VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning

Submitted by

PIYUSH DUBEY (1BM19CS221)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019

Apr-2022 to Aug-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Machine Learning**" carried out by **Piyush Dubey (1BM19CS221),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Machine Learning - (20CS6PCMAL)** work prescribed for the said degree.

Prof. Saritha A.N.	**Dr. Jyothi S Nayak**
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

`

Index Sheet

Course Outcomes:

CO1	Ability to **apply** the different learning algorithms.
CO2	Ability to **analyze** the learning techniques for given dataset.
CO3	Ability to **design** a model using machine learning to solve a problem.
CO4	Ability to **conduct** practical experiments to solve problems using appropriate machine learning techniques.

1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

In [11]:
```
import csv
```

In [12]:
```
def updateHypothesis(x, h):
    if h == []:
        return x

    for i in range(0, len(h)):
        if x[i].upper()!=h[i].upper():
            h[i] = '?'

    return h
```

In [13]:
```
if __name__ == "__main__":
    data = []
    h = []

    with open('data.csv', 'r') as file:
        reader = csv.reader(file)
        print("Data: ")
        for row in reader:
            data.append(row)
            print(row)

    if data:
        for x in data:
            if x[-1].upper() == "YES":
                x.pop() # removing last field
                h = updateHypothesis(x,h)

    print("\nHypothesis: ",h)
```

```
Data:
['sky', 'air temp', 'humidity', 'wind', 'water', 'forecast', 'enjoy sport']
['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']

Hypothesis:  ['sunny', 'warm', '?', 'strong', '?', '?']
```

In [ ]:

2. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
import numpy as np
import pandas as pd

data = pd.read_csv('data.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: "0, specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

```
❯ python candidateElimination.py
Concepts: [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
Target: ['yes' 'yes' 'no' 'yes']
Initialization of specific_h and general_h
        specific_h: ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
        general_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 Steps of Candidate Elimination Algorithm 1
        specific_h ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
        general_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 Steps of Candidate Elimination Algorithm 2
        specific_h ['sunny' 'warm' '?' 'strong' 'warm' 'same']
        general_h: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 Steps of Candidate Elimination Algorithm 3
        specific_h ['sunny' 'warm' '?' 'strong' 'warm' 'same']
        general_h: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]

 Steps of Candidate Elimination Algorithm 4
        specific_h ['sunny' 'warm' '?' 'strong' '?' '?']
        general_h: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

 Final specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']

 Final general_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

3. Write a program to demonstrate the working of the Decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv('id.csv')
features = [feat for feat in data]
features.pop()

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["Answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    gain = entropy(examples)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
    return gain

```

```python

def ID3(examples, attrs):
    root = Node()
    max_gain = 0
    max_feat = ""
    for feature in attrs:
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    uniq = np.unique(examples[max_feat])
    for u in uniq:
        subdata = examples[examples[max_feat] == u]
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["Answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

root = ID3(data, features)
printTree(root)
```

```
Outlook
        overcast -> ['yes']

        rain
                Wind
                        strong -> ['no']

                        weak -> ['yes']

        sunny
                Humidity
                        high -> ['no']

                        normal -> ['yes']
```

4. Write a program to implement the Naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

df = pd.read_csv("diabetes.csv")
feature_col_names = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age']
predicted_class_names = ['diabetes']

X = df[feature_col_names].values
y = df[predicted_class_names].values

print(df.head)
xtrain,xtest,ytrain,ytest=train_test_split(X,y,test_size=0.40)

print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)

clf = GaussianNB().fit(xtrain,ytrain.ravel())
predicted = clf.predict(xtest)
predictTestData= clf.predict([[6,148,72,35,0,33.6,0.627,50]])

print('\n Confusion matrix')
print(metrics.confusion_matrix(ytest,predicted))

print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))

print('\n The value of Precision', metrics.precision_score(ytest,predicted))

print('\n The value of Recall', metrics.recall_score(ytest,predicted))

print("Predicted Value for individual Test Data:", predictTestData)
```

```
<bound method NDFrame.head of      num_preg  glucose_conc  diastolic_bp  thickness  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
..        ...           ...           ...        ...      ...   ...
140         3           128            78          0        0  21.1
141         5           106            82         30        0  39.5
142         2           108            52         26       63  32.5
143        10           108            66          0        0  32.4
144         4           154            62         31      284  32.8

     diab_pred  age  diabetes
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
..         ...  ...       ...
140      0.268   55         0
141      0.286   38         0
142      0.318   22         0
143      0.272   42         1
144      0.237   23         0

[145 rows x 9 columns]>

 the total number of Training Data : (87, 1)

 the total number of Test Data : (58, 1)

 Confusion matrix
[[30 12]
 [ 8  8]]

 Accuracy of the classifier is 0.6551724137931034

 The value of Precision 0.4

 The value of Recall 0.5
Predicted Value for individual Test Data: [1]
```

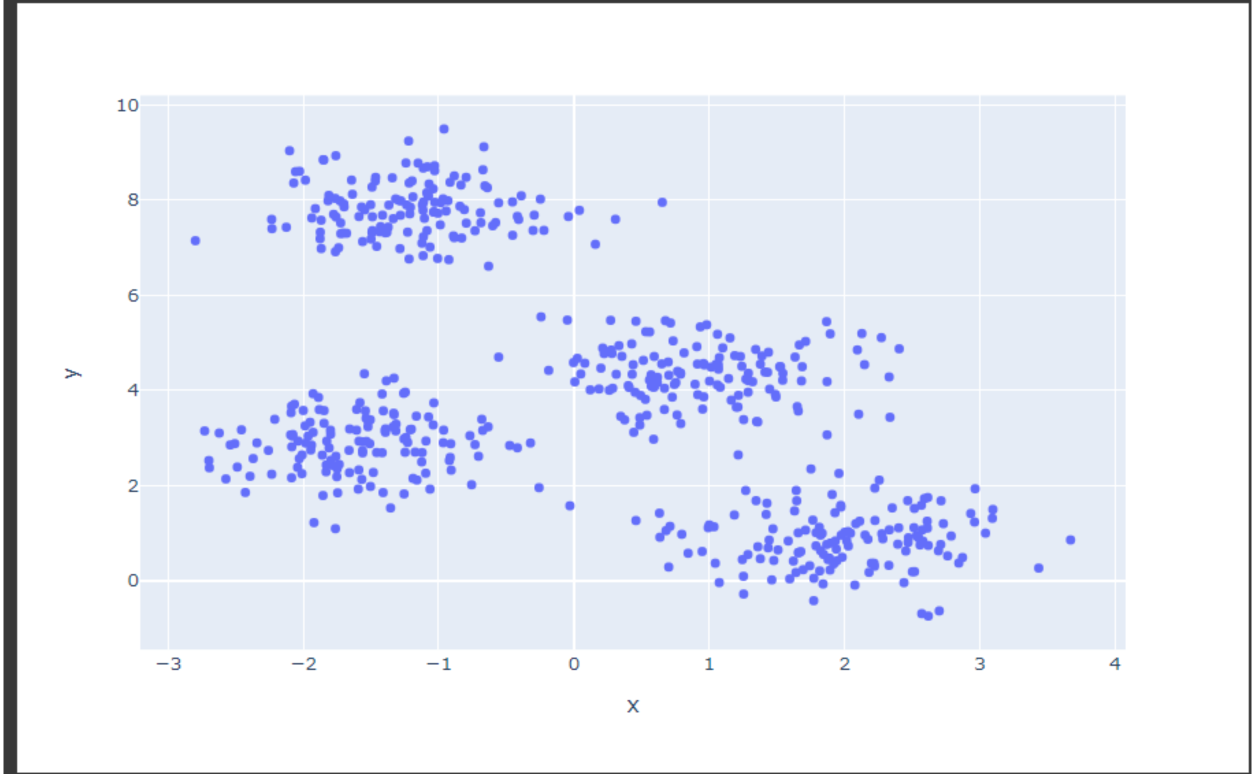
5. Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

```python

get_ipython().system('pip install pgmpy')


from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
import numpy as np

#bayesNet = BayesianModel([("M", "R"),("U", "R"),("B", "R"),("R", "S")])
bayesNet.add_node("M")
bayesNet.add_node("U")
bayesNet.add_node("R")
bayesNet.add_node("B")
bayesNet.add_node("S")

bayesNet.add_edge("M", "R")
bayesNet.add_edge("U", "R")
bayesNet.add_edge("B", "R")
bayesNet.add_edge("B", "S")
bayesNet.add_edge("R", "S")

cpd_A = TabularCPD('M', 2, values=[[.95], [.05]])
cpd_U = TabularCPD('U', 2, values=[[.85], [.15]])
cpd_H = TabularCPD('B', 2, values=[[.90], [.10]])

cpd_S = TabularCPD('S', 2, values=[[0.98, .88, .95, .6], [.02, .12, .05, .40]],
                   evidence=['R', 'B'], evidence_card=[2, 2])

cpd_R = TabularCPD('R', 2,
                   values=[[0.96, .86, .94, .82, .24, .15, .10, .05], [.04, .14, .06, .18, .76, .85, .90, .95]],
                   evidence=['M', 'B', 'U'], evidence_card=[2, 2,2])
bayesNet.add_cpds(cpd_A, cpd_U, cpd_H, cpd_S, cpd_R)

bayesNet.check_model()
print("Model is correct.")
```

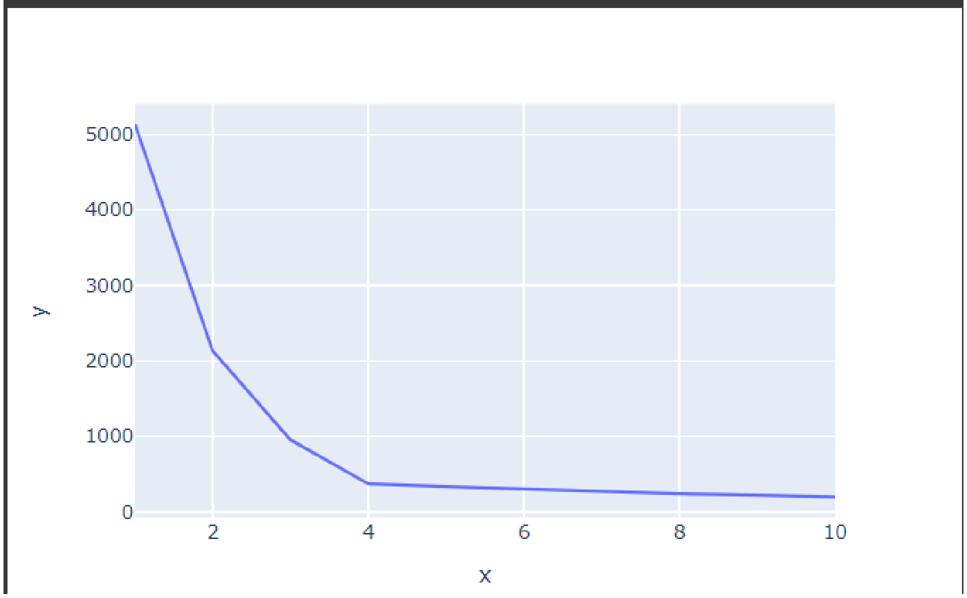
```
Model is correct.
/usr/local/lib/python3.7/dist-packages/pgmpy/models/BayesianModel.py:10: FutureWarning: BayesianModel has been renamed to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in
  FutureWarning,
Finding Elimination Order: : 0%  0/3 [00:00<?, ?it/s]
Eliminating: M: 100%  3/3 [00:00<00:00, 43.94it/s]
R +------+----------+
| R    |   phi(R) |
+======+==========+
| R(0) |   0.9062 |
+------+----------+
| R(1) |   0.0938 |
+------+----------+
Finding Elimination Order: : 100%  2/2 [00:00<00:00, 18.53it/s]
Eliminating: B: 100%  2/2 [00:00<00:00, 32.17it/s]
R| M +------+----------+
| R    |   phi(R) |
+======+==========+
| R(0) |   0.2131 |
+------+----------+
| R(1) |   0.7869 |
+------+----------+
Finding Elimination Order: : 100%  3/3 [00:00<00:00, 17.63it/s]
Eliminating: M: 100%  3/3 [00:00<00:00, 43.05it/s]
S| B +------+----------+
| S    |   phi(S) |
+======+==========+
| S(0) |   0.8465 |
+------+----------+
| S(1) |   0.1535 |
+------+----------+
<TabularCPD representing P(R:2 | M:2, B:2, U:2) at 0x7f578998c6d0>
```

6. Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

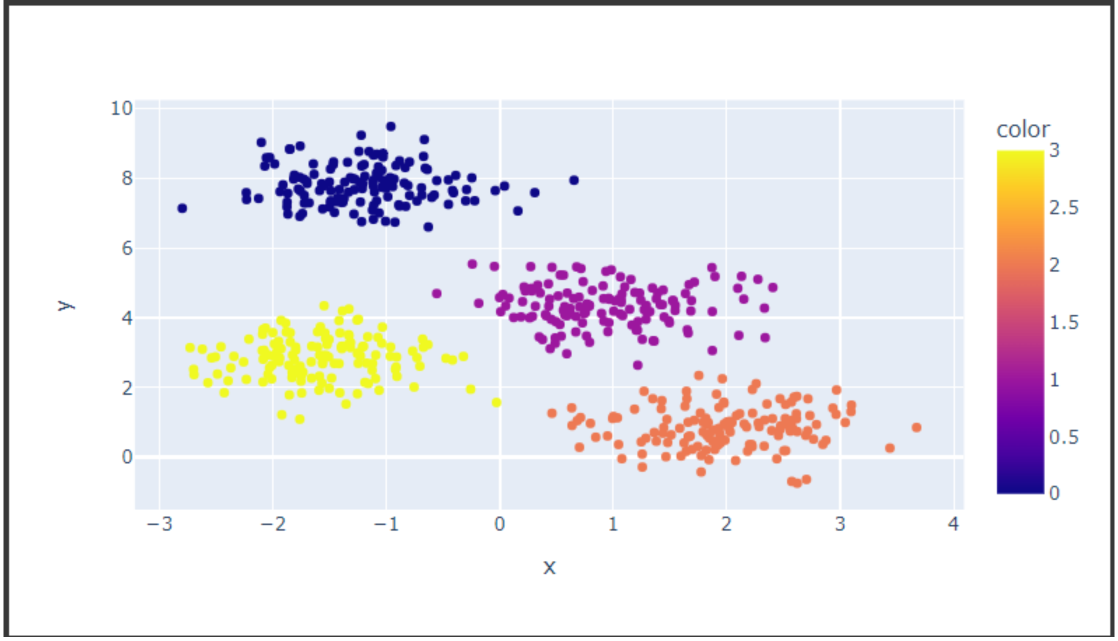
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=550, centers=4, cluster_std=0.60, random_state=0)
import plotly.express as px
fig = px.scatter(x =X[:, 0], y =X[:, 1],width=800,height=500)
fig.show()
```



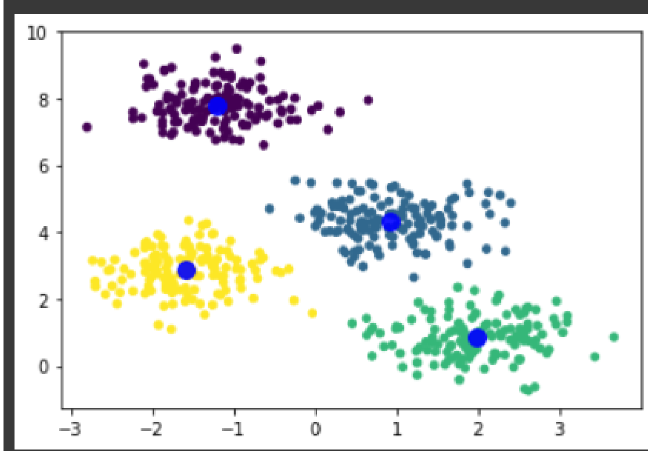
```
cost =[]
for i in range(1, 11):
  KM = KMeans(n_clusters = i, max_iter = 500)
  KM.fit(X)

  cost.append(KM.inertia_)

# plot the cost against K values
fig = px.line(x=range(1, 11), y=cost, width=600, height=400)
fig.show()
```



```
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
fig = px.scatter(x =X[:, 0], y = X[:, 1], color=y_kmeans, width=700,height=400)
trace = px.scatter(x =X[:, 0], y = X[:, 1],  width=700,height=400)
fig.show()
```




```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=20)
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=100, alpha=0.9);
plt.show()
```



```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()
df = pd.DataFrame(iris.data)
df['class']=iris.target
df.columns=['sepal_len', 'sepal_wid', 'petal_len', 'petal_wid', 'class']
df.info()
```

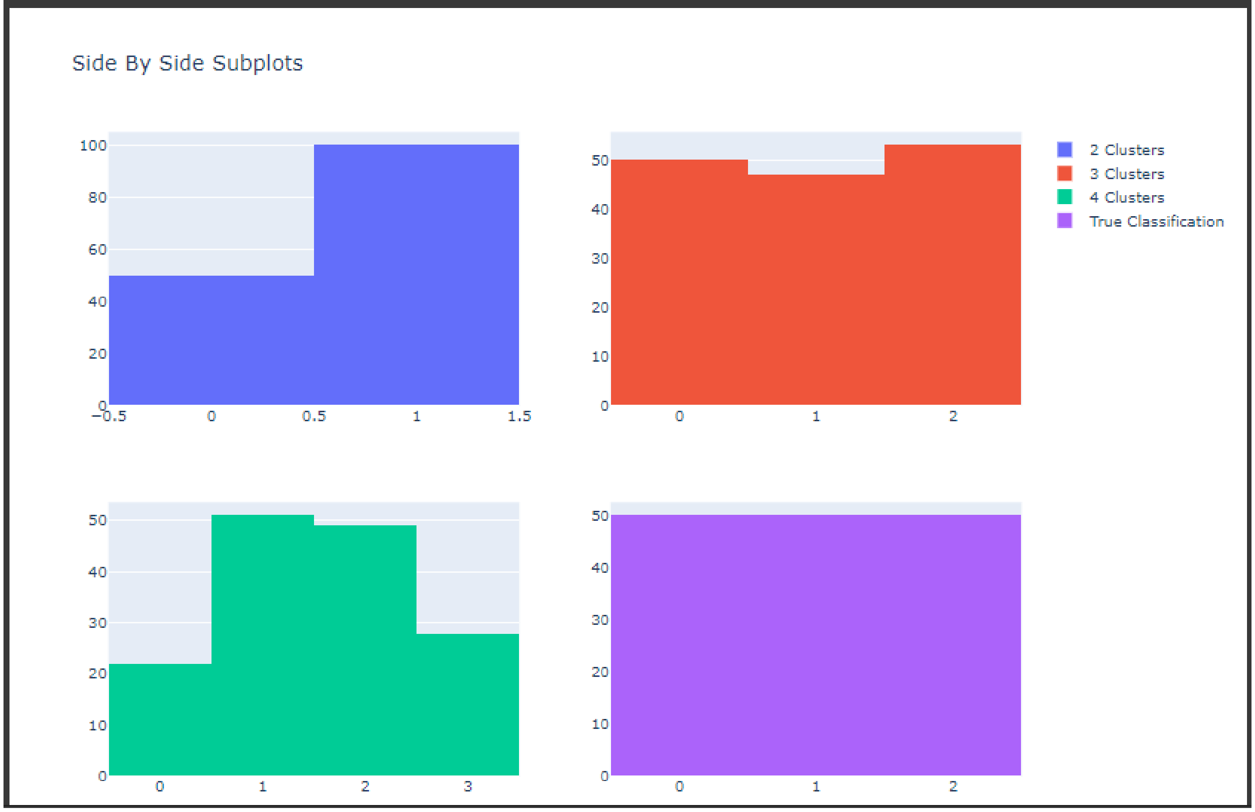
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   sepal_len  150 non-null    float64
 1   sepal_wid  150 non-null    float64
 2   petal_len  150 non-null    float64
 3   petal_wid  150 non-null    float64
 4   class      150 non-null    int64
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

```python
px.histogram(df, x ='class', color='class')
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = df.iloc[:,0:4].values
scaled_x = scaler.fit_transform(X)
model = KMeans(n_clusters=3,init='k-means++',random_state=0)
labels = model.fit_predict(scaled_x)
```


```python
px.histogram(df, x ='class', color='class')
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = df.iloc[:,0:4].values
scaled_x = scaler.fit_transform(X)
model = KMeans(n_clusters=3,init='k-means++',random_state=0)
labels = model.fit_predict(scaled_x)
```


```python
import plotly.graph_objects as go
fig = go.Figure()

 # Add trace
fig.add_trace(go.Histogram(x=labels,name="Predicted Labels"))
fig.add_trace(go.Histogram(x=df['class'],name="True Labels"))

# Overlay both histograms
fig.update_layout(barmode='overlay')
# Reduce opacity to see both histograms
fig.update_traces(opacity=0.75)
fig.show()
```

```
labels =[]
for i in range(2, 5):
  model = KMeans(n_clusters = i, max_iter = 500)
  model.fit(scaled_x)
  labels.append(model.fit_predict(scaled_x))
from plotly.subplots import make_subplots
import plotly.graph_objects as go
fig = make_subplots(rows=2, cols=2)
for i in range(0, 3):
    fig.add_trace(go.Histogram(x=labels[i],name="{} Clusters".format(i+2)),
              row=(i//2 + 1), col=(i%2 + 1))
fig.add_trace(go.Histogram(x=df['class'],name="True Classification"),
              row=(2), col=(2))
fig.update_layout(height=700, width=1000, title_text="Side By Side Subplots")

fig.show()
```



7. Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

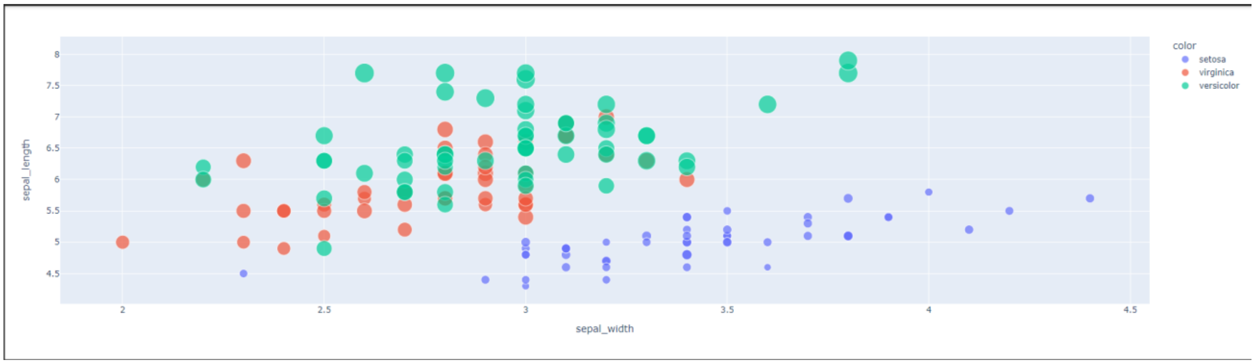
```
import pandas as pd
import plotly.express as px
from sklearn.metrics import confusion_matrix
df = px.data.iris()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
 5   species_id    150 non-null    int64  
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
from sklearn.mixture import GaussianMixture
from sklearn.model_selection import train_test_split as tts
X = df.iloc[:,0:4].values
gm = GaussianMixture(n_components=3, random_state=0).fit(X)
gm.means_
```

```
array([[5.006     , 3.418     , 1.464     , 0.244     ],
       [6.54639415, 2.94946365, 5.48364578, 1.98726565],
       [5.9170732 , 2.77804839, 4.20540364, 1.29848217]])
```

```
def feature(x):
    species = ['setosa','versicolor','virginica']
    return species[x]

pred = gm.predict(X)
pred_features = list(map(feature,pred))
fig1 = px.scatter(df, x="sepal_width", y="sepal_length", color=pred_features,
                 size='petal_length', hover_data=['petal_width'])
fig1.show()
```



8. Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

In [27]:
```
import numpy as np
import matplotlib.pyplot as mtp
import pandas as pd
```

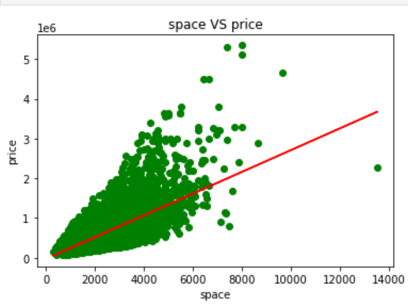
In [28]:
```
dataset = pd.read_csv("kc_house_data.csv")
```

In [29]:
```
space=dataset['sqft_living']
price=dataset['price']

x = np.array(space).reshape(-1, 1)
y = np.array(price)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 1/3, random_state=0)
```

In [30]:
```
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
```

Out[30]: LinearRegression()

In [31]:
```
y_pred= regressor.predict(x_test)
x_pred= regressor.predict(x_train)
mtp.scatter(x_train, y_train, color="green")
mtp.plot(x_train, x_pred, color="red")
mtp.title("space VS price")
mtp.xlabel("space")
mtp.ylabel("price")
mtp.show()
```



9. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split as tts
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv("https://raw.githubusercontent.com/Derek-Stanley/6A_ML/main/LAB%208/iris.csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
X = df.iloc[:,[1,2,3,4]].values
y = df.iloc[:,5].values
X_train, X_test, y_train, y_test = tts(X,y,test_size=0.3)
```

```
import math,numpy as np
math.sqrt(len(df))
```

```
12.24744871391589
```

```
model = KNeighborsClassifier(n_neighbors = 13, metric = 'euclidean')
model.fit(X_train,y_train)
```

```
KNeighborsClassifier(metric='euclidean', n_neighbors=13)
```

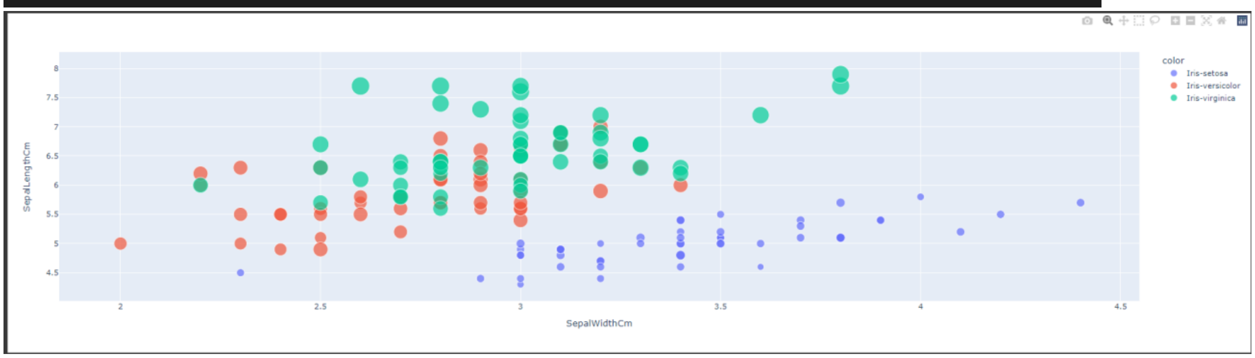
```
y_pred = model.predict(X_test)
```

```python
y_pred = model.predict(X_test)
```

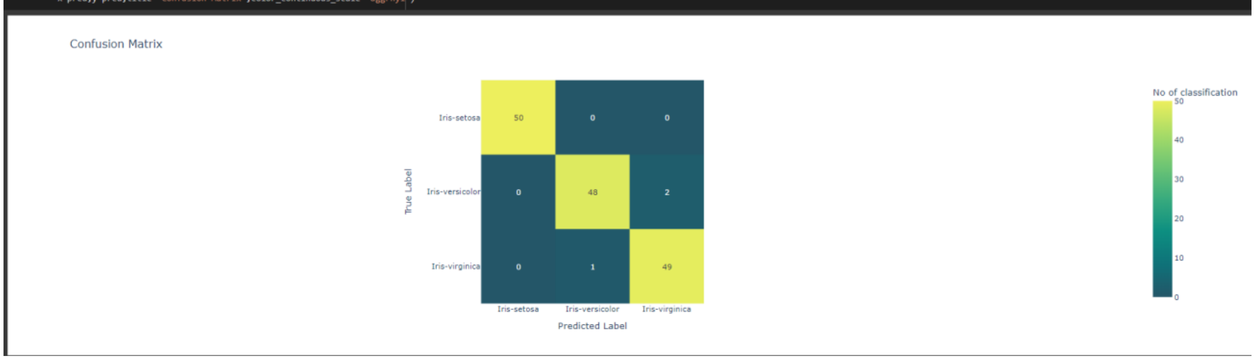
```python
from sklearn.metrics import classification_report, confusion_matrix
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

```
Accuracy Metrics
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        15
Iris-versicolor       1.00      1.00      1.00        16
 Iris-virginica       1.00      1.00      1.00        14

       accuracy                           1.00        45
      macro avg       1.00      1.00      1.00        45
   weighted avg       1.00      1.00      1.00        45
```

```python
import plotly.express as px
pred = model.predict(X)
fig1 = px.scatter(df, x="SepalWidthCm", y="SepalLengthCm", color=pred,
                  size='PetalLengthCm', hover_data=['PetalWidthCm'])
fig1.show()
```




```python
cm = confusion_matrix(df['Species'], pred, labels=pred)
px.imshow(cm,text_auto=True,labels=dict(x="Predicted Label", y="True Label", color="No of classification"),
        x=pred,y=pred,title="Confusion Matrix",color_continuous_scale="aggrnyl")
```



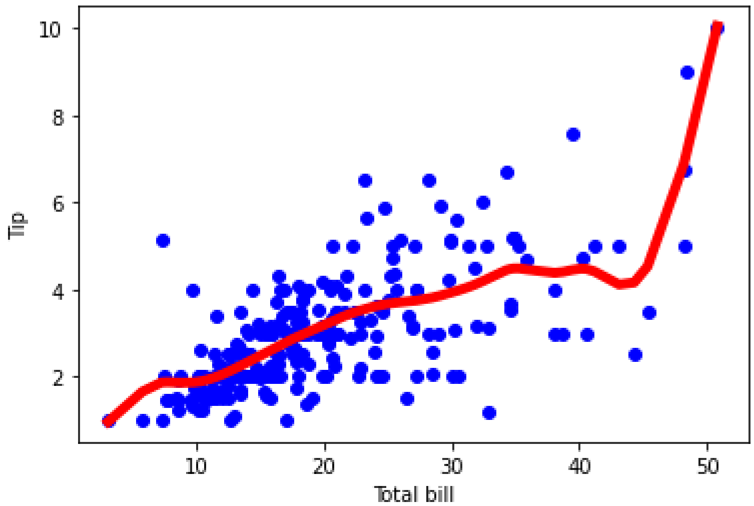
10. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
from numpy import *
from os import listdir
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np1
import numpy.linalg as np
import seaborn as sn
from scipy.stats.stats import pearsonr
```

```
data = sn.load_dataset('tips')
```

```
def kernel(point,xmat, k):
    m,n = np1.shape(xmat)
    weights = np1.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))
    return weights


def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

def localWeightRegression(xmat,ymat,k):
    m,n = np1.shape(xmat)
    ypred = np1.zeros(m)

    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred
```

```
#load data points

bill = np1.array(data.total_bill)
tip = np1.array(data.tip)
#preparing and add 1 in bill
mbill = np1.mat(bill)
mtip = np1.mat(tip)
# mat is used to convert to n dimesiona to 2 dimensional array form
m= np1.shape(mbill)[1] # print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
# print(X)
#set k here
ypred = localWeightRegression(X,mtip,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='blue')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```



10
8
6
4
2
Tip
10
20
30
40
50
Total bill