MINI PROJECT 3

Problem 1

a)

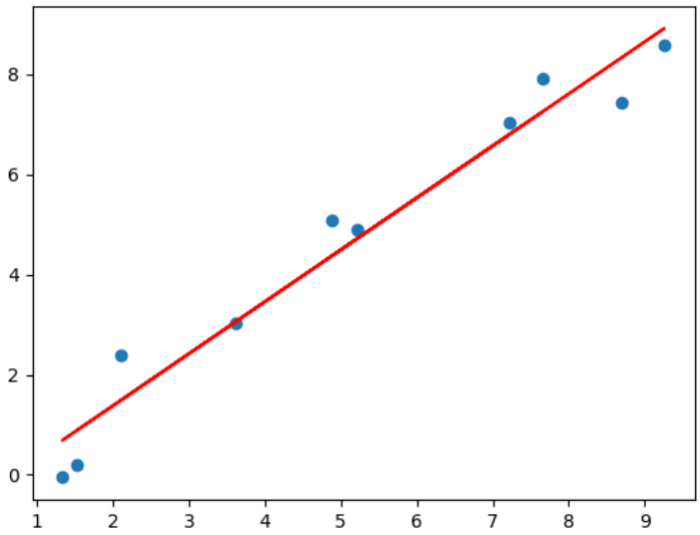
```
x = np.random.rand(10, 1) * 10

def f1(x):
    return x

y = f1(x) + np.random.randn(10, 1)

reg = LinearRegression().fit(x, y)

plt.scatter(x, y)
plt.plot(x, reg.predict(x), color='red')
plt.show()
```



b)

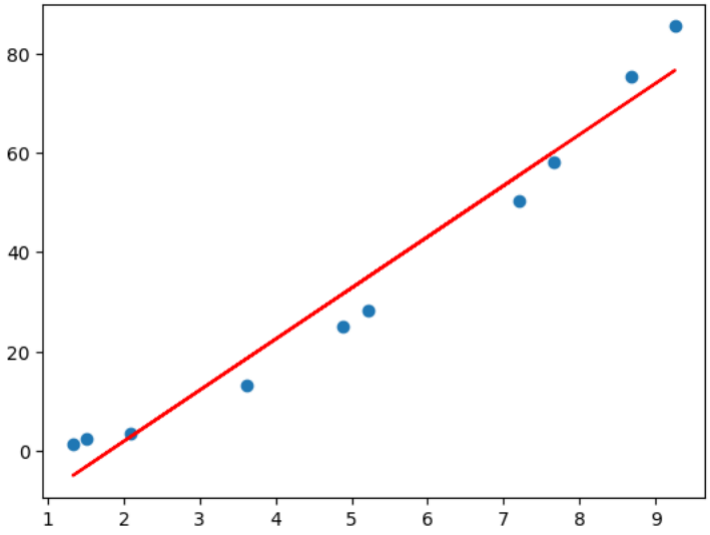
```
x = np.random.rand(10, 1) * 10

def f2(x):
    return x**2

y = f2(x) + np.random.randn(10, 1)

reg = LinearRegression().fit(x, y)

plt.scatter(x, y)
plt.plot(x, reg.predict(x), color='red')
plt.show()
```



c)

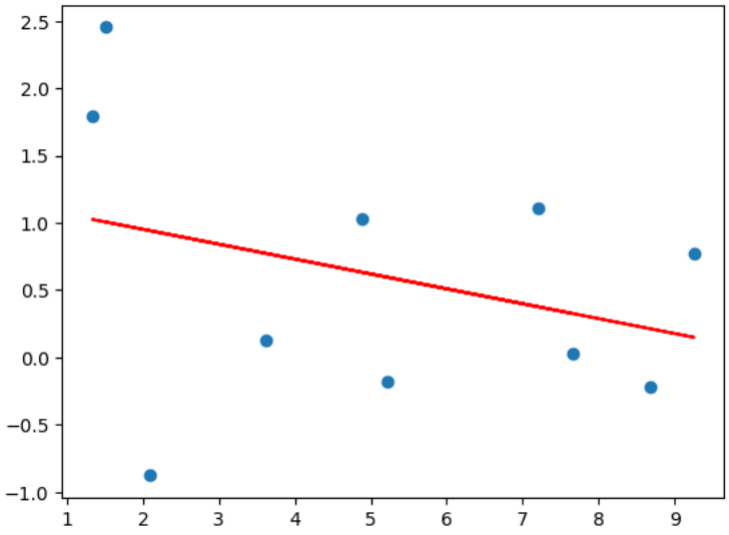
```
x = np.random.rand(10, 1) * 10

def f3(x):
    return 1/x

y = f3(x) + np.random.randn(10, 1)

reg = LinearRegression().fit(x, y)

plt.scatter(x, y)
plt.plot(x, reg.predict(x), color='red')
plt.show()
```

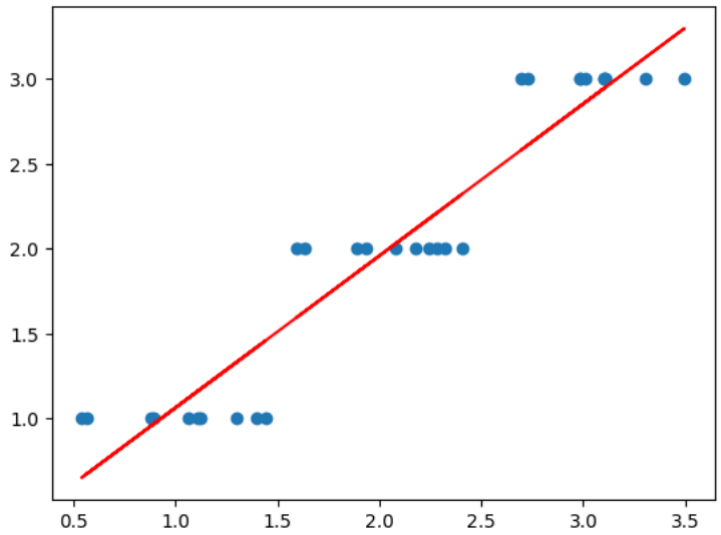


Problem 2

a)

Linear regression is a method used to model the relationship between a continuous dependent variable and one or more independent variables. It is not well-suited for classification problems, where the dependent variable is categorical.

For this data set, the prediction is shown in the figure as the red line



which does not demonstrate the nature of classification.

b)

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.linear_model import SGDClassifier
from sklearn.datasets import make_blobs
data = pd.read_excel('MP3P2.xlsx').values

X = data[:,0].reshape(-1,1)
X = np.hstack((X,np.zeros((30,1))))

y = data[:,1]

# Fit a logistic regression model
clf = LogisticRegression(multi_class='multinomial', solver='lbfgs').fit(x,y)

# Make predictions
y_pred = clf.predict(x)

# Calculate the misclassification rate
misclassification_rate = 1 - accuracy_score(y,y_pred)
print(f"Misclassification rate: {misclassification_rate:.4f}")
```

Get the misclassification rate:

```
Misclassification rate: 0.0333
```

The coefficients and intercept of the boundary:

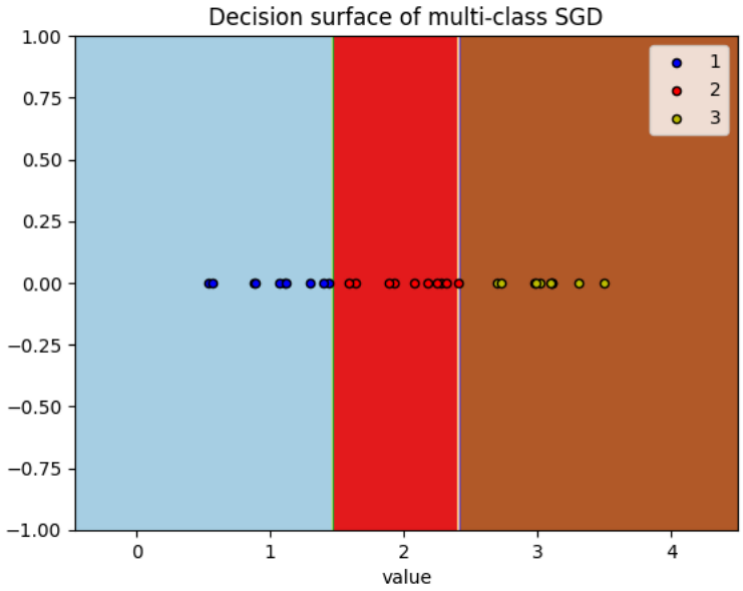
```
clf.intercept_
```

```
array([ 4.01376735,  0.68247022, -4.69623757])
```

```
clf.coef_
```

```
array([[-2.09279122],
       [-0.03206009],
       [ 2.12485131]])
```

The multi-class SGD Plot is shown below:



Code

Problem 1

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Set the random seed for reproducibility
np.random.seed(114)

# Generate 10 random x values
x = np.random.rand(10, 1) * 10

# Define the function f(x) = x
def f1(x):
    return x
def f2(x):
    return x**2
def f3(x):
    return 1/x

# Calculate the corresponding y values
y = f1(x) + np.random.randn(10, 1)
```

```
# Fit the data with linear regression
reg = LinearRegression().fit(x, y)

# Plot the scatter points and the fitted line
plt.scatter(x, y)
plt.plot(x, reg.predict(x), color='red')
plt.show()

# Calculate the corresponding y values
y = f2(x) + np.random.randn(10, 1)

# Fit the data with linear regression
reg = LinearRegression().fit(x, y)

# Plot the scatter points and the fitted line
plt.scatter(x, y)
plt.plot(x, reg.predict(x), color='red')
plt.show()

# Calculate the corresponding y values
y = f3(x) + np.random.randn(10, 1)

# Fit the data with linear regression
reg = LinearRegression().fit(x, y)

# Plot the scatter points and the fitted line
plt.scatter(x, y)
plt.plot(x, reg.predict(x), color='red')
plt.show()
```

Problem 2

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Create the data arrays
X = np.array([[1.10831869], [1.065562187], [1.297008063], [0.539575053],
[1.12193803], [1.441950117], [0.563176654], [1.400034352], [0.880844407],
[0.889932634], [2.282035223], [2.246148139], [2.174932648], [1.635140176],
[1.591358426], [1.931854177], [1.891213646], [2.404056128], [2.080155832],
[2.321821771], [3.105344985], [3.0127448], [2.981448783], [3.108971608],
[3.10061754], [2.986607138], [2.69443207], [3.305202114], [3.497028818],
[2.725654289]])
y = np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3])

# Fit a linear regression model
reg = LinearRegression().fit(X.reshape(-1, 1), y)

# Make predictions on the data
y_pred = reg.predict(X.reshape(-1, 1))

# Plot the data and the predictions
plt.scatter(X[:,0].flatten(), y)
```

```
plt.plot(X[:,0].flatten(), y_pred.flatten(), color='red')
plt.show()


import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.linear_model import SGDClassifier
from sklearn.datasets import make_blobs
data = pd.read_excel('MP3P2.xlsx').values

X = data[:,0].reshape(-1,1)
X = np.hstack((X,np.zeros((30,1))))

y = data[:,1]

# Fit a logistic regression model
clf = LogisticRegression(multi_class='multinomial', solver='lbfgs').fit(x,y)

# Make predictions
y_pred = clf.predict(x)

# Calculate the misclassification rate
misclassification_rate = 1 - accuracy_score(y,y_pred)
print(f"Misclassification rate: {misclassification_rate:.4f}")

clf.intercept_
clf.coef_

colors = "bry"

clf = SGDClassifier(alpha=0.01, max_iter=1000).fit(X, y)
ax = plt.gca()
DecisionBoundaryDisplay.from_estimator(
    clf,
    X,
    cmap=plt.cm.Paired,
    ax=ax,
    response_method="predict",
    xlabel="value",
    ylabel="",
)
plt.axis("tight")

target_names = ["0", "1", "2", "3"]

# Plot also the training points
for i, color in zip(clf.classes_, colors):
```

```
    idx = np.where(y == i)
    plt.scatter(
        X[idx, 0],
        X[idx, 1],
        c=color,
        label=target_names[int(i)],
        cmap=plt.cm.Paired,
        edgecolor="black",
        s=20,
    )
plt.title("Decision surface of multi-class SGD")
plt.axis("tight")

# Plot the three one-against-all classifiers
xmin, xmax = plt.xlim()
ymin, ymax = plt.ylim()
coef = clf.coef_
intercept = clf.intercept_


def plot_hyperplane(c, color):
    def line(x0):
        return (-(x0 * coef[c, 0]) - intercept[c]) / coef[c, 1]

    plt.plot([xmin, xmax], [line(xmin), line(xmax)], ls="--", color=color)


for i, color in zip(clf.classes_, colors):
    if i < 3:
        plot_hyperplane(int(i), color)
plt.legend()
plt.show()
```