جامعة الملك سعود
King Saud University

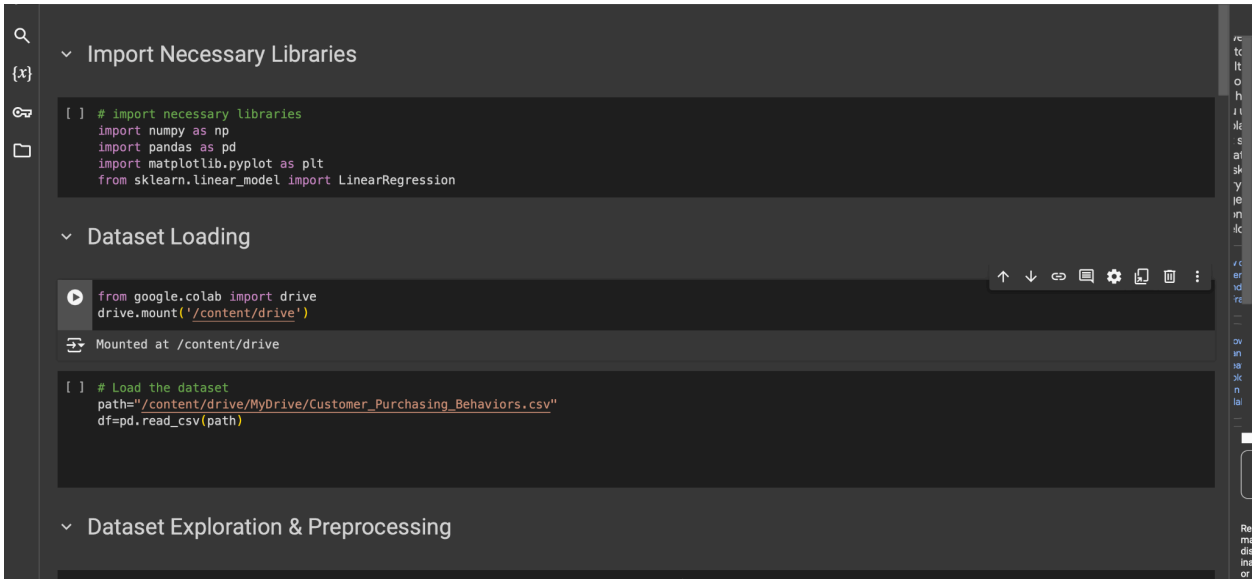
Machine Learning HW1

Yara Alfouzan
443203087

DATA SET DESCRIPTION:

Customer Purchasing Behaviors dataset. The specific features included are userr_id, age, annual_income, purchase_amount, purchase_frequency, region, and loyalty_score. This dataset can be used to analyze and predict customer purchasing behaviors. In this assignment we will be using it to explore the relationship between annual income and purchase frequency and to build a linear regression model to predict purchase frequency based on annual income.

Clear screenshots of the code, evaluations and outputs:



+ Code + Text Connecting Gemini

Dataset Exploration & Preprocessing

```
# Explore and preprocess the dataset
# 1. Display the dimensions of the dataset.
print("dataset dimensions")
print(df.shape)


# 2. Display the first ten columns of the dataset.
print(df.head(10))

# 3. Display the summary statistics of the dataset.
print(df.describe())

# 4. Impute missing values with the mean of the column using the function fillna.
df = df.fillna(df.select_dtypes(include=['number']).mean())

# 5. Select annual income as the input feature (X) and purchase frequency as the target variable (y).
X = df['annual_income']
y = df['purchase_frequency']


# 6. Draw a scatter plot showing the input feature X against the target y.
plt.scatter(X, y)
plt.xlabel('annual_income')
plt.ylabel('purchase_frequency')
plt.title('Scatter Plot of annual_income vs purchase_frequency')
plt.show()

# 7. Normalize the input feature.
X_normalized = (X - X.mean()) / X.std()
print(X_normalized)
```

colab.research.google.com

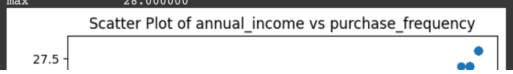
+ Code + Text RAM Disk Gemini

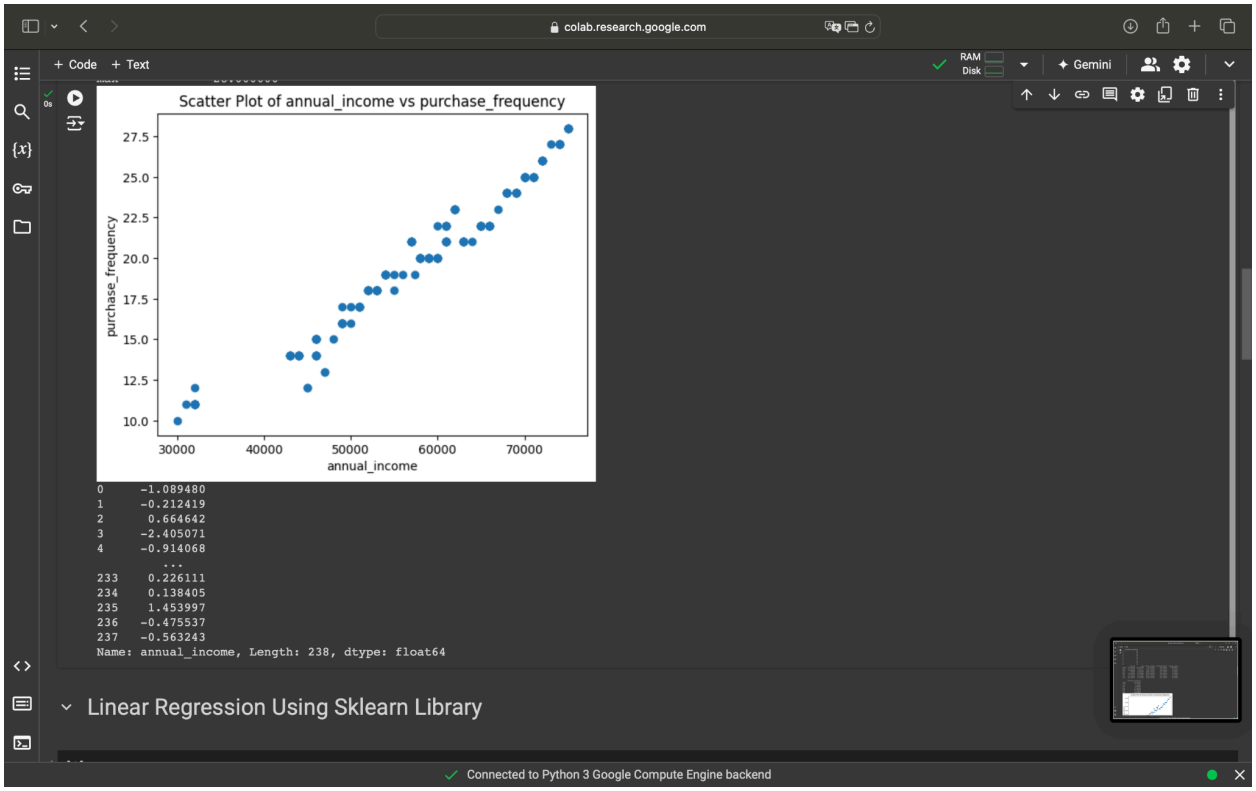
```
dataset dimensions
(238, 7)
     user_id  age  annual_income  purchase_amount  loyalty_score region  \
0          1   25   45000.000000              200            4.5  North
1          2   34   55000.000000              350            7.0  South
2          3   45   65000.000000              500            8.0   West
3          4   22   30000.000000              150            3.0   East
4          5   29   47000.000000              220            4.8  North
5          6   41   61000.000000              480            7.8  South
6          7   36   57421.940928              400            6.5   West
7          8   27   43000.000000              230            4.2   East
8          9   50   70000.000000              600            9.0  North
9         10   31   50000.000000              320            5.5  South

   purchase_frequency
0                  12
1                  18
2                  22
3                  10
4                  13
5                  21
6                  19
7                  14
8                  25
9                  17
          user_id         age  annual_income  purchase_amount  loyalty_score  \
count  238.000000  238.000000     238.000000       238.000000     238.000000
mean   119.500000   38.676471   57421.940928       425.630252       6.794118
std     68.848868    9.351118   11401.718337       140.052062       1.899047
min      1.000000   22.000000   30000.000000       150.000000       3.000000
25%     60.250000   31.000000   50000.000000       320.000000       5.500000
50%    119.500000   39.000000   59000.000000       440.000000       7.000000
75%    178.750000   46.750000   66750.000000       527.500000       8.275000
max    238.000000   55.000000   75000.000000       640.000000       9.500000

       purchase_frequency
count          238.000000
mean            19.798319
std              4.562884
min             10.000000
25%             17.000000
50%             20.000000
75%             23.000000
max             28.000000
```

Scatter Plot of annual_income vs purchase_frequency

Connected to Python 3 Google Compute Engine backend

```
max    238.000000   55.000000   75000.000000   640.000000   9.500000

       purchase_frequency
count          238.000000
mean            19.798319
std              4.562884
min             10.000000
25%             17.000000
50%             20.000000
75%             23.000000
max             28.000000
```

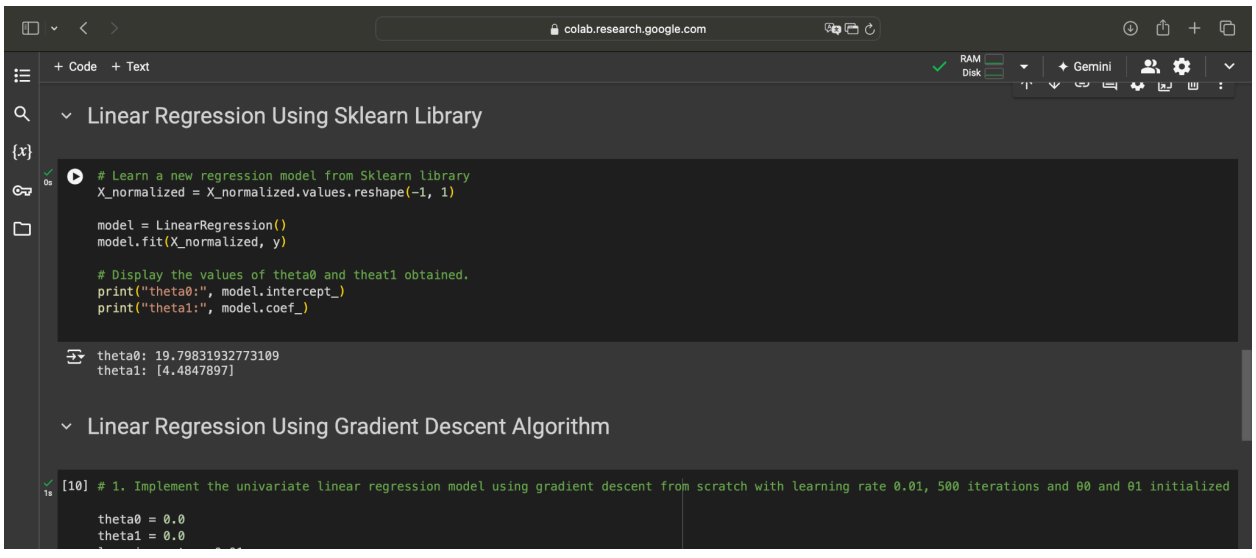





```
0     -1.089480
1     -0.212419
2      0.664642
3     -2.405071
4     -0.914068
         ...
233    0.226111
234    0.138405
235    1.453997
236   -0.475537
237   -0.563243
Name: annual_income, Length: 238, dtype: float64
```

Linear Regression Using Sklearn Library

Connected to Python 3 Google Compute Engine backend



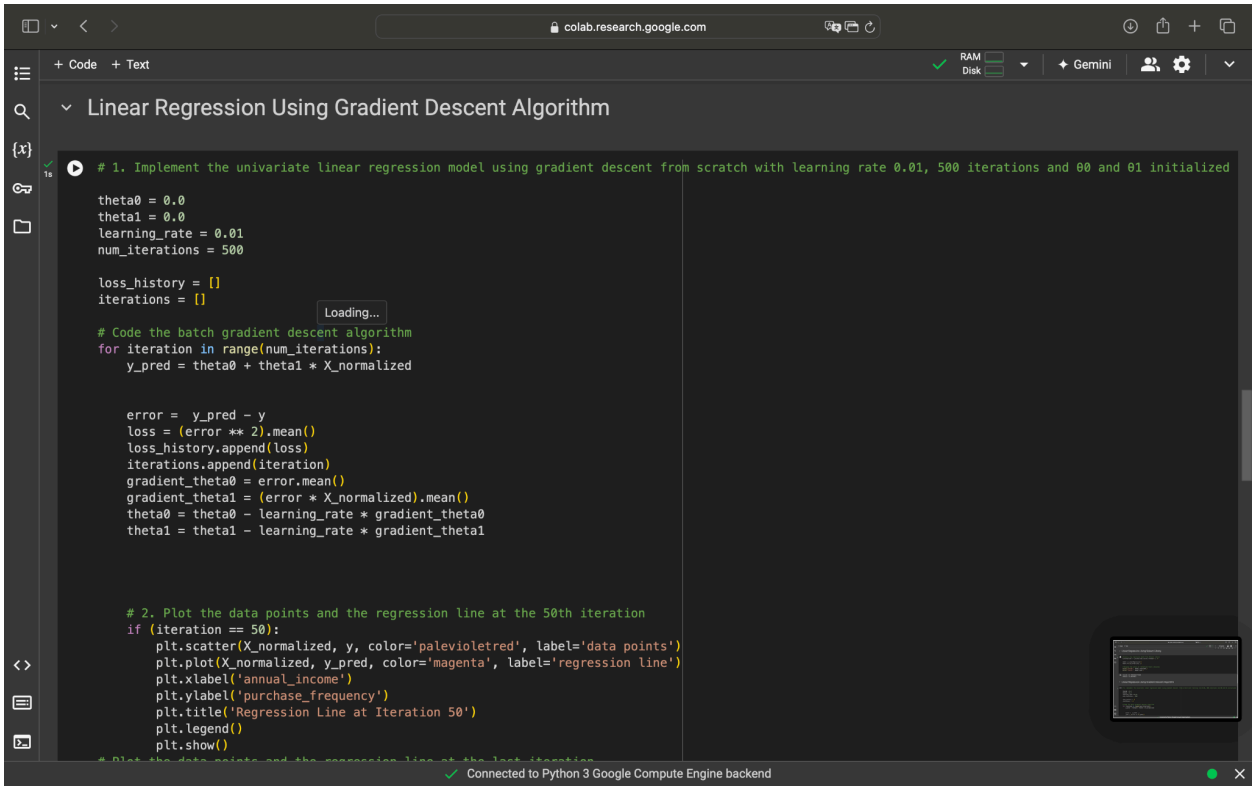
Linear Regression Using Sklearn Library

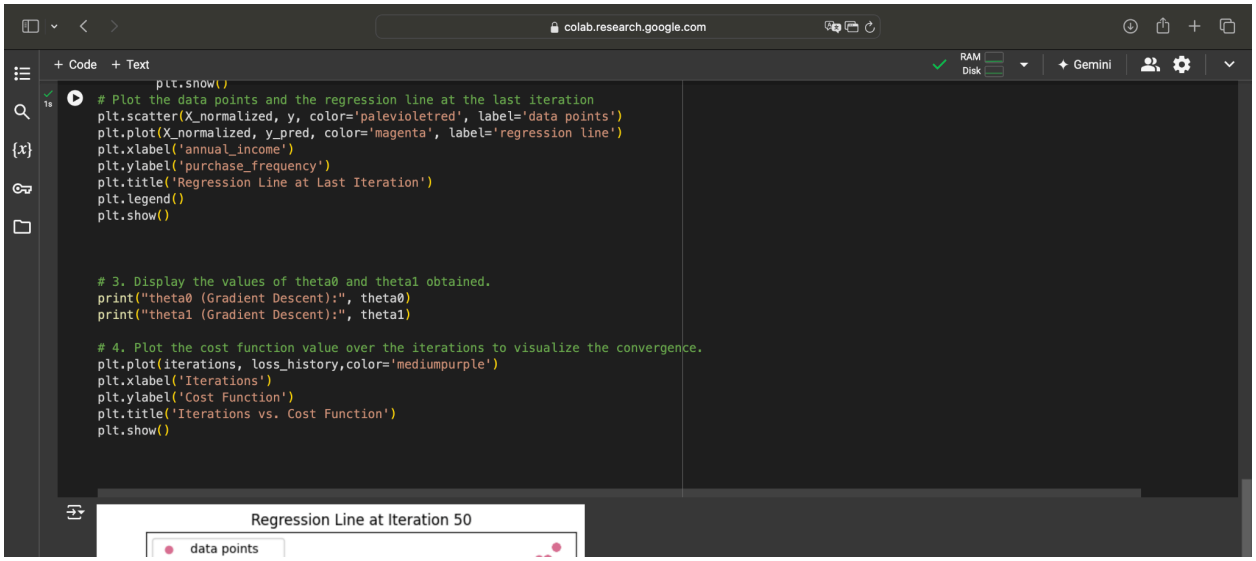
```
# Learn a new regression model from Sklearn library
X_normalized = X_normalized.values.reshape(-1, 1)

model = LinearRegression()
model.fit(X_normalized, y)

# Display the values of theta0 and theat1 obtained.
print("theta0:", model.intercept_)
print("theta1:", model.coef_)
```

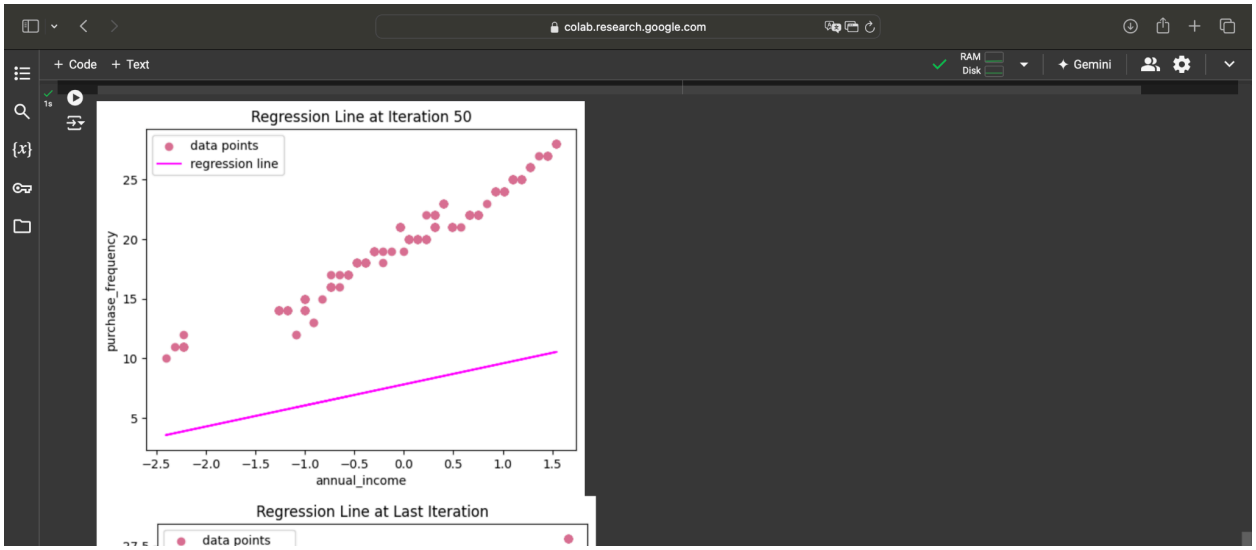
```
theta0: 19.79831932773109
theta1: [4.4847897]
```

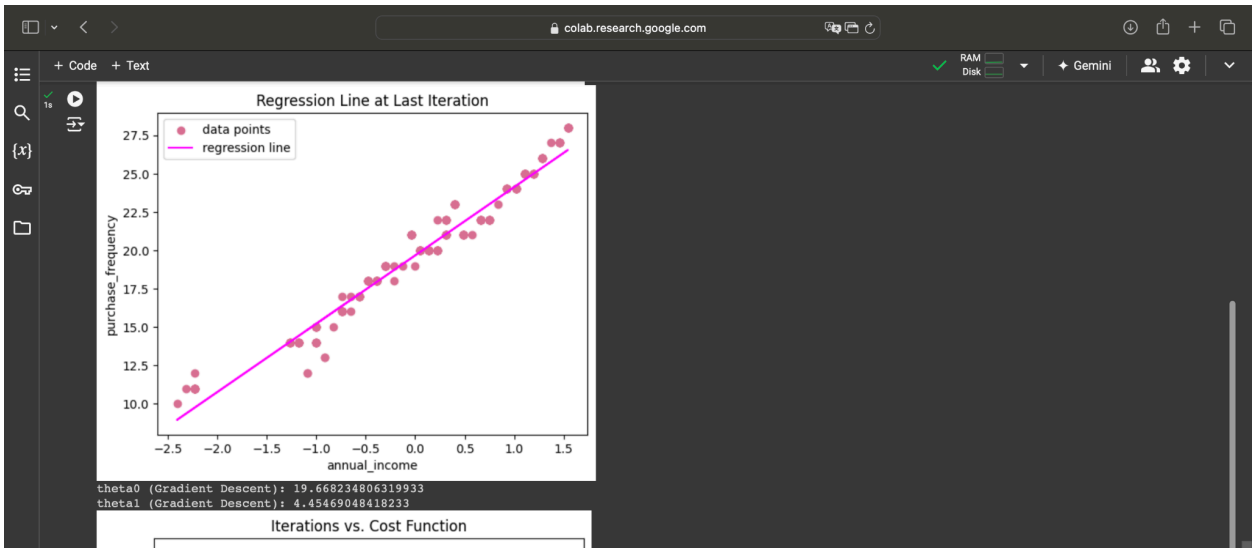
Linear Regression Using Gradient Descent Algorithm

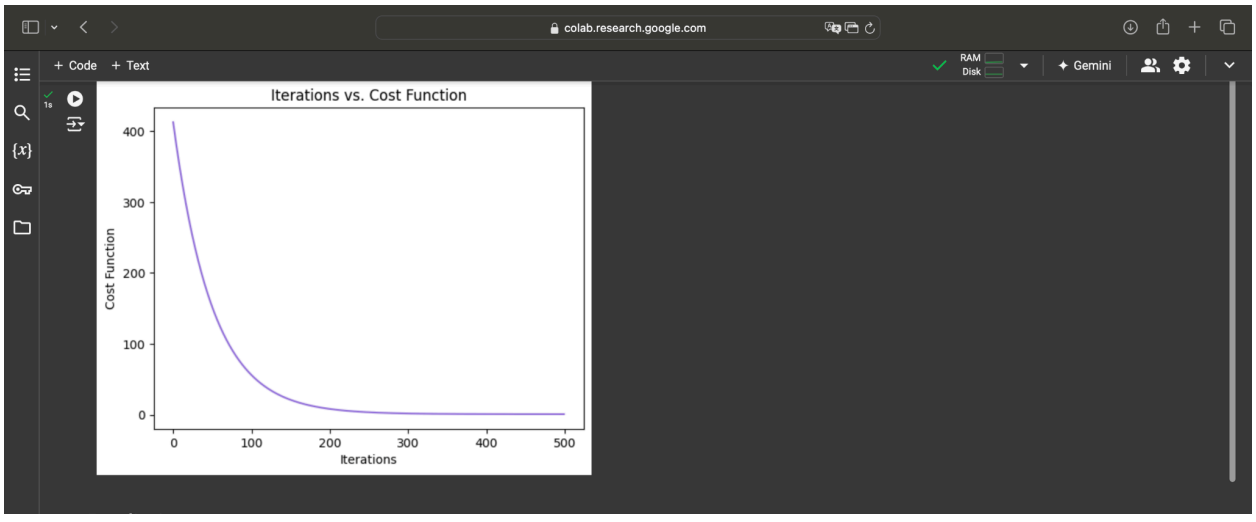
```
[10] # 1. Implement the univariate linear regression model using gradient descent from scratch with learning rate 0.01, 500 iterations and 00 and 01 initialized

     theta0 = 0.0
     theta1 = 0.0
```

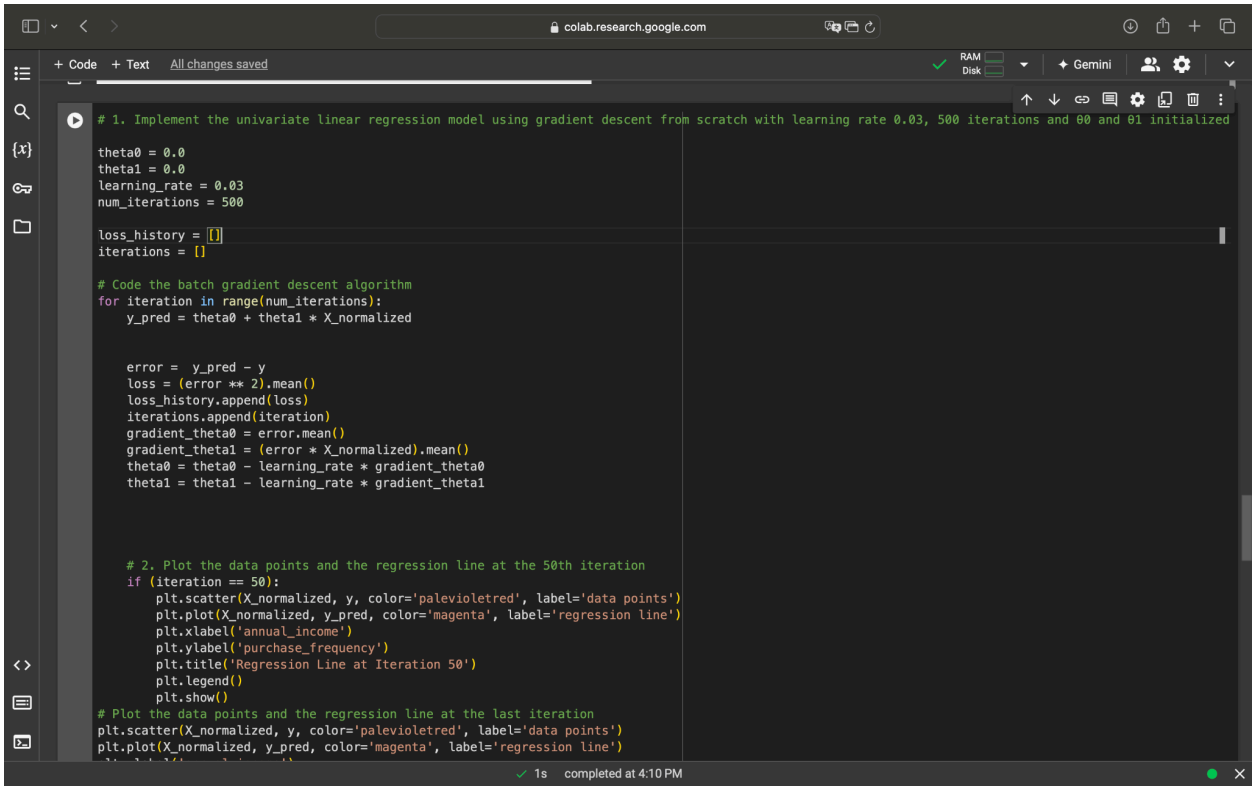

colab.research.google.com
+ Code
+ Text
RAM
Disk
Gemini
Linear Regression Using Gradient Descent Algorithm
# 1. Implement the univariate linear regression model using gradient descent from scratch with learning rate 0.01, 500 iterations and 00 and 01 initialized
theta0 = 0.0
theta1 = 0.0
learning_rate = 0.01
num_iterations = 500
loss_history = []
iterations = []
Loading...
# Code the batch gradient descent algorithm
for iteration in range(num_iterations):
y_pred = theta0 + theta1 * X_normalized
error = y_pred - y
loss = (error ** 2).mean()
loss_history.append(loss)
iterations.append(iteration)
gradient_theta0 = error.mean()
gradient_theta1 = (error * X_normalized).mean()
theta0 = theta0 - learning_rate * gradient_theta0
theta1 = theta1 - learning_rate * gradient_theta1
# 2. Plot the data points and the regression line at the 50th iteration
if (iteration == 50):
plt.scatter(X_normalized, y, color='palevioletred', label='data points')
plt.plot(X_normalized, y_pred, color='magenta', label='regression line')
plt.xlabel('annual_income')
plt.ylabel('purchase_frequency')
plt.title('Regression Line at Iteration 50')
plt.legend()
plt.show()
Connected to Python 3 Google Compute Engine backend

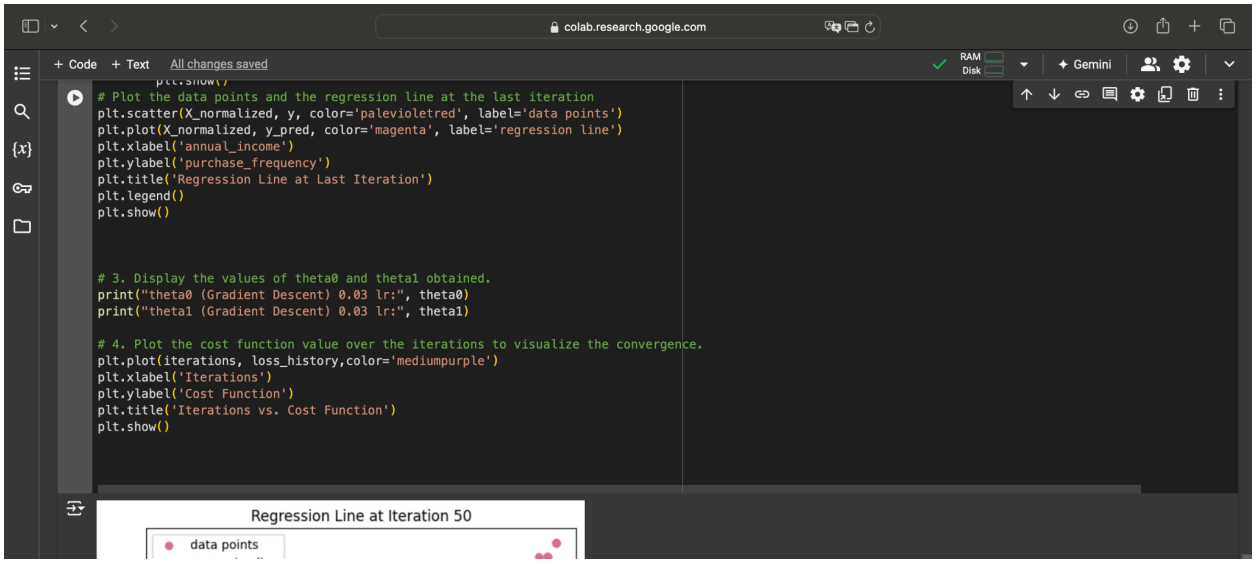

colab.research.google.com
+ Code
+ Text
RAM
Disk
Gemini
plt.show()
# Plot the data points and the regression line at the last iteration
plt.scatter(X_normalized, y, color='palevioletred', label='data points')
plt.plot(X_normalized, y_pred, color='magenta', label='regression line')
plt.xlabel('annual_income')
plt.ylabel('purchase_frequency')
plt.title('Regression Line at Last Iteration')
plt.legend()
plt.show()
# 3. Display the values of theta0 and theta1 obtained.
print("theta0 (Gradient Descent):", theta0)
print("theta1 (Gradient Descent):", theta1)
# 4. Plot the cost function value over the iterations to visualize the convergence.
plt.plot(iterations, loss_history,color='mediumpurple')
plt.xlabel('Iterations')
plt.ylabel('Cost Function')
plt.title('Iterations vs. Cost Function')
plt.show()
Regression Line at Iteration 50
data points

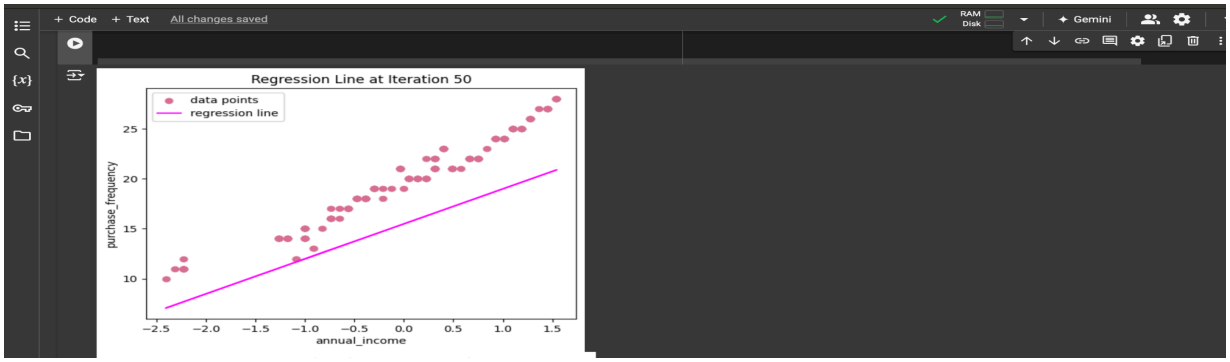

colab.research.google.com
+ Code
+ Text
RAM
Disk
Gemini
Regression Line at Iteration 50
data points
regression line
purchase_frequency
annual_income
Regression Line at Last Iteration
data points

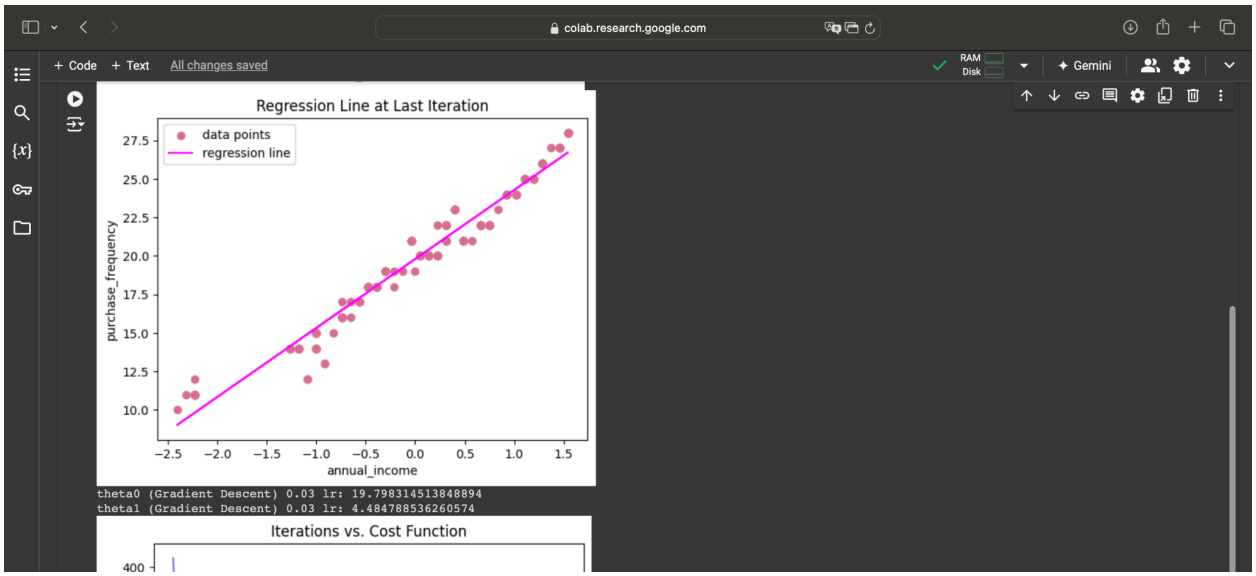

colab.research.google.com
+ Code
+ Text
RAM
Disk
Gemini
Regression Line at Last Iteration
data points
regression line
purchase_frequency
annual_income
theta0 (Gradient Descent): 19.668234806319933
theta1 (Gradient Descent): 4.45469048418233
Iterations vs. Cost Function

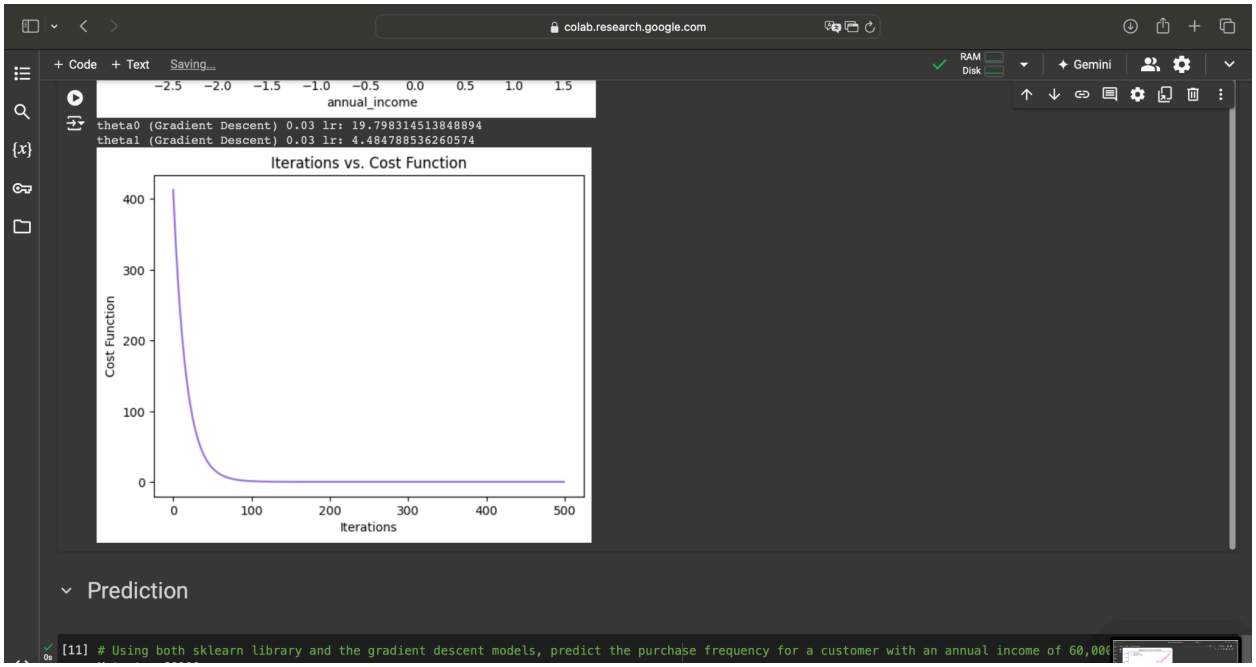

colab.research.google.com
+ Code
+ Text
RAM
Disk
Gemini
Iterations vs. Cost Function
Cost Function
Iterations

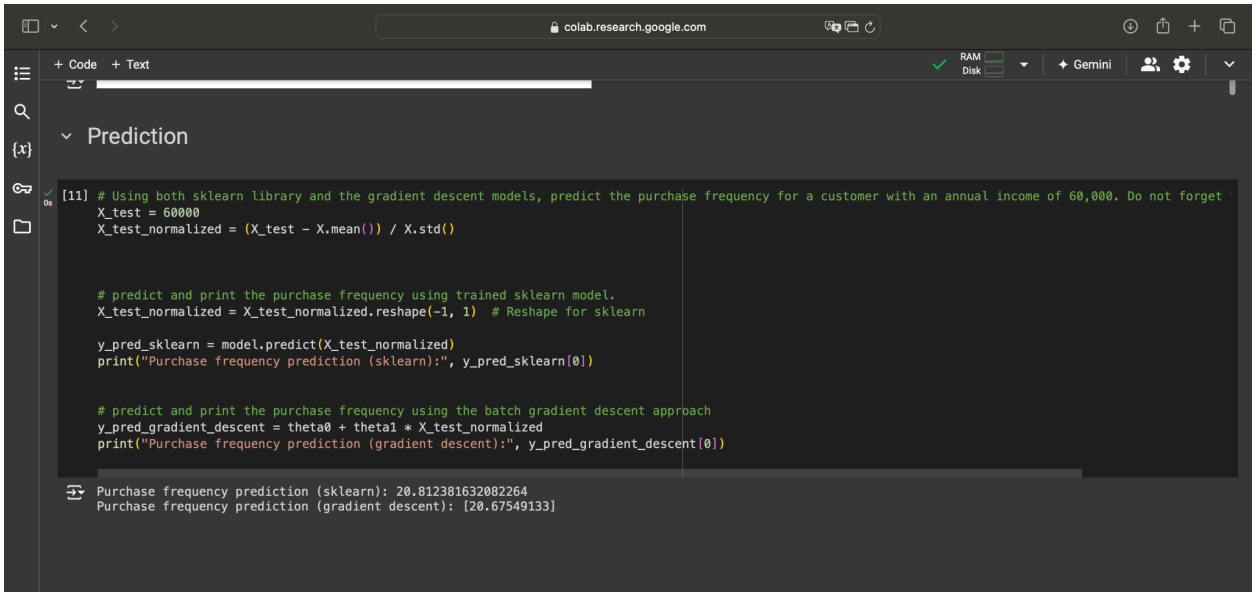

colab.research.google.com
+ Code
+ Text
All changes saved
RAM
Disk
Gemini
# 1. Implement the univariate linear regression model using gradient descent from scratch with learning rate 0.03, 500 iterations and θ0 and θ1 initialized
theta0 = 0.0
theta1 = 0.0
learning_rate = 0.03
num_iterations = 500
loss_history = []
iterations = []
# Code the batch gradient descent algorithm
for iteration in range(num_iterations):
y_pred = theta0 + theta1 * X_normalized
error = y_pred - y
loss = (error ** 2).mean()
loss_history.append(loss)
iterations.append(iteration)
gradient_theta0 = error.mean()
gradient_theta1 = (error * X_normalized).mean()
theta0 = theta0 - learning_rate * gradient_theta0
theta1 = theta1 - learning_rate * gradient_theta1
# 2. Plot the data points and the regression line at the 50th iteration
if (iteration == 50):
plt.scatter(X_normalized, y, color='palevioletred', label='data points')
plt.plot(X_normalized, y_pred, color='magenta', label='regression line')
plt.xlabel('annual_income')
plt.ylabel('purchase_frequency')
plt.title('Regression Line at Iteration 50')
plt.legend()
plt.show()
# Plot the data points and the regression line at the last iteration
plt.scatter(X_normalized, y, color='palevioletred', label='data points')
plt.plot(X_normalized, y_pred, color='magenta', label='regression line')
1s
completed at 4:10 PM


colab.research.google.com
+ Code
+ Text
All changes saved
RAM
Disk
Gemini
# Plot the data points and the regression line at the last iteration
plt.scatter(X_normalized, y, color='palevioletred', label='data points')
plt.plot(X_normalized, y_pred, color='magenta', label='regression line')
plt.xlabel('annual_income')
plt.ylabel('purchase_frequency')
plt.title('Regression Line at Last Iteration')
plt.legend()
plt.show()
# 3. Display the values of theta0 and theta1 obtained.
print("theta0 (Gradient Descent) 0.03 lr:", theta0)
print("theta1 (Gradient Descent) 0.03 lr:", theta1)
# 4. Plot the cost function value over the iterations to visualize the convergence.
plt.plot(iterations, loss_history,color='mediumpurple')
plt.xlabel('Iterations')
plt.ylabel('Cost Function')
plt.title('Iterations vs. Cost Function')
plt.show()
Regression Line at Iteration 50
data points


+ Code
+ Text
All changes saved
RAM
Disk
Gemini
Regression Line at Iteration 50
data points
regression line
purchase_frequency
annual_income


colab.research.google.com
+ Code
+ Text
All changes saved
RAM
Disk
Gemini
Regression Line at Last Iteration
data points
regression line
purchase_frequency
annual_income
theta0 (Gradient Descent) 0.03 lr: 19.798314513848894
theta1 (Gradient Descent) 0.03 lr: 4.484788536260574
Iterations vs. Cost Function


colab.research.google.com
+ Code
+ Text
Saving...
RAM
Disk
Gemini
annual_income
theta0 (Gradient Descent) 0.03 lr: 19.798314513848894
theta1 (Gradient Descent) 0.03 lr: 4.484788536260574
Iterations vs. Cost Function
Cost Function
Iterations
Prediction
[11] # Using both sklearn library and the gradient descent models, predict the purchase frequency for a customer with an annual income of 60,000



A table displaying printed results (e.g., model parameters and/or predicted values):

LR=0.01

LR=0.03

Sklearn library:

The sklearn library uses a closed-form solution to calculate the optimal parameters, while gradient descent iteratively updates the parameters until a stopping criterion is met.When the learning rate is increased to 0.03, the gradient descent algorithm converges faster and may reach a solution closer to the optimal solution obtained by the sklearn library.

Screenshots of the required scatterplots. The gradient descent algorithm converge around iteration 150 with a learning rate of 0.01, where the cost function reaches a relatively stable minimum value.After changing the learning rate to 0.03, the gradient descent appears to converge much faster, around iteration 50.