Practical: 4

Aim: Design and develop at least 5 problem statements which demonstrate the use of Control Structures of any data analytics tool.

Source Code:

#Problem 1: Data Filtering with Conditional Statements

```
import pandas as pd
import numpy as np
data = {'Age': [25, 30, 35, 40, 45, 50, 55, 60],
        'Income': [50000, 75000, 60000, 90000, 120000, 80000, 95000, 110000],
        'Category': ['A', 'B', 'A', 'C', 'B', 'A', 'C', 'B']}
df = pd.DataFrame(data)
high_income = []
medium_income = []
low_income = []
for index, row in df.iterrows():
    if row['Income'] > 100000:
        high_income.append(row['Income'])
    elif row['Income'] > 70000:
        medium_income.append(row['Income'])
    else:
        low_income.append(row['Income'])
print("High Income:", high_income)
print("Medium Income:", medium_income)
print("Low Income:", low_income)
```

Source Code:

#Problem 2: Loop-based Data Transformation

```
sales_data = [1500, 2300, 1800, 2100, 1900, 2500, 3000, 1700]
bonus_list = []
for sale in sales_data:
    if sale > 2500:
```

```
        bonus = sale * 0.15
    elif sale > 2000:
        bonus = sale * 0.10
    else:
        bonus = sale * 0.05
    bonus_list.append(bonus)
print("Sales Bonuses:", bonus_list)
```

Source Code:

#Problem 3: Nested Control Structures for Data Analysis

```
student_scores = {'Rohit': {'Data Science': 85, 'Web-Tech': 92, 'Cloud-Computing': 78},
                  'Sameer': {'Data Science': 72, 'Web-Tech': 88, 'Cloud-Computing': 91},
                  'Rutuja': {'Data Science': 95, 'Web-Tech': 89, 'Cloud-Computing': 84}}
for student, subjects in student_scores.items():
    total = 0
    count = 0
    for subject, score in subjects.items():
        total += score
        count += 1
        if score >= 90:
            print(f"{student} scored A in {subject}")
        elif score >= 80:
            print(f"{student} scored B in {subject}")
        else:
            print(f"{student} scored C in {subject}")
    average = total / count
    print(f"{student}'s average score: {average:.2f}")
```

Source Code:

#Problem 4: While Loop for Data Processing

```
data_stream = [45, 67, 23, 89, 12, 56, 78, 34, 90, 21]
processed_data = []
```

```
index = 0
while index < len(data_stream):
    current_value = data_stream[index]
    if current_value > 50:
        processed_data.append(current_value * 1.1)
    else:
        processed_data.append(current_value * 0.9)
    if current_value == 90:
        break
    index += 1
print("Processed Data:", processed_data)
```

Source Code:

#Problem 5: Complex Conditional Data Categorization

```
customer_data = [{'purchases': 12, 'amount': 1500, 'rating': 4.5},
                 {'purchases': 8, 'amount': 800, 'rating': 3.2},
                 {'purchases': 15, 'amount': 2500, 'rating': 4.8},
                 {'purchases': 5, 'amount': 400, 'rating': 2.9},
                 {'purchases': 20, 'amount': 3000, 'rating': 4.9}]
customer_categories = {'VIP': [], 'Regular': [], 'New': []}
for customer in customer_data:
    if (customer['purchases'] > 10 and customer['amount'] > 1000 and customer['rating'] > 4.0):
        customer_categories['VIP'].append(customer)
    elif (customer['purchases'] > 5 or customer['amount'] > 500):
        customer_categories['Regular'].append(customer)
    else:
        customer_categories['New'].append(customer)
for category, customers in customer_categories.items():
    print(f"{category} Customers: {len(customers)}")
```

Practical: 5

Aim: Implement KNN Classification techniques using any data analytics tool.

Source Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)
y_pred = knn.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
k_values = range(1, 21)
accuracies = []
for k in k_values:
    knn_temp = KNeighborsClassifier(n_neighbors=k)
```

```
    knn_temp.fit(X_train_scaled, y_train)
    y_pred_temp = knn_temp.predict(X_test_scaled)
    accuracies.append(accuracy_score(y_test, y_pred_temp))
best_k = k_values[np.argmax(accuracies)]
best_accuracy = max(accuracies)
print(f"Best K value: {best_k} with accuracy: {best_accuracy:.4f}")
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracies, marker='o', linestyle='-', color='b')
plt.title('K Value vs Accuracy')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```

Source Code:

#Alternative with custom KNN implementation:

```
import numpy as np
from collections import Counter
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

class CustomKNN:
    def __init__(self, k=5):
        self.k = k
    def fit(self, X, y):
        self.X_train = X
        self.y_train = y
    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)
```

```
    def _predict(self, x):
        distances = [np.sqrt(np.sum((x - x_train)**2)) for x_train in self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
custom_knn = CustomKNN(k=5)
custom_knn.fit(X_train_scaled, y_train)
y_pred_custom = custom_knn.predict(X_test_scaled)
accuracy_custom = accuracy_score(y_test, y_pred_custom)
print(f"Custom KNN Accuracy: {accuracy_custom:.4f}")
```