CS 6375
ASSIGNMENT ______1______

Names of students in your group:

Ching-Yi Chang - CXC190002

Xiaokai Rong - XXR230000

Number of free late days used: ________0________

Note: You are allowed a **total** of 4 free late days for the **entire semester**. You can use at most 2 for each assignment. After that, there will be a penalty of 10% for each late day.

Please list clearly all the sources/references that you have used in this assignment.

1 Linear Regression using Gradient Descent Coding in Python (75 points)

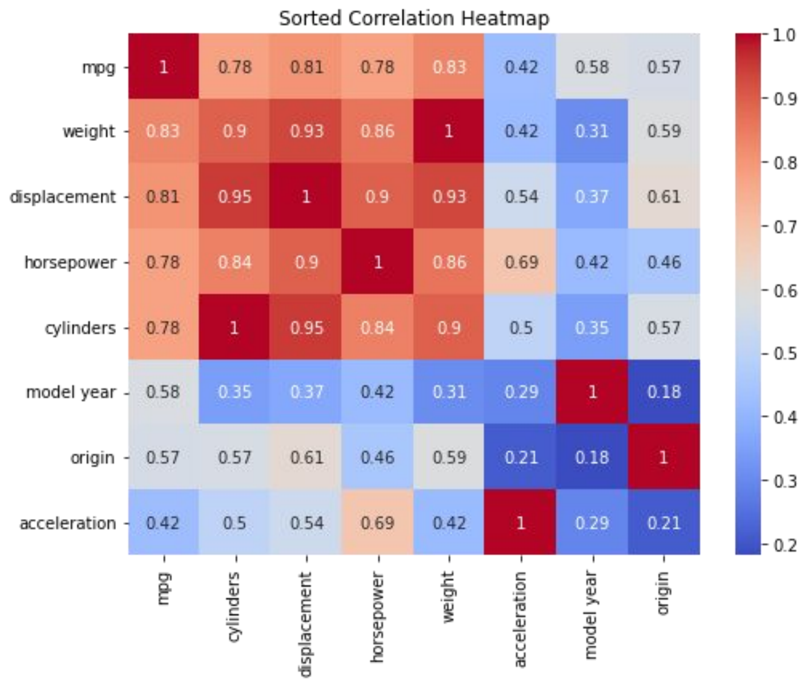
1. Choose a dataset suitable for regression from UCI ML Repository: - https://archive.ics.uci.edu/ml/datasets.php.

A: url='https://raw.githubusercontent.com/SparrowChang/CS6375_assignment1/main/auto%2Bmpg/auto-mpg.data', we use the dataset (auto MPG from UCI ML repository) and upload on Ching-Yi's Github. Overall could see https://github.com/SparrowChang/CS6375_assignment1

2. Pre-process your dataset. Pre-processing includes the following activities:

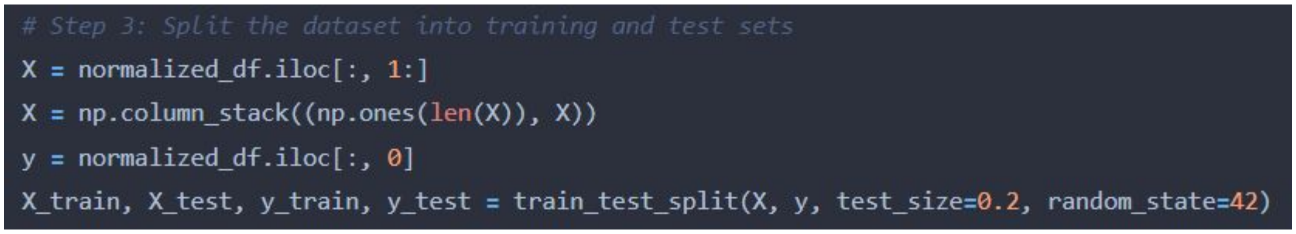
A:

- Rename column name by mapping dictionary (ex: mpg, cylinders, displacement...)
- Change data structures from object to numerical values (ex: horsepower)
- Drop the columns that are not relevant for the regression analysis (ex: car name)
- Normalize data (ex: StandardScaler())
- Remove null or NA values (ex: dropna())
- Remove redundant rows (ex: drop_duplicates())
- Draw a Correlation Heatmap. Y is the output, 'mpg' (mile per gallon), we could know the attitudes 'weight', 'displacement', 'horsepower' and 'cylinders' show higher correlation coefficient with the output 'mpg'.



3. After pre-processing split the dataset into training and test parts. It is up to you to choose the train/test ratio

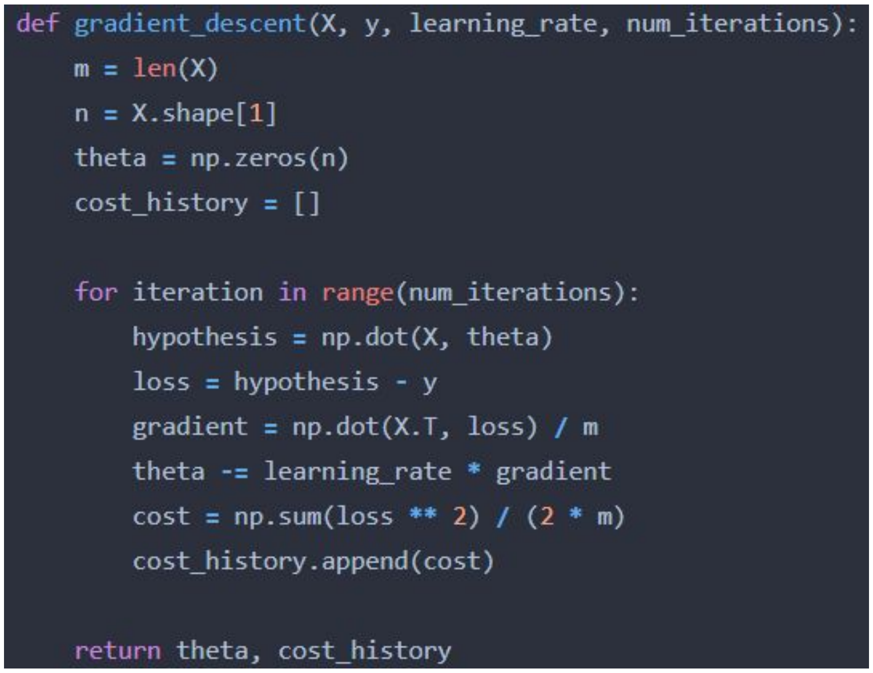
A: We choose train/test ratio = 80/20 by using train_test_split.



```
# Step 3: Split the dataset into training and test sets
X = normalized_df.iloc[:, 1:]
X = np.column_stack((np.ones(len(X)), X))
y = normalized_df.iloc[:, 0]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4. Use the training dataset to construct a linear regression model.

A: as below gradient_descent function.



```
def gradient_descent(X, y, learning_rate, num_iterations):
    m = len(X)
    n = X.shape[1]
    theta = np.zeros(n)
    cost_history = []

    for iteration in range(num_iterations):
        hypothesis = np.dot(X, theta)
        loss = hypothesis - y
        gradient = np.dot(X.T, loss) / m
        theta -= learning_rate * gradient
        cost = np.sum(loss ** 2) / (2 * m)
        cost_history.append(cost)

    return theta, cost_history
```

5. Apply the model you created in the previous step to the test part of the dataset. Report the test dataset error values for the best set of parameters obtained from previous part.

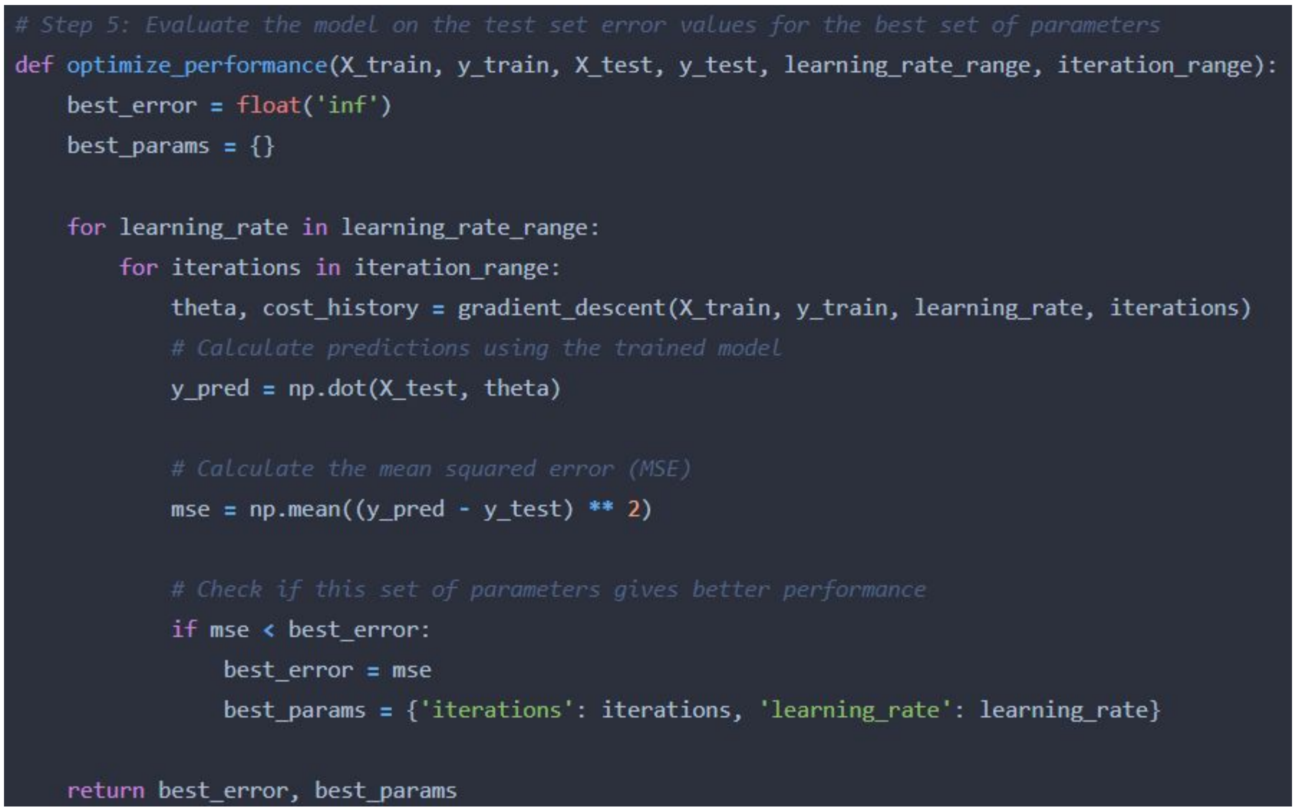
A: Consider 7 multiple attributes and think of the vector from of the weight update equation. We tune these parameters to achieve the optimum error value as below optimize_performance function.

iteration_range = [100, 500, 1000]

learning_rate_range = [0.001, 0.01, 0.1, 0.5]

We create a log file that indicates parameters used to get best error (MSE).

In log file (optimum test error value)
Best Mean squared error (MSE): 10.730665334334235 Best parameters: {'iterations': 1000, 'learning_rate': 0.1}



```python
# Step 5: Evaluate the model on the test set error values for the best set of parameters
def optimize_performance(X_train, y_train, X_test, y_test, learning_rate_range, iteration_range):
    best_error = float('inf')
    best_params = {}

    for learning_rate in learning_rate_range:
        for iterations in iteration_range:
            theta, cost_history = gradient_descent(X_train, y_train, learning_rate, iterations)
            # Calculate predictions using the trained model
            y_pred = np.dot(X_test, theta)

            # Calculate the mean squared error (MSE)
            mse = np.mean((y_pred - y_test) ** 2)

            # Check if this set of parameters gives better performance
            if mse < best_error:
                best_error = mse
                best_params = {'iterations': iterations, 'learning_rate': learning_rate}

    return best_error, best_params
```

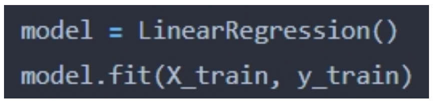
6. Answer this question: Are you satisfied that you have found the best solution? Explain.

A: Yes, we satisfied the result, MSE of gradient decent is around 10.73 which is close to MSE of ML libraries (sklearn.linear_model LinearRegression) 10.71.

2 Linear Regression using ML libraries (25 points)

A: From step 1 to step 3 is exactly the same as the 1st part.

4. The big difference from the 1st part, we use any ML library that performs linear regression from Scikit Learn package. https://scikit-learn.org

A: 

```python
model = LinearRegression()
model.fit(X_train, y_train)
```

5. Apply the model you created in the previous step to the test part of the dataset. Report the test dataset error values for the best set of parameters obtained from previous part.

A: We create a log file that indicates parameters used and error (MSE) value.

In log file (optimum test error value)
Mean squared error (MSE): 10.710864418838403

```
R2 score: 0.7901500386760344
Explained variance: 0.7915327760182195
```

6. Answer this question: Are you satisfied that you have found the best solution? Explain.

A: Yes, MSE is around 10.71 which is close to gradient descent method.