

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:

The screenshot shows a Jupyter Notebook interface with three main sections of code:

- Import Necessary Libraries**:

```
[ ] # import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```
- Dataset Loading**:

```
[ ] # Load the dataset
path='/content/drive/MyDrive/Customer_Purchasing_Behaviors.csv'
df=pd.read_csv(path)
```

A status message indicates "Mounted at /content/drive".
- Dataset Exploration & Preprocessing**: (This section is currently empty in the screenshot.)

The notebook has a dark theme and includes standard Jupyter controls like cell selection, execution, and output display.

+ 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

RAM Disk | Gemini | ☰

```

dataset dimensions
(238, 7)
user_id    age    annual_income  purchase_amount  loyalty_score  region \
0         1     25.000000      200.000000       4.5        North
1         2     34.000000      350.000000       7.0       South
2         3     45.000000      500.000000       8.0       West
3         4     22.000000      150.000000       3.0       East
4         5     29.000000      220.000000       4.8        North
5         6     41.000000      480.000000       7.8       South
6         7     36.57421.940928  400.000000       6.5       West
7         8     27.43000.000000  230.000000       4.2       East
8         9     50.70000.000000  600.000000       9.0        North
9        10    31.50000.000000  320.000000       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

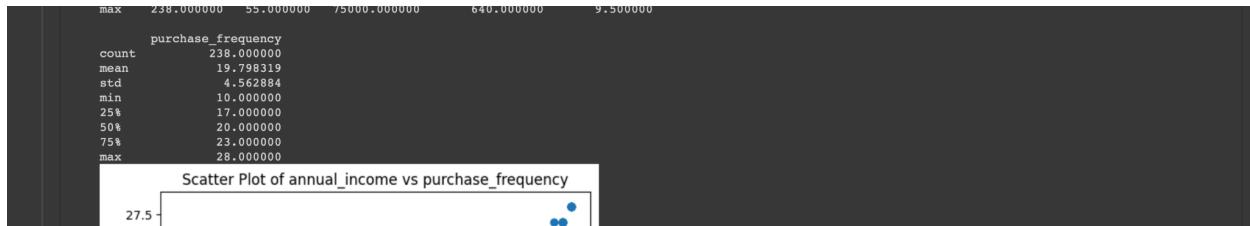
user_id    age    annual_income  purchase_amount  loyalty_score \
count  238.000000  238.000000      238.000000  238.000000
mean   119.500000  38.676471    57421.940928  425.630252  6.794118
std    68.848868   9.351118   11401.718337  140.052062  1.899047
min    1.000000   22.000000   30000.000000  150.000000  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



Scatter Plot of annual_income vs purchase_frequency

The scatter plot includes a fitted regression line, indicating a positive linear relationship between the two variables.

```

annual_income = np.array([32000, 33000, 34000, 35000, 36000, 37000, 38000, 39000, 40000, 41000, 42000, 43000, 44000, 45000, 46000, 47000, 48000, 49000, 50000, 51000, 52000, 53000, 54000, 55000, 56000, 57000, 58000, 59000, 60000, 61000, 62000, 63000, 64000, 65000, 66000, 67000, 68000, 69000, 70000, 71000, 72000, 73000, 74000, 75000])
purchase_frequency = np.array([10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0])

# Create a scatter plot
plt.scatter(annual_income, purchase_frequency)
plt.title('Scatter Plot of annual_income vs purchase_frequency')
plt.xlabel('annual_income')
plt.ylabel('purchase_frequency')

# Fit a linear regression model
model = LinearRegression()
model.fit(annual_income.reshape(-1, 1), purchase_frequency)

# Print the intercept and slope
print("Intercept: ", model.intercept_)
print("Slope: ", model.coef_[0])

```

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 theta1 obtained.
print("theta0:", model.intercept_)
print("theta1:", model.coef_)

theta0: 19.79831932773109
theta1: [4.4847897]

```

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 theta0 and theta1 initialized
theta0 = 0.0
theta1 = 0.0

```

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 theta0 and theta1 initialized

theta0 = 0.0
theta1 = 0.0
learning_rate = 0.01
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')
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()
```

Connected to Python 3 Google Compute Engine backend

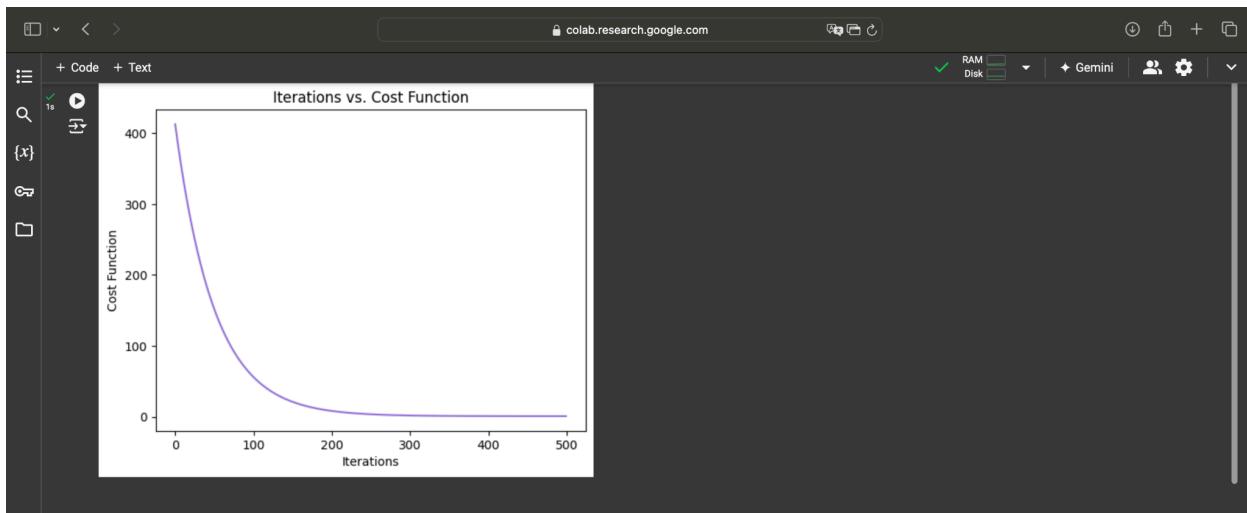
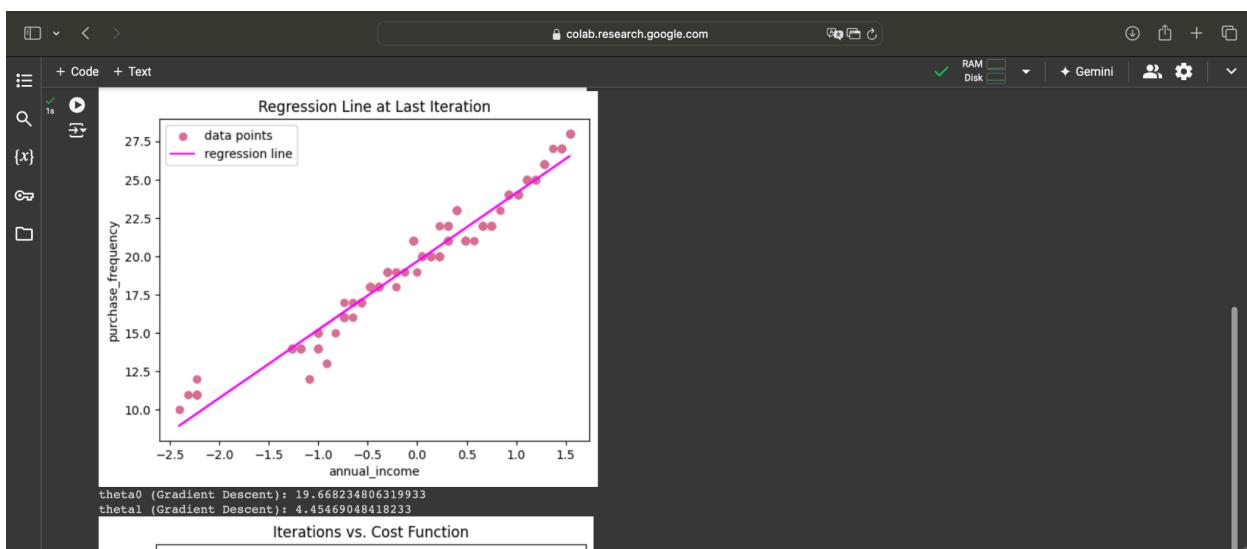
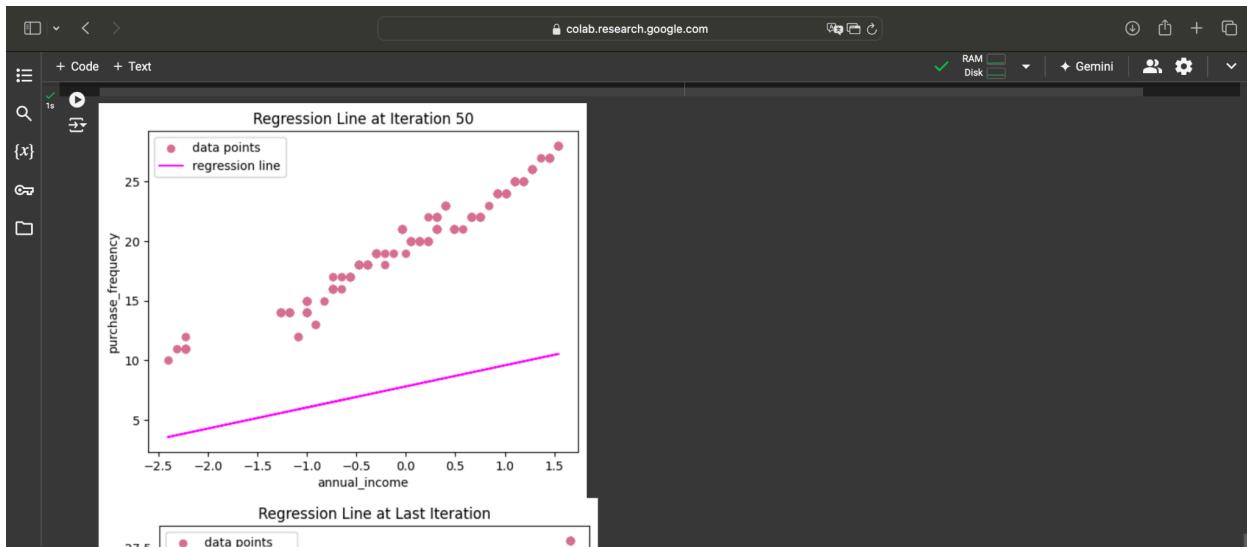
```
plt.show()

# 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()
```





colab.research.google.com

```
+ Code + Text All changes saved
```

```
# 1. Implement the univariate linear regression model using gradient descent from scratch with learning rate 0.03, 500 iterations and theta0 and theta1 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
```

```
plt.show()

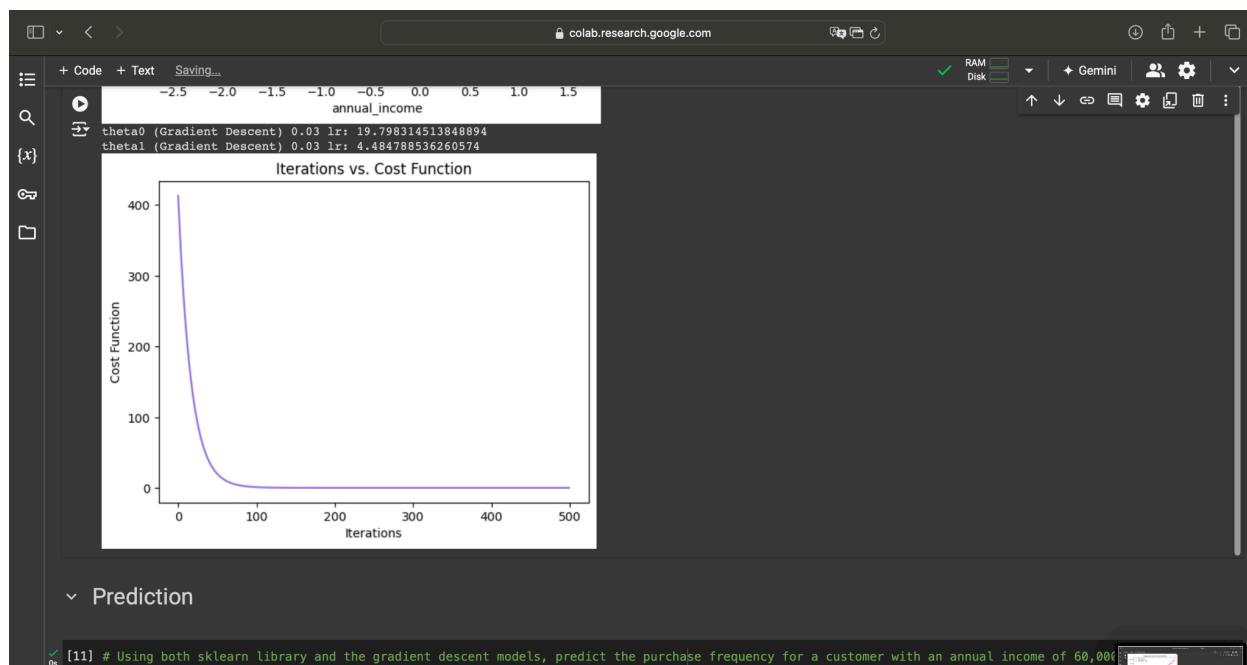
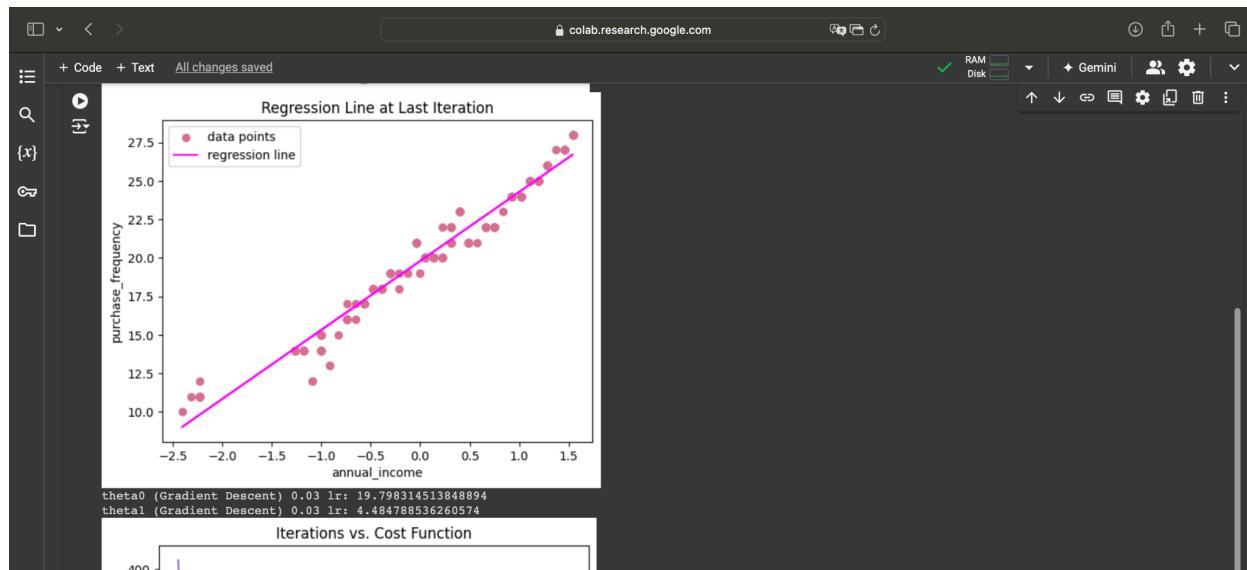
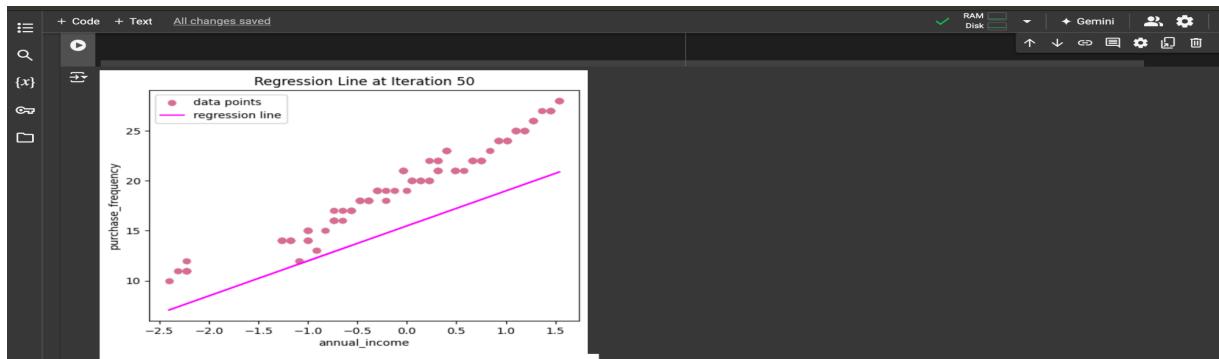
# Plot the data points and the regression line at the last iteration
plt.scatter(X_normalized, y, color='palevioletred', label='data points')
plt.plot(X_normalized, y_pred, color='magenta', label='regression line')
plt.xlabel('annual_income')
plt.ylabel('purchase_frequency')
plt.title('Regression Line at Last Iteration')
plt.legend()
plt.show()

# 3. Display the values of theta0 and theta1 obtained.
print("theta0 (Gradient Descent) 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

The figure shows a scatter plot titled "Regression Line at Iteration 50". It displays two sets of points: "data points" (palevioletred dots) and a "regression line" (magenta line). The x-axis is labeled "annual_income" and the y-axis is labeled "purchase_frequency". The regression line passes through the general trend of the data points.



A screenshot of a Jupyter Notebook interface on colab.research.google.com. The code cell contains Python code for predicting purchase frequency using both the sklearn library and gradient descent. The output shows the predicted values from both methods.

```
[11] # Using both sklearn library and the gradient descent models, predict the purchase frequency for a customer with an annual income of 60,000. Do not forget to import all required libraries at the top.
X_test = 60000
X_test_normalized = (X_test - X.mean()) / X.std()

# predict and print the purchase frequency using trained sklearn model.
X_test_normalized = X_test_normalized.reshape(-1, 1) # Reshape for sklearn
y_pred_sklearn = model.predict(X_test_normalized)
print("Purchase frequency prediction (sklearn):", y_pred_sklearn[0])

# predict and print the purchase frequency using the batch gradient descent approach
y_pred_gradient_descent = theta0 + theta1 * X_test_normalized
print("Purchase frequency prediction (gradient descent):", y_pred_gradient_descent[0])

Purchase frequency prediction (sklearn): 20.812381632082264
Purchase frequency prediction (gradient descent): [20.67549133]
```

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

LR=0.01

A screenshot of a Jupyter Notebook interface on colab.research.google.com. The code cell contains Python code for predicting purchase frequency using both the sklearn library and gradient descent. The output shows the predicted values from both methods.

```
theta0 (Gradient Descent): 19.668234806319933
theta1 (Gradient Descent): 4.45469048418233

Purchase frequency prediction (sklearn): 20.812381632082264
Purchase frequency prediction (gradient descent): [20.67549133]
```

LR=0.03

A screenshot of a Jupyter Notebook interface on colab.research.google.com. The code cell contains Python code for predicting purchase frequency using both the sklearn library and gradient descent. The output shows the predicted values from both methods.

```
theta0 (Gradient Descent) 0.03 lr: 19.798314513848894
theta1 (Gradient Descent) 0.03 lr: 4.484788536260574

Iterations vs. Cost Function

print("Purchase frequency prediction (gradient descent):", y_pred_gradient_descent[0])

Purchase frequency prediction (sklearn): 20.812381632082264
Purchase frequency prediction (gradient descent): [20.81237656]
```

Sklearn library:

A screenshot of a Jupyter Notebook interface on colab.research.google.com. The code cell contains Python code for predicting purchase frequency using the sklearn library. The output shows the predicted values from the sklearn library.

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

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.

