AI for Engineers (UCS321)

Branch

B.E. 2nd Year – ECE

Submitted By: Ms. Manmeet Kaur

Submitted To: Dr. Neeru Jindal

Department of Electronics and Communication Engineering

Thapar Institute of Engineering and Technology

Patiala – 147001

List of Experiments/Index

Experiment - 1

Python Fundamentals

Aim

1. Understand the basic syntax and semantics of Python.
2. Familiarize with Python variables, data types, and operators.
3. Practice input/output operations in Python on jupyter.
4. Familiarize with lists, tuples, and dictionaries.
5. Learn basic control flow: conditional statements and loops.
6. Learn about functions, function definition and calling.
7. Learn about classes and objects.

Theory:

1. *The basic syntax and semantics of Python*

Python is a high-level, interpreted programming language that is easy to read and write because of its simple syntax. It is widely used for fields like web development, data science, and automation. In Python, **comments** begin with # and are not executed by the interpreter, helping explain the code.

```
# This is a comment in Python
x = 10  # Inline comment
```

2. *Python variables, data types, and operators.*

In Python, **variables** are used to store data, and they do not require explicit type declaration . Python supports various **data types**, such as integers, floats, strings, and booleans, which allow us to represent numbers, text, and logical values. To work with these values, Python provides **operators**. Arithmetic operators like +, -, *, / perform calculations, comparison operators like ==, !=, <, > check relationships, and logical operators and help in decision-making.

```
a, b = 5, 2
print(a + b)      # 7 (arithmetic)
print(a > b)      # True (comparison)
print(a > 0 and b > 0)  # True (logical)
```

3. *Input/output operations in Python*

In Python, **input and output (I/O) operations** allow interaction between the user and the program. The input() function is used to take input from the user, and it always returns a string. The print() function is used to display messages, variables, or results to the screen.

```
name = input("Enter your name: ")    # user enters a string
age = int(input("Enter your age: "))  # user input converted to integer
print("Hello,", name, "- You are", age, "years old.")
```

4. *Lists, tuples, and dictionaries*

Lists are ordered, mutable collections that can store multiple items of different types. **Tuples** are ordered but immutable collections, meaning their elements cannot be modified after creation. **Dictionaries** store data in key–value pairs, allowing fast lookups using keys.

```
# List (mutable), Tuple (immutable), and basic Python operations
fruits = ["apple", "banana", "cherry"]      # list
numbers = (1, 2, 3)                         # tuple
fruits.append("mango")                       # modify list
print("Fruits:", fruits)
print("First number in tuple:", numbers[0])  # access tuple element
```

5. *Basic control flow: conditional statements and loops*

Conditional statements allow programs to **make decisions** based on certain conditions. Using if, elif, and else, Python executes code blocks only when the condition evaluates to True.

```
x = 10
if x > 5:
    print("x is greater than 5")
else:
    print("x is 5 or less")
```

Loops let us **repeat a block of code** multiple times, reducing redundancy. A for loop runs a block for a fixed number of iterations, while a while loop continues as long as a condition is True.

6. *Functions, function definition and calling.*

Functions are reusable blocks of code that perform a specific task. They are defined using the def keyword and can be called with or without arguments to execute the code inside.

```
def greet(name):
    print("Hello,", name)

greet("Alice")    # Output: Hello, Alice
```

Function calling allows us to run the code inside the function whenever needed, making programs modular and easier to manage. Functions can also return values for further use.

```
def add(a, b):
    return a + b

result = add(5, 3)
print(result)    # Output: 8
```

7. *Classes and objects.*

Classes are blueprints for creating objects, defining their attributes and behaviors. They are defined using the class keyword and can include methods (functions) and properties (variables).

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Objects are instances of classes, created to use the attributes and methods defined in the class. Each object can have its own unique data.

```
s1 = Student("John", 21)
print(s1.name, s1.age)    # Output: John 21
```

Experiment - 2

Python Libraries

Aim

1. Understand how to import a library and use its functions.
2. Familiarize with basic functions of the math, numpy, Tensorflow, Scikit -Learn and Pandas Library

Theory:

1. *Import a library*

In Python, libraries provide pre-written code to perform common tasks. You can **import** a library using the import keyword and use its functions directly.

```
import math

print(math.sqrt(16))    # Output: 4.0, using sqrt() from math library
print(math.pi)          # Accessing constant pi from math
```

Another example:

```
from math import factorial

print(factorial(5))     # Output: 120
```

This allows us to **reuse powerful functions** without writing them from scratch.

2. *Math library*

Python's built-in math library provides functions for basic and advanced mathematical operations, such as square roots, powers, logarithms, and trigonometry. It also offers constants like pi and e for scientific calculations. Using math avoids manually implementing standard formulas.

```
import math
print(math.sqrt(25))         # 5.0
print(math.factorial(5))     # 120
print(math.sin(math.pi/2))   # 1.0
```

3. *NumPy library*

NumPy is a fundamental library for numerical and matrix operations in Python. It allows creating multi-dimensional arrays and performing element-wise computations efficiently. NumPy is widely used in scientific computing, machine learning, and data processing.

```
import numpy as np
arr = np.array([1, 2, 3])
print(arr * 2)               # [2 4 6]
matrix = np.array([[1,2],[3,4]])
print(matrix + 1)            # [[2 3] [4 5]]
```

4. *TensorFlow*

TensorFlow is a deep learning library for building and training neural networks. It works with tensors (multi-dimensional arrays) and provides operations that can run on CPUs or GPUs for efficient computation. TensorFlow also includes high-level APIs for model development.

```
import tensorflow as tf
x = tf.constant([1, 2, 3])
y = tf.constant([4, 5, 6])
print(x + y)                # tf.Tensor([5 7 9], shape=(3,),
dtype=int32)
print(tf.reduce_sum(x))     # 6
```

5. *Scikit-learn*

Scikit-learn (sklearn) – Scikit-learn is a machine learning library that provides tools for classification, regression, clustering, and data preprocessing. It is easy to use for building models and evaluating their performance. Scikit-learn integrates well with NumPy and Pandas for data handling.

```
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([[1], [2], [3]])
y = np.array([2, 4, 6])
model = LinearRegression()
model.fit(X, y)
print(model.predict([[4]]))    # [8.0]
```

6. *Pandas*

Pandas – Pandas is a library for data manipulation and analysis. It provides DataFrame and Series objects to handle tabular and labeled data efficiently. With Pandas, you can filter, sort, group, and perform computations on datasets easily.

```
import pandas as pd
data = pd.DataFrame({"Name": ["Alice", "Bob", "Charlie"], "Age": [25,
30, 35]})
print(data)
print(data["Age"] * 2)         # Multiply age column by 2
print(data[data["Age"] > 28]) # Filter rows where Age > 28
```

Summary:

Python has powerful libraries for different tasks: **math** handles standard mathematical operations; **NumPy** enables fast numerical computations with arrays; **TensorFlow** is used for building and training neural networks; **Scikit-learn** provides tools for machine learning models; and **Pandas** simplifies data manipulation and analysis with DataFrames. Together, they make Python efficient for scientific computing, ML, and data analysis.

Experiment - 3

Data Preprocessing

Aim

1. Handling Null Values
2. Outlier Detection and Removal
3. Dimensionality Reduction using PCA and SVD

Theory:

1. *Handling Null Values*

Null values, or missing data, can disrupt analysis and reduce the accuracy of predictive models if not addressed properly. Identifying missing values is the first step, and they can be handled through several techniques such as dropping rows or columns containing nulls, filling them with statistical measures like mean, median, or mode, or using advanced methods like K-nearest neighbors (KNN) imputation. Properly managing null values ensures that datasets remain consistent, reliable, and suitable for downstream analysis and machine learning tasks.

2. *Outlier Detection and Removal*

Outliers are extreme values that deviate significantly from the rest of the dataset, potentially skewing analysis and leading to misleading insights. They can be detected using statistical methods such as Z-score analysis or the Interquartile Range (IQR) method, which highlight points that fall far from the central tendency of the data. After identification, outliers can either be removed or transformed, helping maintain dataset integrity and enhancing the performance and robustness of predictive models.

3. *Dimensionality Reduction using PCA and SVD*

High-dimensional datasets can be challenging to analyze due to complexity and redundancy. Dimensionality reduction techniques like Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) transform data into a lower-dimensional space while retaining the most important features. PCA identifies directions (principal components) that capture maximum variance, whereas SVD decomposes matrices to reveal latent structures. Using these methods improves computational efficiency, reduces noise, and can enhance the performance of machine learning models.

Code: using random classifier to impute missing values

```
import numpy as np
import pandas as pd
df=pd.DataFrame({
    "Date":pd.date_range(start="2021-10-01",periods=10,freq="D"),
    "Item":1014,
    "Measure_1":np.random.randint(1,10,size=10),
    "Measure_2":np.random.random(10).round(2),
    "Measure_3":np.random.random(10).round(2),
    "Measure_4":np.random.randn(10)
})
df
df
find mean and standard deciation of measure_4
df.loc[[2,9],"Item"]=np.nan
df.loc[[2,7,9],"Measure_1"]=np.nan
df.loc[[2,3],"Measure_2"]=np.nan
df.loc[[2],"Measure_3"]=np.nan
df.loc[:6,"Measure_4"]=np.nan
df
df=df.astype({
    "Item":pd.Int64Dtype(),
    "Measure_1":pd.Int64Dtype()
})
df
!pip install missingpy
import sklearn.neighbors._base
import sys
sys.modules['sklearn.neighbors.base'] = sklearn.neighbors._base
from missingpy import MissForest
df2=pd.read_csv('train.csv')
df2.drop("Name",axis=1,inplace=True)
df2.drop("Ticket",axis=1,inplace=True)
df2.drop("PassengerId",axis=1,inplace=True)
df2.drop("Cabin",axis=1,inplace=True)
df2.drop("Embarked",axis=1,inplace=True)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df2['Sex'] = le.fit_transform(df2['Sex'])
df2
imputer1=MissForest()
imputer1.fit_transform(df2)
from sklearn.impute import KNNImputer
K=KNNImputer(n_neighbors=5)
K.fit_transform(df2)
```

Code : outlier analysis by visualization and inter uaterile range

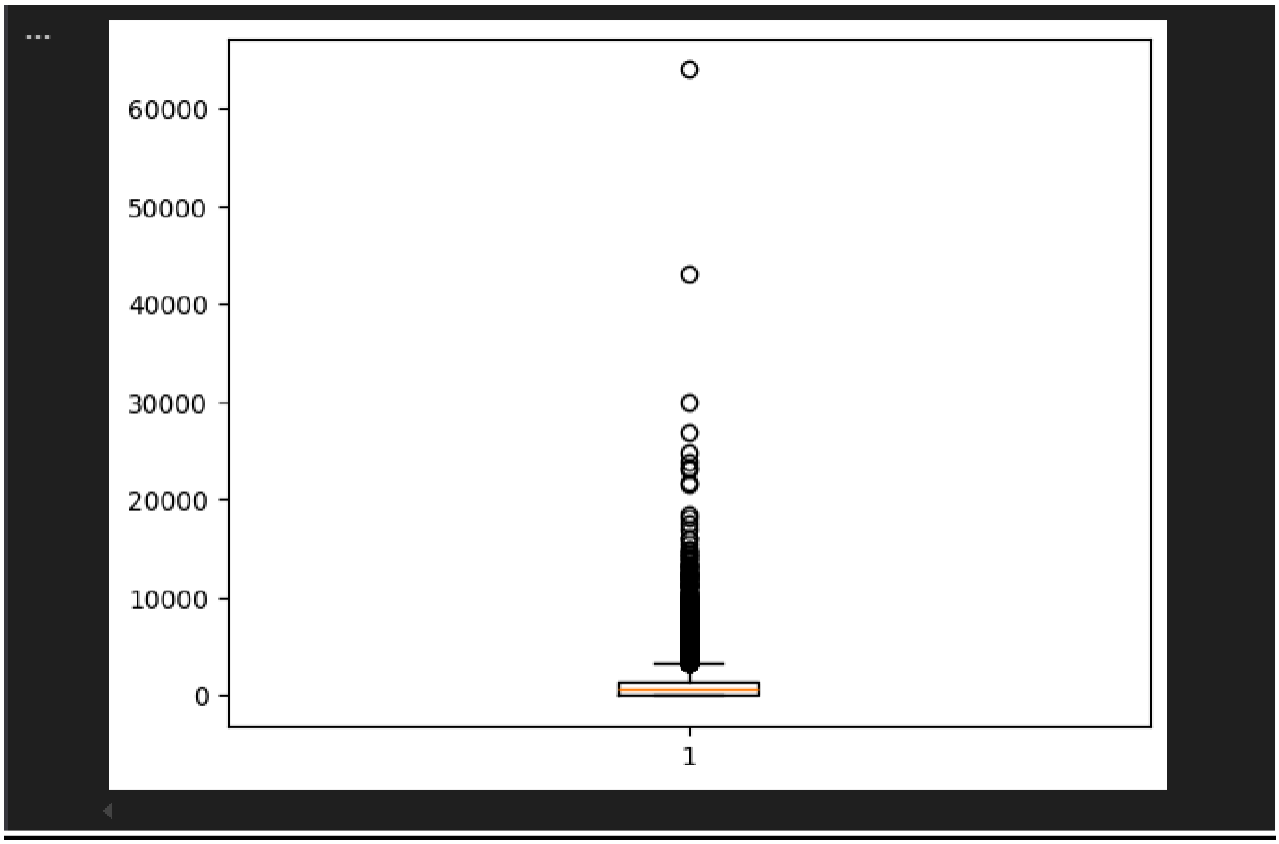
```
import sklearn
from sklearn.datasets import fetch_california_housing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
file_path='/content/online_shoppers_intention.csv'
df = pd.read_csv(file_path)
print(df.shape)
df.head(12331)
import seaborn as sns
plt.boxplot(df['ProductRelated_Duration'])
print(np.where(df['ProductRelated_Duration']>30000))
print(np.where(df['ProductRelated_Duration']>30000))
fig, ax = plt.subplots(figsize = (18,10))
ax.scatter(df['BounceRates'], df['ProductRelated_Duration'])
ax.set_xlabel('BOunce rates')
ax.set_ylabel('product related duration')
plt.show()

# IQR
import numpy as np
Q1 = np.percentile(df['ProductRelated_Duration'], 25,
                   interpolation = 'midpoint')

Q3 = np.percentile(df['ProductRelated_Duration'], 75,
                   interpolation = 'midpoint')
IQR = Q3 - Q1
upper = np.where(df['ProductRelated_Duration'] >= (Q3+1.5*IQR))
print(upper)
lower = np.where(df['ProductRelated_Duration'] <= (Q1-1.5*IQR))
print(lower)
df['ProductRelated_Duration'] >= (Q3+1.5*IQR)
df.drop(upper[0], inplace = True)
df.drop(lower[0], inplace = True)

print("New Shape: ", df.shape)
```

Results

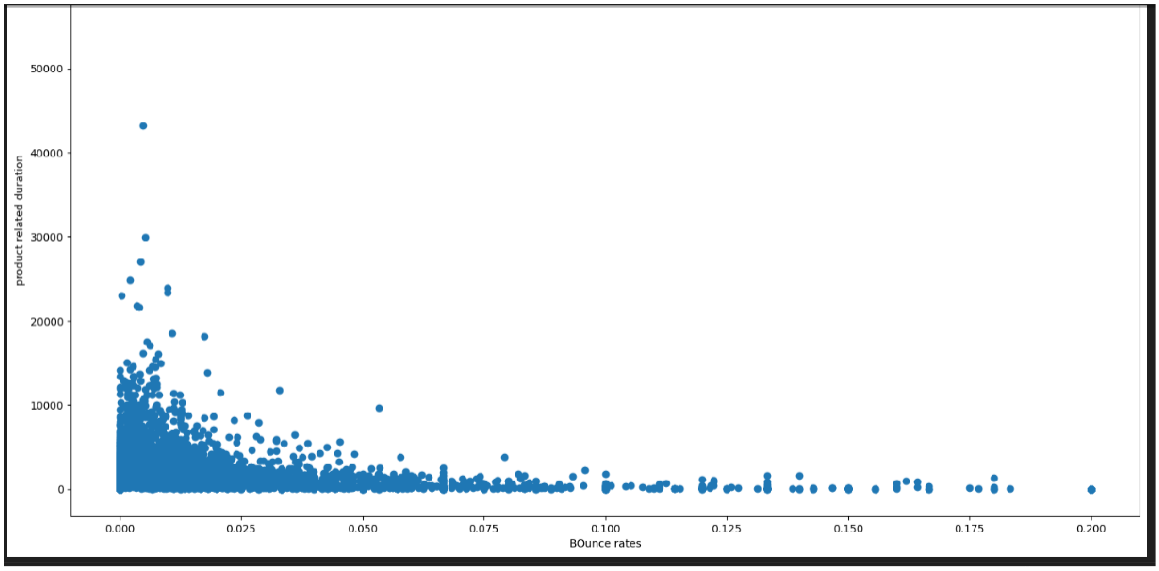
Part 1: Input missing values

```
array([[ 0.   , 3.   , 1.   , ..., 1.   , 0.   , 7.25  ],
       [ 1.   , 1.   , 0.   , ..., 1.   , 0.   , 71.2833],
       [ 1.   , 3.   , 0.   , ..., 0.   , 0.   , 7.925 ],
       ...,
       [ 0.   , 3.   , 0.   , ..., 1.   , 2.   , 23.45  ],
       [ 1.   , 1.   , 1.   , ..., 0.   , 0.   , 30.    ],
       [ 0.   , 3.   , 1.   , ..., 0.   , 0.   , 7.75  ]])
```

Boxplot:



Scatter-Plot:



IQR :

	ProductRelated_Duration
0	False
1	False
2	False
3	False
4	False
...	...
12325	False
12326	False
12327	False
12328	False
12329	False

12330 rows × 1 columns

dtype: bool

Experiment - 4

How to use Git effectively?

Aim

1. Understand the purpose of version control systems.
2. Learn how to set up and configure Git.
3. Practice essential Git commands for managing code effectively.
4. Explore collaboration workflows with branching and merging.

Key Git Concepts:

- Repository (repo): A storage location for your project.
- Commit: A snapshot of your project at a certain time.
- Branch: A parallel version of your project for isolated development.
- Merge: Integrating changes from one branch into another.

Theory:

Git is a distributed version control system that allows developers to track changes in code, collaborate efficiently, and maintain a history of project revisions. It enables multiple people to work on the same project simultaneously without overwriting each other's work by using branches and merges. Git also provides features like staging, commits, and rollback, which make it easy to manage versions, experiment safely, and recover previous states of a project. Widely used in software development, Git is the foundation of platforms like GitHub and GitLab for collaborative coding and project management.

Step 1: Install and Configure Git

```
git –version
git config --global user.name "Your Name"
git config --global user.email your.email@example.com
```

Step 2: Create a New Repository

```
mkdir my_project
cd my_project
git init
```

Step 3: Add and Commit Files

```
echo "Hello Git" > file.txt
git status
git add file.txt
git commit -m "Initial commit with file.txt"
```

Step 4: Create and Switch Branches

```
git branch new_feature
git checkout new_feature
```

Step 5: Merge Branches

```
git  checkout  main
git merge new_feature
```

Step 6: Connect to Remote Repository

```
git remote add origin https://github.com/username/my_project.git
git push -u origin main
```

Summary:

```
# Step 1: Initialize a new Git repository
git init

# Step 2: Check the repository status
git status

# Step 3: Add files to the staging area
git add .

# Step 4: Commit the staged changes
git commit -m "Initial commit"

# Step 5: Add a remote repository
git remote add origin https://github.com/username/myproject.git

# Step 6: Push local commits to the remote repository
git push -u origin main
```

Result:

Through this lab activity we learned how to use Git and Github effectively to publish and keep track of our projects.

Experiment - 5

Build a Regression Model to predict temperature rise in a motor

Aim

The main objectives of this lab are to:
1. Understand regression modeling as a predictive analytics technique for continuous variables.
2. Preprocess and prepare data for training a regression model.
3. Train and evaluate a regression model using performance metrics such as R^2, MAE, and RMSE.
4. Interpret model coefficients to understand the influence of each feature on temperature rise.

Theory

In the era of data-driven decision-making, predicting outcomes accurately can be as powerful as discovering hidden patterns in nature, and regression modeling stands at the heart of this exploration. This lab delves into regression as a predictive analytics technique, focusing on continuous variables such as temperature rise, where understanding subtle relationships between factors is crucial. The journey begins with meticulous data preprocessing, transforming raw and sometimes chaotic data into a structured form suitable for modeling, ensuring both precision and reliability. Once the data is primed, a regression model is trained to uncover the intricate connections between dependent and independent variables, and its performance is critically assessed using metrics like R^2, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE), which quantify both accuracy and predictive consistency. Beyond forecasting, interpreting the model's coefficients offers valuable insights into how each feature influences temperature rise, highlighting the most impactful variables and supporting informed, evidence-based decisions. By combining these steps, the lab not only illustrates the mechanics of regression analysis but also emphasizes its real-world significance in converting complex data into actionable knowledge.

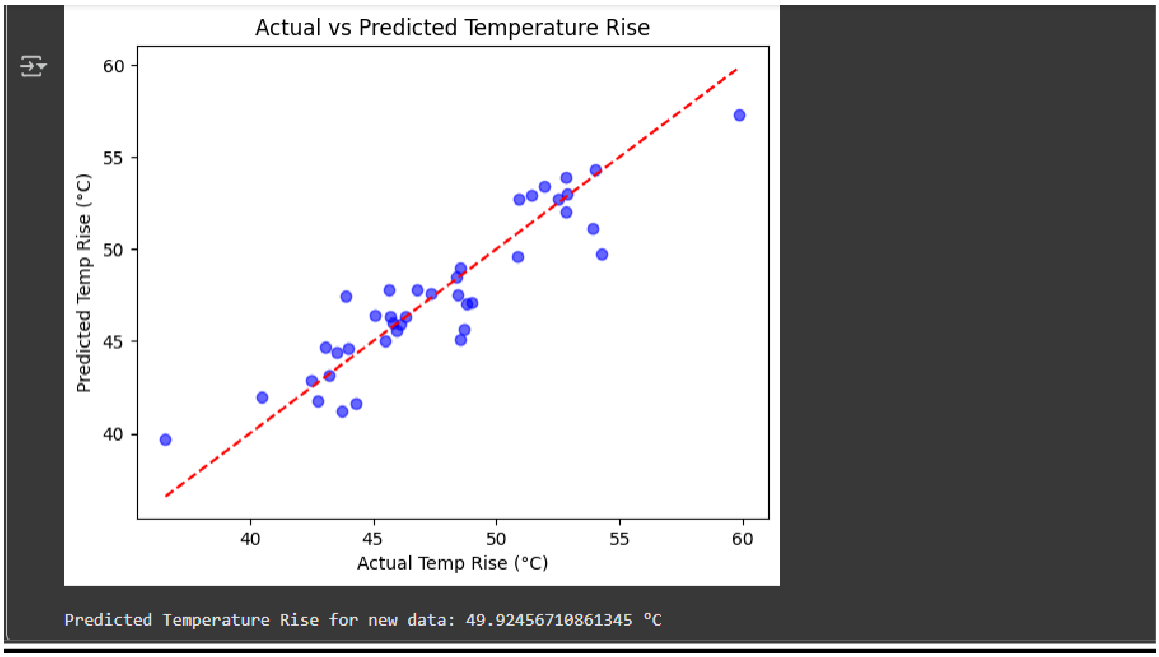
Through this lab, regression modeling has proven to be a robust tool for uncovering relationships in complex data. Interpreting model coefficients provides clear insights into the factors driving temperature rise. Overall, the exercise highlights how predictive analytics transforms raw data into meaningful, actionable knowledge.

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error,
mean_squared_error
np.random.seed(42)
n_samples = 200

voltage = np.random.uniform(200, 250, n_samples)            # in
Volts
current = np.random.uniform(5, 15, n_samples)               # in Amps
ambient_temp = np.random.uniform(20, 35, n_samples)         # °C
load = np.random.uniform(50, 100, n_samples)                # %
running_time = np.random.uniform(10, 120, n_samples)        # minutes
temp_rise = ( 0.05 * voltage +0.8 * current +0.6 * ambient_temp +
    0.1 * load + 0.07 * running_time +
    np.random.normal(0, 2, n_samples)  # noise
)

df = pd.DataFrame({
    "Voltage": voltage,
    "Current": current,
    "AmbientTemp": ambient_temp,
    "Load": load,
    "RunningTime": running_time,
    "TempRise": temp_rise
})

print(df.head())

X = df[["Voltage", "Current", "AmbientTemp", "Load", "RunningTime"]]
y = df["TempRise"]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("\nModel Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("R² Score:", r2_score(y_test, y_pred))
print("MAE:", mean_absolute_error(y_test, y_pred))
print("RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))
plt.scatter(y_test, y_pred, color="blue", alpha=0.6)
plt.plot([y_test.min(), y_test.max()],
```

```
         [y_test.min(), y_test.max()],
         color="red", linestyle="--")
plt.xlabel("Actual Temp Rise (°C)")
plt.ylabel("Predicted Temp Rise (°C)")
plt.title("Actual vs Predicted Temperature Rise")
plt.show()
new_data = pd.DataFrame({
    "Voltage": [230],
    "Current": [10],
    "AmbientTemp": [30],
    "Load": [80],
    "RunningTime": [60]
})
predicted_temp = model.predict(new_data)
print("\nPredicted Temperature Rise for new data:",
predicted_temp[0], "°C")
```

Results



Experiment - 6

Random Forest

Aim

1. To build and evaluate a Random Forest model for classification/regression by combining multiple decision trees.
2. To analyze how ensemble learning improves accuracy, reduces overfitting, and enhances prediction reliability.

Theory:

Random Forest is a powerful ensemble learning algorithm that operates by building a large number of decision trees during training. Instead of relying on a single tree—which can easily over fit or be sensitive to noise-Random Forest introduces randomness to create diverse trees. This is done in two ways:

1. **Bootstrap Sampling (Bagging):** Each tree is trained on a randomly selected subset of the original dataset, taken with replacement.
2. **Random Feature Selection:** At every split inside a tree, only a random subset of features is considered. This ensures that trees are less correlated and more diverse.

Once all the trees are built, the algorithm combines their predictions. For classification tasks, it uses majority voting, and for regression tasks, it takes the average of their outputs. This aggregation process reduces variance and improves prediction accuracy.

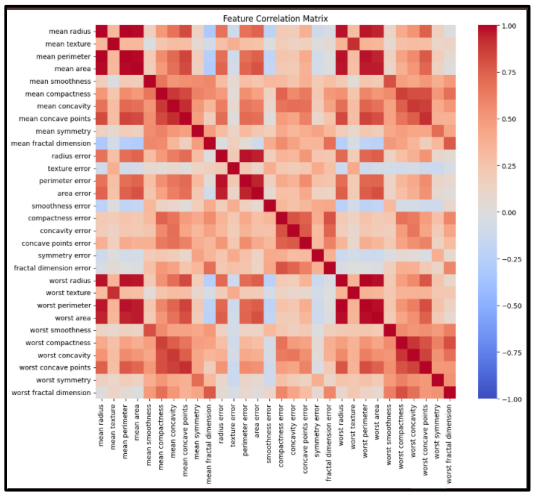
Random Forest is highly robust, works well with large and complex datasets, and handles missing values, noise, and non-linear relationships effectively. It also provides feature importance scores**,** allowing users to understand which variables influence the model's decisions the most. Overall, Random Forest improves reliability, reduces over fitting compared to a single decision tree, and provides strong generalization performance.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)
print("Feature names:", data.feature_names)
print("Target names:", data.target_names)
```

```
print("Shape of X:", X.shape)
print(X.head())
print(X.info())
print(X.describe())
plt.figure(figsize=(12, 10))
sns.heatmap(X.corr(), cmap='coolwarm', vmax=1.0, vmin=-1.0)
plt.title("Feature Correlation Matrix")
plt.show()
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y,
    test_size=0.2,
    random_state=42
)
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
dt_clf = DecisionTreeClassifier(random_state=42)
dt_clf.fit(X_train, y_train)

rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train, y_train)

from sklearn.metrics import classification_report
print(classification_report(y_test,dt_clf.predict(X_test)))
print(classification_report(y_test,rf_clf.predict(X_test)))
```

Results:



	precision	recall	f1-score	support
0	0.98	0.93	0.95	43
1	0.96	0.99	0.97	71
accuracy			0.96	114
macro avg	0.97	0.96	0.96	114
weighted avg	0.97	0.96	0.96	114

Experiment - 7

K-Means

Aim:

1. To implement the K-Means clustering algorithm for grouping data into K clusters based on similarity.
2. To analyze how the algorithm partitions data by minimizing the distance between points and their assigned cluster centroids.

Theory:

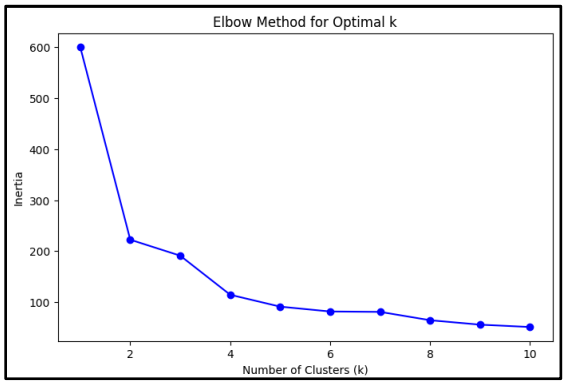
K-Means is an unsupervised clustering algorithm used to group similar data points into K distinct clusters. It works by initially selecting K centroids (either randomly or using strategies like K-means++). Each data point is then assigned to the cluster whose centroid is closest, forming K groups.Once the assignment is done, the centroid of each cluster is recalculated by taking the mean of all points within that cluster. This process of **assignment** and **centroid update** is repeated iteratively until the centroids no longer change significantly or a fixed number of iterations is reached.The algorithm aims to minimize the **Within-Cluster Sum of Squares (WCSS)**, also known as inertia, which measures how close the points in a cluster are to their centroid. K-Means is simple, efficient, and works well for large datasets, especially when clusters are spherical and well separated. However, it is sensitive to the choice of K and initial centroid positions.

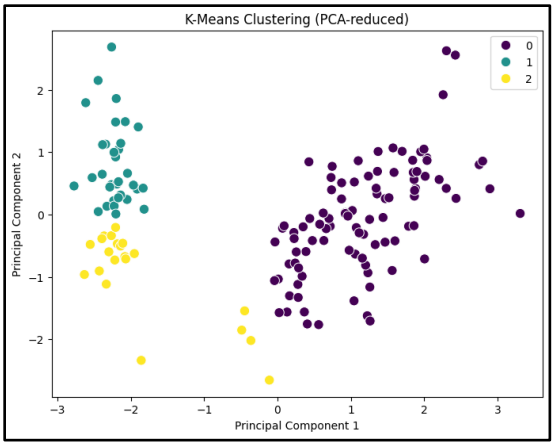
Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target, name="species")  # actual species labels (for
evaluation)
print("Dataset shape:", X.shape)
print(X.head())
print(X.describe())
sns.pairplot(X)
plt.show()
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

inertia = []
K = range(1, 11)

for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(K, inertia, 'bo-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
y_kmeans = kmeans.fit_predict(X_scaled)
sil_score = silhouette_score(X_scaled, y_kmeans)
print(f"Silhouette Score: {sil_score:.3f}")
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(8,6))
sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1], hue=y_kmeans,
palette='viridis', s=80)
plt.title("K-Means Clustering (PCA-reduced)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

Results:




K-Means Clustering (PCA-reduced)
Principal Component 2
Principal Component 1
0
1
2
−3
−2
−1
0
1
2
3
−2
−1
0
1
2

Experiment – 8

Anomaly Detection

Aim:

1. To detect unusual or abnormal data patterns using anomaly detection techniques.
2. To identify data points that deviate significantly from normal behavior, which may indicate errors, fraud, faults, or unusual events.

Theory:

Anomaly detection is a data analysis technique used to identify rare items, events, or observations that differ significantly from the majority of the data. These abnormal points are called anomalies or outliers. The goal is to learn the normal pattern of data and then detect any instance that does not follow this pattern.

Several methods are used for anomaly detection:

- **Statistical Methods:** Assume data follows a normal distribution; points far from the mean (based on z-score, variance) are considered anomalies.
- **Distance-Based Methods:** Points far away from others in terms of Euclidean distance are flagged as outliers.
- **Density-Based Methods:** Algorithms like LOF (Local Outlier Factor) detect anomalies based on how sparse or dense the neighborhood is.
- **Machine Learning Approaches:**
    - **Unsupervised methods** like **Isolation Forest**, where anomalies are isolated earlier during random partitioning.

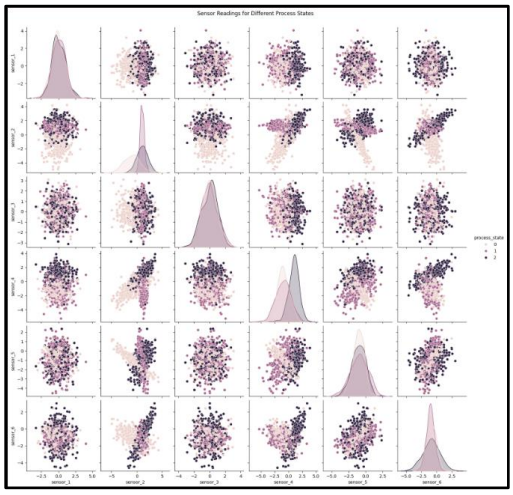
Anomaly detection is widely used in fraud detection, network security, medical diagnosis, sensor monitoring, and industrial fault detection. The main challenge is that anomalies are rare, unpredictable, and differ in nature from one dataset to another.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier, IsolationForest
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
X, y = make_classification(
    n_samples=1000,
    n_features=6,
```

```
    n_informative=4,
    n_redundant=0,
    n_clusters_per_class=1,
    n_classes=3,
    random_state=42
)

df = pd.DataFrame(X, columns=[f"sensor_{i}" for i in range(1, 7)])
df['process_state'] = y

print("Dataset shape:", df.shape)
print(df.head())
sns.pairplot(df, hue="process_state", diag_kind="kde")
plt.suptitle("Sensor Readings for Different Process States", y=1.02)
plt.show()
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.drop(columns=['process_state']))
y = df['process_state']
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
from sklearn.metrics import classification_report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Results:



```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.87      0.91        67
           1       0.84      0.89      0.87        66
           2       0.86      0.88      0.87        67

    accuracy                           0.88       200
   macro avg       0.88      0.88      0.88       200
weighted avg       0.88      0.88      0.88       200
```

Experiment - 9

ANN

Aim:

1. To design and implement an Artificial Neural Network model for learning patterns from data through training.
2. To analyze how an ANN processes inputs, adjusts weights, and makes predictions using forward and backward propagation.

Theory:

An Artificial Neural Network (ANN) is a computational model inspired by the structure and functioning of the human brain. It consists of interconnected units called **neurons**, arranged in layers: an **input layer**, one or more **hidden layers**, and an **output layer**. Each neuron receives inputs, multiplies them by weights, adds a bias, and applies an activation function (like ReLU, sigmoid, or tanh) to produce an output.

ANN training involves two main steps:

1. **Forward Propagation:**
   Input data flows through the network to generate outputs. The network's prediction is compared with the actual target to compute an error.
2. **Backward Propagation (Backpropagation):**
   The error is propagated backward through the network. Weights are updated using optimization algorithms like **Gradient Descent** to reduce the prediction error.

Through repeated training epochs, the network learns complex patterns, relationships, and decision boundaries. ANNs are widely used in classification, regression, pattern recognition, speech processing, and computer vision. They excel because they can approximate non-linear functions and learn directly from data without explicitly programmed rules.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

X, y = make_classification(
    n_samples=2000,
    n_features=6,
    n_informative=5,
    n_redundant=0,
    n_clusters_per_class=1,
    weights=[0.7, 0.3],   # 70% normal, 30% faulty
    class_sep=1.5,
    random_state=42
)

columns = ['voltage', 'current', 'resistance', 'temperature', 'frequency',
'phase_shift']
df = pd.DataFrame(X, columns=columns)
df['fault_status'] = y

print("Dataset Sample:")
print(df.head())

X = df.drop('fault_status', axis=1)
y = df['fault_status']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

print("\nTraining set size:", X_train.shape)
print("Testing set size:", X_test.shape)

model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
```

```
    Dense(1, activation='sigmoid')
])

model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=50,
    batch_size=32,
    verbose=1
)

y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)
```

Results:

```
     Classification Report:
                   precision    recall  f1-score   support

               0       0.99      1.00      1.00       279
               1       1.00      0.98      0.99       121

        accuracy                           0.99       400
       macro avg       1.00      0.99      0.99       400
    weighted avg       1.00      0.99      0.99       400
```

Experiment - 10

LSTM

Aim:

1. To implement an LSTM-based neural network for learning long-term dependencies in sequential data.
2. To analyze how LSTM cells store, update, and forget information using gated mechanisms for improved sequence prediction.

Theory:

Long Short-Term Memory (LSTM) is a special type of Recurrent Neural Network (RNN) designed to overcome the limitations of traditional RNNs, especially the vanishing and exploding gradient problems. LSTMs are highly effective for sequence-based tasks such as time-series forecasting, language modeling, and speech recognition.

The LSTM architecture is built around a memory cell that can preserve information over long periods. It uses three gates:

1. **Forget Gate:** Decides what information from the previous cell state should be removed.
2. **Input Gate:** Determines which new information should be stored in the cell state.
3. **Output Gate:** Controls what part of the cell state should be output to the next layer or next time step.

These gates use sigmoid and tanh activation functions to regulate information flow, allowing the network to learn both short-term and long-term patterns effectively. Because of this controlled memory mechanism, LSTMs perform exceptionally well on sequential and time-dependent data where context and order are important.

Code:

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt

max_features = 5000
max_len = 200

(x_train, y_train), (x_test, y_test) =
imdb.load_data(num_words=max_features)

x_train = pad_sequences(x_train, maxlen=max_len)
```

```
x_test = pad_sequences(x_test, maxlen=max_len)

model = Sequential([
    Embedding(max_features, 128, input_length=max_len),
    LSTM(64, dropout=0.2, recurrent_dropout=0.2),
    Dense(1, activation='sigmoid')
])

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

print("Training model...")
history = model.fit(x_train, y_train,
                    batch_size=64,
                    epochs=3,
                    validation_split=0.2,
                    verbose=1)

loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f"\nTest Accuracy: {accuracy*100:.2f}%")

plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

Results:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Training model...
Epoch 1/3
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
313/313 ━━━━━━━━━━━━━━━━━━━━ 110s 335ms/step - accuracy: 0.6814 - loss: 0.5696 - val_accuracy: 0.8334 - val_loss: 0.3899
Epoch 2/3
313/313 ━━━━━━━━━━━━━━━━━━━━ 97s 311ms/step - accuracy: 0.8459 - loss: 0.3619 - val_accuracy: 0.8494 - val_loss: 0.3531
Epoch 3/3
313/313 ━━━━━━━━━━━━━━━━━━━━ 97s 310ms/step - accuracy: 0.8706 - loss: 0.3147 - val_accuracy: 0.8510 - val_loss: 0.3655

Test Accuracy: 84.35%
```