Machine Learning Lab Sheet02

Step 1: Working with Libraries

Code Example 1: Importing Libraries

```
# Importing common libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
```

Step2: Data Transformation: Transform data through operations like feature scaling (e.g., Min-Max scaling, Z-score normalization), one-hot encoding.

```
import pandas as pd

from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder, OneHotEncoder


# Create a sample DataFrame with different data types

data = {

    'Age': [25, 32, 19, 45, 28],

    'Salary': [50000, 60000, 45000, 75000, 55000],

    'Gender': ['Male', 'Female', 'Male', 'Male', 'Female']

}


df = pd.DataFrame(data)


# Display the original DataFrame

print("Original DataFrame:")

print(df)


# Min-Max Scaling for 'Age' and 'Salary' columns

scaler = MinMaxScaler()

df[['Age', 'Salary']] = scaler.fit_transform(df[['Age', 'Salary']])


# Display the DataFrame after Min-Max scaling

print("\nDataFrame after Min-Max scaling for 'Age' and 'Salary' columns:")

print(df)


# Z-score Normalization for 'Age' and 'Salary' columns
```

```
scaler = StandardScaler()
df[['Age', 'Salary']] = scaler.fit_transform(df[['Age', 'Salary']] )

# Display the DataFrame after Z-score normalization
print("\nDataFrame after Z-score normalization for 'Age' and 'Salary' columns:")
print(df)

# One-Hot Encoding for 'Gender' column
df = pd.get_dummies(df, columns=['Gender'], prefix=['Gender'])

# Display the DataFrame after one-hot encoding
print("\nDataFrame after one-hot encoding for 'Gender' column:")
print(df)
```

Step3: Perform the previous data transformation techniques using the iris dataset and show the results at each step of data transformation.

Step4: perform the following data cleaning example that demonstrates some common data cleaning tasks such as handling missing values, removing outliers, and converting data types:

```
import pandas as pd
import numpy as np

# Create a sample DataFrame with missing values and outliers
data = {
    'Age': [25, 32, 19, 45, 999],  # 999 is an outlier
    'Salary': [50000, 60000, 45000, np.nan, 55000],  # NaN is missing
    'Gender': ['Male', 'Female', 'Male', 'Male', 'Female'],
    'Education': ['Bachelors', 'Masters', 'PhD', 'Bachelors', 'Masters'],
}

df = pd.DataFrame(data)

# Display the original DataFrame
print("Original DataFrame:")
print(df)
```

```
# Handling Missing Values
df['Salary'].fillna(df['Salary'].mean(), inplace=True)

# Remove Outliers (considering values greater than 100 as outliers)
df = df[df['Age'] < 100]

# Convert Data Types (e.g., converting 'Age' to int)
df['Age'] = df['Age'].astype(int)

# Display the cleaned and standardized DataFrame
print("\nCleaned and Standardized DataFrame:")
print(df)
```

Step5: Data visualization is a powerful way to gain insights data. Perform the following example which is:

- create a sample DataFrame with random data for 'Age', 'Salary', 'Education', and 'Gender'.
- use Matplotlib and Seaborn to create visualizations.
- The first visualization is a **histogram** that shows the distribution of 'Age' with a kernel density estimation (KDE) plot.
- The second visualization is a **scatter plot** that explores the relationship between 'Age' and 'Salary', with points coloured by 'Gender'.
- The third visualization is a **bar plot** that displays the distribution of 'Education' levels in the dataset.

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

# Create a sample DataFrame
data = {
    'Age': np.random.randint(20, 60, 100),
    'Salary': np.random.randint(30000, 100000, 100),
    'Education': np.random.choice(['High School', 'Bachelor', 'Master', 'PhD'], 100),
    'Gender': np.random.choice(['Male', 'Female'], 100)
}

df = pd.DataFrame(data)

# Visualize the distribution of 'Age' using a histogram
plt.figure(figsize=(8, 6))
plt.title("Age Distribution")
sns.histplot(data=df, x='Age', kde=True)
plt.show()

# Visualize the relationship between 'Age' and 'Salary' using a scatter plot
plt.figure(figsize=(8, 6))
plt.title("Age vs. Salary")
sns.scatterplot(data=df, x='Age', y='Salary', hue='Gender')
plt.xlabel("Age")
plt.ylabel("Salary")
plt.show()

# Visualize the distribution of 'Education' using a bar plot
plt.figure(figsize=(8, 6))
plt.title("Education Distribution")
sns.countplot(data=df, x='Education', order=['High School', 'Bachelor', 'Master', 'PhD'])
plt.xticks(rotation=45)
plt.show()
```

Step 6: Use the previous data visualization code with the uploaded iris dataset to show:

- The first visualization is a histogram that shows the distribution of 1st attribute with a kernel density estimation (KDE) plot.
- The second visualization is a scatter plot that explores the relationship between 1st attribute and 3rd attribute.
- The third visualization is a bar plot that displays the distribution of 2nd attributes levels in the iris dataset.

Step 7: Features Selection Technique:

In this example using the built-in iris dataset to define the most important features using decision tree classifier on the data to calculate feature importance’s:

- We load the Iris dataset from scikit-learn.
- We split the dataset into features (X) and the target (y).
- We train a decision tree classifier on the data to calculate feature importance’s.
- We create a DataFrame containing feature names and their importance scores, and then sort it in descending order of importance.
- We use Seaborn to create a bar plot that visualizes the feature importance’s.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Load the Iris dataset
iris = load_iris()
data = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
            columns=iris['feature_names'] + ['target'])

# Split the dataset into features and target
X = data[iris['feature_names']]
y = data['target']

# Fit a decision tree classifier to the data to get feature importances
tree_classifier = DecisionTreeClassifier(random_state=0)
tree_classifier.fit(X, y)

# Get feature importances
feature_importances = tree_classifier.feature_importances_

# Create a DataFrame with feature names and their importance scores
feature_importance_df = pd.DataFrame({'Feature': iris['feature_names'], 'Importance': feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
```

```
# Visualize feature importances

plt.figure(figsize=(8, 6))

sns.barplot(x='Importance', y='Feature', data=feature_importance_df)

plt.title("Feature Importance")

plt.show()
```