**Linear Regression with single and multiple variables**

[**Introduction:**](#_g66mjpagh597) **1**

[**2. Data**](#_209ap575b86x) **1**

[**3. Python code for Linear Regression with Single Variable**](#_1xeb10m6rqki) **2**

[**4. Python code for Linear Regression with Multiple Variable**](#_erfmtnijbah6) **4**

[4.1 Feature Normalization](#_9cymt3d0wkw3) 4

[4.2 Code: Linear Regression with Multiple Variable](#_i0m32hqyinwo) 4

# Introduction:

The goal of this project is to implement a linear regression model with one variable to predict profits for a food truck. Suppose you are the CEO of a restaurant franchise and are considering different cities for opening a new outlet. The chain already has trucks in various cities and you have data for profits and populations from the cities. You would like to use this data to help you select which city to expand to next.

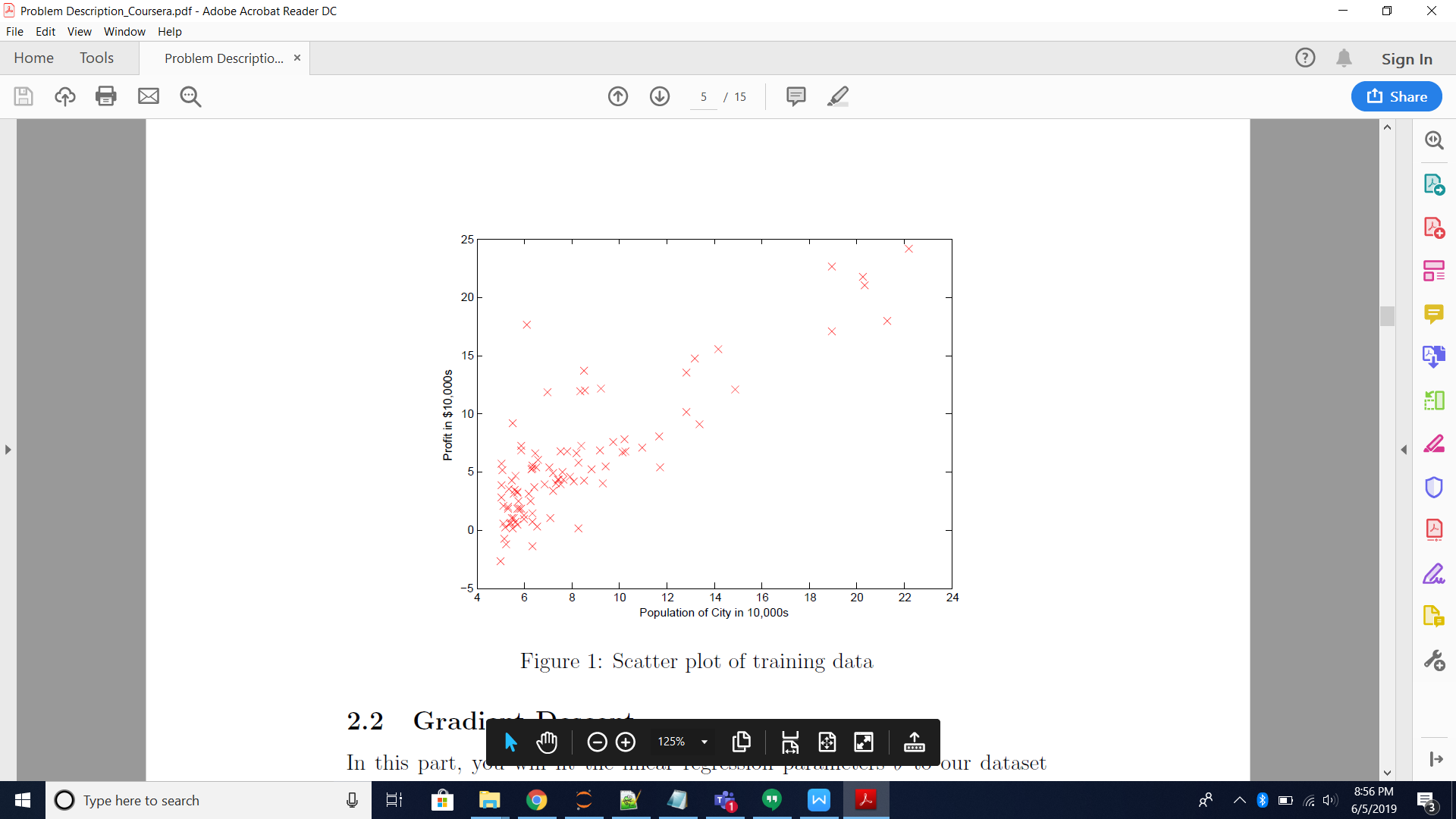
The le *ex1data1.txt* contains the dataset for our linear regression problem. The first column is the population of a city and the second column is the profit of a food truck in that city. A negative value for profit indicates a loss.

**Problem Statement:**

Given input data (population) and output data (profit of food truck), develop a linear regression model that can predict profit/loss for any new city with a known population.

# 2. Data

Before starting on any task, it is often useful to understand the data by visualizing it. For this dataset, you can use a scatter plot to visualize the data, since it has only two properties to plot (prot and population).

****

# 3. Python code for Linear Regression with Single Variable

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import sklearn

from sklearn import preprocessing

from sklearn import linear\_model

#from sklearn import cross\_validation

path='your/path/here/'

df=pd.read\_csv(path + 'ex1data1.txt',header=None)

df.columns=['Population','Profit']

df.head() # shows top five rows in the dataset

df.tail() # shows bottom five rows in the dataset

df.head() # shows top five rows in the dataset

# selecting linear regression model

# maximum iterations #1500

Lreg=linear\_model.LassoCV(eps=0.04,max\_iter=1500,tol=1e-4)

Lreg.fit(np.reshape(X,(-1,1)),y)

print('------ Linear Regression------------')

print('Accuracy of Linear Regression Model is ',round(Lreg.score(np.reshape(X,(-1,1)),y)\*100,2))

# output

# Accuracy of Linear Regression Model is 70.09

Prediction:

# predicting expected profits if the population is 35000 and 70,000

Predict1=Lreg.predict(np.reshape(3.5,(-1,1)))

Predict2=Lreg.predict(np.reshape(7,(-1,1)))

print('For a population of 35,000 we pridict a profit of ',Predict1\*10000)

print('For a population of 70,000 we pridict a profit of ',Predict2\*10000)

# For a population of 35,000 we pridict a profit of [5022.0880336]

# For a population of 70,000 we pridict a profit of [45108.01847837]

fig = plt.figure(figsize=(18,9))

a = .2

plt.scatter(X,y)

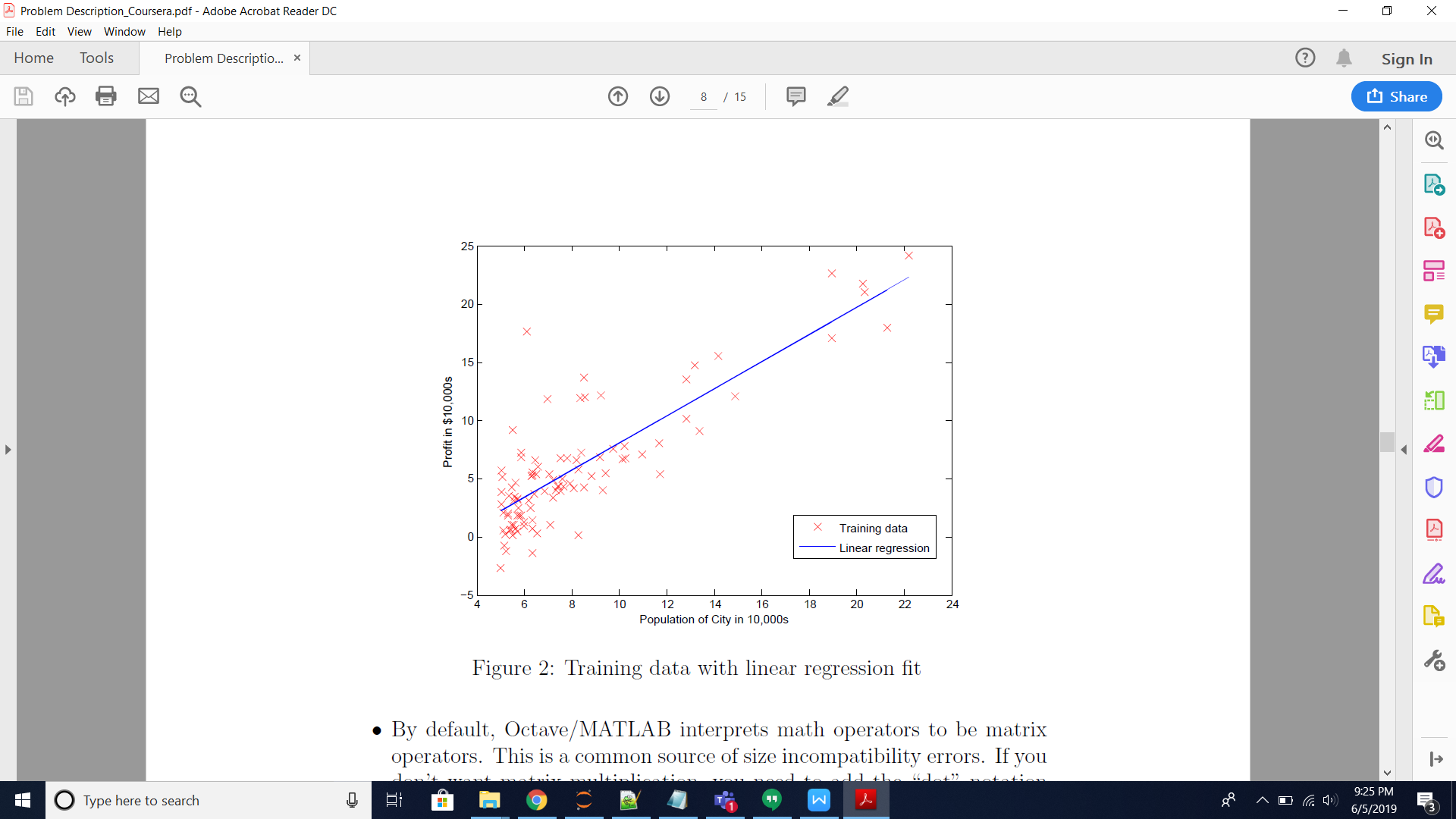
plt.grid(b=True, which='major', axis='x')

plt.xlabel("Population of City in 10,000s")

plt.ylabel("Profit in $10,000s")

plt.title("Populations versus Profit")

plt.grid()

****

# 4. Python code for Linear Regression with Multiple Variable

Suppose you are selling your house and you want to know what a good market price would be. One way to do this is to first collect information on recent houses sold and make a model of housing prices. The file *ex1data2.txt* contains a training set of housing prices in Portland, Oregon. The first column is the size of the house (in square feet), the second column is the number of bedrooms, and the third column is the price of the house. So this problem has two input variables ( size of house and number of bedrooms) and one output (price of house).

## 4.1 Feature Normalization

The *ex1\_multi.m* script will start by loading and displaying some values from this dataset. By looking at the values, note that house sizes are about 1000 times the number of bedrooms. When features differ by orders of magnitude, first performing feature scaling can make gradient descent converge much more quickly.

## 4.2 Code: Linear Regression with Multiple Variable

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import sklearn

from sklearn import preprocessing, linear\_model

# load data

path='C:/Users/Jeyarani/Desktop/Vishnu/Gitub\_Uploads/Logistic Regression/'

df=pd.read\_csv(path + 'ex2data2.txt',header=None)

df.columns=['X1','X2','Acceptance'] # rename columns

# Plot data

plt.figure()

idx1=df.index[df['Acceptance']==1]

idx2=df.index[df['Acceptance']==0]

plt.plot(df['X1'].loc[idx1],df['X2'].loc[idx1],'+',color='b',label='pass')

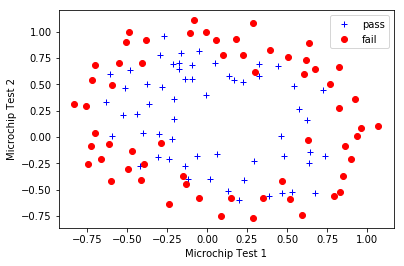
plt.plot(df['X1'].loc[idx2],df['X2'].loc[idx2],'o',color='r',label='fail')

plt.xlabel('Microchip Test 1')

plt.ylabel('Microchip Test 2')

plt.legend()

plt.show()



# feature mapping

# as the classifier is nonlinear, it is required to add more features to generate a better decision boundary

# features are added upto the power 6

count=len(df.columns)

for k in range(2,7):

for i in range(k+1):

for j in range(k+1):

if (i+j==k):

df['X'+str(count)]=df['X1'].pow(i)\*df['X2'].pow(j)

count+=1

## Inputs (X) and labels (y) (Score #1, Score #2, and admission status)

y=np.array(df['Acceptance'])

X=np.array(df.drop(['Acceptance'],1))

# normalize data

Sscaler=preprocessing.StandardScaler()

Xs=Sscaler.fit\_transform(X)

# logistic regression model

LogisticR=linear\_model.LogisticRegression(C=1e5)

#

LogisticR.fit(Xs,y)

#

#print('------ Logistic Regression------------')

print('Accuracy of Linear Regression Model is ',round(LogisticR.score(Xs,y)\*100,2))

# Output:

# Accuracy of Linear Regression Model is 87.29