Module 4 - Python Functions and Linear Regression Basics

Author: Favio Vázquez and Jessica Cervi

Index:

Instructions:

Welcome to Module 4. In this module, you learned about how to define Python functions and the basics of linear regression. We will practice linear regression with two libraries: `statsmodel` and `scikit-learn`.

Make sure to watch the coding demos before doing the assigment!

Importing the libraries

Before getting started, make sure that you can run the cell below with no issues. We will be importing all the libraries to work on this assigment.

In [1]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn import linear_model
from sklearn import metrics
```

Part 1. Python Functions

Question 1

Create a simple Python function called `Hello_world` that returns the String `"Hello World!"`.

In [2]:
```
### GRADED
### YOUR SOLUTION HERE
def Hello_world():
    return "Hello World!"

###
### YOUR CODE HERE
###
```

In [3]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Question 2

Assign the integer 5 to a variable called `x` and the integer 3 to a variable called `y`. Create a Python function called `plus` that takes two numbers as arguments and returns the sum of them. Use the function with `x` and `y` and assign the result to a variable called `total`.

In [4]:
```
### GRADED
### YOUR SOLUTION HERE
x = 5
y = 3
def plus(x, y):
    total = x+y
    return total
total = plus(x,y)
print(total)
###
### YOUR CODE HERE
###
```

8

In [5]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Question 3

Create a Python function called `plus_args` that takes a variable number of arguments and returns the sum of them. Then call the function to sum the numbers `1,4,2,7` and assign the result to a variable called `sum_total`.

In [6]:
```
def test(*args):
    print(args)

test(1,2,3)
```

```
(1, 2, 3)
```

In [7]:
```
### GRADED
### YOUR SOLUTION HERE
def plus_args(*args): #use arterics args and it makes it a tuple
    total = 0
    for i in args:
        total += i
    return total

sum_total = plus_args(1,4,2,7)
print(sum_total)

###
### YOUR CODE HERE
###
```

```
14
```

In [8]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Question 4

Define a lambda function called `add_one` that adds `1` to a variable `x`. Use this function to add 1 to 89 and assign the result to the variable `y`.

In [9]:
```
(lambda x: x+2)(2)
```

Out[9]:
```
4
```

In [10]:
```
### GRADED
### YOUR SOLUTION HERE
add_one = lambda x:x+1
y = add_one(89)
print(y)
###
### YOUR CODE HERE
###
```

```
90
```

In [11]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Part 2. Linear Regression

Question 5

Using only the statsmodel library, read the file `data/data.csv` and assign to a Pandas dataframe called `bikes`. Perform a simple linear regression using the variable `temp` to predict the variable `count`. Save your **fitted model** in a variable called `count_model`.

Hint: Remember to add a constant that will work as the Bias or Y-intercept. Use the `sm.OLS()` method.

When you create an x variable you need to also add a constant

X = bikes['columns'] X = sm.add_constant(X)

y = ... Check!

Check which arguments are passed! count_model = sm.OLS(ARGUMENTS).fit()

Run this cell to load the dataset bikes = pd.read_csv("data/data.csv") bikes.head(1)

In [12]:
```
bikes = pd.read_csv("data/Mod4_data.csv")
bikes.head(1)
```

Out[12]:

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	hour	year
0	2011-01-01 00:00:00	1	0	0	1	9.84375	14.398438	81	0.0	3	13	16	0	2011

In [13]:
```
### GRADED
### YOUR SOLUTION HERE

import statsmodels.api as sm

X = bikes['temp']
Y = bikes['count']

X = sm.add_constant(X)

count_model = sm.OLS(Y, X).fit()
count_model.summary()


###
### YOUR CODE HERE
###
```

Out[13]:

OLS Regression Results

Dep. Variable:	count	**R-squared:**	0.156
Model:	OLS	**Adj. R-squared:**	0.156
Method:	Least Squares	**F-statistic:**	2006.
Date:	Mon, 12 Aug 2024	**Prob (F-statistic):**	0.00
Time:	19:58:34	**Log-Likelihood:**	-71125.
No. Observations:	10886	**AIC:**	1.423e+05
Df Residuals:	10884	**BIC:**	1.423e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	6.0523	4.439	1.363	0.173	-2.649	14.754
temp	9.1704	0.205	44.784	0.000	8.769	9.572

Omnibus:	1871.808	**Durbin-Watson:**	0.369
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3222.277
Skew:	1.123	**Prob(JB):**	0.00
Kurtosis:	4.434	**Cond. No.**	60.4

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [14]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Question 6

Using the dataframe `bikes` from above, use the statsmodel library to perform a simple linear regression using the variables `temp` and `humidity` to predict the variable `casual`. Save your model in a variable called `casual_model`.

Hint: Remember to add a constant that will work as the Bias or Y-intercept. Use the `sm.OLS()` method.

X = dataframe[["column1", "column2"]] Import to use two variables

In [15]:
```
### GRADED
### YOUR SOLUTION HERE
casual_model = None


X = bikes[['temp', 'humidity']]
Y = bikes['casual']

X = sm.add_constant(X)

casual_model = sm.OLS(Y, X).fit()
casual_model.summary()
```

```
###
### YOUR CODE HERE
###
print(casual_model)
```

```
<statsmodels.regression.linear_model.RegressionResultsWrapper object at 0x7f9b101831c0>
```

In [16]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Question 7

Using the dataframe `bikes` from above, use the statsmodel library to perform a multiple linear regression using the variables `temp`, `humidity`, `season` and `holiday` to predict the variable `count`. Save your model in a variable called `model_multiple`.

Hint: Remeber to add a constant that will work as the Bias or Y-intercept. Use the `sm.OLS()` method.

In [17]:
```
### GRADED
### YOUR SOLUTION HERE

X = bikes[['temp', 'humidity', 'season','holiday']]
Y = bikes['count']

X = sm.add_constant(X)

model_multiple = sm.OLS(Y, X).fit()
model_multiple.summary()

model_multiple.summary()

###
### YOUR CODE HERE
###
```

Out[17]:

OLS Regression Results

Dep. Variable:	count	**R-squared:**	0.258
Model:	OLS	**Adj. R-squared:**	0.258
Method:	Least Squares	**F-statistic:**	945.5
Date:	Mon, 12 Aug 2024	**Prob (F-statistic):**	0.00
Time:	19:58:34	**Log-Likelihood:**	-70422.
No. Observations:	10886	**AIC:**	1.409e+05
Df Residuals:	10881	**BIC:**	1.409e+05
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	164.2718	6.709	24.487	0.000	151.122	177.422
temp	7.8573	0.200	39.243	0.000	7.465	8.250
humidity	-3.0272	0.080	-37.952	0.000	-3.184	-2.871
season	22.3278	1.421	15.708	0.000	19.542	25.114
holiday	-9.6923	8.984	-1.079	0.281	-27.302	7.917

Omnibus:	2099.893	**Durbin-Watson:**	0.428
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3986.031
Skew:	1.189	**Prob(JB):**	0.00
Kurtosis:	4.770	**Cond. No.**	407.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [18]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Question 8

Using the dataframe `bikes` from above, use the scikit-learn library to perform a simple linear regression using only the variable `temp` to predict the variable `count`. Save your model in a variable called `model_sci`.

Then save your intercept in a variable called `intercept_simple` and your coefficients in a variable called `coefs_simple`.

Hint: Use the `linear_model.LinearRegression()` method.

In [19]:
```
### GRADED
### YOUR SOLUTION HERE

from sklearn import linear_model

X = bikes[['temp']]
Y = bikes['count']

regr = linear_model.LinearRegression()
regr.fit(X,Y)

model_sci = regr
intercept_simple = regr.intercept_
coefs_simple = regr.coef_
###
### YOUR CODE HERE
###
```

In [20]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Question 9

Predict the value of `count` at `temp = 78`. Assign the result to `count_predict`.

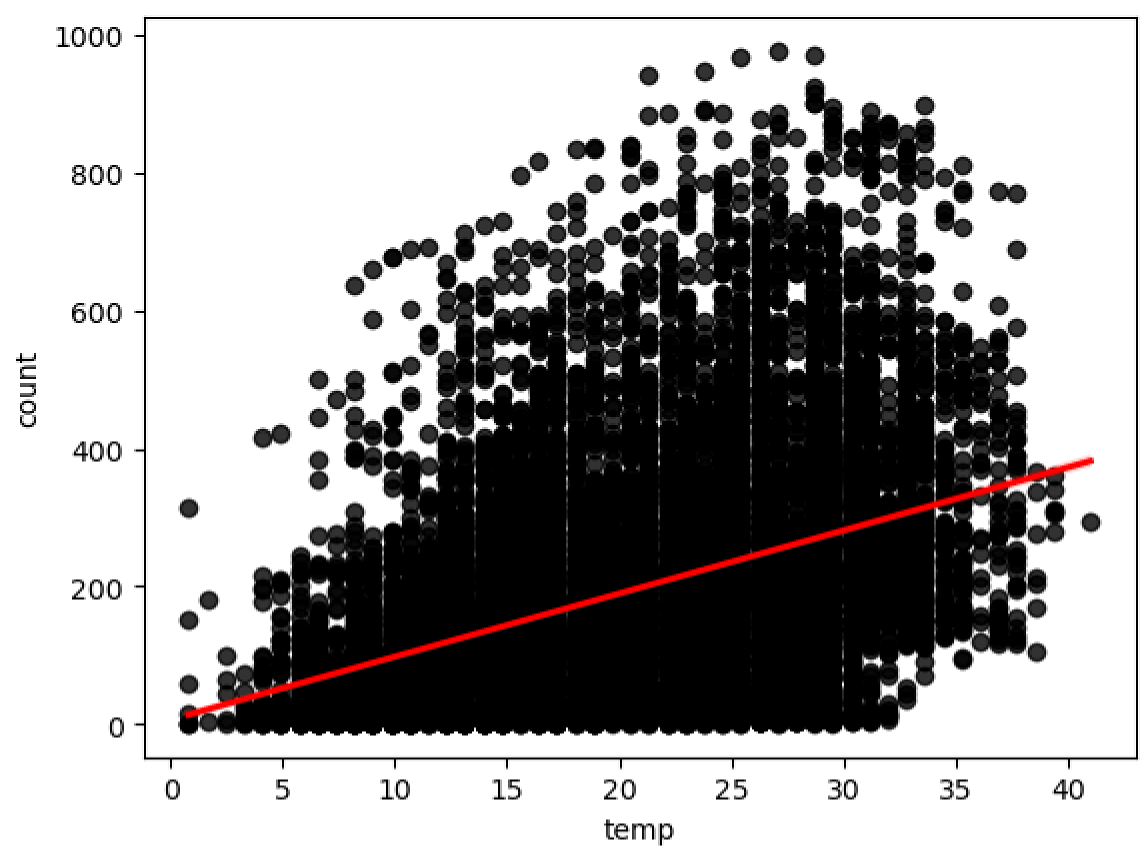
model_sci.predict(ARGUMENT)

In [23]:
```
import seaborn as sns
import matplotlib as plt

sns.regplot(x = X, y = Y, data = bikes, scatter_kws={"color": "black"}, line_kws={"color": "red"})

#sns.show()
```

Out[23]:
```
<AxesSubplot:xlabel='temp', ylabel='count'>
```



In [24]:
```
### GRADED
### YOUR SOLUTION HERE
count_predict = model_sci.predict([[78]])
print(78*regr.coef_ + regr.intercept_)

count_predict
###
### YOUR CODE HERE
###
```

```
[721.34719247]
/Users/dempseywade/opt/anaconda3/lib/python3.9/site-packages/sklearn/base.py:450: UserWarning: X does not hav
e valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

Out[24]:
```
array([721.34719247])
```

In [25]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Question 10

Using the dataframe `bikes` from above, use the scikit-learn library to perform a simple linear regression using only the variables `temp`, `humidity`, `season` and `holiday` to predict the variable `count`. Save your model in a variable called `model_sci_multi`.

Hint: Use the `linear_model.LinearRegression()` method.

In [26]:
```
### GRADED
### YOUR SOLUTION HERE

X = bikes[['temp','humidity','season','holiday']]
Y = bikes['count']

rerg = linear_model.LinearRegression()
regr.fit(X,Y)

model_sci_multi = regr
###
### YOUR CODE HERE
###
```

In [27]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac 1n\sum_{i=1}^n|y_i-\hat{y}_i|$$
Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac 1n\sum_{i=1}^n(y_i-\hat{y}_i)^2$$
Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac 1n\sum_{i=1}^n(y_i-\hat{y}_i)^2}$$
Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, hence we want to minimize them.

Question 11

Suppose a model has some true and some predicted values. Define the *true* values in a list called `x_true` which contains the following values: 10,20,35,60,87. Define the *predicted* values in a list called `x_pred` with entries: 14,22,38,79, 93.

Using scikit-learn, compute the Mean Absolute Error (MAE). Assign the value to a variable called `mae`.

metrics.mean_absolute_error(ARGUMENTS)

related to x_true and x_pred

In [28]:
```
### GRADED
### YOUR SOLUTION HERE
x_true = [10,20,35,60,87]
x_pred = [14,22,38,79,93]
mae = metrics.mean_absolute_error(x_true, x_pred)

mae
###
### YOUR CODE HERE
###
```

Out[28]:
```
6.8
```

In [29]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```

Question 12

With the same previous true and predicted values, compute the Mean Squared Error (MSE). Assign the value to a variable called `mse`.

In [30]:
```
### GRADED
### YOUR SOLUTION HERE
mse = metrics.mean_squared_error(x_true, x_pred)
mse
###
### YOUR CODE HERE
###
```
Out[30]:
```
85.2
```
In [31]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```
Question 13

With the same previous true and predicted values, compute the Root Mean Squared Error (RMSE). Assign the value to a variable called `rmse`.

In [32]:
```
### GRADED
### YOUR SOLUTION HERE
import math
rmse = math.sqrt(metrics.mean_squared_error(x_true, x_pred))
rmse
###
### YOUR CODE HERE
###
```
Out[32]:
```
9.23038460737146
```
In [33]:
```
###
### AUTOGRADER TEST - DO NOT REMOVE
###
```
In [ ]:
```
```
In [ ]:
```
```