MACHINE LEARNING

LAB EXPERIEMENTS

SUFYAAN EXP 1

```python
# a) Operation of data types in Python.

# Create a variable of type integer.
print("experiement 1a")
x = 10

# Create a variable of type float.
y = 10.5

# Create a variable of type string.
z = "Hello, world!"

# Print the values of the variables.
print(x)
print(y)
print(z)

# b) Different Arithmetic Operations on numbers in Python.
print("\n experiement 1b")
# Add two numbers.
x = 10
y = 5

z = x + y

# Print the sum of the two numbers.
print(z)

# Subtraction
d = x - y
print("Subtraction:", d)

# Multiplication
e = x * y
print("Multiplication:", e)

# Division
f = x / y
print("Division:", f)

# Modulo
g = x % y
print("Modulo:", g)
```

```python
# Exponentiation
h = x ** y
print("Exponentiation:", h)

# Floor division
i = x // y
print("Floor division:", i)
# c) Create, concatenate and print a string and access substring from
a given string.
print("\n experiement 1c")
# Create a string.
str = "Hello, world!"

# Print the string.
print(str)

# Concatenate two strings.
str1 = "Hello"
str2 = "world!"

str = str1 + str2

# Print the concatenated string.
print(str)

# Access a substring from a string.
str = "Hello, world!"

substring = str[0:5]

# Print the substring.
print(substring)

# d) Append, and remove lists in python.
print("\n experiement 1d")
# Create a list.
list = [1, 2, 3, 4, 5]

# Append an element to the list.
list.append(6)

# Print the list.
print(list)

# Remove an element from the list.
list.remove(3)

# Print the list.
print(list)

# e) Demonstrate working with tuples in python.
```

```
print("\n experiement 1e")
# Create a tuple.
tuple = (1, 2, 3, 4, 5)

# Print the tuple.
print(tuple)

# Access an element from a tuple.
element = tuple[0]

# Print the element.
print(element)

# f) Demonstrate working with dictionaries in python.
print("\n experiement 1f")
# Create a dictionary.
dict = {"name": "John", "age": 30}

# Print the dictionary.
print(dict)

# Access a value from a dictionary.
value = dict["name"]

# Print the value.
print(value)

dict["occupation"] = 'engineer'
del dict['name']
print(dict)
```

```
experiement 1a
10
10.5
Hello, world!

 experiement 1b
15
Subtraction: 5
Multiplication: 50
Division: 2.0
Modulo: 0
Exponentiation: 100000
Floor division: 2

 experiement 1c
Hello, world!
Helloworld!
Hello

 experiement 1d
```

```
[1, 2, 3, 4, 5, 6]
[1, 2, 4, 5, 6]

 experiement 1e
(1, 2, 3, 4, 5)
1

 experiement 1f
{'name': 'John', 'age': 30}
John
{'age': 30, 'occupation': 'engineer'}
```

SUFYAAN EXP2

```python
import numpy as np

# Define the two arrays.
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([6, 7, 8, 9, 10])

# Compute the expected value.
expected_value = np.mean(array1)

# Compute the mean.
mean = np.mean(array1)

# Compute the standard deviation.
standard_deviation = np.std(array1)

# Compute the variance.
variance = np.var(array1)

# Compute the covariance.
covariance = np.cov(array1, array2)[0][1]

# Compute the covariance matrix.
covariance_matrix = np.cov(array1, array2)

# Print the results.
print("Expected value:", expected_value)
print("Mean:", mean)
print("Standard deviation:", standard_deviation)
print("Variance:", variance)
print("Covariance:", covariance)
print("Covariance matrix:", covariance_matrix)

Expected value: 3.0
Mean: 3.0
Standard deviation: 1.4142135623730951
Variance: 2.0
```

```
Covariance: 2.5
Covariance matrix: [[2.5 2.5]
 [2.5 2.5]]
```

SUFYAAN EXP3

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn

# Path to the CSV file
csv_file = 'data.csv'

# Read the CSV file into a DataFrame
df = pd.read_csv(csv_file)

# Print the contents of the DataFrame
print(df)

def find_missing_numbers(numbers):
    missing_numbers = []
    for i in range(1, len(numbers) + 1):
        if i not in numbers:
            missing_numbers.append(i)
    return missing_numbers


# Example usage
data = [1, 3, 5, 6, 8, 10]
missing = find_missing_numbers(data)
print("Missing numbers:", missing)
from sklearn.preprocessing import LabelEncoder

# Categorical data
data = ['red', 'blue', 'green', 'blue', 'red', 'green', 'green']

# Create an instance of LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the data
encoded_data = label_encoder.fit_transform(data)

# Print the encoded data
print("Encoded Data:", encoded_data)

# Print the original classes
print("Original Classes:", label_encoder.classes_)
import random
```

```python
# Original dataset
dataset = ['data_point1', 'data_point2', 'data_point3', 'data_point4',
'data_point5']

# Set the seed for reproducibility (optional)
random.seed(42)

# Specify the split ratio (e.g., 80% training, 20% testing)
split_ratio = 0.8

# Calculate the number of training samples based on the split ratio
num_train_samples = int(len(dataset) * split_ratio)

# Shuffle the dataset randomly
random.shuffle(dataset)

# Split the dataset into training and test data
train_data = dataset[:num_train_samples]
test_data = dataset[num_train_samples:]

# Print the training and test data
print("Training Data:")
print(train_data)
print("\nTest Data:")
print(test_data)
import numpy as np

# Input data
data = np.array([[2, 1000], [5, 2000], [10, 5000], [3, 1500]])

# Min-Max Scaling
min_val = np.min(data, axis=0)
max_val = np.max(data, axis=0)
scaled_data = (data - min_val) / (max_val - min_val)

print("Min-Max Scaling:")
print(scaled_data)

# Standardization
mean_val = np.mean(data, axis=0)
std_val = np.std(data, axis=0)
standardized_data = (data - mean_val) / std_val

print("\nStandardization:")
print(standardized_data)
```

	23	0	63	90	20	100	30.4	0.25	1
0	20	10	110	80	40	140	26.4	0.20	1
1	25	5	140	70	30	120	33.6	0.28	0
2	30	8	110	60	20	90	25.2	0.35	0

```
3    35  12  120   70  40  110  30.8  0.40  0
4    40   4   90   50  20   80  23.4  0.38  0
5    45   2  120   60  30  100  28.0  0.45  0
6    50   6  130   70  40  130  32.4  0.48  0
7    55   9   80   50  20   70  21.6  0.32  0
8    60  11  100   60  30   90  26.4  0.38  0
9    65  15  160   90  30  180  38.0  0.59  1
10   70  17  190  100  35  200  42.4  0.68  1
11   75  19  220  110  40  220  46.8  0.77  1
12   80  21  250  120  45  240  51.2  0.86  1
13   85  23  280  130  50  260  55.6  0.95  1
14   90  25  310  140  55  280  60.0  1.04  1
15   95  27  340  150  60  300  64.4  1.13  1
16  100  29  370  160  65  320  68.8  1.22  1
17  105  31  400  170  70  340  73.2  1.31  1
18  110  33  430  180  75  360  77.6  1.40  1
Missing numbers: [2, 4]
Encoded Data: [2 0 1 0 2 1 1]
Original Classes: ['blue' 'green' 'red']
Training Data:
['data_point4', 'data_point2', 'data_point3', 'data_point5']

Test Data:
['data_point1']
Min-Max Scaling:
[[0.    0.   ]
 [0.375 0.25 ]
 [1.    1.   ]
 [0.125 0.125]]

Standardization:
[[-0.97332853 -0.88354126]
 [ 0.         -0.2409658 ]
 [ 1.62221421  1.68676059]
 [-0.64888568 -0.56225353]]
```

SUFYAAN EXP 4

```
import pandas as pd
import numpy as np

# Function to generate synthetic diabetes dataset with 100 rows
def generate_synthetic_dataset(num_rows=100):
    np.random.seed(42)

    # Generate random values for each attribute based on some ranges
    feature1 = np.random.randint(20, 70, num_rows)
    feature2 = np.random.randint(0, 15, num_rows)
    feature3 = np.random.randint(60, 200, num_rows)
```

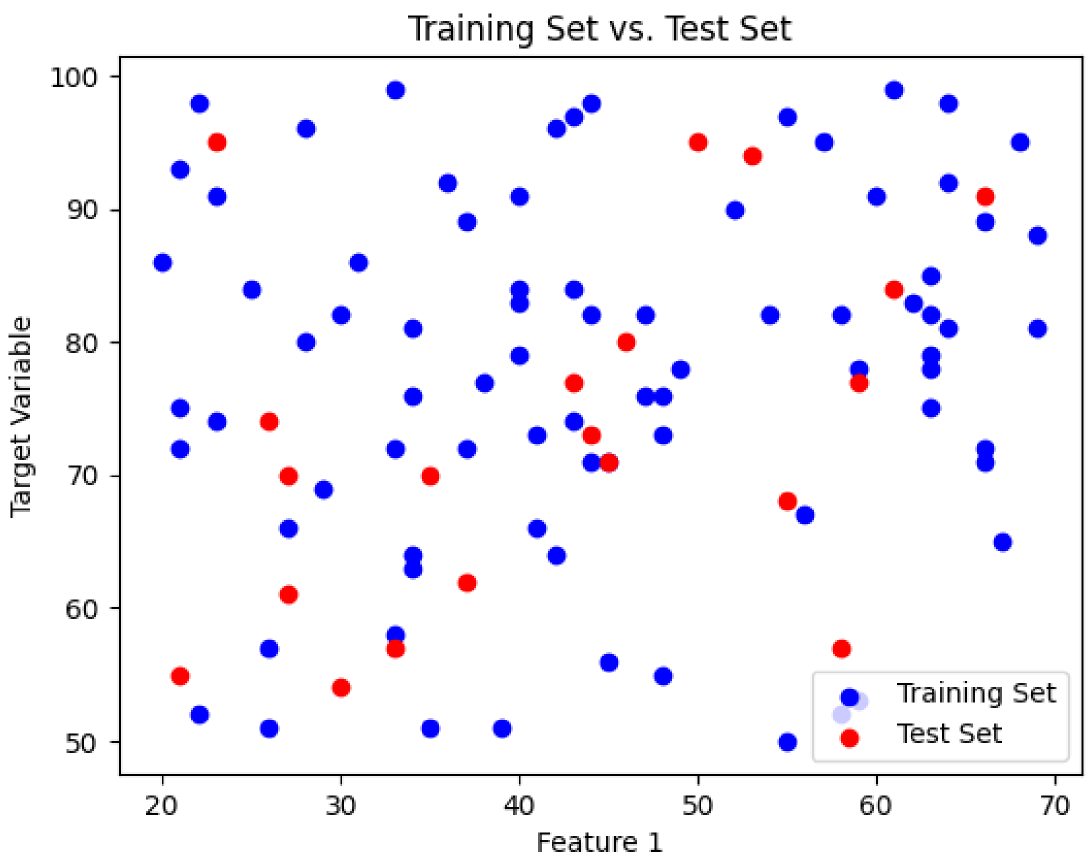
```
    target_variable = np.random.randint(50, 100, num_rows)

    # Create the synthetic dataset
    dataset = pd.DataFrame({
        'feature1': feature1,
        'feature2': feature2,
        'feature3': feature3,
        'target_variable': target_variable
    })

    return dataset

# Generate the synthetic dataset with 100 rows
dataset = generate_synthetic_dataset(num_rows=100)

# Save the dataset to a CSV file
dataset.to_csv('dataset.csv', index=False)
```

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


# Load your dataset into a pandas DataFrame
# Assuming you have a CSV file named 'data.csv' with features X and
target variable y
data = pd.read_csv('dataset.csv')

# Split the data into features (X) and target variable (y)
X = data[['feature1', 'feature2', 'feature3']]  # Replace 'feature1',
'feature2', 'feature3' with your actual feature names
y = data['target_variable']  # Replace 'target_variable' with your
actual target variable name

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)  # Adjust test_size as desired


# Create a linear regression model
model = LinearRegression()
# Train the model using the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

```python
# Compare actual values with predicted values
result_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(result_df)

# Plot the training set
plt.scatter(X_train['feature1'], y_train, color='blue',
label='Training Set')
# Plot the test set
plt.scatter(X_test['feature1'], y_test, color='red', label='Test Set')
# Plot the best-fit line
# plt.plot(X_train['feature1'], model.predict(X_train), color='black',
label='Best Fit')

plt.xlabel('Feature 1')
plt.ylabel('Target Variable')
plt.title('Training Set vs. Test Set')
plt.legend()
plt.show()



# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the evaluation metrics
print('Mean Squared Error:', mse)
print('R-squared:', r2)
```

	Actual	Predicted
83	77	78.569972
53	91	76.837071
70	62	73.285512
45	73	81.989229
44	95	75.673908
39	74	75.428690
22	55	72.829900
80	61	73.984318
10	54	76.826774
0	57	76.984376
18	77	74.738071
30	80	75.733680
73	94	76.136241
33	70	76.251352
90	71	74.311676
4	70	73.686093
76	57	75.263717

```
77       95  77.680035
12       68  79.384234
31       84  78.143493
```




```
Mean Squared Error: 169.86028188186393
R-squared: 0.013515604315853857
```

SUFYAAN EXP 5

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the dataset
data = pd.read_excel('iml exp 5.xlsx')  # Replace 'your_dataset.csv'
with the actual filename

# Display the first few rows of the dataset
print(data.head())
# Split the dataset into features (X) and target variable (y)
```

```
X = data[['Gender', 'Age', 'EstimatedSalary']]  # Replace with
relevant feature columns
y = data['Purchased']  # Replace with the target variable column

# Convert categorical variable 'Gender' into numeric using one-hot
encoding
X = pd.get_dummies(X, drop_first=True)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Split the dataset into features (X) and target variable (y)
X = data[['Gender', 'Age', 'EstimatedSalary']]  # Replace with
relevant feature columns
y = data['Purchased']  # Replace with the target variable column

# Convert categorical variable 'Gender' into numeric using one-hot
encoding
X = pd.get_dummies(X, drop_first=True)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Create a logistic regression model
model = LogisticRegression()

# Train the model using the training data
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
Accuracy: 0.75
```

SUFYAAN EXP 6A

```
probAbsentFriday=0.03
probFriday=0.2
```

```
# bayes Formula
#p(Absent|Friday)=p(Friday|Absent)p(Absent)/p(Friday)
#p(Friday|Absent)=p(Friday∩Absent)/p(Absent)
# Therefore the result is:
bayesResult=(probAbsentFriday/probFriday)
print(bayesResult * 100)
```

```
15.0
```

SUFYAAN EXP 6B

```
import csv
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
# Step 1: Load the data
def load_data(filename):
 with open(filename, 'r') as csvfile:
    data = list(csv.reader(csvfile))
    return data
# Step 2: Preprocess the data (if needed)
# Step 3: Split the data
import numpy as np
def split_data(data, test_size):
 X = np.array([row[:-1] for row in data],dtype=np.float64)
 y = [row[-1] for row in data]
 X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size, random_state=42)
 return X_train, X_test, y_train, y_test
# Step 4: Train the Naïve Bayesian classifier
def train_naive_bayes(X_train, y_train):
 clf = GaussianNB()
 clf.fit(X_train, y_train)
 return clf
# Step 5: Make predictions
def predict(clf, X_test):
 return clf.predict(X_test)
# Step 6: Evaluate accuracy
def evaluate_accuracy(y_true, y_pred):
 return accuracy_score(y_true, y_pred)
# Main function
if __name__ == "__main__":
 filename = "data.csv"
 data = load_data(filename)
 # Set the test size for splitting the data
 test_size = 0.2
 X_train, X_test, y_train, y_test = split_data(data, test_size)
 clf = train_naive_bayes(X_train, y_train)
```

```
 y_pred = predict(clf, X_test)
 accuracy = evaluate_accuracy(y_test, y_pred)
 print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.5
```

SUFYAAN EXP 7

```
import pandas as pd
import numpy as np
import csv
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination

# Step 1: Load and preprocess the dataset
data = pd.read_csv("archive/diabetes.csv")
# data = data.replace('?',np.nan)

# data = data.apply(pd.to_numeric)
print('Few examples from the dataset are given below')
print(data.head())

# Step 2: Define the structure of the Bayesian network
from pgmpy.estimators import MaximumLikelihoodEstimator

model = BayesianNetwork(    [('Age', 'Outcome'),
    ('BMI', 'Outcome'),
    ('BloodPressure', 'Outcome'),
    ('Insulin', 'Outcome'),
    ('Glucose', 'Outcome'),
    ('Pregnancies', 'Outcome')
    ])

# Step 3: Learn the parameters of the Bayesian network
print('\n Learning CPD using Maximum likelihood estimators')
# model.fit(data)
model.fit(data, estimator=MaximumLikelihoodEstimator)

# estimator = MaximumLikelihoodEstimator(model, data)
# model.fit(data, estimator)

# Step 4: Perform diabetic patient detection
print('\n Inferencing with Bayesian Network:')

print('\n Inferencing with Bayesian Network:')

inference = VariableElimination(model)
# q=inference.query(variables=["Outcome"],evidence={'Age': 34})
```

```
# Sample test data for inference (you can use real test data here)
test_data = {
    'Age': 50,
 'BMI': 31,
 'BloodPressure': 1,
 'Insulin': 3,
    'Glucose' : 148,
    'Pregnancies' : 1
}

# Query to detect Diabetes (set 'Outcome' to None to calculate its
value)
result = inference.map_query(variables=['Outcome'],
evidence=test_data)

# Print the result
# print("Result: Diabetic" if result['Outcome'] == 1 else "Result:
Non-diabetic")
print("Result: Diabetic" if result['Outcome'] else "Result: Non-
diabetic")
# Print the result
print("Result: Diabetic" if result['Outcome'] == 1 else "Result: Non-
diabetic")

Few examples from the dataset are given below
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin
BMI  \
0            6      148             72             35        0  33.6

1            1       85             66             29        0  26.6

2            8      183             64              0        0  23.3

3            1       89             66             23       94  28.1

4            0      137             40             35      168  43.1


   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1

 Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 Inferencing with Bayesian Network:
```

```
c:\Python310\lib\site-packages\pgmpy\factors\discrete\
DiscreteFactor.py:541: UserWarning: Found unknown state name. Trying
to switch to using all state names as state numbers
  warnings.warn(
Finding Elimination Order: : : 0it [00:00, ?it/s]
0it [00:00, ?it/s]

Result: Non-diabetic
Result: Non-diabetic
```

```python
#Step 1: Import necessary libraries
import pandas as pd
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
#Step 2: Load and preprocess the dataset
data = pd.read_csv("diabetes_data.csv")
#Step 3: Define the structure of the Bayesian network
# Define the structure of the Bayesian network
model = BayesianModel([
 ('Age', 'Diabetes'),
 ('BMI', 'Diabetes'),
 ('BloodPressure', 'Diabetes'),
 ('Insulin', 'Diabetes'),
 ('Glucose', 'Diabetes'),
 ('Pregnancies', 'Diabetes'),
 ('DiabetesPedigreeFunction', 'Diabetes')
])
#Step 4: Learn the parameters of the Bayesian network
# Learn the parameters of the Bayesian network using Maximum
Likelihood Estimation
estimator = MaximumLikelihoodEstimator(model, data)
model.fit(data, estimator)
#Step 5: Perform diabetic patient detection
# Perform inference using Variable Elimination
inference = VariableElimination(model)
# Sample test data for inference (you can use real test data here)
test_data = {
 'Age': 40,
 'BMI': 25,
 'BloodPressure': 70,
 'Insulin': 100,
 'Glucose': 140,
 'Pregnancies': 0,
 'DiabetesPedigreeFunction': 0.2
}
# Query to detect Diabetes (set 'Diabetes' to None to calculate its
value)
result = inference.map_query(variables=['Diabetes'],
evidence=test_data)
```

```python
# Print the result
print("Result: Diabetic" if result['Diabetes'] else "Result: Non-diabetic")
```

```
c:\Python310\lib\site-packages\tqdm\auto.py:22: TqdmWarning: IProgress
not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
c:\Python310\lib\site-packages\pgmpy\models\BayesianModel.py:8:
FutureWarning: BayesianModel has been renamed to BayesianNetwork.
Please use BayesianNetwork class, BayesianModel will be removed in
future.
  warnings.warn(
```

```
-----------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
f:\sufyaaneng_sem4\ml\New folder\labprog\lab.ipynb Cell 6 in 2
     <a href='vscode-notebook-cell:/f%3A/sufyaaneng_sem4/ml/New
%20folder/labprog/lab.ipynb#W5sZmlsZQ%3D%3D?line=18'>19</a> #Step 4:
Learn the parameters of the Bayesian network
     <a href='vscode-notebook-cell:/f%3A/sufyaaneng_sem4/ml/New
%20folder/labprog/lab.ipynb#W5sZmlsZQ%3D%3D?line=19'>20</a> # Learn
the parameters of the Bayesian network using Maximum Likelihood
Estimation
     <a href='vscode-notebook-cell:/f%3A/sufyaaneng_sem4/ml/New
%20folder/labprog/lab.ipynb#W5sZmlsZQ%3D%3D?line=20'>21</a> estimator
= MaximumLikelihoodEstimator(model, data)
---> <a href='vscode-notebook-cell:/f%3A/sufyaaneng_sem4/ml/New
%20folder/labprog/lab.ipynb#W5sZmlsZQ%3D%3D?line=21'>22</a>
model.fit(data, estimator)
     <a href='vscode-notebook-cell:/f%3A/sufyaaneng_sem4/ml/New
%20folder/labprog/lab.ipynb#W5sZmlsZQ%3D%3D?line=22'>23</a> #Step 5:
Perform diabetic patient detection
     <a href='vscode-notebook-cell:/f%3A/sufyaaneng_sem4/ml/New
%20folder/labprog/lab.ipynb#W5sZmlsZQ%3D%3D?line=23'>24</a> # Perform
inference using Variable Elimination
     <a href='vscode-notebook-cell:/f%3A/sufyaaneng_sem4/ml/New
%20folder/labprog/lab.ipynb#W5sZmlsZQ%3D%3D?line=24'>25</a> inference
= VariableElimination(model)

File c:\Python310\lib\site-packages\pgmpy\models\
BayesianNetwork.py:577, in BayesianNetwork.fit(self, data, estimator,
state_names, complete_samples_only, n_jobs, **kwargs)
    575     estimator = MaximumLikelihoodEstimator
    576 else:
--> 577     if not issubclass(estimator, BaseEstimator):
    578         raise TypeError("Estimator object should be a valid
pgmpy estimator.")
```

```
    580 _estimator = estimator(
    581     self,
    582     data,
    583     state_names=state_names,
    584     complete_samples_only=complete_samples_only,
    585 )

TypeError: issubclass() arg 1 must be a class
```

SUFYAAN EXP 8

```
import pandas as pd
import numpy as np
# Sample dataset
data = {
 'Can Fly': ['Yes', 'Yes', 'No', 'No', 'Yes', 'No'],
 'Has Fur': ['No', 'Yes', 'Yes', 'Yes', 'No', 'Yes'],
 'Is Mammal': ['No', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}
df = pd.DataFrame(data)
def entropy(target_col):
 elements, counts = np.unique(target_col, return_counts=True)
 probabilities = counts / counts.sum()
 entropy_val = -sum(probabilities * np.log2(probabilities))
 return entropy_val
def information_gain(data, split_feature, target_feature):
 total_entropy = entropy(data[target_feature])
 # Calculate the weighted entropy for the two splits
 unique_values = data[split_feature].unique()
 weighted_entropy = 0
 for value in unique_values:
    subset = data[data[split_feature] == value]
    subset_entropy = entropy(subset[target_feature])
    weighted_entropy += (len(subset) / len(data)) * subset_entropy
    information_gain_val = total_entropy - weighted_entropy
    return information_gain_val
def build_tree(data, original_data, features, target_feature,
parent_node_class=None):
 # Base cases
 if len(np.unique(data[target_feature])) == 1:
    return np.unique(data[target_feature])[0]

 elif len(data) == 0:
    return np.unique(original_data[target_feature])
[np.argmax(np.unique(original_data[target_feature],
    return_counts=True)[1])]
 elif len(features) == 0:
    return parent_node_class
```

```
 else:
    parent_node_class = np.unique(data[target_feature])
[np.argmax(np.unique(data[target_feature],
return_counts=True)[1])]
 # Select the feature with the highest information gain
 item_values = [information_gain(data, feature, target_feature) for
feature in features]
 best_feature_index = np.argmax(item_values)
 best_feature = features[best_feature_index]
 # Create the tree structure
 tree = {best_feature: {}}
 # Remove the selected feature from the features list
 features = [f for f in features if f != best_feature]
 # Recurse on the subsets
 for value in np.unique(data[best_feature]):
    subset = data[data[best_feature] == value]
    subtree = build_tree(subset, data, features, target_feature,
parent_node_class)
    tree[best_feature][value] = subtree
    return tree
# Building the decision tree
features = ['Can Fly', 'Has Fur']
target_feature = 'Is Mammal'
tree = build_tree(df, df, features, target_feature)
print(tree)

{'Has Fur': {'No': {'Can Fly': {'Yes': 'No'}}}}
```

SUFYAAN EXP 9

```
import numpy as np
import pandas as pd
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
# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Standardize the features (optional but recommended for KNN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Define the value of k for KNN
k = 3
```

```
# Create the KNN classifier and fit it to the training data
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(X_train, y_train)
# Make predictions on the test set
y_pred = knn_classifier.predict(X_test)
# Calculate and print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

SUFYAAN EXP 10

```
from sklearn.cluster import KMeans
import numpy as np
X = np.array([[1.713,1.586], [0.180,1.786], [0.353,1.240],
[0.940,1.566], [1.486,0.759],
[1.266,1.106],[1.540,0.419],[0.459,1.799],[0.773,0.186]])
y=np.array([0,1,1,0,1,0,1,1,1])
kmeans = KMeans(n_clusters=3, random_state=0).fit(X,y)
print("The input data is ")
print("VAR1 \t VAR2 \t CLASS")
i=0
for val in X:
    print(val[0],"\t",val[1],"\t",y[i])
    i+=1
    print("="*20)
# To get test data from the user
print("The Test data to predict ")
test_data = []
VAR1 = float(input("Enter Value for VAR1 :"))
VAR2 = float(input("Enter Value for VAR2 :"))
test_data.append(VAR1)
test_data.append(VAR2)
print("="*20)
print("The predicted Class is : ", kmeans.predict([test_data]))
```

```
c:\Python310\lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
```

```
The input data is
VAR1   VAR2        CLASS
1.713       1.586       0
====================
0.18   1.786       1
====================
0.353       1.24       1
```

```
====================
0.94	1.566	0
====================
1.486	0.759	1
====================
1.266	1.106	0
====================
1.54	0.419	1
====================
0.459	1.799	1
====================
0.773	0.186	1
====================
The Test data to predict
====================
The predicted Class is :  [1]
```

SUFYAAN EXP 11

```
# Sample data (features: study hours, previous test scores; label: pass/fail)
import numpy as np
np.random.seed(42)
study_hours = np.random.randint(1, 8, 100)
test_scores = np.random.randint(40, 100, 100)

# Generate the target variable 'pass_fail' with a mix of class labels
pass_fail = np.zeros(100)
pass_indices = np.random.choice(100, 30, replace=False)
pass_fail[pass_indices] = 1

# Split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    np.column_stack((study_hours, test_scores)),
    pass_fail, test_size=0.2, random_state=42)


# Train different classifiers and evaluate their performance
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
# Logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred_logreg = logreg.predict(X_test)
# Decision Tree
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
```

```python
y_pred_dtree = dtree.predict(X_test)
# Random Forest
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
# Evaluate metrics
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score
# Logistic Regression
print("Logistic Regression:")
print("Accuracy:", accuracy_score(y_test, y_pred_logreg))
print("Precision:", precision_score(y_test, y_pred_logreg))
print("Recall:", recall_score(y_test, y_pred_logreg))
print("F1-score:", f1_score(y_test, y_pred_logreg))
print("AUC-ROC:", roc_auc_score(y_test, y_pred_logreg))
print()
# Decision Tree
print("Decision Tree:")
print("Accuracy:", accuracy_score(y_test, y_pred_dtree))
print("Precision:", precision_score(y_test, y_pred_dtree))
print("Recall:", recall_score(y_test, y_pred_dtree))
print("F1-score:", f1_score(y_test, y_pred_dtree))
print("AUC-ROC:", roc_auc_score(y_test, y_pred_dtree))
print()
# Random Forest
print("Random Forest:")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Precision:", precision_score(y_test, y_pred_rf))
print("Recall:", recall_score(y_test, y_pred_rf))
print("F1-score:", f1_score(y_test, y_pred_rf))
print("AUC-ROC:", roc_auc_score(y_test, y_pred_rf))
```

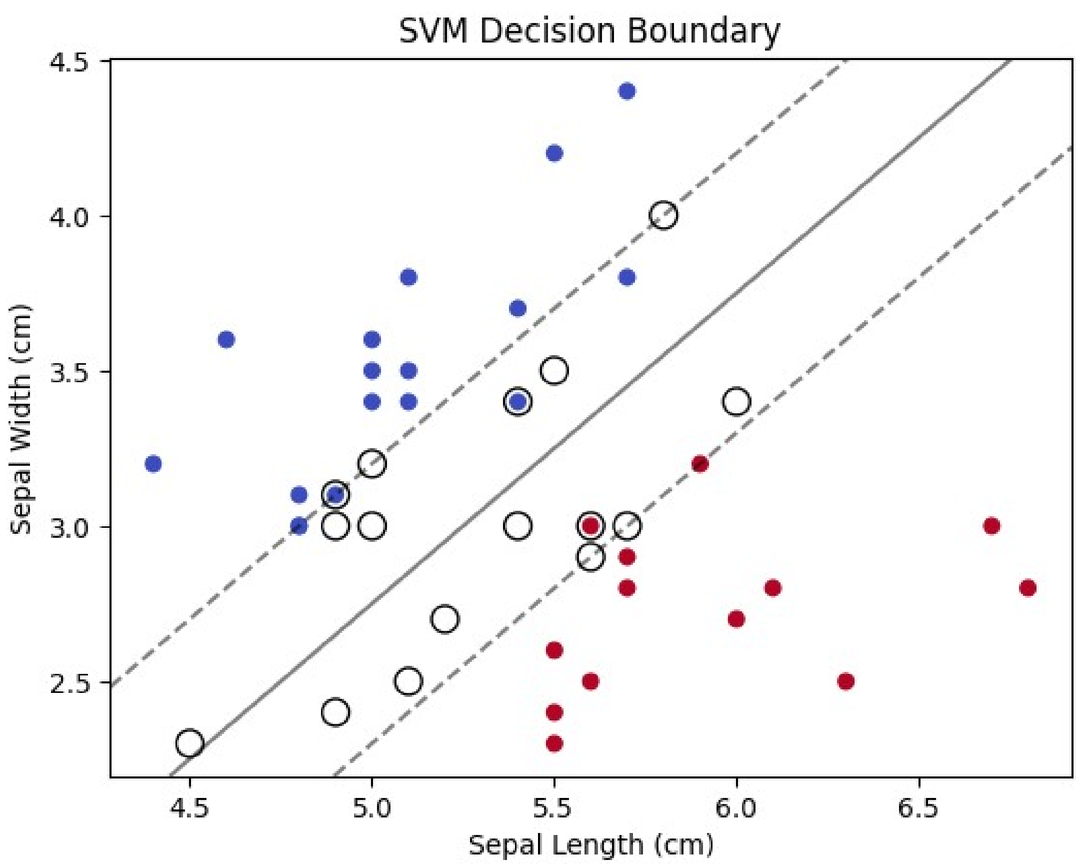
```
Logistic Regression:
Accuracy: 0.65
Precision: 0.0
Recall: 0.0
F1-score: 0.0
AUC-ROC: 0.4642857142857143

Decision Tree:
Accuracy: 0.55
Precision: 0.2
Recall: 0.16666666666666666
F1-score: 0.18181818181818182
AUC-ROC: 0.44047619047619047

Random Forest:
Accuracy: 0.5
Precision: 0.16666666666666666
Recall: 0.16666666666666666
```

```
F1-score: 0.16666666666666666
AUC-ROC: 0.40476190476190477
```

SUFYAAN EXP 12

```
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
# Load the Iris dataset and consider only two classes: setosa and
versicolor
iris = datasets.load_iris()
X = iris.data
y = iris.target
X = X[y != 2, :2] # Consider only setosa and versicolor classes
y = y[y != 2]
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
# Create and train the SVM classifier
svm_classifier = SVC(kernel='linear', C=1.0)
svm_classifier.fit(X_train, y_train)
# Predict on the test set
y_pred = svm_classifier.predict(X_test)
# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
# Create a confusion matrix to evaluate the performance
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
# Plot the decision boundary
# Function to plot the decision boundary
def plot_decision_boundary(clf, X, y, title):
 plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, s=30)
 ax = plt.gca()
 xlim = ax.get_xlim()
 ylim = ax.get_ylim()

 # Create grid to evaluate the model
 xx = np.linspace(xlim[0], xlim[1], 30)
 yy = np.linspace(ylim[0], ylim[1], 30)
 YY, XX = np.meshgrid(yy, xx)
 xy = np.vstack([XX.ravel(), YY.ravel()]).T
 Z = clf.decision_function(xy).reshape(XX.shape)
```

```
 # Plot decision boundary and margins
 ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
 linestyles=['--', '-', '--'])

 # Plot support vectors
 ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
s=100,
 linewidth=1, facecolors='none', edgecolors='k')

 plt.xlabel('Sepal Length (cm)')
 plt.ylabel('Sepal Width (cm)')
 plt.title(title)
 plt.show()
# Plot the decision boundary on the test data
plot_decision_boundary(svm_classifier, X_test, y_test, title='SVM
Decision Boundary')

Accuracy: 1.00
Confusion Matrix:
[[17  0]
 [ 0 13]]
```



SUFYAAN DATASET

```python
import pandas as pd
import numpy as np

# Function to generate synthetic diabetes dataset with 100 rows
def generate_synthetic_dataset(num_rows=100):
    np.random.seed(42)

    # Generate random values for each attribute based on some ranges
    age = np.random.randint(20, 70, num_rows)
    pregnancies = np.random.randint(0, 15, num_rows)
    glucose = np.random.randint(60, 200, num_rows)
    blood_pressure = np.random.randint(50, 100, num_rows)
    skin_thickness = np.random.randint(10, 50, num_rows)
    insulin = np.random.randint(50, 200, num_rows)
    bmi = np.random.uniform(20, 40, num_rows)
    diabetes_pedigree = np.random.uniform(0, 1, num_rows)

    # Create the target variable 'Diabetes'
    diabetes = np.random.choice([0, 1], num_rows)

    # Create the synthetic dataset
    dataset = pd.DataFrame({
        'Age': age,
        'Pregnancies': pregnancies,
        'Glucose': glucose,
        'BloodPressure': blood_pressure,
        'SkinThickness': skin_thickness,
        'Insulin': insulin,
        'BMI': bmi,
        'DiabetesPedigreeFunction': diabetes_pedigree,
        'Diabetes': diabetes
    })

    return dataset

# Generate the synthetic dataset with 100 rows
dataset = generate_synthetic_dataset(num_rows=100)

# Save the dataset to a CSV file
dataset.to_csv('synthetic_diabetes_data.csv', index=False)
```