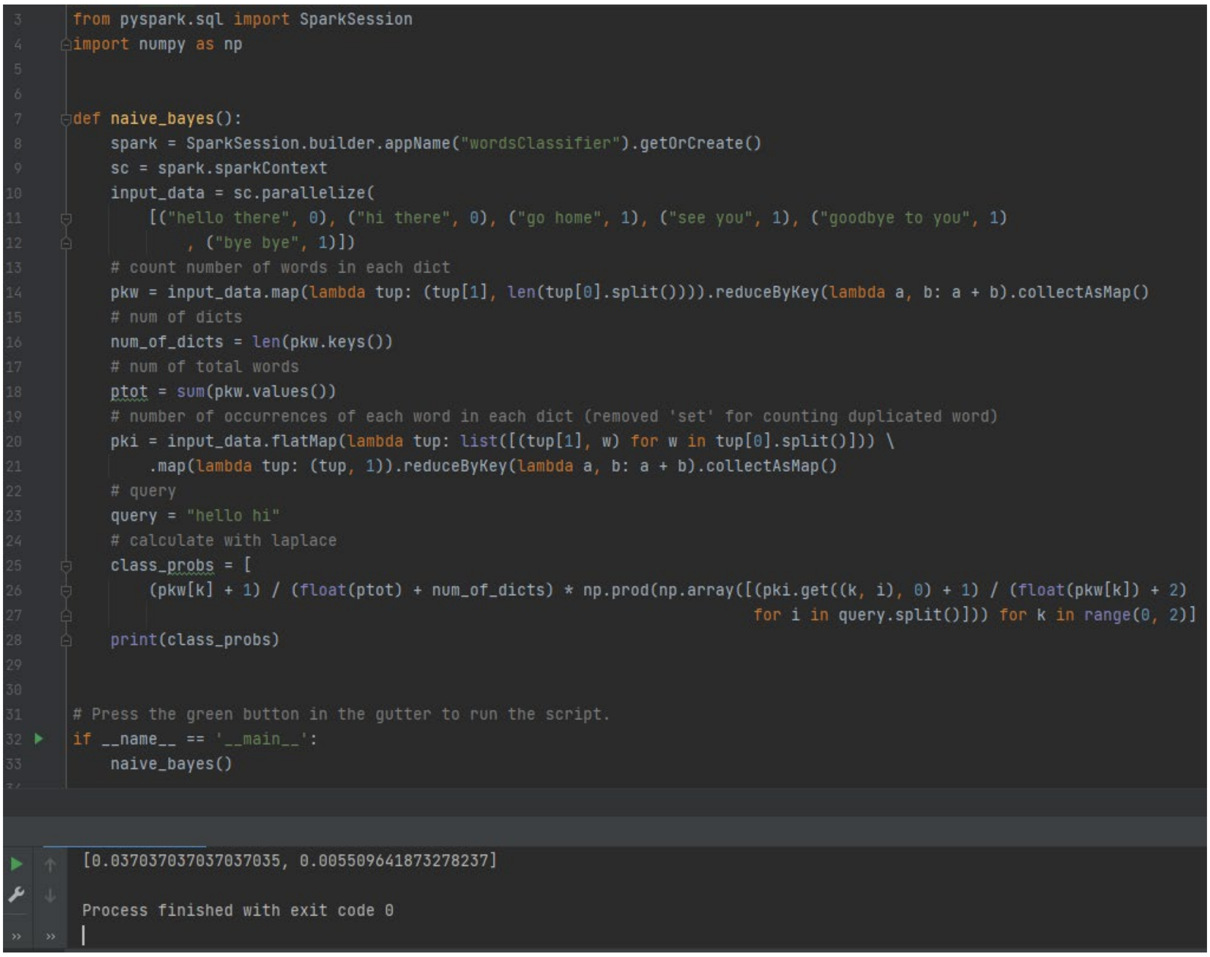
מטלה 4 – מסדי נתונים

Spark + Naïve Bayes

```
from pyspark import SparkContext
import psutil
from pyspark.sql import SparkSession
import numpy as np


def naive_bayes():
   spark = SparkSession.builder.appName("wordsClassifier").getOrCreate()
   sc = spark.sparkContext
   input_data = sc.parallelize(
      [("hello there", 0), ("hi there", 0), ("go home", 1), ("see you", 1), ("goodbye to you", 1)
         , ("bye bye", 1)])
   # count number of words in each dict
   pkw = input_data.map(lambda tup: (tup[1], len(tup[0].split()))).reduceByKey(lambda a, b: a +
b).collectAsMap()
   # num of dicts
   num_of_dicts = len(pkw.keys())
   # num of total words
   ptot = sum(pkw.values())
   # number of occurrences of each word in each dict (removed 'set' for counting duplicated word)
   pki = input_data.flatMap(lambda tup: list([(tup[1], w) for w in tup[0].split()])) \
      .map(lambda tup: (tup, 1)).reduceByKey(lambda a, b: a + b).collectAsMap()
   # query
   query = "hello hi"
   # calculate with laplace
   class_probs = [
      (pkw[k] + 1) / (float(ptot) + num_of_dicts) * np.prod(np.array([(pki.get((k, i), 0) + 1) /
(float(pkw[k]) + 2)
                                                   for i in query.split()])) for k in range(0, 2)]
   print(class_probs)


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
   naive_bayes()
```

צילום מסך:



```
from pyspark.sql import SparkSession
import numpy as np


def naive_bayes():
    spark = SparkSession.builder.appName("wordsClassifier").getOrCreate()
    sc = spark.sparkContext
    input_data = sc.parallelize(
        [("hello there", 0), ("hi there", 0), ("go home", 1), ("see you", 1), ("goodbye to you", 1)
            , ("bye bye", 1)])
    # count number of words in each dict
    pkw = input_data.map(lambda tup: (tup[1], len(tup[0].split()))).reduceByKey(lambda a, b: a + b).collectAsMap()
    # num of dicts
    num_of_dicts = len(pkw.keys())
    # num of total words
    ptot = sum(pkw.values())
    # number of occurrences of each word in each dict (removed 'set' for counting duplicated word)
    pki = input_data.flatMap(lambda tup: list([(tup[1], w) for w in tup[0].split()])) \
        .map(lambda tup: (tup, 1)).reduceByKey(lambda a, b: a + b).collectAsMap()
    # query
    query = "hello hi"
    # calculate with laplace
    class_probs = [
        (pkw[k] + 1) / (float(ptot) + num_of_dicts) * np.prod(np.array([(pki.get((k, i), 0) + 1) / (float(pkw[k]) + 2)
                                                                        for i in query.split()])) for k in range(0, 2)]
    print(class_probs)


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    naive_bayes()
```

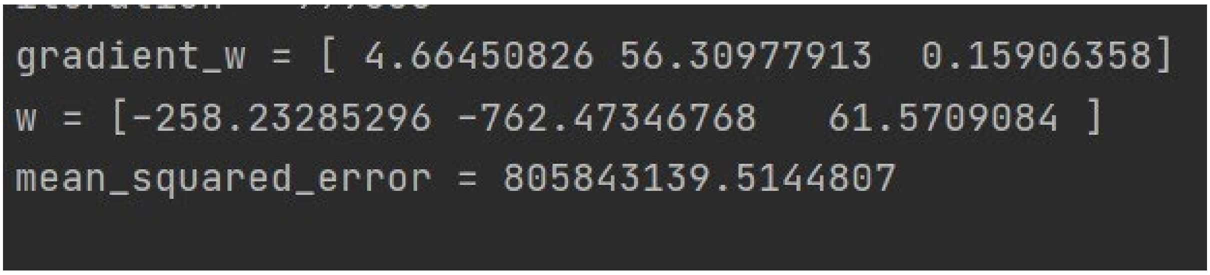
```
[0.037037037037037035, 0.005509641873278237]

Process finished with exit code 0
```

Linear and logistic regression

.1

```
import numpy as np
import pandas as pd

file_name = 'PUF2019.xlsx'
df = pd.read_excel(file_name)  # (file_name, index_col=0)
x = df[['LOCATION', 'BEDROOMS', 'SQFT']][:]  # , 'TITLED', 'STATUS'
y = df[['PRICE']][:]
train = int(y.size * 0.7)
test = y.size - train
data_x = []
data_y = []

data_test = []
data_result = []

for i in range(0, train):
    data_x.append(x.iloc[i].tolist())
    data_y.append(y.iloc[i].tolist())

for i in range(train, y.size):
    data_test.append(x.iloc[i].tolist())
    data_result.append(y.iloc[i].tolist())

data_x = np.array(data_x)
data_y = np.array(data_y)
data_x = np.squeeze(np.asarray(data_x))
data_y = np.squeeze(np.asarray(data_y))
data_test = np.squeeze(np.asarray(data_test))
data_result = np.squeeze(np.asarray(data_result))


def leaner_vector(x_data, y_data):
    w = np.array([-260, -730, 60])
    b = 0
    alpha = 0.0000001

    for iteration in range(1000000):
        deriv_b = np.mean(1 * ((np.dot(x_data, w) + b) - y_data))
        gradient_w = (1.0 / len(y_data)) * np.dot(((np.dot(x_data, w) + b) - y_data), x_data)
        b -= alpha * deriv_b
        w = w - (alpha * gradient_w)

    data_pred = []
    for i1 in range(0, test):
        data_pred.append(np.dot(np.array(data_test[i1]), w) + b)

    data_pred = np.squeeze(np.asarray(data_pred))
    n = data_pred.size
    summation = 0
    for i in range(0, n):
        difference = data_result[i] - data_pred[i]
        squared_difference = difference ** 2
        summation = summation + squared_difference
    MSE = summation / n
    print("mean_squared_error = " + str(MSE))


leaner_vector(data_x, data_y)
```

<u>צילום מסך של תוצאה סופית:</u>

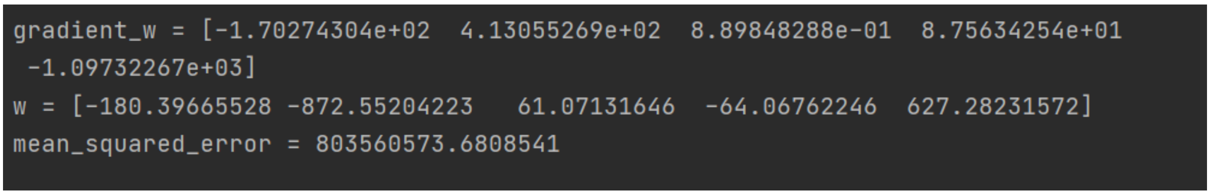


```
gradient_w = [ 4.66450826 56.30977913  0.15906358]
w = [-258.23285296 -762.47346768   61.5709084 ]
mean_squared_error = 805843139.5144807
```

.3

```
import numpy as np
import pandas as pd

file_name = 'PUF2019.xlsx'
df = pd.read_excel(file_name)  # (file_name, index_col=0)
x = df[['LOCATION', 'BEDROOMS', 'SQFT', 'TITLED', 'STATUS']][:]  # , 'TITLED', 'STATUS'
y = df[['PRICE']][:]
train = int(y.size * 0.7)
test = y.size - train
data_x = []
data_y = []

data_test = []
data_result = []

for i in range(0, train):
    data_x.append(x.iloc[i].tolist())
    data_y.append(y.iloc[i].tolist())

for i in range(train, y.size):
    data_test.append(x.iloc[i].tolist())
    data_result.append(y.iloc[i].tolist())

data_x = np.array(data_x)
data_y = np.array(data_y)
data_x = np.squeeze(np.asarray(data_x))
data_y = np.squeeze(np.asarray(data_y))
data_test = np.squeeze(np.asarray(data_test))
data_result = np.squeeze(np.asarray(data_result))


def leaner_vector(x_data, y_data):
    w = np.array([-260, -730, 60,0,0])
    b = 0
    alpha = 0.0000001

    for iteration in range(1000000):
        deriv_b = np.mean(1 * ((np.dot(x_data, w) + b) - y_data))
        gradient_w = (1.0 / len(y_data)) * np.dot(((np.dot(x_data, w) + b) - y_data), x_data)
        b -= alpha * deriv_b
        w = w - (alpha * gradient_w)
        if iteration == 1000000-1:
            print("gradient_w = " + str(gradient_w))
            print("w = " + str(w))

    data_pred = []
    for i1 in range(0, test):
        data_pred.append(np.dot(np.array(data_test[i1]), w) + b)

    data_pred = np.squeeze(np.asarray(data_pred))
    n = data_pred.size
    summation = 0
    for i in range(0, n):
        difference = data_result[i] - data_pred[i]
        squared_difference = difference ** 2
        summation = summation + squared_difference
    MSE = summation / n
    print("mean_squared_error = " + str(MSE))


leaner_vector(data_x, data_y)
```

צילום מסך של תוצאה סופית:



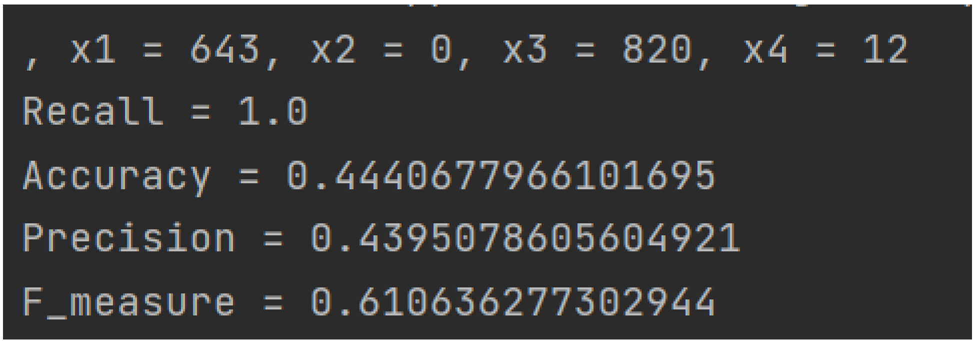
```
gradient_w = [-1.70274304e+02  4.13055269e+02  8.89848288e-01  8.75634254e+01
 -1.09732267e+03]
w = [-180.39665528 -872.55204223   61.07131646  -64.06762246  627.28231572]
mean_squared_error = 803560573.6808541
```

ניתן לראות כי אכן חלה ירידה ב - mean_squared_error כאשר הוספנו את הפרמטרים 'TITLED', 'STATUS'

.4

```
import numpy as np
import pandas as pd

file_name = 'PUF2019.xlsx'
df = pd.read_excel(file_name)  # (file_name, index_col=0)
x = df[['LOCATION', 'BEDROOMS', 'SQFT', 'PRICE']][:]
y = df[['SECURED']][:]
train = int(y.size * 0.7)
test = y.size - train
data_x = []
data_y = []

data_test = []
data_result = []

for i in range(0, train):
    data_x.append(x.iloc[i].tolist())
    data_y.append(y.iloc[i].tolist())

for i in range(train, y.size):
    data_test.append(x.iloc[i].tolist())
    data_result.append(y.iloc[i].tolist())

data_x = np.array(data_x)
data_y = np.array(data_y)
data_x = np.squeeze(np.asarray(data_x))
data_y = np.squeeze(np.asarray(data_y))
data_test = np.squeeze(np.asarray(data_test))
data_result = np.squeeze(np.asarray(data_result))
for i in range(0, train):
    if data_y[i] == 9:
        data_y[i] = 0

for i in range(0, data_result.size):
    if data_result[i] == 9:
        data_result[i] = 0


def h(x, w, b):
    return 1 / (1 + np.exp(-(np.dot(x, w) + b)))


def logistic(x_data, y_data):
    w = np.array([0., 0, 0, 0])
    b = 0
    alpha = 0.00000001
    for iteration in range(100000):
        gradient_b = np.mean(1 * (y_data - (h(x_data, w, b))))
        gradient_w = np.dot((y_data - h(x_data, w, b)), x_data) * 1 / len(y_data)
        b += alpha * gradient_b
        w += alpha * gradient_w

    data_pred = []
    for i1 in range(0, test):
        j = h(np.array(data_test[i1]), w, b)
        if j > 0.5:
            j1 = 1
        else:
            j1 = 0
        data_pred.append(j1)

    x1, x2, x3, x4 = 0, 0, 0, 0
    for i1 in range(0, data_result.size):
        pr, res = data_pred[i1], data_result[i1]
        if pr == res == 1:
            x1 = x1 + 1
        if pr == res == 0:
            x4 = x4 + 1
        if pr == 1 and res == 0:
            x3 = x3 + 1
        if pr == 0 and res == 1:
            x2 = x2 + 1

    Recall = x1 / (x1 + x2)
    Accuracy = (x1 + x4) / (x1 + x2 + x3 + x4)
    Precision = x1 / (x1 + x3)
    F_measure = 2 * Precision * Recall / (Precision + Recall)
    print(", x1 = " + str(x1) + ", x2 = " + str(x2) + ", x3 = " + str(x3) + ", x4 = " + str(x4))
    print("Recall = " + str(Recall))
    print("Accuracy = " + str(Accuracy))
    print("Precision = " + str(Precision))
    print("F_measure = " + str(F_measure))


logistic(data_x, data_y)
```

<u>צילום מסך של תוצאה סופית:</u>



```
, x1 = 643, x2 = 0, x3 = 820, x4 = 12
Recall = 1.0
Accuracy = 0.4440677966101695
Precision = 0.4395078605604921
F_measure = 0.610636277302944
```