Data Analytics Practical Compilation

PR1: Problem Statement

Data Wrangling I:

Perform various preprocessing operations like checking missing values, data normalization, and converting categorical to numerical variables using an open-source dataset.

Code:

```
#!/usr/bin/env python
# coding: utf-8
# In[1]:
# 1. Import necessary libraries
import pandas as pd
from sklearn.preprocessing import LabelEncoder
# In[4]:
df = pd.read_csv('/content/sample_data/Student_performance_data _.csv')
# In[5]:
print('Data                                                                   Source
:https://www.kaggle.com/api/v1/datasets/download/rabieelkharoua/Students_performance_data_')
# In[7]:
#check missing values
print(df.isnull().sum())
# In[8]:
#get basic statistics
print(df.describe())
# In[9]:
#check dimention and data types
print(df.shape)
print(df.dtypes)
# In[12]:
#data formatting
df['Extracurricular']=df['Extracurricular'].astype(float)
# In[13]:
print(df.dtypes)
# In[17]:
le=LabelEncoder()
for col in df.select_dtypes(include='object').columns:
    df[col] = le.fit_transform(df[col])
# In[18]:
print(df.dtypes)
# In[19]:
print(df.head())
# In[ ]:
```

Data Analytics Practical Compilation

PR2: Problem Statement

Data Wrangling II:

Create an Academic Performance dataset and perform data cleaning, outlier detection, and transformation techniques.

Code:

```
#!/usr/bin/env python
# coding: utf-8
# In[1]:
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# In[2]:
# Step 1: Create Sample Dataset
data = {
    'Student_ID': [1, 2, 3, 4, 5, 6],
    'Math_Score': [85, 90, np.nan, 40, 200, 95],
    'English_Score': [78, 82, 75, 60, 58, None],
    'Science_Score': [88, 92, 85, np.nan, 65, 100],
    'Attendance_%': [95, 80, 85, 50, 110, 88]
}
df = pd.DataFrame(data)
# In[3]:
# Step 2: Handle Missing Values
df.fillna({
    'Math_Score': df['Math_Score'].mean(),
    'English_Score': df['English_Score'].mean(),
    'Science_Score': df['Science_Score'].mean()
}, inplace=True)
# In[4]:
# Fix inconsistencies: Attendance > 100
df['Attendance_%'] = np.where(df['Attendance_%'] > 100, 100, df['Attendance_%'])
# In[5]:
# Step 3: Detect and Handle Outliers using IQR
def treat_outliers(column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    df[column] = np.where(df[column] < lower, lower,
                 np.where(df[column] > upper, upper, df[column]))
for col in ['Math_Score', 'English_Score', 'Science_Score', 'Attendance_%']:
    treat_outliers(col)
# In[6]:
# Step 4: Data Transformation (Log) on Math_Score
df['Log_Math_Score'] = np.log1p(df['Math_Score'])
# In[12]:
```

```
# Plot: Before vs After Transformation
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
sns.histplot(df['Math_Score'], kde=True)
plt.title('Math Score - Before Transformation')
# In[13]:
plt.subplot(1, 2, 2)
sns.histplot(df['Log_Math_Score'], kde=True)
plt.title('Math Score - After Log Transformation')
# In[ ]:
```

Data Analytics Practical Compilation

PR3: Problem Statement

Data Wrangling III:

Perform basic data wrangling tasks such as handling missing values and filtering on students_data.csv.

Code:

```
#!/usr/bin/env python
# coding: utf-8
# In[ ]:
# Step 1: Create the students_data.csv file (optional if you're generating it);
import pandas as pd
# Sample data dictionary
data = {
    "Name": ["Alice", "Bob", "Charlie", "David", "Eve", "Frank", "Grace", "Heidi"],
    "Age": [20, 21, 22, 20, 23, 21, 22, 24],
    "Marks": [85, 78, None, 92, 67, 88, None, 95]
}
# In[ ]:
# Create DataFrame and save to CSV
df = pd.DataFrame(data)
df.to_csv("students_data.csv", index=False)
# In[ ]:
# 1. Load the CSV file
df = pd.read_csv("students_data.csv")
# In[ ]:
# 2. Display the first 5 rows
print("First 5 rows:",df.head())
# In[ ]:
# 3. Get the number of rows and columns
rows, cols = df.shape
print(f"\nTotal Rows: {rows}, Total Columns: {cols}")
# In[ ]:
# 4. List all column names
print("\nColumn Names:")
df.columns.tolist()
# In[ ]:
# 5. Check for missing values
print("\nMissing Values:")
df.isnull().sum()
# In[ ]:
# 6. Replace missing values in "Marks" with average
average_marks = df["Marks"].mean()
df["Marks"] = df["Marks"].fillna(average_marks)
print("\nMissing values in 'Marks' column replaced with average.")
# In[ ]:
# 7. Select rows where Marks > 80
high_scorers = df[df["Marks"] > 80]
print("\nStudents with Marks > 80:")
print(high_scorers)
# In[ ]:
```

PR4: Problem Statement

Descriptive Statistics:

Generate central tendency and dispersion metrics grouped by categorical variables. Analyze the Iris dataset.

Code:

```
#!/usr/bin/env python
# coding: utf-8
# In[6]:
import seaborn as sns
import numpy as np
# In[2]:
df=sns.load_dataset('iris')
# In[3]:
# Grouped summary statistics for 'sepal_length' by 'species'
grouped_stats = df.groupby('species')['sepal_length'].agg(['mean', 'median', 'min', 'max', 'std'])
print("Summary statistics for 'sepal_length' grouped by 'species':\n")
print(grouped_stats)
# In[4]:
# Create a list with sepal_length values for each species
grouped_list = df.groupby('species')['sepal_length'].apply(list)
print("\nList of sepal_length values for each species:\n")
print(grouped_list)
# In[10]:
# 2: Statistical details for each species
for species in df['species'].unique():
    print(f"\nStatistical details for {species}:\n", df[df['species'] == species].describe())
# In[ ]:
```

Data Analytics Practical Compilation

PR5: Problem Statement

Data Analytics I:

Implement Linear Regression on the Boston Housing dataset to predict home prices.

Code:

```
# PR5: Linear Regression on California Housing Data
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
# Load dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['PRICE'] = data.target
# Select feature and target
X = df[['MedInc']]
y = df['PRICE']
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
# Train Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Predict and evaluate
y_pred = model.predict(X_test)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))
# Plot results
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Predicted')
plt.xlabel('Median Income')
plt.ylabel('House Price')
plt.title('Linear Regression: Income vs Price')
plt.legend()
plt.show()
```

PR6: Problem Statement

Data Analytics II:

Apply Logistic Regression on Social_Network_Ads dataset and evaluate model performance using confusion matrix.

Code:

```
# PR6: Logistic Regression on Social Network Ads
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
# Load dataset
df = pd.read_csv('/content/sample_data/Social_Network_Ads.csv')
# Select features and target
X = df[['Age', 'EstimatedSalary']]
y = df['Purchased']
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
# Train model
model = LogisticRegression()
model.fit(X_train, y_train)
# Predict
y_pred = model.predict(X_test)
# Confusion matrix and metrics
cm = confusion_matrix(y_test, y_pred)
TP, FN, FP, TN = cm[1, 1], cm[1, 0], cm[0, 1], cm[0, 0]
print("Confusion Matrix:\n", cm)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Error Rate:", 1 - accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("TP:", TP, "FP:", FP, "TN:", TN, "FN:", FN)
```

Data Analytics Practical Compilation

PR7: Problem Statement

Data Analytics III:

Use Naive Bayes algorithm on the Iris dataset and evaluate using confusion matrix metrics.

Code:

```
#!/usr/bin/env python
# coding: utf-8
# In[1]:
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
# In[2]:
# Load dataset
iris = sns.load_dataset('iris')
# In[3]:
# Encode labels (species → numeric)
iris['species'] = iris['species'].astype('category').cat.codes
# In[4]:
# Split data
X = iris.drop('species', axis=1)
y = iris['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# In[5]:
# Train Naïve Bayes
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
# In[6]:
# Confusion Matrix
labels = model.classes_  # ['setosa', 'versicolor', 'virginica']
cm = confusion_matrix(y_test, y_pred, labels=labels)
print("Confusion Matrix:\n", cm)
# In[7]:
# Simple TP, FP, FN, TN calculation per class
print("\nTP, FP, FN, TN for each class (simplified):")
for i, label in enumerate(labels):
    TP = cm[i, i]
    FP = cm[:, i].sum() - TP
    FN = cm[i, :].sum() - TP
    TN = cm.sum() - (TP + FP + FN)
    print(f"{label}: TP={TP}, FP={FP}, FN={FN}, TN={TN}")
# In[ ]:
# Accuracy & Error Rate
accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
# In[ ]:
# Precision & Recall (average='macro' for multiclass)
precision = precision_score(y_test, y_pred, average='macro')
```

```
recall = recall_score(y_test, y_pred, average='macro')
# In[ ]:
print(f"\nAccuracy: {accuracy:.2f}")
print(f"Error Rate: {error_rate:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
# In[ ]:
```

Data Analytics Practical Compilation

PR8: Problem Statement

Data Visualization I:

Use the Titanic dataset and Seaborn to explore visual patterns. Plot fare distribution using histograms.

Code:

```
# Code not available
```

PR9: Problem Statement

Text Analytics I:

Apply document preprocessing: Tokenization, POS Tagging, stop words removal, Stemming, and Lemmatization.

Code:

```
#!/usr/bin/env python
# coding: utf-8
# In[1]:
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
# In[7]:
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger_eng')
# In[8]:
text = "Text preprocessing is important for NLP applications."
tokens = word_tokenize(text)
print("Tokens:", tokens)
# In[9]:
tags = pos_tag(tokens)
print("POS Tags:", tags)
# In[10]:
filtered = [word for word in tokens if word.lower() not in stopwords.words('english')]
print("Without Stopwords:", filtered)
# In[11]:
stemmer = PorterStemmer()
stems = [stemmer.stem(word) for word in filtered]
print("Stems:", stems)
# In[12]:
lemmatizer = WordNetLemmatizer()
lemmas = [lemmatizer.lemmatize(word) for word in filtered]
print("Lemmas:", lemmas)
# In[ ]:
```

PR10: Problem Statement

Text Analytics II:

Preprocess a document using Stemming and Lemmatization techniques.

Code:

```
#!/usr/bin/env python
# coding: utf-8
# In[1]:
from nltk.stem import PorterStemmer, WordNetLemmatizer
import nltk
nltk.download('wordnet')
# In[2]:
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
# In[3]:
words = ["running", "flies", "denied", "playing", "better"]
stems = [stemmer.stem(word) for word in words]
lemmas = [lemmatizer.lemmatize(word, pos='v') for word in words]
# In[4]:
print("Stems:", stems)
print("Lemmas:", lemmas)
# In[ ]:
```

PR11: Problem Statement

Data Visualization I (Duplicate of PR8):

Explore the Titanic dataset using Seaborn and visualize fare distribution.

Code:

```
#!/usr/bin/env python
# coding: utf-8
# In[1]:
import seaborn as sns
import matplotlib.pyplot as plt
# In[2]:
titanic = sns.load_dataset('titanic')
# In[3]:
sns.histplot(data=titanic, x='fare', kde=True, bins=40, color='skyblue')
plt.title('Fare Distribution')
plt.show()
# In[4]:
# Pattern: Survival by gender
sns.countplot(x='sex', hue='survived', data=titanic)
plt.title('Survival by Gender')
plt.show()
# In[ ]:
```

PR12: Problem Statement

Data Visualization II:

Create a boxplot of age vs gender in the Titanic dataset, segmented by survival, and write observations.

Code:

```
#!/usr/bin/env python
# coding: utf-8
# In[1]:
import seaborn as sns
import matplotlib.pyplot as plt
# In[3]:
df = sns.load_dataset('titanic')
# In[4]:
sns.boxplot(data=df, x='sex', y='age', hue='survived')
plt.title("Age Distribution by Gender and Survival")
plt.show()
# In[ ]:
```

Data Analytics Practical Compilation

PR13: Problem Statement

Data Visualization III:

Load Iris dataset and create histograms, boxplots, and analyze outliers.

Code:

```
#!/usr/bin/env python
# coding: utf-8
# In[5]:
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# In[ ]:
df = sns.load_dataset('iris')
# In[ ]:
print(df.dtypes)
# In[ ]:
df.hist(figsize=(10,8))
plt.suptitle("Histogram of  Iris feature")
plt.show()
# In[4]:
df.plot(kind='box',subplots=True,layout=(2,2),figsize=(10,8))
plt.suptitle("Boxplot of Iris feature")
plt.show()
# In[3]:
# In[ ]:
```