2. Exploratory Data Analysis (EDA): Visualize datasets using Matplotlib and Seaborn, identify trends, outliers, and correlations.

EDA Program: Visualize datasets, identify trends, outliers, correlations

# Import libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Settings

sns.set\_style('whitegrid')

plt.rcParams['figure.figsize'] = (10, 6)

# Load dataset

df = sns.load\_dataset('titanic') # Built-in dataset from seaborn

# 1. Basic Information

print("Dataset Head:")

print(df.head())

print("\nDataset Info:")

print(df.info())

print("\nSummary Statistics:")

print(df.describe())

# 2. Missing Values

print("\nMissing Values:")

print(df.isnull().sum())

# 3. Distribution Plots

# Age distribution

plt.figure()

sns.histplot(df['age'], bins=30, kde=True, color='blue')

plt.title('Age Distribution')

plt.xlabel('Age')

plt.ylabel('Count')

plt.show()

# 4. Categorical Counts

# Class counts

plt.figure()

sns.countplot(x='class', data=df, palette='Set2')

plt.title('Passenger Class Distribution')

plt.xlabel('Passenger Class')

plt.ylabel('Count')

plt.show()

# 5. Boxplot for Outliers

# Age by Class

plt.figure()

sns.boxplot(x='class', y='age', data=df, palette='Set3')

plt.title('Age Distribution by Passenger Class')

plt.show()

# 6. Trends (if time series data is available, here just simulated)

# Survival by Age

plt.figure()

sns.lineplot(x='age', y='survived', data=df)

plt.title('Survival Trend by Age')

plt.xlabel('Age')

plt.ylabel('Survival Rate')

plt.show()

# 7. Correlation Heatmap

# Select only numerical columns for correlation

numerical\_cols = df.select\_dtypes(include=['float64', 'int64']).columns

corr = df[numerical\_cols].corr()

plt.figure()

sns.heatmap(corr, annot=True, cmap='coolwarm', center=0)

plt.title('Correlation Heatmap')

plt.show()

# 8. Scatter Plot (example: Fare vs Age)

plt.figure()

sns.scatterplot(x='age', y='fare', hue='survived', data=df, palette='deep')

plt.title('Fare vs Age (colored by Survival)')

plt.show()

4.Logistic Regression: Classify emails as spam or not using the Spam dataset.

# Import libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.preprocessing import StandardScaler

# 1. Load the dataset

# Example dataset: UCI ML Spam dataset (simulated as CSV or you can download it)

# For now, let's assume you have 'spam.csv' with the last column named 'spam' (1 = spam, 0 = not spam)

df = pd.read\_csv('spam.csv')

# Preview the dataset

print("Dataset Preview:")

print(df.head())

# 2. Prepare features and labels

X = df.drop('spam', axis=1) # Features (drop the target column)

y = df['spam'] # Target

# 3. Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# 4. Feature Scaling (important for Logistic Regression)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# 5. Train Logistic Regression model

model = LogisticRegression()

model.fit(X\_train\_scaled, y\_train)

# 6. Predictions

y\_pred = model.predict(X\_test\_scaled)

# 7. Evaluation

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

print("\nAccuracy Score:")

print(accuracy\_score(y\_test, y\_pred))

5.Decision Trees & Random Forest: Predict loan approval using a banking dataset.

# Import libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

import seaborn as sns

# 1. Load the dataset

# Example: Assuming 'loan\_data.csv' where 'Loan\_Status' is the target (Y=Approved, N=Not Approved)

df = pd.read\_csv('loan\_data.csv')

# Preview the dataset

print("Dataset Preview:")

print(df.head())

# 2. Handle missing values (simple fill or drop)

df = df.dropna() # Or you can use df.fillna(method='ffill') depending on your data

# 3. Encode categorical variables

# Identify categorical columns

cat\_cols = df.select\_dtypes(include='object').columns

# Label Encoding

le = LabelEncoder()

for col in cat\_cols:

df[col] = le.fit\_transform(df[col])

# 4. Prepare features and labels

X = df.drop('Loan\_Status', axis=1) # Features

y = df['Loan\_Status'] # Target (1=Approved, 0=Not Approved after encoding)

# 5. Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# 6. Decision Tree Model

dtree = DecisionTreeClassifier(random\_state=42)

dtree.fit(X\_train, y\_train)

# 7. Random Forest Model

rforest = RandomForestClassifier(n\_estimators=100, random\_state=42)

rforest.fit(X\_train, y\_train)

# 8. Predictions

dtree\_preds = dtree.predict(X\_test)

rforest\_preds = rforest.predict(X\_test)

# 9. Evaluation

print("\n--- Decision Tree Performance ---")

print("Confusion Matrix:\n", confusion\_matrix(y\_test, dtree\_preds))

print("Classification Report:\n", classification\_report(y\_test, dtree\_preds))

print("Accuracy Score:", accuracy\_score(y\_test, dtree\_preds))

print("\n--- Random Forest Performance ---")

print("Confusion Matrix:\n", confusion\_matrix(y\_test, rforest\_preds))

print("Classification Report:\n", classification\_report(y\_test, rforest\_preds))

print("Accuracy Score:", accuracy\_score(y\_test, rforest\_preds))

# 10. Visualization of Decision Tree

plt.figure(figsize=(20,10))

plot\_tree(dtree, feature\_names=X.columns, class\_names=['Not Approved', 'Approved'], filled=True)

plt.title('Decision Tree Visualization')

plt.show()

# 11. Feature Importance (Random Forest)

importances = rforest.feature\_importances\_

features = X.columns

feature\_importance\_df = pd.DataFrame({'Feature': features, 'Importance': importances})

feature\_importance\_df = feature\_importance\_df.sort\_values(by='Importance', ascending=False)

plt.figure(figsize=(12,6))

sns.barplot(x='Importance', y='Feature', data=feature\_importance\_df)

plt.title('Feature Importance from Random Forest')

plt.show()