0

**LAB MANUAL**

**FOR**

**Data Analytics and Visualization (CSL-601)**

**(SEM - VI)**

**Artificial Intelligence & Data Science**

**FH-2024**

SUBJECT TITLE: DATA ANALYTICS AND VISUALIZATION

CLASS: T.E. AI & DS SEM: VI

**Lab Objectives**

1. To effectively use graph libraries such as matplotlib/seaborn/excel plots.
2. To perform exploratory data analysis on a given dataset.
3. To fit a statistical model (Regression, ANOVA, ARIMA) on a given data set.
4. To apply suitable visualization techniques for identifying patterns, trends and outliers in large data set.

**Lab Outcomes**

1. Use graph libraries such as matplotlib/Seaborn/Excel plots.
2. Perform exploratory data analysis and prepare the data for fitting a model.
3. Build a statistical model (Regression, ANOVA, ARIMA) on a given dataset.
4. Apply suitable visualization techniques to get insights from a given data set.

**LO-PO-PSO Mapping**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **LO vs (PO, PSO) Mapping** | | | | | | | | | | | | | | |
| **PO** | | | | | | | | | | | | | **PSO** | |
| **CO** | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO10** | **PO11** | **PO12** | **PSO1** | **PSO2** |
| CSL601.1 | 2 | 1 | 1 | 1 | 1 | 3 | - | - | - | - | - | - | 1 | 1 |
| CSL601.2 | 2 | 2 | 2 | 1 | 2 | 2 | - | - | - | - | - | - | 1 | 2 |
| CSL601.3 | 2 | 2 | 2 | 2 | 2 | 2 | - | - | - | - | - | - | 2 | 2 |
| CSL601.4 | 2 | 2 | 2 | 2 | 3 | 1 | - | - | - | - | - | - | 3 | 3 |
| **AVG** | **2** | **1.75** | **1.75** | **1.5** | **2** | **2** | **-** | **-** | **-** | **-** | **-** | **-** | **1.75** | **2** |

**List of Experiments**

| **Sr. No.** | **Name of Experiment** | **BL** | **Lab Outcomes** |
| --- | --- | --- | --- |
| 1 | Getting introduced to graph libraries such as matplotlib/Seaborn/Excel plots. | L-2 | LO1 |
| 2 | Data Exploration: Knowing the data, Data preparation and Cleaning. | L-2 | LO2 |
| 3 | To understand and implement visualization of data. | L-2 | LO4 |
| 4 | To implement Correlation and Covariance. | L-3 | LO4 |
| 5 | To understand and implement Hypothesis Testing. | L-4 | LO3 |
| 6 | To implement Simple Linear Regression. | L-3 | LO3 |
| 7 | To implement Multiple Linear Regression. | L-3 | LO3 |
| 8 | To implement Time Series Analysis. | L-4 | LO3 |
| **Additional Experiments:** | | | |
| 9 | To explore and understand data visualization using Tableau. | L-5 | L-2 |
| 10 | Develop a small Data Analysis Project for a real time application. | L-6 | L-3 & L-4 |

Prof. Anagha S. Dhavalikar Dr. Pramod Bhavarthe

**Subject In-charge HOD**

EXPERIMENT NO. 1

**Aim:** Introduction to graph libraries such as matplotlib/Seaborn/Excel plots.

**1. Matplotlib:**

Matplotlib is a widely used plotting library for Python. It's powerful and provides a wide variety of plots like line plots, scatter plots, histograms, bar charts, etc. It's highly customizable, allowing users to control almost every aspect of the plot.

Example using Matplotlib in Python:

**Code:**

import matplotlib.pyplot as plt

# Sample data

x = [1, 2, 3, 4, 5]

y = [2, 4, 6, 8, 10]

# Creating a line plot

plt.plot(x, y)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title('Sample Line Plot')

plt.show()

**2. Seaborn:**

Seaborn is built on top of Matplotlib and provides a higher-level interface for creating attractive and informative statistical graphics. It simplifies many common visualization tasks and works well with Pandas dataframes.

Example using Seaborn in Python:

**Code:**

import seaborn as sns

import pandas as pd

# Sample data

data = pd.DataFrame({'X': [1, 2, 3, 4, 5], 'Y': [2, 4, 6, 8, 10]})

# Creating a scatter plot

sns.scatterplot(x='X', y='Y', data=data)

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.title('Sample Scatter Plot')

plt.show()

**3. ggplot2 (R Programming):**

ggplot2 is a powerful and popular plotting system in R that follows the Grammar of Graphics. It allows users to create sophisticated plots with ease and provides a high level of customization.

Example using ggplot2 in R:

library(ggplot2)

# Sample data

data <- data.frame(X = c(1, 2, 3, 4, 5), Y = c(2, 4, 6, 8, 10))

# Creating a scatter plot

ggplot(data, aes(x = X, y = Y)) +

geom\_point() +

labs(x = 'X-axis', y = 'Y-axis', title = 'Sample Scatter Plot')

```

# Create some variables

x <- 1:10

y1 <- x\*x

y2 <- 2\*y1

# Create a basic stair steps plot

plot(x, y1, type = "S")

# Show both points and line

plot(x, y1, type = "b", pch = 19,

col = "red", xlab = "x", ylab = "y")

# Create a first line

plot(x, y1, type = "b", frame = FALSE, pch = 19,

col = "red", xlab = "x", ylab = "y")

# Add a second line

lines(x, y2, pch = 18, col = "blue", type = "b", lty = 2)

# Add a legend to the plot

legend("topleft", legend=c("Line 1", "Line 2"),

col=c("red", "blue"), lty = 1:2, cex=0.8)

**4. Excel:**

Excel is a widely used spreadsheet program that also provides various charting and graphing capabilities. Users can create different types of charts (line charts, bar charts, pie charts, etc.) by selecting data and using the Chart tools available in Excel.

To create a simple plot in Excel:

- Enter your data into cells.

- Select the data range.

- Go to the "Insert" tab and choose the type of chart you want to create.

These libraries/tools offer a range of functionalities and flexibility for creating visualizations based on your data and programming preferences. Explore them further to leverage their full potential in data visualization tasks.

**Learning Objectives:**

To understand the graph libraries.

**Conclusion/Learning outcome:**

The concept of graph libraries is studied and understood using matplotlib/Seaborn/Excel plots.

EXPERIMENT NO. 2

**Aim:** Data Exploration: Knowing the data, Data preparation and Cleaning

**Prior Concepts:**

Exploring data is a crucial step in understanding its characteristics, trends, and underlying patterns. Conducting experiments in data exploration involves various techniques and tools to gain insights into the dataset. Following are the different approaches to conduct a data exploration.

**1.** **Data Collection and Understanding:**

- Collect the dataset you want to explore. This could be a CSV file, database, or any structured/unstructured data source.

- Understand the data sources, variables, and their meanings. Look into data dictionaries or metadata to comprehend what each column represents.

**2. Data Cleaning and Preprocessing:**

- Check for missing values, duplicates, and outliers in the dataset.

- Handle missing values by imputation or deletion, depending on the nature of the data.

- Normalize or standardize data if necessary for certain algorithms or analyses.

**3. Statistical Summaries and Visualizations:**

- Calculate descriptive statistics (mean, median, mode, standard deviation, etc.) for numerical variables.

- Generate frequency counts for categorical variables.

- Create visualizations like histograms, box plots, scatter plots, and heatmaps to understand the distribution and relationships between variables.

**4. Exploratory Data Analysis (EDA):**

- Conduct correlation analysis to identify relationships between variables.

- Perform dimensionality reduction techniques like PCA (Principal Component Analysis) or t-SNE (t-Distributed Stochastic Neighbor Embedding) for visualization in lower dimensions.

- Cluster analysis to identify natural groupings within the data.

**5. Hypothesis Testing and Feature Engineering:**

- Formulate hypotheses about relationships or patterns within the data.

- Perform hypothesis tests to validate or reject these hypotheses.

- Create new features by transforming existing ones to improve model performance if working on a predictive modeling task.

**6. Interactive Exploration and Tools:**

- Utilize tools like Jupyter Notebooks, Pandas, Matplotlib, Seaborn, Plotly, or Tableau for interactive exploration.

- Utilize widgets or interactive visualization libraries for dynamic exploration of data subsets or trends.

**7. Documentation and Communication:**

- Document all findings, insights, and assumptions made during exploration.

- Create visualizations, summaries, and reports to communicate the insights gained.

- Prepare presentations or reports to share the findings with stakeholders or team members.

**8. Iterative Process:**

- Data exploration is often iterative. Revisit steps, try different techniques, and compare results to gain a comprehensive understanding of the dataset.

**9. Ethical Considerations:**

- Ensure ethical use of data, especially regarding privacy, biases, and the implications of insights drawn from the data.

**New Concept:**

**Data loading in R**

1. **CSV File**

# Load a CSV file

data <- read.csv("your\_file.csv")

# View the first few rows of the dataset

head(data)

# View summary statistics of the dataset

summary(data)

# Check the structure of the dataset

str(data)

# Check the dimensions of the dataset

dim(data)

**2.** **EXCEL File**

# Load an Excel file (assuming 'readxl' package is installed)

library(readxl)

data <- read\_excel("your\_file.xlsx")

# View the first few rows of the dataset

head(data)

# View summary statistics of the dataset

summary(data)

# Check the structure of the dataset

str(data)

# Check the dimensions of the dataset

dim(data)

**3.** **Identifying Missing values**

# Assuming 'data' is your dataframe

# Check for missing values in the entire dataset

colSums(is.na(data))

# Check for missing values in a specific column

sum(is.na(data$column\_name))

**4. Removing missing values**

# Remove rows with any missing values

data\_clean <- na.omit(data)

# Remove rows with missing values in specific columns

data\_clean <- data[complete.cases(data$column\_name), ]

Data preparation and cleaning involve various steps to ensure that the dataset is in a suitable format for analysis or modeling.

**Data Cleaning Steps:**

**1. Handling Missing Values:**

- Identify missing values using functions like `is.na()` or `complete.cases()`.

- Decide whether to remove missing values using `na.omit()` or impute them with methods like mean, median, mode, or using advanced imputation techniques from packages like `mice` or `missForest`.

**2. Handling Outliers:**

- Detect outliers using statistical methods like z-scores, IQR (Interquartile Range), or visualization techniques such as boxplots.

- Decide whether to remove outliers or transform them based on the context of your analysis.

**3. Dealing with Duplicates:**

- Check for and remove duplicate rows using `duplicated()` and `unique()` functions.

**Data Preparation Steps:**

**1. Feature Engineering:**

- Create new features from existing ones based on domain knowledge or insights.

- Transform variables (e.g., log transformation for skewed data) to improve the distribution of data.

**2. Standardization and Normalization:**

- Scale numerical variables to a standard scale using methods like `scale()` for standardization or min-max scaling for normalization.

- Normalize features to bring them on a similar scale, especially when using distance-based algorithms.

**3. Handling Categorical Variables:**

- Convert categorical variables to factors using `as.factor()` or one-hot encode them using techniques from packages like `dummies`.

- Handle ordinal variables appropriately by assigning levels.

**4. Data Splitting:**

- Split the dataset into training and testing subsets using functions like `sample()` or from packages like `caret` or `tidymodels`.

**5. Handling Date and Time Variables:**

- Convert date/time variables to appropriate formats using functions like `as.Date()` or `as.POSIXct()`.

**New Concepts:**

Example of R Code for Data Cleaning and Preparation:

# Handling missing values

data\_clean <- na.omit(data) # Removing rows with missing values

# OR

data$column\_name[is.na(data$column\_name)] <- mean(data$column\_name, na.rm = TRUE) # Imputing missing values with mean

# Handling outliers (example: using z-score)

threshold <- 3

data\_clean <- data[abs(scale(data$numeric\_column)) < threshold, ]

# Dealing with duplicates

data\_unique <- unique(data)

# Feature engineering (example: creating a new feature)

data$feature\_sum <- rowSums(data[, c("feature1", "feature2", "feature3")])

# Handling categorical variables (example: one-hot encoding)

library(dummies)

data <- dummy.data.frame(data, names = "categorical\_column")

# Data splitting (example: 70-30 split for training and testing)

library(caret)

set.seed(123)

train\_index <- createDataPartition(data$target\_variable, p = 0.7, list = FALSE)

train\_data <- data[train\_index, ]

test\_data <- data[-train\_index, ]

These steps ensure that your data is clean, formatted correctly, and ready for analysis or modeling tasks in R. Adjust these methods based on your specific dataset and analysis requirements.

**Learning Objectives:**

To understand the different techniques of Data exploration, Data preparation and Cleaning.

**Conclusion/Learning outcome:**

The use of different tools and commands for understanding the data are studied and implemented.The concept of Data preparation and Cleaning is understood and implemented in R language.

EXPERIMENT NO. 3

**Aim:** To understand and implement visualization of data.

**Prior Concepts:**

R provides numerous packages for data visualization. One of the most commonly used packages is **ggplot2**, which offers a flexible and powerful system for creating a wide variety of visualizations. Here are some basic examples of data visualization using **ggplot2**:

**New Concept:**

1. **Install and load ggplot2 Package:**

# Install ggplot2 if not already installed

install.packages("ggplot2")

# Load ggplot2 library

library(ggplot2)

1. **Histogram**

# Create a histogram of a numerical variable 'numeric\_column' from dataframe 'data'

ggplot(data, aes(x = numeric\_column)) +

geom\_histogram(binwidth = 5, fill = "skyblue", color = "black") +

labs(title = "Histogram of Numeric Column", x = "Values", y = "Frequency")

1. **Scatter plot**

# Create a scatter plot of 'numeric\_column1' against 'numeric\_column2'

ggplot(data, aes(x = numeric\_column1, y = numeric\_column2)) +

geom\_point(color = "blue") +

labs(title = "Scatter Plot", x = "X-axis Label", y = "Y-axis Label")

1. **Boxplot**

# Create a boxplot to visualize distribution of 'numeric\_column' across 'group\_column'

ggplot(data, aes(x = group\_column, y = numeric\_column)) +

geom\_boxplot(fill = "lightgreen", color = "black") +

labs(title = "Boxplot of Numeric Column by Group", x = "Groups", y = "Values")

1. **Barchart**

# Create a bar chart to visualize counts of categories in 'categorical\_column'

ggplot(data, aes(x = categorical\_column)) +

geom\_bar(fill = "orange") +

labs(title = "Bar Chart of Categorical Column", x = "Categories", y = "Count")

1. **Lineplot**

# Create a line plot to show trends over time using 'date\_column' and 'numeric\_column'

ggplot(data, aes(x = date\_column, y = numeric\_column)) +

geom\_line(color = "red") +

labs(title = "Line Plot Over Time", x = "Date", y = "Values")

**Learning Objectives:**

To understand the different techniques of Data Visualization.

**Conclusion/Learning outcome:**

Different techniques of data visualization are understood and implemented in R programming.

EXPERIMENT NO. 4

**Aim:** To implement Correlation and Covariance.

**Prior Concept:**

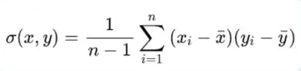
## Covariance

It’s a statistical term demonstrating a systematic association between two random variables, where the change in the other mirrors the change in one variable.

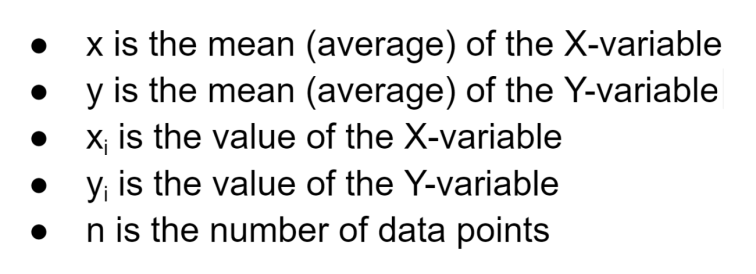
#### **Definition and Calculation of Covariance**

Covariance implies whether the two variables are directly or inversely proportional.

The covariance formula determines data points in a dataset from their average value. For instance, you can compute the Covariance between two random variables, X and Y, using the following formula:



Where,

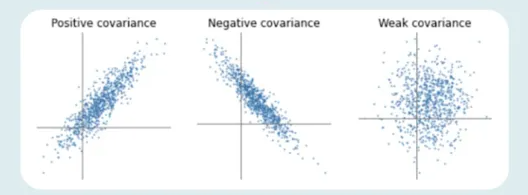


#### **Interpreting Covariance Values**

Covariance values indicate the magnitude and direction (positive or negative) of the relationship between variables. The covariance values range from -∞ to +∞. The positive value implies a positive relationship, whereas the negative value represents a negative relationship.

#### **Positive, Negative, and Zero Covariance**

The higher the number, the more reliant the relationship between the variables. Let’s comprehend each variance type individually:



**Positive Covariance**

If the relationship between the two variables is a positive covariance, they are progressing in the same direction. It represents a direct relationship between the variables. Hence, the variables will behave similarly.

The relationship between the variables will be positive Covariance only if the values of one variable (smaller or more significant) are equal to the importance of another variable.

**Negative Covariance**

A negative number represents negative Covariance between two random variables. It implies that the variables will share an inverse relationship. In negative Covariance, the variables move in the opposite direction.

In contrast to the positive Covariance, the greater of one variable correspond to the smaller value of another variable and vice versa.

**Zero Covariance**

Zero Covariance indicates no relationship between two variables.

#### **Significance of Covariance in Assessing Linear Relationship**

Covariance is significant in determining the linear relationship between variables. It suggests the direction (negative or positive) and magnitude of the relationship between variables.

A higher covariance value indicates a strong linear relationship between the variables, while a zero covariance suggests no ties.

#### **Limitations and Considerations of Covariance**

The scales of measurements influence the Covariance and are highly affected by outliers. Covariance is restricted to measuring only the linear relationships and doesn’t apprehend the direction or strength.

Moreover, comparing covariances across various datasets demand caution due to different variable ranges.

## Correlation

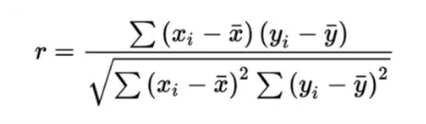
Unlike Covariance, correlation tells us the direction and strength of the relationship between multiple variables. Correlation assesses the extent to which two or more random variables progress in sequence.

#### **Definition and Calculation of Correlation Coefficient**

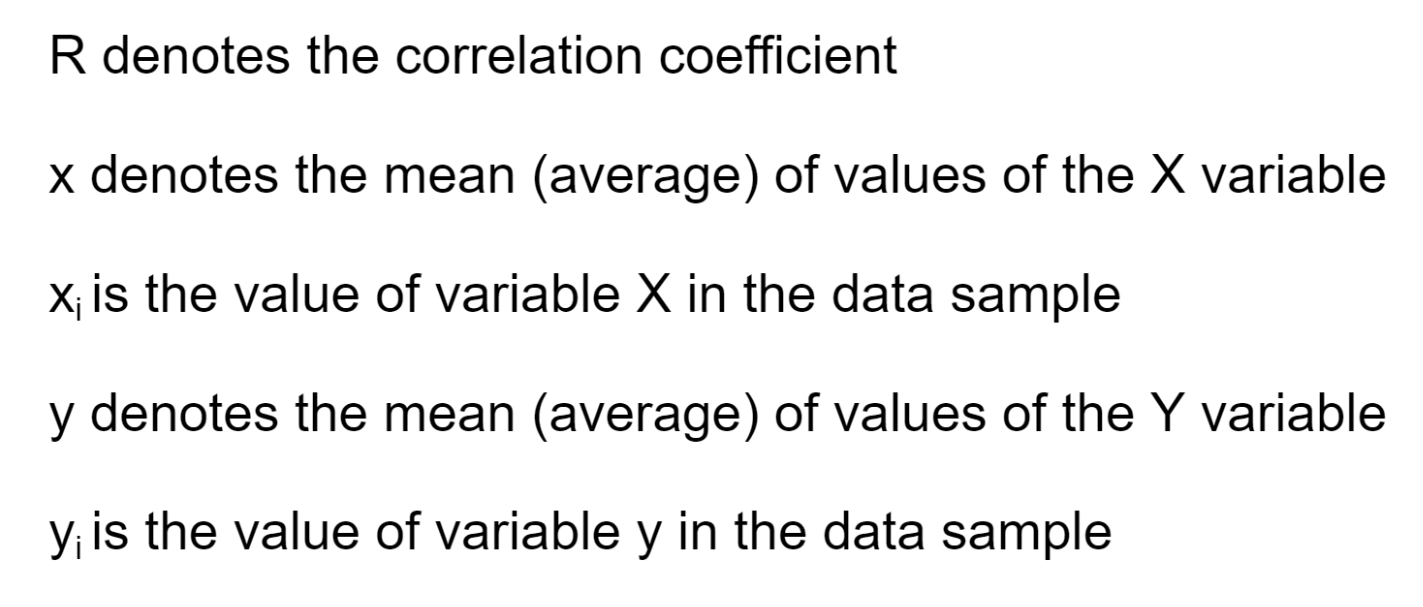
Correlation is a statistical concept determining the relationship potency of two numerical variables. While deducing the relation between variables, we conclude the change in one variable that impacts a difference in another.

When an analogous movement of another variable reciprocates the progression of one variable in some manner or another throughout the study of two variables, the variables are correlated.

The formula for calculating the correlation coefficient is as follows:



Where,



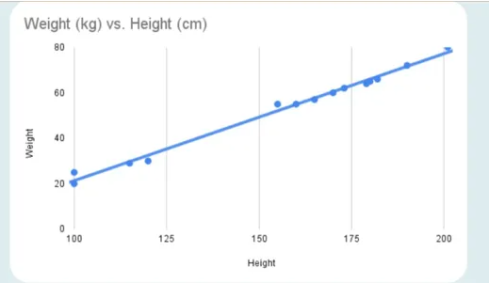
#### **Interpreting Correlation Values**

There are three types of correlation based on diverse values. Negative correlation, positive correlation, and no or zero correlation.

#### **Positive, Negative, and Zero Correlation**

If the variables are directly proportional to one another, the two variables are said to hold a positive correlation. This implies that if one variable’s value rises, the other’s value will exceed. An ideal positive correlation possesses a value of 1.

Here’s what a positive correlation looks like:



In a negative correlation, one variable’s value increases while the second one’s value decreases. A perfect negative correlation has a value of -1.

The negative correlation appears as follows:



Just like in the case of Covariance, a zero correlation means no relation between the variables. Therefore, whether one variable increases or decreases won’t affect the other variable.

#### **Strength and Direction of Correlation**

Correlation assesses the direction and strength of a linear relationship between multiple variables. The correlation coefficient varies from -1 to 1, with values near -1 or 1 implying a high association (negative or positive, respectively) and values near 0 suggesting a weak or no correlation.

#### **Pearson Correlation Coefficient and Its Properties**

The Pearson correlation coefficient (r) measures the linear connection between two variables. The properties of the Pearson correlation coefficient include the following:

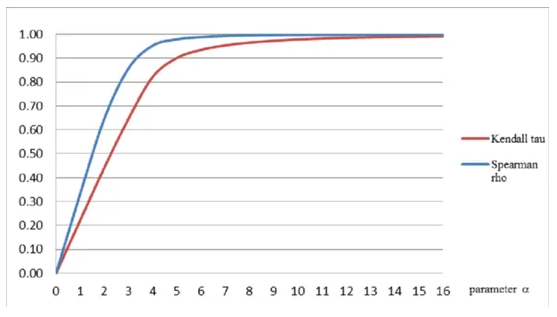
* **Strength:** The coefficient’s absolute value indicates the relationship’s strength. The closer the value of the coefficient is to 1, the stronger the correlation between variables. However, a value nearer to 0 represents a weaker association.
* **Direction:**The coefficient’s sign denotes the direction of the relationship. If the value is positive, there is a positive correlation between the two variables, which means that if one variable rises, the other will also rise. If the value is negative, there is a negative correlation, which suggests that when one variable increases, the other will fall.
* **Range:**The coefficient’s content varies from -1 to 1. The perfect linear relationship is represented by several -1, the absence of a linear relationship is represented by 0, and an ideal linear relationship is denoted by a value of 1.
* **Independence:**The Pearson correlation coefficient quantifies how linearly dependent two variables are but does not imply causality. There is no guarantee that a strong correlation indicates a cause-and-effect connection.
* **Linearity:** The Pearson correlation coefficient only assesses linear relationships between variables. The coefficient could be insufficient to describe non-linear connections fully.
* **Sensitivity to Outliers:** Outliers in the data might influence the correlation coefficient’s value, thereby boosting or deflating its size.

#### **Other Types of Correlation Coefficients**

Other correlation coefficients are:

* **Spearman’s Rank Correlation:**It’s a nonparametric indicator of rank correlation or the statistical dependency between the ranks of two variables. It evaluates how effectively a monotonic function can capture the connection between two variables.
* **Kendall Rank Correlation:** A statistic determines the ordinal relationship between two measured values. It represents the similarity of the data orderings when ordered by each quantity, which is a measure of rank correlation.

An image of an anti-symmetric family of copulas’ Spearman rank correlation and Kendall’s tau are inherently odd parameter functions.



## Advantages and Disadvantages of Covariance

Following are the advantages and disadvantages of Covariance:

#### **Advantages**

* **Easy to Calculate:** Calculating covariance doesn’t require any assumptions of the underlying data distribution. Hence, it’s easy to calculate covariance with the formula given above.
* **Apprehends Relationship:** Covariance gauges the extent of linear association between variables, furnishing information about the relationship’s magnitude and direction (positive or negative).
* **Beneficial in Portfolio Analysis:**Covariance is typically employed in portfolio analysis to evaluate the diversification advantages of integrating different assets.

#### **Disadvantages**

* **Restricted to Linear Relationships:**Covariance only gauges linear relationships between variables and does not capture non-linear associations.
* **Doesn’t Offer Relationship Magnitude:** Covariance doesn’t offer a standardized estimation of the intensity or strength of the relationship between variables.
* **Scale Dependency:**Covariance is affected by the variables’ measurement scales, making comparing covariances across various datasets or variables with distinct units challenging.

## Advantages and Disadvantages of Correlation

The advantages and disadvantages of correlation are as follows:

#### **Advantages**

* **Determining Non-Linear Relationships:** While correlation primarily estimates linear relationships, it can also demonstrate the presence of non-linear connections, especially when using alternative correlation standards like Spearman’s rank correlation coefficient.
* **Standardized Criterion:** Correlation coefficients, such as the Pearson correlation coefficient, are standardized, varying from -1 to 1. This allows for easy comparison and interpretation of the direction and strength of relationships across different datasets.
* **Robustness to Outliers:** Correlation coefficients are typically less sensitive to outliers than Covariance, delivering a more potent standard of the association between variables.
* **Scale Independencies:** Correlation is not affected by the measurement scales, making it convenient for comparing affinities between variables with distinct units or scales.

#### **Disadvantages**

* **Driven by Extreme Values: Extreme values can still affect the correlation coefficient**, even though it is less susceptible to outliers than Covariance.
* **Data Requirements:**Correlation assumes that the data is distributed according to a bivariate normal distribution, which may not always be accurate.
* **Limited to Bivariate Analysis:**Because correlation only examines the connection between two variables simultaneously, it can only capture simple multivariate correlations.

**New Concept:**

Python code for Correlation & Covariance

* 1. Using Numpy:

import numpy as np

# Sample data

data1 = np.array([1, 2, 3, 4, 5])

data2 = np.array([5, 6, 7, 8, 9])

# Covariance calculation

covariance = np.cov(data1, data2)[0][1]

# Correlation calculation

correlation = np.corrcoef(data1, data2)[0][1]

print("Covariance:", covariance)

print("Correlation:", correlation)

Output:

Covariance: 2.5

Correlation: 0.9999999999999999

* 1. Using Pandas:

import pandas as pd

# Sample data in a DataFrame

data = {

    'data1': [1, 2, 3, 4, 5],

    'data2': [5, 6, 7, 8, 9]

}

df = pd.DataFrame(data)

# Covariance calculation

covariance = df['data1'].cov(df['data2'])

# Correlation calculation

correlation = df['data1'].corr(df['data2'])

print("Covariance:", covariance)

print("Correlation:", correlation)

Output:

Covariance: 2.5

Correlation: 0.9999999999999999

**R-code for Correlation & Covariance**

# Sample data

data1 <- c(1, 2, 3, 4, 5)

data2 <- c(5, 6, 7, 8, 9)

# Calculate correlation coefficient

correlation <- cor(data1, data2)

print(paste("Correlation coefficient:", correlation))

# Compute covariance

covariance <- cov(data1, data2)

print(paste("Covariance:", covariance))

**Output:**

"Correlation coefficient: 1"

"Covariance: 2.5"

**Learning Objectives:**

To understand the correlation and covariance between the variables.

**Conclusion/Learning outcome:**

Correlation and Covariance between the variables between variables is understood and implemented in Python and R-code.

EXPERIMENT NO. 5

**Aim:** To understand and implement Hypothesis Testing.

**Prior Concepts:**

Hypothesis testing is a statistical method used to make inferences about a population parameter based on sample data. The process involves stating a hypothesis, collecting and analyzing data, and then determining whether the data provides enough evidence to reject or fail to reject the null hypothesis.

Following are the steps involved in hypothesis testing:

**1.** **State the Hypotheses:**

- Null Hypothesis (H0): Represents the default assumption or no effect.

- Alternative Hypothesis (H1 or Ha): Represents the hypothesis to be tested.

**2. Choose the Significance Level (α):**

- The significance level (α) determines how extreme the evidence must be before rejecting the null hypothesis.

- Common levels include 0.05 (5%) or 0.01 (1%).

**3. Select the Appropriate Test:**

- The choice of test depends on the nature of the data and the hypothesis being tested (e.g., t-test, chi-square test, ANOVA, etc.).

**4. Collect Data and Compute Test Statistic:**

- Use sample data to calculate a test statistic (e.g., t-statistic, z-statistic) based on the chosen test.

**5. Determine the Critical Region or P-value:**

- The critical region is determined based on the significance level.

- The p-value represents the probability of observing the test statistic or more extreme results if the null hypothesis is true.

**6. Make a Decision:**

- If the test statistic falls into the critical region (or if the p-value is less than the significance level), reject the null hypothesis.

- If the test statistic does not fall into the critical region (or if the p-value is greater than the significance level), fail to reject the null hypothesis.

**New Concepts:**

**Example using Student's t-test in Python:**

Suppose we want to test if there's a significant difference in the mean of two independent groups (e.g., group A and group B). You can use the t-test to perform this hypothesis test.

import numpy as np

from scipy.stats import ttest\_ind

# Generate sample data for two groups (Group A and Group B)

group\_A = np.random.normal(loc=50, scale=10, size=30)

group\_B = np.random.normal(loc=45, scale=12, size=30)

# Perform a two-sample t-test

t\_statistic, p\_value = ttest\_ind(group\_A, group\_B)

# Define the significance level

alpha = 0.05

# Print the results

print(f"T-statistic: {t\_statistic}")

print(f"P-value: {p\_value}")

if p\_value < alpha:

    print("Reject the null hypothesis: There is a significant difference between the means of Group A and Group B.")

else:

    print("Fail to reject the null hypothesis: There is no significant difference between the means of Group A and Group B.")

**Output:**

T-statistic: 1.6080745406164842

P-value: 0.11324943419680585

Fail to reject the null hypothesis: There is no significant difference between the means of Group A and Group B.

**Learning Objectives:**

To understand and implement Hypothesis Testing.

**Conclusion/Learning outcome:**

Hypothesis testing is understood and implemented with a sample dataset.

EXPERIMENT NO. 6

**Aim:** To implement Simple Linear Regression.

**Prior Concepts:**

Linear Regression is an algorithm that belongs to supervised Machine Learning. It tries to apply relations that will predict the outcome of an event based on the independent variable data points. The relation is usually a straight line that best fits the different data points as close as possible. The output is of a continuous form, i.e., numerical value. For example, the output could be revenue or sales in currency, the number of products sold, etc. In the above example, the independent variable can be single or multiple.

### 

### Linear Regression Line

Linear regression can be expressed mathematically as:

y= β0+ β 1x+ ε

Here,

Y= Dependent Variable

X= Independent Variable

β 0= intercept of the line

β1 = Linear regression coefficient (slope of the line)

ε = random error

The last parameter, random error ε, is required as the best fit line also doesn't include the data points perfectly.

### **Linear Regression Model**

Since the Linear Regression algorithm represents a linear relationship between a dependent (y) and one or more independent (y) variables, it is known as Linear Regression. This means it finds how the value of the dependent variable changes according to the change in the value of

## Types of Linear Regression

Linear Regression can be broadly classified into two types of algorithms:

### **Simple Linear Regression**

A simple straight-line equation involving slope (dy/dx) and intercept (an integer/continuous value) is utilized in simple Linear Regression. Here a simple form is y=mx+c where y denotes the output x is the independent variable, and c is the intercept when x=0. With this equation, the algorithm trains the model of machine learning and gives the most accurate output

### **Multiple Linear Regression**

When a number of independent variables more than one, the governing linear equation applicable to regression takes a different form like: y= c+m1x1+m2x2… mnxn

where represents the coefficient responsible for impact of different independent variables x1, x2 etc. This machine learning algorithm, when applied, finds the values of coefficients m1, m2, etc., and gives the best fitting line.

### **Non-Linear Regression**

When the best fitting line is not a straight line but a curve, it is referred to as Non-Linear Regression.   **Algorithm:**

i. Use a LoadDataSet() function to open text file with tab delimited values and assume

the last value is the target value.

ii.Use second function, standRegres(), to compute the best-fit line as follows:

Load the x and y arrays and then convert them into matrices.

Compute XTX and then test if its determinate is zero .

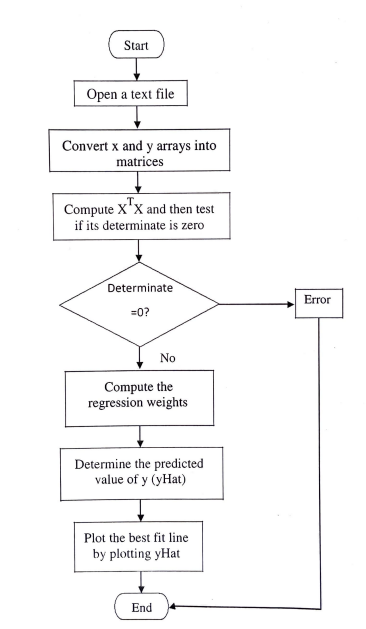
if the determinate is zero then you’ll get error

if the determinate is nonzero, you compute regression weights, ws and

return them

iii. Using ws, determinate the predicted value of y, by multiplying the x Matrix and ws.

iv. sort the point in ascending order, and plot the best fit line by plotting y.

**Flowchart:**

**New Concepts:**

Example of simple linear regression using a simple dataset in R-code:

# Sample data

x <- c(1, 2, 3, 4, 5)

y <- c(2, 4, 6, 8, 10)

# Fit a linear regression model

model <- lm(y ~ x)

# Summary of the linear regression model

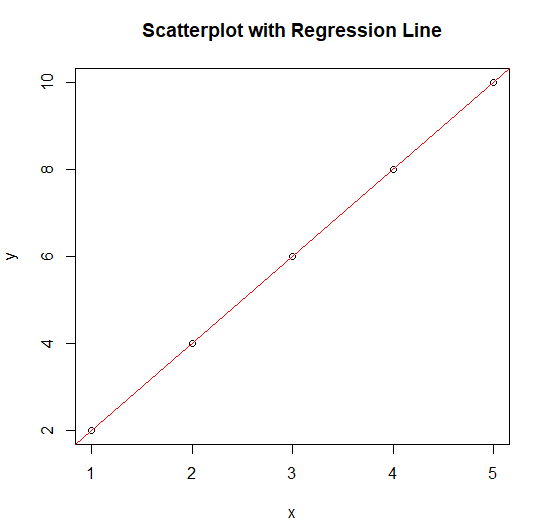
summary(model)

# Plotting the data with the regression line

plot(x, y, main = "Scatterplot with Regression Line")

abline(model, col = "red")

**Output:**

****

**Learning Objectives:**

To understand and implement linear regression on sample dataset.

**Conclusion/Learning outcome:**

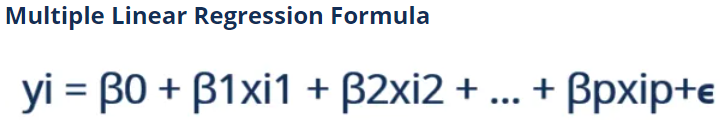
Regression analysis is a family of statistical tools that can help business analysts build models to predict trends, make tradeoff decisions, and model the real world for decision-making support. These models can be used to predict the value of one or more variables from knowledge of the value of other variables. Specific regression techniques include simple linear regression analysis, multiple linear regression analysis, multiple curvilinear regression, multivariate linear regression, and multivariate polynomial regression.

EXPERIMENT NO. 7

**Aim:** To implement Multiple Linear Regression.

**Prior Concepts:**

* Multiple linear regression refers to a statistical technique that uses two or more independent variables to predict the outcome of a dependent variable.
* The technique enables analysts to determine the variation of the model and the relative contribution of each independent variable in the total variance.
* Multiple regression can take two forms, i.e., linear regression and non-linear regression.



Where:

* **yi​** is the dependent or predicted variable
* **β0** is the y-intercept, i.e., the value of y when both xi and x2 are 0.
* **β1** and **β2** are the regression coefficients representing the change in y relative to a one-unit change in **xi1** and **xi2**, respectively.
* **βp** is the slope coefficient for each independent variable
* **ϵ** is the model’s random error (residual) term.

Simple linear regression enables statisticians to predict the value of one variable using the available information about another variable. Linear regression attempts to establish the relationship between the two variables along a straight line.

Multiple regression is a type of regression where the dependent variable shows a **linear** relationship with two or more independent variables. It can also be **non-linear**, where the dependent and [independent variables](https://corporatefinanceinstitute.com/resources/financial-modeling/independent-variable/) do not follow a straight line.

Both linear and non-linear regression track a particular response using two or more variables graphically. However, non-linear regression is usually difficult to execute since it is created from assumptions derived from trial and error.

### Assumptions of Multiple Linear Regression

Multiple linear regression is based on the following assumptions:

#### 1. A linear relationship between the dependent and independent variables

The first assumption of multiple linear regression is that there is a linear relationship between the dependent variable and each of the independent variables. The best way to check the linear relationships is to create scatterplots and then visually inspect the scatterplots for linearity. If the relationship displayed in the scatterplot is not linear, then the analyst will need to run a non-linear regression or transform the data using statistical software, such as SPSS.

#### 2. The independent variables are not highly correlated with each other

The data should not show multicollinearity, which occurs when the independent variables (explanatory variables) are highly correlated. When independent variables show multicollinearity, there will be problems figuring out the specific variable that contributes to the variance in the dependent variable. The best method to test for the assumption is the Variance Inflation Factor method.

#### 3. The variance of the residuals is constant

Multiple linear regression assumes that the amount of error in the residuals is similar at each point of the linear model. This scenario is known as homoscedasticity. When analyzing the data, the analyst should plot the standardized residuals against the predicted values to determine if the points are distributed fairly across all the values of independent variables. To test the assumption, the data can be plotted on a scatterplot or by using statistical software to produce a scatterplot that includes the entire model.

#### 4. Independence of observation

The model assumes that the observations should be independent of one another. Simply put, the model assumes that the values of residuals are independent. To test for this assumption, we use the Durbin Watson statistic.

The test will show values from 0 to 4, where a value of 0 to 2 shows positive autocorrelation, and values from 2 to 4 show negative autocorrelation. The mid-point, i.e., a value of 2, shows that there is no autocorrelation.

#### 5. Multivariate normality

Multivariate normality occurs when residuals are normally distributed. To test this assumption, look at how the values of residuals are distributed. It can also be tested using two main methods, i.e., a histogram with a superimposed normal curve or the Normal Probability Plot method.

**New Concepts:**

**R-code:**

# Sample data

x1 <- c(1, 2, 3, 4, 5)

x2 <- c(3, 4, 5, 6, 7)

y <- c(2, 4, 6, 8, 10)

# Combine the predictors into a data frame

data <- data.frame(x1, x2, y)

# Fit a multiple linear regression model

model <- lm(y ~ x1 + x2, data = data)

# Summary of the multiple linear regression model

summary(model)

# Making predictions using the model

new\_data <- data.frame(x1 = c(6, 7), x2 = c(8, 9)) # New data for prediction

predicted\_values <- predict(model, newdata = new\_data)

print(predicted\_values)

**Output:**

print(predicted\_values)

1 2

12 14

**Python code:**

from sklearn.linear\_model import LinearRegression

# Sample data

x1 = [1, 2, 3, 4, 5]

x2 = [3, 4, 5, 6, 7]

y = [2, 4, 6, 8, 10]

# Combine predictors into a feature matrix

X = list(zip(x1, x2))

# Fit the multiple linear regression model

model = LinearRegression().fit(X, y)

# Print model coefficients and intercept

print("Coefficients:", model.coef\_)

print("Intercept:", model.intercept\_)

# Predict using the model

new\_data = [[6, 8], [7, 9]]  # New data for prediction

predicted\_values = model.predict(new\_data)

print(predicted\_values)

**Output:**

Coefficients: [1. 1.]

Intercept: -2.0

[12. 14.]

CodeText

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from sklearn.linear\_model import LinearRegression

# Sample data

x1 = np.array([1, 2, 3, 4, 5])

x2 = np.array([3, 4, 5, 6, 7])

y = np.array([2, 4, 6, 8, 10])

# Reshape variables for sklearn

X = np.column\_stack((x1, x2))

# Fit the multiple linear regression model

model = LinearRegression().fit(X, y)

# Make predictions

y\_pred = model.predict(X)

# Plotting the actual data

fig = plt.figure(figsize=(10, 6))

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(x1, x2, y, color='blue', label='Actual data')

ax.scatter(x1, x2, y\_pred, color='red', label='Predicted data')

# Create a meshgrid for plotting the regression plane

x1\_mesh, x2\_mesh = np.meshgrid(np.linspace(min(x1), max(x1), 10), np.linspace(min(x2), max(x2), 10))

y\_mesh = model.predict(np.column\_stack((x1\_mesh.ravel(), x2\_mesh.ravel())))

y\_mesh = y\_mesh.reshape(x1\_mesh.shape)

# Plotting the regression plane

ax.plot\_surface(x1\_mesh, x2\_mesh, y\_mesh, alpha=0.5, color='green', label='Regression plane')

ax.set\_xlabel('X1')

ax.set\_ylabel('X2')

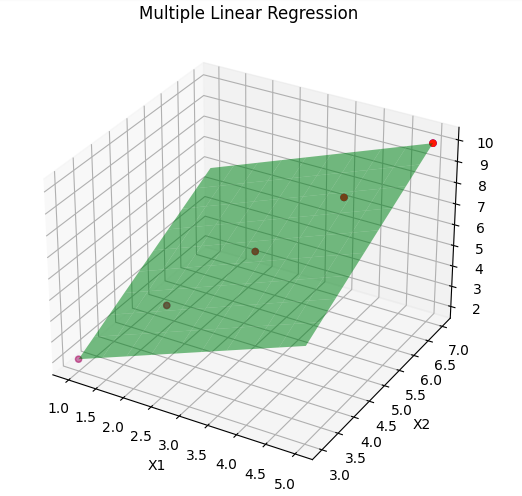
ax.set\_zlabel('Y')

ax.set\_title('Multiple Linear Regression')

#ax.legend()

plt.show()

**Output:**

****

**Learning Objectives:**

To understand and implement Multiple Linear Regression in R/Python.

**Conclusion/Learning outcome:**

Multiple Linear Regression is understood and implemented in R and Python.

EXPERIMENT NO. 8

**Aim:** To implement Time Series Analysis.

**Prior Concepts:**

**[Ref:** [**https://www.simplilearn.com/tutorials/python-tutorial/time-series-analysis-in-python**](https://www.simplilearn.com/tutorials/python-tutorial/time-series-analysis-in-python)**]**

Sometimes data changes over time. This data is called time-dependent data. Given time-dependent data, the past data can be analyzed to predict the future. The future prediction will also include time as a variable, and the output will vary with time. Using time-dependent data, patterns that repeat over time can be found. A Time Series is a set of observations that are collected after regular intervals of time. If plotted, the Time series would always have one of its axes as time.

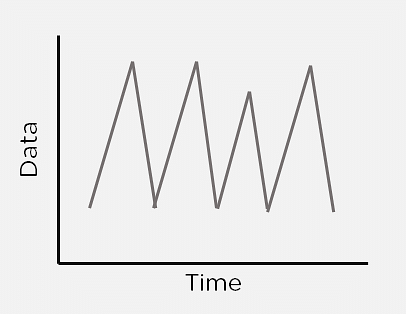


Figure 1: Time Series

Time Series Analysis in Python considers data collected over time might have some structure; hence it analyses Time Series data to extract its valuable characteristics.

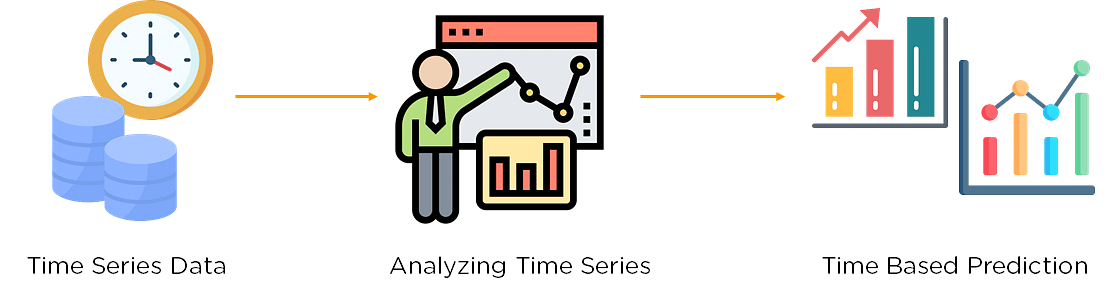


Figure 2: Time Series Analysis

Consider the running of a bakery. Given the data of the past few months, one can predict what items are needed to bake at what time. The morning crowd would need more bread items, like bread rolls, croissants, breakfast muffins, etc. At night, people may come in to buy cakes and pastries or other dessert items. Using time series analysis, one can predict items popular during different times and even different seasons.

**Different Components of Time Series Analysis:**

The diagram depicted below shows the different components of Time Series Analysis:

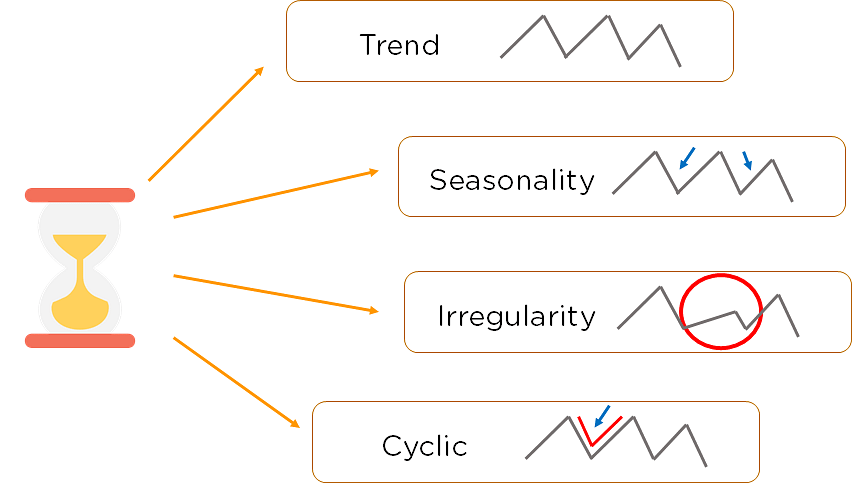


Figure 3: Components of Time Series Analysis

1. **Trend:** The Trend shows the variation of data with time or the frequency of data. Using a Trend, you can see how your data increases or decreases over time. The data can increase, decrease, or remain stable. Over time, population, stock market fluctuations, and production in a company are all examples of trends.
2. **Seasonality:** Seasonality is used to find the variations which occur at regular intervals of time. Examples are festivals, conventions, seasons, etc. These variations usually happen around the same time period and affect the data in specific ways which you can predict.
3. **Irregularity:** Fluctuations in the time series data do not correspond to the trend or seasonality. These variations in your time series are purely random and usually caused by unforeseeable circumstances, such as a sudden decrease in population because of a natural calamity.
4. **Cyclic:** Oscillations in time series which last for more than a year are called cyclic. They may or may not be periodic.
5. **Stationary:** A time series that has the same statistical properties over time is stationary. The properties remain the same anywhere in the series. Your data needs to be stationary to perform time-series analysis on it. A stationary series has a constant mean, variance, and covariance.

## ARIMA Model

ARIMA Model stands for Auto-Regressive Integrated Moving Average. It is used to predict the future values of a time series using its past values and forecast errors. The below diagram shows the components of an ARIMA model:

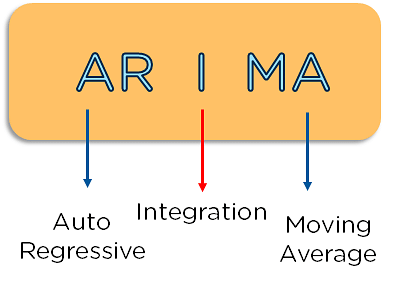


Figure 4: Components of ARIMA

### **Auto Regressive Model**

Auto-Regressive models predict future behavior using past behavior where there is some correlation between past and future data. The formula below represents the autoregressive model. It is a modified version of the slope formula with the target value being expressed as the sum of the intercept, the product of a coefficient and the previous output, and an error correction term.

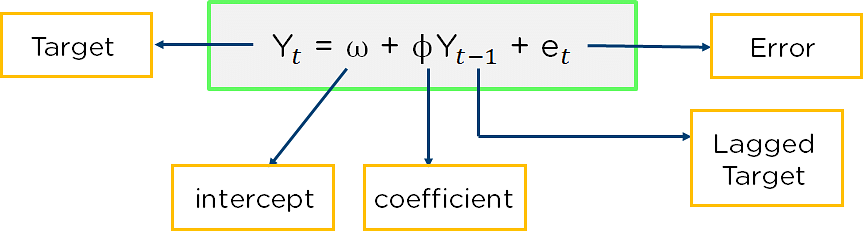


Figure 5: Auto-Regressive Model

### **Moving Average**

Moving Average is a statistical method that takes the updated average of values to help cut down on noise. It takes the average over a specific interval of time. You can get it by taking different subsets of your data and finding their respective averages.

You first consider a bunch of data points and take their average. You then find the next average by removing the first value of the data and including the next value of the series.

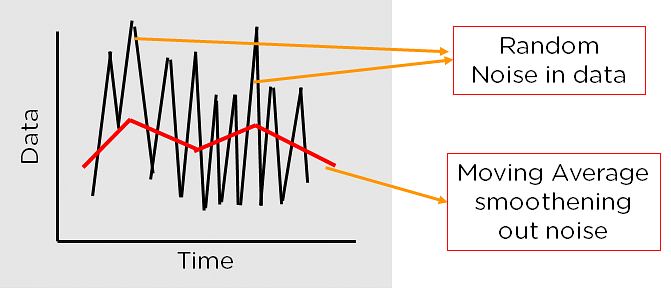


Figure 6: Stationarity using Moving Average

### **Integration**

Integration is the difference between present and previous observations. It is used to make the time series stationary. Each of these values acts as a parameter for an ARIMA model. Instead of representing the ARIMA model by these various operators and models, one can use parameters to represent them.

These parameters are:

1. p: Previous lagged values for each time point. Derived from the Auto-Regressive Model.
2. q: Previous lagged values for the error term. Derived from the Moving Average.
3. d: Number of times data is differenced to make it stationary. It is the number of times it performs integration.

**Code:**

**[Ref:** [**https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/**](https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/)**]**

from pandas import read\_csv

from pandas import datetime

from matplotlib import pyplot

def parser(x):

return datetime.strptime('190'+x, '%Y-%m')

series = read\_csv('shampoo-sales.csv', header=0, parse\_dates=[0], index\_col=0, squeeze=True, date\_parser=parser)

print(series.head())

series.plot()

pyplot.show()

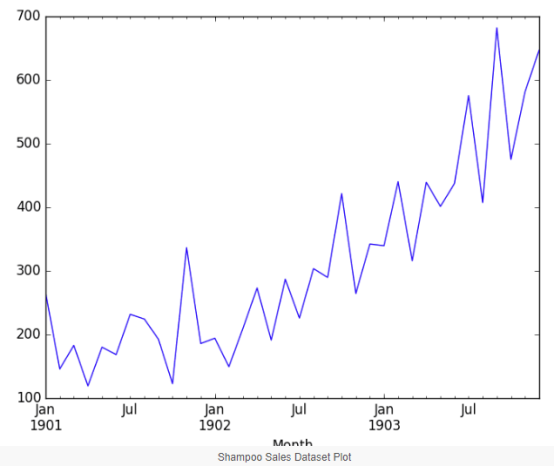
from pandas import read\_csv

from pandas import datetime

from matplotlib import pyplot

from pandas.plotting import autocorrelation\_plot

**Output:**



def parser(x):

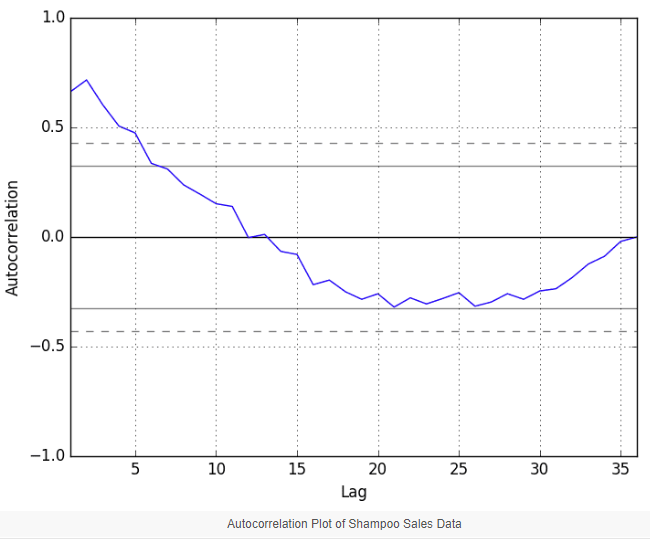
return datetime.strptime('190'+x, '%Y-%m')

series = read\_csv('shampoo-sales.csv', header=0, parse\_dates=[0], index\_col=0, squeeze=True, date\_parser=parser)

autocorrelation\_plot(series)

pyplot.show()

**Output:**

****

## ARIMA with Python

The statsmodels library stands as a vital tool for those looking to harness the power of ARIMA for time series forecasting in Python. Building an ARIMA Model:

A Step-by-Step Guide:

1. **Model Definition**: Initialize the ARIMA model by invoking ARIMA() and specifying the p, d, and q parameters.
2. **Model Training**: Train the model on your dataset using the fit() method.
3. **Making Predictions**: Generate forecasts by utilizing the predict() function and designating the desired time index or indices.

Let us fit an ARIMA model to the entire Shampoo Sales dataset and review the residual errors.

We’ll employ the ARIMA(5,1,0) configuration:

* 5 lags for autoregression (AR)
* 1st order differencing (I)
* No moving average term (MA)

# fit an ARIMA model and plot residual errors

from pandas import datetime

from pandas import read\_csv

from pandas import DataFrame

from statsmodels.tsa.arima.model import ARIMA

from matplotlib import pyplot

# load dataset

def parser(x):

return datetime.strptime('190'+x, '%Y-%m')

series = read\_csv('shampoo-sales.csv', header=0, index\_col=0, parse\_dates=True, squeeze=True, date\_parser=parser)

series.index = series.index.to\_period('M')

# fit model

model = ARIMA(series, order=(5,1,0))

model\_fit = model.fit()

# summary of fit model

print(model\_fit.summary())

# line plot of residuals

residuals = DataFrame(model\_fit.resid)

residuals.plot()

pyplot.show()

# density plot of residuals

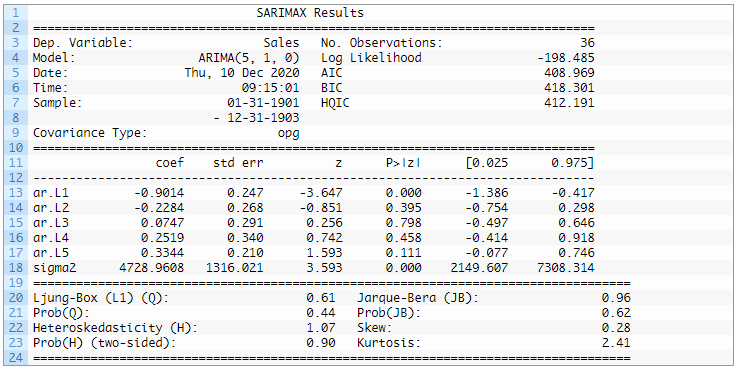
residuals.plot(kind='kde')

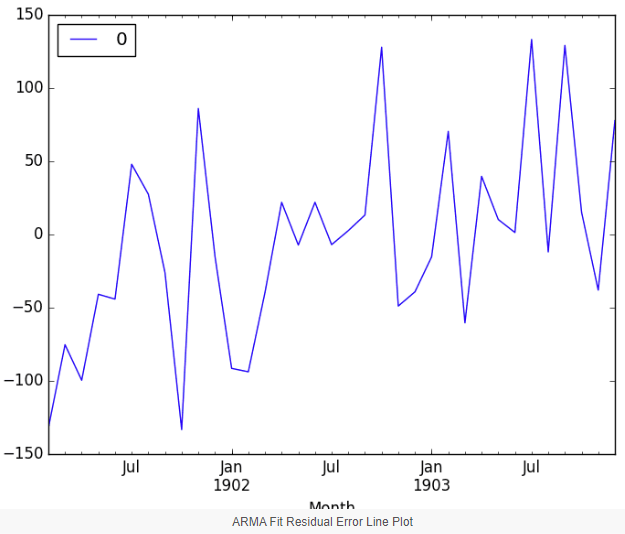
pyplot.show()

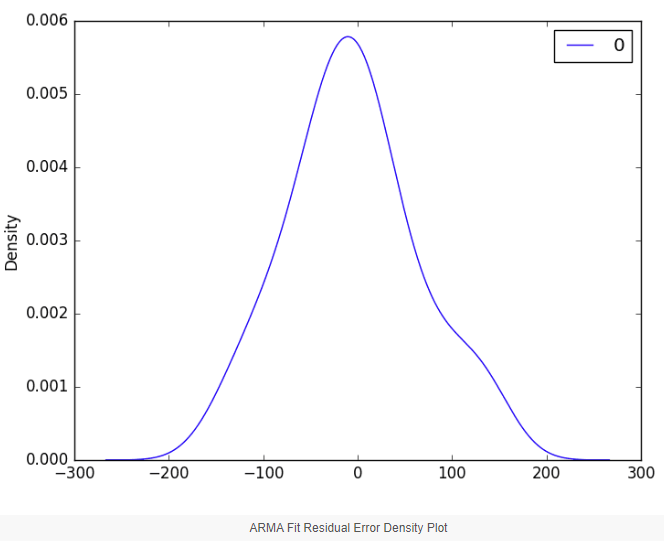
# summary stats of residuals

print(residuals.describe())

**Output:**

****

****

****

## Rolling Forecast ARIMA Model

The ARIMA model can be used to forecast future time steps.

The ARIMA model is adept at forecasting future time points. In a rolling forecast, the model is often retrained as new data becomes available, allowing for more accurate and adaptive predictions.

We can use the predict() function on the [ARIMAResults](https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMAResults.html) object to make predictions. It accepts the index of the time steps to make predictions as arguments. These indexes are relative to the start of the training dataset used to make predictions.

How to Forecast with ARIMA:

1. Use the predict() function on the ARIMAResults object. This function requires the index of the time steps for which predictions are needed.
2. To revert any differencing and return predictions in the original scale, set the typ argument to ‘levels’.
3. For a simpler one-step forecast, employ the forecast() function.

We can split the training dataset into train and test sets, use the train set to fit the model and generate a prediction for each element on the test set.

A rolling forecast is required given the dependence on observations in prior time steps for differencing and the AR model. A crude way to perform this rolling forecast is to re-create the ARIMA model after each new observation is received.

# evaluate an ARIMA model using a walk-forward validation

from pandas import read\_csv

from pandas import datetime

from matplotlib import pyplot

from statsmodels.tsa.arima.model import ARIMA

from sklearn.metrics import mean\_squared\_error

from math import sqrt

# load dataset

def parser(x):

return datetime.strptime('190'+x, '%Y-%m')

series = read\_csv('shampoo-sales.csv', header=0, index\_col=0, parse\_dates=True, squeeze=True, date\_parser=parser)

series.index = series.index.to\_period('M')

# split into train and test sets

X = series.values

size = int(len(X) \* 0.66)

train, test = X[0:size], X[size:len(X)]

history = [x for x in train]

predictions = list()

# walk-forward validation

for t in range(len(test)):

model = ARIMA(history, order=(5,1,0))

model\_fit = model.fit()

output = model\_fit.forecast()

yhat = output[0]

predictions.append(yhat)

obs = test[t]

history.append(obs)

print('predicted=%f, expected=%f' % (yhat, obs))

# evaluate forecasts

rmse = sqrt(mean\_squared\_error(test, predictions))

print('Test RMSE: %.3f' % rmse)

# plot forecasts against actual outcomes

pyplot.plot(test)

pyplot.plot(predictions, color='red')

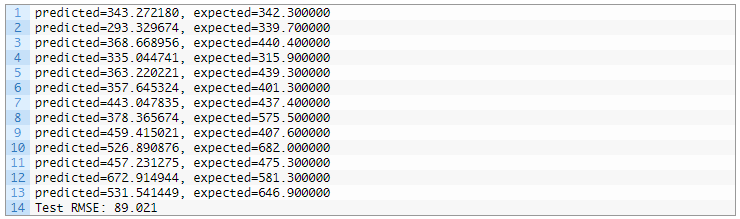
pyplot.show()

We manually keep track of all observations in a list called history that is seeded with the training data and to which new observations are appended each iteration.

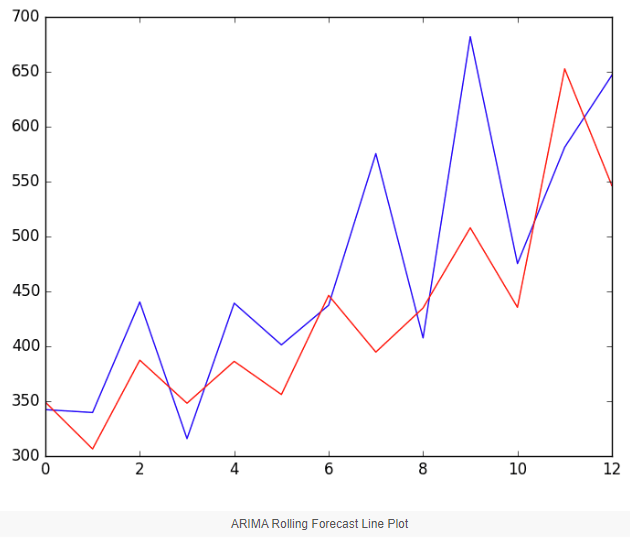
Putting this all together, below is an example of a rolling forecast with the ARIMA model in Python.

Running the example prints the prediction and expected value each iteration.

We can also calculate a final root mean squared error score (RMSE) for the predictions, providing a point of comparison for other ARIMA configurations.

****

A line plot is created showing the expected values (blue) compared to the rolling forecast predictions (red). We can see the values show some trend and are in the correct scale.

****

The model could use further tuning of the p, d, and maybe even the q parameters.

**Learning Objectives:**

To understand and implement Time Series Analysis using Python/Excel

**Conclusion/Learning outcome:**

Time Series Analysis using ARIMA Model has been understood and implemented in Python.