Lesson 9: Statistical measure & Data Analysis using Python

9.1. Introduction

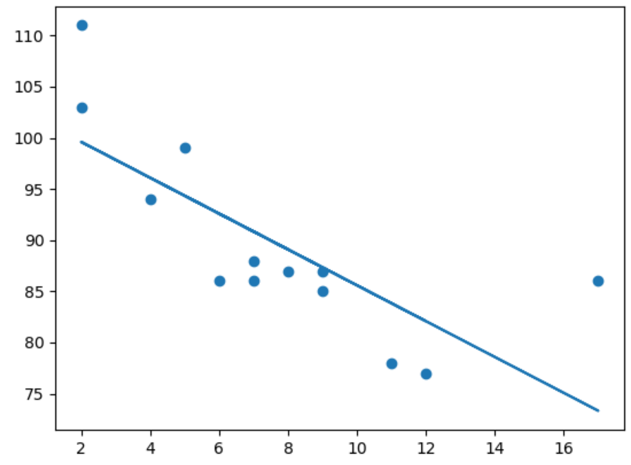
The term regression is used when you try to find the relationship between variables.

In data analysis, and in statistical modeling, that relationship is used to predict the outcome of future events.

Python is used in exploring many different types of data. You will learn how to prepare data for analysis, perform simple statistical analysis, create meaningful data visualizations, and predict future trends from data.

9.2. Linear Regression

Linear regression uses the relationship between the data-points to draw a straight line through all them. This line can be used to predict future values.



In Machine Learning, predicting the future is very important.

How Does it Work?

Python has methods for finding a relationship between data-points and to draw a line of linear regression.

In the example below, the x-axis represents age, and the y-axis represents speed. We have registered the age and speed of 13 cars as they were passing a tollbooth. Let us see if the data we collected could be used in a linear regression:

Example

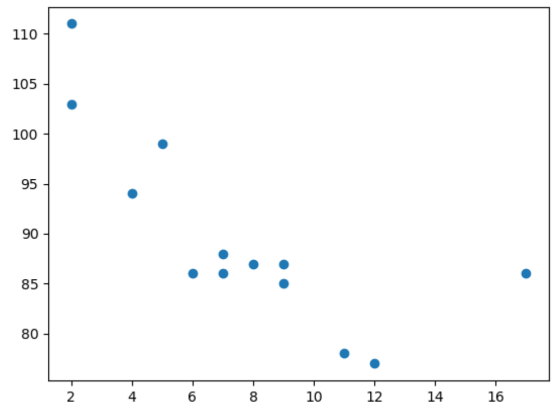
Start by drawing a scatter plot:

```
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()
```

Result:

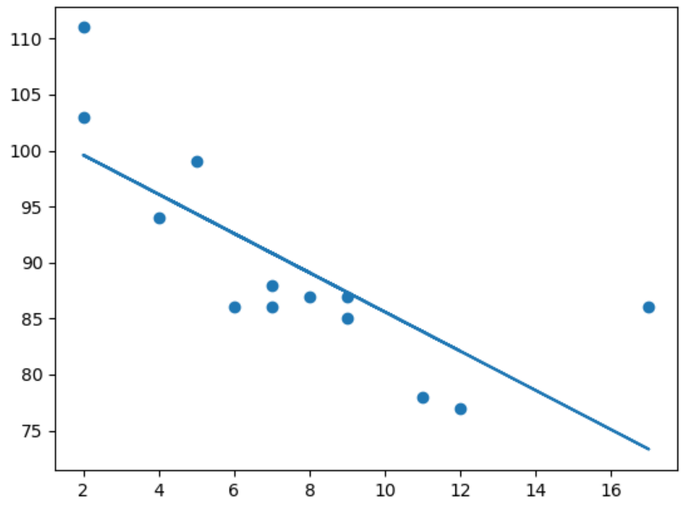


Example

Import scipy and draw the line of Linear Regression:

```
import matplotlib.pyplot as plt
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Result:



Example Explained

Import the modules you need.

```
import matplotlib.pyplot as plt
from scipy import stats
```

Create the arrays that represent the values of the x and y axis:

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

Execute a method that returns some important key values of Linear Regression:

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

Create a function that uses the slope and intercept values to return a new value. This new value represents where on the y-axis the corresponding x value will be placed:

```
def myfunc(x):
  return slope * x + intercept
```

Run each value of the x array through the function. This will result in a new array with new values for the y-axis:

```
mymodel = list(map(myfunc, x))
```

Draw the original scatter plot:

```
plt.scatter(x, y)
```

Draw the line of linear regression:

```
plt.plot(x, mymodel)
```

Display the diagram:

```
plt.show()
```

R for Relationship

It is important to know how the relationship between the values of the x-axis and the values of the y-axis is, if there are no relationship the linear regression can not be used to predict anything.

This relationship - the coefficient of correlation - is called r.

The r value ranges from -1 to 1, where 0 means no relationship, and 1 (and -1) means 100% related.

Python and the Scipy module will compute this value for you, all you have to do is feed it with the x and y values.

Example

How well does my data fit in a linear regression?

```
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)
```

Note: The result -0.76 shows that there is a relationship, not perfect, but it indicates that we could use linear regression in future predictions.

Predict Future Values

Now we can use the information we have gathered to predict future values.

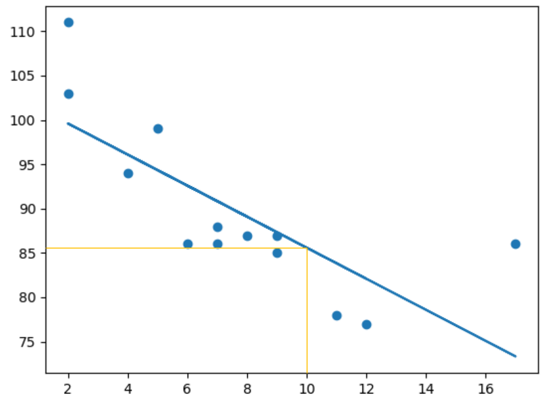
Example: Let us try to predict the speed of a 10 years old car.
To do so, we need the same myfunc() function from the example above:

```
def myfunc(x):
  return slope * x + intercept
```

Example
Predict the speed of a 10 years old car:

```
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
  return slope * x + intercept
speed = myfunc(10)
print(speed)
```

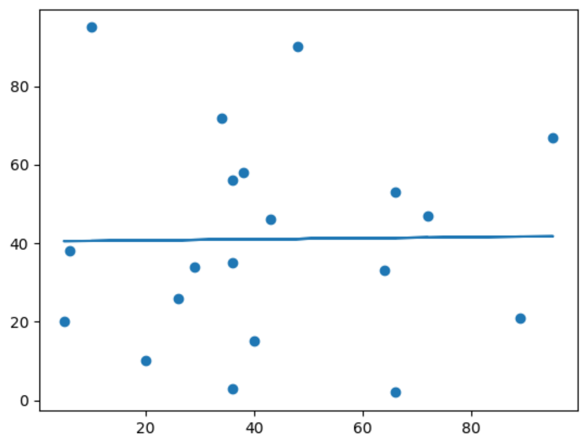
The example predicted a speed at 85.6, which we also could read from the diagram:



Bad Fit?
Let us create an example where linear regression would not be the best method to predict future values.
Example
These values for the x- and y-axis should result in a very bad fit for linear regression:

```
import matplotlib.pyplot as plt
from scipy import stats
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Result:



And the r for relationship?

Example

You should get a very low r value.

```
import numpy
from scipy import stats
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)
```

9.3. Logistic Regression

Logistic regression aims to solve classification problems. It does this by predicting categorical outcomes, unlike linear regression that predicts a continuous outcome.

In the simplest case there are two outcomes, which is called binomial, an example of which is predicting if a tumor is malignant or benign. Other cases have more than two outcomes to classify, in this case it is called multinomial. A common example for multinomial logistic regression would be predicting the class of an iris flower between 3 different species.

Here we will be using basic logistic regression to predict a binomial variable. This means it has only two possible outcomes.

How does it work?

In Python we have modules that will do the work for us. Start by importing the NumPy module.

```
import numpy
```

Store the independent variables in X.

Store the dependent variable in y.

Below is a sample dataset:

```
#X represents the size of a tumor in centimeters.
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-
1,1)
```

```
#Note: X has to be reshaped into a column from a row for the LogisticRegression() function to
work.
#y represents whether or not the tumor is cancerous (0 for "No", 1 for "Yes").
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

We will use a method from the sklearn module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the LogisticRegression() method to create a logistic regression object.

This object has a method called fit() that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
logr = linear_model.LogisticRegression()
logr.fit(X,y)
```

Now we have a logistic regression object that is ready to whether a tumor is cancerous based on the tumor size:

```
#predict if tumor is cancerous where the size is 3.46mm:
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
```

Result

```
[0]
```

We have predicted that a tumor with a size of 3.46mm will not be cancerous.

Coefficient

In logistic regression the coefficient is the expected change in log-odds of having the outcome per unit change in X.

This does not have the most intuitive understanding so let's use it to create something that makes more sense, odds.

Example

See the whole example in action:

```
import numpy
from sklearn import linear_model
#Reshaped for Logistic function.
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-
1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
logr = linear_model.LogisticRegression()
logr.fit(X,y)
log_odds = logr.coef_
odds = numpy.exp(log_odds)
print(odds)
```

Result

```
[4.03541657]
```

This tells us that as the size of a tumor increases by 1mm the odds of it being a tumor increases by 4x.

Probability

The coefficient and intercept values can be used to find the probability that each tumor is cancerous. Create a function that uses the model's coefficient and intercept values to return a new value. This new value represents probability that the given observation is a tumor:

```
def logit2prob(logr,x):
  log_odds = logr.coef_ * x + logr.intercept_
  odds = numpy.exp(log_odds)
  probability = odds / (1 + odds)
  return(probability)
```

Function Explained

To find the log-odds for each observation, we must first create a formula that looks similar to the one from linear regression, extracting the coefficient and the intercept.

```
log_odds = logr.coef_ * x + logr.intercept_
```

To then convert the log-odds to odds we must exponentiate the log-odds.

```
odds = numpy.exp(log_odds)
```

Now that we have the odds, we can convert it to probability by dividing it by 1 plus the odds.

```
probability = odds / (1 + odds)
```

Let us now use the function with what we have learned to find out the probability that each tumor is cancerous.

Example

See the whole example in action:

```
import numpy
from sklearn import linear_model
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
logr = linear_model.LogisticRegression()
logr.fit(X,y)
def logit2prob(logr, X):
  log_odds = logr.coef_ * X + logr.intercept_
  odds = numpy.exp(log_odds)
  probability = odds / (1 + odds)
  return(probability)
print(logit2prob(logr, X))
```

Result

```
[[0.60749955]
 [0.19268876]
 [0.12775886]
 [0.00955221]
 [0.08038616]
 [0.07345637]
 [0.88362743]
 [0.77901378]
 [0.88924409]
 [0.81293497]
 [0.57719129]
 [0.96664243]]
```

Results Explained

3.78 0.61 The probability that a tumor with the size 3.78cm is cancerous is 61%.
2.44 0.19 The probability that a tumor with the size 2.44cm is cancerous is 19%.
2.09 0.13 The probability that a tumor with the size 2.09cm is cancerous is 13%.

9.4. Correlation

Correlation is the statistical analysis of the relationship or dependency between two variables. Correlation allows us to study both the strength and direction of the relationship between two sets of variables.

A great aspect of the Pandas module is the corr() method.
The corr() method calculates the relationship between each column in your data set.
To show the relationship between the columns:

```
df.corr()
```

Result

```
           Duration     Pulse  Maxpulse  Calories
Duration   1.000000 -0.155408  0.009403  0.922721
Pulse     -0.155408  1.000000  0.786535  0.025120
Maxpulse   0.009403  0.786535  1.000000  0.203814
Calories   0.922721  0.025120  0.203814  1.000000
```

Note: The corr() method ignores "not numeric" columns.

Result Explained

The Result of the corr() method is a table with a lot of numbers that represents how well the relationship is between two columns.
The number varies from -1 to 1.
1 means that there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went up as well.
0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.
-0.9 would be just as good relationship as 0.9, but if you increase one value, the other will probably go down.
0.2 means NOT a good relationship, meaning that if one value goes up does not mean that the other will.
What is a good correlation? It depends on the use, but I think it is safe to say you have to have at least 0.6 (or -0.6) to call it a good correlation.

Perfect Correlation:
We can see that "Duration" and "Duration" got the number 1.000000, which makes sense, each column always has a perfect relationship with itself.

Good Correlation:
"Duration" and "Calories" got a 0.922721 correlation, which is a very good correlation, and we can predict that the longer you work out, the more calories you burn, and the other way around: if you burned a lot of calories, you probably had a long work out.

Bad Correlation:
"Duration" and "Maxpulse" got a 0.009403 correlation, which is a very bad correlation, meaning that we cannot predict the max pulse by just looking at the duration of the work out, and vice versa.

Lesson 9: Review Questions

1. Differentiate between linear and logistic regression. Use diagrams for illustration.
2. Discuss circumstances where to use linear regression instead of logistic and vice versa.
3. Using an online dataset, draw a relationship using regression between height and weight of a human being. Is it linear or logistic?
4. Explain various types of correlation.