Linear regression

Linear regression with one variable

Please import basic libraries and database::

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In this project, we will implement a linear regression model with one variable to predict profits for a food truck.The file

Dane

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.

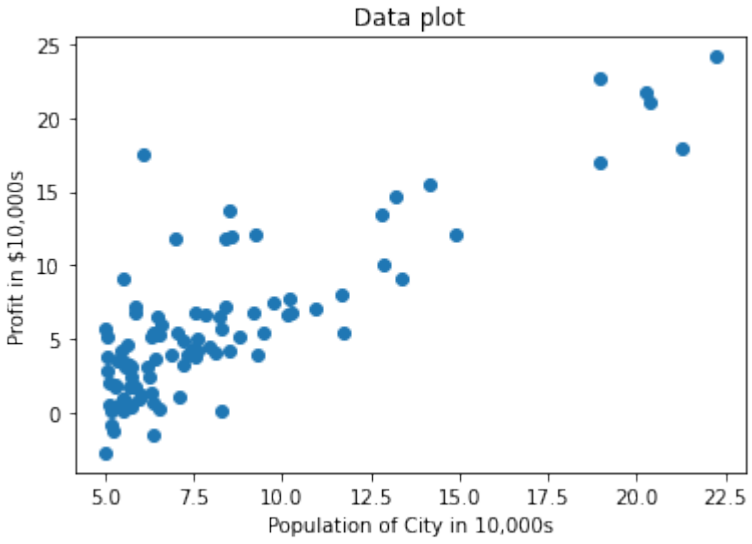
```
import os
path = os.getcwd() + '/ex1data1.txt'
data = pd.read_csv(path, header=None, names=['Population', 'Profit'])
```

Ex. 1. Use the `head` and `describe` function to examine the data and display few columns.
Ex. 2. Extract data from pandas structure to new variables of ndarray type

```
x = data.values[:, 0]
y = data.values[:, 1]
```

and plot the data, where: x - population, y - profit.

Example of a 2D plot:



Ex. 3. In order to make the cost function work (intercept value θ_0 has to be multiplied by 1) we need to insert a column of ones at the top of x.

Ex. 4. We need to separate our data into independent variables X and our dependent variable y. We also need to initialise parameter θ.

```
X = np.stack([np.ones(x.shape, dtype=np.float32), x], axis=0)
y.shape = [1, y.shape[0]]
theta = np.zeros((X.shape[0], 1), dtype=np.float32)
```

The goal of linear regression is to find a straight line which accurately represents the relationship between population and profit. This is called the hypothesis function and it's formulated as □(□)→□:

\begin{equation} f(x) = \theta_0 + \theta_1x_1+\theta_2x_2+...\theta_nx_n = \theta^Tx \label{eq1} \end{equation}

where $x \in R^N $ corresponds to a feature vector (N features) and $\theta \in R^N$ corresponds to the vector of model parameters. Parameters $\theta $ are the values that we will adjust to minimize the cost function.

The objective in training a linear regression model is to minimize a cost function, which measures the difference between actual y values in the training sample and predictions made by the hypothesis function: \begin{equation} J(\theta) = { \frac{1}{2m} }\sum_{i=1}^{m}\left(f(x_i)-y_i\right)^2 \label{eq2} \end{equation} **Ex. 5** The cost function evaluates the quality of our model by calculating the error between our model's prediction for a data point, using the model parameters, and the actual data point. Please write the cost function based on X,y and parameter θ.

```
def computeCost(X, y, theta):
    # 2-3 code lines as series of matrix operation
```

Ex. 6 Test the cost function with ($\theta = 0$).
Output: 32.07 (may differ...)

Gradient descent method
Since our hypothesis is based on the model parameters θ, we must somehow adjust them to minimize our cost function $J(\theta)$:

\begin{equation} J(\theta) = { \frac{1}{2m} }\sum_{i=1}^{m}\left(h_{\theta}(x_i)-y_i\right)^2 \label{eq3} \end{equation}

where the hypothesis $h_{\theta}(x)$ is given by the linear model:
\begin{equation} h_{\theta}(x)=\theta^{T}x=\theta_{0}+\theta_{1}x \label{eq4} \end{equation}

to optimize the θ parameter we will implement the batch gradient descent algorithm. *Batch* means that in each step we will use the whole training dataset.

So, how it works?

In each iteration, it takes a small step in the opposite gradient direction of $J(\theta)$. This makes the model parameters θ gradually come closer to the optimal values. This process is repeated until eventually the minimum cost is achieved.

More formally, gradient descent performs the following update in each iteration:

$\theta_{j} := \theta_{j} - \alpha \frac{\partial}{\partial \theta_{j}} J(\theta_{0},\theta_{1})$
Parameter α is called learning rate. It allows us to control the step size to update θ in each iteration. Choosing a too large learning rate may prevent us from converging to a minimum cost, whereas choosing a too small learning rate may significantly slow down the algorithm.

Based on eq. (1) we need to calculate the derivative: $\frac{\partial}{\partial \theta_{j}} J(\theta_{0},\theta_{1})$:

$\frac{\partial}{\partial \theta_{j}} J(\theta_{0},\theta_{1}) = \frac{\partial}{\partial \theta_{j}} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i)-y_i)^{2} = \frac{\partial}{\partial \theta_{j}} \frac{1}{2m} \sum_{i=1}^{m} (\theta_{0} + \theta_{1}x_i-y_i)^{2} $

We need the outcome for $ j = 0 $ and $ j = 1 $:

$ j = 0: \frac{\partial}{\partial \theta_{0}} J(\theta_{0},\theta_{1}) = \frac{1}{m} \sum_{i=1}^{m}(h_\theta(x_i)-y_i) $$ $$ j = 1: \frac{\partial}{\partial \theta_{1}} J(\theta_{0},\theta_{1}) = \frac{1}{m} \sum_{i=1}^{m}(h_\theta(x_i)-y_i)x_i $

Let's put the obtained values into the gradient formula:

$ \theta_{0} := \theta_{0} - \alpha \frac{1}{m} \sum_{i=1}^{m}(h_\theta(x_i)-y_i) $

$ \theta_{1} := \theta_{1} - \alpha \frac{1}{m} \sum_{i=1}^{m}(h_\theta(x_i)-y_i)x_i $

Of course, we improve the above parameters simultaneously.
Ex. 7 Please implement the gradient descent algorithm:

- the function returns optimized values θ and a vector of the cost function value for each iteration

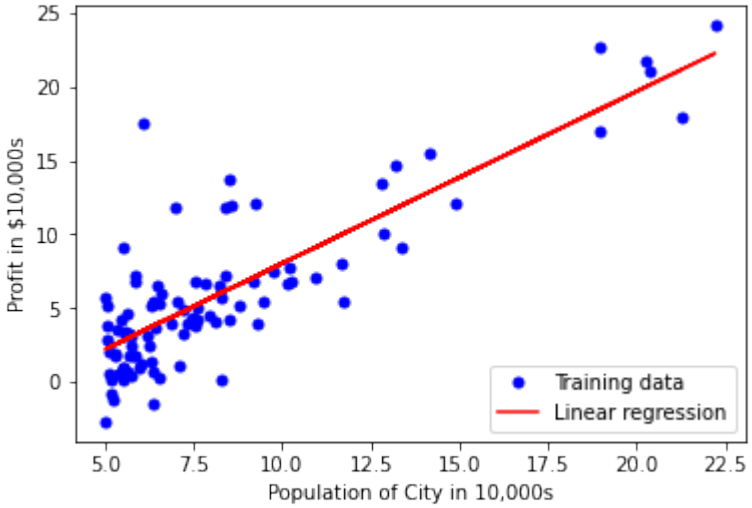
```
def simple_gradient(X, y, theta, alpha, it):
    # it - number of iterations


    return theta, cost
```

Parameter initialization:

```
alpha = 0.01
it = 1000
```

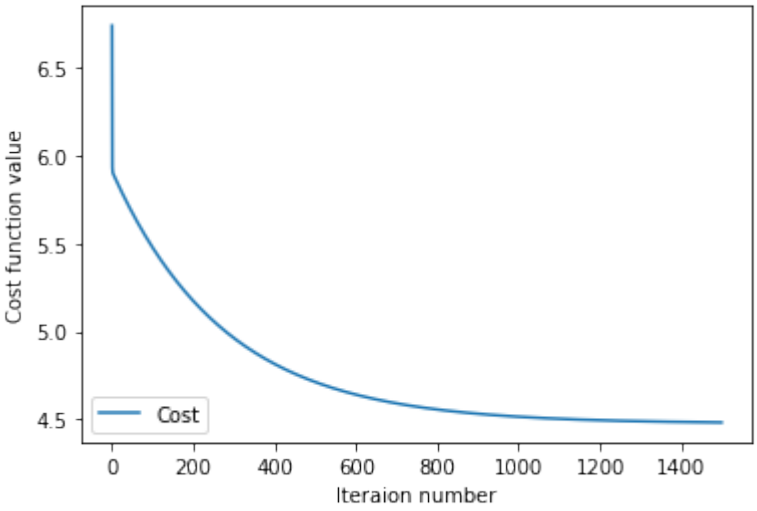
Ex. 8 Please calculate the optimal parameters for the dataset.
Ex. 9 Please calculate the cost function for the given θ (Ex.8).
Ex. 10 Plot the regression function and our dataset.

Expected 2D plot:



Ex. 11 Plot the relationship between the cost function and number of iterations.

Expected 2D plot:



Linear regression with multiple variables

To implement the linear regression algorithm for multiple variables we will use the

housing

database. We're given both the size of the house in square feet, and the number of bedrooms in the house.
Previously implemented functions will be used to complete also this task. If implemented correctly - in the vectorized version - they will also work for this example.
Read the data:

```
path = os.getcwd() + '/ex1data2.txt'
data2 = pd.read_csv(path, header=None, names=['Size', 'Bedrooms', 'Price'])
data2.head()
```

Ex. 1. Notice that the scale of the values for each variable is vastly different. A house will typically have 2-5 bedrooms but may have anywhere from hundreds to thousands of square feet. To fix this, we need to do something called “feature normalization”. One way to do this is by subtracting from each value in a feature the mean of that feature, and then dividing by the standard deviation (2 lines of code).
Ex.2 Please perform steps 3-11 from the previous task by using the same functions.

Linear regression - Python packages

An alternative solution to explicit implementation of the linear regression problem is to use the library `scikit-learn` with the object `LinearRegression`.

The implementation of any type of regression is very simple, we create an object of the appropriate class (`LinearRegression`, `Ridge`, `Lasso`, `ElasticNet`), on which we call the fit method, giving it the training set arguments, target values and possible parameters (□,□). Then we call `predict`, which will return the predicted value.

The solution template can be presented as follows:

```
import numpy as np
from sklearn import datasets, linear_model

# Data import


# Normalization


# Split into train and test sets (70-30%)


# Creating an object
regr = linear_model.LinearRegression()

# Learning model on training data
regr.fit(X_train, Y_train)
# Predicting values using test data
Y_predicted = regr.predict(X_test)

# Regression coefficients (theta)
print('Coefficients: \n', regr.coef_)

#  Residual sum of squares error
error = np.mean((regr.predict(X_test) - Y_test) ** 2)
print("Residual sum of squares: {}".format(error))
```