Lab Assignment 4

Linear Regression

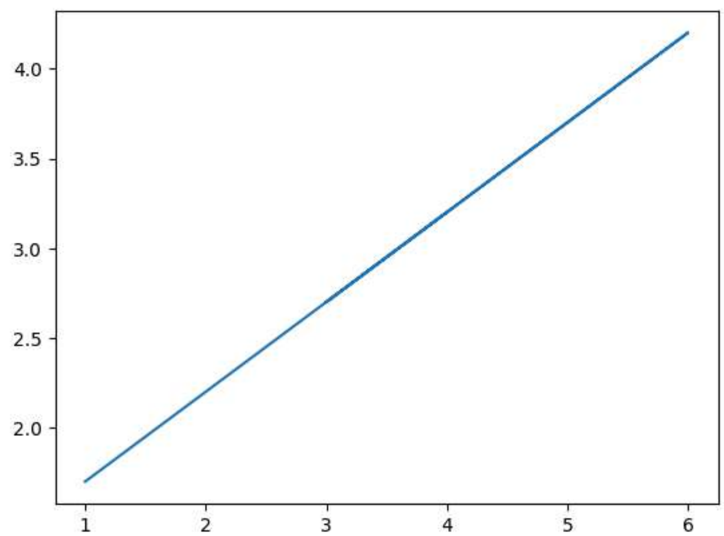
PART A : Prerequisite for linear regression implementation

1. Create an array x = [1, 1, 2, 3, 4, 3, 4, 6, 4] using numpy. Calculate a function h(x)=t0+t1*x, where t0=1.2 and t1=0.5, for all values of x and plot a graph with x on one axis and h(x)on another axis.

```
import numpy as np
import pandas as pd
from sympy import Array as disp
import matplotlib.pyplot as plt
%matplotlib inline
```

```
x = np.array([1, 1, 2, 3, 4, 3, 4 ,6, 4])
print(disp(x))
t0, t1 = 1.2, 0.5
h = lambda x: t0 + t1*x
plt.plot(x, h(x))
```

```
[1, 1, 2, 3, 4, 3, 4, 6, 4]
[<matplotlib.lines.Line2D at 0x7e060bd32740>]
```



2. Create two arrays A and B with the following values using numpy array. Let (Ai,Bi) represent a data point with i th element of A and B. A = [1, 1, 2, 3, 4, 3, 4, 6, 4] B = [2, 1, 0.5, 1, 3, 3, 2, 5, 4] Find out the dot product of the vectors. [Hint use numpy np.dot(a,b)]

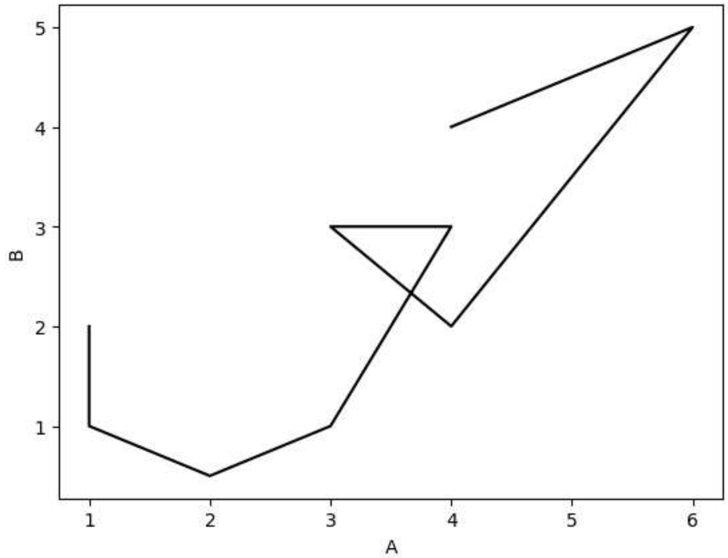
```
A = np.array([1, 1, 2, 3, 4, 3, 4, 6, 4])
B = np.array([2, 1, 0.5, 1, 3, 3, 2, 5, 4])
disp(np.dot(A, B))
```

82.0

3. Plot a graph marking the data points (Ai,Bi) with A on the X-axis and B on the Y-axis.

```
plt.plot(A, B, 'k-')
plt.xlabel('A')
plt.ylabel('B')
```

```
Text(0, 0.5, 'B')
```



4. Calculate Mean Square Error (MSE) of A and B with the formulae where n is the no: of sample data points.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(A^i - B^i)^2$$

```
MSE = lambda A, B: np.sum(np.square(A-B))/len(A)
MSE(A, B)
```

```
1.4722222222222223
```

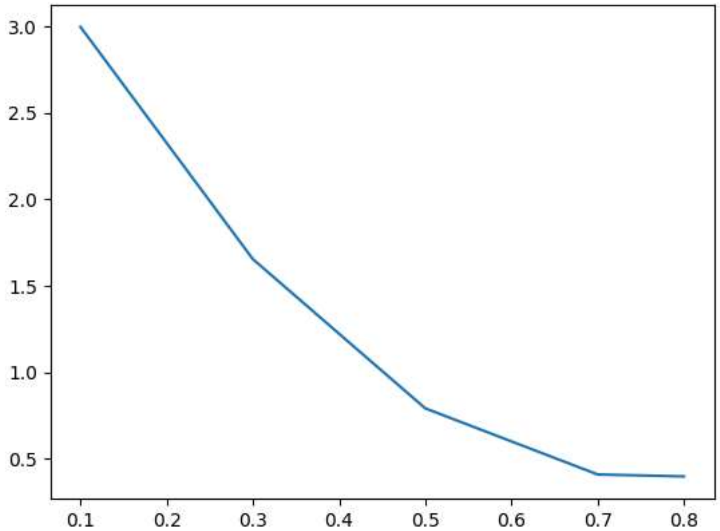
5. Modify the above equation with the following cost function. Implement as a function with prototype def compute_cost_function(n,t1,A,B):

$$J(t_1) = \frac{1}{2n}\sum_{i=1}^{n}(h(A^i) - B^i))^2$$

Take $h(x) = t_1 * x$ and $t_1 = 0.5$ Modify the above code iterating for different values of t_1 and calculate $J(t_1)$.Try with t_1 =0.1,0.3,0.5,0.7,0.8. Plot a graph with t_1 on X-axis and $J(t_1)$ on Y-axis. [hint sum_squared_error = np.square(np.dot(features, theta) - values).sum() cost = sum_squared_error / (2*m)]

```
h = lambda x, t1: t1*x
J = lambda n, t1, A, B: np.sum(np.square(h(t1,A)-B))/(2*n)
t1 = [0.1,0.3,0.5,0.7,0.8]
out = [J(9, i, A, B) for i in t1]
plt.plot(t1, out)
```

```
[<matplotlib.lines.Line2D at 0x7e0609aab6d0>]
```



PART B : Linear Regression Implementation

1. Linear regression with one variable.

   a. Generate a new data set from student scores with one feature studytime and output variable average grade = (G1+G2+G3)/3

```
df = pd.read_csv('/content/Students.csv')
```

```
df.head()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4
3	GP	F	15	U	GT3	T	4	2	health	services	...	3
4	GP	F	16	U	GT3	T	3	3	other	other	...	4

5 rows × 33 columns

```
df.columns
```

```
Index(['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
       'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime',
       'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
       'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc',
       'Walc', 'health', 'absences', 'G1', 'G2', 'G3'],
      dtype='object')
```

b. Load the new data set

c. Plot data

```
db = pd.DataFrame(df['studytime'])
```

```
db.insert(1, 'Grade', (df['G1']+df['G2']+df['G3'])/3)
```

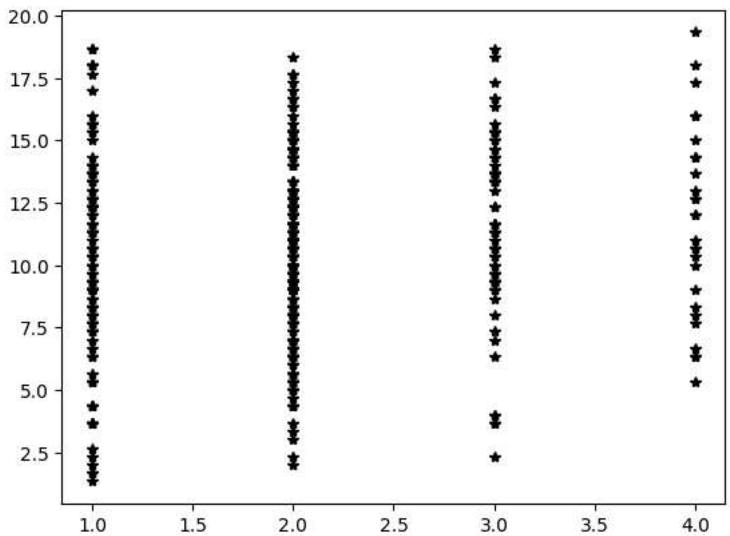
```
db.head()
```

	studytime	Grade
0	2	5.666667
1	2	5.333333
2	2	8.333333
3	3	14.666667
4	2	8.666667

Next steps: Generate code with db View recommended plots

```
plt.plot(db["studytime"], db["Grade"], 'k*')
```

```
[<matplotlib.lines.Line2D at 0x7e06099a2ce0>]
```



d. Implement linear regression using inbuilt package python Scikit

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error


X = db['studytime'].values.reshape(-1, 1)
y = db['Grade'].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

scaler = StandardScaler()
scaler.fit(y_train)
y_train = scaler.transform(y_train)
y_test = scaler.transform(y_test)

reg = LinearRegression()

reg.fit(X_train, y_train)

y_pred = reg.predict(X_test)


print(reg.score(X_test, y_test))

print(mean_squared_error(y_test, y_pred))
```

```
-0.0006771080882395086
1.1366651438604667
```

e. Implement gradient descent algorithm with the function prototype def gradient_descent(alpha, x, y, max_iter=1500): where alpha is the learning rate, x is the input feature vector. y is the target. Subject the feature vector to normalisation step if needed. Convergence criteria: when no: of iterations exceed max_iter.

[hint sum_squared_error = np.square(np.dot(features, theta) - values).sum() cost = sum_squared_error / (2*m)]

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \qquad \text{(simultaneously update } \theta_j \text{ for all } j\text{).}$$

```
h = lambda x, params: params[0]*x + params[1]
grad = lambda x, y, params: (-2/y.shape[0])*np.array([np.sum(x*(y-h(x, params))), np.sum(y-h(x, params))])
cost = lambda x, y, params: np.sum(np.square(y - h(x, params)))/y.shape[0]
```

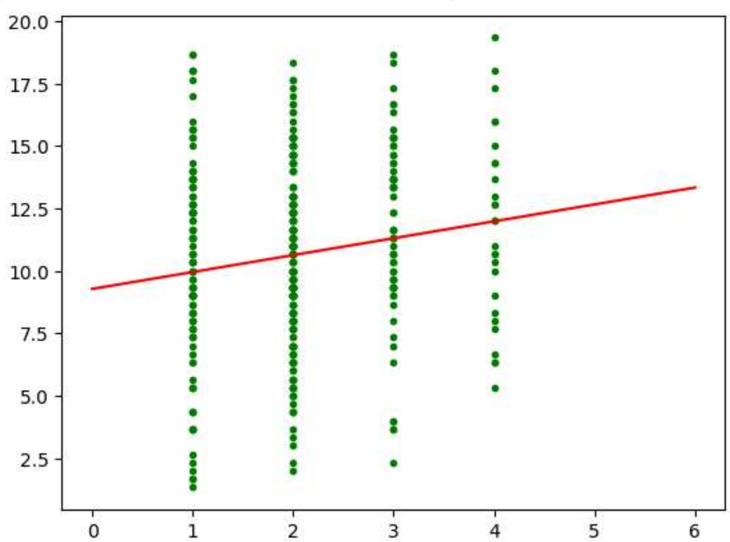
```
def gradient_descent(alpha, x, y, max_iter=1500, tolerance=[1e-52, 1e-52]):
  params = np.zeros(x.shape[1]+1)
  history = []
  history.append([params, cost(x, y, params)])
  for i in range(max_iter):
    gradient = grad(x, y, params)
    if abs(gradient[0]) <= tolerance[0] and abs(gradient[1]) <= tolerance[1] :
      return params, history
    params = params - alpha*grad(x, y, params)
    history.append([params, cost(x, y, params)])
  return params, history
params, history = gradient_descent(0.01, X, y)
```

```
print(params)
```

```
[0.67662816 9.27819384]
```

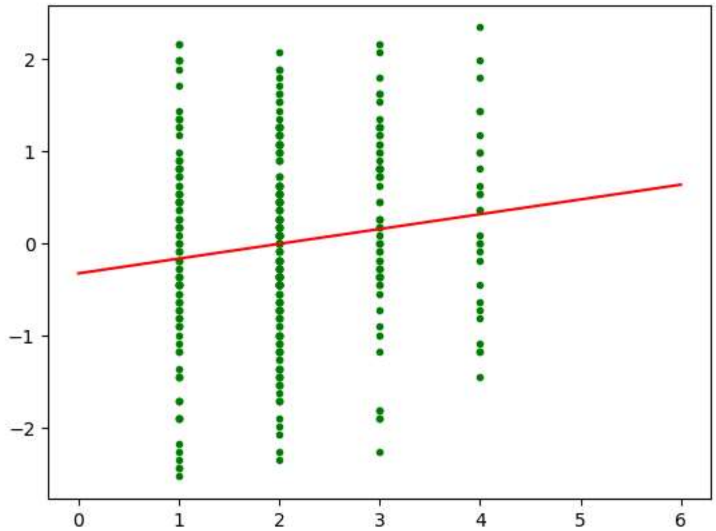
```
plt.plot(X, y, 'g.')
plt.plot(np.linspace(0, 6, 100), h(np.linspace(0, 6, 100) ,params), 'r-')
```

```
[<matplotlib.lines.Line2D at 0x7e06012d3a90>]
```



```
scaler = StandardScaler()
y_scaled = scaler.fit_transform(y)
params, history = gradient_descent(0.17, X, y_scaled)
print(params)
plt.plot(X, y_scaled, 'g.')
plt.plot(np.linspace(0, 6, 100), h(np.linspace(0, 6, 100) ,params), 'r-')
```

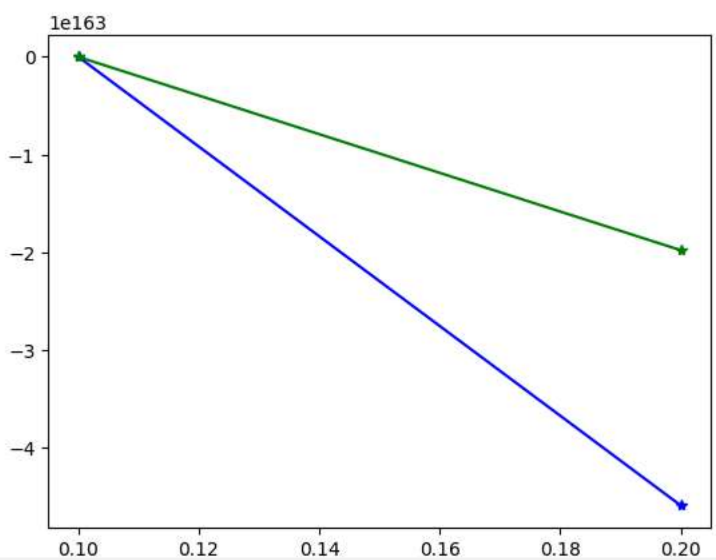
```
[ 0.16054445 -0.32677908]
[<matplotlib.lines.Line2D at 0x7e05ff1bf5e0>]
```



f. Vary learning rate from 0.1 to 0.9 and observe the learned parameter.

```
alpha = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
parameters = []
for i in alpha:
  parameters.append(gradient_descent(i, X, y_scaled)[0])
plt.plot(alpha, [i[0] for i in parameters], 'b*-')
plt.plot(alpha, [i[1] for i in parameters], 'g*-')
```

```
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py:88: RuntimeWarning: ov
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
<ipython-input-16-b3a87491cd4f>:3: RuntimeWarning: overflow encountered in square
  cost = lambda x, y, params: np.sum(np.square(y - h(x, params)))/y.shape[0]
<ipython-input-16-b3a87491cd4f>:14: RuntimeWarning: invalid value encountered in subtrac
  params = params - alpha*grad(x, y, params)
[<matplotlib.lines.Line2D at 0x7e05ff08c8e0>]
```



g. Draw the contour plot of cost function and simulate the steps of gradient descent.

Example contour for a function

xmesh, ymesh = np.mgrid[-2:2:50j,-2:2:50j]

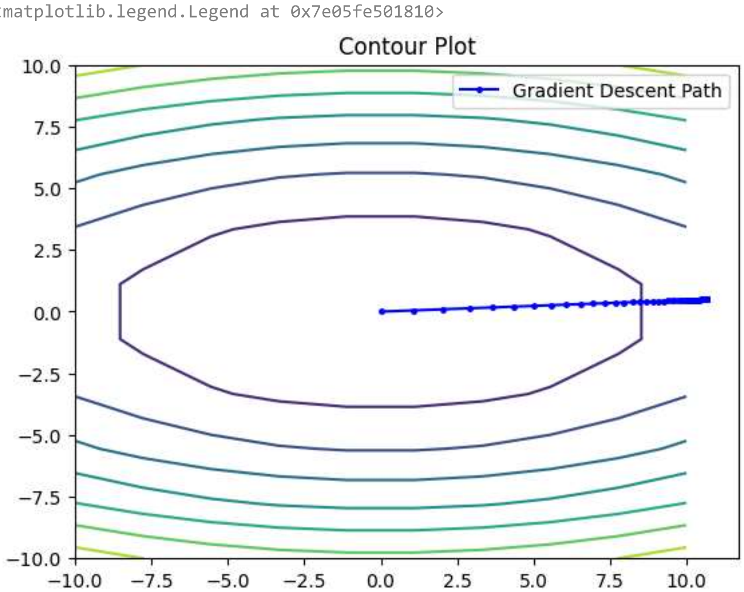
fmesh = f(np.array([xmesh, ymesh]))

plt.contour(xmesh, ymesh, fmesh) def f(x):

return 0.5*x[0]**2 + 2.5*x[1]**2

```
def f(x):
    return 0.5 * x[0]**2 + 2.5 * x[1]**2

def gradient_descent(alpha, x, y, max_iter=1500):
    x = (x - np.mean(x, axis=0)) / np.std(x, axis=0)
    x = np.c_[np.ones(x.shape[0]), x]
    m, n = x.shape
    o = np.zeros(n)
    pre = [o.copy()]

    for i in range(max_iter):
        h = np.dot(x, o)
        e = h - y
        gradient = np.dot(x.T, e) / m
        o -= alpha * gradient
        pre.append(o.copy())

    return pre

X = db['studytime'].values
y = db['Grade'].values

alpha = 0.1
pr = gradient_descent(alpha, X, y)


xmesh, ymesh = np.mgrid[-10:10:10j, -10:10:10j]
fmesh = f(np.array([xmesh, ymesh]))
plt.contour(xmesh, ymesh, fmesh)
plt.title('Contour Plot')


pre = np.array(pr)
plt.plot(pre[:, 0], pre[:, 1], 'b.-', markersize=5, label='Gradient Descent Path')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7e05fe501810>
```



h. Do simple k-fold and repeated k-fold. Compute error metrics ME, MAE, MSE, RMSE and compare.

```
from sklearn.model_selection import KFold, RepeatedKFold
from sklearn.metrics import mean_absolute_error, mean_squared_error


n_splits = 7
n_repeats = 5

a, b, c, rc = [], [], [], []

X = db['studytime'].values.reshape(-1, 1)
y = db['Grade'].values.reshape(-1, 1)


kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
for tr, te in kf.split(X):
    X_train, X_test = X[tr], X[te]
    y_train, y_test = y[tr], y[te]

    regressor = LinearRegression()
    regressor.fit(X_train, y_train)
    y_pred = regressor.predict(X_test)

    me = np.mean(y_test - y_pred)
    a.append(me)
    mae = mean_absolute_error(y_test, y_pred)
    b.append(mae)
    mse = mean_squared_error(y_test, y_pred)
    c.append(mse)
    rmse = np.sqrt(mse)
    rc.append(rmse)

me, mae, mse, rmse = np.mean(a), np.mean(b), np.mean(c), np.mean(rc)
print("Simple K-Fold:")
print("ME: ", me, "MAE: ",mae,"MSE: ",mse,"RMSE: ",rmse)
```

```
Simple K-Fold:
ME:  -0.0037840739282395475 MAE:  2.9731256142021385 MSE:  13.525221129907312 RMSE:  3.6720114116473206
```

```
rkf = RepeatedKFold(n_splits=n_splits, n_repeats=n_repeats, random_state=42)
ar, br, cr, rcr = [],[],[],[]

for train_index, test_index in rkf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    regressor = LinearRegression()
    regressor.fit(X_train, y_train)
    y_pred = regressor.predict(X_test)

    me = np.mean(y_test - y_pred)
    ar.append(me)
    mae = mean_absolute_error(y_test, y_pred)
    br.append(mae)
    mse = mean_squared_error(y_test, y_pred)
    cr.append(mse)
    rmse = np.sqrt(mse)
    rcr.append(rmse)

mer, maer, mser, rmser = np.mean(ar), np.mean(br), np.mean(cr), np.mean(rcr)
print("\nRepeated K-Fold Metrics:")
print("ME: ",mer, " MAE: ",maer, " MSE: ",mser, " RMSE: ",rmser)
```