PRACTICAL 2

Aim-integrate data from multiple sources by merging and transforming datasets using python's pandas library and manipulation techniques

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


# Load datasets from different sources

df_customers = pd.read_csv("E:/Data Mining & Warehousing
Practicals/data_integration/customers.csv")

df_orders = pd.read_excel("E:/Data Mining & Warehousing Practicals/data_integration/orders.xlsx")

df_payments = pd.read_json("E:/Data Mining & Warehousing
Practicals/data_integration/payments.json")


merged_df = df_orders.merge(df_customers, on="CustomerID", how="left")

merged_df = merged_df.merge(df_payments, on="OrderID", how="left")



merged_df.fillna({"PaymentStatus": "Pending", "AmountPaid": 0}, inplace=True)



merged_df["OrderDate"] = pd.to_datetime(merged_df["OrderDate"])


# Creating new feature: Total Amount with Discount

merged_df["FinalAmount"] = merged_df["AmountPaid"] * 0.9



print(merged_df.info())

print(merged_df.head())



plt.figure(figsize=(10, 5))
```

```
merged_df.groupby("PaymentStatus")["FinalAmount"].sum().plot(kind="bar", color=['blue', 'orange'])

plt.xlabel("Payment Status")

plt.ylabel("Total Amount")

plt.title("Total Payment Amount by Status")

plt.xticks(rotation=0)

plt.show()
```

PRACTICAL 3

Aim-apply feature selection techniques like various thresholding and correlation analysis using python's scikit-learn library to reduce dimensionality in a dataset

```
# Apply feature selection techniques like variance thresholding and correlation analysis
# using Python's scikit-learn library to reduce dimensionality in a dataset.

import pandas as pd
import numpy as np
from sklearn.feature_selection import VarianceThreshold
from sklearn.datasets import load_iris
import seaborn as sns
import matplotlib.pyplot as plt

# Load sample dataset (Iris dataset)
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Feature Selection using Variance Thresholding
threshold = 0.1
selector = VarianceThreshold(threshold=threshold)
selected_features = selector.fit_transform(df.iloc[:, :-1])
selected_feature_names = df.iloc[:, :-1].columns[selector.get_support()]

# Create a new DataFrame with selected features
df_selected = pd.DataFrame(selected_features, columns=selected_feature_names)
df_selected['target'] = df['target']

# Feature Selection using Correlation Analysis
correlation_matrix = df_selected.corr()
plt.figure(figsize=(8,6))
```

```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Feature Correlation Matrix')
plt.show()

# Removing highly correlated features
corr_threshold = 0.85
correlated_features = set()
corr_matrix = correlation_matrix.abs()
for i in range(len(corr_matrix.columns)):
    for j in range(i):
        if corr_matrix.iloc[i, j] > corr_threshold:
            colname = corr_matrix.columns[i]
            correlated_features.add(colname)

df_final = df_selected.drop(columns=correlated_features)

# Save the final dataset
df_final.to_csv("selected_features.csv", index=False)
print("Final dataset with selected features saved as 'selected_features.csv'")
```

PRACTICAL 4

Aim - Discretize continuous variables and create concept hierarchies for categorical variables in a market basket dataset using Python's pandas library.

```
# Discretize continuous variables and create concept hierarchies for categorical variables
# in a market basket dataset using Python's pandas library.

import pandas as pd
import numpy as np

data = {
    'Transaction_ID': [1, 2, 3, 4, 5],
    'Total_Amount': [150, 250, 35, 400, 120],
    'Product_Category': ['Electronics', 'Groceries', 'Electronics', 'Clothing', 'Groceries']
}

df = pd.DataFrame(data)

df['Amount_Binned'] = pd.cut(df['Total_Amount'], bins=3, labels=['Low', 'Medium', 'High'])

category_hierarchy = {
    'Electronics': 'Technology',
    'Groceries': 'Daily Needs',
    'Clothing': 'Apparel'
}

df['General_Category'] = df['Product_Category'].map(category_hierarchy)

print(df)
```

PRACTICAL 5

Aim- Implement the Apriori algorithm in Python to mine frequent itemsets from a retail transaction dataset and extract association rules.

```
# Implement the Apriori algorithm in Python to mine frequent itemsets from a retail

# transaction dataset and extract association rules.



import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules

# pip install mlxtend


data = {

   'Transaction': [1, 2, 3, 4, 5],

   'Items': [

      ['Milk', 'Bread', 'Butter'],

      ['Milk', 'Bread'],

      ['Milk', 'Butter'],

      ['Bread', 'Butter'],

      ['Milk', 'Bread', 'Butter', 'Eggs']

   ]

}


df = pd.DataFrame(data)


items = set(item for sublist in df['Items'] for item in sublist)

basket = pd.DataFrame([{item: (item in transaction) for item in items} for transaction in df['Items']])


# Apply Apriori algorithm

frequent_itemsets = apriori(basket, min_support=0.4, use_colnames=True)
```

```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.6)


print("Frequent Itemsets:")
print(frequent_itemsets)
print("\nAssociation Rules:")
print(rules)
```

PRACTICAL 6

Aim- Build a decision tree classifier using Python's scikit-learn library to predict customer churn based on historical data.

```
# Build a decision tree classifier using Python's scikit-learn library to predict customer
# churn based on historical data.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.utils import resample

df = pd.read_csv("customer_churn.csv")

label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['Geography'] = label_encoder.fit_transform(df['Geography'])

drop_columns = ['CustomerID', 'Surname']
df.drop(columns=[col for col in drop_columns if col in df.columns], inplace=True)

class_0 = df[df['Churn'] == 0]
class_1 = df[df['Churn'] == 1]
class_1_upsampled = resample(class_1, replace=True, n_samples=len(class_0), random_state=42)
df_balanced = pd.concat([class_0, class_1_upsampled])

X = df_balanced.drop(columns=['Churn'])
```

```
y = df_balanced['Churn']


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


clf = DecisionTreeClassifier(max_depth=5, min_samples_split=5, min_samples_leaf=2,
random_state=42)

clf.fit(X_train, y_train)


y_pred = clf.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")

print(classification_report(y_test, y_pred, zero_division=1))


plt.figure(figsize=(6,4))

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Churn', 'Churn'],
yticklabels=['Not Churn', 'Churn'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()


plt.figure(figsize=(12,8))

plot_tree(clf, feature_names=X.columns, class_names=['Not Churn', 'Churn'], filled=True,
rounded=True)

plt.title('Decision Tree Structure')

plt.show()
```

PRACTICAL 7

Aim - Implement a Naive Bayes classifier in Python using scikit-learn to classify emails as spam or non-spam based on their content.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

import seaborn as sns


df = pd.read_csv("spam.csv", encoding='latin-1')

df = df[['v1', 'v2']]

df.columns = ['label', 'message']


df['label'] = df['label'].map({'ham': 0, 'spam': 1})


X_train, X_test, y_train, y_test = train_test_split(df['message'], df['label'], test_size=0.2,
random_state=42)


vectorizer = TfidfVectorizer(stop_words='english')

X_train_tfidf = vectorizer.fit_transform(X_train)

X_test_tfidf = vectorizer.transform(X_test)


classifier = MultinomialNB()

classifier.fit(X_train_tfidf, y_train)


y_pred = classifier.predict(X_test_tfidf)


accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
```

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))


conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,4))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham', 'Spam'],
yticklabels=['Ham', 'Spam'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()


df['label'].value_counts().plot(kind='bar', color=['blue', 'red'])

plt.xticks(ticks=[0,1], labels=['Ham', 'Spam'], rotation=0)

plt.xlabel('Message Type')

plt.ylabel('Count')

plt.title('Distribution of Ham and Spam Messages')

plt.show()
```

PRACTICAL 8

Aim -Implement a linear regression method to make predictions based on the sample data set using

Python.

```
# Implement a linear regression method to make predictions based on the sample data set
# using Python.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

data = {
    'Feature': np.random.rand(100) * 100,
    'Target': np.random.rand(100) * 200
}
df = pd.DataFrame(data)

df.to_csv("linear_regression_dataset.csv", index=False)
print("Dataset saved as 'linear_regression_dataset.csv'")

df = pd.read_csv("linear_regression_dataset.csv")
X = df[['Feature']]
y = df['Target']

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)


mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared Score: {r2}")


plt.scatter(X_test, y_test, color='blue', label='Actual Data')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Regression Line')
plt.xlabel("Feature")
plt.ylabel("Target")
plt.title("Linear Regression Model")
plt.legend()
plt.show()
```

PRACTICAL 9

Aim -Implement a logistic regression method to make predictions based on the sample data set using

Python.

```
# Implement a logistic regression method to make predictions based on the sample data
# set using Python

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

df = pd.read_csv("sample_logistic_data.csv")

X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

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

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

model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Model Accuracy:", accuracy)

print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred))


plt.figure(figsize=(6, 4))

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues",
xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix Heatmap")

plt.show()


plt.figure(figsize=(6, 4))

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))

Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.3, cmap="coolwarm")

plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors="k", cmap="coolwarm")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.title("Decision Boundary of Logistic Regression")

plt.show()


plt.figure(figsize=(6, 4))

sns.histplot(df, x="Feature1", hue="Target", kde=True, bins=20, palette="coolwarm")

plt.title("Feature Distribution of Class Labels")

plt.show()
```

PRACTICAL 10

Aim -Implement K-means clustering algorithm in Python using scikit-learn to group customers based on their purchasing behavior.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler


df = pd.read_csv("customer_segmentation.csv")


print(df.head())


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


X = df[['Annual Income (k$)', 'Spending Score (1-100)']]


scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


wcss = []

for i in range(1, 11):

    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)

    kmeans.fit(X_scaled)

    wcss.append(kmeans.inertia_)


plt.figure(figsize=(8, 5))

plt.plot(range(1, 11), wcss, marker='o', linestyle='--', color='b')

plt.xlabel('Number of Clusters')

plt.ylabel('WCSS')
```

```
plt.title('Elbow Method for Optimal K')
plt.show()

kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

print("Cluster Centers:\n", kmeans.cluster_centers_)

plt.figure(figsize=(10, 6))
sns.scatterplot(x=df['Annual Income (k$)'], y=df['Spending Score (1-100)'],
                hue=df['Cluster'], palette='viridis', s=100, edgecolors='black')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=300, c='red', label='Centroids', marker='X')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.title('Customer Segments')
plt.legend()
plt.show()
```