TAMILNADU GOVERNMENT POLYTECHNIC COLLEGE

MADURAI-11

DEPARTMENT OF WEB DESIGNING

DATA SCIENCE

Name:_____

Reg. No. _____

Roll No. _____

Semester: _____

TAMILNADU GOVERNMENT POLYTECHNIC COLLEGE

MADURAI-11

DEPARTMENT OF WEB DESIGNING

BONAFIDE CERTIFICATE

Name: ……. Reg. No…….
Roll No. …… Semester: …….

Certify that this is a bonafide record of work done by …… in the **DATA SCIENCE** *during the year 2025 - 2026*

Signature of Lab in-charge Head of the Department

Submitted for the practical examination held on ……

Internal Examiner External Examiner

List of Experiments

Ex No	Date	Name	Page	Marks	Signature
1		STORE STUDENT DETAILS IN PYTHON LIST / TUPLE			
2		CONVERT PYTHON LIST TO NumPy ARRAY			
3		PERFORM NumPy ARRAY OPERATIONS			
4		LOAD DATA IN PANDAS DATAFRAME			
5		FIND MISSING VALUES AND FILL IT			
6		COMBINE TWO DATAFRAME, REMOVE DUPLICATE ROWS AND RENAME INDEXES			
7		VISUALIZE AVERAGE VALUE OF EACH FEATURE OF SETOSA IRIS CLASS			
8		PERFORM PAIRPLOT MULTIVARIATE ANALYSIS AND SAVE AS JPEG			
9		ANALYZE DECISION TREE USING IRIS DATASET			
10		ANALYZE K-MEANS CLUSTERING USING IRIS DATASET			

STORE STUDENT DETAILS IN PYTHON LIST / TUPLE

Ex No: 1

Aim: To create a python list / tuple to store student details and print the values.

Algorithm:

Step1: Initialize Data: Create a list named students containing tuples. Each tuple stores specific attributes: (Roll No, Name, Department, Type, Percentage).

Step2: Display the student details aligned in a tabular format

Step3: Initialize Search: Define a variable search_roll with the roll number you wish to find

Step4: Using Loop and If condition search for the given roll number.

Step5: If the given number is found display the student details

Source Code:

```
# Storing multiple students using a list of tuples
students = [
   (101, "Anbananth", "Computer Engineering", "Regular", 88.5),
   (102, "Bharath", "Computer Engineering", "Regular", 76.2),
   (103, "Chandru", "Computer Engineering", "Regular", 91.8),
   (104, "Dinesh", "Computer Engineering", "Regular", 79.3)
]

print("\n" + "=" * 80)
print("STUDENT DETAILS")
print("=" * 80)

# Print all students in tabular format
print(f"{'Roll No':<10} {'Name':<15} {'Department':<20} {'Type':<15}
        {'Percentage':<10}")
print("-" * 80)

for student in students:
   per_str = f"{student[4]:.2f}%"
   print(f"{student[0]:<10} {student[1]:<15} {student[2]:<20} {student[3]:<15}
        {per_str:^10}")

# Find student by roll number
search_roll = 102
print(f"\nSearching for student with Roll No: {search_roll}")
for student in students:
   if student[0] == search_roll:
      print(f"Found: {student[1]} - {student[3]} - {student[4]:.2f}%")
      break
```

Result:

Thus a python list / tuple are created to store student details and print the values.

CONVERT PYTHON LIST TO NumPy ARRAY

Ex No: 2

Aim: To create a python list / tuple, convert it as NumPy array and perform slicing operations.

Algorithm:

Step1: Install NumPy library using
pip install numpy

Step2: Import the ***numpy*** library as ***np***.

Step3: Initialize Data: Create a list of tuples named *students_list*.

Step4: Convert students_list into a NumPy array called np_students.

Step5: Perform Data Slicing Operations Row Slicing, Column Indexing and Negative Indexing.

Step6: Display the Sliced Data.

Source Code:

```
import numpy as np

students_list = [
   (101, "Anbananth", "Computer Engineering", "Regular", 88.5),
   (102, "Bharath", "Computer Engineering", "Regular", 76.2),
   (103, "Chandru", "Computer Engineering", "Regular", 91.8),
   (104, "Dinesh", "Computer Engineering", "Regular", 79.3)
]

np_students = np.array(students_list)

print(f"\n{'Full Student NumPy Array':=^80}\n")
print(np_students)

slice_1 = np_students[0:2]

slice_2 = np_students[:, [1, 4]]

slice_3 = np_students[-1:]

print(f"\n{'Slice 1: First Two Students':*^80}\n")
print(f"{'Roll No':<10} {'Name':<15} {'Department':<20} {'Type':<15}
        {'Percentage':<10}")
print("-" * 80)
for student in slice_1:
   per_str = f"{student[4]}%"
   print(f"{student[0]:<10} {student[1]:<15} {student[2]:<20} {student[3]:<15}
        {per_str:^10}")
```

```
print(f"\n{'Slice 2: Names and Percentage Only':*^80}\n")
print(f"{'Name':<15} {'Percentage':<10}")
print("-" * 80)
for student in slice_2:
    per_str = f"{student[1]}%"
    print(f"{student[0]:<15} {per_str:^10}")

print(f"\n{'Slice 3: Last Student Record':*^80}\n")
print(f"{'Roll No':<10} {'Name':<15} {'Department':<20} {'Type':<15}
        {'Percentage':<10}")
print("-" * 80)
for student in slice_3:
    per_str = f"{student[4]}%"
    print(f"{student[0]:<10} {student[1]:<15} {student[2]:<20} {student[3]:<15}
        {per_str:^10}")
```

Result:

Thus a python list / tuple is created, converted into NumPy array and slicing operation is performed.

PERFORM NumPy ARRAY OPERATIONS

Ex No: 3

Aim: To load marklist data from csv file into NumPy array and perform array operations.

Algorithm:

Step1: Import the ***numpy*** library as ***np***.

Step2: Load CSV file using ***np.getfromtxt()*** method

Step3: Display the Loaded Data.

Step4: Select only Numeric columns for the data.

Step5: Perform the various Array Inspection Operations and display the details.

Source Code:

```
import numpy as np

print("\n LOADING CSV FILE INTO NUMPY ARRAY")
print("-" * 80)

try:
   # Load CSV - adjust delimiter if needed (comma by default)
   data = np.genfromtxt('stud_marks.csv', delimiter=',', dtype=None,encoding='utf-8',
          names=True)

except FileNotFoundError:
   print("File 'stud_marks.csv' not found. Creating sample data...")

# Display loaded data

print("\n LOADED DATA PREVIEW")
print("-" * 80)

# Display the structured array
print("\nFirst 3 rows of data:")
for i in range(min(3, len(data))):
   print(f"Row {i}: {data[i]}")

# Display column names
print(f"\nColumn names: {data.dtype.names}")
print(f"Number of columns: {len(data.dtype.names)}")


# Convert to regular NumPy array (numerical data only)
print("\n CONVERTING TO NUMERICAL NUMPY ARRAY")
print("-" * 40)
```

```
try:
    # Get numerical data (Subject1 to Percentage)
    numerical_data = np.array([list(row)[2:] for row in data])

    print("Numerical array created successfully!")
    print(f"Array shape: {numerical_data.shape}")
    print("\nNumerical Data Preview:")
    print(numerical_data[:3])  # Show first 3 rows

except Exception as e:
    print(f"Error creating numerical array: {e}")


print("\n ARRAY INSPECTION OPERATIONS")
print("-" * 40)

# Create a dictionary of inspection operations
inspection_results = {
    "len()": len(numerical_data),
    "ndim": numerical_data.ndim,
    "size": numerical_data.size,
    "dtype": numerical_data.dtype,
    "shape": numerical_data.shape
}

# Print all operations
print(f"{'Operation':<15} {'Result':<30}")
print(f"{'-' * 15} {'-' * 30}")
for operation, result in inspection_results.items():
    print(f"{operation:<15} {str(result):<30}")
```

Result:

Thus marklist data from csv file is loaded into NumPy array and array operations are performed.

LOAD DATA IN PANDAS DATAFRAME

Ex No: 4

Aim: To load data into pandas data frame and perform statistical operations.

Algorithm:

Step1: Install pandas using the following command
pip install pandas
Step2: Import the ***pandas*** and ***numpy*** library.
Step3: Load the Students Marks CSV file into the pandas DataFrame
Step4: Select only marks column.
Step5: Perform the statistical operations and display the results.

Source Code:

```
import pandas as pd
import numpy as np

#Load the CSV file into a pandas DataFrame
df = pd.read_csv("stud_marks.csv")

print("=== DataFrame Preview ===")
print(df.head())

#Select only marks columns (exclude rollno and name)
marks_columns = ['subj1', 'subj2', 'subj3', 'subj4', 'subj5', 'total', 'per']
marks_df = df[marks_columns]

print("\n=== Minimum of marks columns (min) ===")
print(marks_df.min())

print("\n=== Maximum of marks columns (max) ===")
print(marks_df.max())

print("\n=== Cumulative sum of marks columns (cumsum) ===")
print(marks_df.cumsum().head())  # Show first 5 rows for readability

print("\n=== Mean of marks columns (mean) ===")
print(marks_df.mean())

print("\n=== Median of marks columns (median) ===")
print(marks_df.median())

print("\n=== Correlation matrix of marks columns (corr) ===")

print("Using numpy.corrcoef()")
```

```
subjects = ['subj1', 'subj2', 'subj3', 'subj4', 'subj5', 'per']
subject_data = df[subjects].values.T  # Transpose for corrcoef

correlation_matrix = np.corrcoef(subject_data)

print("Correlation Matrix (Subjects only):")
print(f"{":<10} {'subj1':<10} {'subj2':<10} {'subj3':<10} {'subj4':<10} {'subj5':<10}
        {'per':<10}")
print("-" * 70)
for i, subject in enumerate(subjects):
   print(f"{subject:<10}", end="")
   for j in range(len(subjects)):
      print(f"{correlation_matrix[i][j]:<10.3f}", end="")
   print()

print("\n=== Standard deviation of marks columns (std) ===")
print(marks_df.std())
```

Result:

Thus data is loaded into pandas data frame and statistical operations were performed.

FIND MISSING VALUES AND FILL IT

Ex No: 5

Aim: To load data into pandas data frame and find the missing values and replace them with suitable values.

Algorithm:

Step1: Import the ***pandas*** and ***numpy*** library.

Step2: Load the "***Titanic_Train.csv"*** into the pandas DataFrame.

Step3: Display initial missing values from the loaded data.

Step4: Fill the missing values with suitable method.

Step5: Verify all the missing values are replaced.

Step6: Display Cleaned Data Sample.

Source Code:

```
import pandas as pd
import numpy as np

# Load the Titanic dataset
df = pd.read_csv('Titanic_Train.csv')

# List missing values in each column
print(f"--- [Initial Missing Values] ---\n{df.isnull().sum()}\n")

# Fill missing values with suitable default values

# For 'Age': Use the Median mean
df['Age'] = df['Age'].fillna(df['Age'].median())

# For 'Embarked': Use the Mode
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mode()[0])

# For 'Cabin': Since ~77% is missing in real data, use a placeholder
df['Cabin'] = df['Cabin'].fillna('Unknown')

# Verify results
print(f"--- [Missing Values After Imputation] ---\n{df.isnull().sum()}\n")

# Display the first few rows of cleaned data
print("--- [Cleaned Data Sample] ---")
print(df[['Name', 'Age', 'Cabin', 'Embarked']].head())
```

Result:

Thus data is loaded into pandas data frame and missing values are identified and replaced with suitable values.

COMBINE TWO DATAFRAME, REMOVE DUPLICATE ROWS AND RENAME INDEXES

Ex No: 6

Aim: To load two CSV files into two DataFrame, combine the DataFrame and remove duplicate and rename indexes.

Algorithm:

Step1: Import the ***pandas*** library.

Step2: Load the CSV files "***customer_Transaction.csv"*** and "***customer_Demograph.csv"*** into two pandas DataFrame.

Step3: Display original DataFrames.

Step4: Combine DataFrames using Merge function on "customer_id" and store the result in a new dataframe.

Step5: Find and remove duplicate rows from the combined data.

Step6: Rename the indexes.

Source Code:

```
import pandas as pd

# Load the two CSV files into DataFrames d1 and d2
d1 = pd.read_csv("customer_Transaction.csv")  # Transaction data
d2 = pd.read_csv("customer_Demograph.csv")    # Demographic data

print("=== Original DataFrames ===")
print(f"d1 (Transactions) shape: {d1.shape}")
print(f"d2 (Demographics) shape: {d2.shape}")
print("\nd1 preview:")
print(d1.head(2))
print("\nd2 preview:")
print(d2.head(2))

# Combine both DataFrames using merge on customerid
combined_df = pd.merge(d1, d2, on='customer_id', how='inner')

print(f"\n=== Combined DataFrame shape: {combined_df.shape}")

# Find and remove duplicate rows
print(f"\n=== Before duplicate removal: {combined_df.shape[0]} rows")

# Check for duplicates across all columns
duplicates_before = combined_df.duplicated().sum()
print(f"Duplicate rows found: {duplicates_before}")

# Remove duplicate rows
```

```
combined_df_no_duplicates = combined_df.drop_duplicates()
print(f"After duplicate removal: {combined_df_no_duplicates.shape[0]} rows")

# Rename indexes to start from 1 (reset_index)
final_df = combined_df_no_duplicates.reset_index(drop=True)
final_df.index += 1  # Make indexes start from 1 instead of 0

print(f"\n=== Final DataFrame (after cleaning) ===")
print(f"Shape: {final_df.shape}")
print("New index range:", final_df.index.min(), "to", final_df.index.max())
print("\nFinal preview:")
print(final_df.head())

print("\n=== Verification ===")
print("No missing customerid:", final_df['customer_id'].isnull().sum() == 0)
print("No duplicates:", final_df.duplicated().sum() == 0)
```

Result:

Thus two CSV files are loaded into two DataFrames, the DataFrames are combined duplicates are removed and indexes are renamed.

VISUALIZE AVERAGE VALUE OF EACH FEATURE OF SETOSA IRIS CLASS

Ex No: 7

Aim: To load IRIS dataset and visualize average value of each feature of SETOSA IRIS Class.

Algorithm:

Step1: Install the Matplotlib library using the following command
pip install matplotlib

Step2: Import the ***pandas***, ***matplotlib*** and ***numpy*** library.

Step3: Load IRIS Dataset into the DataFrame.

Step4:Display Dataset information and filter **Setosa Class**.

Step5: Calculate Average Features for **Setosa Class**.

Step6: Create Bar Chart for Visualization using matplotlib.

Step7: Display the created Bar Chart.

Source Code:

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

# Load IRIS dataset (filename fixed)
df = pd.read_csv("iris.csv")

print("=== Actual Column Names ===")
print(df.columns.tolist())
print("\nDataset shape:", df.shape)
print("\nFirst 3 rows:")
print(df.head(3))

# Filter Setosa class (using correct column name)
setosa_df = df[df['Species'] == 'Iris-setosa']  # 'Species' not 'class'
print(f"\nSetosa observations: {len(setosa_df)}")

# Calculate averages (standard Iris column names)
setosa_avg = setosa_df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',
        'PetalWidthCm']].mean()
print("\nSetosa Average Features:")
print(setosa_avg)

# Bar chart visualization
plt.figure(figsize=(10, 6))
features = ['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width']
colors = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4']

bars = plt.bar(features, setosa_avg.values,
```

```
                color=colors,
                linewidth=3,      # Suitable line width
                edgecolor='black',
                alpha=0.8)

    plt.title('Average Feature Values - Iris Setosa Class', fontsize=16, fontweight='bold',
              pad=20)
    plt.xlabel('Features', fontsize=14, fontweight='bold')
    plt.ylabel('Average Value (cm)', fontsize=14, fontweight='bold')
    plt.grid(axis='y', alpha=0.3, linestyle='--')

    # Value labels on bars
    for bar, value in zip(bars, setosa_avg.values):
        plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.02,
                 f'{value:.2f}', ha='center', va='bottom', fontweight='bold', fontsize=12)

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

Result:

Thus IRIS dataset is loaded and average value of each feature of SETOSA IRIS Class is visualized.

PERFORM PAIRPLOT MULTIVARIATE ANALYSIS AND SAVE AS JPEG

Ex No: 8

Aim: To load IRIS dataset and visualize all columns relationships using pairplot multivariate analysis.

Algorithm:

Step1: Install the seaborn library using the following command
pip install seaborn

Step2: Import the ***pandas***, ***matplotlib*** and ***seaborn*** library.

Step3: Load IRIS Dataset into the DataFrame.

Step4: Prepare the data for Pair Plot and create Pair Plot using Seaborn library.

Step5: Add necessary details to the Pair Plot and save the Pair Plot as JPEG.

Source Code:

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

# Load IRIS dataset
df = pd.read_csv("iris.csv")

# Prepare data (numeric columns + Species)
numeric_cols = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']
df_pair = df[numeric_cols + ['Species']]

# Create Pair Plot
g = sns.pairplot(df_pair, hue='Species', diag_kind='hist',
                 height=2.5,
                 plot_kws={'linewidth': 1.5, 'alpha': 0.7},
                 diag_kws={'alpha': 0.7})

# Add suptitle to the PairGrid's figure object
g.fig.suptitle('IRIS Dataset - Multivariate Pair Plot Analysis',
               fontsize=16, fontweight='bold', y=1.02)

# Save as JPEG
plt.savefig('iris_pairplot_multivariate.jpg', dpi=300, bbox_inches='tight',
            facecolor='white', edgecolor='none')
print("\n  Pair plot saved as 'iris_pairplot_multivariate.jpg'")

# Display the plot
plt.show()
```

Result:

Thus IRIS dataset is loaded and all columns relationships are visualized using pairplot multivariate analysis.

ANALYZE DECISION TREE USING IRIS DATASET

Ex No: 9

Aim: To analyze Decision Tree Classification model using IRIS dataset.

Algorithm:

Step1: Install the sklearn library using the following command
pip install scikit-learn

Step2: Import the ***pandas***, ***matplotlib***, ***seaborn*** and ***sklearn*** library.

Step3: Load IRIS Dataset into the DataFrame.

Step4: Prepare the data by selecting features X and target Y.

Step5: Split the data into training and testing data.

Step6: Build the decision tree using the training data.

Step7: Test the decision tree using the testing data.

Step8: Display the Decision Tree and Confusion Matrix

Source Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix

print("IRIS FLOWER CLASSIFICATION")
print("=" * 50)

# Load Data (150 flowers, 3 types)
df = pd.read_csv('Iris.csv')
print("Data loaded! 150 flowers from 3 species")
print(df['Species'].value_counts())

# Prepare Data (4 measurements, 1 target)
X = df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]  #
    Measurements
y = df['Species']  # Flower type

# Split Data (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"\nTraining: {len(X_train)} flowers")
print(f"Testing:  {len(X_test)} flowers")

# Build Decision Tree (Simple tree, depth=3)
tree = DecisionTreeClassifier(max_depth=3, random_state=42)
```

```
tree.fit(X_train, y_train)

# Test the Tree
predictions = tree.predict(X_test)
accuracy = accuracy_score(y_test, predictions)

print(f"\n🎯 ACCURACY: {accuracy:.1%} CORRECT!")
print("Tree correctly identifies flower species!")

# PLOT 1 - Decision Tree (How it decides)
plt.figure(figsize=(12, 8))
plot_tree(tree, feature_names=X.columns, class_names=['Setosa', 'Versicolor', 'Virginica'],
          filled=True, rounded=True)
plt.title('DECISION TREE: How it identifies flowers')
plt.savefig('tree_simple.png', dpi=200)
plt.show()

# PLOT 2 - Confusion Matrix (Right/Wrong predictions)
cm = confusion_matrix(y_test, predictions)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens',
            xticklabels=['Setosa', 'Versicolor', 'Virginica'],
            yticklabels=['Setosa', 'Versicolor', 'Virginica'])
plt.title('RIGHT vs WRONG Predictions')
plt.ylabel('Actual Flower')
plt.xlabel('Predicted Flower')
plt.savefig('confusion_simple.png', dpi=200)
plt.show()
```

Result:

Thus Decision Tree Classification model is analyzed using IRIS dataset.

ANALYZE K-MEANS CLUSTERING USING IRIS DATASET

Ex No: 10

Aim: To analyze K-Means Clustering model using IRIS dataset.

Algorithm:

Step1: Import the ***pandas***, ***matplotlib***, ***seaborn*** and ***sklearn*** library.
Step2: Load IRIS Dataset into the DataFrame.
Step3: Analyze K-Means clustering using Elbow Method and Silhouette.
Step4: Prepare the data by selecting features X and target Y.
Step5: Plot Elbow Method and Silhouette Score.
Step6: Create K-Means Clustering using k=3.
Step7: Create Clustering Visualization.
Step8: Display the final results.

Source Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, confusion_matrix
from sklearn.preprocessing import StandardScaler

# Load Data
df = pd.read_csv('iris.csv')
X = df.drop(['Id', 'Species'], axis=1)
true_labels = df['Species']

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

# K-Means Analysis - Elbow Method & Silhouette
wcss = []
silhouette_avg = []
k_range = range(2, 11)

for k in k_range:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
    silhouette_avg.append(silhouette_score(X_scaled, kmeans.labels_))

# Plotting Elbow and Silhouette
```

```
fig, ax1 = plt.subplots(figsize=(10, 5))

ax1.set_xlabel('Number of clusters (k)')
ax1.set_ylabel('WCSS', color='tab:red')
ax1.plot(k_range, wcss, marker='o', color='tab:red', label='WCSS (Elbow)')
ax1.tick_params(axis='y', labelcolor='tab:red')

ax2 = ax1.twinx()
ax2.set_ylabel('Silhouette Score', color='tab:blue')
ax2.plot(k_range, silhouette_avg, marker='s', color='tab:blue', label='Silhouette Score')
ax2.tick_params(axis='y', labelcolor='tab:blue')

plt.title('K-Means Analysis: Elbow Method and Silhouette Score')
plt.savefig('kmeans_analysis.png')

# Final Model (k=3)
kmeans = KMeans(n_clusters=3, n_init=10, random_state=42)
clusters = kmeans.fit_predict(X_scaled)
df['Cluster'] = clusters

# Mapping clusters to species for analysis (Approximate)
# Setosa is usually very distinct.
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='PetalLengthCm', y='PetalWidthCm', hue='Cluster',
        palette='viridis', style='Species', s=100)
plt.title('K-Means Clustering vs True Species Labels')
plt.savefig('iris_clustering_final.png')
plt.show()

print("WCSS for k=3:", kmeans.inertia_)
print("Silhouette Score for k=3:", silhouette_score(X_scaled, clusters))
```

Result:

Thus K-Means Clustering model is analyzed using IRIS dataset.