum (Y_i - \hat{Y}_i)^2$

# Case Study – Modeling Job Performance Index

## Background

- A company conducts different written tests before recruiting employees. The company wishes to see if the scores of these tests have any relation with post-recruitment performance of those employees.

## Objective

- To predict employees' job performance index after a probationary period, based on test scores conducted at the time of recruitment

## Available Information

- Sample size is 33
- Independent Variables: Test scores conducted before recruitment on the basis of four criteria – **Aptitude, Test of Language, Technical Knowledge, General Information**
- Dependent Variable: **Job Performance Index**, calculated after an employee finishes a probationary period (6 months)



# Data Snapshot

Performance Index

**Dependent  
Variable**



**4 Independent  
Variables**



empid	jpi	aptitude	tol	technical	general
1	45.52	43.83	55.92	51.82	43.58
2	40.1	32.71	32.56	51.49	51.03
3	50.61	56.64	54.84	52.29	52.47

**Observations**

Columns	Description	Type	Measurement	Possible values
empid	Employee ID	integer	-	-
jpi	Job performance Index	numeric	-	positive values
aptitude	Aptitude score	numeric	-	positive values
tol	Test of Language	numeric	-	positive values
technical	Technical Knowledge	numeric	-	positive values
general	General Information	numeric	-	positive values

# Graphical Representation of Data

- It is always recommended to have a general look at your data and behavior of all variables before moving to modelling.
- This helps you to make intuitive inferences about the data, which can be statistically validated by your final model.
- The simplest way of doing this is to create a scatter plot matrix, which will give bivariate relationships between variables.

#Importing the Data

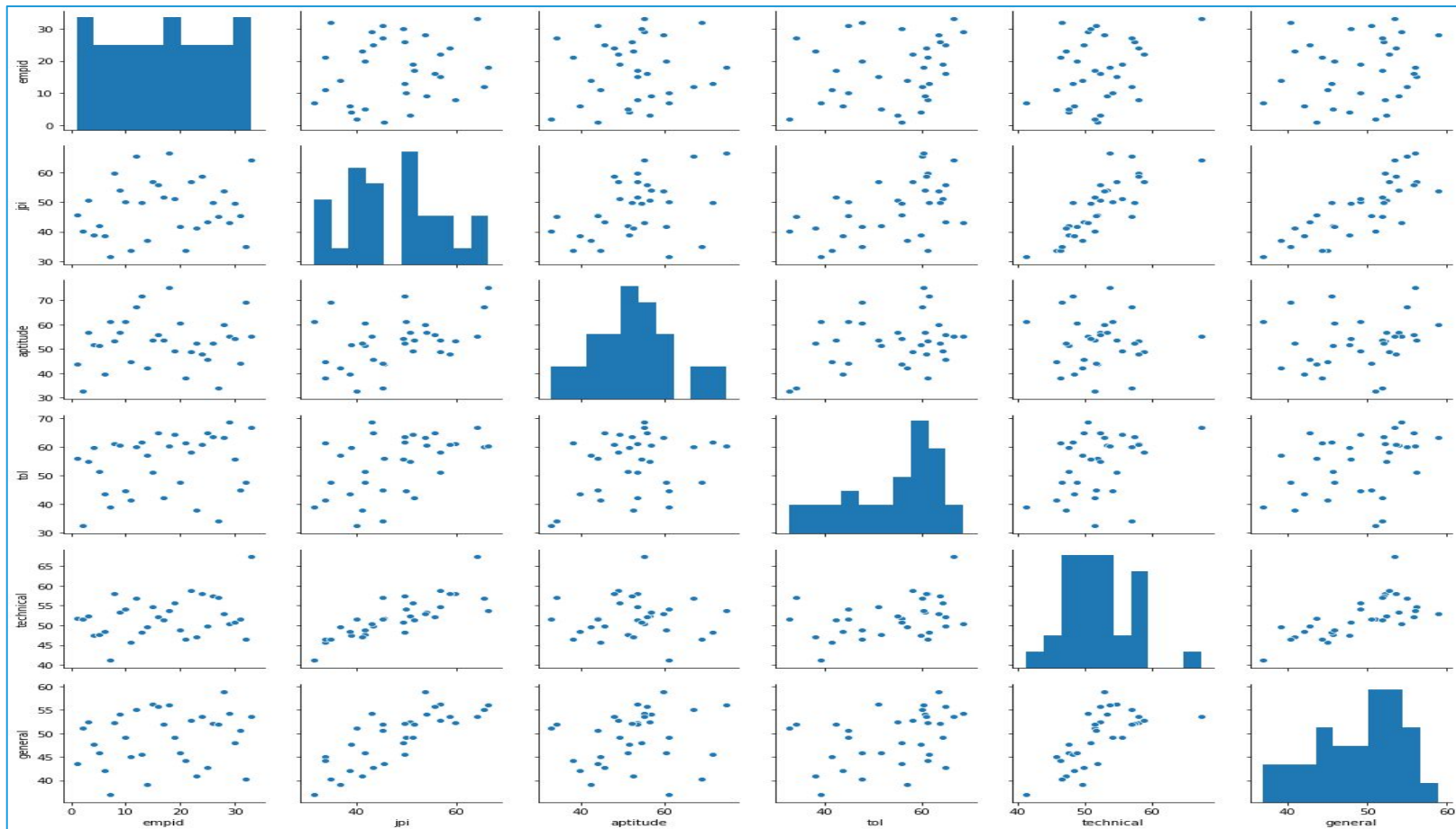
```
import pandas as pd  
perindex = pd.read_csv("Performance Index.csv")
```

#Graphical Representation of the Data

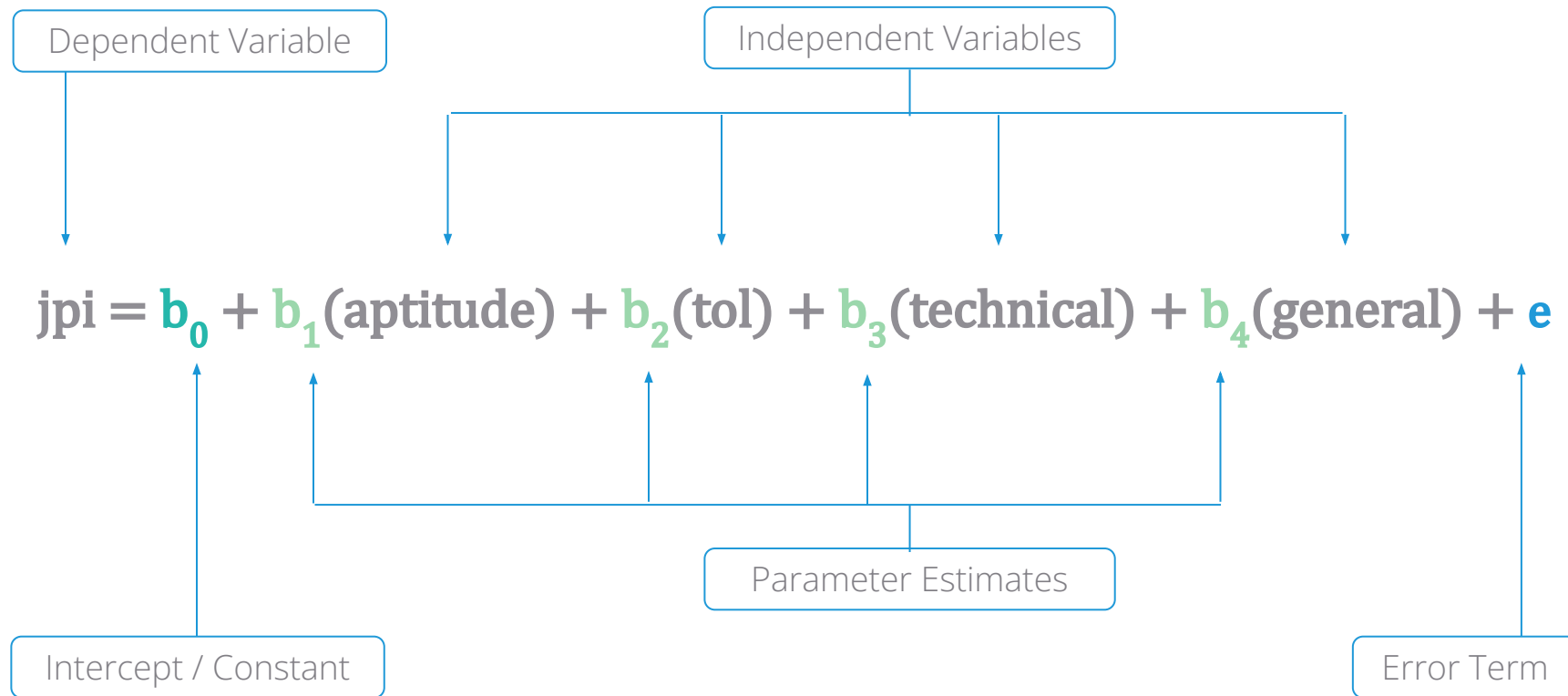
```
import seaborn as sns  
sns.pairplot(perindex)
```

# Scatter Plot Matrix

The `pairplot()` function in the **seaborn** library gives a scatter plot matrix and distribution of all variables using histograms.



# Model for the Case Study



# Parameter Estimation using Least Square Method

Parameters	Coefficients
Intercept	-54.2822
aptitude	0.3236
tol	0.0334
technical	1.0955
general	0.5368

$$E(jpi) = -54.2822 + 0.3236 (\text{aptitude}) + 0.0334 (\text{tol}) + 1.0955 (\text{technical}) + 0.5368 (\text{general})$$

# Parameter Estimation Using ols() function in Python

#Model Fit

```
import statsmodels.formula.api as smf
```

```
jpimodel=smf.ols('jpi ~ tol + aptitude + technical +general',  
data=perindex).fit()
```

```
jpimodel.params
```

- ❑ *ols() fits a linear regression.*
- ❑ *~ separates dependent and independent variables*
- ❑ *Left hand side of tilde(~) represents the dependent variable and right-hand side shows independent variables*
- ❑ *+ separates multiple independent variables.*

#Output

```
Intercept    -54.282247  
tol           0.033372  
aptitude     0.323562  
technical     1.095467  
general       0.536834  
dtype: float64
```

*Interpretation :*

- ❑ *jpimodel.params gives the model parameters.*
- ❑ *Signs of each parameter represent their relationship with the dependent variable.*

# Interpretation of Partial Regression Coefficients

- For every unit increase in the independent variable (X), the expected value of the dependent variable (Y) will change by the corresponding parameter estimate (b), keeping all other variables constant

Parameters	Coefficients
Intercept	-54.2822
aptitude	0.3236
tol	0.0334
technical	1.0955
general	0.5368

- From the parameter estimates table, we observe that the parameter estimate for Aptitude Test is 0.3236

We can infer that for one unit increase in aptitude test score, the expected value of job performance index will increase by 0.3236 units

# Quick Recap

In this session we have covered the **basics of multiple linear regression using Python**. Follow these simple steps to carry out your first analysis:

## Understand the Data

- Ensure the data is complete and consistent
- Identify dependent and independent variables

## Data Visualization

- **`pairplot()`** function from **seaborn** library gives scatter plot matrix

## Fit a Model

- **`ols()`** function from library **statsmodels** fits a linear regression model



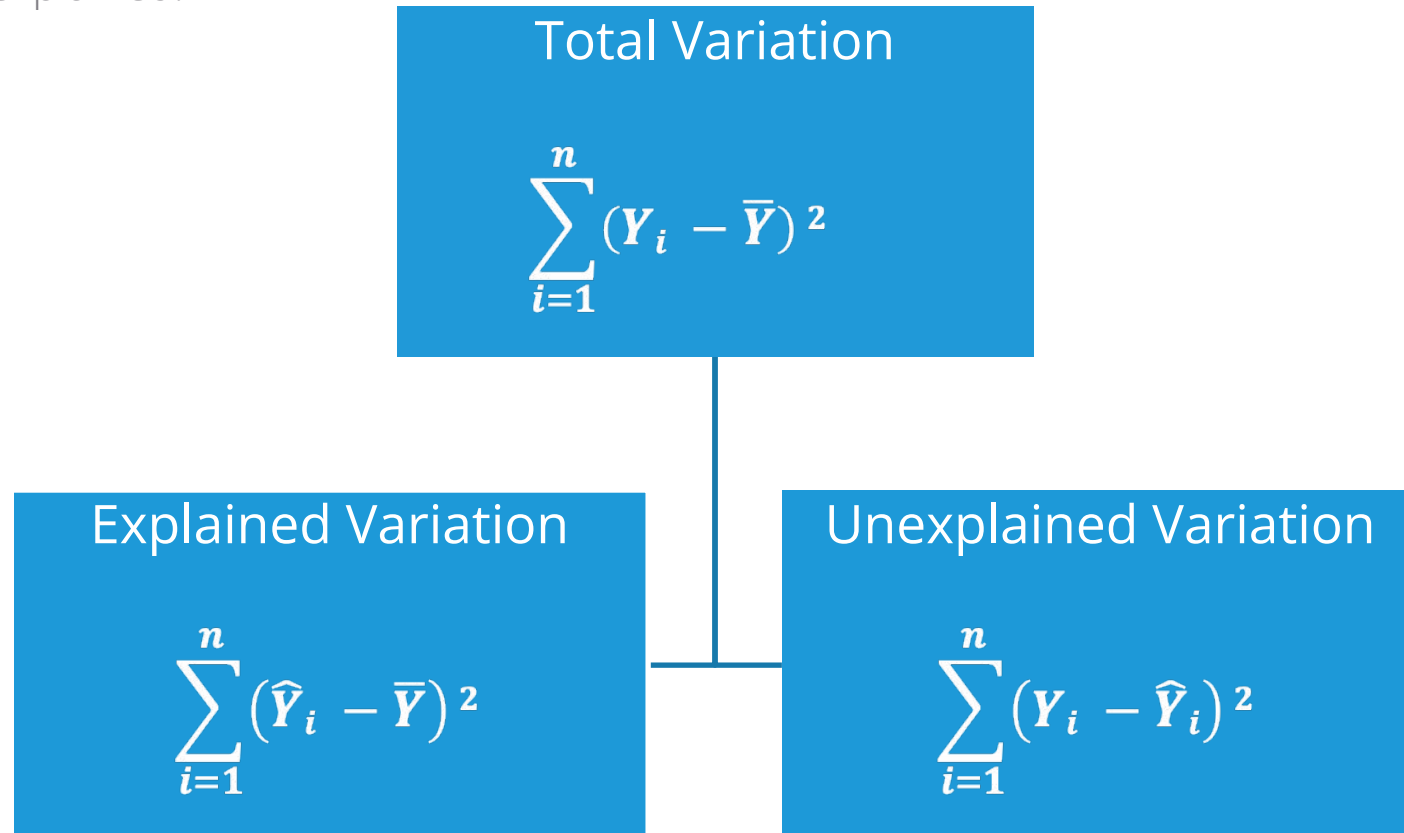
# Introduction to Multiple Linear Regression II

# Content

1. Global Testing – ANOVA
2. Individual Testing – t Test
3. Measure of Goodness of Fit – R Squared
4. Fitted values and Residuals
5. Predictions for New Dataset
6. Standardizing Coefficients

# Partitioning Total Variance

- Total Variation in dependent variables Y can be split into two: Explained and Unexplained.
- Explained variation is summation of the squared difference between estimated values of Y and the mean value of Y. Whereas, the sum of the squared difference between the actual values of Y and estimated values is considered to be unexplained.



# Global Testing – Using F Test

Testing whether at least one variable is significant

Objective	To test the null hypothesis that all the parameters are simultaneously equal to zero
-----------	--

Null Hypothesis ( $H_0$ ):  $b_1 = b_2 = \dots = b_p = 0$

Alternate Hypothesis ( $H_1$ ): At least one coefficient is not zero

Test Statistic	$F = \frac{\text{Mean Square of Regression}}{\text{Mean Square of Error}}$
Decision Criteria	Reject the null hypothesis if p-value < 0.05

# Individual Testing – Using t Test

Testing which variable is significant

Objective	To test the null hypothesis that parameters of individual variables are equal to zero
-----------	---

Null Hypothesis ( $H_0$ ):  $b_i = 0$

Alternate Hypothesis ( $H_1$ ):  $b_i \neq 0$

where  $i = 1, 2, \dots, p$

Test Statistic	$t = \frac{\text{Estimated } b_i}{\text{Standard Error of Estimated } b_i}$
Decision Criteria	Reject the null hypothesis if $p\text{-value} < 0.05$

# Measure of Goodness of Fit – R Squared

- $R^2$  is the proportion of variation in a dependent variable which is explained by independent variables. Note that  $R^2$  always increases if variable is added in the model

$$R^2 = \frac{\text{Explained Variation}}{\text{Total Variation}} = \frac{\sum_{i=1}^n (\hat{Y}_i - \bar{Y})^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2}$$

The adjusted R-squared is a modified version of R-squared that has been adjusted for the number of predictors in the model

$$R_a^2 = 1 - \frac{n-1}{n-p-1} (1 - R^2)$$

The adjusted R-squared is a modified version of R-squared that has been adjusted for the number of predictors in the model. Normally,  $R^2$  greater than 0.7 is considered as the benchmark for accepting the goodness of fit of a model.

# Understanding Summary Output

#Model Summary

```
jpimodel.summary()
```

*summary() generates a detailed description of the model.*

OLS Regression Results						
=====						
Dep. Variable:	jpi	R-squared:	0.877			
Model:	OLS	Adj. R-squared:	0.859			
Method:	Least Squares	F-statistic:	49.81			
Date:	Wed, 23 Oct 2019	Prob (F-statistic):	2.47e-12			
Time:	14:01:20	Log-Likelihood:	-85.916			
No. Observations:	33	AIC:	181.8			
Df Residuals:	28	BIC:	189.3			
Df Model:	4					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]
-----						
Intercept	-54.2822	7.395	-7.341	0.000	-69.429	-39.135
tol	0.0334	0.071	0.468	0.643	-0.113	0.179
aptitude	0.3236	0.068	4.774	0.000	0.185	0.462
technical	1.0955	0.181	6.039	0.000	0.724	1.467
general	0.5368	0.158	3.389	0.002	0.212	0.861
=====						
Omnibus:	2.124	Durbin-Watson:		1.379		
Prob(Omnibus):	0.346	Jarque-Bera (JB):		1.944		
Skew:	-0.544	Prob(JB):		0.378		
Kurtosis:	2.518	Cond. No.		1.25e+03		
=====						

*Interpretation :*

- Reject Global Testing null hypothesis that no variables are significant as p-value is  $< 0.05$
- Intercept, aptitude, technical, general are significant variables (p-values  $< 0.05$ )
- tol is not significant (p-value  $> 0.05$ )

# Summary of Findings

Significant variables → Aptitude  
→ Technical knowledge  
→ General information

Out of four dependent variables, **three affect job performance index positively**

---

$R^2$  → **0.88**

88% of the variation in job performance index is explained by the model & 12% is unexplained variation



# Fitted Values and Residuals

#Model Fitting after eliminating the insignificant variable

```
jpimodel_new=smf.ols('jpi ~ aptitude + technical +general',  
data=perindex).fit()  
jpimodel_new.params
```

*The insignificant variable tol is not included in the new model*

#Output

```
Intercept    -54.406443  
aptitude      0.333346  
technical     1.116627  
general       0.543157  
dtype: float64
```

*Estimated values of the model parameters using the new model*



To get fitted values and residuals values, the model should include significant variables only

# Fitted Values and Residuals

#Adding Fitted Values and Residuals to the Original Dataset

```
perindex=perindex.assign(pred=pd.Series(jpimodel_new.fittedvalues))  
perindex=perindex.assign(res=pd.Series(jpimodel_new.resid))  
perindex.head()
```

*fittedvalues() and resid() fetch fitted values and residuals respectively.*

#Output

	empid	jpi	aptitude	tol	technical	general	pred	res
0	1	45.52	43.83	55.92	51.82	43.58	41.738503	3.781497
1	2	40.10	32.71	32.56	51.49	51.03	41.709731	-1.609731
2	3	50.61	56.64	54.84	52.29	52.47	51.362151	-0.752151
3	4	38.97	51.53	59.69	47.48	47.69	41.691486	-2.721486
4	5	41.87	51.35	51.50	47.59	45.77	40.711451	1.158549

*Interpretation :*

- *pred values are calculated based on the values of the model parameters*
- *res is the difference between the actual jpi values and the pred values.*
- *Lower the residuals, lesser is the difference between fitted and observed and better is the model.*

# Predictions for a New Dataset

- A new data set should have all the independent variables used in the model
- Column names of all common variables in the new and old datasets should be identical
- Note that missing values will be taken as 0 (which can be incorrect)

## #Importing New Dataset

```
perindex_new=pd.read_csv("Performance Index new.csv")
perindex_new=perindex_new.assign(pred=pd.Series(jpimodel_new.predict(perindex_new)))
```

*predict() returns predicted values. The fitted model is the first argument and new dataset object is the second argument. This ensures Python uses parameters from the fitted model for predictions on new data.*

```
perindex_new.head()
```

	empid	jpi	tol	technical	general	aptitude	pred
0	34	66.35	59.20	57.18	54.98	66.74	61.552576
1	35	56.10	64.92	52.51	55.78	55.45	53.008978
2	36	48.95	63.59	57.76	52.08	51.73	55.621537
3	37	43.25	64.90	50.13	42.75	45.09	39.820600
4	38	41.20	51.50	47.89	45.77	50.85	40.879766

# Predictions with Confidence Interval

#Predictions with Confidence Interval

```
result = jpimodel_new.get_prediction(perindex_new)  
result.conf_int()
```

- *conf\_int() generates 95% confidence intervals by default.*
- *Left hand side values in array gives lower confidence interval values, right gives upper.*

#Output

```
array([[59.00955719, 64.09559387],  
       [50.67791702, 55.34003898],  
       [53.65401364, 57.58906082],  
       [37.73389546, 41.90730465],  
       [39.23363549, 42.52589584],  
       [45.41626758, 47.98650295]])
```

**Q. Why are confidence intervals needed for predictions?**

**A.** The point estimate is the best guess of the true value of the parameter, while the interval estimate gives a measure of accuracy of that point estimate by providing an interval that contains plausible values.



If you wish to specify the level of tolerance/confidence, use `alpha=` argument in the `conf_int()` function. For example, to calculate 90% confidence intervals, `alpha = 0.1`

# Standardized Coefficients

How to determine relative importance of predictors?

One possible answer is standardized regression coefficient

Predictors can have very different types of units, which make comparing regression coefficients meaningless. One solution is to standardize all variables before performing regression analysis.

**standardization refers to the process of subtracting the mean ( $\mu$ ) from each value and dividing by the standard deviation ( $\sigma$ ).**

$$Z = \frac{X - \mu}{\sigma}$$

	X1	X2	Standardized X1	Standardized X2
	32	1052	-0.20	-1.74
	37	1237	0.46	-1.06
	25	1672	-1.12	0.54
	39	1724	0.72	0.74
	23	1555	-1.38	0.11
	41	1423	0.99	-0.37
	43	1870	1.25	1.27
	28	1661	-0.72	0.50
Mean	33.5	1524.25		
SD	7.60	271.69		

# Standardized Coefficient - Python code

## Generation of standardized parameter estimate

```
import pandas as pd
import numpy as np
from scipy import stats
import statsmodels.formula.api as smf

# standardizing dataframe
df_z =
perindex.select_dtypes(include=[np.number]).dropna().apply(stats.zscore)

# fitting regression
formula = 'jpi ~ aptitude + technical + general'
std_coef = smf.ols(formula, data=df_z).fit()
std_coef.params
```

- ❑ *stats.zscore* standardizes the specified variables.
- ❑ *.dropna()*, Otherwise, *stats.zscore* will return all NaN for a column if it has any missing values.
- ❑ *.select\_dtypes(include=[np.number])* selects the numeric columns from data frame
- ❑ *.params* gives the standardized coefficients.

# Standardized Coefficient - Python code

#Output

```
Intercept    -9.072604e-16  
aptitude      3.543742e-01  
technical     5.880966e-01  
general       3.236793e-01  
dtype: float64
```

*Interpretation:*

▣ *technical has highest impact on job performance index followed by aptitude*

# Quick Recap

Till now, we learnt the **basics of multiple linear regression**. Follow these simple steps to carry out your first analysis:

## Check Variable Significance

- Undertake global and individual testing

## Measure Goodness of Fit

- Check R-squared, Adjusted R-squared to see how much variation is explained by the model
- Generally, R-squared greater than 0.7 is considered to be a good indicator

## Summary Output

- Summary of **ols()** output is exhaustive and gives t statistics, p-value,  $R^2$  to draw fundamental conclusions about the model



# Quick Recap

In this session, we learnt how to **perform basic multiple linear regression in R**:

## Fitted Values and Errors

- **fitted()** and **resid()** are used to fetch fitted values and residuals respectively

## Predictions

- **predict()** function predicts values for new data
- Predictions can be obtained as either point estimates or as confidence intervals

## Standardizing Coefficients

- **stats.zscore** function in package **scipy** gives the standardized coefficients.
- It is used to compare the relative importance of independent variables when the variables are in different metric units

# Multiple Linear Regression Using Categorical Variables

# Content

1. MLR using Categorical Independent Variables
2. Statistical Model Using Dummy Variables
3. MLR with Categorical Dummy Variables in Python
4. Changing the base category

# Case Study – Predicting Restaurant Sales

## Background

- A city-based association of restaurants and cafes records all sorts of transactions and descriptive data for the purpose of industry-level analysis. The association wishes to find out if this data can be used to determine sales of restaurants.

## Objective

- To predict sales of restaurants

## Available Information

- Sample size is 16
- Independent Variables: **Location of the Restaurant** – Categorical Variable with 3 Categories – Mall, Street and Highway and **Number of Households in the Area**
- Dependent Variable: **Sales of the Restaurant**

# Data Snapshot

## RESTAURANT SALES DATA

**Independent Variables**

**Dependent Variable**



RESTAURANT	NOH	LOCATION	SALES
1	155	highway	131.27
2	93	highway	68.14

Columns	Description	Type	Measurement	Possible values
RESTAURANT	Restaurant Number	numeric	-	-
NOH	Number of Households in the Vicinity of the Restaurant	numeric	-	positive values
LOCATION	Whether the Restaurant is Situated in a Mall, on a Street or on a Highway	Categorical	highway, mall, street	3
SALES	Annual Sales of the Restaurant	numeric	-	positive values

# MLR using Categorical Independent Variables

If there is a categorical independent variable with  $K$  categories we must define only  $K - 1$  dummy variables

In the case study,

Dependent Variable	Sales of a Restaurant
Independent Variables	1. Location of the Restaurant 2. Number of Households in the Area

- Location is a categorical variable with 3 categories – Mall, Street and Highway  
Therefore, there will be  $3-1=2$  dummy variables
- The category for which dummy variable is not defined is called 'Base Category'

# Data Snapshot – With Dummy Variables

Y : Sales of a Restaurant

X1 : No. of Households

X2 : 1 if location is 'Mall'

and 0 otherwise

X3 : 1 if location is 'Street' and 0

otherwise

- The first six rows have 0 under MALL and STREET columns
- This clearly implies that the restaurant is located neither in mall nor on street
- It is located on HIGHWAY

RESTAURANT	NOH	LOCATION	SALES	MALL	STREET
1	155	highway	135.27	0	0
2	93	highway	72.74	0	0
3	128	highway	114.95	0	0
4	114	highway	102.93	0	0
5	158	highway	131.77	0	0
6	183	highway	160.91	0	0
7	178	mall	179.86	1	0
8	215	mall	220.14	1	0
9	172	mall	179.64	1	0
10	197	mall	185.92	1	0
11	207	mall	207.82	1	0
12	95	mall	113.51	1	0
13	224	street	203.98	0	1
14	199	street	174.48	0	1
15	240	street	220.43	0	1
16	100	street	93.19	0	1

# Why Not K Dummy Variables?

Can there be as many dummy variables as categories?

- If  $k$  dummy variables are created for  $k$  categories, there will be perfect multicollinearity – The Dummy Variable Trap
- In order to avoid falling into this trap, model with  $k$  categories and  $k$  dummy variables must have no intercept
- In such a model, coefficients will directly represent mean value of that variable

However, it is desirable to stick to the rule of  $k$  categories =  $k - 1$

Dummy Variables



# Statistical Model Using Dummy Variables

Basic Multiple Linear Regression Model

$$Y = b_0 + b_1X_1 + b_2X_2 + \dots + b_pX_p + e$$

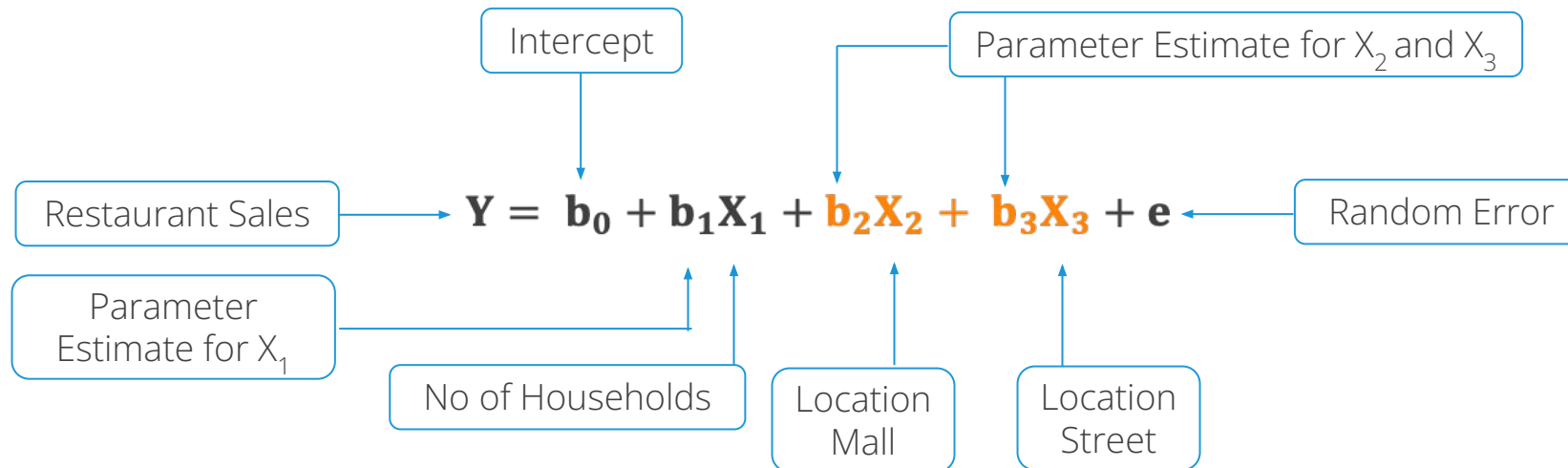
where,

$Y$  : Dependent Variable

$X_1, X_2, \dots, X_p$  : Independent Variables

$b_0, b_1, \dots, b_p$  : Parameters of Model

$e$  : Random Error Component



# Interpretation of Results

Regression Coefficients of categorical dummy variables are interpreted **relative to the base category**

- The positive beta coefficient of mall ( $b_2$ ) implies that if the restaurant is located in a mall, the sale amount will be higher than sale amount of restaurant on highway by  $b_2$  units.
- If the coefficient is negative  $b_2$  then it implies that restaurant located in mall will have lower sales than restaurant on highway by  $b_2$  units.
- The same applies to street v/s highway
- Remember, dummy variable inferences are useful only if the variable is significant

# MLR with Categorical Dummy Variables in Python

#Importing the Data

```
import pandas as pd
restaurantsales = pd.read_csv("RESTAURANT SALES DATA.csv")
restaurantsales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16 entries, 0 to 15
Data columns (total 4 columns):
RESTAURANT    16 non-null int64
NOH           16 non-null int64
LOCATION        16 non-null object
SALES         16 non-null float64
dtypes: float64(1), int64(2), object(1)
memory usage: 640.0+ bytes
```

```
restaurantsales['LOCATION']=restaurantsales['LOCATION'].astype('category')
restaurantsales['LOCATION'].cat.categories
```

```
Index(['highway', 'mall', 'street'], dtype='object')
```

- *info() shows class and levels of variables in the data.*
- *.astype is used to convert object type into “category”*
- *cat.categories() is used to check categorical variable’s levels and their order.*

# MLR with Categorical Dummy Variables in Python

```
#Fitting Multiple Linear Regression Model
```

```
import statsmodels.formula.api as smf
```

```
salesmodel=smf.ols('SALES~NOH+LOCATION',data=restaurantsales).fit()
```

```
salesmodel.summary()
```

← *summary() generates a detailed description of the model.*

# MLR with Categorical Dummy Variables in Python

#Output

```
=====
              coef      std err          t      P>|t|      [0.025      0.975]
-----
Intercept              2.1892         8.592         0.255         0.803        -16.531         20.910
LOCATION[T.mall]       37.0524         5.814         6.373         0.000         24.385         49.720
LOCATION[T.street]      7.1537         6.731         1.063         0.309         -7.513         21.820
NOH                   0.8383         0.056        14.920         0.000          0.716          0.961
=====
Omnibus:                 1.196   Durbin-Watson:                 2.713
Prob(Omnibus):            0.550   Jarque-Bera (JB):                 0.403
Skew:                    -0.387   Prob(JB):                 0.817
Kurtosis:                 3.071   Cond. No.                 637.
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

## OLS Regression Results

### Interpretation:

- Python orders factor levels alphabetically and takes the first level as the base category by default
- NOH and Mall are significant variables
- Beta coefficient for mall is 37.0524. This implies that if a restaurant is located in a mall, its sales will be more than the restaurant located on highway by 37.0524.
- Street too has a positive coefficient, implying that sales of restaurant located on street will be 7.1537 times higher than highway.

# Changing the Base Category in Python

```
#Importing the required library
```

```
from patsy.contrasts import Treatment
```

```
#Fitting Model on Data with Reordered Levels
```

```
salesmodel=smf.ols("SALES ~ C(LOCATION, Treatment(reference='mall')) +  
NOH",data=restaurantsales).fit()  
salesmodel.summary()
```

- *Treatment() reorders levels of a factor variables.*
- *reference= is used to specify changed reference (base) level.*

# Changing the Base Category in Python

#Output

```
=====
              coef    std err          t      P>|t|      [0.025    0.975]
-----
Intercept                39.2416     10.502      3.737      0.003     16.360     62.124
C(LOCATION, Treatment(reference='mall'))[T.highway]   -37.0524      5.814     -6.373      0.000    -49.720    -24.385
C(LOCATION, Treatment(reference='mall'))[T.street]    -29.8987      6.123     -4.883      0.000    -43.239    -16.558
NOH                      0.8383      0.056     14.920      0.000      0.716      0.961
=====
Omnibus:                 1.196    Durbin-Watson:           2.713
Prob(Omnibus):           0.550    Jarque-Bera (JB):         0.403
Skew:                   -0.387    Prob(JB):                 0.817
Kurtosis:                3.071    Cond. No.                  812.
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
=====

                        OLS Regression Results
=====
Dep. Variable:          SALES      R-squared:                0.970
Model:                  OLS        Adj. R-squared:           0.963
Method:                 Least Squares   F-statistic:             130.0
Date:                  Wed, 24 Feb 2021   Prob (F-statistic):      2.05e-09
=====
```

*Interpretation:*

- *Mall has now become the base category*
- *Coefficient of highway is just negative of coefficient of mall observed in previous model (Degree of association between mall and highway is the same, changed sign indicates that relativity has reversed)*
- *Note that street is also significant*

# Quick Recap

In this session, we learnt how to **handle categorical variables in multiple linear regression by introducing Dummy Variables**

Number of Dummy Variables	<ul style="list-style-type: none"><li>• The number of dummy variables must be one less than the number of levels in the categorical variable</li></ul>
Interpretation	<ul style="list-style-type: none"><li>• The coefficient attached to the dummy variables must always be interpreted in relation to the base or reference group.</li></ul>
Dummy Variables in Python	<ul style="list-style-type: none"><li>• Python automatically assigns dummies to categorical variables in <code>ols()</code></li><li>• Use <code>Treatment()</code> to change the base category for modeling</li></ul>



Multiple Linear Regression

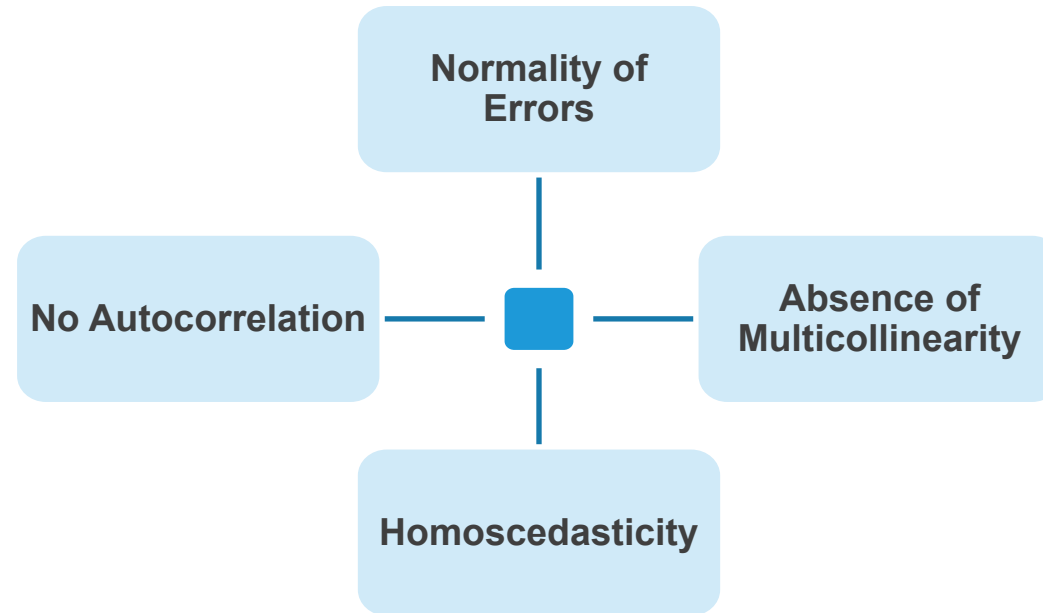
Multicollinearity problem

# Contents

1. Key Assumptions of Multiple Linear Regression
2. Understanding The Problem of Multicollinearity
3. Detecting Multicollinearity – Variance Inflation Factor
4. Detecting Multicollinearity in Python
5. Multicollinearity – Remedial Measures

# Key Assumptions of Multiple Linear Regression

Multiple Linear Regression makes four key assumptions



Violations of these assumptions may result in biased variable relationships, over or under-estimation of parameters (i.e., biased standard errors), and unreliable confidence intervals and significance tests

# Problem of Multicollinearity

Multicollinearity exists if there is strong linear relationship among the independent variables

Multicollinearity has two serious consequences:

## 1. Highly Unstable Model Parameters

As standard errors of their estimates are inflated

## 2. Model Fails to Accurately Predict for Out of Sample Data

Therefore, it is important to check for Multicollinearity in regression analysis



# Detecting Multicollinearity Through VIF

VIF (Variance Inflation Factor) Method:

Dependent Variable : Y

Independent variables : X1, X2, X3, X4

Dependent Variable	Independent Variables	$R^2$	$1 - R^2 = \text{Tolerance}$	$\text{VIF} = 1/(\text{Tolerance})$
X1	X2, X3, X4			
X2	X1, X3, X4			
X3	X1, X2, X4			
X4	X1, X2, X3			

Any VIF > 5, indicates presence of Multicollinearity

# Detecting Multicollinearity in Python

#Importing the Data, Fitting Linear Model

```
import pandas as pd
perindex=pd.read_csv("Performance Index.csv")

import statsmodels.formula.api as smf
jpimodel=smf.ols('jpi~aptitude+tol+technical+general',data=perindex).fit()
```

#Variance Inflation Factor

```
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Break data into left and right hand side; y and X
y, X = dmatrices('jpi ~ aptitude + tol + technical +general',
data=perindex, return_type="dataframe")
```

- *patsy is a library that helps in converting data frames into design matrices.*
- *dmatrices Construct two design matrices using specified formula.By convention, the first matrix is the "y" data, and the second is the "x" data.*
- *variance\_inflation\_factor() requires a design matrix as input to calculate vif.*



We use the same dataset "Performance Index" which was used in previous ppt

# Detecting Multicollinearity in Python

```
# Calculating VIF & getting vif with their corresponding variable  
# name
```

```
vif = pd.Series([variance_inflation_factor(X.values, i) for i in  
range(X.shape[1])], index=X.columns)
```

```
vif
```

*variance\_inflation\_factor() calculates VIFs.*

```
# Output
```

```
Intercept    143.239081  
aptitude      1.179906  
tol           1.328205  
technical     2.073907  
general       2.024968  
dtype: float64
```

*Interpretation :  
All VIFs are less than 5, Multicollinearity is not present.*

# Multicollinearity – Remedial Measures

The problem of Multicollinearity can be solved by different approaches:

Drop one of the independent variables, which is explained by others

Use Principal Component Regression in case of severe Multicollinearity

Use Ridge Regression



Dropping a variable may not be a good idea if many VIFs are large. Principal Component Method will be discussed in detail under Data Reduction and Segmentation



# Case Study - Modelling Resale Price of Cars

## Background

- A car garage has old cars for resale. They keep records for different models of cars and their specifications.

## Objective

- To predict the resale price based on the information available about the engine size, horse power, weight and years of use of the cars

## Available Information

- Records -26
- Independent Variables: engine size, horse power, weight and years
- Dependent Variable: resale price

# Data Snapshot

ridge regression  
Dependent variable { data Independent variables }

MODEL	RESALE PRICE	ENGINE SIZE	HORSE POWER	WEIGHT	YEARS
Daihatsu Cuore	3870	846	32	650	2.9
Suzuki Swift 1.0 GL	4163	993	39	790	2.9
Fiat Panda Mambo L	3490	899	29	730	3.1
Vauxhall Corsa 1.0	5711	1000	44	855	2.9

Observation	Columns	Description	Type	Measurement	Possible values
	MODEL	Model of the car	character	-	-
	RESALE PRICE	Resale price	numeric	Euro	positive values
	ENGINE SIZE	Size of the engine	numeric	cc	positive values
	HORSE POWER	Power of the engine	numeric	kW	positive values
	WEIGHT	Weight of the car	numeric	kg	positive values
	YEARS	Number of years in use	numeric	-	positive values

# Correlation Matrix

# Importing the Data

```
ridgedata=pd.read_csv("ridge regression data.csv")
```

# Graphical representation of data

# Install and load package “seaborn”

```
import seaborn as sns
```

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

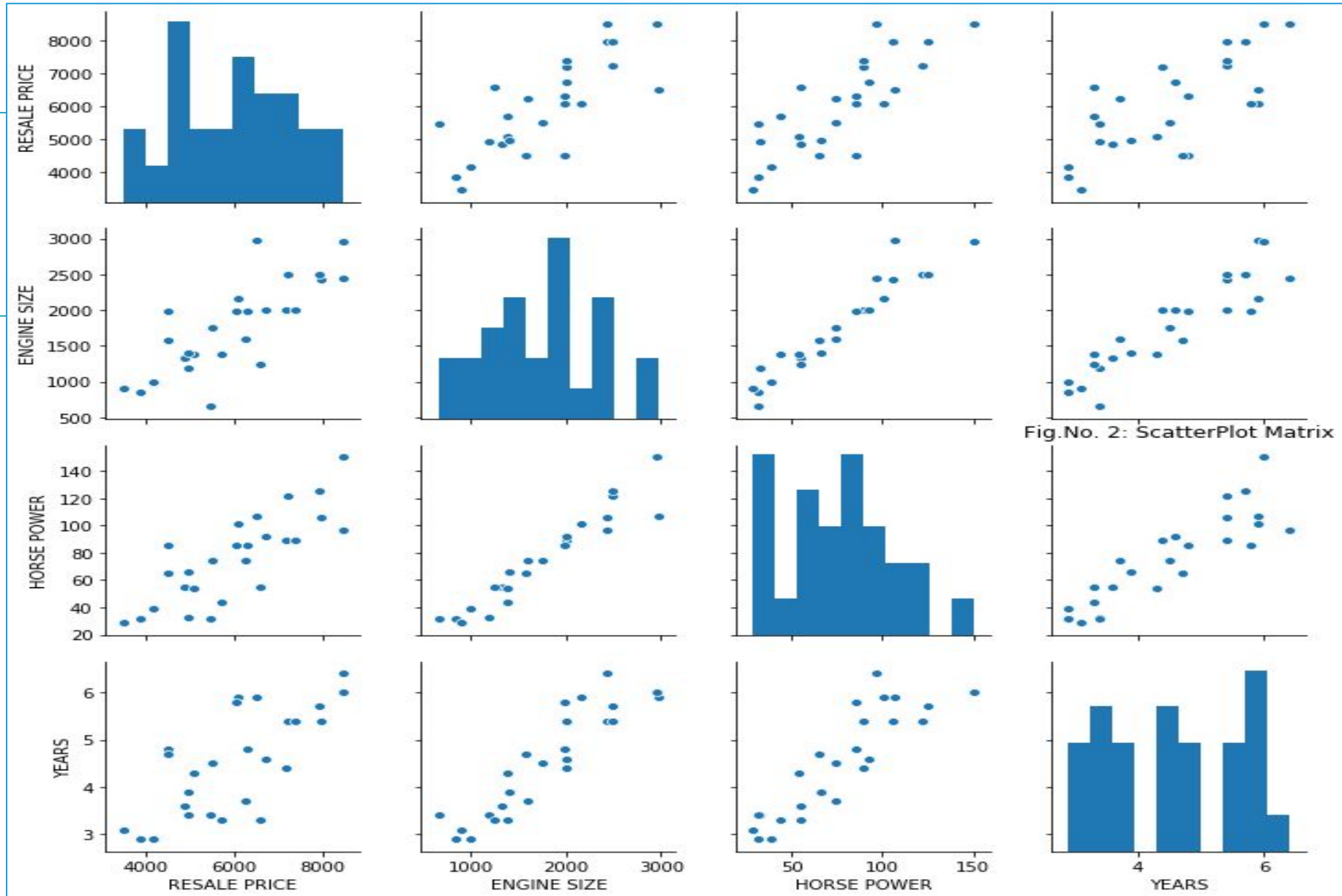
```
sns.pairplot(ridgedata[['MODEL', 'RESALE PRICE', 'ENGINE SIZE', 'HORSE  
POWER', 'YEARS']]);plt.title('Fig.No. 2: ScatterPlot Matrix')
```

*pairplot() in the package seaborn is used to plot the scatter plot matrix*

# Scatter Plot Matrix

# Output

*Interpretation :*  
□ The independent variables have high positive correlation among themselves .



# Detecting Multicollinearity in Python

#Importing the Data, Fitting Linear Model

```
ridgedata.columns = [c.replace(' ', '_') for c in ridgedata.columns]
model = smf.ols('RESALE_PRICE~ENGINE_SIZE+ HORSE_POWER + WEIGHT + YEARS',
data = ridgedata).fit()
```

*In pandas, the column names cannot contain spaces in between. Hence, before applying ols() remove spaces from column names wherever required.*

#Variance Inflation Factor

```
y, X = dmatrices('RESALE_PRICE~ENGINE_SIZE+ HORSE_POWER + WEIGHT +
YEARS', data=ridgedata, return_type="dataframe")
vif = pd.Series([variance_inflation_factor(X.values, i)for i in
range(X.shape[1])],index=X.columns)
vif
```

# Output

Intercept	26.193279
ENGINE_SIZE	15.759113
HORSE_POWER	12.046734
WEIGHT	9.113045
YEARS	13.978640
dtype:	float64

*Interpretation:*

*VIF values for all the variables are greater than 5, hence we can conclude that there exist Multicollinearity between the independent variables.*

# Quick Recap

This session explained the **problem of Multicollinearity**, along with its consequences and remedial measures:

## Multicollinearity Exists

- When independent variables have strong linear relationship

## Results in

- Unstable model parameters
- Inaccurate predictions for out of sample data

## Indicators

- High pairwise correlation
- Significant F value but very few significant t values

## Checking in Python

- Variance Inflation Factor  
**`variance_inflation_factor()`** function in package **`statsmodel`**

## Remedial Measures

- Drop variables
- Use Principal Component Regression
- Ridge regression

# Normality and Homoscedasticity Assumptions Influential Observations

# Contents

1. The Assumptions of Normality and Homoscedasticity
2. Residual v/s Predicted Plot in R
3. Q-Q plot of residuals
4. Shapiro Wilk test to assess Normality of Errors
5. Absence of Normality – Remedial Measure
6. Box cox Transformation in R
7. Influential Observations



# Normality and Homoscedasticity

- The errors in Multiple Linear Regression are assumed to follow Normal Distribution.
- If Normality of Errors is not true then statistical tests and associated P values based on F and t distribution are not reliable.
- **Homoscedasticity** describes a situation in which variance of error term is same across all values of the independent variables.
- In the absence of Homoscedasticity ( Or presence of Heteroscedasticity) the standard errors of parameter estimates are incorrect.

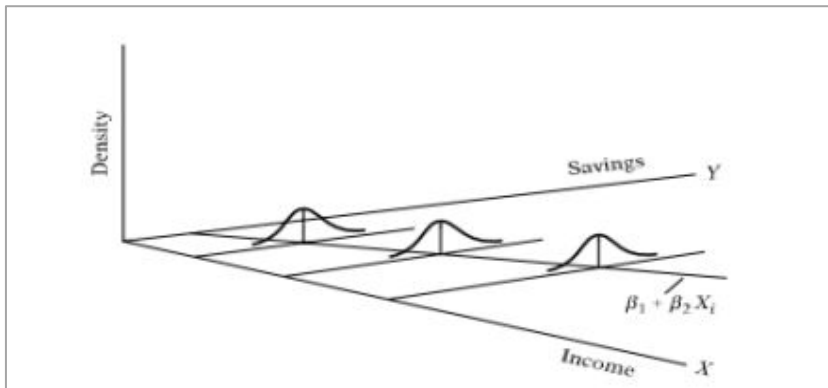
# Assumption of Homoscedasticity

- Variance of error term must be constant across the independent variables (defined by  $X$  values)

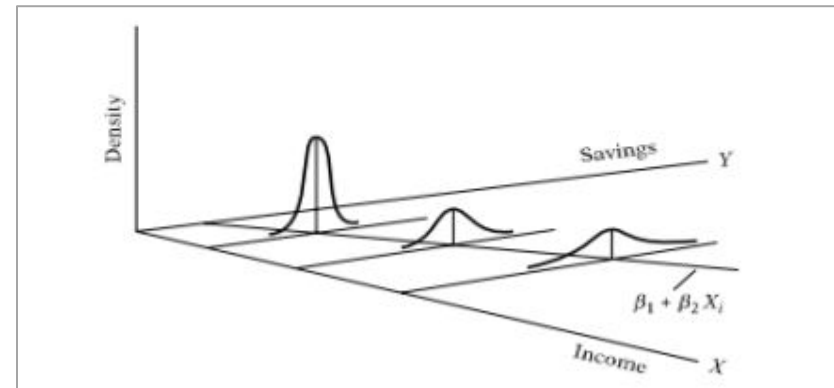
$$V(e_i/x_i) = \sigma^2 \text{ indicates homoscedasticity}$$

$$V(e_i/x_i) = \sigma_i^2 \text{ indicates heteroscedasticity}$$

Homoscedastic Errors

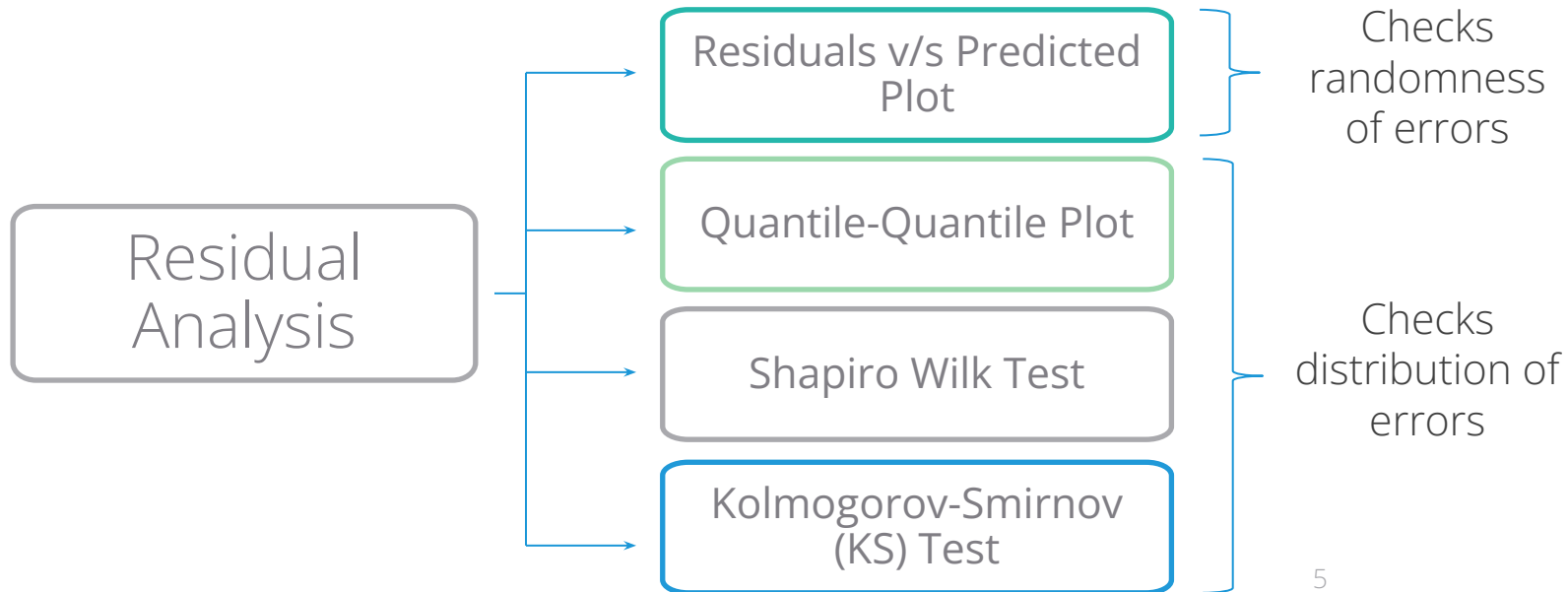


Heteroscedastic Errors



# Residual Analysis

$$\text{Observed Value} - \text{Predicted value} = \text{Residual}$$



# Residual Analysis for Performance Index Data

Continuing with the “Performance Index” data,

- **Model** job performance index ( **jpi** ) based on aptitude score ( **aptitude** ), test of language ( **tol** ), technical knowledge ( **technical** ) and general information ( **general** )
- Get fitted values and residuals.
- Analyse the distribution of residuals

# Residual v/s Predicted Plot in Python

#Importing the Data, Fitting Linear Model and Calculating Fitted Values and Residuals

```
import pandas as pd
perindex= pd.read_csv("Performance Index.csv")

import statsmodels.formula.api as smf
jpimodel = smf.ols('jpi ~ tol + aptitude + technical +general',
data=perindex).fit()

perindex = perindex.assign(pred=pd.Series(jpimodel.fittedvalues))
perindex = perindex.assign(res=pd.Series(jpimodel.resid))
```

- *ols() fits a linear regression.*
- *fittedvalues() and resid() fetch fitted values and residuals*

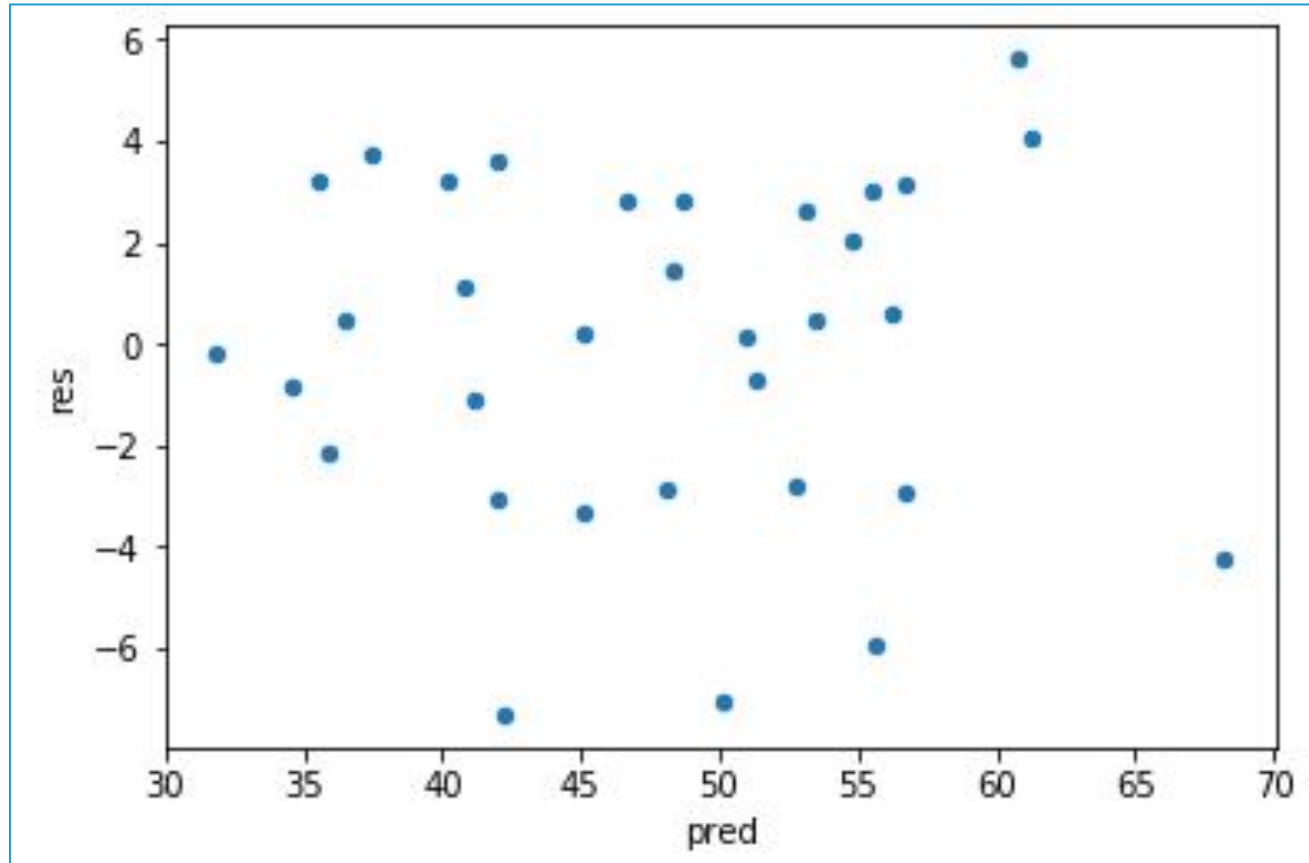
#Residuals v/s Predicted Plot

```
perindex.plot.scatter(x='pred', y='res')
```

*.plot.scatter() is used to obtain scatter plot of predicted values against residuals.*

# Residual v/s Predicted Plot in Python

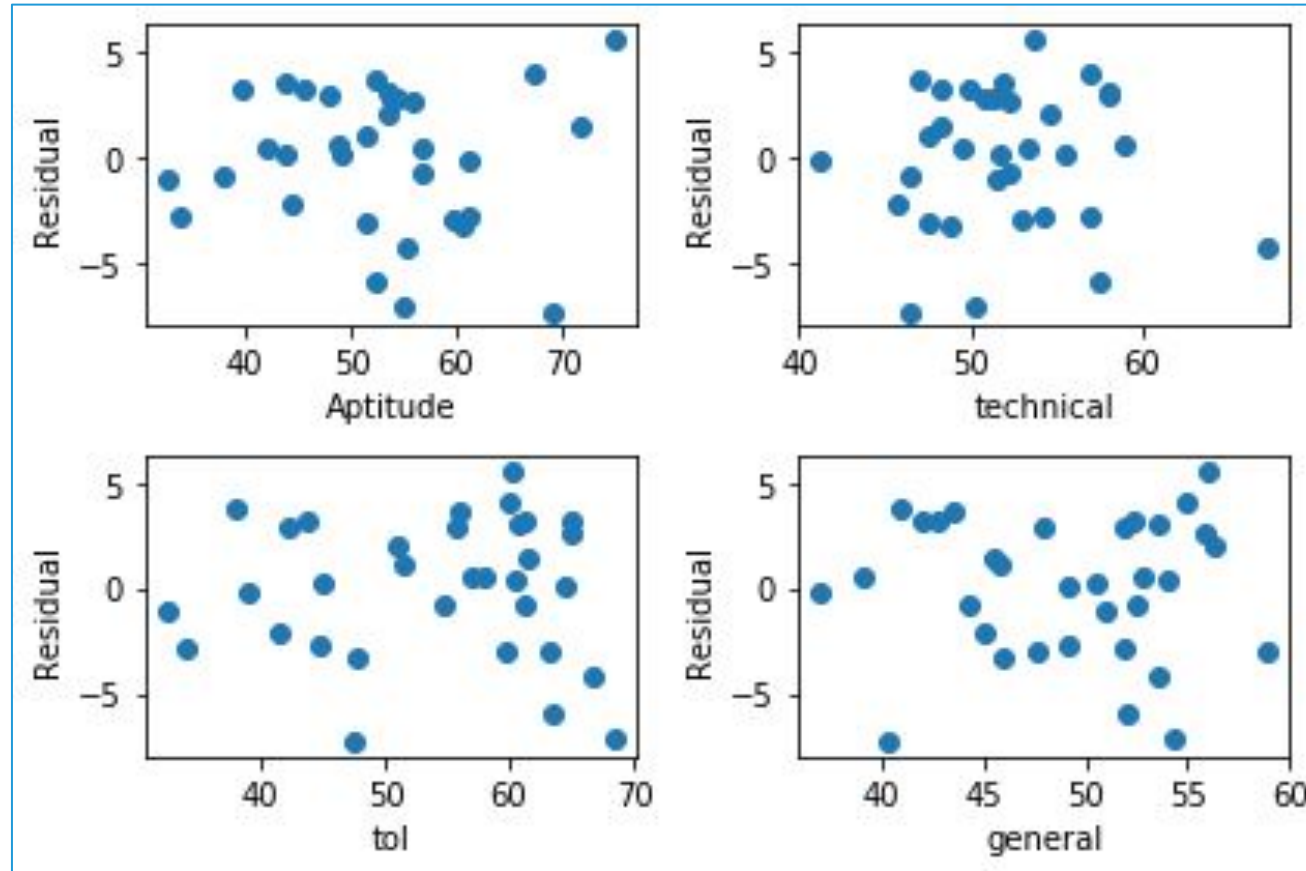
# Output



*Interpretation:*

- *Residuals in our model are randomly distributed which indicates presence of Homoscedasticity*

# Residual v/s Independent variables Plot in Python



*Interpretation:*

- *Residuals in our model are randomly distributed which indicates presence of Homoscedasticity*

# QQ Plot

- The Quantile-Quantile (QQ) Plot is a powerful graphical tool for assessing normality.
- Quantiles are calculated using sample data and plotted against expected quantiles under Normal distribution.

High Correlation between Sample Quantiles and  
Theoretical Quantiles



Normalit  
y

- If the data are truly sampled from a Gaussian (Normal) distribution, the QQ plot will be linear.



# QQ Plot in Python

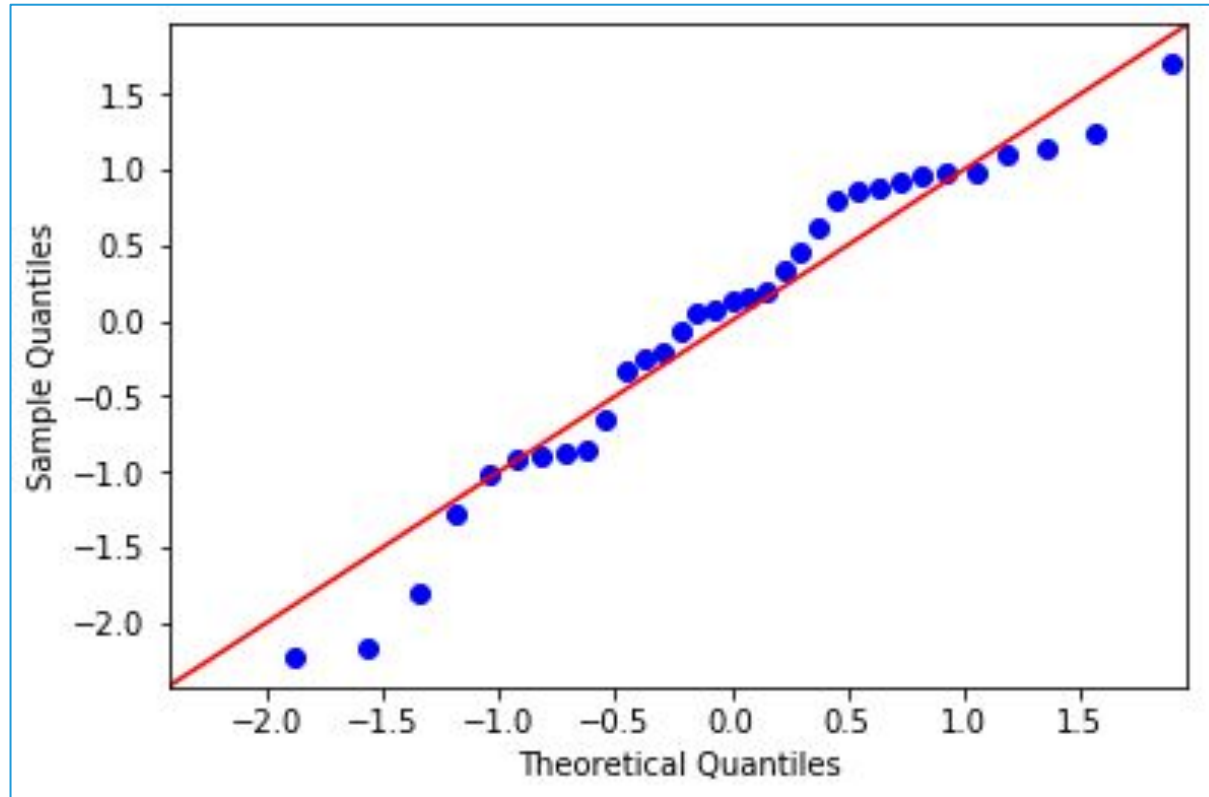
#QQ Plot

```
import statsmodels.api as sm  
fig = sm.graphics.qqplot(perindex.res, line='45', fit=True)
```

- *qqplot() produces a plot with theoretical quantiles on x axis against the sample quantiles on y axis. Column for which normality is being tested is specified in the first argument.*
- *line= is an argument that adds reference line to the qqplot. Here it adds a 45-degree line*
- *fit=True indicates, parameters are fit using the distribution's fit() method.*

# QQ Plot in Python

# Output



*Interpretation:*

- *Most of these points are close to the line except few values indicating no serious deviation from Normality.*

# Shapiro Wilk Test

Objective	To <b>correlate</b> , sample ordered values with expected Normal scores in order to <b>test normality of the sample</b>
-----------	---

Null Hypothesis ( $H_0$ ): Sample is drawn from Normal Population

Alternate Hypothesis ( $H_1$ ): Not  $H_0$

Test Statistic	
Decision Criteria	Reject the null hypothesis if p-value < 0.05

# Shapiro Wilk Test in Python

```
# Shapiro Wilk Test
```

```
import scipy as sp  
sp.stats.shapiro(perindex.res)
```

*shapiro() from scipy package, returns correlation coefficient  $w$  and  $p$ -value.*

```
# Output
```

```
(0.9498621821403503, 0.1318102478981018)
```

*Interpretation:*

*$p$ -value > 0.05, Do not reject  $H_0$ . Normality can be assumed.*

# Absence of Normality – Remedial Measure

Mathematical Transformation of the dependent variable is used as a remedial measure in case of serious departure from Normality.

Typically Log Transformation is used. However, there is general transformation called as Box Cox Transformation given as :

- Box Cox transformation

$$Y^* = \frac{Y^\lambda - 1}{\lambda} \quad \lambda \neq 0$$
$$= \log Y \quad \lambda = 0$$

Where Y is the response variable

- R can automatically detect the optimum  $\lambda$  using **boxcox()** in package MASS

# Influential Observation

- An **influential observation** is an observation whose deletion from the dataset would noticeably change the result of the calculation.
- In particular, in regression analysis an influential point is one whose deletion has a large effect on the parameter estimates.

# Cook's Distance Method

Cook's distance measures the effect of deleting a given observation.

Let  $D_i$  be the Cook's distance for observation  $i$ .

$$D_i = \frac{\sum_{j=1}^n (\hat{Y}_j - \hat{Y}_{j(i)})^2}{p \text{ MSE}}$$

$\hat{Y}_j$  = prediction from the full regression model for observation  $j$

$\hat{Y}_{j(i)}$  = prediction of  $j^{\text{th}}$  observation from a refitted model after removing  $i^{\text{th}}$  observation

$MSE$  = mean square error of the regression model

$p$  = number of fitted parameters in the model

Cut off to indicate influential observation,

- Simple operational guideline  $D_i > 1$
- Alternative  $D_i > 4/n$ , where  $n$  is the number of observations



**It is recommended to check model performance by excluding highly influential observation**

# DFBETAs

DFBETA  
Statistics



DFBETA measures the difference in each parameter estimate with and without a specific observation. There is a DFBETA for each data point and for each parameter estimate.

Large  
values of  
DFBETAs

Indicate



Observations are influential in  
estimating a given parameter

- Cut off to indicate influential observation,
- general cut off value recommended is 2
  - size adjusted cut off is taken to be  $2/\sqrt{n}$



# Finding Influential Observations in Python

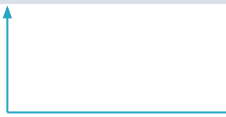
#Importing the Data

```
import pandas as pd
perindex=pd.read_csv("Performance Index.csv")

import statsmodels.formula.api as smf
jpimodel=smf.ols('jpi ~ aptitude + tol + technical +general',
data=perindex).fit()
```

#Finding Influential Observations

```
influence = jpimodel.get_influence()
influence.summary_frame()
```

- 
- *Influence is an object calling the method `get_influence()` which in turn allows us to call various measures of influence.*
  - *`summary_frame()` calls a dataframe of 6 influence measures - Cook's Distance, Standardized residuals, `dffits`, `dfbetas` among others.*

# Finding Influential Observations in Python

# Output

	dfb_Intercept	dfb_apptitude	dfb_tol	dfb_technical	dfb_general	cooks_d	standard_resid	hat_diag	dffits_internal	student_resid	dffits
0	0.122740655	-0.148903135	0.12930015	0.111295503	-0.231928116	0.027053557	1.070156154	0.105636531	0.367787693	1.073046027	0.368780874
1	-0.06974523	0.116652494	0.133587819	0.02982789	-0.09548941	0.010912673	-0.364255404	0.291400351	-0.233588024	-0.358542217	-0.229924297
2	0.007297141	-0.008657864	0.004774056	0.014949118	-0.026376846	0.000463024	-0.206575027	0.051460374	-0.048115671	-0.203007405	-0.047284697
3	-0.156960588	0.094733386	-0.173852663	0.214043541	-0.081100685	0.018150707	-0.899495717	0.100854509	-0.301253273	-0.896332474	-0.30019386
4	0.053857876	-0.010940236	0.00867167	-0.040889003	0.003662146	0.001250165	0.323269275	0.056438877	0.079062145	0.31803818	0.077782773
5	0.244563668	-0.167648781	-0.058829828	0.006424002	-0.120351526	0.0247544	0.95727578	0.118994501	0.351812453	0.9557968	0.351268907
6	-0.02187633	-0.017309924	0.012271264	0.006428915	0.013869655	0.000302058	-0.06455514	0.266006198	-0.03886248	-0.063396607	-0.038165038
7	-0.168200899	0.013234244	0.04743142	0.175275244	-0.069100347	0.01545173	0.918453917	0.083902352	0.277954402	0.915804589	0.277152627
8	-0.00893895	0.000748261	0.010029123	-0.012775933	0.020594356	0.000259695	0.13315947	0.068233348	0.03603436	0.130801426	0.035396249
9	0.112048775	-0.224748965	0.240141498	-0.17548137	0.078322088	0.023475703	-0.842281862	0.141964282	-0.342605483	-0.837785996	-0.340776751
10	-0.180550626	0.079534465	0.074017536	0.120829488	-0.057492908	0.01106923	-0.644822905	0.117472155	-0.235257629	-0.63795804	-0.232753046
11	-0.340527019	0.322925945	-0.062977416	0.148776604	0.044546954	0.044366369	1.221464622	0.129438005	0.470990281	1.232747401	0.475340861
12	-0.002789222	0.152126452	0.050719785	-0.020241173	-0.060819239	0.009463569	0.461241654	0.181948526	0.217526652	0.454660862	0.214423078
13	0.035353405	-0.027136734	0.033243364	0.016690271	-0.055540496	0.001256856	0.156009829	0.20521207	0.079273454	0.153265238	0.077878842
14	-0.060996651	1.62E-05	-0.079422031	-0.034363374	0.147870676	0.008619132	0.603984002	0.105654499	0.207594942	0.597002274	0.205195257
15	-0.026013807	-0.056282646	0.145739562	-0.191264061	0.222520013	0.020491421	0.791281772	0.140624982	0.320089214	0.78585952	0.317895805
16	0.005763587	0.05225852	-0.223037193	-0.045632185	0.148853441	0.018578284	0.85294471	0.113226147	0.30478094	0.848673035	0.303254552
17	-0.470812698	0.716128082	-0.106108271	-0.101654484	0.324161247	0.172591001	1.78156995	0.213764291	0.928953715	1.8579384	0.968774074
18	-0.002557978	-0.004065261	0.008450592	0.005075285	-0.005409883	4.13E-05	0.044548271	0.094144224	0.014361446	0.043747084	0.01410316
19	-0.052126913	-0.182788497	0.123094282	7.18E-05	0.058590983	0.01661995	-0.969160992	0.081281261	-0.288270271	-0.968072934	-0.287946635
20	-0.097589025	0.108404591	-0.09966008	0.077516309	-0.005991789	0.004816919	-0.272865	0.244414631	-0.155192119	-0.268305076	-0.152598659

## Interpretation

*One can use the threshold of  $4/n$  where  $n$  is sample size and check cases with Cook's distance greater than the threshold.*

# Finding Influential Observations in Python

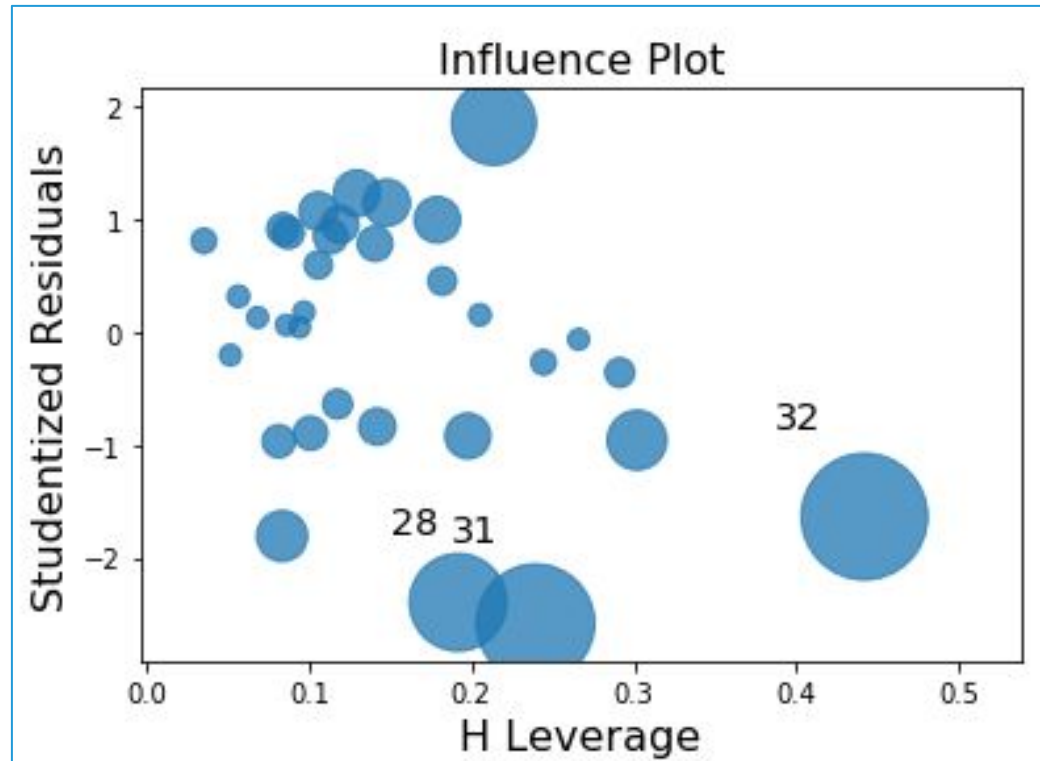
#Influence Plot

```
from statsmodels.graphics.regressionplots import *  
influence_plot(jpimodel, criterion = 'Cooks')
```

□ *influence\_plot() creates a “bubble” plot of Studentized residuals by hat values, with the areas of the circles representing the observations proportional to criteria specified (in this case Cook's distance).*

# Influence Plot in Python

# Output



*Interpretation:*

*The data points 28,31 and 32 are detected as influential observations.*

# Quick Recap

Normality Assumption	<ul style="list-style-type: none"><li>• Error terms should be normally distributed</li></ul>
Homoscedasticity	<ul style="list-style-type: none"><li>• Errors should have constant variance across X values</li></ul>
Residual v/s Predicted Plot	<ul style="list-style-type: none"><li>• Ideally, residuals should be randomly distributed</li></ul>
Residual v/s Independent variables Plot	<ul style="list-style-type: none"><li>• Ideally, residuals should be randomly distributed</li></ul>
QQ Plot	<ul style="list-style-type: none"><li>• Used to check if errors follow Normal distribution</li></ul>
Shapiro Wilk Test	<ul style="list-style-type: none"><li>• Test for Normality assessment of errors</li></ul>

# Quick Recap

## Box Cox Transformation

- Transforming non normal response to normal

## Influential Observations in Python

- **get\_influence()** produces object giving influential observations by different measures
- **Influence\_plot()** creates a “bubble” plot of Studentized residuals by hat values, with the areas of the circles representing the observations proportional to Cook’s distances

# Multiple Linear Regression

## Cross Validation - I

# Contents

1. Cross Validation in Predictive Modelling
2. Model Fitting
3. Hold-out Cross Validation



# Cross Validation in Predictive Modeling

Cross Validation is a  
process of evaluating the model on  
'Out of Sample' data

- Model performance measures such as R-squared or Root Mean Squared Error (RMSE) tend to be optimistic on 'In sample data'
- Model performance on out of sample data gives more realistic picture of model performance.

Cross validation is important because although a model is built on historical data, ultimately it is to be used on future data. However good the model, if it fails on out of sample data then it defeats the purpose of predictive modeling.

# Cross Validation in Predictive Modeling

There are different approaches to cross validation. Five most important of them are:

Hold-Out  
Validation

K-Fold Cross  
Validation

Repeated K-Fold  
Cross Validation

Leave-One-Out  
Cross Validation  
(LOOCV)

Re-sampling  
Validation Method  
(Bootstrap  
Method)

# Case Study – Modeling Motor Insurance Claims

## Background

- A car insurance company collects range of information from their customers at the time of buying and claiming insurance. The company wishes to check if any of this information can be used to model and predict claim amount

## Objective

- To model motor insurance claim amounts based on vehicle related information collected at the time of registering and claiming insurance

## Available Information

- Sample size is 1000
- Independent Variables: Vehicle Information – Vehicle Age, Engine Capacity, Length and Weight of the Vehicle
- Dependent Variable: Claim Amount

# Data Snapshot

Motor_Claim					
Independent variables				Dependent variable	
Observations	vehage	CC	Length	Weight	claimamt
	4	1495	4250	1023	72000
	2	1061	3495	875	72000
	2	1405	3675	980	50400
	7	1298	4090	930	39960
	2	1495	4250	1023	106800
	1	1086	3565	854	69592.8
Columns	Description	Type	Measurement	Possible values	
vehage	Age of the vehicle at the time of claim	integer	Years	positive values	
CC	Engine capacity	numeric	cc	positive values	
Length	Length of the vehicle	numeric	mm	positive values	
Weight	Weight of the vehicle	numeric	kg	positive values	
claimamt	Claim amount	numeric	INR	positive values	

# Data Visualization

#Importing the Data

```
import pandas as pd  
motor=pd.read_csv('Motor_Claims.csv')
```

# Install package “seaborn” if not installed previously  
# Obtain scatter plot matrix

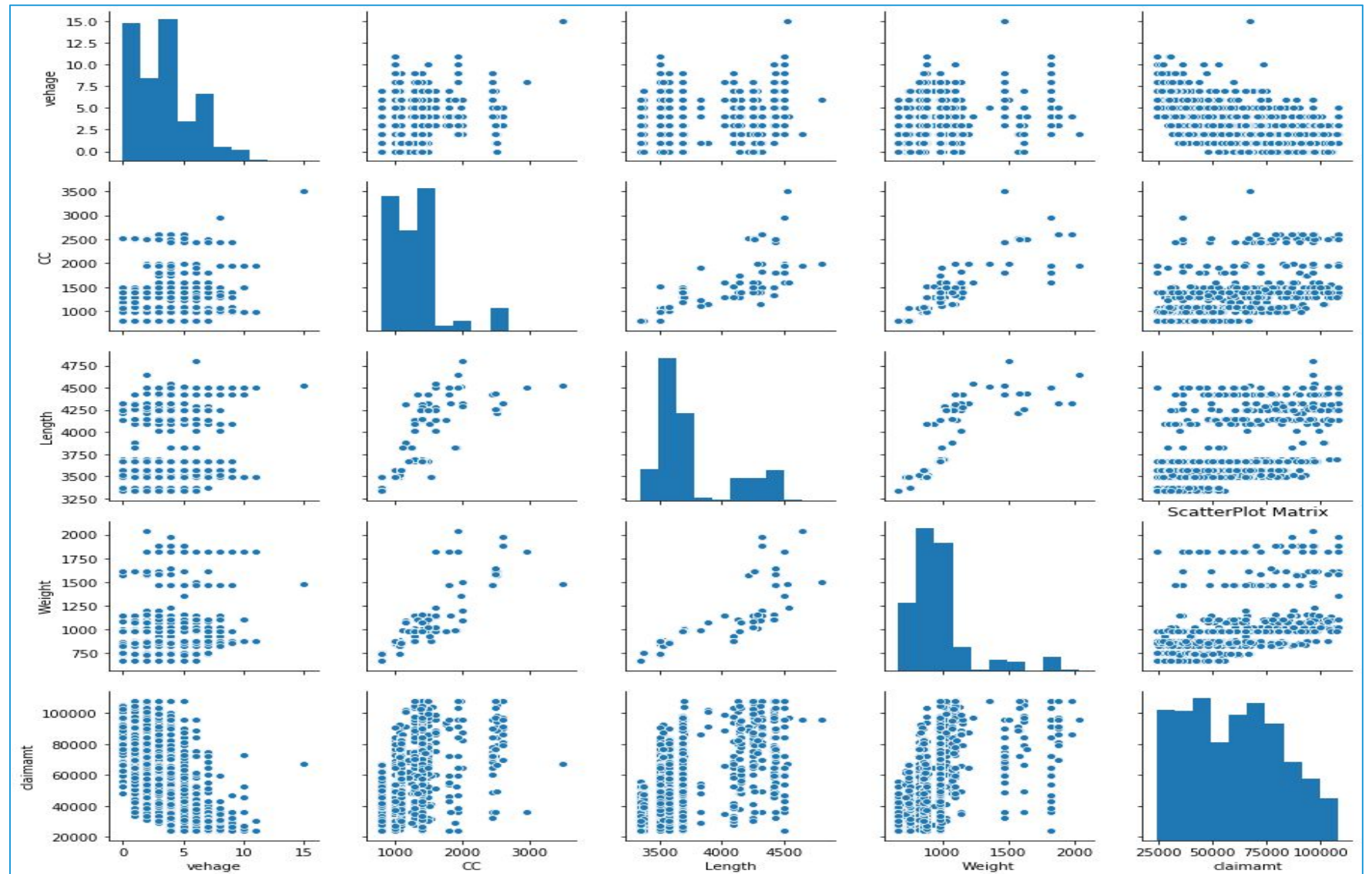
```
import seaborn as sns  
import matplotlib.pyplot as plt  
  
sns.pairplot(motor);plt.title('ScatterPlot Matrix')
```

*Using the pairplot function in the seaborn library to get a scatter plot of the variables in the data set*

# Scatter Plot

# Output

*Interpretation :  
The scatter plot matrix gives an indication of multicollinearity.*



# Modeling Using ols Function

# Linear regression model

```
import statsmodels.formula.api as smf
motormodel = smf.ols('claimamt~Length+CC+vehage+Weight', data=motor).fit()
motormodel.summary()
```

# Output

OLS Regression Results						
=====						
Dep. Variable:	claimamt		R-squared:	0.738		
Model:	OLS		Adj. R-squared:	0.737		
Method:	Least Squares		F-statistic:	700.3		
Date:	Fri, 25 Oct 2019		Prob (F-statistic):	1.83e-287		
Time:	16:38:15		Log-Likelihood:	-10754.		
No. Observations:	1000		AIC:	2.152e+04		
Df Residuals:	995		BIC:	2.154e+04		
Df Model:	4					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]
-----						
Intercept	-5.477e+04	5569.375	-9.833	0.000	-6.57e+04	-4.38e+04
Length	35.4607	1.990	17.824	0.000	31.557	39.365
CC	15.4133	2.114	7.292	0.000	11.265	19.561
vehage	-6637.2134	154.098	-43.071	0.000	-6939.607	-6334.820
Weight	-16.2547	3.678	-4.420	0.000	-23.472	-9.038
=====						
Omnibus:	7.335	Durbin-Watson:	2.094			
Prob(Omnibus):	0.026	Jarque-Bera (JB):	9.587			
Skew:	-0.058	Prob(JB):	0.00828			
Kurtosis:	3.466	Cond. No.	6.33e+04			
=====						

*Interpretation:*

*All independent variables in the model are significant.*

# Detecting Multicollinearity

```
# Obtaining vif
```

```
from patsy import dmatrices
from statsmodels.stats.outliers_influence import
variance_inflation_factor
```

```
# Break into left and right hand side; y and X
y, X = dmatrices('claimamt~Length+CC+vehage+Weight', data=motor,
return_type="dataframe")
```

```
#Calculating VIF
```

```
vif = pd.Series([variance_inflation_factor(X.values, i)for i in
range(X.shape[1])],index=X.columns)
vif
```

```
Intercept    240.261728
Length        3.396171
CC            5.881428
vehage        1.038357
Weight        6.552811
dtype: float64
```

□ *variance\_inflation\_factor* in library *statsmodels* gives the VIFs of the independent variables in the regression model.

*Interpretation:*

□ *CC and Weight have VIF >5*



# Re- Modeling

```
# New model
```

```
motormodel1 = smf.ols('claimamt~Length+CC+vehage', data=motor).fit()  
motormodel1.summary()
```

*New model after removing weight to remove multicollinearity.*

```
# Output of the new model
```

OLS Regression Results						
=====						
Dep. Variable:	claimamt		R-squared:	0.733		
Model:	OLS		Adj. R-squared:	0.732		
Method:	Least Squares		F-statistic:	910.3		
Date:	Thu, 31 Oct 2019		Prob (F-statistic):	8.79e-285		
Time:	12:48:57		Log-Likelihood:	-10764.		
No. Observations:	1000		AIC:	2.154e+04		
Df Residuals:	996		BIC:	2.156e+04		
Df Model:	3					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]
-----						
Intercept	-4.92e+04	5475.151	-8.985	0.000	-5.99e+04	-3.85e+04
Length	32.0652	1.852	17.312	0.000	28.431	35.700
CC	8.6886	1.481	5.867	0.000	5.783	11.595
vehage	-6638.0765	155.525	-42.682	0.000	-6943.270	-6332.883
=====						
Omnibus:	10.930	Durbin-Watson:	2.081			
Prob(Omnibus):	0.004	Jarque-Bera (JB):	15.892			
Skew:	-0.072	Prob(JB):	0.000354			
Kurtosis:	3.600	Cond. No.	5.99e+04			
=====						

*Interpretation:  
All independent variables in  
the model are significant.*



Dropping one independent variable is one of the remedial measures to adjust for multicollinearity (when not many variables are multicollinear). As weight had the maximum VIF value, it is excluded from the model to adjust for multicollinearity.

# VIF of New Model

# VIF

```
# Break into left and right hand side; y and X  
y, X = dmatrices('claimamt~Length+CC+vehage', data=motor,  
return_type="dataframe")
```

*Getting VIFs of the independent variables in the new model*

```
#Calculating VIF  
vif = pd.Series([variance_inflation_factor(X.values, i)for i in  
range(X.shape[1])],index=X.columns)  
vif
```

# VIFs of variables in the new model

Intercept	227.959103
Length	2.889718
CC	2.833931
vehage	1.038355
dtype: float64	

*Interpretation:  
All VIF s are <5.*

# RMSE of the Model

# RMSE of the model

```
motor=motor.assign(res=pd.Series(motormodel1.resid))
```

```
from math import sqrt
```

```
RMSE = sqrt((motor['res']**2).mean())
```

```
RMSE
```

# Output

11444.512861029943

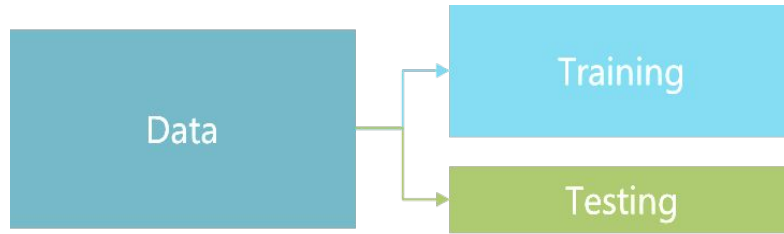
*Interpretation :*

*RMSE for the model is 1144.51*



RMSE of this model will be used later to measure the performance of the model on cross validation

# Hold-Out Validation



In Hold-Out validation method, available data is split into two non-overlapped parts: 'Training Data' and 'Testing Data'

- The model is,
  - Developed using training data
  - Evaluated using testing data

Training data should have more sample size. Typically 70%-80% data is used for model development

# Hold Out Validation in Python

```
# Import train_test_split  
# Creation of Datasets for Validation
```

```
import numpy as np  
from sklearn.model_selection import train_test_split  
import statsmodels.formula.api as smf
```

```
motor=pd.read_csv('Motor_Claims.csv')
```

```
motor_train, motor_test = train_test_split(motor, test_size=0.2,  
random_state=0)
```

*We use the function train\_test\_split to split the data 80 -20, with test\_size=0.2 specifying that 20 % of data used as test data*

```
motor_train.shape  
motor_test.shape
```

*training and testing data sets for the Motor Insurance Data*

```
# Output
```

```
(800, 5)
```

*Dimension of training set*

```
(200, 5)
```

*Dimension of testing set*

# Hold Out Validation in Python

# RMSE of training data

```
motor_model=smf.ols('claimamt~vehage+CC+Length', data =  
motor_train).fit()  
  
motor_train=motor_train.assign(res=pd.Series(motor_model.resid))  
motor_train.head()  
  
RMSEtrain=pd.Series(np.sqrt((motor_train.res)**2).mean())  
RMSEtrain
```

# RMSE for testing data

```
motor_test=motor_test.assign(pred=pd.Series(motor_model.predict(motor_  
test)))  
motor_test=motor_test.assign(res=pd.Series(motor_test.claimamt -  
motor_test.pred))  
  
RMSEtest=pd.Series(np.sqrt((motor_test.res)**2).mean())  
RMSEtest
```



For testing data, since we work with predictors, we calculate residuals as observed values minus the predicted values

Note : Since data is randomly taken, output will differ slightly

# Hold Out Validation in Python

# Output

	vehage	CC	Length	Weight	claimamt	res
0	4	1495	4250	1023	72000.0	-3603.862083
2	2	1405	3675	980	50400.0	-17806.818626
3	7	1298	4090	930	39960.0	-8741.177051
4	2	1495	4250	1023	106800.0	18045.949781
6	4	796	3495	740	38400.0	-4537.892032

*res column gives the residual for the training set data*

11444.512861029943

*RMSE for the original data with all data points*

11456.386840541307

*RMSE for the training data*

10846.219628859553

*RMSE for testing data*

*Interpretations :*

- ▣ *Comparing RMSE of training and testing data shows not much difference between the two and also are in line with the RMSE of the original model. Thus we can say that the model is stable.*



**There is no rule of thumb for comparison of model performance. The only criterion is that performance measure for training and testing data should not be too diverse**

# Quick Recap

## Cross Validation – Meaning and Need

- Process of evaluating the model on 'Out of Sample' data
- Important because although a model is built on historical data, ultimately it is to be used on future data

## Hold Out Validation

- Data is split into training and testing. Model developed on training and validated on testing