นาย สัณหณัฐ พรมจรรย์ 63070501069

Quiz 5

1. Calibrate Camera. 2 hrs. Using the "Calibrate Camera by ChatGPT" program shown in class, calibrate your laptop or mobile phone camera to find its intrinsic parameters using 10-15 checkerboard images. Make sure you are not using mirror images. If the processing is slow, it may help to reduce the size of each image to a width of around 1,000.

Link: https://colab.research.google.com/drive/1Xhw6L6uFHZ3TcRHl8EvMi8XBhI0TIa5k?usp=sharing

```
[11] import numpy as np
     import cv2
     import glob
```

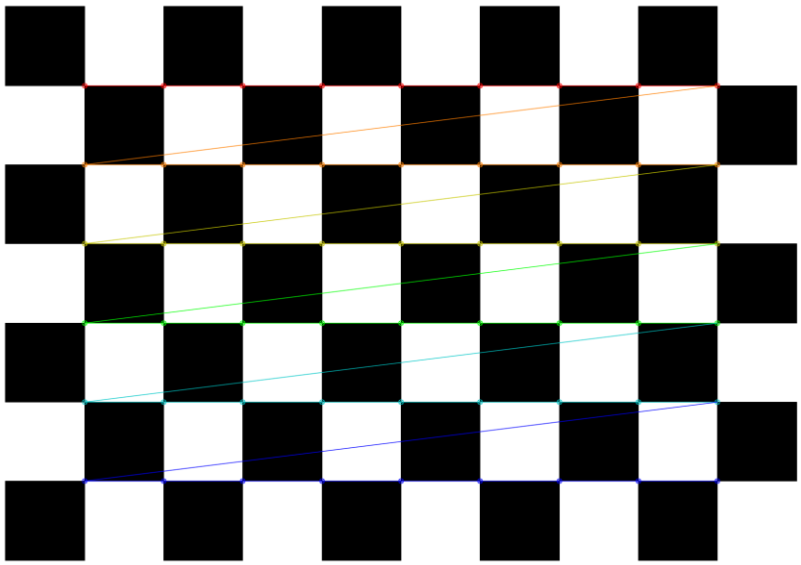
```
[12] from google.colab.patches import cv2_imshow #only used when running in Google Colab
     def my_imshow(title, img ):
       print(title)
       cv2_imshow(img)
```

```
[13] # Define the size of the checkerboard used in the images
     checkerboard_size = (9, 6) #9 rows (height: y-axis) by 6 columns (width: x-axis).
```

```
[14] # GREATE GRID OPTION 1.  ChatGPT.
     # Grid created by Chat GPT into objp.
     # Define the real-world coordinates of the corners of the checkerboard
     # comment by Suthep. objp has 54 x 3 zeros. Actually we use only 54 x 2, but here 3 because Point3D call in CV library
     objp = np.zeros((checkerboard_size[0]*checkerboard_size[1], 3), np.float32)

     print("objp before: \n", objp)


     # comment by Suthep. create a 9x6 mesh of 2D points {0..8} x {0..5} (54 elements, 2D) , transpose it to 6 x 9,
     # reshape it to (-1, 2) which means (54,2). 9 x 6 x 2 transposed to 6 x 9 x 2 reshaped to 54 x 2
     # Means give me 54 (x, y) point values on a uniform grid of chess board
     objp[:,:2] = np.mgrid[0:checkerboard_size[0],0:checkerboard_size[1]].T.reshape(-1,2)


     print ("\nmgrid output is y is [0], x is [1]:\n", np.mgrid[0:checkerboard_size[0],0:checkerboard_size[1]])

     print("\nobjp after reshaped mesh: \n", objp)
     print ("\nOutput of mgrid T & reshaped: \n", np.mgrid[0:checkerboard_size[0],0:checkerboard_size[1]].T.reshape(-1,2))
```

```
objp before:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
```

```
[15] # CREATE GRID OPTION 2
     # Alternative grid, reading corners directly from ideal 9x6 image
     objp = np.zeros((checkerboard_size[0]*checkerboard_size[1], 3), np.float32)
     fname = "checker_board_9x6_corners.png"
     img = cv2.imread(fname)
     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

     ret, corners = cv2.findChessboardCorners(gray, checkerboard_size, None)

     if ret == True:
        objp[:,:2] = corners [:, -1]
        print("\nobjp: \n", objp)
        img = cv2.drawChessboardCorners(img, checkerboard_size, corners, ret)
        my_imshow(fname + " with corners found", img )
     else:
        print("Error. World image grid corners not found.")
```



1.1. *10 points.* Report the fx, fy, cx, cy, and lens distortion (k1, k2, k3, p1, p2) parameters found using left##.jpg, frame-##.png, and your camera's images.

```
[16] # Create arrays to store object points and image points from all images
     objpoints = []
     imgpoints = []
```

```
[17] # Load all images of the checkerboard
     # Comment by Suthep.  Images will have a list of jpg file names read
     # images = glob.glob('*.jpg')
     from glob import glob
     images = glob('*.jpg') + glob('*.png')
```

```
[18] print ("Image Files Found: ", images)

     Image Files Found:  ['left13.jpg', 'left07.jpg', 'left10.jpg', 'left02.jpg', 'Mon03.jpg', 'left06.jpg', 'Mon02.jpg', 'left09.jpg', 'left12.jpg', 'left11.jpg',
```

```
[19]  # Loop through each image and find the corners of the checkerboard
      for fname in images:
          img = cv2.imread(fname)
          gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

          ret, corners = cv2.findChessboardCorners(gray, checkerboard_size, None)

          if ret == True:
              objpoints.append(objp)
              imgpoints.append(corners)

              # Draw the corners on the image
              img = cv2.drawChessboardCorners(img, checkerboard_size, corners, ret)
              my_imshow(fname + " with corners found", img )
```

[20]
```
#Understand array shape
ar = np.array([[12,20, 30] ,[13,15, 23]])
print (ar)
print("Shape of the array:", ar.shape) # Output will be (3, 2)
```

```
[[12 20 30]
 [13 15 23]]
Shape of the array: (2, 3)
```

[21]
```
# Use the object points and image points to compute the camera matrix, distortion coefficients, etc.

# intrinsic camera matrix, instrinsic lens distortion coefficients,
# extrinsic rotation,  and extrinsic translation vectors etc.
width = gray.shape[1]
height = gray.shape[0]
#ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None) #removed
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, (width, height), None, None) #Simplified

# Print the camera calibration parameters
print("\nCamera matrix:\n")
print(mtx)
print("\nDistortion coefficients:\n")
print(dist)
```

```
Camera matrix:

[[527.94650752   0.         203.75778674]
 [  0.         525.66815786 148.07044577]
 [  0.           0.           1.        ]]

Distortion coefficients:

[[-0.01281514  0.00221635 -0.00595345 -0.00308855 -0.00011889]]
```

[22]
```
print(width, height)
```

```
400 300
```

```
[25]  #mtx = np.array([[fx, 0, cx],
      #               [0, fy, cy],
      #               [0, 0, 1]])
      #dist = np.array([k1, k2, p1, p2, k3])

      fx = mtx[0,0]
      fy = mtx[1,1]
      cx = mtx[0,2]
      cy = mtx[1,2]
      k1 = dist[0,0]
      k2 = dist[0,1]
      p1 = dist[0,2]
      p2 = dist[0,3]
      k3 = dist[0,4]
      print("fx: ", fx)
      print("fy: ", fy)
      print("cx: ", cx)
      print("cy: ", cy)
      print("k1: ", k1)
      print("k2: ", k2)
      print("k3: ", k3)
      print("p1: ", p1)
      print("p2: ", p2)
```

```
fx:  527.9465075244948
fy:  525.6681578571174
cx:  203.7577867421932
cy:  148.07044577394754
k1:  -0.012815141235242168
k2:  0.0022163496140302453
k3:  -0.00011888544246553795
p1:  -0.0059534453081531765
p2:  -0.0030885528311611608
```

```
[26]  # Save the calibration results to a file
      np.savez("calibration_results.npz", mtx=mtx, dist=dist)
```

1.2. *10 points.* Show the Original and Undistorted image for *one* of your checkerboard images. Draw straight lines across the original image and undistorted image to see if the distortion has improved.

Instructions provided by ChatGPT:

In this code, replace your_image.jpg with the path to the image you want to undistort, and replace fx, fy, cx, cy, k1, k2, p1, p2, and k3 with the values for the camera matrix and distortion coefficients that were printed out by the previous program.

This code will display the original and undistorted images side by side. If you just want to display the undistorted image, you can remove the cv2.imshow('Original', img) line.

```
[27] img = cv2.imread("Mon01.jpg")
     # gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
     # Undistort the image using the camera matrix and distortion coefficients
     undistorted = cv2.undistort(img, mtx, dist)
     # Find image size
     h, w = undistorted.shape[:2] #height is [0], width is [1]
```

```
[28] # Show the original and undistorted images side by side
     my_imshow('Original', img)
     my_imshow('Undistorted', undistorted)
```

Undistorted

[29]
```
cv2.imwrite('distorted.png',img)
cv2.imwrite('undistorted.png',undistorted)
```

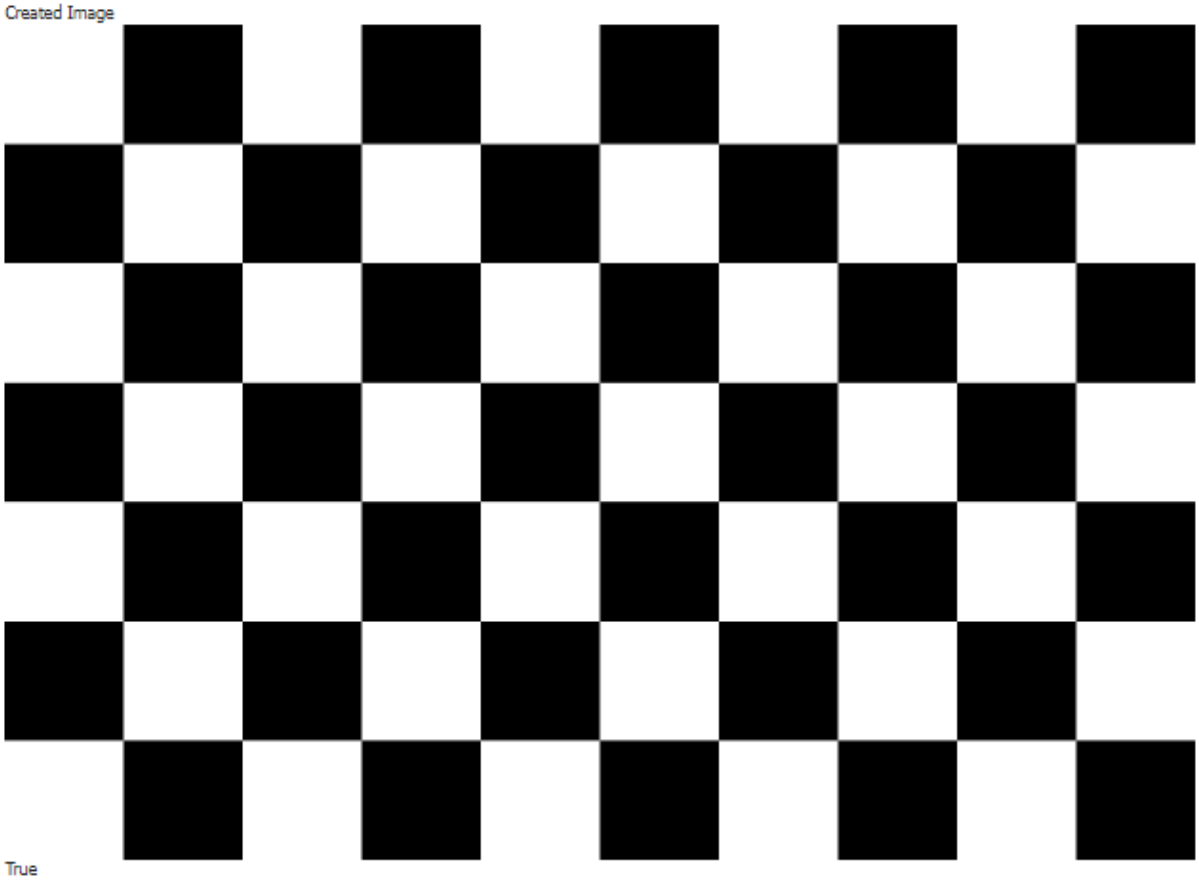
```
True
```

[30]
```
#Use mtx, dist, rvecs, tvecs.  Use checkerboard_size, objp
# renaming intrinsic camera parameters:
camera_matrix = mtx
dist_coeffs = dist
pattern_size = checkerboard_size
undistorted_img = cv2.imread("Mon01.jpg") #index 11
rvec = rvecs[11]
tvec = tvecs[11]
h,w = undistorted_img.shape[0], undistorted_img.shape[1]
```

[31]
```
# Define the full checkerboard size it should be inner corner + 1 for width and height
pattern_size = (10,7) #x - width, y - height

# Generate the object points for the checkerboard pattern
objp = np.zeros((np.prod(pattern_size), 3), dtype=np.float32)
objp[:, :2] = np.mgrid[0:pattern_size[0], 0:pattern_size[1]].T.reshape(-1, 2)
```

[32]
```
# Create an image with the checkerboard pattern
img = np.zeros((pattern_size[1]*100, pattern_size[0]*100), dtype=np.uint8)
for i in range(pattern_size[1]):
    for j in range(pattern_size[0]):
        if (i+j) % 2 == 0:
            img[i*100:(i+1)*100, j*100:(j+1)*100] = 255

my_imshow('Created Image', img)
cv2.imwrite('checkerboard.png', img)
```

```
Created Image
```



```
True
```

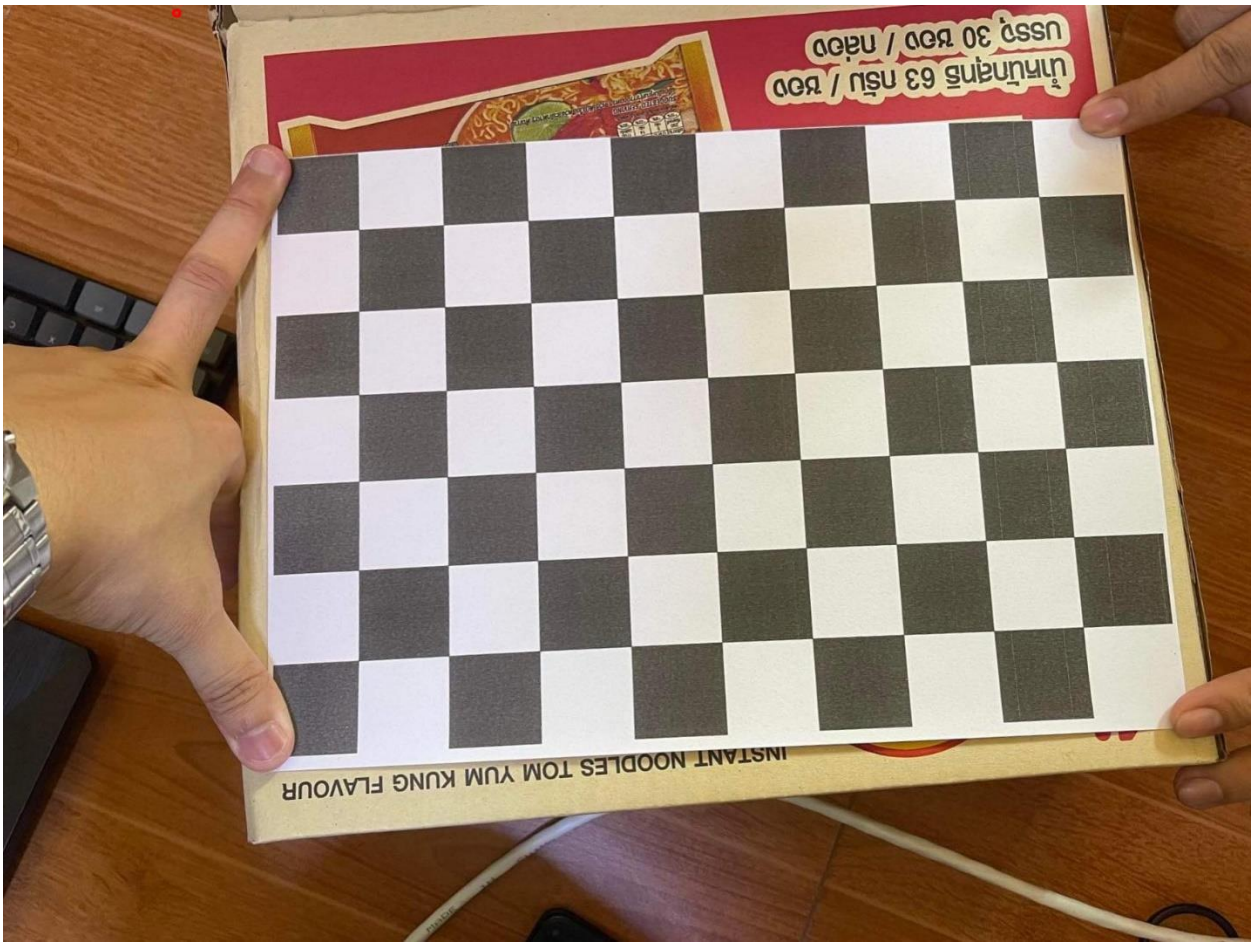
```
[33] # Compute extrinsic matrix
     R = cv2.Rodrigues(rvec)[0]
     t = tvec.reshape(-1, 1)
     extrinsic_matrix = np.hstack((R, t)) #horizontal stack 2 matrices into 1
     print('R: \n', R)
     print('t: \n', t)
     print('extrinsic matrix: \n', extrinsic_matrix)
```

```
R:
 [[ 0.92457369 -0.35597826  0.13580489]
 [ 0.32806726  0.92507466  0.19133412]
 [-0.19374045 -0.13234936  0.9720845 ]]
t:
 [[ 157.9756409 ]
 [-580.65987718]
 [2226.56841181]]
extrinsic matrix:
 [[ 9.24573692e-01 -3.55978256e-01  1.35804894e-01  1.57975641e+02]
 [ 3.28067256e-01  9.25074661e-01  1.91334123e-01 -5.80659877e+02]
 [-1.93740454e-01 -1.32349357e-01  9.72084505e-01  2.22656841e+03]]
```

```
[34] #corrected_img = cv2.warpPerspective(undistorted_img,extrinsic_matrix,(w,h),flags=cv2.INTER_LINEAR)

     # Project image points to world coordinates
     img_points = cv2.findChessboardCorners(undistorted_img, pattern_size)[1]
     world_points = cv2.projectPoints(objp, rvec, tvec, camera_matrix, dist_coeffs)[0]
```

```
[35] # Draw projected points on image and display
     for i in range(len(world_points)):
         x, y = int(world_points[i][0][0]), int(world_points[i][0][1])
         #cv2.circle(image, center_coordinates, radius, color, thickness)
         cv2.circle(undistorted_img, (x, y), 5, (0, 0, 255), 1) #color = (0,0,255)

     my_imshow("Undistorted Image with Projected Points", undistorted_img)
```

Undistorted Image with Projected Points



2. Using the Regression on diabetes data example:

Link:

https://colab.research.google.com/drive/1TVocZOqwBEgOI8vO6rzll72UO9PRNsmu?usp=sharing

```
[15] import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn import datasets, linear_model
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression

     # Load diabetes dataset
     diabetes = datasets.load_diabetes()

     # Convert to pandas dataframe
     df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
     print("Data : \n", df.head())

     # Add target variable to dataframe
     df['target'] = diabetes.target
```

```
Data :
        age      sex       bmi        bp        s1        s2        s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
2  0.085299  0.050680  0.044451 -0.005670 -0.045599 -0.034194 -0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142

         s4        s5        s6
0 -0.002592  0.019907 -0.017646
1 -0.039493 -0.068332 -0.092204
2 -0.002592  0.002861 -0.025930
3  0.034309  0.022688 -0.009362
4 -0.002592 -0.031988 -0.046641
```

2.1. *5 points.* 1 hr. Is age highly correlated with total cholesterol / HDL (column 'S4')?

```
[16] # Check correlation between age and S4
     corr = df['age'].corr(df['s4'])

     print(f"Correlation between Age and Total Cholesterol/HDL: {corr:.3f}")
```

```
Correlation between Age and Total Cholesterol/HDL: 0.204
```

2.2. *5 points.* 0.5 hr. Is blood pressure highly correlated with total cholesterol / HDL (column 'S4')?

```
[17] # Check correlation between bp and S4
     corr = df['bp'].corr(df['s4'])

     print(f"Correlation coefficient between Blood Pressure and Total Cholesterol/HDL: {corr:.3f}")
```

```
Correlation coefficient between Blood Pressure and Total Cholesterol/HDL: 0.258
```

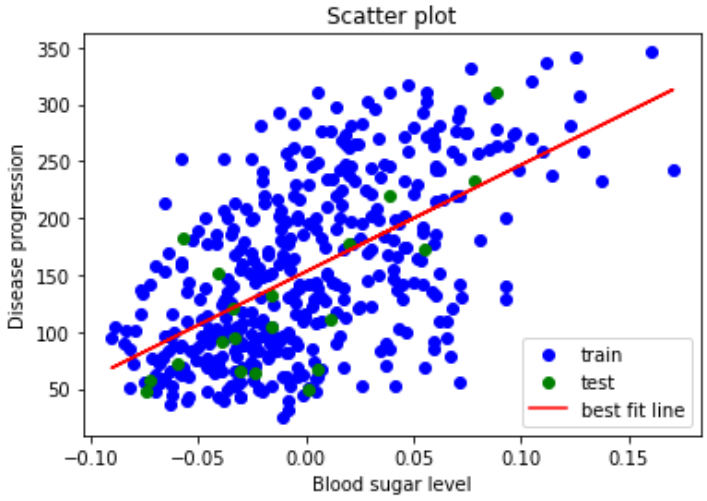
2.3. *15 points (4+3+3+5).* 1 hr. Report Linear fit results for y = ax + b where x is the blood sugar level

i. Linear fit coefficients and intercept of the training data
ii. What is the R^2 for the training data?
iii. What is the R^2 for the prediction of y based on blood sugar level for the test data?
iv. Show a scatter plot of the train set (x, y) as blue circles and predicted (x, y) as green circles. Also show the best fit line in red.

```
[18] diabetes = datasets.load_diabetes()
     # Use only one feature
     X = diabetes.data[:, 2].reshape(-1, 1)
     y = diabetes.target.reshape(-1, 1)

     # Split the data into training/testing sets
     X_train = X[:-20]
     X_test = X[-20:]

     # Split the targets into training/testing sets
     y_train = y[:-20]
     y_test = y[-20:]

     # Create linear regression object
     regr = linear_model.LinearRegression()

     # Train the model using the training sets
     regr.fit(X_train, y_train)
```

```python
# Obtain the linear fit coefficients and intercept of the training data
# Ex. for y = ax+b, Coefficients = [a], Intercept = b.
a = regr.coef_[0]
b = regr.intercept_
print('Coefficients:', a)
print('Intercept:', b)

# Calculate the R^2 for the training data
train_r2 = regr.score(X_train, y_train)
print(f"R^2 for training data: {train_r2:.3f}")

# Calculate the R^2 for the test data
test_r2 = regr.score(X_test, y_test)
print(f"R^2 for test data: {test_r2:.3f}")

# Create a scatter plot of the train set (x, y) as blue circles and predicted (x, y) as green circles, and show the best fit line in red
y_pred_train = regr.predict(X_train)
y_pred_test = regr.predict(X_test)

plt.scatter(X_train, y_train, color='blue', label='train')
plt.scatter(X_test, y_test, color='green', label='test')
plt.plot(X_train, y_pred_train, color='red', label='best fit line')
plt.title("Scatter plot")
plt.xlabel('Blood sugar level')
plt.ylabel('Disease progression')
plt.legend()
plt.show()
```

```
Coefficients: [938.23786125]
Intercept: [152.91886183]
R^2 for training data: 0.335
R^2 for test data: 0.473
```



3. Use the data provided in the shared file gasoline_use.txt to:

Link: https://colab.research.google.com/drive/19yz4-TsHyAuFw2cJ8SFywL2A6X0FRmic?usp=sharing

ผมนำข้อมูลใน gasoline_use.txt มาแปลงเป็น gasoline_use.csv

```
[72] import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error
     from sklearn.metrics import r2_score

     # Load the dataset
     data = pd.read_csv("gasoline_use.csv")
     print("Data : \n", data.head())

     # Define the independent and dependent variables
     X = data[['Gasolin tax (cents per gallon)', 'Average income (dollars)',
             'Paved Highways (miles)', 'Proportion of population with driver\'s licenses']]
     y = data['Consumption of gasoline (millions of gallons)']

     # Split the data into training/testing sets
     X_train = X[:-20]
     X_test = X[-20:]

     # Split the targets into training/testing sets
     y_train = y[:-20]
     y_test = y[-20:]
```

```
Data :
   Index  One  Gasolin tax (cents per gallon)  Average income (dollars)  \
0      1    1                             9.0                      3571
1      2    1                             9.0                      4092
2      3    1                             9.0                      3865
3      4    1                             7.5                      4870
4      5    1                             8.0                      4399

   Paved Highways (miles)  Proportion of population with driver's licenses  \
0                    1976                                           0.525
1                    1250                                           0.572
2                    1586                                           0.580
3                    2351                                           0.529
4                     431                                           0.544

   Consumption of gasoline (millions of gallons)
0                                            541
1                                            524
2                                            561
3                                            414
4                                            410
```

3.1. *10 points.* Show the equation found by fitting the training data:

$$y = f(x_1, x_2, x_3, x_4) = a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4$$

```
[73] # Create linear regression object
     regr = LinearRegression()

     # Train the model using the training sets
     regr.fit(X_train, y_train)

     # Show the equation found by fitting the training data
     print(f"y = {regr.intercept_:.3f} + ({regr.coef_[0]:.3f})*x_1 + ({regr.coef_[1]:.3f})*x_2 + ({regr.coef_[2]:.3f})*x_3 + ({regr.coef_[3]:.3f})*x_4")
```

```
y = 206.877 + (-10.550)*x_1 + (-0.083)*x_2 + (0.004)*x_3 + (1356.264)*x_4
```

3.2. *5 points.* What is the R^2 for the prediction of y? Use testing data.

```
[74] # Make predictions using the testing set
     y_pred = regr.predict(X_test)

     # Calculate the mean squared error (MSE)
     mse = mean_squared_error(y_test, y_pred)
     print(f"Mean squared error: {mse:.3f}")
     # Calculate the root mean squared error (RMSE)
     rmse = np.sqrt(mse)
     print(f"Root mean squared error: {rmse:.3f}")
     # Calculate the R^2 for the prediction of y using test data
     test_r2 = r2_score(y_test, y_pred)
     print(f"R^2 for test data: {test_r2:.3f}")
```

```
Mean squared error: 7060.785
Root mean squared error: 84.028
R^2 for test data: 0.363
```

3.3. *5 points.* What would happen to gasoline consumption if taxes are increased by $2.00? Use training data.

To find out what would happen to gasoline consumption if taxes are increased by $2.00

```
[75] # We can use the equation we found in clause 3.1 and substitute x_1 with itscurrent value plus $2.00
     x_1_plus2 = regr.coef_[0]*2
     print(f"y = {regr.intercept_:.3f} + ({x_1_plus2:.3f}) + ({regr.coef_[0]:.3f})*x_1  + ({regr.coef_[1]:.3f})*x_2 + ({regr.coef_[2]:.3f})*x_3 + ({regr.coef_[3]:.3f})*x_4")
```

```
y = 206.877 + (-21.099) + (-10.550)*x_1  + (-0.083)*x_2 + (0.004)*x_3 + (1356.264)*x_4
```

ไฟล์ที่ใช้เพิ่มเติมสำหรับรูปข้อ 1 และ ไฟล์ .csv ข้อ 3:

https://drive.google.com/drive/folders/11tyJ6AhgvItCosW4cisgiR4ef1CL_8YY?usp=sharing