# Python for ML

### Week November 7th, 2020

### Prepared by: Moe Fadae & Najem Bouazza

#### Exercise from last session

In [ ]:

```
'''
review today's lesson and answer the following question :

Grab the number 34 from the this array:

arry = [23,4,[12,45,7,[2,67,8,4,8,[234,76,7,[231,6,6,[5,34,5]]]]]]

'''
```

In [46]:

```
import numpy as np
```

In [47]:

```
arry = np.array([23,4,[12,45,7,[2,67,8,4,8,[234,76,7,[231,6,6,[5,34,5]]]]]])
```

In [54]:

```
arry[2][3][5][3][3][1]
```

Out[54]:

```
34
```

In [ ]:

```
```

In [1]:

```
import numpy as np
arry = np.array([23,4,[12,45,7,[2,67,8,4,8,[234,76,7,[231,6,6,[5,34,5]]]]]])
```

In [8]:

```
arry[2][3][5][3][3][1]
```

Out[8]:

```
34
```

#### Question from last session : How to delete an element from a list??

In [55]:

```
L = [1,2,3,4,5]
L.remove(L[3])
L
```

Out[55]:

```
[1, 2, 3, 5]
```

# I - NumPy

In [1]:

```
import numpy as np
```

# - NumPy array Indexing / Slicing :

### Get an element in a list, 1d array, 2d array

In [19]:

```
import numpy as np

my_list = [1,'w', 98, "Get"]
array_1d = np.array([1,3,4,5,6,12])
array_2d = np.array([[1,2,3],[23,54,4],[21,54,67]])


print("Index for a list's element:",my_list[2])
print("Index for a 1d array's element:", array_1d[4])
print("Index for a 2d array's element:", array_2d[1][2])
```

```
Index for a list's element: 98
Index for a 1d array's element: 6
Index for a 2d array's element: 4
```

### Get a slice in a list, 1d array, 2d array

In [20]:

```
print("slice from 1d array:", array_1d[1:])
print("slice from 2d array_:", array_2d[1:2,1:3])
```

```
slice from 1d array: [ 3  4  5  6 12]
slice from 2d array_: [[54  4]]
```

# II - Pandas :

In [2]:

```
import pandas as pd
```

## 1 - Series and DataFrames

Series are similar to numpy arrays: what differenciate Pandas Series from Numpy arrays is that we can index Series using labels.

In [ ]:

```
```

In [60]:

```
labels = ["a", "b", "c"]
my_data = [11,"hello", 33]
arry = np.array(my_data)
S = pd.Series(data= my_data)
S
```

Out[60]:

```
0       11
1    hello
2       33
dtype: object
```

In [62]:

```
S[2]
```

Out[62]:

```
33
```

In [63]:

```
# Convert a NumPy arry into a pandas Series

pd.Series(arry)
```

Out[63]:

```
0       11
1    hello
2       33
dtype: object
```

In [13]:

```
#Convert a dictionary into Series

d = {"one": 11, "two": 22, "three": 33}
pd.Series(d)
```

Out[13]:

```
one      11
two      22
three    33
dtype: int64
```

In [14]:

```
# Series can hold any type of data (Strings / numbers /built in functions ...)

pd.Series(data = [sum, print, len])
```

Out[14]:

```
0      <built-in function sum>
1    <built-in function print>
2      <built-in function len>
dtype: object
```

**Pandas** provides in-memory 2d table object called Dataframe

In [64]:

```
from numpy.random import rand

df = pd.DataFrame(data = rand(4,3), index = ["first", "second", "Third", "Fourth'
```

In [65]:
```
df
```

Out[65]:

|  | a | b | c |
|---|---|---|---|
| **first** | 0.690696 | 0.772040 | 0.000844 |
| **second** | 0.948885 | 0.979848 | 0.785738 |
| **Third** | 0.644835 | 0.390349 | 0.462431 |
| **Fourth** | 0.293501 | 0.181962 | 0.328634 |

In [66]:
```
df["b"]
```

Out[66]:
```
first     0.772040
second    0.979848
Third     0.390349
Fourth    0.181962
Name: b, dtype: float64
```

In [ ]:

### A Dataframe is a bench of Series charing the same index !!!!

In [41]:
```
#df[name of column] to grab a specific column
# or df.columnname
df["b"]
df.a  # it's not recommanded to use this method as it induces confusion with meth

#df.(hit Tab on the keyboard) to see the available methods
```

Out[41]:
```
first     0.546217
second    0.539157
Third     0.742983
Fourth    0.717028
Name: a, dtype: float64
```

In [69]:
```
df[["a","b"]]
```

Out[69]:

|  | a | b |
|---|---|---|
| **first** | 0.690696 | 0.772040 |
| **second** | 0.948885 | 0.979848 |
| **Third** | 0.644835 | 0.390349 |
| **Fourth** | 0.293501 | 0.181962 |

In [42]:
```
# Grab mutliple columns, use the brakets and a list of columns names inside the b
df[["a", "c"]]
```

Out[42]:

| | a | c |
|---|---|---|
| **first** | 0.546217 | 0.177938 |
| **second** | 0.539157 | 0.126492 |
| **Third** | 0.742983 | 0.306025 |
| **Fourth** | 0.717028 | 0.361270 |

In [79]:
```
df["new_column"] = df["a"]*100
df
```

Out[79]:

| | a | b | c | Sum | new_column |
|---|---|---|---|---|---|
| **first** | 0.690696 | 0.772040 | 0.000844 | 1.462736 | 69.069582 |
| **second** | 0.948885 | 0.979848 | 0.785738 | 1.928732 | 94.888462 |
| **Third** | 0.644835 | 0.390349 | 0.462431 | 1.035184 | 64.483483 |
| **Fourth** | 0.293501 | 0.181962 | 0.328634 | 0.475463 | 29.350100 |

In [80]:
```
# Create new column :
#df["Sum"] = df["a"] + df["b"]
#df

# to eliminate a column : df.drop(column name, axis = 1)

df.drop("new_column", axis = 1)
```

Out[80]:

| | a | b | c | Sum |
|---|---|---|---|---|
| **first** | 0.690696 | 0.772040 | 0.000844 | 1.462736 |
| **second** | 0.948885 | 0.979848 | 0.785738 | 1.928732 |
| **Third** | 0.644835 | 0.390349 | 0.462431 | 1.035184 |
| **Fourth** | 0.293501 | 0.181962 | 0.328634 | 0.475463 |

In [81]:
```
df
```

Out[81]:

| | a | b | c | Sum | new_column |
|---|---|---|---|---|---|
| **first** | 0.690696 | 0.772040 | 0.000844 | 1.462736 | 69.069582 |
| **second** | 0.948885 | 0.979848 | 0.785738 | 1.928732 | 94.888462 |
| **Third** | 0.644835 | 0.390349 | 0.462431 | 1.035184 | 64.483483 |
| **Fourth** | 0.293501 | 0.181962 | 0.328634 | 0.475463 | 29.350100 |

In [82]:
```
# Pandas wants you to confirm if you want to drop a column by using "inplace"

df.drop("new_column", axis = 1, inplace = True)
df
```

Out[82]:

| | a | b | c | Sum |
|---|---|---|---|---|
| **first** | 0.690696 | 0.772040 | 0.000844 | 1.462736 |
| **second** | 0.948885 | 0.979848 | 0.785738 | 1.928732 |
| **Third** | 0.644835 | 0.390349 | 0.462431 | 1.035184 |
| **Fourth** | 0.293501 | 0.181962 | 0.328634 | 0.475463 |

In [60]:
```
df.drop("first", inplace = True)
```

In [61]:
```
df
```

Out[61]:

| | a | b | c |
|---|---|---|---|
| **second** | 0.539157 | 0.278046 | 0.126492 |
| **Third** | 0.742983 | 0.508975 | 0.306025 |
| **Fourth** | 0.717028 | 0.820784 | 0.361270 |

In [83]:
```
#Why we refer to axis = 0 for rows and axis = 1for columns : because it's taken f
df.shape
```

Out[83]:
```
(4, 4)
```

In [ ]:
```
###### Selecting Rows
# we have to use a method to do this
# 1 df.loc[label of index, rows] Label based index
# 2 df.iloc[index number] numerical based index
```

In [88]:
```
df.iloc[2]
#df.loc["first"]
# Error ? Why ?
```

Out[88]:
```
a      0.644835
b      0.390349
c      0.462431
Sum    1.035184
Name: Third, dtype: float64
```

In [63]:
```
df.loc["Third"]
```

Out[63]:
```
a    0.742983
b    0.508975
c    0.306025
Name: Third, dtype: float64
```

In [64]:
```
df.iloc[1]
```

Out[64]:
```
a    0.742983
b    0.508975
c    0.306025
Name: Third, dtype: float64
```

In [89]:
```
df
```

Out[89]:

| | a | b | c | Sum |
|---|---|---|---|---|
| **first** | 0.690696 | 0.772040 | 0.000844 | 1.462736 |
| **second** | 0.948885 | 0.979848 | 0.785738 | 1.928732 |
| **Third** | 0.644835 | 0.390349 | 0.462431 | 1.035184 |
| **Fourth** | 0.293501 | 0.181962 | 0.328634 | 0.475463 |

In [90]:
```
df.loc["second", "b"]
```

Out[90]:
```
0.9798478061198371
```

In [91]:
```
df.iloc[1,1]
```

Out[91]:
```
0.9798478061198371
```

In [65]:
```
# Selecting a subset of rows/columns

df.loc["Third", "b"]
```

Out[65]:
```
0.5089752424453037
```

In [68]:
```
df.loc[["second", "Third"] , ["a","c"]]
```

Out[68]:

| | a | c |
|---|---|---|
| **second** | 0.539157 | 0.126492 |
| **Third** | 0.742983 | 0.306025 |

In [76]:
```
# Conditional Selection

cond = df> 0.5

df[cond]
```

Out[76]:

| | a | b | c |
|---|---|---|---|
| **second** | 0.539157 | NaN | NaN |
| **Third** | 0.742983 | 0.508975 | NaN |
| **Fourth** | 0.717028 | 0.820784 | NaN |

In [96]:
```
df[df["a"]>0.6]["b"]
```

Out[96]:
```
first     0.772040
second    0.979848
Third     0.390349
Name: b, dtype: float64
```

In [94]:
```
df1 = df[df["a"]>0.6][["b","c"]]
df1
```

Out[94]:

| | b | c |
|---|---|---|
| **Third** | 0.508975 | 0.306025 |
| **Fourth** | 0.820784 | 0.361270 |

In [ ]:

# III - Functions

**Create a function :**

using the **def** keyword

It's a block of code which only runs when it is called

In [97]:
```
def my_function():
    print("Hello from a function")
```

**Calling a function :**

use the function name followed by parenthesis

In [98]:
```
my_function()
```

```
Hello from a function
```

In [100]:
```
def my_function(name = "default"):
    print("Hello my name is {}".format(name))
```

In [102]:
```
my_function("Jack")
```

```
Hello my name is Jack
```

In [ ]:

In [104]:

```
def Add(a, b):
    """
    This function returns the sum of 2 numbers
    """
    print("The first number is:", a)
    print("The second number is:",b)
    print("the sum of {} + {} is :".format(a,b), a+b)
    return a + b
```

In [106]:

```
Add(4,8)
```

```
The first number is: 4
The second number is: 8
the sum of 4 + 8 is : 12
```

Out[106]:

```
12
```

In [ ]:

# IV - Linear Regression :

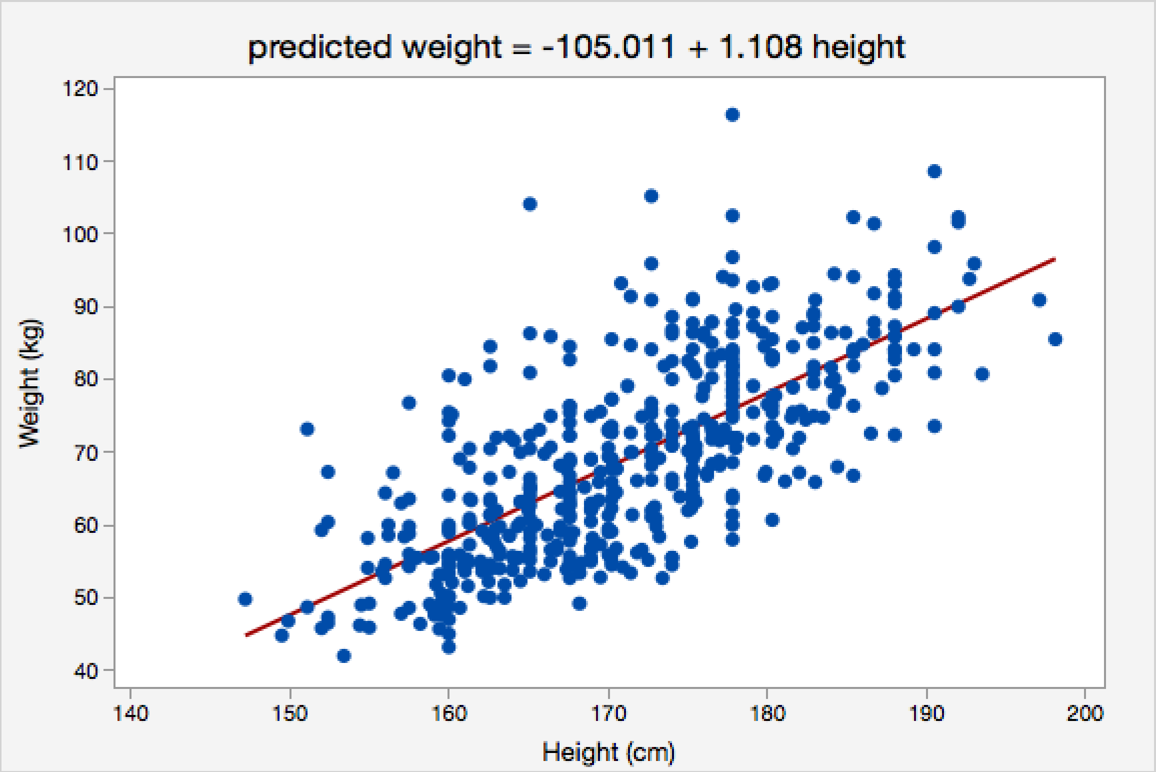
## Practice on *house_price* data

In this part, you will understand and learn how to implement the following Machine Learning Regression model:

**Simple Linear Regression**

- Fit a line to a dataset of observations
- Use this line to predict unobserved values

Exemple of Linear regression model :



## STEP 1 : Importing the libraries

In [107]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

## STEP 2 : Importing the Dataset

In [110]:

```
pd.read_csv("house_price.csv")
```

Out[110]:

|  | LotArea | SalePrice |
|---|---|---|
| **0** | 8450 | 208500 |
| **1** | 9600 | 181500 |
| **2** | 11250 | 223500 |
| **3** | 9550 | 140000 |
| **4** | 14260 | 250000 |
| **...** | ... | ... |
| **1402** | 7917 | 175000 |
| **1403** | 13175 | 210000 |
| **1404** | 9042 | 266500 |
| **1405** | 9717 | 142125 |
| **1406** | 9937 | 147500 |

1407 rows × 2 columns

In [119]:

```
# Importing the dataset
dataset = pd.read_csv('house_price.csv')
X = dataset.iloc[:, 0].values # .values convert dataframe into numpy array
y = dataset.iloc[:, 1].values # dataset.to_numpy()

dataset
```

Out[119]:

|  | LotArea | SalePrice |
|---|---|---|
| **0** | 8450 | 208500 |
| **1** | 9600 | 181500 |
| **2** | 11250 | 223500 |
| **3** | 9550 | 140000 |
| **4** | 14260 | 250000 |
| **...** | ... | ... |
| **1402** | 7917 | 175000 |
| **1403** | 13175 | 210000 |
| **1404** | 9042 | 266500 |
| **1405** | 9717 | 142125 |
| **1406** | 9937 | 147500 |

1407 rows × 2 columns

## STEP 3 : Splitting the dataset into training_set and test_set

In [113]:

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

## STEP 4 : Training the model using the training_set

In [115]:

```
# Training the Simple Linear Regression model on the Training set
# import linear_model from sklearn

from sklearn.linear_model import LinearRegression

regressor = LinearRegression() # created our model

regressor.fit(X_train, y_train) # Train the model
```

Out[115]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

## STEP 5 : Predicting the results for the test_set

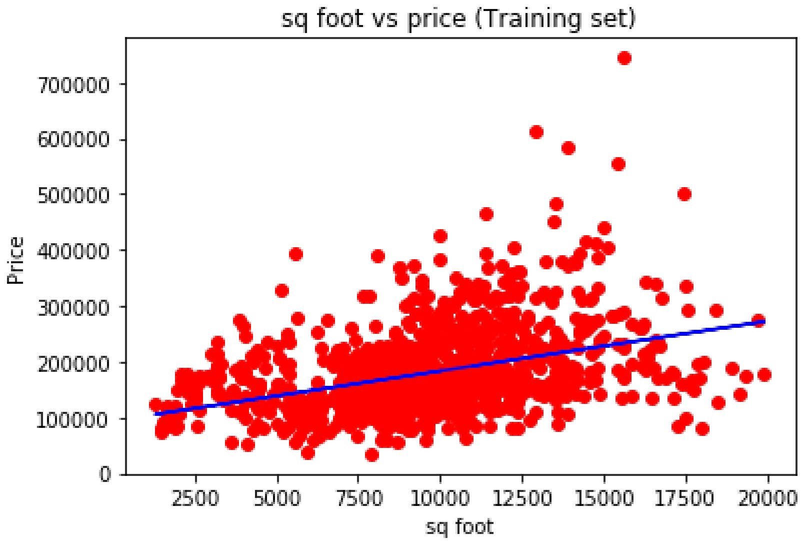
In [132]:

```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
#print(y_pred)
```

## STEP 6 : Visualizing the results

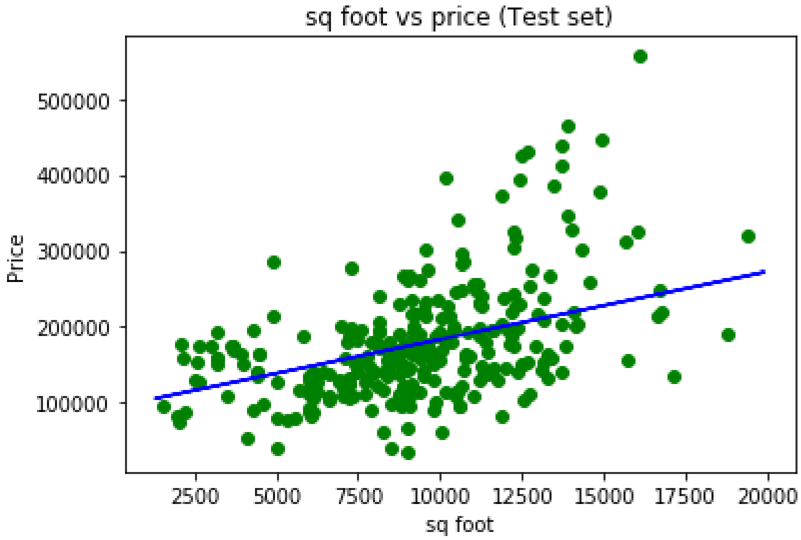
In [127]:

```
# Visualising the Training set results
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('sq foot vs price (Training set)')
plt.xlabel('sq foot')
plt.ylabel('Price')
plt.show()
```



In [128]:

```
# Visualising the Test set results
plt.scatter(X_test, y_test, color = 'green')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('sq foot vs price (Test set)')
plt.xlabel('sq foot')
plt.ylabel('Price')
plt.show()
```



In [129]:

```
# Model Intercept and Slope
intercept = regressor.intercept_
slope = regressor.coef_

print ("slope is: ", slope, " and intercept is: " , intercept)
```

```
slope is:  [8.95119579]  and intercept is:  93999.9307251301
```

In [131]:

```
# R-squared
r_squared = regressor.score(X.reshape(-1,1), y.reshape(-1,1))
print ("R_squared is: ", r_squared)
```

```
R_squared is:  0.1746813314901382
```

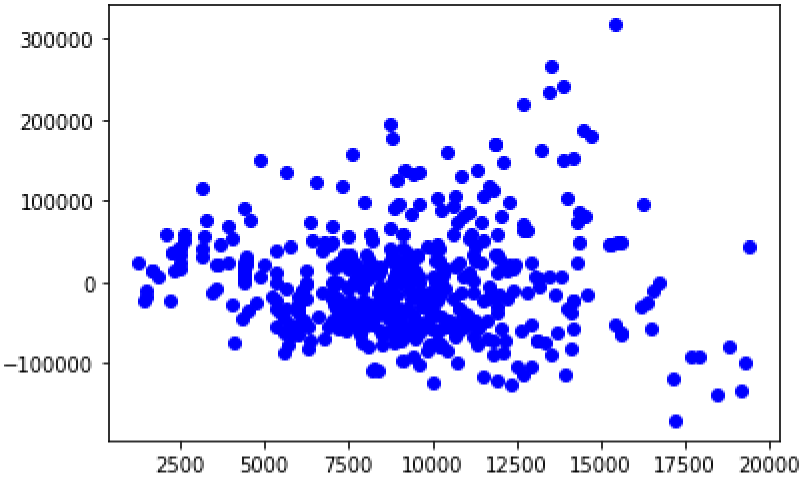
In [32]:

```
#Plot residuals
# calculate residuals
resi = y_test - y_pred

plt.plot (X_test, resi, 'bo')
```

Out[32]:

```
[<matplotlib.lines.Line2D at 0xc3c0748>]
```



In [134]:

```
#Predict a value using the trained model

regressor.predict(np.array([[15000]]))
```

Out[134]:

```
array([228267.86757217])
```

In [ ]: