Course Title	**–**	**Practical V: Machine Learning Lab**
Course Code	**–**	**20YP5**
Semester	**–**	**III**

Exercises

1) Implement a classifier for the sales data.
2) Develop a predictive model for predicting house prices.
3) Implement the FIND-S algorithm. Verify that it successfully produces the trace in for the Enjoy Sport example. (Tom Mitchell Reference).
4) Implement a decision tree algorithm for sales prediction/classification in retail sector.
5) Implement K-nearest neighbor algorithm to classify iris dataset.
6) Implement backpropagation algorithm for stock prices prediction.
7) Implement clustering algorithm for Insurance fraud detection.
8) Implement clustering algorithm for identifying cancerous data.
9) Apply reinforcement learning and develop a game of your own.
10) Develop a traffic signal control system using reinforcement learning technique.

Ex no: 01 Implement a classifier for the sales data.

Aim:

To implement a classifier for the sales data.

Dataset: Churn-Modelling.csv

Training and Testing Dataset Size: 65% (Training) and 35% (Testing)

Algorithm used: Random Forest

Dataset Description:

No. of attributes: 14
No. of instances: 10000
No. of features: 10, No. of target: 1

Packages:

Pandas

sklearn.preprocessing – LabelEncoder

sklearn.preprocessing – StandardScaler

sklearn.model_selection – train_test_split

sklearn.ensemble – RandomForestClassifier

sklearn.metrics – accuracy_score, confusion_matrix, f1_score

Algorithm:

Step 1: Start.

Step 2: Import the necessary libraries.

Step 3: Load the dataset as a pandas dataframe.

Step 4: Select the features and the target.

Step 5: Encode the categorical data using LabelEncoder.

Step 6: Scale the features using StandardScaler.

Step 7: Split the training and testing data and display their shapes.

Step 8: Fit the training data using the RandomForestClassifier.

Step 9: Use the trained classifier to make predictions on unseen data.

Step 10: Evaluate and display the classification metrics accuracy score, confusion matrix, and f1-score.

Step 11: End.

Sample Input:

CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
619	France	Female	42	2	0	1	1
608	Spain	Female	41	1	83807.86	1	0

IsActiveMember	EstimatedSalary	Exited
1	101348.9	1
1	112542.6	0

Expected Output:

Model Accuracy: 0.84

Confusion Matrix:

		Predicted 1	Predicted 0
Actual	1	2700	90
	0	360	350

F1 Score: 0.83

Program:

```
# 1. IMPORTING LIBRARIES AND LOADING DATASET
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score

print("1. LOADING THE DATASET".center(60), "\n")
df = pd.read_csv("Datasets\\P1-Churn-Modelling.csv")

print("Example Records From The Dataset")
print(df.head(5))
print("\n")
# =======================================================================

# 2. DATA PREPROCESSING
print("2. DATA PREPROCESSING".center(60), "\n")

print("Selecting the Label and Features")
print("Chosen Label - 'Exited'")
y = df.iloc[:, 13]
print(y.head(5), "\n")

print("Chosen Features - 'CreditScore' to 'EstimatedSalary'")
X = df.iloc[:, 3:13]
print(X.head(5), "\n")

label = LabelEncoder()
X['Gender'] = label.fit_transform(X['Gender'])
print("Encoding Gender Feature")
print(X['Gender'].head(5), "\n")

X['Geography'] = label.fit_transform(X['Geography'])
print("Encoding Geography Feature")
print(X['Geography'].head(5), "\n")

print("Splitting the Training and Testing Data")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.35, random_state=42)
```

```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

print('Shape of Train Set:', X_train.shape, y_train.shape)
print('Shape of Test Set:', X_test.shape, y_test.shape)
print("\n")

# ==========================================================================

# 3. BUILDING AND IMPLEMENTING A RANDOM FOREST CLASSIFIER
print("3. BUILDING AND IMPLEMENTING A RANDOM FOREST CLASSIFIER".center(60), "\n")

classifier = RandomForestClassifier(n_estimators=100)  # warning 10 to 100
classifier.fit(X_train, y_train)
prediction = classifier.predict(X_test)
df_prediction = pd.DataFrame(prediction)

print("Churn Prediction")
print(df_prediction.head(5))
print("\n")

# ==========================================================================

# 4. CLASSIFICATION METRICS
print("4. CLASSIFICATION METRICS".center(60), "\n")

accuracy = accuracy_score(y_test, df_prediction)
print("Model Accuracy: ", accuracy)

cm = confusion_matrix(y_test, prediction)
print("Confusion Matrix:\n", cm)

f1 = f1_score(y_test, prediction, average='weighted')
print("F1 Score: ", f1)
```

Output:

```
              1. LOADING THE DATASET

Example Records From The Dataset
   RowNumber  CustomerId   Surname  ...  IsActiveMember EstimatedSalary Exited
0          1    15634602  Hargrave  ...               1       101348.88      1
1          2    15647311      Hill  ...               1       112542.58      0
2          3    15619304      Onio  ...               0       113931.57      1
3          4    15701354      Boni  ...               0        93826.63      0
4          5    15737888  Mitchell  ...               1        79084.10      0

[5 rows x 14 columns]
```

2. DATA PREPROCESSING

```
Selecting the Label and Features
Chosen Label - 'Exited'
0    1
1    0
2    1
3    0
4    0
Name: Exited, dtype: int64

Chosen Features - 'CreditScore' to 'EstimatedSalary'
   CreditScore Geography  Gender  ...  HasCrCard  IsActiveMember  EstimatedSalary
0          619    France  Female  ...          1               1        101348.88
1          608     Spain  Female  ...          0               1        112542.58
2          502    France  Female  ...          1               0        113931.57
3          699    France  Female  ...          0               0         93826.63
4          850     Spain  Female  ...          1               1         79084.10

[5 rows x 10 columns]

Encoding Gender Feature
0    0
1    0
2    0
3    0
4    0
Name: Gender, dtype: int32

Encoding Geography Feature
0    0
1    2
2    0
3    0
4    2
Name: Geography, dtype: int32

Splitting the Training and Testing Data
Shape of Train Set: (6500, 10) (6500,)
Shape of Test Set: (3500, 10) (3500,)
```

3. BUILDING AND IMPLEMENTING A RANDOM FOREST CLASSIFIER

```
Churn Prediction
   0
0  0
1  0
2  0
3  0
4  0
```

4. CLASSIFICATION METRICS

```
Model Accuracy:  0.8657142857142858
Confusion Matrix:
 [[2705   98]
 [ 372  325]]
F1 Score:  0.8524170310981535
```

Ex no: 02 Develop a predictive model for predicting house prices.

Aim:

To develop a prediction model for predicting house prices.

Dataset: USA-Housing.csv

Training and Testing Dataset Size: 80% (Training) and 20% (Testing)

Algorithm used: Linear Regression

Dataset Description:

No. of attributes: 18
No. of instances: 4600
No. of features: 8, No. of target: 1

Packages:

NumPy

Pandas

Matplotlib

sklearn.model_selection – train_test_split

sklearn.preprocessing – StandardScaler

sklearn.linear_model – LinearRegression

sklearn.metrics – mean_squared_error, mean_absolute_error, r2_score

Algorithm:

Step 1: Start.

Step 2: Import the necessary libraries.

Step 3: Load the dataset as a pandas dataframe.

Step 4: Check for missing values.

Step 5: Select the features and the target.

Step 6: Split the training and testing data and display their shapes.

Step 7: Use the StandardScaler to scale the features.

Step 8: Use the LinearRegression model to fit the training data.

Step 9: Use the trained model to make predictions on unseen data.

Step 10: Evaluate the model's performance using mean squared error, mean absolute error, and r-squared value.

Step 11: Visualize the predictions using scatter plot.

Step 12: End.

Sample Input:

price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition
313000	3	1.5	1340	7912	1.5	0	0	3
2384000	5	2.5	3650	9050	2	0	4	5

Expected Output:

Predicted Price: 330359000
Mean Squared Error: 986869410000
Mean Absolute Error: 215800
R-squared: 0.032

Program:

```
# 1. IMPORTING LIBRARIES AND LOADING DATASET
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

print()
print("1. LOADING THE DATASET".center(60), "\n")
df = pd.read_csv("Datasets\\P2-USA-Housing.csv")

print("Example Records From The Dataset")
print(df.head(5))
print("\n")

# ============================================================================


# 2. DATA PREPROCESSING
print("2. DATA PREPROCESSING".center(60), "\n")

# Check for missing values
print("Checking Missing Values:")
print(df.isnull().sum())
print()

print("Selecting the Label and Features")
print("Chosen Label - 'price'")
y = df.iloc[:, 1]
print(y.head(5), "\n")

print("Chosen Features - 'bedrooms' to 'condition'")
X = df.iloc[:, 2:10]
print(X.head(5), "\n")

print("Splitting the Training and Testing Data")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

print('Shape of Train Set:', X_train.shape, y_train.shape)
print('Shape of Test Set:', X_test.shape, y_test.shape)
print("\n")

# ============================================================================
```

```
# 3. BUILDING AND EVALUATING LINEAR REGRESSION MODEL
print("3. BUILDING AND EVALUATING LINEAR REGRESSION MODEL".center(60), "\n")

# Model Building
model = LinearRegression()
model.fit(X_train, y_train)

# Model Evaluation
y_pred = model.predict(X_test)  # prediction
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("R-squared:", r2)
print("\n")

# ======================================================================

# 4. PREDICTIONS AND VISUALIZATION
print("4. PREDICTIONS AND VISUALIZATION".center(60), "\n")

# Scatter plot to visualize the predictions against actual prices
plt.scatter(y_test, y_pred)
m, b = np.polyfit(y_pred, y_test, 1)
plt.plot(y_pred, m*y_pred + b, color='black')
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs. Predicted Prices")
plt.show()

# Residual plot to check the model's performance
residuals = y_test - y_pred
plt.scatter(y_test, residuals)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel("Actual Prices")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()

# Using the trained model to make predictions on new data.
new_data = [[3, 2, 1500, 4000, 1, 0, 0, 3]]
predicted_price = model.predict(new_data)
print("Predicted Price:", predicted_price[0])
```

Output:

```
                  1. LOADING THE DATASET

Example Records From The Dataset
                  date      price  bedrooms  ...       city  statezip country
0  2014-05-02 00:00:00   313000.0       3.0  ...  Shoreline  WA 98133     USA
1  2014-05-02 00:00:00  2384000.0       5.0  ...    Seattle  WA 98119     USA
2  2014-05-02 00:00:00   342000.0       3.0  ...       Kent  WA 98042     USA
3  2014-05-02 00:00:00   420000.0       3.0  ...   Bellevue  WA 98008     USA
4  2014-05-02 00:00:00   550000.0       4.0  ...    Redmond  WA 98052     USA

[5 rows x 18 columns]
```

2. DATA PREPROCESSING

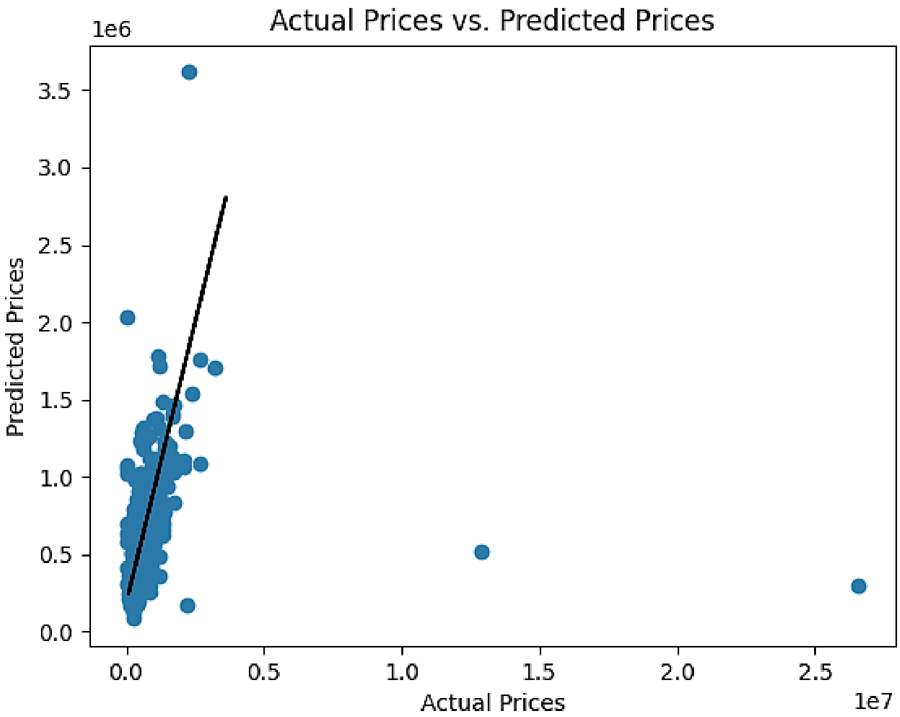
```
Checking Missing Values:
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
dtype: int64

Selecting the Label and Features
Chosen Label - 'price'
0     313000.0
1    2384000.0
2     342000.0
3     420000.0
4     550000.0
Name: price, dtype: float64

Chosen Features - 'bedrooms' to 'condition'
   bedrooms  bathrooms  sqft_living  ...  waterfront  view  condition
0       3.0       1.50         1340  ...           0     0          3
1       5.0       2.50         3650  ...           0     4          5
2       3.0       2.00         1930  ...           0     0          4
3       3.0       2.25         2000  ...           0     0          4
4       4.0       2.50         1940  ...           0     0          4

[5 rows x 8 columns]

Splitting the Training and Testing Data
Shape of Train Set: (3680, 8) (3680,)
Shape of Test Set: (920, 8) (920,)
```

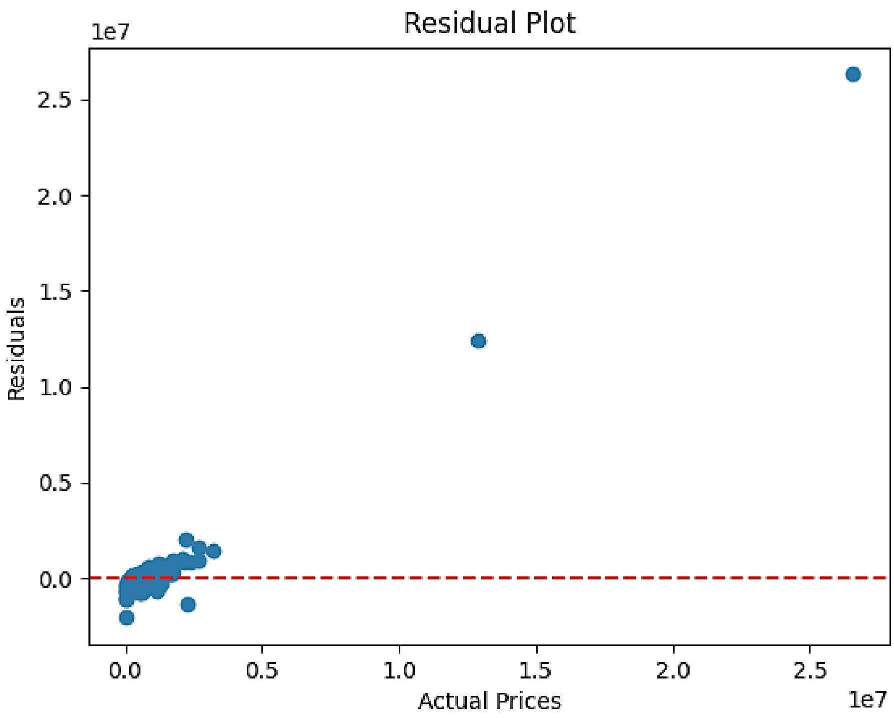
3. BUILDING AND EVALUATING LINEAR REGRESSION MODEL

```
Mean Squared Error: 986869414953.9763
Mean Absolute Error: 215838.55739962033
R-squared: 0.03233518995632867
```

4. PREDICTIONS AND VISUALIZATION

```
Predicted Price: 330359049.5233514
```


Actual Prices vs. Predicted Prices
1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Predicted Prices
0.0
0.5
1.0
1.5
2.0
2.5
Actual Prices
1e7


Residual Plot
1e7
2.5
2.0
1.5
1.0
0.5
0.0
Residuals
0.0
0.5
1.0
1.5
2.0
2.5
Actual Prices
1e7

Ex no: 03 Implement the FIND-S algorithm. Verify that it successfully produces the trace in for the Enjoy Sport example. (Tom Mitchell Reference).

Aim:

To implement the Find-S algorithm and to verify that it successfully produces the trace in for the enjoy sport example.

Dataset: EnjoySports.csv

Algorithm used: Find-S

Dataset Description:

No. of attributes: 7
No. of instances: 4

Algorithm:

Step 1: Start.

Step 2: Initialize h with the most specific hypothesis in H, h = {∅, ∅, ∅, ∅, ∅, ∅}.

Step 3: For each positive instance x, repeat steps 4 to 6.

Step 4: For each attribute constraint a_i in h, repeat steps 5 and 6.

Step 5: If the constraint a_i is satisfied by x, then do nothing.

Step 6: Else replace a_i in h by the next more general constraint that is satisfied by x.

Step 7: Output hypothesis h.

Step 8: Stop.

Pseudocode:

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
    - For each attribute constraint a_i in h

        If the constraint a_i is satisfied by x
        Then do nothing
        Else replace a, in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Sample Input:

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Expected Output:

The Maximally Specific Hypothesis for the given Training Examples:
['Sunny', 'Warm', '?', 'Strong', '?', '?']

Program:

```
import csv

attributes = [['Sunny', 'Rainy'],
          ['Warm', 'Cold'],
          ['Normal', 'High'],
          ['Strong', 'Weak'],
          ['Warm', 'Cool'],
          ['Same', 'Change']]
num_attributes = len(attributes)

print("\nThe Most General Hypothesis: ['?', '?', '?', '?', '?', '?']")
print("The Most Specific Hypothesis: ['0', '0', '0', '0', '0', '0']")

a = []

print("\nThe Given Training Dataset \n")
with open("Datasets\\P3-Enjoy-Sport.csv") as csvFile:
  reader = csv.reader(csvFile)
  for row in reader:
    a.append(row)
    print(row)
del a[0]

print("\nThe initial value of hypothesis: ")
hypothesis = ['0'] * num_attributes
print(hypothesis)

print("\nFind S: Finding a Maximally Specific Hypothesis\n")
for i in range(0, len(a)):
  if a[i][num_attributes] == 'Yes':
    for j in range(0, num_attributes):
      if a[i][j] != hypothesis[j] and hypothesis[j] == '0':
        hypothesis[j] = a[i][j]
      elif a[i][j] != hypothesis[j] and hypothesis[j] != '0':
        hypothesis[j] = '?'
  print(f"For Training Example {i}, the hypothesis is {hypothesis}")

print("\nThe Maximally Specific Hypothesis for the given Training Examples:")
print(hypothesis)
```

Output:

```
The Most General Hypothesis: ['?', '?', '?', '?', '?', '?']
The Most Specific Hypothesis: ['0', '0', '0', '0', '0', '0']

The Given Training Dataset

['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water', 'Forcast', 'EnjoySport']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']
```

```
Find S: Finding a Maximally Specific Hypothesis

For Training Example 0, the hypothesis is ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm',
'Same']
For Training Example 1, the hypothesis is ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For Training Example 2, the hypothesis is ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For Training Example 3, the hypothesis is ['Sunny', 'Warm', '?', 'Strong', '?', '?']

The Maximally Specific Hypothesis for the given Training Examples:
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Ex no: 04 Implement a decision tree algorithm for sales prediction / classification in retail sector

Aim:

To implement the decision tree algorithm for sales prediction.

Dataset: Churn-RawData.csv

Training and Testing Dataset Size: 90% (Training) and 10% (Testing)

Algorithm used: Decision Tree

Dataset Description:

No. of attributes: 14
No. of instances: 10000
No. of features: 10, No. of target: 1

Packages:

NumPy, Pandas, Graphviz, Seaborn

Matplotlib.pyplot

sklearn.preprocessing – MinMaxScaler

sklearn.model_selection – train_test_split

sklearn.tree – DecisionTreeClassifier

sklearn.metrics – accuracy_score, confusion_matrix, f1_score

Algorithm:

Step 1: Start.

Step 2: Import the necessary libraries.

Step 3: Load the dataset as a pandas dataframe.

Step 4: Check for missing values.

Step 5: Limit the data by keeping only the columns necessary and excluding others from the dataframe.

Step 6: Convert the categorical variables into numeric representation.

Step 7: Scale the columns data using MinMaxScaler.

Step 8: Select the features and the target.

Step 9: Split the training and testing data and display their shapes.

Step 10: Use the DecisionTreeClassifier model to fit the training data.

Step 11: Visualize the decision tree using the Graphviz module.

Step 12: Use the trained model to make predictions.

Step 13: Evaluate and display the performance metrics accuracy score, confusion matrix, and f1 score.

Step 14: End.

Pseudocode:

```
GenerateTree(𝒳)
    If NodeEntropy(𝒳)< θ_I /* eq. 9.3
      Create leaf labelled by majority class in 𝒳
      Return
    i ← SplitAttribute(𝒳)
    For each branch of x_i
      Find 𝒳_i falling in branch
      GenerateTree(𝒳_i)
SplitAttribute(𝒳)
    MinEnt← MAX
    For all attributes i = 1,...,d
        If x_i is discrete with n values
          Split 𝒳 into 𝒳_1,...,𝒳_n by x_i
          e ← SplitEntropy(𝒳_1,...,𝒳_n) /* eq. 9.8 */
          If e<MinEnt MinEnt ← e; bestf ← i
        Else /* x_i is numeric */
          For all possible splits
              Split 𝒳 into 𝒳_1, 𝒳_2 on x_i
              e←SplitEntropy(𝒳_1, 𝒳_2)
              If e<MinEnt MinEnt ← e; bestf ← i
    Return bestf
```

Classification tree construction

Sample Input:

CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
619	France	Female	42	2	0	1	1
608	Spain	Female	41	1	83807.86	1	0

IsActiveMember	EstimatedSalary	Exited
1	101348.9	1
1	112542.6	0

Expected Output:

Accuracy: 0.84
F1 Score: 0.80
Confusion Matrix:
[[0.99 0.011]
[0.72 0.28]]

Program:

```
# 1. IMPORTING LIBRARIES AND LOADING DATASET
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import graphviz
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score

print()
print("1. LOADING THE DATASET".center(60), "\n")
df = pd.read_csv('Datasets\\P4-Churn-RawData.csv', encoding='latin-1')

print("Example Records From The Dataset")
print(df.head(5))
print("\n")

# ======================================================================

# 2. DATA PREPROCESSING
print("2. DATA PREPROCESSING".center(60), "\n")

print("CHECKING MISSING VALUES:")
print(df.isnull().sum())
print()

print("LIMITING THE DATA")
limited_data = df.iloc[:, 3:]
print("Columns:")
print(limited_data.columns)
print("\nExample Records:")
print(limited_data.head(5), "\n")

print("CONVERTING CATEGORICAL VARIABLES INTO NUMERIC REPRESENTATION")
processed_data = pd.get_dummies(limited_data, columns=['Geography', 'Gender', 'HasCrCard',
                                'IsActiveMember'], dtype=int)
print("Example Records:")
print(processed_data.head(5), "\n")

print("SCALING THE COLUMNS")
scale_vars = ['CreditScore', 'EstimatedSalary', 'Balance', 'Age']
scaler = MinMaxScaler()
processed_data[scale_vars] = scaler.fit_transform(processed_data[scale_vars])
print("Example Records:")
print(processed_data.head(5), "\n")

print("SELECTING FEATURES AND LABELS")
X = processed_data.drop('Exited', axis=1).values  # Input features (attributes)
y = processed_data['Exited'].values  # Target vector
print('X shape: {}'.format(np.shape(X)))
print('y shape: {}'.format(np.shape(y)))
print()
```

```
print("SPLITTING THE TRAINING AND TESTING DATA")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=0)
print('Shape of Train Set:', X_train.shape, y_train.shape)
print('Shape of Test Set:', X_test.shape, y_test.shape)
print("\n")

# =========================================================================

# 3. BUILDING AND IMPLEMENTING THE DECISION TREE CLASSIFIER
print("3. BUILDING AND IMPLEMENTING A DECISION TREE CLASSIFIER".center(60), "\n")

dt = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=1)
dt.fit(X_train, y_train)

dot_data = tree.export_graphviz(dt, out_file=None,
                    feature_names=processed_data.drop('Exited', axis=1).columns,
                    class_names=processed_data['Exited'].unique().astype(str),
                    filled=True, rounded=True,
                    special_characters=True)
graph = graphviz.Source(dot_data)
graph.view()

prediction = dt.predict(X_test)
df_prediction = pd.DataFrame(prediction)

print("Churn Prediction")
print(df_prediction.head(5))
print("\n")

print("Accuracy: ", accuracy_score(y_test, df_prediction))
# f1 = f1_score(y_test, prediction, average='weighted')
print("F1 Score: ", f1_score(y_test, prediction, average='weighted'))

cm = confusion_matrix(y_test, prediction)
cm_norm = cm / cm.sum(axis=1)[:, np.newaxis]
plt.figure()

if dt.classes_ is not None:
  sns.heatmap(cm_norm, xticklabels=dt.classes_, yticklabels=dt.classes_, vmin=0., vmax=1., annot=True,
        annot_kws={'size': 50})
else:
  sns.heatmap(cm_norm, vmin=0., vmax=1.)
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

Output:

```
                    1. LOADING THE DATASET

Example Records From The Dataset
   RowNumber  CustomerId   Surname  ...  IsActiveMember EstimatedSalary Exited
0          1    15634602  Hargrave  ...               1       101348.88      1
1          2    15647311      Hill  ...               1       112542.58      0
2          3    15619304      Onio  ...               0       113931.57      1
3          4    15701354      Boni  ...               0        93826.63      0
4          5    15737888  Mitchell  ...               1        79084.10      0

[5 rows x 14 columns]
```

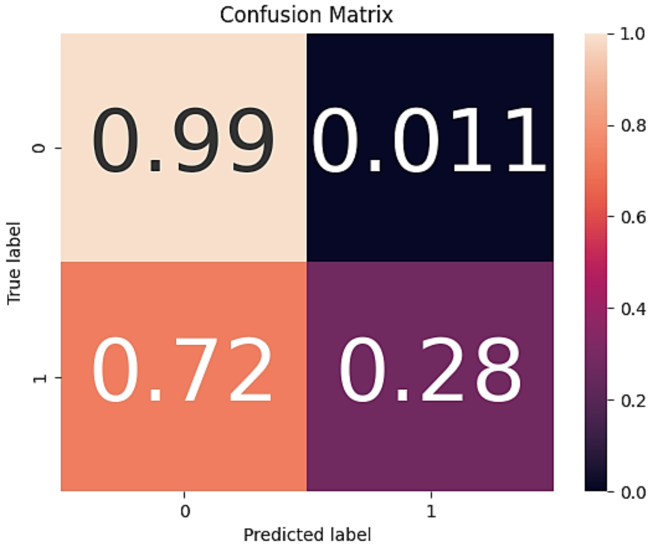
2. DATA PREPROCESSING

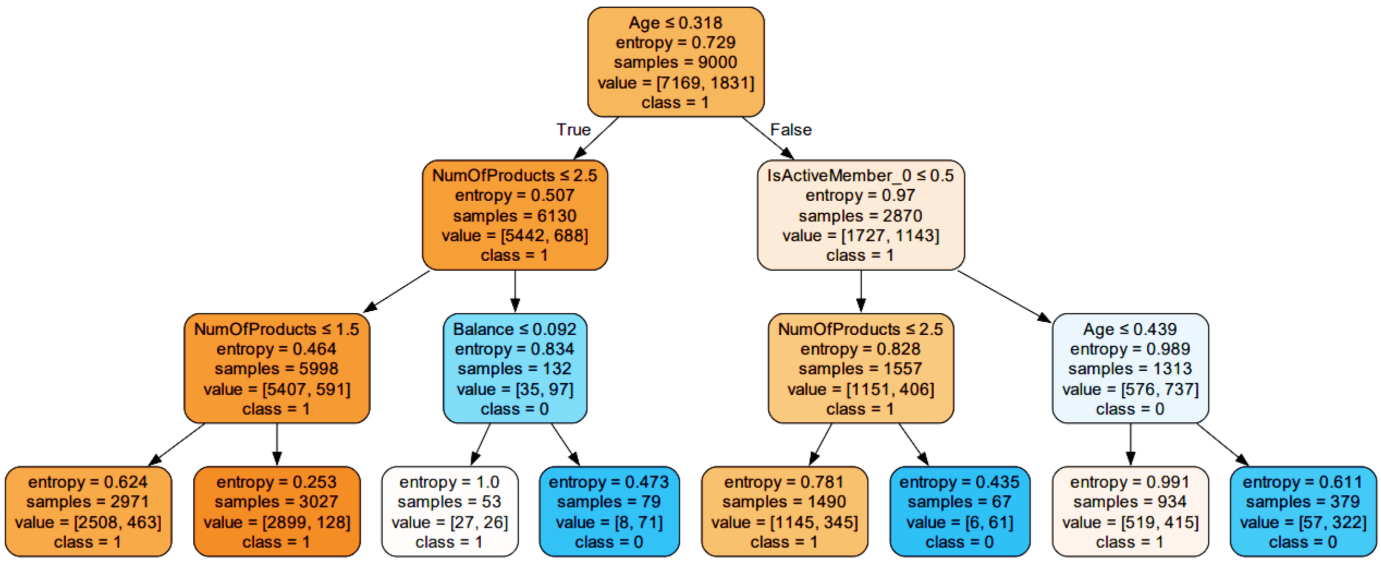
```
CHECKING MISSING VALUES:
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64

LIMITING THE DATA
Columns:
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')

Example Records:
   CreditScore Geography  Gender  ...  IsActiveMember  EstimatedSalary  Exited
0          619    France  Female  ...               1        101348.88       1
1          608     Spain  Female  ...               1        112542.58       0
2          502    France  Female  ...               0        113931.57       1
3          699    France  Female  ...               0         93826.63       0
4          850     Spain  Female  ...               1         79084.10       0
[5 rows x 11 columns]


CONVERTING CATEGORICAL VARIABLES INTO NUMERIC REPRESENTATION
Example Records:
   CreditScore  Age  Tenure  ...  HasCrCard_1  IsActiveMember_0  IsActiveMember_1
0          619   42       2  ...            1                 0                 1
1          608   41       1  ...            0                 0                 1
2          502   42       8  ...            1                 1                 0
3          699   39       1  ...            0                 1                 0
4          850   43       2  ...            1                 0                 1

[5 rows x 16 columns]

SCALING THE COLUMNS
Example Records:
   CreditScore       Age  ...  IsActiveMember_0  IsActiveMember_1
0        0.538  0.324324  ...                 0                 1
1        0.516  0.310811  ...                 0                 1
2        0.304  0.324324  ...                 1                 0
3        0.698  0.283784  ...                 1                 0
4        1.000  0.337838  ...                 0                 1

[5 rows x 16 columns]

SELECTING FEATURES AND LABELS
X shape: (10000, 15)
y shape: (10000,)
```

```
SPLITTING THE TRAINING AND TESTING DATA
Shape of Train Set: (9000, 15) (9000,)
Shape of Test Set: (1000, 15) (1000,)
```

3. BUILDING AND IMPLEMENTING A DECISION TREE CLASSIFIER

```
Churn Prediction
   0
0  0
1  0
2  0
3  0
4  0

Accuracy:  0.842
F1 Score:  0.8077386982570807
```





Ex no: 05 Implement K-nearest neighbor algorithm to classify iris dataset.

Aim:

To implement the K-Nearest Neighbour algorithm to classify iris flowers.

Dataset: Iris.csv

Training and Testing Dataset Size: 80% (Training) and 20% (Testing)

Algorithm used: K-Nearest Neighbour (KNN algorithm)

Dataset Description:

No. of attributes: 5
No. of instances: 150
No. of features: 4, No. of target: 1

Packages:

Pandas

sklearn.model_selection – train_test_split

sklearn.preprocessing – StandardScaler

sklearn.neighbors – KNeighborsClassifier

sklearn.metrics – confusion_matrix, classification_report

Algorithm:

Step 1: Start.

Step 2: Import the necessary libraries.

Step 3: Load the dataset as a pandas dataframe.

Step 4: Select the features and the target.

Step 5: Split the training and testing data and display their shapes.

Step 6: Scale the features using StandardScaler.

Step 7: Use the KNeighboursClassifier model to fit the training data.

Step 8: Use the trained model to make predictions.

Step 9: Display the confusion matrix, and the classification report.

Step 10: End.

Sample Input:

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa

Expected Output:

Accuracy: 0.90
Precision: 0.92
Recall: 0.83

```
Confusion Matrix:
[[13  0  0]
 [ 0 10  0]
 [ 0  2  5]]
```

Program:

```
# IMPORTING THE LIBRARIES
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report

print("1. LOADING THE DATASET AND PREPROCESSING".center(60), "\n")

# LOADING THE DATASET
dataset = pd.read_csv("Datasets\\P5-Iris.csv")
print("Example Records From The Dataset")
print(dataset.head(5))
print()

# DATA PREPROCESSING
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

# ==========================================================================

print("2. PREDICTIONS".center(60), "\n")

predictions = classifier.predict(X_test)

for i in range(0, 5):
  print(f"{X_test[i]} - {predictions[i]}")
print()

# ==========================================================================

print("3. EVALUATION".center(60), "\n")

print("Confusion Matrix:")
print(confusion_matrix(y_test, predictions))
print("\nClassification Report:")
print(classification_report(y_test, predictions))
```

Output:

```
          1. LOADING THE DATASET AND PREPROCESSING

Example Records From The Dataset
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa

                       2. PREDICTIONS

[ 0.94558216 -0.05390188  0.75521731  1.40528976] - virginica
[-0.84040723  2.58130115 -1.3444476  -1.5182445 ] - setosa
[ 0.35025236 -0.29346579  0.24448801  0.07641055] - versicolor
[-1.79293491  0.42522594 -1.457943   -1.38535658] - setosa
[-0.00694551 -0.05390188  0.69846961  0.74085016] - virginica

                       3. EVALUATION

Confusion Matrix:
[[13  0  0]
 [ 0 11  0]
 [ 0  1  5]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       0.92      1.00      0.96        11
   virginica       1.00      0.83      0.91         6

    accuracy                           0.97        30
   macro avg       0.97      0.94      0.96        30
weighted avg       0.97      0.97      0.97        30
```

Ex no: 06 Implement backpropagation algorithm for stock prices prediction.

Aim:

To implement the backpropagation algorithm to predict stock prices.

Dataset: Synthetically generated.

Algorithm used: Backpropagation Algorithm

Dataset Description:

No. of attributes: 3
No. of instances: 10
No. of features: 2, No. of target: 1

Packages:

NumPy

Algorithm:

Step 1: Start.

Step 2: Let each training example be a pair of the form $(\vec{x}, \vec{t})$, where $(\vec{x})$ is the vector of network input values, and $(\vec{t})$ is the vector of target network output values.

Step 3: Let η be the learning rate.

Step 4: Let n_i be the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

Step 5: Let the input from unit i to unit j be x_{ji} and the weight from unit i to unit j be w_{ji}.

Step 6: Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.

Step 7: Initialize all network weights to small random numbers.

Step 8: For each epoch, repeat steps 9 to 11.

Step 9: For each pair in training examples, repeat steps 10 to 11.

Step 10: Propagate the input forward through the network: Input the instance $\vec{x}$, to the network and compute the output o_u of every unit u in the network.

Step 11: Propagate the errors backward through the network: For each network output unit k, calculate its error term, for each hidden unit h, calculate its error term, and update each network weight w_{ji}.

Step 12: End.

Sample Input:

Relative Strength Index (RSI)	Dividend Yield
2	9
1	5
3	6
4	4

Expected Output:

Price
92
86
89
87

Program:

```
import numpy as np

# Features (X): [Relative Strength Index (RSI), Dividend Yield]
# Target (Y): Price
X = np.array(([2, 9], [1, 5], [3, 6], [4, 4], [7, 8], [5, 8], [8, 9], [6, 7], [7, 9], [10, 12]), dtype=float)
y = np.array(([92], [86], [89], [87], [94], [87], [96], [90], [91], [88]), dtype=float)
X = X / np.amax(X, axis=0)  # Maximum of X array longitudinally
y = y / 100

# SIGMOID FUNCTION
def sigmoid(x):
  return 1 / (1 + np.exp(-x))

# DERIVATIVE OF SIGMOID FUNCTION
def derivatives_sigmoid(x):
  return x * (1 - x)

# VARIABLE INITIALIZATION
epoch = 5000  # Setting training iterations
lr = 0.1  # Setting learning rate
input_layer_neurons = 2  # Number of features in data set
hidden_layer_neurons = 3  # Number of hidden layers neurons
output_neurons = 1  # Number of neurons at output layer

# WEIGHT AND BIAS INITIALIZATION
# Draws a random range of numbers uniformly of dim x*y
wh = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
bh = np.random.uniform(size=(1, hidden_layer_neurons))
wout = np.random.uniform(size=(hidden_layer_neurons, output_neurons))
bout = np.random.uniform(size=(1, output_neurons))

output = None
for i in range(epoch):
  # FORWARD PROPAGATION
  hinp1 = np.dot(X, wh)
  hinp = hinp1 + bh
  hlayer_act = sigmoid(hinp)
  outinp1 = np.dot(hlayer_act, wout)
  outinp = outinp1 + bout
  output = sigmoid(outinp)

  # BACKPROPAGATION
  EO = y - output
  outgrad = derivatives_sigmoid(output)
  d_output = EO * outgrad
  EH = d_output.dot(wout.T)

  # HOW MUCH HIDDEN LAYER wts CONTRIBUTED TO THE ERROR
  hiddengrad = derivatives_sigmoid(hlayer_act)
  d_hiddenlayer = EH * hiddengrad
  wout += hlayer_act.T.dot(d_output) * lr
```

```
# DOTPRODUCT OF NEXTLAYERERROR AND CURRENTLAYEROP
  wout += hlayer_act.T.dot(d_output) * lr
  wh += X.T.dot(d_hiddenlayer) * lr

print("Normalized Training Examples: \n" + str(X))
print("\nActual Prices: \n" + str(y))
print("\nPredicted Prices: \n", output)
```

Output:

```
Normalized Training Examples:
[[0.2        0.75      ]
 [0.1        0.41666667]
 [0.3        0.5       ]
 [0.4        0.33333333]
 [0.7        0.66666667]
 [0.5        0.66666667]
 [0.8        0.75      ]
 [0.6        0.58333333]
 [0.7        0.75      ]
 [1.         1.        ]]

Actual Prices:
[[0.92]
 [0.86]
 [0.89]
 [0.87]
 [0.94]
 [0.87]
 [0.96]
 [0.9 ]
 [0.91]
 [0.88]]

Predicted Prices:
 [[0.89801609]
 [0.89166727]
 [0.89589332]
 [0.89484696]
 [0.9026645 ]
 [0.90053923]
 [0.90445229]
 [0.90068218]
 [0.90351905]
 [0.90813285]]
```

Ex no: 07 Implement clustering algorithm for insurance fraud detection.

Aim:

To implement a clustering algorithm for insurance fraud detection.

Dataset: Insurance-Claims.csv

Training and Testing Dataset Size: 100% (Training) and NIL (Testing)

Algorithm used: K-Means Clustering

Dataset Description:

No. of attributes: 39
No. of instances: 1000
No. of features: 30, No. of target: 0

Packages:

Pandas

Matplotlib.pyplot

sklearn.cluster – Kmeans

Algorithm:

Step 1: Start.

Step 2: Import the necessary libraries.

Step 3: Load the dataset as a pandas dataframe.

Step 4: Check for missing values.

Step 5: Drop the unnecessary columns.

Step 6: Convert the categorical variables into numeric representation.

Step 7: Select the features.

Step 8: Create a KMeans clustering model with two clusters.

Step 9: Fit the clustering model to the selected features.

Step 10: Predict the cluster labels for each data point.

Step 11: Identify the fraudulent claims.

Step 12: Visualize the clusters.

Step 13: End.

Sample Input:

months_as_customer	age	Policy_state	policy_deductable	policy_annual_premium	umbrella_limit
328	48	OH	1000	1406.91	0
228	42	IN	2000	1197.22	5000000

insured_sex	insured_education_level	insured_occupation	insured_relationship	capital-gains	capital-loss
MALE	MD	craft-repair	husband	53300	0
MALE	MD	machine-op-inspct	other-relative	0	0

incident_type	collision_type	incident_severity	incident_state	incident_city	incident_hour_of_the_day	number_of_vehicles_involved
Single Vehicle Collision	Side Collision	Major Damage	SC	Columbus	5	1
Vehicle Theft	?	Minor Damage	VA	Riverwood	8	1

property_damage	bodily_injuries	witnesses	police_report_available	total_claim_amount	injury_claim	property_claim	vehicle_claim	auto_make
YES	1	2	YES	71610	6510	13020	52080	Saab
?	0	0	?	5070	780	780	3510	Mercedes

auto_year	fraud_reported
2004	Y
2007	Y

Expected Output:

A scatter plot with clusters identifying legitimate and fraudulent insurance claims.

Program:

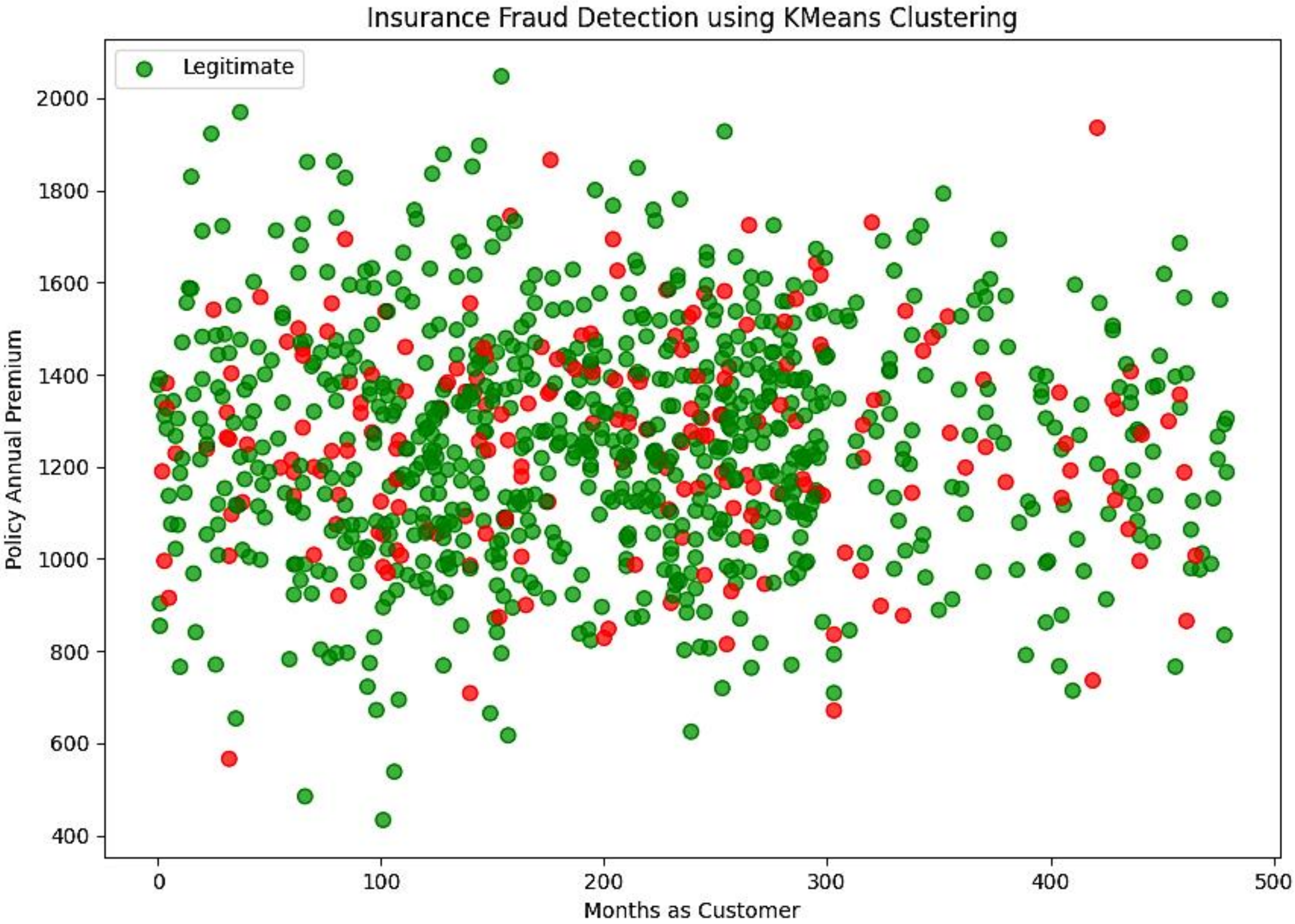
```
# 1. IMPORTING THE LIBRARIES AND LOADING THE DATASET
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

print()
print("1. LOADING THE DATASET".center(60), "\n")
df = pd.read_csv("Datasets\\P7-Insurance-Claims.csv")

print("Example Records From The Dataset")
print(df.head(5))
print("\n")

# ======================================================================

# 2. DATA PREPROCESSING
print("2. DATA PREPROCESSING".center(60), "\n")

print("Checking Missing Values:")
print(df.isnull().sum())
print()

print("Dropping Unnecessary Columns...")
df.drop(['policy_number', 'policy_bind_date', 'policy_csl', 'insured_hobbies', 'incident_date',
      'incident_location', 'auto_model', 'authorities_contacted', '_c39'], axis=1, inplace=True)
print()

print("Converting Categorical Values into Numeric Representation")
processed_data = pd.get_dummies(df, columns=['policy_state', 'insured_sex', 'insured_education_level',
                          'insured_occupation', 'insured_relationship', 'incident_type',
                          'collision_type', 'incident_severity', 'incident_state', 'incident_city',
                          'property_damage', 'police_report_available', 'auto_make', 'auto_year'],
                    dtype=int)
```

```
print("Example Records:")
print(processed_data.iloc[:, 3:].head(5), "\n")

# SPLIT THE FEATURES AND THE TARGET
X = processed_data.drop(['fraud_reported'], axis=1)
y = processed_data['fraud_reported']

# ======================================================================

kmeans = KMeans(n_clusters=2, n_init=10)  # CREATE A KMEANS CLUSTERING MODEL
kmeans.fit(X)  # FIT THE MODEL TO THE DATA
y_pred = kmeans.predict(X)  # PREDICT THE CLUSTER LABELS FOR EACH DATA POINT

fraudulent_claims = X[y_pred == 1]  # IDENTIFY THE FRAUDULENT CLAIMS

# VISUALIZE THE RESULTS
plt.figure(figsize=(10, 7))
cluster_labels = y_pred
cluster_colors = ["Green" if label == 0 else "Red" for label in cluster_labels]
plt.scatter(X['months_as_customer'], X['policy_annual_premium'], c=cluster_colors, s=50, alpha=0.7)
plt.xlabel('Months as Customer')
plt.ylabel('Policy Annual Premium')
plt.title('Insurance Fraud Detection using KMeans Clustering')
plt.legend(['Legitimate', 'Fraudulent'], loc='upper left')
plt.show()
```

Output:

```
                    1. LOADING THE DATASET

Example Records From The Dataset
   months_as_customer  age  policy_number  ... auto_year fraud_reported _c39
0                 328   48         521585  ...      2004              Y  NaN
1                 228   42         342868  ...      2007              Y  NaN
2                 134   29         687698  ...      2007              N  NaN
3                 256   41         227811  ...      2014              Y  NaN
4                 228   44         367455  ...      2009              N  NaN

[5 rows x 40 columns]


                    2. DATA PREPROCESSING

Checking Missing Values:
months_as_customer                  0
age                                 0
policy_number                       0
policy_bind_date                    0
policy_state                        0
policy_csl                          0
policy_deductable                   0
policy_annual_premium               0
umbrella_limit                      0
insured_zip                         0
insured_sex                         0
insured_education_level             0
insured_occupation                  0
insured_hobbies                     0
```

```
insured_relationship             0
capital-gains                    0
capital-loss                     0
incident_date                    0
incident_type                    0
collision_type                   0
incident_severity                0
authorities_contacted           91
incident_state                   0
incident_city                    0
incident_location                0
incident_hour_of_the_day         0
number_of_vehicles_involved      0
property_damage                  0
bodily_injuries                  0
witnesses                        0
police_report_available          0
total_claim_amount               0
injury_claim                     0
property_claim                   0
vehicle_claim                    0
auto_make                        0
auto_model                       0
auto_year                        0
fraud_reported                   0
_c39                          1000
dtype: int64

Dropping Unnecessary Columns...

Converting Categorical Values into Numeric Representation
Example Records:
   policy_annual_premium  umbrella_limit  ...  auto_year_2014  auto_year_2015
0                1406.91               0  ...               0               0
1                1197.22         5000000  ...               0               0
2                1413.14         5000000  ...               0               0
3                1415.74         6000000  ...               1               0
4                1583.91         6000000  ...               0               0

[5 rows x 113 columns]
```


Insurance Fraud Detection using KMeans Clustering
Legitimate
Policy Annual Premium
2000
1800
1600
1400
1200
1000
800
600
400
0
100
200
300
400
500
Months as Customer

Ex no: 08 Implement clustering algorithm for identifying cancerous data.

Aim:

To implement the KMeans clustering algorithm for identifying cancerous data.

Dataset: CancerData.csv

Training and Testing Dataset Size: 100% (Training) and NIL (Testing)

Algorithm used: K-Means Clustering

Dataset Description:

No. of attributes: 32
No. of instances: 569
No. of features: 3, No. of target: 0

Packages:

NumPy
Pandas
Matplotlib.pyplot
sklearn.cluster – Kmeans

Algorithm:

Step 1: Start.

Step 2: Import the necessary libraries.

Step 3: Load the dataset as a pandas dataframe.

Step 4: Select the features and generate feature vectors.

Step 5: Initialize the KMeans cluster module and set it to find two clusters, hoping to find malignant vs benign datapoints.

Step 6: Fit the model to the selected feature vectors.

Step 7: Compute the centroids and labels.

Step 8: Visualize the clusters by plotting the features and assigning color based on cluster identity label and plotting the centroids.

Step 9: Calculate and display the percentage matched between benign and malignant.

Step 10: End.

Sample Input:

radius_mean	concavity_mean	symmetry_mean
17.99	0.3001	0.2419
20.57	0.0869	0.1812

Expected Output:

Precent matched between benign and malignant datapoints: 85.0
A scatter plot with 3 axes (Radius mean, Concavity mean, and Symmetry mean) and two clusters.

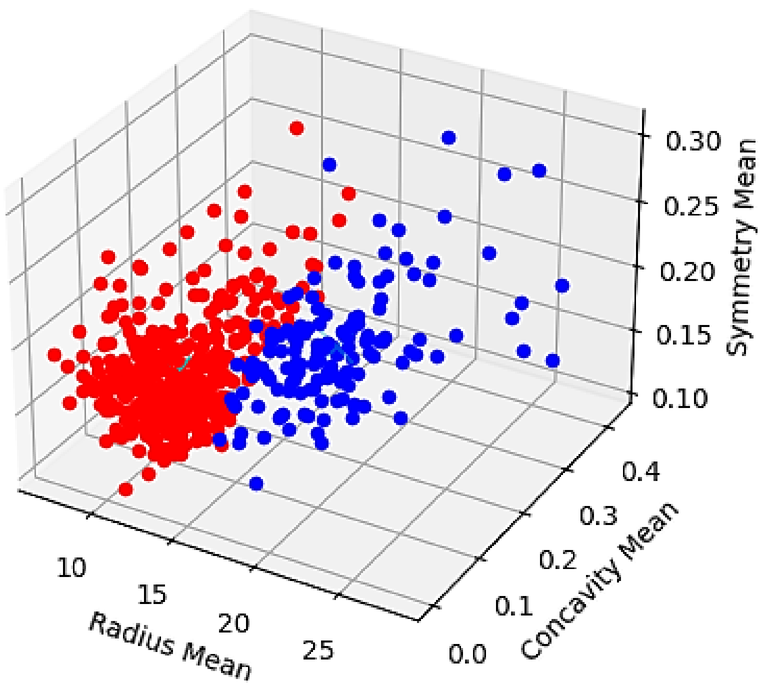
Program:

```
# 1. IMPORTING THE LIBRARIES AND LOADING THE DATASET
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans

data = pd.read_csv('Datasets\\P8-CancerData.csv')

# ===========================================================================


# 2. SELECTING THE FEATURES AND GENERATING THE FEATURE VECTORS.
feat_cols_sm = ["radius_mean", "concavity_mean", "symmetry_mean"]
features = np.array(data[feat_cols_sm])

# ===========================================================================


# 3. IMPLEMENTING THE KMeans CLUSTERING MODEL.
# Initialize the KMeans cluster module.
# Setting it to find two clusters, hoping to find malignant vs benign.
clusters = KMeans(n_clusters=2, n_init=10, max_iter=300)

# Fit model to our selected features.
clusters.fit(features)

# Put centroids and results into variables.
centroids = clusters.cluster_centers_
labels = clusters.labels_

# Sanity check
print("Sanity check: Centroids")
print(centroids)

# ===========================================================================


# 4. VISUALIZING THE CLUSTERS.

fig = plt.figure()  # Create new MatPlotLib figure
ax = fig.add_subplot(111, projection='3d')  # Add 3rd dimension to figure
colors = ["r", "b"]  # This means "red" and "blue"

# Plot all the features and assign color based on cluster identity label
for i in range(len(features)):
    ax.scatter(xs=features[i][0], ys=features[i][1], zs=features[i][2],
           c=colors[labels[i]], zdir='z')

# Plot centroids, though you can't really see them.
ax.scatter(xs=centroids[:, 0], ys=centroids[:, 1], zs=centroids[:, 2],
        marker="x", s=150, c="c")

# Create array of diagnosis data, which should be same length as labels.
diag = np.array(data['diagnosis'])

# Create variable to hold matches in order to get percentage accuracy.
matches = 0
```

```
# Transform diagnosis vector from B||M to 0||1 and matches++ if correct.
for i in range(0, len(diag)):
  if diag[i] == "B":
    diag[i] = 0
  if diag[i] == "M":
    diag[i] = 1
  if diag[i] == labels[i]:
    matches = matches + 1

# Calculate percentage matches and print.
percentMatch = (matches / len(diag)) * 100
print("Percent matched between benign and malignant ", percentMatch)

# Set labels on figure and show 3D scatter plot to visualize data and clusters.
ax.set_xlabel("Radius Mean")
ax.set_ylabel("Concavity Mean")
ax.set_zlabel("Symmetry Mean")
plt.show()
```

Output:

```
Sanity check: Centroids
[[12.44571194  0.06207506  0.17827541]
 [19.18387324  0.16916028  0.18984155]]
Percent matched between benign and malignant  85.58875219683657
```



Ex no: 09 Apply reinforcement learning and develop a game of your own.

Aim:

To develop a game by applying reinforcement learning.

Task: Reinforcement learning

Algorithm used: Q-Learning

Packages:

Pygame
Random

Formula:

The Bellman Equation: $V(s) = \max_{a} (R(s, a) + \gamma v(s'))$

State (s)	Current state where the agent is in the environment
Next state (s')	After taking action(a) at state(s) the agent reaches s'
Value (v)	Numeric representation of a state which helps the agent to find its path
Reward (R)	Treat which the agent gets after performing an action(a)
V(s)	Value of the state s
R(s, a)	Reward for being in the state and performing an action a
Action (a)	Set of possible actions that can be taken by the agent in the state (s)
Discount factor (γ)	Determines how much the agent cares about rewards in the distant future relative to those in the immediate future. It has a value between 0 and 1. Lower value encourages short–term rewards while higher value promises long-term reward.
max	Denotes the most optimum action among all the actions that the agent can take in a particular state which can lead to the reward after repeating this process every consecutive step.

Formula to update Q-Table:

$$\underbrace{Q(S_t, A_t)}_{\text{New Q-value estimation}} \leftarrow \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}} + \underbrace{\alpha}_{\text{Learning Rate}} \underbrace{[\underbrace{\underbrace{R_{t+1}}_{\text{Immediate Reward}} + \underbrace{\gamma max_a Q(S_{t+1}, a)}_{\text{Discounted Estimate optimal Q-value of next state}}}_{\text{TD Target}} - \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}}]}_{\text{TD Error}}$$

State(s)	The current position of the agent in the environment
Action (a)	A step taken by the agent in a particular state
Rewards	For every action, the agent receives a reward and penalty
Episodes	The end of the stage, where agents can't take new action. It happens when the agent has achieved the goal or failed.
$Q(S_{t+1}, a)$	Expected optimal Q-value of doing the action in a particular state
$Q(S_t, A_t)$	Current estimation of **$Q(S_{t+1}, a)$**
Q-Table	The agent maintains the Q-table of sets of states and actions
Temporal Differences (TD)	Used to estimate the expected value of $Q(S_{t+1}, a)$ by using the current state and action and previous state and action.

Algorithm:

Step 1: Start.

Step 2: Import the necessary libraries.

Step 3: Instantiate the game clock.

Step 4: Create the Q-table and populate it.

Step 5: Initialize the game variables such as ball coordinates, ball trajectory, paddle coordinates, and scores.

Step 6: Run the game loop and execute steps 7 to 13 until quit event.

Step 7: Get new state and update Q-values.

Step 8: Check for quit event. If True, go to step 14.

Step 9: Get current state and Q-values.

Step 10: Choose action with highest Q-value.

Step 11: Update game state based on chosen action.

Step 12: Draw game objects.

Step 13: Tick the clock at 60FPS.

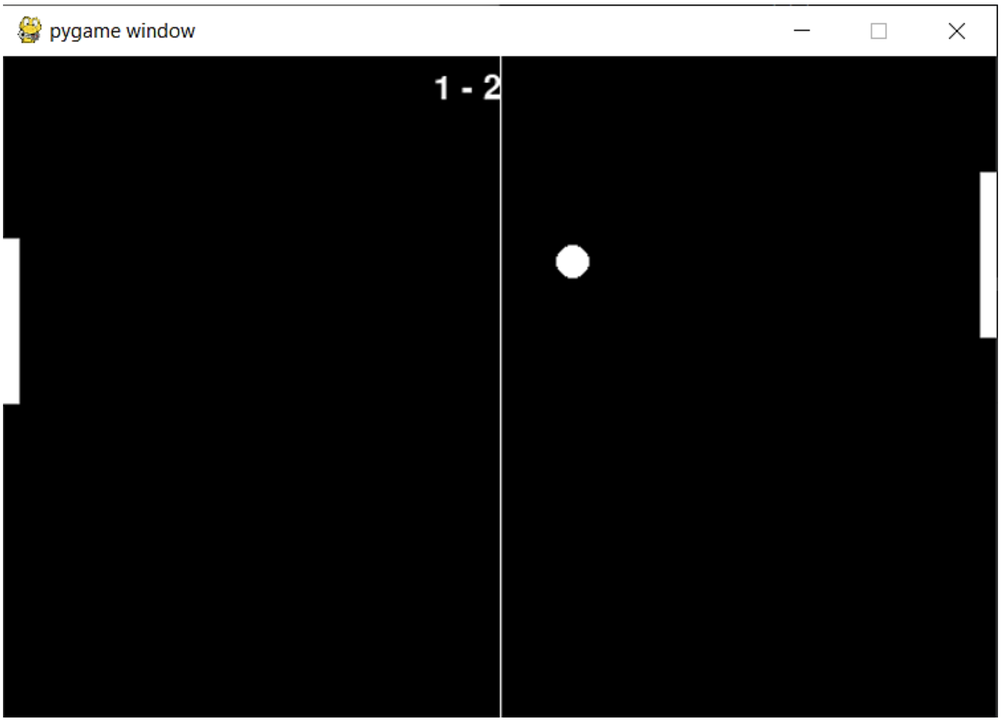
Step 14: End.

Program:

```
import pygame
import random

pygame.init()
screen = pygame.display.set_mode((600, 400))
clock = pygame.time.Clock()

# INITIALIZE Q TABLE
q_table = {}
for i in range(-10, 11):
    for j in range(-10, 11):
        for k in range(-10, 11):
            for l in range(-10, 11):
                q_table[(i, j, k, l)] = [random.uniform(0, 1) for _ in range(3)]

# INITIALIZE GAME VARIABLES
ball_x, ball_y = 300, 200
ball_dx, ball_dy = random.choice([-4, 4]), random.choice([-4, 4])
paddle1_y, paddle2_y = 150, 150
score1, score2 = 0, 0


# UPDATE GAME STATE
def update_game_state(action):
    global ball_x, ball_y, ball_dx, ball_dy, paddle1_y, paddle2_y, score1, score2
    if action == 0 and paddle1_y > 0:
        paddle1_y -= 5
    elif action == 2 and paddle1_y < 300:
        paddle1_y += 5
    ball_x += ball_dx
    ball_y += ball_dy

    if ball_y < 0 or ball_y > 390:
        ball_dy *= -1
    if ball_x < 20 and paddle1_y < ball_y < paddle1_y + 100:
        ball_dx *= -1
        score1 += 1
    elif ball_x > 580 and paddle2_y < ball_y < paddle2_y + 100:
        ball_dx *= -1
        score2 += 1
    elif ball_x < 0 or ball_x > 600:
        ball_x, ball_y = 300, 200
        ball_dx, ball_dy = random.choice([-4, 4]), random.choice([-4, 4])
        score1, score2 = 0, 0
```

```
    if ball_y < paddle2_y + 50 and paddle2_y > 0:
        paddle2_y -= 5
    elif ball_y > paddle2_y + 50 and paddle2_y < 300:
        paddle2_y += 5


# DRAW GAME OBJECTS
def draw_game_objects():
    screen.fill((0, 0, 0))
    pygame.draw.rect(screen, (255, 255, 255), pygame.Rect(0, paddle1_y, 10, 100))
    pygame.draw.rect(screen, (255, 255, 255), pygame.Rect(590, paddle2_y, 10, 100))
    pygame.draw.circle(screen, (255, 255, 255), (int(ball_x), int(ball_y)), 10)
    pygame.draw.line(screen, (255, 255, 255), (300, 0), (300, 400))
    font = pygame.font.SysFont('', 30)
    score_text = font.render(str(score1) + " - " + str(score2), True, (255, 255, 255))
    screen.blit(score_text, (260, 10))
    pygame.display.flip()


# RUN GAME LOOP
while True:
    # Get New State and Update Q-Value
    new_state = (int(ball_x / 10) - int(paddle1_y / 10), int(ball_y / 10), int(paddle2_y / 10),
            int(ball_dx / abs(ball_dx)), int(ball_dy / abs(ball_dy)))

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()

    # Get Current State and Q-Values
    state = new_state
    q_values = q_table.get(state)
    if q_values is None:
        # Initialize New State with Random Q-Values
        q_table[state] = [random.uniform(0, 1) for _ in range(3)]

    # Choose Action with Highest Q-Value
    action = q_table[state].index(max(q_table[state]))

    # Update Game State based on Chosen Action
    update_game_state(action)

    # Draw Game Objects
    draw_game_objects()

    # reward = score1 - score2

    # Limit Game to 60 FPS
    clock.tick(60)
```

Output:



Ex no: 10 Develop a traffic signal control system using reinforcement learning technique

Aim:

To develop a traffic signal control system using reinforcement learning technique.

Algorithm:

Step 1: Start.

Step 2: Import the necessary libraries.

Step 3: Initialize each signal timer with default values.

Step 4: Create appropriate data structures to store the list of signals, status of each signal, average speeds of each vehicles, and coordinates of each element of the game.

Step 5: Initialize pygame, set screen dimensions, load image assets, and initialize the simulation.

Step 6: Repeat steps 7 to 9 until the quit event is encountered.

Step 7: Generate vehicles in the simulation.

Step 8: Implement the traffic signal control by traffic signal sequencing and signal timer switch between red, yellow, and green signals.

Step 9: Update vehicle coordinates for movement.

Step 10: End.

Program:

```
import random
import time
import threading
import pygame
import sys

# Default values of signal timers
defaultGreen = {0: 10, 1: 10, 2: 10, 3: 10}
defaultRed = 150
defaultYellow = 5

signals = []
noOfSignals = 4
currentGreen = 0  # Indicates which signal is green currently
nextGreen = (currentGreen + 1) % noOfSignals  # Indicates which signal will turn green next
currentYellow = 0  # Indicates whether yellow signal is on or off

speeds = {'car': 2.25, 'bus': 1.8, 'truck': 1.8, 'bike': 2.5}  # average speeds of vehicles

# Coordinates of vehicles' start
x = {'right': [0, 0, 0], 'down': [755, 727, 697], 'left': [1400, 1400, 1400], 'up': [602, 627, 657]}
y = {'right': [348, 370, 398], 'down': [0, 0, 0], 'left': [498, 466, 436], 'up': [800, 800, 800]}

vehicles = {'right': {0: [], 1: [], 2: [], 'crossed': 0}, 'down': {0: [], 1: [], 2: [], 'crossed': 0},
        'left': {0: [], 1: [], 2: [], 'crossed': 0}, 'up': {0: [], 1: [], 2: [], 'crossed': 0}}
vehicleTypes = {0: 'car', 1: 'bus', 2: 'truck', 3: 'bike'}
directionNumbers = {0: 'right', 1: 'down', 2: 'left', 3: 'up'}
```

```
# Coordinates of signal image, timer, and vehicle count
signalCoods = [(530, 230), (810, 230), (810, 570), (530, 570)]
signalTimerCoods = [(530, 210), (810, 210), (810, 550), (530, 550)]

# Coordinates of stop lines
stopLines = {'right': 590, 'down': 330, 'left': 800, 'up': 535}
defaultStop = {'right': 580, 'down': 320, 'left': 810, 'up': 545}
# stops = {'right': [580,580,580], 'down': [320,320,320], 'left': [810,810,810], 'up': [545,545,545]}

# Gap between vehicles
stoppingGap = 15  # stopping gap
movingGap = 15  # moving gap

pygame.init()
simulation = pygame.sprite.Group()


class TrafficSignal:
  def __init__(self, red, yellow, green):
    self.red = red
    self.yellow = yellow
    self.green = green
    self.signalText = ""


class Vehicle(pygame.sprite.Sprite):
  def __init__(self, lane, vehicleClass, direction_number, direction):
    pygame.sprite.Sprite.__init__(self)
    self.lane = lane
    self.vehicleClass = vehicleClass
    self.speed = speeds[vehicleClass]
    self.direction_number = direction_number
    self.direction = direction
    self.x = x[direction][lane]
    self.y = y[direction][lane]
    self.crossed = 0
    vehicles[direction][lane].append(self)
    self.index = len(vehicles[direction][lane]) - 1
    path = "images/" + direction + "/" + vehicleClass + ".png"
    self.image = pygame.image.load(path)

    if (len(vehicles[direction][lane]) > 1 and vehicles[direction][lane][
      # if more than 1 vehicle in the lane of vehicle before it has crossed stop line
      self.index - 1].crossed == 0):
      if direction == 'right':
        self.stop = vehicles[direction][lane][self.index - 1].stop - vehicles[direction][lane][
          self.index - 1].image.get_rect().width - stoppingGap  # setting stop coordinate as: stop coordinate of
                                                                # next vehicle - width of next vehicle - gap
      elif direction == 'left':
        self.stop = vehicles[direction][lane][self.index - 1].stop + vehicles[direction][lane][
          self.index - 1].image.get_rect().width + stoppingGap
      elif direction == 'down':
        self.stop = vehicles[direction][lane][self.index - 1].stop - vehicles[direction][lane][
          self.index - 1].image.get_rect().height - stoppingGap
      elif direction == 'up':
        self.stop = vehicles[direction][lane][self.index - 1].stop + vehicles[direction][lane][
          self.index - 1].image.get_rect().height + stoppingGap
    else:
      self.stop = defaultStop[direction]
```

```
    # Set new starting and stopping coordinate
    if direction == 'right':
        temp = self.image.get_rect().width + stoppingGap
        x[direction][lane] -= temp
    elif direction == 'left':
        temp = self.image.get_rect().width + stoppingGap
        x[direction][lane] += temp
    elif direction == 'down':
        temp = self.image.get_rect().height + stoppingGap
        y[direction][lane] -= temp
    elif direction == 'up':
        temp = self.image.get_rect().height + stoppingGap
        y[direction][lane] += temp
    simulation.add(self)

def render(self, screen):
    screen.blit(self.image, (self.x, self.y))

def move(self):
    if self.direction == 'right':
        if self.crossed == 0 and self.x + self.image.get_rect().width > stopLines[self.direction]:
            # if the image has crossed stop line now
            self.crossed = 1
        if ((self.x + self.image.get_rect().width <= self.stop or self.crossed == 1 or (
                currentGreen == 0 and currentYellow == 0)) and (
                self.index == 0 or self.x + self.image.get_rect().width < (
                vehicles[self.direction][self.lane][self.index - 1].x - movingGap))):
            # (if the image has not reached its stop coordinate or has crossed stop line or has green signal) and
            # (it is either the first vehicle in that lane or it is has enough gap to the next vehicle in that lane)
            self.x += self.speed  # move the vehicle
    elif self.direction == 'down':
        if self.crossed == 0 and self.y + self.image.get_rect().height > stopLines[self.direction]:
            self.crossed = 1
        if ((self.y + self.image.get_rect().height <= self.stop or self.crossed == 1 or (
                currentGreen == 1 and currentYellow == 0)) and (
                self.index == 0 or self.y + self.image.get_rect().height < (
                vehicles[self.direction][self.lane][self.index - 1].y - movingGap))):
            self.y += self.speed
    elif self.direction == 'left':
        if self.crossed == 0 and self.x < stopLines[self.direction]:
            self.crossed = 1
        if ((self.x >= self.stop or self.crossed == 1 or (currentGreen == 2 and currentYellow == 0)) and (
                self.index == 0 or self.x > (
                vehicles[self.direction][self.lane][self.index - 1].x + vehicles[self.direction][self.lane][
            self.index - 1].image.get_rect().width + movingGap))):
            self.x -= self.speed
    elif self.direction == 'up':
        if self.crossed == 0 and self.y < stopLines[self.direction]:
            self.crossed = 1
        if ((self.y >= self.stop or self.crossed == 1 or (currentGreen == 3 and currentYellow == 0)) and
                (self.index == 0 or self.y >
                 (vehicles[self.direction][self.lane][self.index - 1].y +
                  vehicles[self.direction][self.lane][self.index - 1].image.get_rect().height + movingGap))):
            self.y -= self.speed
```

```
# Initialization of signals with default values
def initialize():
    ts1 = TrafficSignal(0, defaultYellow, defaultGreen[0])
    signals.append(ts1)
    ts2 = TrafficSignal(ts1.red + ts1.yellow + ts1.green, defaultYellow, defaultGreen[1])
    signals.append(ts2)
    ts3 = TrafficSignal(defaultRed, defaultYellow, defaultGreen[2])
    signals.append(ts3)
    ts4 = TrafficSignal(defaultRed, defaultYellow, defaultGreen[3])
    signals.append(ts4)
    repeat()


def repeat():
    global currentGreen, currentYellow, nextGreen
    while signals[currentGreen].green > 0:  # while the timer of current green signal is not zero
        updateValues()
        time.sleep(1)
    currentYellow = 1  # set yellow signal on
    # reset stop coordinates of lanes and vehicles
    for i in range(0, 3):
        for vehicle in vehicles[directionNumbers[currentGreen]][i]:
            vehicle.stop = defaultStop[directionNumbers[currentGreen]]
    while signals[currentGreen].yellow > 0:  # while the timer of current yellow signal is not zero
        updateValues()
        time.sleep(1)
    currentYellow = 0  # set yellow signal off

    # reset all signal times of current signal to default times
    signals[currentGreen].green = defaultGreen[currentGreen]
    signals[currentGreen].yellow = defaultYellow
    signals[currentGreen].red = defaultRed

    currentGreen = nextGreen  # set next signal as green signal
    nextGreen = (currentGreen + 1) % noOfSignals  # set next green signal
    signals[nextGreen].red = signals[currentGreen].yellow + signals[
        currentGreen].green  # set the red time of next to next signal as (yellow time + green time) of next signal
    repeat()

# Update values of the signal timers after every second
def updateValues():
    for i in range(0, noOfSignals):
        if i == currentGreen:
            if currentYellow == 0:
                signals[i].green -= 1
            else:
                signals[i].yellow -= 1
        else:
            signals[i].red -= 1

# Generating vehicles in the simulation
def generateVehicles():
    while True:
        vehicle_type = random.randint(0, 3)
        lane_number = random.randint(1, 2)
        temp = random.randint(0, 99)
        direction_number = 0
        dist = [25, 50, 75, 100]
        if temp < dist[0]:
            direction_number = 0
```

```
    elif temp < dist[1]:
      direction_number = 1
    elif temp < dist[2]:
      direction_number = 2
    elif temp < dist[3]:
      direction_number = 3
    Vehicle(lane_number, vehicleTypes[vehicle_type], direction_number,
                                directionNumbers[direction_number])
    time.sleep(1)

class Main:
  thread1 = threading.Thread(name="initialization", target=initialize, args=())  # initialization
  thread1.daemon = True
  thread1.start()
  # Colours
  black = (0, 0, 0)
  white = (255, 255, 255)

  # Screensize
  screenWidth = 1400
  screenHeight = 800
  screenSize = (screenWidth, screenHeight)

  # Setting background image i.e. image of intersection
  background = pygame.image.load('images/intersection.png')
  screen = pygame.display.set_mode(screenSize)
  pygame.display.set_caption("SIMULATION")

  # Loading signal images and font
  redSignal = pygame.image.load('images/signals/red.png')
  yellowSignal = pygame.image.load('images/signals/yellow.png')
  greenSignal = pygame.image.load('images/signals/green.png')
  font = pygame.font.Font(None, 30)

  thread2 = threading.Thread(name="generateVehicles", target=generateVehicles, args=())  # Generating
                                                                                          # vehicles
  thread2.daemon = True
  thread2.start()

  while True:
    for event in pygame.event.get():
      if event.type == pygame.QUIT:
        sys.exit()

    screen.blit(background, (0, 0))  # display background in simulation
    for i in range(0,
           noOfSignals):  # display signal and set timer according to current status: green, yello, or red
      if i == currentGreen:
        if currentYellow == 1:
          signals[i].signalText = signals[i].yellow
          screen.blit(yellowSignal, signalCoods[i])
        else:
          signals[i].signalText = signals[i].green
          screen.blit(greenSignal, signalCoods[i])
      else:
        if signals[i].red <= 10:
          signals[i].signalText = signals[i].red
        else:
          signals[i].signalText = "---"
        screen.blit(redSignal, signalCoods[i])
    signalTexts = ["", "", "", ""]
```

```
    # display signal timer
    for i in range(0, noOfSignals):
      signalTexts[i] = font.render(str(signals[i].signalText), True, white, black)
      screen.blit(signalTexts[i], signalTimerCoods[i])

    # display the vehicles
    for vehicle in simulation:
      screen.blit(vehicle.image, [vehicle.x, vehicle.y])
      vehicle.move()
    pygame.display.update()


Main()
```

Output: