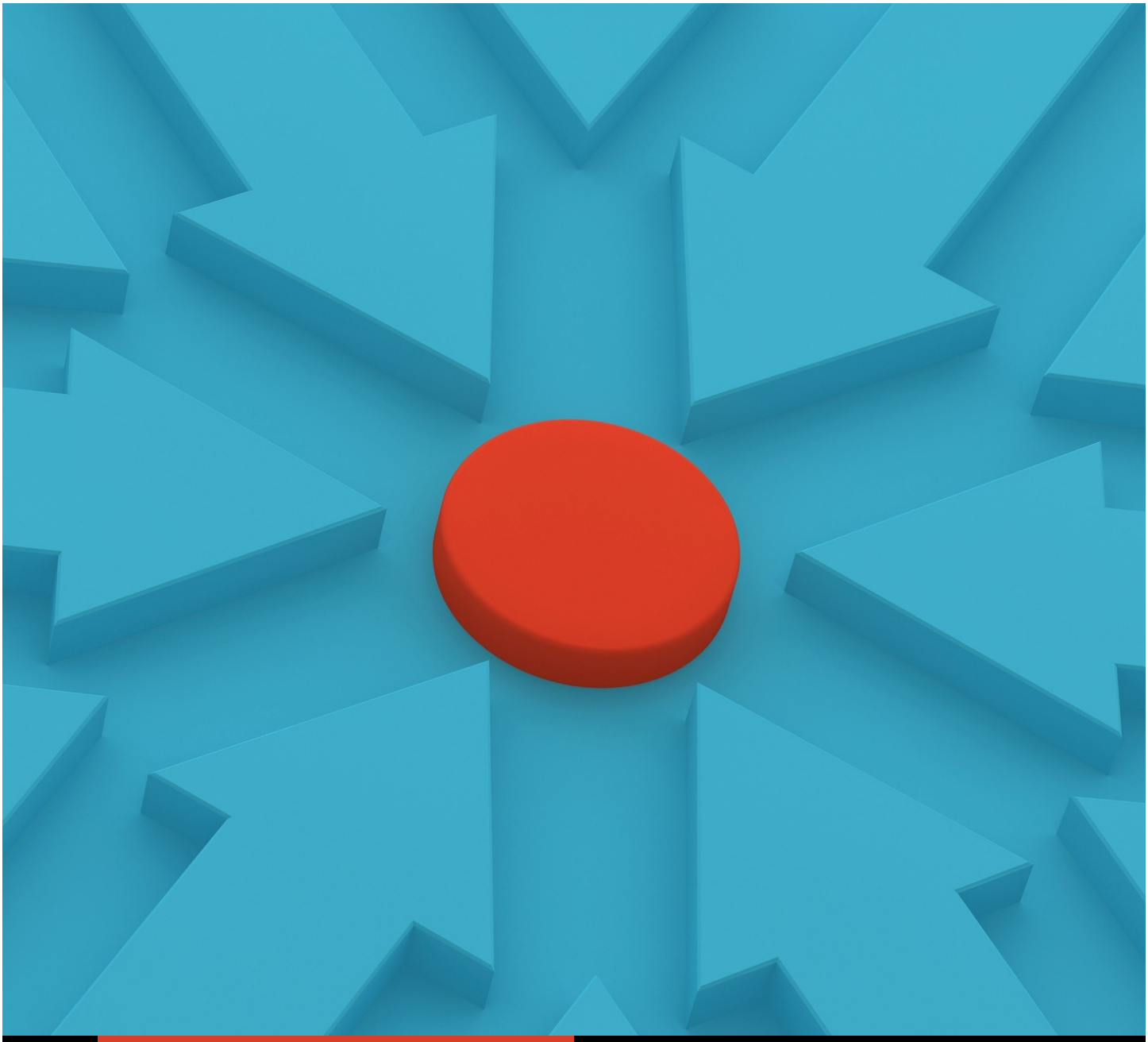
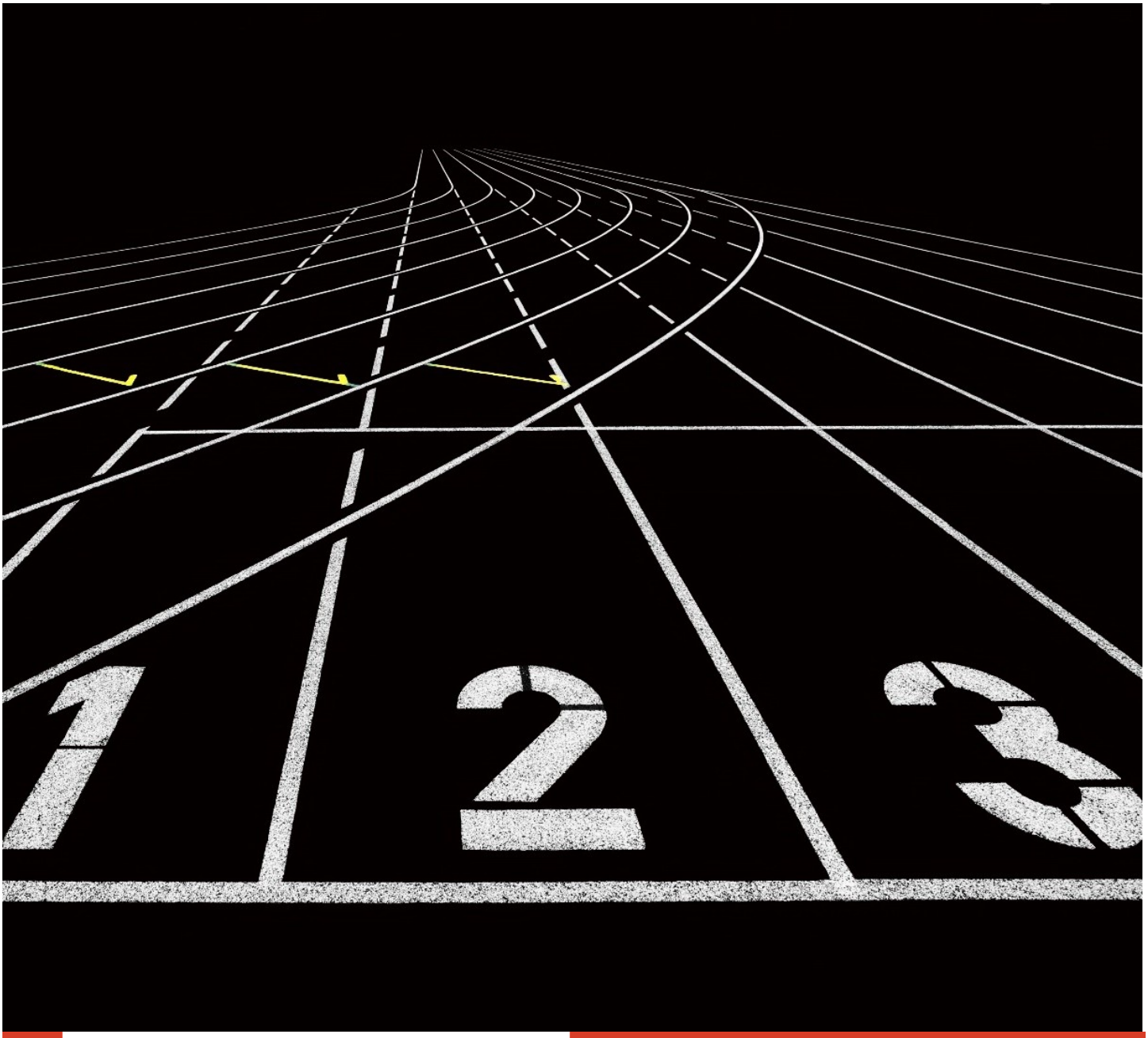


MACHINE LEARNING

ASSINGMENT 1

LAMAN KHUDADATZADA

Assignment 1



INTRODUCTION

The report presents the application and analysis of linear regression to predict car prices using a dataset from turbo.az. It outlines the steps taken to install, visualize, and implement linear regression both from scratch and using a library.

Importing libraries:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
```

Data Loading and Visualization:

Using the pandas library in Python the data is loaded and three relevant columns – Yurush, model Buraxilish ili and Qiymet are extracted from the dataset.

```
data = pd.read_csv("turboaz.csv")[["Yurush", "Buraxilish ili", "Qiymet"]]
data.head() # displaying few rows
```

- With the given path three columns are read.
- Few lines from the dataset are displayed.

```
def yurush_string_to_int(data):
    data['Yurush'] = data['Yurush'].str.replace(' km', '').str.replace(' ', '').astype(int)
```

```
yurush_string_to_int(data) #calling the convert function
data.head() #displaying
```

- This function converts the string values of “Yurush” into integers. Simply, it removes the word “km” from the data.
- Calling the function

```
def qiymet_dollar_to_azn(data):
    converted_qiymet = [float(price.replace(' $', '').strip()) * 1.7 if '$' in price else float(price.replace(' AZN', '').strip()) for price in data['Qiymet']]
    data['Qiymet'] = converted_qiymet
```

```
qiymet_dollar_to_azn(data)
data.head()
```

- This function converts the string values of “Qiymet” into integers. Simply, it removes the word “AZN” or “$” sign from the data.
- Calling the function and displaying the data

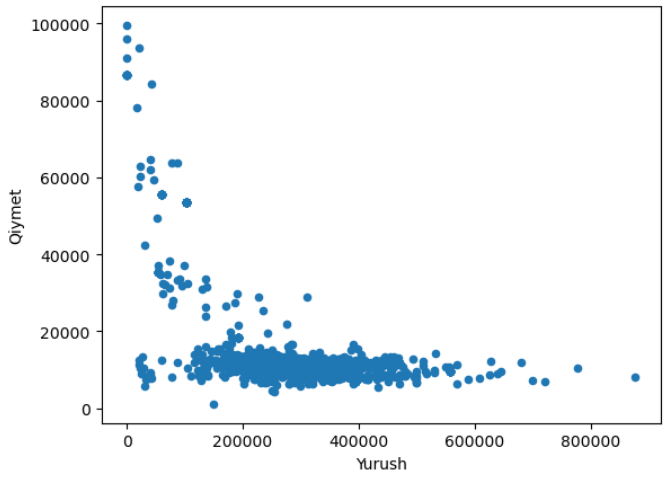
Visualization:

Visualizations are created using matplotlib, including scatter plots and a 3D plot.

Scatter plots using the pandas dataframe:

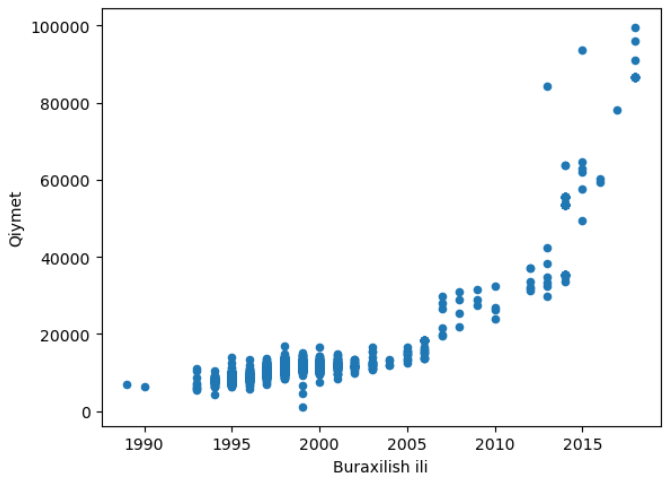
```
data.plot.scatter(x = "Yurush", y = "Qiymet")
```

The output:




```
data.plot.scatter(x = "Buraxilish ili", y = "Qiymet")
```

The output:



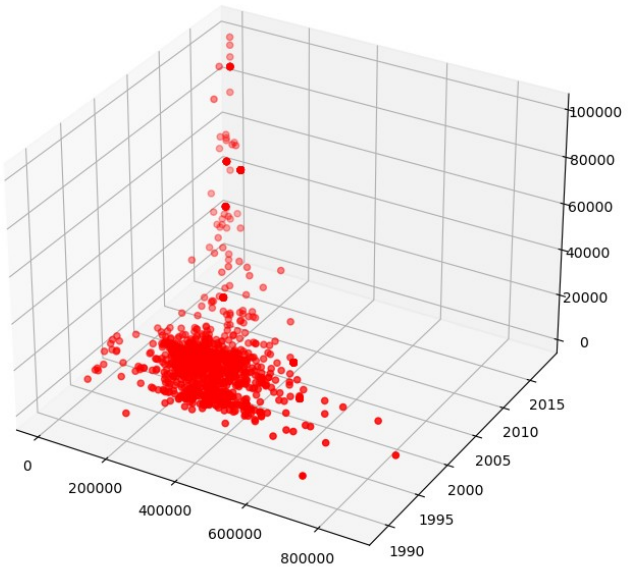

```
def three_d_plot(data):
    x1 = data['Yurush'].to_numpy()
    x2 =  data['Buraxilish ili'].to_numpy()
    y =  data['Qiymet'].to_numpy()

    fig = plt.figure(figsize = (10, 8))

    ax = plt.axes(projection ='3d')
    ax.scatter(x1, x2, y, c = 'red')
    plt.show()
```

- Extracting “Yurush”, “Buraxilish ili” and “Qiymet” columns.
- Creating a new figure with size of 10x8 inches.
- Then creating a scatter plot in 3D space.

```
three_d_plot(data)
```

- Calling the function

The output:



Linear Regression:

Here, we apply Z-score normalization in order to ensure convergence during gradient descent. Then using the gradient descent algorithm, the cost function is calculated.

```
x = data.drop(columns=['Qiymet']).values # the values of all columns in data except the 'Qiymet'
y = data['Qiymet'].values # the values of 'Qiymet' column

m, n = x.shape
theta = np.zeros((n+1))

# for matrix multipication
x_o = np.hstack((np.ones((m, 1)), x))
```

- Extracting all data except “Qiymet” column to the x variable.
- “Qiymet” column to the y variable
- Unpacking the shape of the array x into two variables m (rows) and n (columns).
- Creating a column vector of ones using np.ones((m, 1)), then it horizontally stacks using np.hstack().
- (m, n+1) is used for matrix multiplication during the training.

```
def multiply_matrix(x, theta):
    return np.dot(x, theta)
```

- The function is used for matrix multiplication between a matrix (x) and a vector (theta)

Cost Function:

```
def calc_cost(x, y, theta):
    m = len(y)
    hyp_values = multiply_matrix(x, theta)
    squared_errors = (hyp_values - y) ** 2
    total_cost = np.sum(squared_errors) / (2 * m)
    return total_cost
```

- m represents the number of samples.
- Calculating the predicted values (hyp_values) for the given matrix and vector by calling the multiply_matrix function.
- Calculating the squared errors
- Then, by summing up all the squared errors and dividing by twice the number of samples, it computes the total cost.

```
calc_cost(x_o, y, theta)
```

- Calling the function

The output:

```
207350411.8323343
```

Normalization:

```
def normalize(x):
```

```
    epsilon = 1e-10 # prevent division by zero
    sigma = x.std(0)
    mean = x.mean(0)
    sigma[sigma == 0] += epsilon
    normalized_x = (x - mean) / sigma
    return normalized_x, mean, sigma
```

- Normalization is a common technique used to standardize features by subtracting the mean and dividing by the standard deviation.
- Checking for any feature with zero standard deviation and adds a very small value epsilon.

```
x_normalized, mean, sigma = normalize(x)
x_no = np.hstack((np.ones((m, 1)), x_normalized))

calc_cost(x_no, y, theta)
```

- It returns the normalized feature matrix x_normalized, as well as the mean and standard deviation (stored in sigma) used for normalization.
- The next line creates a column vector of ones with the same number of rows then horizontally stacks it with x_normalized using np.hstack().
- calculates the cost function using the normalized feature matrix.

The output:

```
207350411.8323343
```

Gradient Descent:

```
learning_rate = 0.001
iterations = 10000
```

```
def gradient_desc(x, y, theta):
    n = len(y)
    cost_values =[]
    for i in range(iterations):
        hypo_cost = multiply_matrix(x, theta) - y
        theta = theta - (learning_rate / n) *
np.matmul(x.transpose(), hypo_cost)
        cost_values.append(calc_cost(x, y, theta))

    return theta, cost_values
```

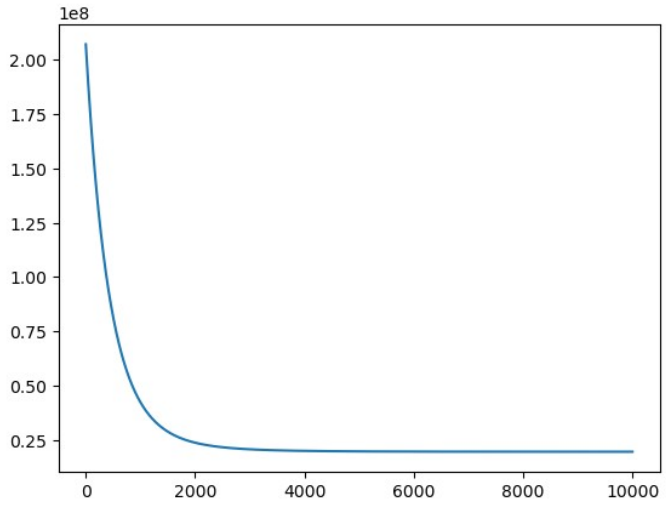
```
alt, cost_values = gradient_desc(x_no, y, theta)
```

- Initializing learning rate and the number of iterations,
- gradient descent to optimize the parameters of the linear regression model.
- Calculating the number of samples in the dataset.
- Initializing an empty list (cost_values) to store the cost values.
- Finding the difference between the predicted values and the target values.
- It returns the optimized parameter vector and the list of cost function values for each iteration.
- Then, calling the gradient descent function.

```
def cost_function_graph(vals):
    plt.plot(vals)
    plt.show()
cost_function_graph(cost_values)
```

- Displaying the cost function graph

The output:



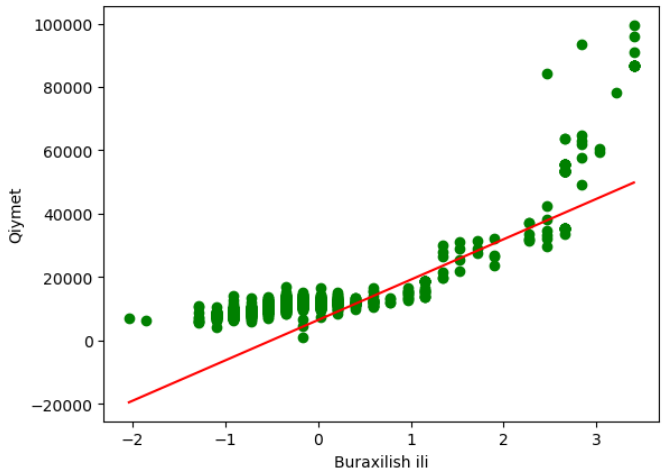
Line of predictions for Qiymet and Buraxilish ili:

```
def year_prediction(x, y, z):
    plt.scatter(x[:, 2], y, color = 'green')
    plt.xlabel('Buraxilish ili')
    plt.ylabel('Qiymet')

    #min and max values of 'Buraxilish ili'
```

```
    min_index = x[:,2].argmin()
    max_index = x[:,2].argmax()

    x_val = [x[:,2][min_index], x[:,2][max_index]]
    y_val = [np.matmul(x[min_index], z),
np.matmul(x[max_index], z)]

    # subtracting a constant value from y values
    offset = np.mean(y) - np.mean(y_val)
    y_values_adjusted = [val + offset for val in y_val]

    plt.plot(x_val, y_values_adjusted, color = "red")
```

```
year_prediction(x_no, y, alt)
```

The output:

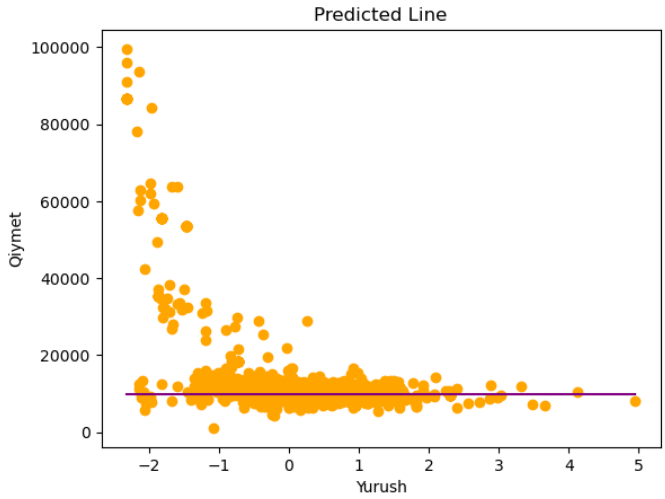


Line of predictions for Qiymet and Yurush:

```
def yurush_prediction(x, y):

    plt.scatter(x[:, 1], y, color = 'orange')
    plt.xlabel('Yurush')
    plt.ylabel('Qiymet')
    plt.title('Predicted Line')
    #min and max values of 'Yurush'
    min_index = x[:,1].argmin()
    max_index = x[:,1].argmax()

    x_value = [x[:,1][ min_index], x[:,1][max_index]]
    y_value = 10000

    plt.plot(x_value, [y_value, y_value], color = "purple")
```

- Setting labels for x and y axes
- Finding the index of the maximum and minimum values in the third column of the matrix
- The offset is used to get the predicted prices.
- Taking the predicted “Qiymet”s by adding the offset (already calculated)

The output:

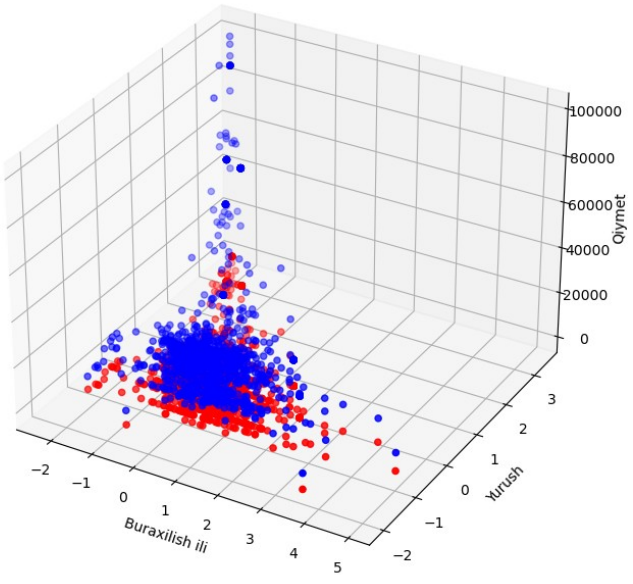


Line of predictions for Qiymet, Buraxilis ili and Yurush:

```
def scatter_predictions_3D(x, y, z):
    # Creating a figure
    fig = plt.figure(figsize = (12, 8))
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(x[:, 1], x[:, 2], y, color="blue")
    ax.scatter(x[:, 1], x[:, 2], np.matmul(x, theta),
color='red')
    ax.set_xlabel('Buraxilish ili')
    ax.set_ylabel('Yurush')
    ax.set_zlabel('Qiymet')
    plt.show()
scatter_predictions_3D(x_no, y, alt)
```

- It creates a new figure with a specified size (12x8).
- It adds a 3D subplot.
- Plotting the predicted data points using models on all axes.
- Setting labels for the x, y, and z axes
- Display the 3D scatter plot.

The output:



Two New Cars

```
car_1 = np.array([240000, 2000])
car_2 = np.array([415558, 1996])
```

```
normal_car_1 = (car_1 - mean) / sigma
normal_car_2 = (car_2 - mean) / sigma
```

```
normal_car_1 = np.concatenate(([1], normal_car_1))
normal_car_2 = np.concatenate(([1], normal_car_2))
```

- First, NumPy arrays represent the features of two new cars with their mileage and model years.
- It standardizes new cars using Z-score normalization.

```
price_car_1 = np.matmul(normal_car_1, alt)
price_car_2 = np.matmul(normal_car_2, alt)
```

- The predicted price for the first new car is calculated by multiplying its normalized feature vector with the parameter vector (alt).
- The same process happened to the car 2 as well.

```
print("  Actual price for car 1: 11500 AZN -> " +
"Predicted price: " + str(int(price_car_1)) + " AZN")
```

```
print("  Actual price for car 2: 8800 AZN -> "+
"Predicted price: " + str(int(price_car_2)) + " AZN")
```

- These lines print the actual and predicted prices for the cars with their actual and predicted prices.

The output:

```
  Actual price for car 1: 11500 AZN -> Predicted price: 15840 AZN
  Actual price for car 2: 8800 AZN -> Predicted price: 5425 AZN
```

Linear Regression Using Library

Linear regression is performed using a library (scikit-learn) to fit the model to the data. At the end, the results are compared with those obtained from the application.

```
features = data[['Buraxilish ili', 'Yurush']].to_numpy()
target = data['Qiymet'].to_numpy()
```

- Extracting inputs and labels

```
kf = KFold(n_splits=10)

regressor = LinearRegression()
```

- Wth KFold, we create 10 folds.
- Then, created an instance of the LinearRegression class.

```
for train_indices, test_indices in kf.split(features):
   train_x, test_x = features[train_indices],
features[test_indices]
   train_y, test_y = target[train_indices],
target[test_indices]

   regressor.fit(train_x, train_y)

   r_squared = regressor.score(test_x, test_y)

   print(r_squared)
```

- First, we obtain indices for training and test data.
- Splitting the features and target variables into training and testing sets.
- Then, I calculated the r-squared values on the testing data.
- Lastly, printing the r-squared values/

```
car_1_price = regressor.predict([[2000, 240000]])
car_2_price = regressor.predict([[1996, 415558]])
```

- This part of the code predicts the price of cars by calling the predict function of regressor.

```
print("Predicted price for car 1: " + str(car_1_price) + "
AZN" + "  Actual price: 11500 AZN")
print("Predicted price for car 2: " + str(car_2_price) + "
AZN" + "  Actual price: 8800 AZN")
```

- Showing predicted and actual prices for cars.

The output:

```
Predicted price for car 1: [15710.98031657] AZN  Actual price: 11500 AZN
Predicted price for car 2: [5539.89014912] AZN  Actual price: 8800 AZN
```

To summrize, in the report, I showed the comparison of custom application and library. In additional, I added visualizations of graphs

Assignment 1