CS 488 HW2

Yingren Wang

1. Iris Data Visualization

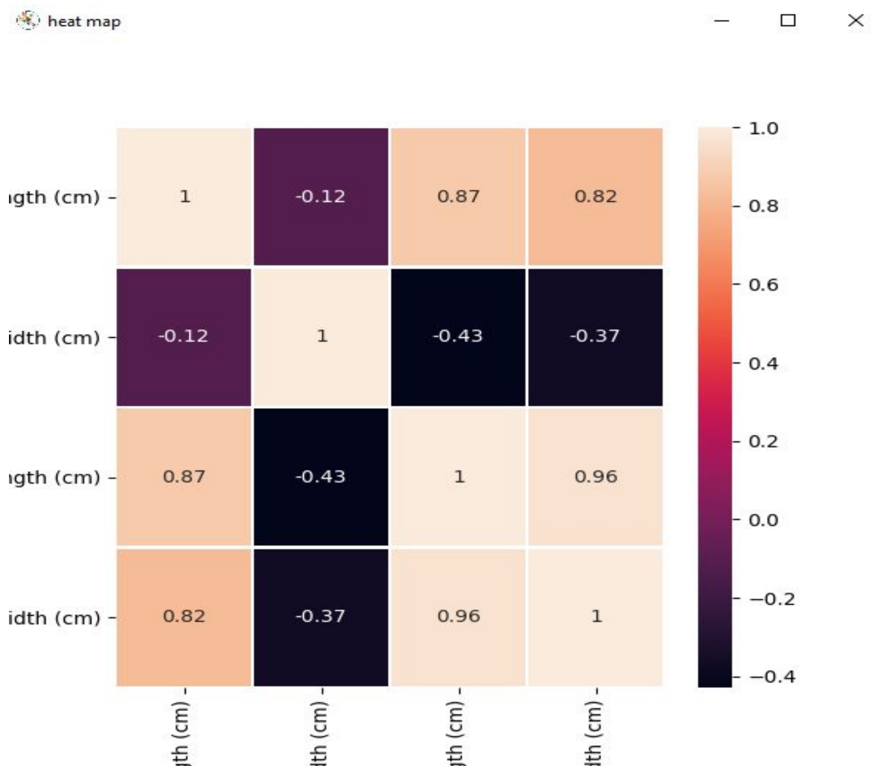
1a. Data visualization screenshots

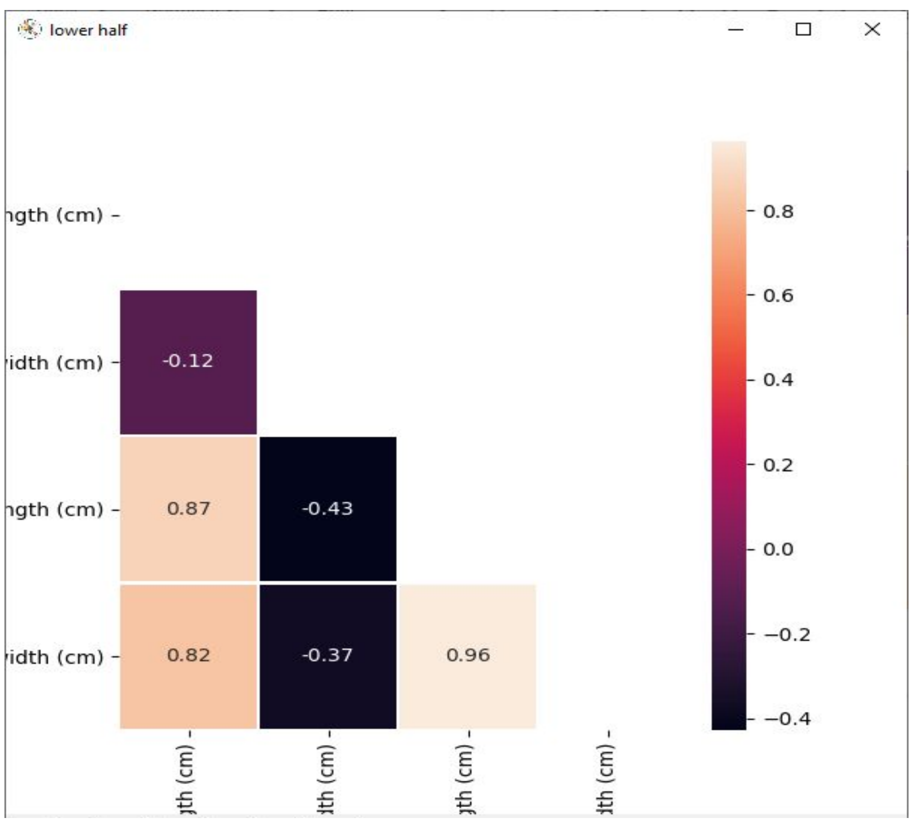
1b. Implications and Inferences

a) overlap means the data that overlaps on top of each other are not easy to be distinguished from each other. In this case, for example, the versicolor and virginica have a great overlap on top of each other, leading to a situation that is hard to distinguish between the two. Some features have a negative correlation, for example, the sepal length to sepal width one, while some other features have a positive correlation.

b) petal width to sepal length and petal length to sepal length has strong positive correlations according to the heat map, which can be more accurate to be used to predict. Comparing to the other two species, setosa is easier to be distinguished according to the color-coded features graph. Sepal width is standardly distributed compared to other features according to all the histograms of different features. This can help us to predict better.

Data visualization screenshots
(Question 1)





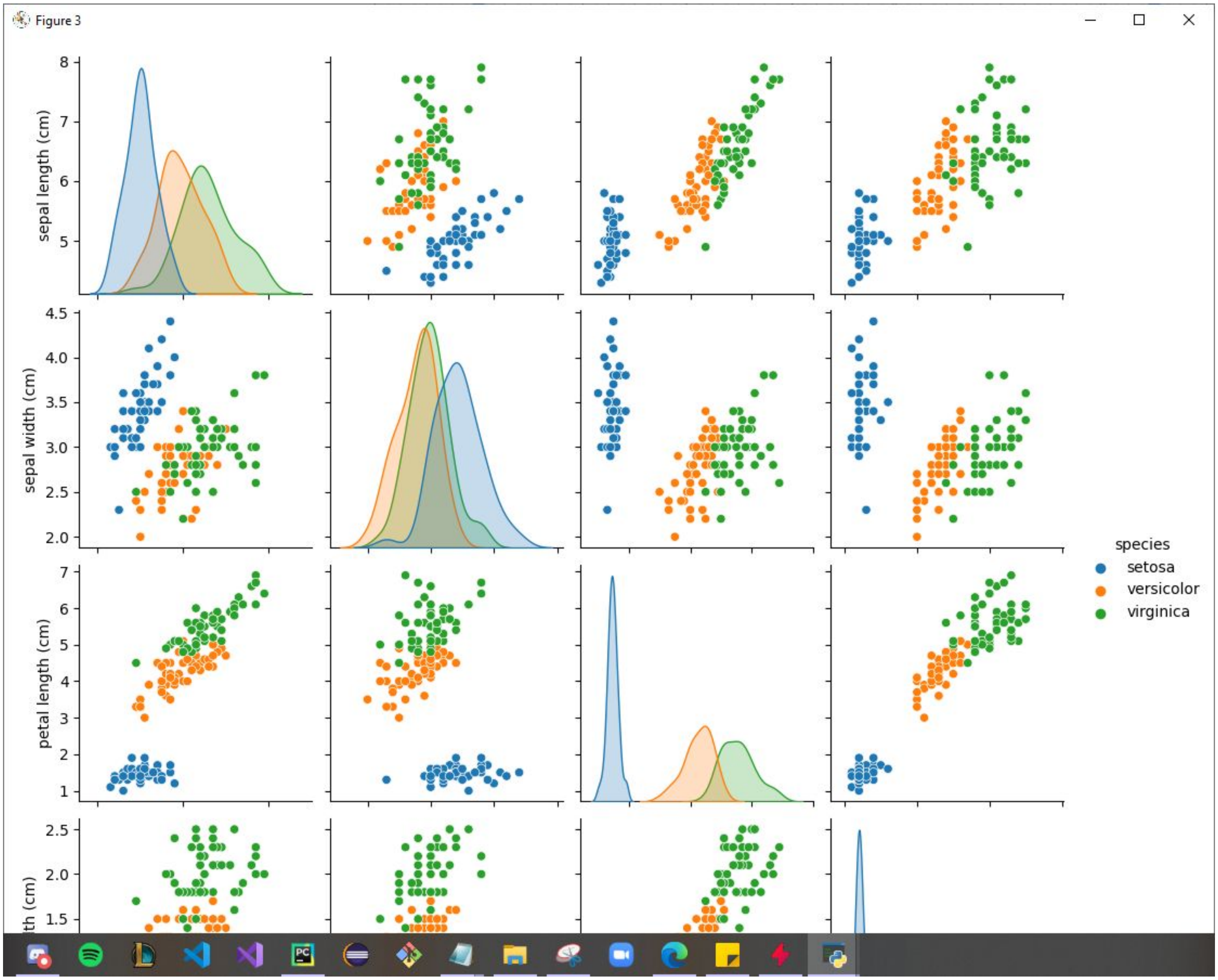
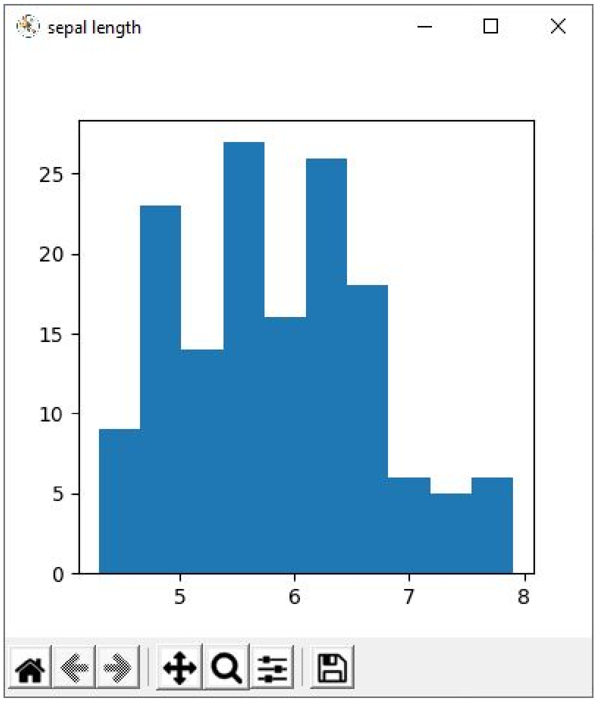
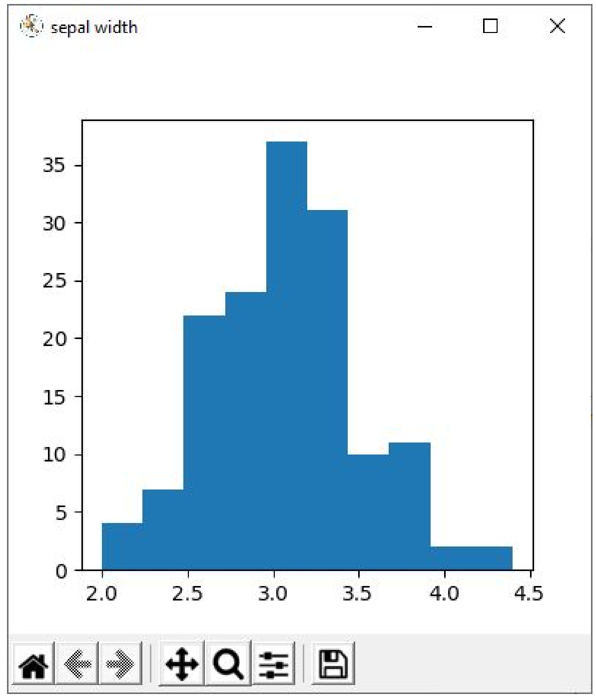
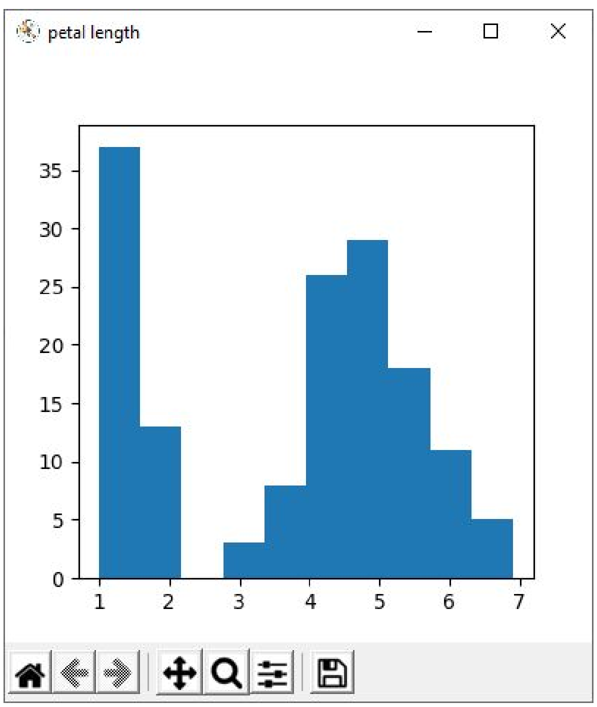
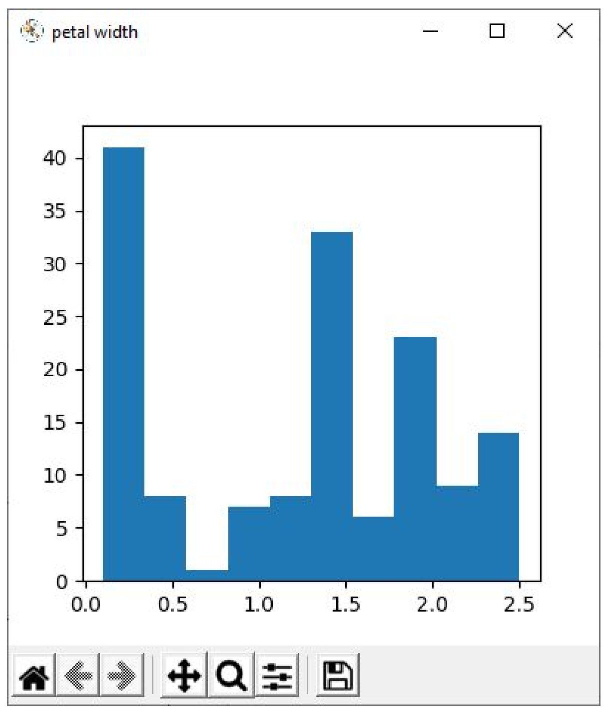


Figure 3
sepal length (cm)
sepal width (cm)
petal length (cm)
th (cm)
species
setosa
versicolor
virginica


sepal length
0
5
10
15
20
25
5
6
7
8


sepal width
0
5
10
15
20
25
30
35
2.0
2.5
3.0
3.5
4.0
4.5


petal length
0
5
10
15
20
25
30
35
1
2
3
4
5
6
7


petal width
0
5
10
15
20
25
30
35
40
0.0
0.5
1.0
1.5
2.0
2.5

```
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load Iris Data
iris = load_iris()

# Creating pd DataFrames
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
target_df = pd.DataFrame(data=iris.target, columns=['species'])


# generate labels
def converter(specie):
    if specie == 0:
        return 'setosa'
    elif specie == 1:
        return 'versicolor'
    else:
        return 'virginica'


target_df['species'] = target_df['species'].apply(converter)

# Concatenate the data frames
df = pd.concat([iris_df, target_df], axis=1)

# output data
print(df)

# compute the correlation coefficient for iris data set
df.corr()

# visualize iris features as a heat map
cor_eff = df.corr()
plt.figure(num='heat map', figsize=(6, 6))
sns.heatmap(cor_eff, linecolor='white', linewidths=1, annot=True)

# plot the lower half of the correlation matrix
fig, ax = plt.subplots(num='lower half', figsize=(6, 6))
# compute the correlation matrix
mask = np.zeros_like(cor_eff)

# mask = 0; display the correlation matrix, mask = 1; display the unique lower triangular val
mask[np.triu_indices_from(mask)] = 1
sns.heatmap(cor_eff, linecolor='white', linewidths=1, mask=mask, ax=ax, annot=True)

# iris feature analysis
```

```
g = sns.pairplot(df, hue='species')

# histogram for sepal length
plt.figure(num='sepal length', figsize=(4, 4))
sepalLength = df['sepal length (cm)']
plt.hist(sepalLength, bins=10)

# histogram for sepal width
plt.figure(num='sepal width', figsize=(4, 4))
sepalWidth = df['sepal width (cm)']
plt.hist(sepalWidth, bins=10)

# histogram for petal length
plt.figure(num='petal length', figsize=(4, 4))
petalLength = df['petal length (cm)']
plt.hist(petalLength, bins=10)

# histogram for petal width
plt.figure(num='petal width', figsize=(4, 4))
petalWidth = df['petal width (cm)']
plt.hist(petalWidth, bins=10)

plt.show()
```

2. XYZ and ABC Linear Regression

2a - 2c. See the following page

2d. Linear Regression Visualization

```
Estimated coefficients:
 alpha (slope intercept) = -5.403877221324706
 beta (slope) = 52.72213247172859
Estimated coefficients:
 alpha (slope intercept) = -5.403877221324706
 beta (slope) = [52.72213247]
For new x = 30 the estimated new y prediction = [1576.26009693]
Do not invest in ABC
```

n=7	X_i	Y_i	X_i^2	Y_i^2	$X_i Y_i$
	3	100	9	10000	300
	5	250	25	62500	1250
	7	330	49	108900	2310
	9	590	81	348100	5310
	12	660	144	435600	7920
	15	780	225	608400	11700
	18	890	324	792100	16020
sum	69	3600	857	2365600	44810
average	9.857	514.286	122.429	337942.857	6401.429

$$\beta = \frac{44810 - \frac{69(3600)}{7}}{857 - \frac{(69)^2}{7}} = 52.722$$

$$\alpha = \bar{Y} - \beta\bar{X} = 514.286 - 52.722(9.857) = -5.404$$

2b). $\because y = \beta x + a$

$\therefore y = 52.722x - 5.404$

if $x = 18 + 12 = 30$

then $y = 52.722(30) - 5.404 = 1576.256$

2c) $1576.256 \div 890 = 1.771$

$1.771 > 1.5$

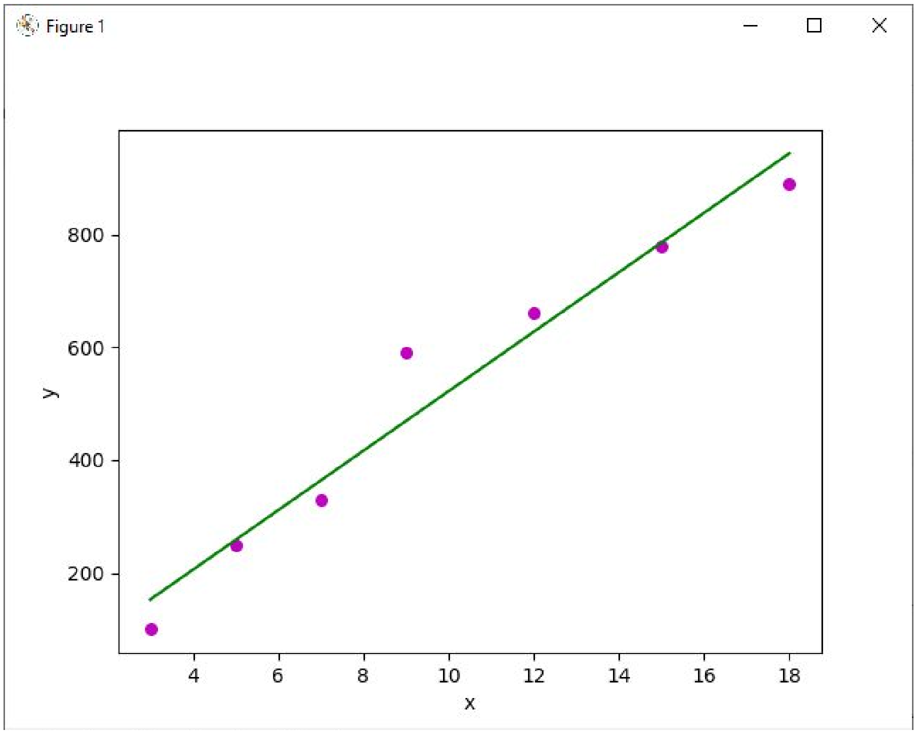
So do not invest in ABC

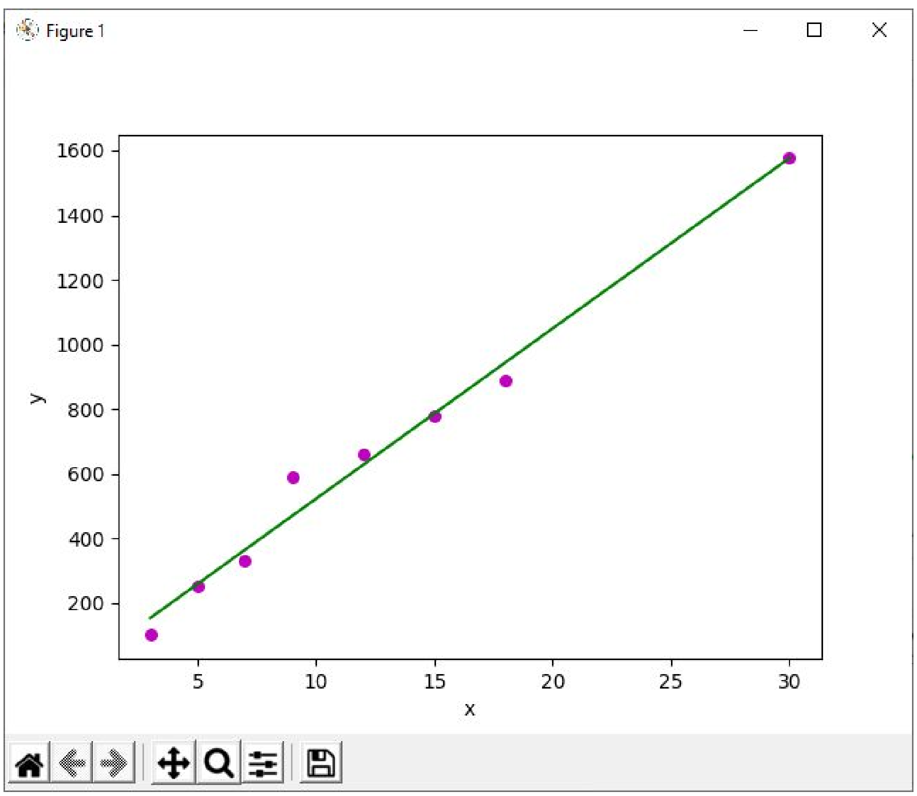
2b) inferences:

① As the months goes longer, the profit get bigger so it's a positive trend with the β being positive

② When first started (month = 0), the profit was -5.404. So it's worth investing since there will be more profits

Linear Regression Visualization
(Question 2)





```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression


def lin_reg(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x, m_y = np.mean(x), np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y * x) - n * m_y * m_x
    SS_xx = np.sum(x * x) - n * m_x * m_x

    # calculating regression coefficients
    beta = SS_xy / SS_xx
    alpha = m_y - beta * m_x

    return alpha, beta


def plot_lin_reg_model(x, y, a, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color="m", marker="o", s=30)

    # predicted response vector
    # y_pred = alpha + beta * x
    y_pred = a + b * x

    # plotting the regression line
    plt.plot(x, y_pred, color="g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()


def main():
    # observations
    x = np.array([3,5,7,9,12,15,18])
    y = np.array([100,250,330,590,660,780,890])

    # estimating coefficients
    a, b = lin_reg(x, y)
    print("Estimated coefficients:\n alpha (slope intercept) = {}   "
```

```
          "\n beta (slope) = {}".format(a, b))

    # plotting regression line
    plot_lin_reg_model(x, y, a, b)

    # compare with sklearn
    X = np.array([3,5,7,9,12,15,18])
    Y = np.array([100,250,330,590,660,780,890])
    XX = np.reshape(X, (-1, 1))
    reg = LinearRegression().fit(XX, Y)

    # Coefficient of determination
    # c_det = reg.score(XX, Y)
    # print("Estimated Coefficient of determination = {}".format(c_det))

    # estimating coefficients
    print("Estimated coefficients:\n alpha (slope intercept) = {}    "
          "\n beta (slope) = {}".format(reg.intercept_, reg.coef_))

    # Predict a new data point
    # add a new point and see how the data change
    new_x = 30
    new_y = reg.predict(np.reshape(new_x, (-1, 1)))
    print("For new x = {} the estimated new y prediction = {}".format(new_x, new_y))

    # check if should invest in ABC or not
    if (float(new_y / 890)) > 1.5:
        print("Do not invest in ABC")
    else:
        print("Invest in ABC")

    # plotting regression line
    plot_lin_reg_model(np.append(x, new_x), np.append(y, new_y), reg.intercept_, reg.coef_)


if __name__ == "__main__": main()
```

3. Iris Dataset Linear Regression

i) 30% sample

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
count         150.000000        150.000000         150.000000        150.000000
mean            5.843333          3.057333           3.758000          1.199333
std             0.828066          0.435866           1.765298          0.762238
min             4.300000          2.000000           1.000000          0.100000
25%             5.100000          2.800000           1.600000          0.300000
50%             5.800000          3.000000           4.350000          1.300000
75%             6.400000          3.300000           5.100000          1.800000
max             7.900000          4.400000           6.900000          2.500000
LR beta/slope Coefficient: [ 0.67760814 -0.53212752  1.00051066  0.50420577]
LR alpha/slope_intercept Coefficient: -0.27251857592731277
Coefficient of determination:  0.966361264753838
Root Mean Squared Error (RMSE): 0.32455495722112165
Mean Squared Error (MSE): 0.10533592025680412
```

ii) 70% sample

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
count         150.000000        150.000000         150.000000        150.000000
mean            5.843333          3.057333           3.758000          1.199333
std             0.828066          0.435866           1.765298          0.762238
min             4.300000          2.000000           1.000000          0.100000
25%             5.100000          2.800000           1.600000          0.300000
50%             5.800000          3.000000           4.350000          1.300000
75%             6.400000          3.300000           5.100000          1.800000
max             7.900000          4.400000           6.900000          2.500000
LR beta/slope Coefficient: [ 0.62339783 -0.80150855  1.11422801  0.34931187]
LR alpha/slope_intercept Coefficient: 0.855761816862834
Coefficient of determination:  0.9607565199785818
Root Mean Squared Error (RMSE): 0.33421561706995867
Mean Squared Error (MSE): 0.11170007869345326
```

3a. The 30% samples one is better because it has a lower value of root mean squared error (RMSE) comparing to the other one.

3b. Predict ‘petal length’ for sample 50

i) 30% samples

```
   sepal length (cm)  sepal width (cm)  petal width (cm)  species
0                7.0               3.2               1.4        0
Predicted Petal Length (cm) 4.168645276630132
Actual Petal Length (cm) 4.7
```

ii) 70% samples

```
   sepal length (cm)  sepal width (cm)  petal width (cm)  species
0                7.0               3.2               1.4        0
Predicted Petal Length (cm) 4.214638490811135
Actual Petal Length (cm) 4.7
```

Here looks like .7 train size gets a better result than the .3 train size.

iii) RSME for 30% samples

```
Root Mean Squared Error (RMSE): 0.5313547233698683
```

iv) RSME for 70% samples

```
Root Mean Squared Error (RMSE): 0.48536150918886545
```

3c. I think the .3 train size is better because though the .7 train size has a more precise result, it requires way more train cases which will increase the computational cost. However, in a situation that precision is the most important thing, .7 train size will be better.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import pandas as pd
import seaborn as sns

# load iris data
iris = load_iris()

# Creating pd DataFrames
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
target_df = pd.DataFrame(data=iris.target, columns=['species'])


# generate labels
def converter(specie):
    if specie == 0:
        return 'setosa'
    elif specie == 1:
        return 'versicolor'
    else:
        return 'virginica'


target_df['species'] = target_df['species'].apply(converter)

# concatenate the dataframes
iris_df = pd.concat([iris_df, target_df], axis=1)

# iris data statistics
print(iris_df.describe())

from sklearn.metrics import mean_squared_error, r2_score

# converting objects to numerical datatype
iris_df.drop('species', axis=1, inplace=True)
target_df = pd.DataFrame(columns=['species'], data=iris.target)
iris_df = pd.concat([iris_df, target_df], axis=1)

# Variables
X = iris_df.drop(labels='petal length (cm)', axis=1)
y = iris_df['petal length (cm)']
#
# X = iris_df.drop(labels='sepal length (cm)', axis=1)
# y = iris_df['sepal length (cm)']

# splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.7, random_state=111)
```

```
# linear regression-LR model
lr = LinearRegression()

# fit LR model
lr.fit(X_train, y_train)

# LR prediction
lr.predict(X_test)
y_pred = lr.predict(X_test)

# Quantative analysis - evaluate LR performance

# LR coefficients - beta/slope
print('LR beta/slope Coefficient:', lr.coef_)

# LR coefficients - alpha/slope_intercept
print('LR alpha/slope_intercept Coefficient:', lr.intercept_)

# coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: ', r2_score(y_test, y_pred))

# Model performance - Error
print('Root Mean Squared Error (RMSE):', np.sqrt(mean_squared_error(y_test, y_pred)))
print('Mean Squared Error (MSE):', mean_squared_error(y_test, y_pred))

# predict a new datapoint

# select any datapoint to predict
iris_df.loc[50]

# create a new dataframe
d = {'sepal length (cm)': [7.0],
     'sepal width (cm)': [3.2],
     'petal width (cm)': [1.4],
     'species': 0}
pred_df = pd.DataFrame(data=d)

with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(pred_df)

# predict the new data point using LR
pred = lr.predict(pred_df)
print('Predicted Petal Length (cm)', pred[0])
print('Actual Petal Length (cm)', 4.7)

print('Root Mean Squared Error (RMSE):', np.sqrt(mean_squared_error([4.7], [pred])))
```