Experiment 1:

[Simple Regression]

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

✓ 9.1s

```
np.random.seed(0)
X = 2.5 * np.random.randn(10000) + 1.5
res = 0.5 * np.random.randn(10000)
y = 2 + 0.3 * X + res
```

✓ 0.8s

```
df = pd.DataFrame(
    {'X': X,
     'y': y}
)
df.head()
```

✓ 0.1s

	X	y
0	5.910131	3.671981
1	2.500393	2.333502
2	3.946845	4.050854
3	7.102233	4.225994
4	6.168895	3.761763

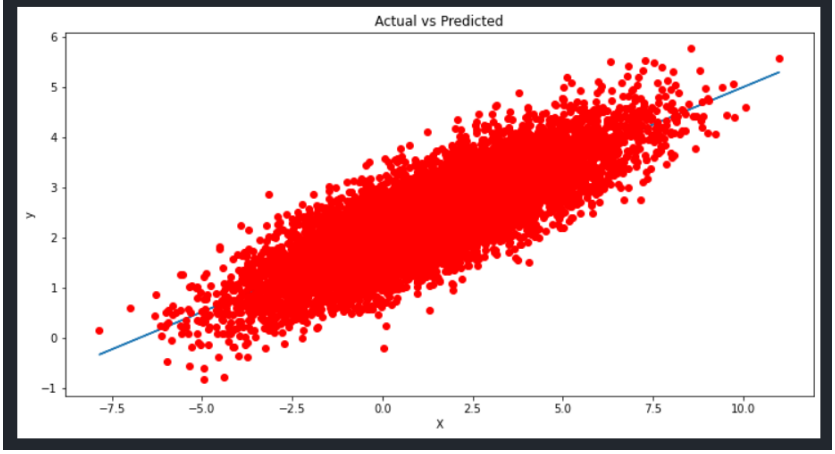
```
xmean = np.mean(X)x```
ymean = np.mean(y)
df['xycov'] = (df['X'] - xmean) * (df['y'] - ymean)
df['xvar'] = (df['X'] - xmean)**2
beta = df['xycov'].sum() / df['xvar'].sum()
alpha = ymean - (beta * xmean)
print(f'alpha = {alpha}')
print(f'beta = {beta}')
```

✓ 0.1s

```
alpha = 2.0077628313059805
beta = 0.29843950486941884
```

```
ypred = alpha + beta * X
plt.figure(figsize=(12, 6))
plt.plot(X, ypred)
plt.plot(X, y, 'ro')
plt.title('Actual vs Predicted')
plt.xlabel('X')
plt.ylabel('y')
plt.show()
```

✓ 0.3s



Experiment 2:

[Multiple Regression]

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

✓ 15.5s

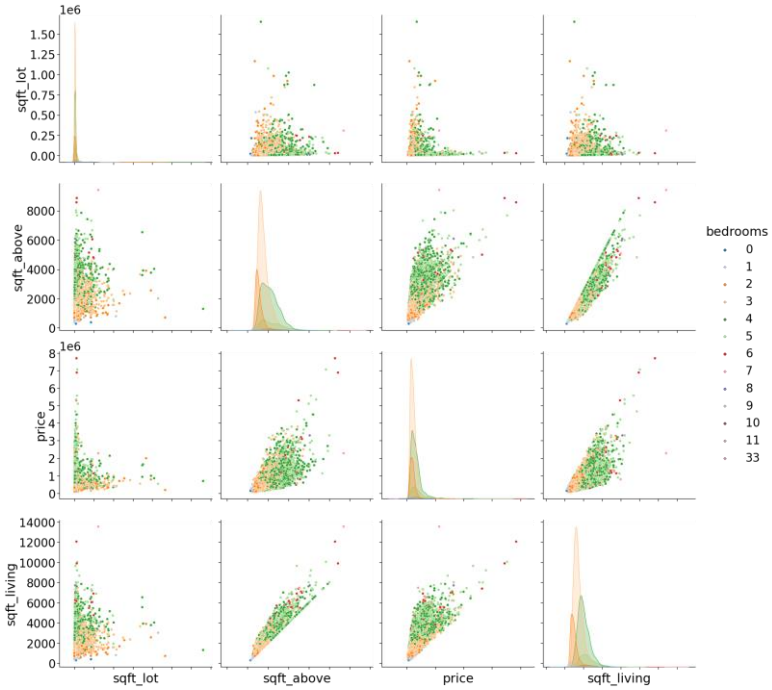
```
dataset = pd.read_csv('Datasets/kc_house_data.csv')
dataset.head()
print(dataset.dtypes)
print(dataset.isnull().any())
```

✓ 0.3s

```
Output exceeds the size limit. Open the full output data in a text editor
id                int64
date             object
price           float64
bedrooms          int64
bathrooms       float64
sqft_living       int64
sqft_lot          int64
floors          float64
waterfront        int64
view              int64
condition         int64
```

```
dataset = dataset.drop(['id','date'], axis = 1)
with sns.plotting_context("notebook",font_scale=2.5):
    g = sns.pairplot(dataset[['sqft_lot','sqft_above','price','sqft_living','bedrooms']],
    hue='bedrooms', palette='tab20',size=6)
g.set(xticklabels=[]);
```

✓ 27.9s



```
X = dataset.iloc[:,1:].values
y = dataset.iloc[:,0].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, random_state = 0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
print(y_pred)
```

✓ 0.1s

```
[ 386540.99847832 1516969.01534072  538662.72575238 ...  526000.75505743
  313924.63663322  400525.6731456 ]
```

Experiment 3:

[Problems on Anova, plotting the table and solving it using Python]

```
import pandas
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
```

✓ 0.6s

```
df = pandas.read_excel('Datasets/MLR-House Prize.xlsx')
df = df.sample(5)
df.head()
```

✓ 0.8s

	Unnamed: 0	X1	X2	X3	Y
13	13	2994	20.4	3	618.7
20	20	2550	20.2	3	606.6
21	21	3380	19.6	4.5	758.9
24	24	2710	21.6	3	622.4
7	7	2978	17.3	3	643.3

```
x1 = df['X1'].mean()
x2 = df['X2'].mean()
x3 = df['X3'].mean()
```

✓ 0.5s

```
total_mean = (x1 + x2 + x3)/3
```

✓ 0.1s

```
def regression_ssr(df):
    regression_ssr = 5*(x1 - total_mean)**2 + 5*(x2 - total_mean)**2 + 5*(x3 - total_mean)**2
    return regression_ssr
```

✓ 0.5s

```
def error_sse(df):
    g1 = sum((df['X1'] - total_mean)**2)
    g2 = sum((df['X2'] - total_mean)**2)
    g3 = sum((df['X3'] - total_mean)**2)
    return g1 + g2 + g3
```

✓ 0.8s

```python
def total_sst(df):
    total_sst = regression_ssr(df) + error_sse(df)

    return total_sst
```

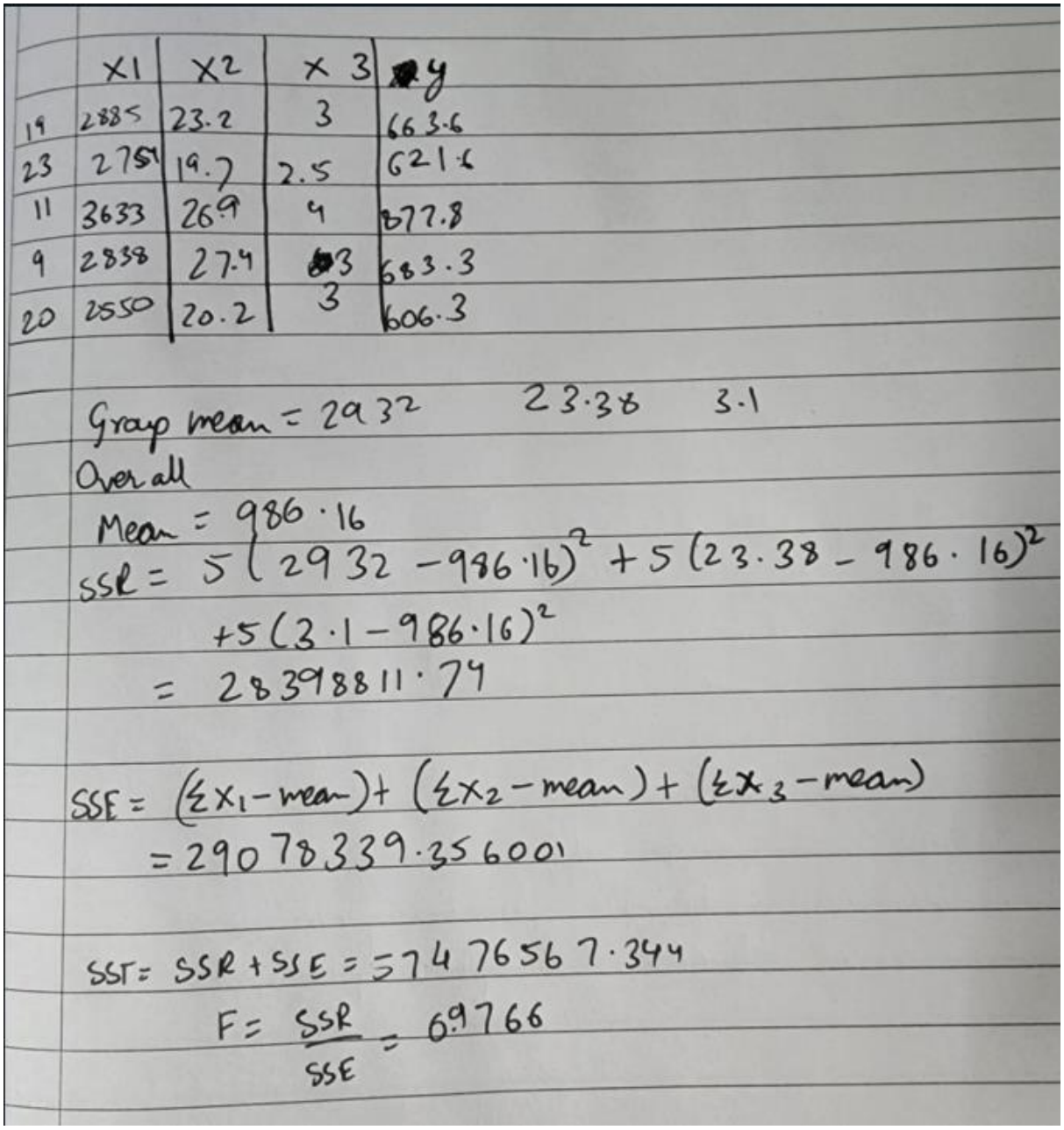
✓ 0.1s

```python
def main():
    print('''
    Regression Sum of Squares: \n{}
    Error Sum of Squares: \n{}
    Total Sum of Squares: \n{}
    '''.format(regression_ssr(df), error_sse(df), total_sst(df)))

    p_value = stats.f.sf(regression_ssr(df)/3, 3, 12)
    print('''P value: \n{}'''.format(p_value))

    f_value = regression_ssr(df)/error_sse(df)
    print('''F value: \n{}'''.format(f_value))

    r_squared = regression_ssr(df)/total_sst(df)
    print('''R squared: \n{}'''.format(r_squared))

    adjusted_r_squared = 1 - (1 - r_squared)*(12/9)
    print('''Adjusted R squared: \n{}'''.format(adjusted_r_squared))

main()
```

✓ 0.1s

```
 Regression Sum of Squares:
28243980.628
 Error Sum of Squares:
28645403.675999995
 Total Sum of Squares:
56889384.30399999

P value:
1.724988626547702e-38
F value:
0.9859864761362633
R squared:
0.4964718984665495
Adjusted R squared:
0.32862919795539935
```

Manual Calculations:



	X1	X2	X 3	y
19	2885	23.2	3	663.6
23	2751	19.7	2.5	621.6
11	3633	26.9	4	877.8
9	2838	27.4	3	683.3
20	2550	20.2	3	606.3

Grap mean = 2932 23.38 3.1

Overall

Mean = 986.16

$SSR = 5(2932 - 986.16)^2 + 5(23.38 - 986.16)^2 + 5(3.1 - 986.16)^2$

$= 28398811.74$

$SSE = (\Sigma X_1 - mean) + (\Sigma X_2 - mean) + (\Sigma X_3 - mean)$

$= 29078339.356001$

$SST = SSR + SSE = 74765671.344$

$F = \frac{SSR}{SSE} = 0.9766$

Experiment 4:

[Naïve Bayes Classification]

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import CategoricalNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

✓ 0.7s

```
df = pd.read_csv('Datasets/naivebayes.csv')
df.head()
df.describe()
df.info()
```

✓ 0.1s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Day               14 non-null     object
 1   Outlook           14 non-null     object
 2   Temperature       14 non-null     object
 3   Humidity          14 non-null     object
 4   Wind              14 non-null     bool  
 5   Class: Play ball  14 non-null     object
dtypes: bool(1), object(5)
memory usage: 702.0+ bytes
```

```
labels = df['Class: Play ball'].unique()
```

✓ 0.6s

```
def entropy(data):
    entropy_val = 0
    try:
        for i in labels:
            p_label = len(data[data['Class: Play ball'] == i])/len(data)
            entropy_val=entropy_val - p_label*math.log2(p_label)
        return entropy_val
    except:
        return 0
```

✓ 0.9s

```
entropy_parent = entropy(df)
groups = df.groupby('Outlook')
categories = df['Outlook'].unique()
```

✓ ✓ 0.1s

```
info_gain = 0
for i in categories:
    data = groups.get_group(i)
    entropy_child = entropy(data)
    probability_child = len(data) / len(df)
    info_gain = info_gain - probability_child * entropy_child
print('information gain on outlook column = ', entropy_parent+info_gain)
groups = df.groupby('Temperature')
categories = df['Temperature'].unique()
```

✓ 0.1s

```
information gain on outlook column =  0.7084922088251645
```

```
    info_gain = 0
    for i in categories:
     data = groups.get_group(i)
     entropy_child = entropy(data)
     probability_child = len(data) / len(df)
     info_gain = info_gain - probability_child * entropy_child
    print('information gain on humidity column =', entropy_parent + info_gain)
    groups = df.groupby('Wind')
    categories = df['Wind'].unique()
    info_gain = 0
    for i in categories:
     data = groups.get_group(i)
     entropy_child = entropy(data)
     probability_child = len(data) / len(df)
     info_gain = info_gain-probability_child * entropy_child
    print('information gain on wind column =',entropy_parent + info_gain)
```

✓ 0.1s

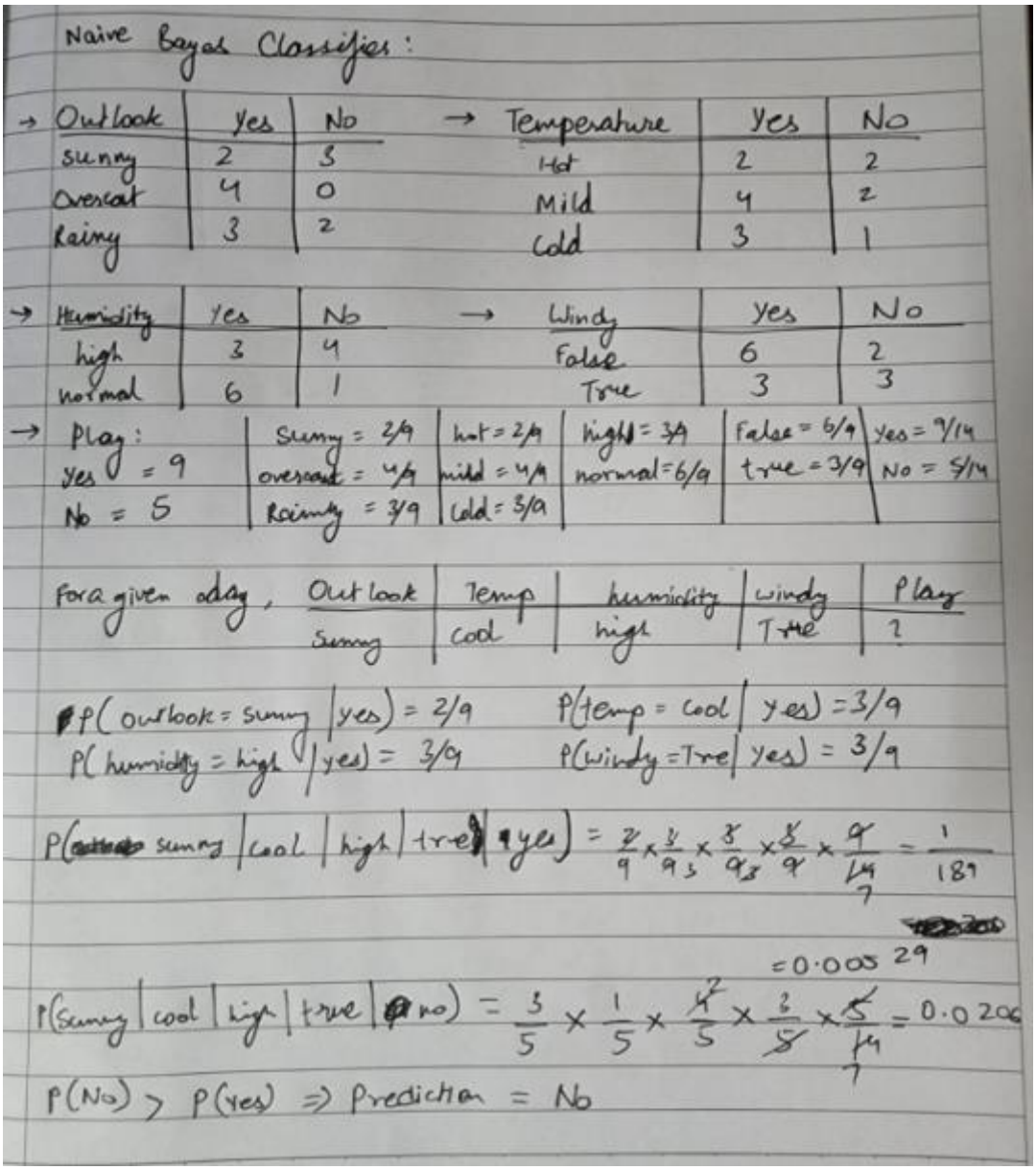
```
information gain on humidity column = 0.6617049208213365
information gain on wind column = 0.5727683138330127
```

```
    #naive bayes classification
    from array import array


    df['Outlook']=df['Outlook'].astype('category')
    df['Temperature']=df['Temperature'].astype('category')
    df['Humidity']=df['Humidity'].astype('category')
    df['Wind']=df['Wind'].astype('category')
    df['Class: Play ball']=df['Class: Play ball'].astype('category')

    df['Outlook']=df['Outlook'].cat.codes
    df['Temperature']=df['Temperature'].cat.codes
    df['Humidity']=df['Humidity'].cat.codes
    df['Wind']=df['Wind'].cat.codes
    df['Class: Play ball']=df['Class: Play ball'].cat.codes

    X=df[['Outlook','Temperature','Humidity','Wind']]
    y=df['Class: Play ball']

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

    clf = CategoricalNB()
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print('accuracy=',accuracy_score(y_test, y_pred))
    print('confusion matrix=',confusion_matrix(y_test, y_pred))

    #new day - sunny, cool, high, true
    new_day=np.array([2,1,0,0])
    array_new_day=new_day.reshape(1,-1)

    print('Prediction for new day=',clf.predict(array_new_day), '\n2 means no, 1 means yes')
```

✓ 0.1s

```
accuracy= 0.6666666666666666
confusion matrix= [[0 1]
 [0 2]]
prediction for new day= [2]
2 means no, 1 means yes
```

Manual Calculations:



Naive Bayes Classifier:

Outlook	yes	No
sunny	2	3
Overcast	4	0
Rainy	3	2

Temperature	yes	No
Hot	2	2
Mild	4	2
Cold	3	1

Humidity	yes	No
high	3	4
normal	6	1

Windy	yes	No
False	6	2
True	3	3

Play: yes = 9, No = 5

sunny = 2/9, overcast = 4/9, Rainy = 3/9

hot = 2/9, mild = 4/9, cold = 3/9

high = 3/9, normal = 6/9

False = 6/9, true = 3/9

yes = 9/14, No = 5/14

For a given day,

Outlook	Temp	humidity	windy	Play
Sunny	cool	high	True	?

P(outlook = sunny | yes) = 2/9

P(temp = cool | yes) = 3/9

P(humidity = high | yes) = 3/9

P(windy = True | yes) = 3/9

$$P(\text{sunny} \mid \text{cool} \mid \text{high} \mid \text{true} \mid \text{yes}) = \frac{2}{9} \times \frac{3}{9_3} \times \frac{3}{9_3} \times \frac{3}{9} \times \frac{9}{14_7} = \frac{1}{189}$$

$$= 0.00529$$

$$P(\text{sunny} \mid \text{cool} \mid \text{high} \mid \text{true} \mid \text{no}) = \frac{3}{5} \times \frac{1}{5} \times \frac{4^2}{5} \times \frac{3}{5} \times \frac{5}{14_7} = 0.0206$$

P(No) > P(Yes) ⇒ Prediction = No

Experiment 5:

[ID3 and Gini Index]

```
import pandas as pd
import numpy as np
import math
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

✓ 3.4s

```
df = pd.read_csv(
    'Datasets/Play Tennis.csv')
df.head()
```

✓ 0.1s

	Day	Outlook	Temperature	Humidity	Wind	play
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes

```
def entropy(target_col):
    elements, counts = np.unique(target_col, return_counts=True)
    entropy = np.sum([(-counts[i] / np.sum(counts)) * np.log2(counts[i] / np.sum(counts)) for i in range(len(elements))])
    return entropy
```

✓ 0.8s

```
def InfoGain(data, split_attribute_name, target_name="play"):
    total_entropy = entropy(data[target_name])
    vals, counts = np.unique(data[split_attribute_name], return_counts=True)
    Weighted_Entropy = np.sum([(counts[i] / np.sum(counts)) * entropy(data.where(data[split_attribute_name] == vals[i]).dropna()[target_name])
        for i in range(len(vals))])
    Information_Gain = total_entropy - Weighted_Entropy
    return Information_Gain
```

✓ 0.1s

```
def ID3(data, originaldata, features, target_attribute_name="play", parent_node_class=None):
    if len(np.unique(data[target_attribute_name])) <= 1:
        return np.unique(data[target_attribute_name])[0]
    elif len(data) == 0:
        return np.unique(originaldata[target_attribute_name])[np.argmax(np.unique(originaldata[target_attribute_name], return_counts=True)[1])]
    elif len(features) == 0:
        return parent_node_class
    else:
        parent_node_class = np.unique(data[target_attribute_name])[np.argmax(np.unique(data[target_attribute_name], return_counts=True)[1])]
        item_values = [InfoGain(data, feature, target_attribute_name) for feature in features]
    best_feature_index = np.argmax(item_values)
    best_feature = features[best_feature_index]
    tree = {best_feature: {}}
    features = [i for i in features if i !=best_feature]
    for value in np.unique(data[best_feature]):
        value = value
        sub_data = data.where(data[best_feature] == value).dropna()
    subtree = ID3(sub_data, data, features, target_attribute_name, parent_node_class)
    tree[best_feature][value] = subtree
    return tree
```

✓ 0.5s

```
def gini(x):
    elements, counts = np.unique(x, return_counts=True)
    gini = np.sum([(-counts[i] / np.sum(counts)) * np.log2(counts[i] / np.sum(counts)) for i in range(len(elements))])
    return gini
```

✓ 0.8s

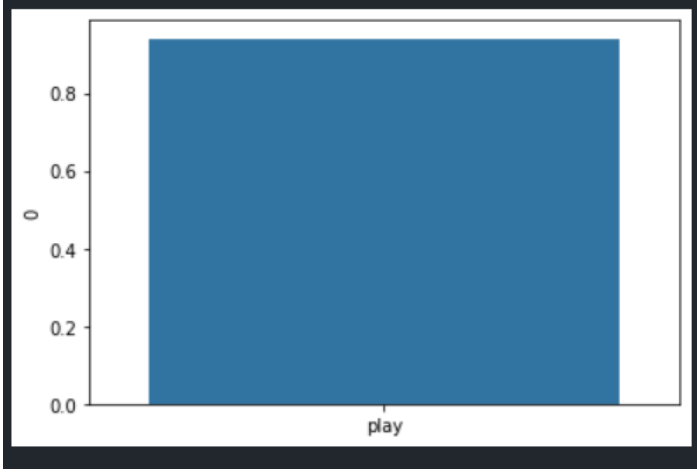
```python
def giniIndex(data, target_name="play"):
    total_gini = gini(data[target_name])
    vals, counts = np.unique(data[target_name], return_counts=True)
    Weighted_gini = np.sum([(counts[i] / np.sum(counts)) *
gini(data.where(data[target_name] == vals[i]).dropna()[target_name])
for i in range(len(vals))])
    Information_gini = total_gini - Weighted_gini
    return Information_gini
```

✓ 0.5s

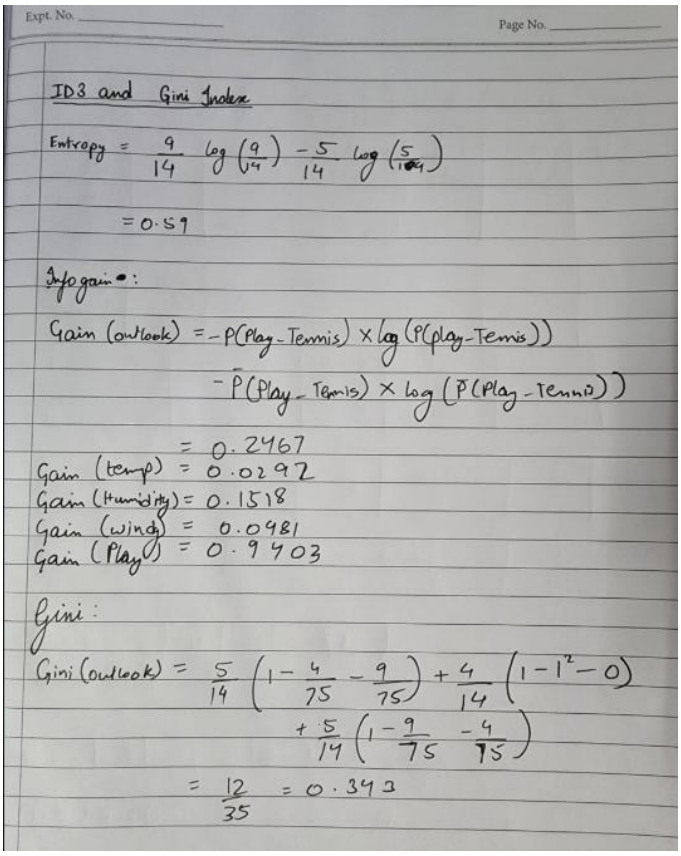
```python
def main():
    data = df
    originaldata = df
    features = np.array(data.columns)[:-1]
    tree = ID3(data, originaldata, features)
    #info gain
    print(InfoGain(data, 'Outlook', 'play'))
    print(InfoGain(data, 'Temperature', 'play'))
    print(InfoGain(data, 'Humidity', 'play'))
    print(InfoGain(data, 'Wind', 'play'))
    #entropy
    print(entropy(data['play']))
    #print gini as a table
    table = pd.DataFrame(columns=['play'])
    table.loc[0] = [giniIndex(data, 'play')]
    print(table)
    #print gini as a graph
    sns.barplot(x=table.columns, y=table.loc[0])
    plt.show()
    #print tree
    print()
    print(tree)
main()
```

✓ 0.5s

```
0.24674981977443933
0.02922256565895487
0.15183550136234159
0.04812703040826949
0.9402859586706311
       play
0  0.940286          {'Day': {'D9': 'Yes'}}
```



Manual Calculations:



ID3 and Gini Index

Entropy = $\frac{9}{14}\log\left(\frac{9}{14}\right) - \frac{5}{14}\log\left(\frac{5}{14}\right)$

$= 0.59$

Info gain:

Gain (outlook) $= -P(\text{Play\_Tennis}) \times \log(P(\text{play\_Tenis})) - \bar{P}(\text{Play\_Tenis}) \times \log(\bar{P}(\text{Play\_Tennis}))$

$= 0.2467$

Gain (temp) $= 0.0292$

Gain (Humidity) $= 0.1518$

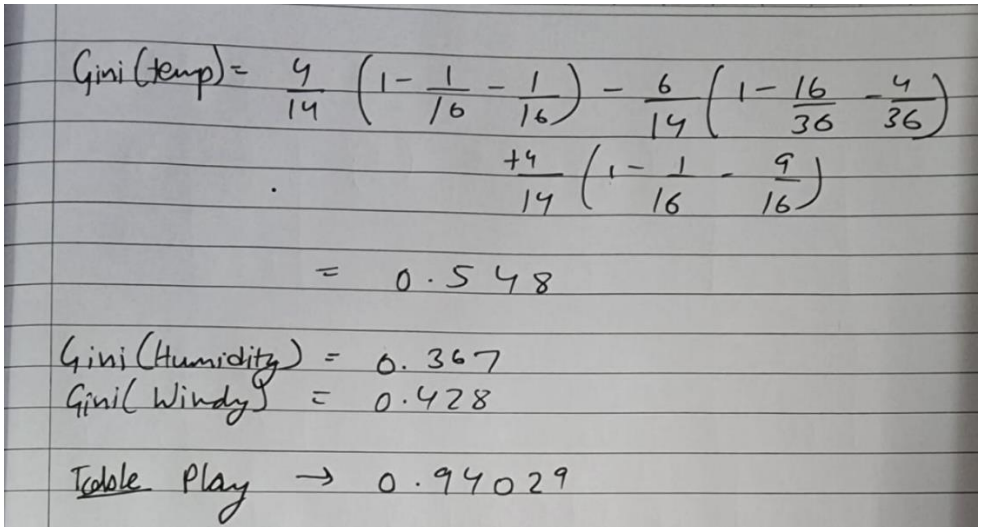
Gain (wind) $= 0.0481$

Gain (Play) $= 0.9403$

Gini:

Gini (outlook) $= \frac{5}{14}\left(1 - \frac{4}{75} - \frac{9}{75}\right) + \frac{4}{14}\left(1 - 1^2 - 0\right) + \frac{5}{14}\left(1 - \frac{9}{75} - \frac{4}{75}\right)$

$= \frac{12}{35} = 0.343$



Gini (temp) $= \frac{4}{14}\left(1 - \frac{1}{16} - \frac{1}{16}\right) - \frac{6}{14}\left(1 - \frac{16}{36} - \frac{4}{36}\right) + \frac{4}{14}\left(1 - \frac{1}{16} - \frac{9}{16}\right)$

$= 0.548$

Gini (Humidity) $= 0.367$

Gini (Windy) $= 0.428$

Table Play $\rightarrow$ 0.94029

Experiment 6:

[Regression Trees]

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from io import StringIO
from sklearn.tree import export_graphviz
import pydotplus
from IPython.display import Image
import graphviz
```

✓ 0.6s

```
dataset = np.array([['Asset Flip', 100, 1000],
                    ['Text Based', 500, 3000],
                    ['Visual Novel', 1500, 5000],
                    ['2D Pixel Art', 3500, 8000],
                    ['2D Vector Art', 5000, 6500],
                    ['Strategy', 6000, 7000],
                    ['First Person Shooter', 8000, 15000],
                    ['Simulator', 9500, 20000],
                    ['Racing', 12000, 21000],
                    ['RPG', 14000, 25000],
                    ['Sandbox', 15500, 27000],
                    ['Open-World', 16500, 30000],
                    ['MMOFPS', 25000, 52000],
                    ['MMORPG', 30000, 80000]
])
```

✓ 0.8s

```
x = dataset[:, 1:2].astype(int)
y = dataset[:, 2].astype(int)
```

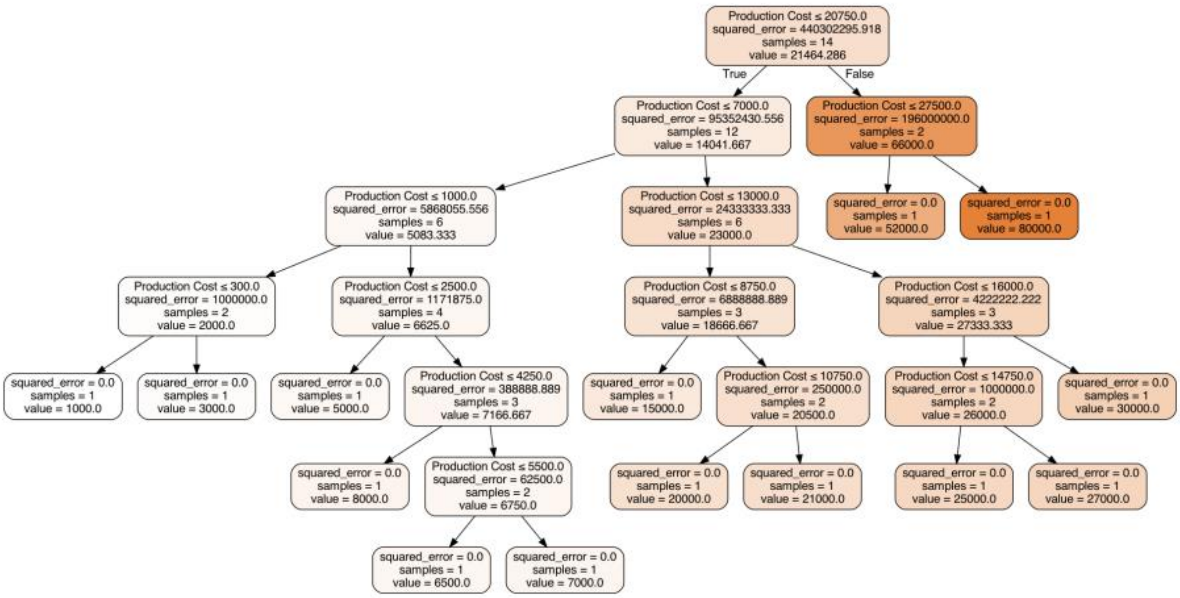
✓ 0.8s

```
regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(x, y)
y_pred = regressor.predict(x)
```

✓ 0.1s

```
tree_data = StringIO()
export_graphviz(regressor,
    out_file=tree_data,
    feature_names=['Production Cost'],
    filled=True,
    rounded=True,
    special_characters=True)
graph = pydotplus.graph_from_dot_data(tree_data.getvalue())
graph.write_png('tree.png')
Image(graph.create_png())
```

OUTPUT:


Production Cost ≤ 20750.0
squared_error = 440302295.918
samples = 14
value = 21464.286
True
False
Production Cost ≤ 7000.0
squared_error = 95352430.556
samples = 12
value = 14041.667
Production Cost ≤ 27500.0
squared_error = 196000000.0
samples = 2
value = 66000.0
Production Cost ≤ 1000.0
squared_error = 5868055.556
samples = 6
value = 5083.333
Production Cost ≤ 13000.0
squared_error = 24333333.333
samples = 6
value = 23000.0
squared_error = 0.0
samples = 1
value = 52000.0
squared_error = 0.0
samples = 1
value = 80000.0
Production Cost ≤ 300.0
squared_error = 1000000.0
samples = 2
value = 2000.0
Production Cost ≤ 2500.0
squared_error = 1171875.0
samples = 4
value = 6625.0
Production Cost ≤ 8750.0
squared_error = 6888888.889
samples = 3
value = 18666.667
Production Cost ≤ 16000.0
squared_error = 4222222.222
samples = 3
value = 27333.333
squared_error = 0.0
samples = 1
value = 1000.0
squared_error = 0.0
samples = 1
value = 3000.0
squared_error = 0.0
samples = 1
value = 5000.0
Production Cost ≤ 4250.0
squared_error = 388888.889
samples = 3
value = 7166.667
squared_error = 0.0
samples = 1
value = 15000.0
Production Cost ≤ 10750.0
squared_error = 250000.0
samples = 2
value = 20500.0
Production Cost ≤ 14750.0
squared_error = 1000000.0
samples = 2
value = 26000.0
squared_error = 0.0
samples = 1
value = 30000.0
squared_error = 0.0
samples = 1
value = 8000.0
Production Cost ≤ 5500.0
squared_error = 62500.0
samples = 2
value = 6750.0
squared_error = 0.0
samples = 1
value = 20000.0
squared_error = 0.0
samples = 1
value = 21000.0
squared_error = 0.0
samples = 1
value = 25000.0
squared_error = 0.0
samples = 1
value = 27000.0
squared_error = 0.0
samples = 1
value = 6500.0
squared_error = 0.0
samples = 1
value = 7000.0

Experiment 7

[Linear and Non-Linear SVM]

```
from sklearn import datasets
from sklearn import svm
from sklearn import metrics
from sklearn.model_selection import train_test_split
```

✓ 6.4s

```
digits = datasets.load_digits()
X_train_digits, X_test_digits, y_train_digits, y_test_digits = train_test_split(digits.data, digits.target, test_size=0.2, random_state=0)
```

✓ 0.1s

Linear SVM:

```
#Linear
clf = svm.SVC(kernel='linear', C=1)
clf.fit(X_train_digits, y_train_digits)
y_pred = clf.predict(X_test_digits)
print("Accuracy:",metrics.accuracy_score(y_test_digits, y_pred))
print("Precision:",metrics.precision_score(y_test_digits, y_pred, average='macro'))
print("Recall:",metrics.recall_score(y_test_digits, y_pred, average='macro'))
print("F1:",metrics.f1_score(y_test_digits, y_pred, average='macro'))
```

✓ 0.1s

```
Accuracy: 0.9777777777777777
Precision: 0.977551023009427
Recall: 0.9797997733973343
F1: 0.9785225911712278
```

Non-Linear SVM:

```
#Non-Linear
clf = svm.SVC(kernel='rbf', C=1)
clf.fit(X_train_digits, y_train_digits)
y_pred = clf.predict(X_test_digits)
print("Accuracy:",metrics.accuracy_score(y_test_digits, y_pred))
print("Precision:",metrics.precision_score(y_test_digits, y_pred, average='macro'))
print("Recall:",metrics.recall_score(y_test_digits, y_pred, average='macro'))
print("F1:",metrics.f1_score(y_test_digits, y_pred, average='macro'))
```

✓ 0.1s

```
Accuracy: 0.9916666666666667
Precision: 0.9922831978319785
Recall: 0.9924968730456534
F1: 0.9923538236068297
```

Experiment 8

[Apply **4** Supervised Learning Algorithms]

CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
```

✓ 0.3s

```
def plotModels(X_train, X_test, y_train, y_test):
    #TRAINING SET
    plt.figure(figsize=(15, 10))
    plt.subplot(2, 2, 1)
    plt.title('KNN')
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='rainbow')
    plt.xlabel('Age')
    plt.ylabel('Estimated Salary')
    plt.subplot(2, 2, 2)
    plt.title('SVM')
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='rainbow')
    plt.xlabel('Age')
    plt.ylabel('Estimated Salary')
    plt.subplot(2, 2, 3)
    plt.title('Decision Tree')
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='rainbow')
    plt.xlabel('Age')
    plt.ylabel('Estimated Salary')
    plt.subplot(2, 2, 4)
    plt.title('Random Forest')
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='rainbow')
    plt.xlabel('Age')
    plt.ylabel('Estimated Salary')
    plt.show()
```

```
    #TESTING SET
plt.figure(figsize=(15, 10 ))
plt.subplot(2, 2, 1)
plt.title('KNN')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='rainbow')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.subplot(2, 2, 2)
plt.title('SVM')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='rainbow')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.subplot(2, 2, 3)
plt.title('Decision Tree')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='rainbow')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.subplot(2, 2, 4)
plt.title('Random Forest')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='rainbow')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.show()
```

```python
def plotConfusionMatrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True)
    plt.xlabel('Predicted')
    plt.ylabel('Truth')
    plt.show()
```

✓ 0.4s

```python
def plotRocCurves(X_train, X_test, y_train, y_test):

    y_pred_knn = knn(X_train, X_test, y_train, y_test)
    y_pred_svm = svm(X_train, X_test, y_train, y_test)
    y_pred_dt = decisionTree(X_train, X_test, y_train, y_test)
    y_pred_rf = randomForest(X_train, X_test, y_train, y_test)
    y_pred_lr = logisticRegression(X_train, X_test, y_train, y_test)
    fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_pred_knn)
    fpr_svm, tpr_svm, thresholds_svm = roc_curve(y_test, y_pred_svm)
    fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, y_pred_dt)
    fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test, y_pred_rf)
    fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, y_pred_lr)
    plt.figure(figsize=(10, 7))
    plt.plot(fpr_knn, tpr_knn, label='KNN')
    plt.plot(fpr_svm, tpr_svm, label='SVM')
    plt.plot(fpr_dt, tpr_dt, label='Decision Tree')
    plt.plot(fpr_rf, tpr_rf, label='Random Forest')
    plt.plot(fpr_lr, tpr_lr, label='Logistic Regression')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend()
    plt.show()
```

✓ 0.1s

```python
#KNN Algorithm
def knn(X_train, X_test, y_train, y_test):
    knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print(accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    plotModels(X_train, X_test, y_train, y_test)
    return y_pred
```

```python
#SVM
def svm(X_train, X_test, y_train, y_test):
    svm = SVC(kernel='linear', random_state=0)
    svm.fit(X_train, y_train)
    y_pred = svm.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print(accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    return y_pred
```

✓ 0.7s

```python
#Decision Tree
def decisionTree(X_train, X_test, y_train, y_test):
    dt = DecisionTreeClassifier(criterion='entropy', random_state=0)
    dt.fit(X_train, y_train)
    y_pred = dt.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print(accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    return y_pred
```

✓ 0.6s

```python
def randomForest(X_train, X_test, y_train, y_test):
    rf = RandomForestClassifier(n_estimators=10, criterion='entropy', random_state=0)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print(accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    return y_pred
```

✓ 0.6s

```python
#Logistic Regression
def logisticRegression(X_train, X_test, y_train, y_test):
    lr = LogisticRegression(random_state=0)
    lr.fit(X_train, y_train)
    y_pred = lr.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print(accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))
    return y_pred
```

✓ 0.7s

```python
def main():
    dataset = pd.read_csv('datasets/Social_Network_Ads.csv')
    X = dataset.iloc[:, [2, 3]].values
    y = dataset.iloc[:, 4].values
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
    y_pred = knn(X_train, X_test, y_train, y_test)
    y_pred = svm(X_train, X_test, y_train, y_test)
    y_pred = decisionTree(X_train, X_test, y_train, y_test)
    y_pred = randomForest(X_train, X_test, y_train, y_test)
    y_pred = logisticRegression(X_train, X_test, y_train, y_test)
    plotConfusionMatrix(y_test, y_pred)
    plotRocCurves(X_train, X_test, y_train, y_test)
    plotModels(X_train, X_test, y_train, y_test)
```

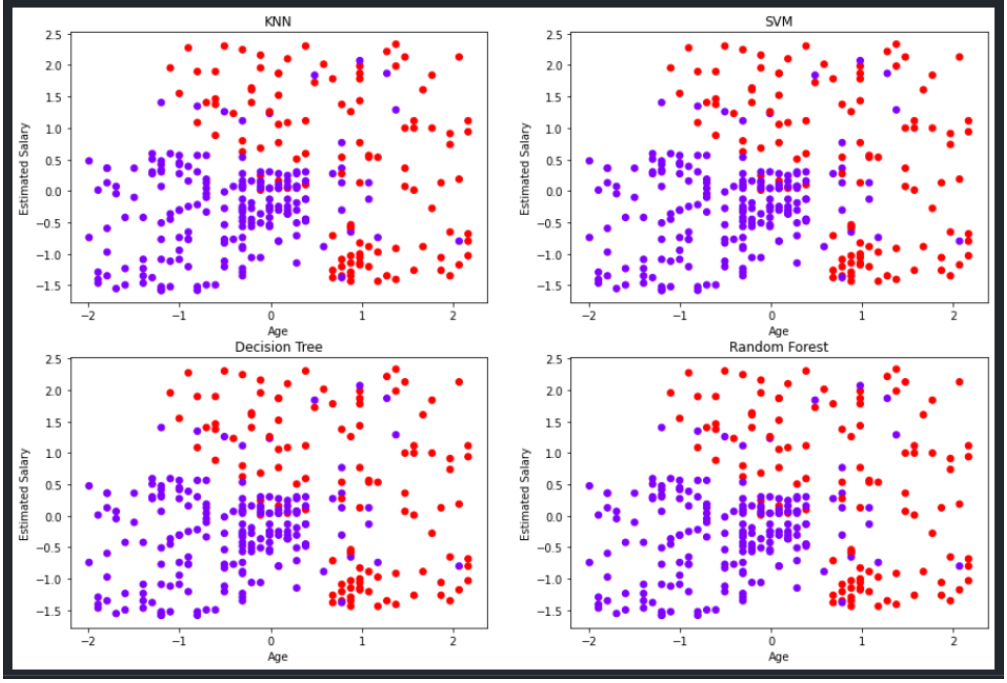
✓ 0.5s

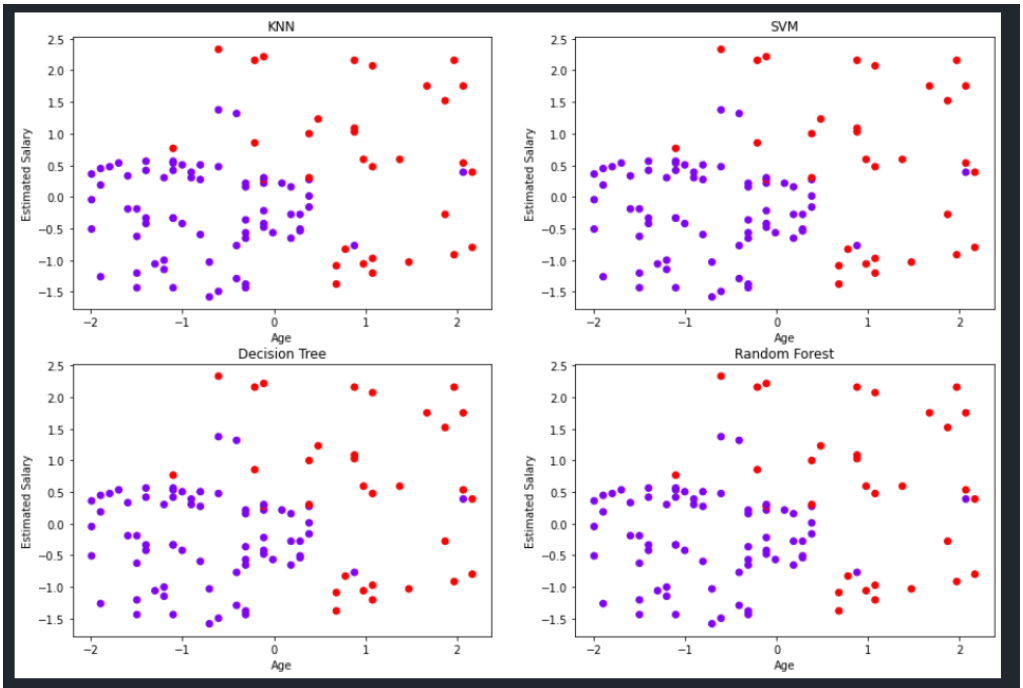
```python
main()
```

✓ 5.7s

OUTPUT:

```
[[64  4]
 [ 3 29]]
0.93
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        68
           1       0.88      0.91      0.89        32

    accuracy                           0.93       100
   macro avg       0.92      0.92      0.92       100
weighted avg       0.93      0.93      0.93       100
```

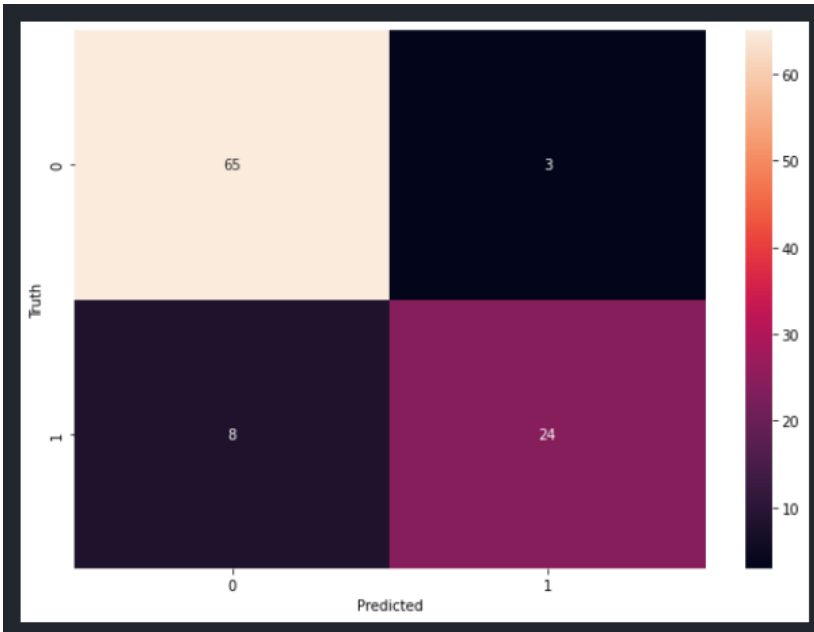




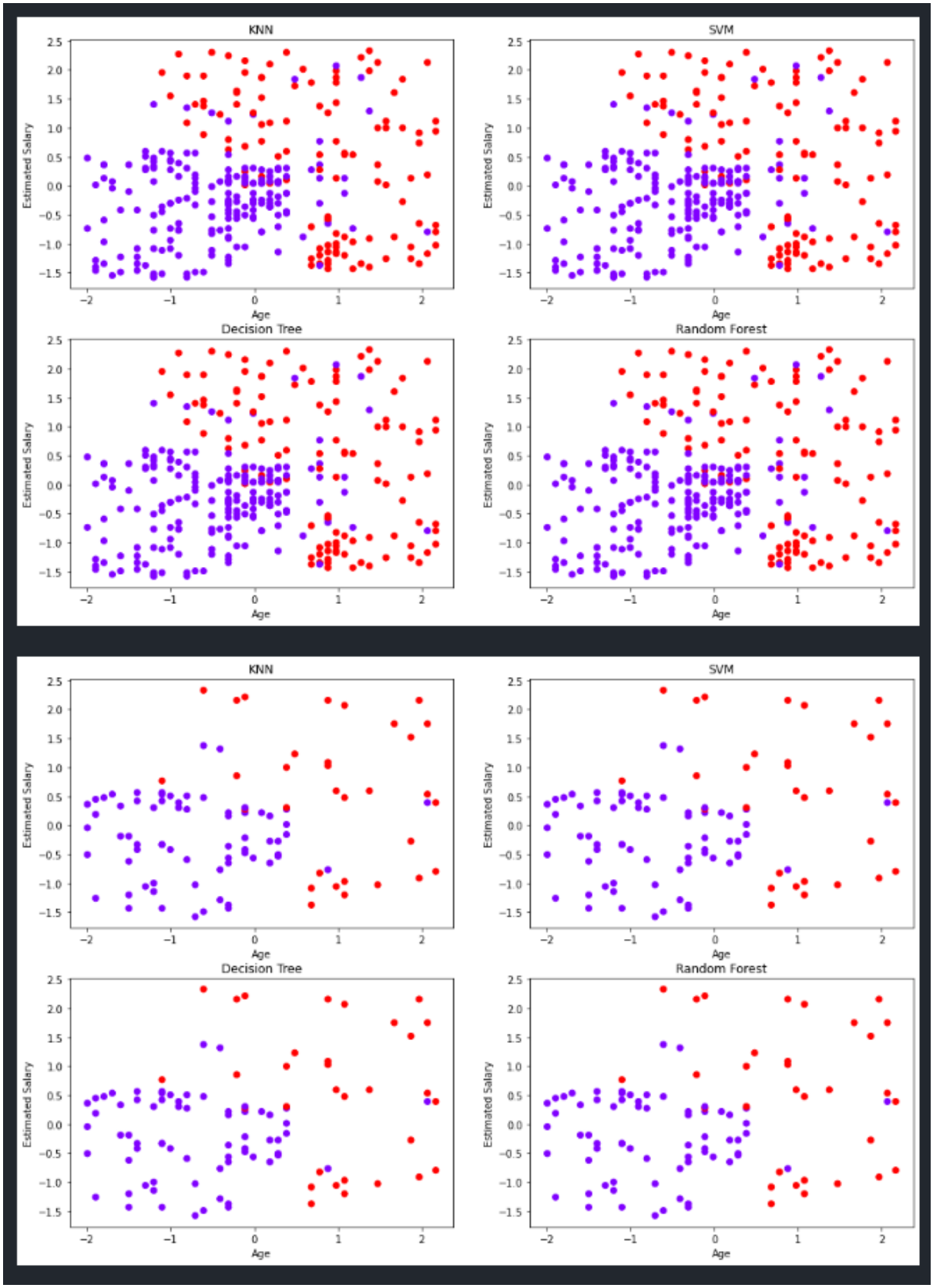
Output exceeds the size limit. Open the full output data in a text editor

```
[[66  2]
 [ 8 24]]
0.9
```

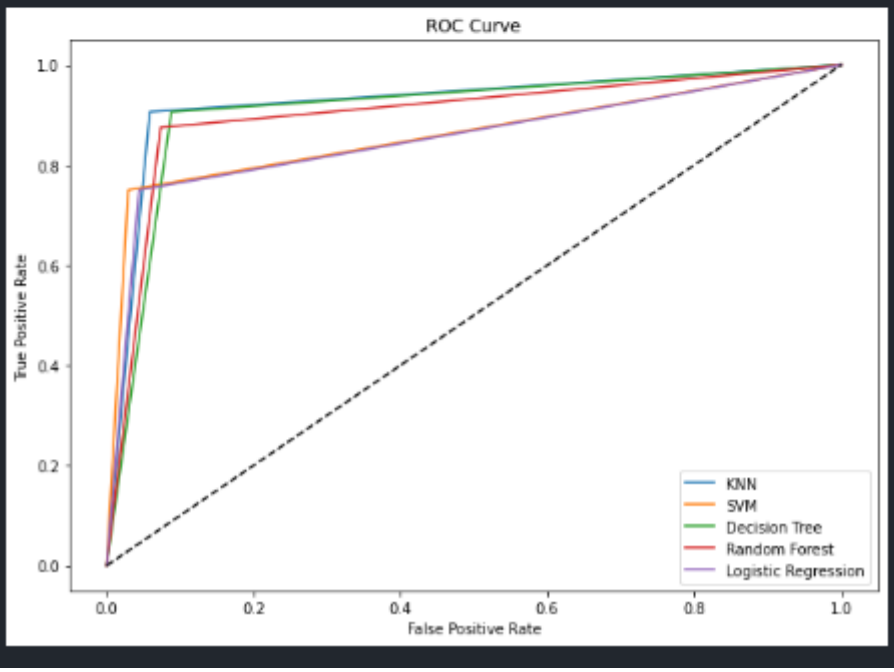
```
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        68
           1       0.92      0.75      0.83        32

    accuracy                           0.90       100
   macro avg       0.91      0.86      0.88       100
weighted avg       0.90      0.90      0.90       100

[[62  6]
 [ 3 29]]
0.91
              precision    recall  f1-score   support

           0       0.95      0.91      0.93        68
           1       0.83      0.91      0.87        32

    accuracy                           0.91       100
   macro avg       0.89      0.91      0.90       100
weighted avg       0.91      0.91      0.91       100

[[63  5]
...
    accuracy                           0.89       100
   macro avg       0.89      0.85      0.87       100
weighted avg       0.89      0.89      0.89       100
```

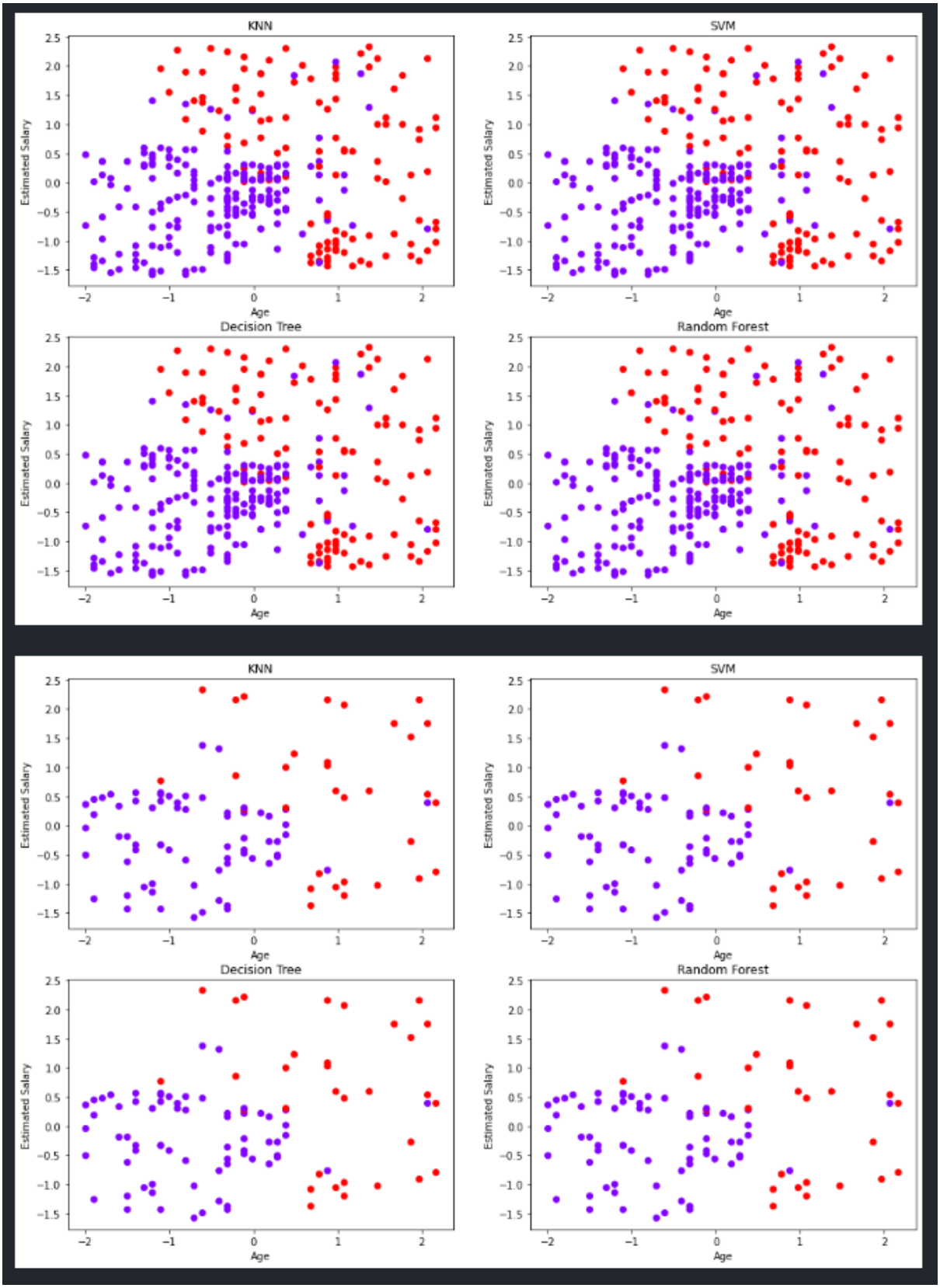



```
[[64  4]
 [ 3 29]]
0.93
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        68
           1       0.88      0.91      0.89        32

    accuracy                           0.93       100
   macro avg       0.92      0.92      0.92       100
weighted avg       0.93      0.93      0.93       100
```


KNN
SVM
Decision Tree
Random Forest
Estimated Salary
Age
KNN
SVM
Decision Tree
Random Forest
Estimated Salary
Age

```
[[66  2]
 [ 8 24]]
0.9
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        68
           1       0.92      0.75      0.83        32

    accuracy                           0.90       100
   macro avg       0.91      0.86      0.88       100
weighted avg       0.90      0.90      0.90       100

[[62  6]
 [ 3 29]]
0.91
              precision    recall  f1-score   support

           0       0.95      0.91      0.93        68
           1       0.83      0.91      0.87        32

    accuracy                           0.91       100
   macro avg       0.89      0.91      0.90       100
weighted avg       0.91      0.91      0.91       100

[[63  5]
...
    accuracy                           0.89       100
   macro avg       0.89      0.85      0.87       100
weighted avg       0.89      0.89      0.89       100
```




KNN
SVM
Decision Tree
Random Forest
Estimated Salary
Age
KNN
SVM
Decision Tree
Random Forest
Estimated Salary
Age

Experiment 9

[K-means clustering]

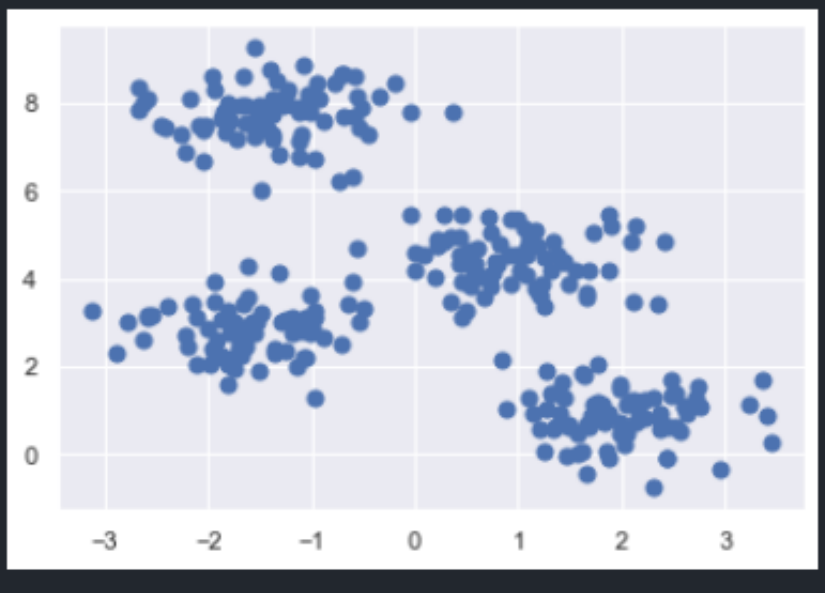
CODE and OUTPUT:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin
```

✓ 0.6s

```
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);
```

✓ 0.4s

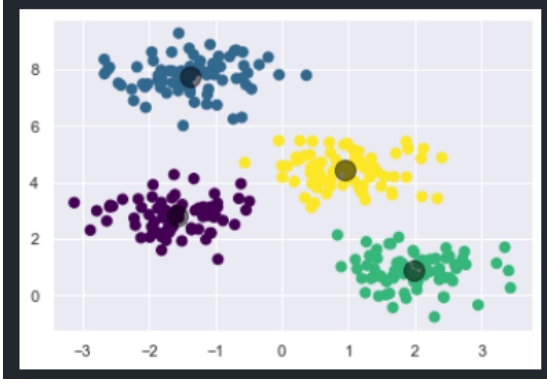



```
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

✓ 0.2s

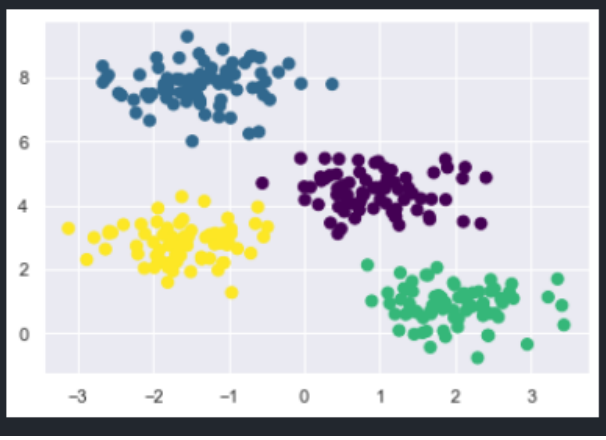
```
plt.scatter(X[:, 0], X[:, 1], c = y_kmeans, s = 50, cmap = 'viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c = 'black', s = 200, alpha = 0.5);
```

✓ 0.3s



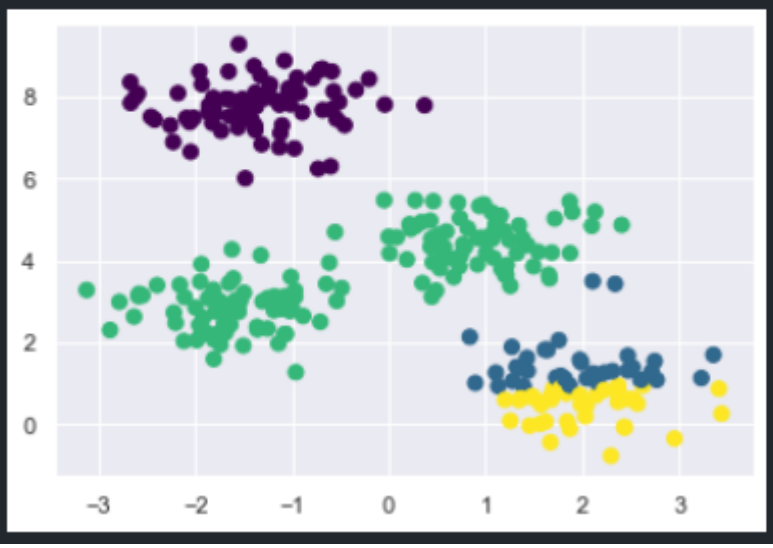

```
def find_clusters(X, n_clusters, rseed=2):
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]

    while True:
        labels = pairwise_distances_argmin(X, centers)

        new_centers = np.array([X[labels == i].mean(0) for i in range(n_clusters)])

        if np.all(centers == new_centers):
            break
        centers = new_centers

    return centers, labels

centers, labels = find_clusters(X, 4)
plt.scatter(X[:, 0], X[:, 1], c = labels, s = 50, cmap = 'viridis');
```

✓ 0.5s



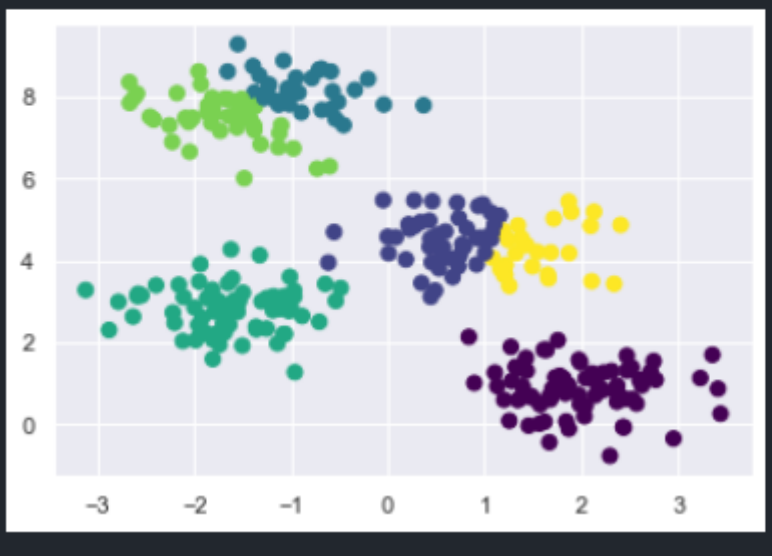
```
centers, labels = find_clusters(X, 4, rseed = 0)
plt.scatter(X[:, 0], X[:, 1], c = labels, s = 50, cmap = 'viridis');
```

✓ 0.3s




```
labels = KMeans(6, random_state = 0).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c = labels, s = 50, cmap = 'viridis');
```

✓ 0.4s

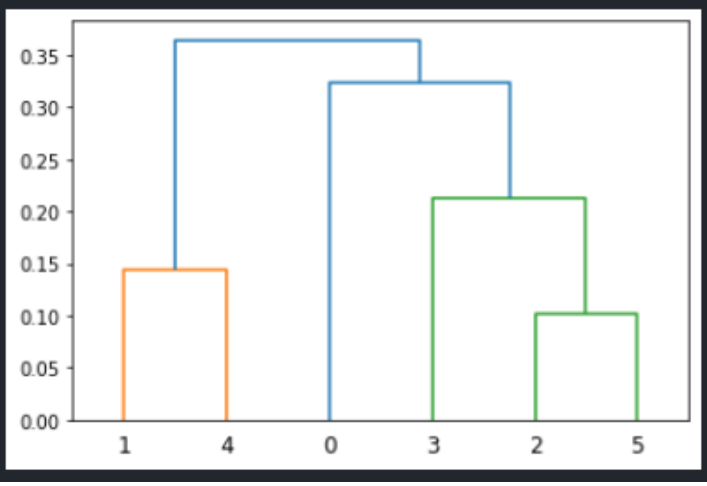


Experiment 10

[Hierarchical Clustering]

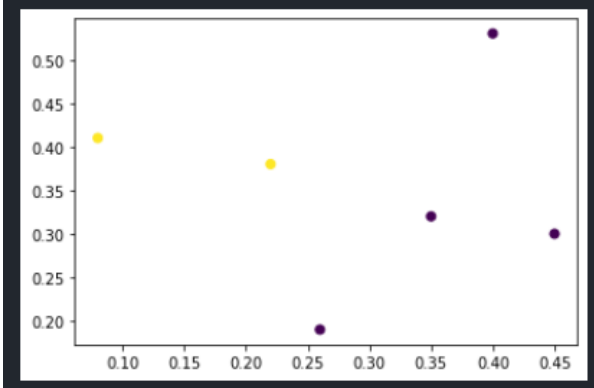
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

x = [0.40, 0.22, 0.35, 0.26, 0.08, 0.45]
y = [0.53, 0.38, 0.32, 0.19, 0.41, 0.30]

data = list(zip(x, y))

linkage_data = linkage(data, method='ward', metric='euclidean')
dendrogram(linkage_data)

plt.show()
```

✓ 11.5s




```
hierarchical_cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
labels = hierarchical_cluster.fit_predict(data)

plt.scatter(x, y, c=labels)
plt.show()
```

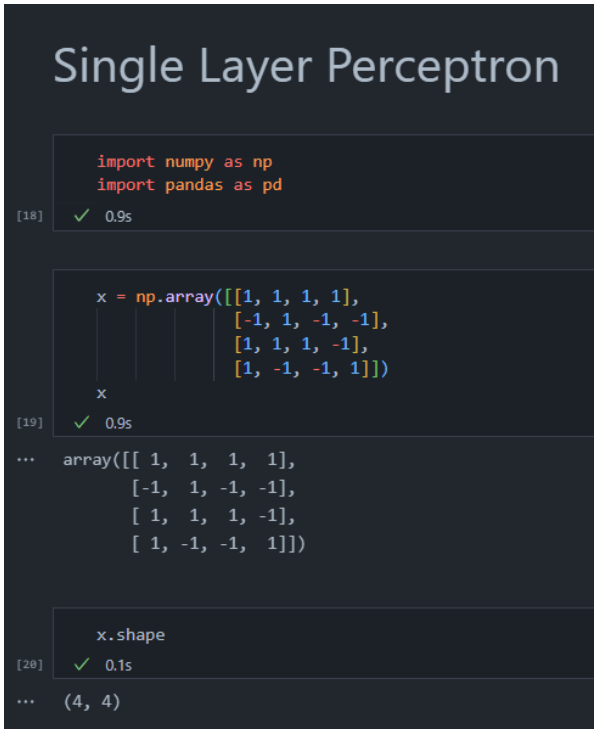
✓ 0.2s

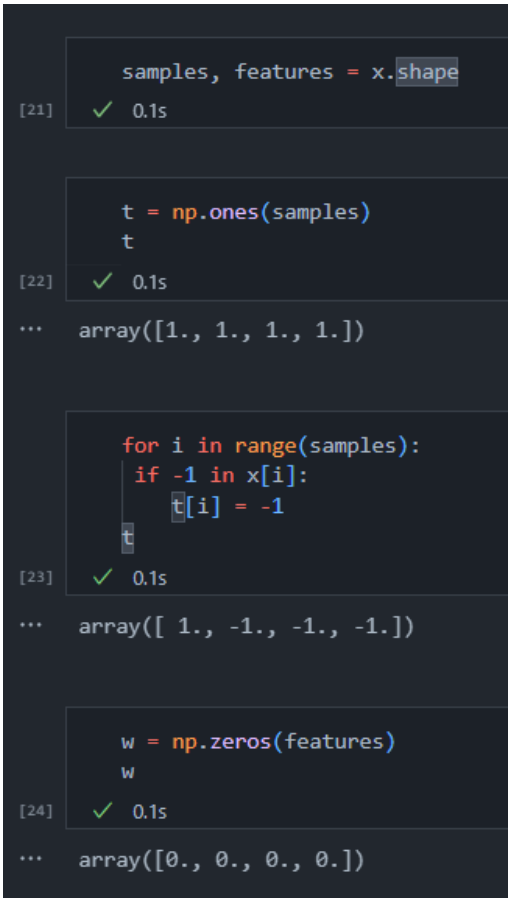


Experiment 11

[Single Layer Perceptron]

CODE & OUTPUT:





```
b = 0
alpha = 1
```

[25] ✓ 0.9s

Activation Function

```
def activation_func(y_in):
    if y_in > 0:
        return 1
    elif y_in == 0:
        return 0
    else:
        return -1
```

[26] ✓ 0.6s

Creating Perceptron

```
iterations = pd.DataFrame()
iterations[['x1','x2','x3','x4','t','y_in','y','w1','w2','w3','w4','b']] = None
```

[27] ✓ 0.7s

```
def perceptron(x,samples,features,t,w,b,alpha,epochs,iterations):
    for epoch in range(1,epochs+1):
        itr_x = []
        itr_w = []
        for id, x_i in enumerate(x):
            y_in = np.dot(x_i,w.T) + b
            y = activation_func(y_in)
            if y!= t[id]:
                for i in range(features):
                    w[i] = w[i] + alpha*t[id]*x_i[i];
                b = b + alpha*t[id]
            itr_x.extend(x_i)
            itr_w.extend(w)
            row = []
            row.extend(itr_x)
            row.append(t[id])
            row.append(y_in)
            row.append(y)
            row.extend(itr_w)
            row.append(b)
            iterations.loc[len(iterations.index)] = row
            row.clear()
            itr_x.clear()
            itr_w.clear()
        print('Iteration',epoch)
        print(iterations)
        iterations = iterations[0:0]
    return w,b
```

[28] ✓ 0.7s

```
w,b = perceptron(x,samples,features,t,w,b,alpha,2,iterations)
```
✓ 0.1s

Iteration 1

	x1	x2	x3	x4	t	y_in	y	w1	w2	w3	w4	b
0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0
1	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	-1.0	-1.0	3.0	1.0	0.0	0.0	0.0	2.0	0.0
3	1.0	-1.0	-1.0	1.0	-1.0	2.0	1.0	-1.0	1.0	1.0	1.0	-1.0

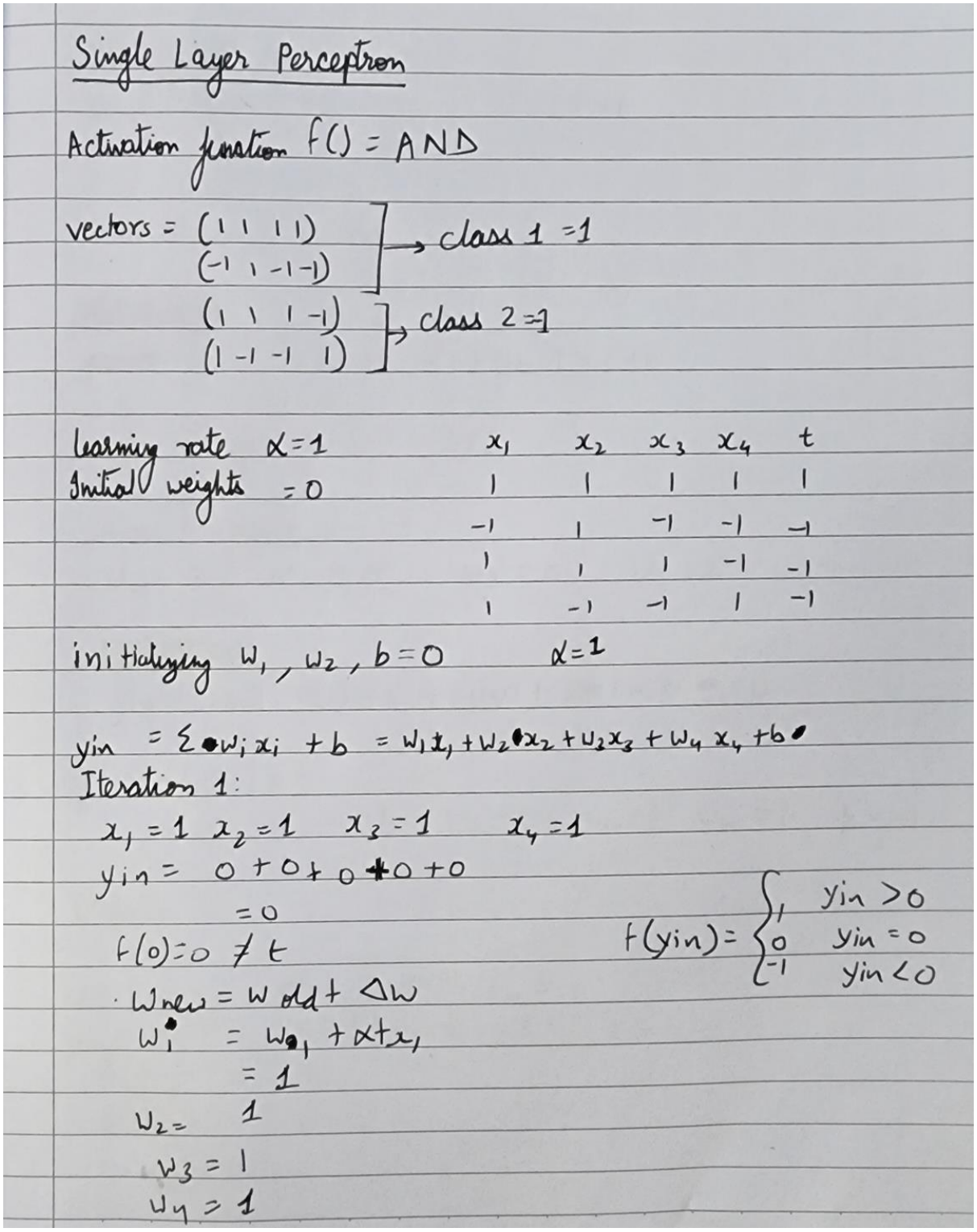
Iteration 2

	x1	x2	x3	x4	t	y_in	y	w1	w2	w3	w4	b
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	-1.0	1.0	1.0	1.0	-1.0
1	-1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	1.0	1.0	-1.0
2	1.0	1.0	1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	1.0	1.0	-1.0
3	1.0	-1.0	-1.0	1.0	-1.0	-3.0	-1.0	-1.0	1.0	1.0	1.0	-1.0

```
print('Weights: ',w)
print('Bias: ',b)
```
[30] ✓ 0.6s

```
Weights:  [-1.  1.  1.  1.]
Bias:  -1.0
```

MANUAL CALCULATIONS:



Single Layer Perceptron

Activation function f() = AND

vectors = (1 1 1 1) → class 1 = 1
(-1 1 -1 -1)
(1 1 1 -1) → class 2 = -1
(1 -1 -1 1)

learning rate $\alpha = 1$
Initial weights = 0

x_1	x_2	x_3	x_4	t
1	1	1	1	1
-1	1	-1	-1	-1
1	1	1	-1	-1
1	-1	-1	1	-1

initializing $w_1, w_2, b = 0$ $\alpha = 1$

$y_{in} = \sum w_i x_i + b = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b$

Iteration 1:

$x_1 = 1$ $x_2 = 1$ $x_3 = 1$ $x_4 = 1$

$y_{in} = 0 + 0 + 0 + 0 + 0$

$= 0$

$f(0) = 0 \neq t$

$f(y_{in}) = \begin{cases} 1 & y_{in} > 0 \\ 0 & y_{in} = 0 \\ -1 & y_{in} < 0 \end{cases}$

$w_{new} = w_{old} + \Delta w$

$w_1 = w_{o1} + \alpha t x_1$

$= 1$

$w_2 = 1$

$w_3 = 1$

$w_4 = 1$

Iteration

$y_{in} = (-1)1 + 1 \times 1 + (-1)1 + (-1)1 + 1$

$= -1$

$f(-1) = -1 = t$

$\therefore w_{new} = w_{old}$

Iteration

$y_{in} = 1 \times 1 + 1 \times 1 + 1 \times 1 + -1 \times 1 + 1$

$= 3$

$f(y_{in}) = f(3) = 1 \neq t$

$w_{new} = w_{old} + \Delta w$

$w_1 = 0 \quad w_2 = 0 \quad w_3 = 0 \quad w_4 \neq 2$

Iteration

$y_{in} = 1 \times 0 + (-1) \times 0 + (-1) \times 0 + 1 \times 2 + 0$

$= 2$

$f(y_{in}) \neq t$

$\therefore w_1 = -1 \quad w_2 = 1 \quad w_3 = 1 \quad w_4 = 1 \quad b = -1$

Iteration 2:

$x_1 = 1 \quad x_2 = 1 \quad x_3 = 1 \quad x_4 = 1 \quad t = 1$

$y_{in} = 1(-1) + 1 \times 1 + 1 \times 1 + 1 \times 1 - 1 = 1$

$f(y_{in}) = 1 = t$

$w_{new} = w_{old}$

$x_1 = -1 \quad x_2 = 1 \quad x_3 = -1 \quad x_4 = -1 \quad t = -1$

$y_{in} = (-1)(-1) + 1 \times 1 + (-1)1 + (-1)1 - 1$

$= -1$

$f(y_{in}) = -1 = t$

$x_1 = 1 \quad x_2 = 1 \quad x_3 = 1 \quad x_4 = -1 \quad t = -1$

$y_{in} = 1(-1) + 1 \times 1 + 1 \times 1 + (-1)1 - 1 = -1$

$f(y_{in}) = -1 = t$

$x_1 = 1 \quad x_2 = -1 \quad x_3 = -1 \quad x_4 = 1 \quad t = -1$

$y_{in} = 1(-1) + (-1)1 + (-1)1 + 1 \times 1 - 1 = -3$

$f(y_{in}) = -1 = t$

Table:

x_1	x_2	x_3	x_4	t	y_{in}	y	w_1	w_2	w_3	w_4	b	Δw_1	Δw_2	Δw_3	Δw_4	Δb
1	1	1	1	1	1	1	-1	1	1	1	-1	0	0	0	0	0
-1	1	-1	-1	-1	-1	-1	-1	1	1	1	-1	0	0	0	0	0
1	1	1	-1	-1	-1	-1	-1	1	1	1	-1	0	0	0	0	0
1	-1	-1	1	-1	-3	-1	-1	1	1	1	-1	0	0	0	0	0

$w = (w_1 \; w_2 \; w_3 \; w_4)$

$= (-1 \; 1 \; 1 \; 1)$

$b = -1$