More Hypothesis Testing with SciPy and Stats Models

Lesson Goals

In this lesson we will learn more about tests that we can perform with SciPy. These tests allow us to make decisions based on data and compare information in two or more variables.

Introduction

The field of statistics helps us make decisions using data. In previous lessons, we have looked at the comparison of one sample to a constant or the comparison of two samples to each other. In this lesson, we will use statistical tools to examine a number of features at once. We will also learn about linear regression using SciPy.

ANOVA and the F-Test

ANOVA (or ANalysis Of VAriance) is a technique meant to compare the means of three or more independent samples. An example of when we might use ANOVA is when conducting a test on an e-commerce website and trying out multiple UI designs at once to see if there is a change in sales.

The hypothesis test that we are examining is:

$$H_0: \mu_1 = \mu_2 = \cdots = \mu_k$$
$$H_1: At\ least\ one\ mean\ is\ different$$

Where μ represents a mean and there are a total of k means that we are comparing.

Typically, the ANOVA is a table consisting of values that help us compute a p-value for our hypothesis. The p-value will be found by performing the F-test. The F-test is a test for comparing variances.

Let’s look at an example. Let’s say we have the following data:

	Group 1	**Group 2**	**Group 3**
Number of Samples	n_1	n_2	n_3
Sample Mean	$\bar{X}_1$	$\bar{X}_2$	$\bar{X}_3$
Sample Standard Deviation	s_1	s_2	s_3

We would like to compare these three samples and see whether there is a significant difference in at least one of them.

With the ANOVA, we compare the difference in variation between the groups and the difference in variation within the groups themselves. If the F statistic is sufficiently large, this means the p-value will be sufficiently small. This will lead us to reject the null hypothesis and conclude that there is significant variation between the groups and therefore at least one of the means is different.

This is how we would construct an ANOVA:

Source of Variation	Sums of Squares (SS)	Degrees of Freedom (df)	Mean Squares (MS)	F
Between Treatments	$\mathrm{SSB} = \Sigma n_j(\bar{X}_j - \bar{X})^2$	k-1	$\mathrm{MSB} = \frac{SSB}{k-1}$	$F = \frac{\mathrm{MSB}}{\mathrm{MSE}}$
Error (or Residual)	$\mathrm{SSE} = \Sigma\Sigma(X - \bar{X}_j)^2$	N-k	$\mathrm{MSE} = \frac{\mathrm{MSE}}{N-k}$	
Total	$\mathrm{SST} = \Sigma\Sigma(X - \bar{X})^2$	N-1		

ANOVA in Python

Using SciPy

There are a number of ways to perform the ANOVA F-test in Python. The first way is using SciPy. We can pass all groups to the f_oneway function. This function returns the result of the hypothesis test. Below is an example of a dataset containing 8 observations of car loan interest rates from 6 different cities. We would like to show that there is a difference in the rates based on city. The dataset rate_by_city.csv can be obtained here.

```
import pandas as pd
from scipy.stats import f_oneway


#let's load the dataset
rate = pd.read_csv('rate_by_city.csv')
rate.head(15)
```

	Rate	City
0	13.75	1
1	13.75	1
2	13.50	1
3	13.50	1
4	13.00	1
5	13.00	1
6	13.00	1
7	12.75	1
8	12.50	1
9	14.25	2
10	13.00	2
11	12.75	2
12	12.50	2
13	12.50	2
14	12.40	2

The dataset contains two columns - rate and city. To test our hypothesis, we need to either pass in multiple filtered subsets to our function or to pivot the dataset to have one column per city. We'll choose the second option. We'll start off by using the cumcount function to create a new index and then use the pivot function to create 6 city columns. We will then rename the columns to allow us to access them more easily.

```
rate['city_count'] = rate.groupby('City').cumcount()
```

```
rate_pivot = rate.pivot(index='city_count', columns='City', values='Rate')
rate_pivot.columns = ['City_'+str(x) for x in rate_pivot.columns.values]
rate_pivot.head()
```

	City_1	City_2	City_3	City_4	City_5	City_6
city_count						
0	13.75	14.25	14.00	15.00	14.50	13.50
1	13.75	13.00	14.00	14.00	14.00	12.25
2	13.50	12.75	13.51	13.75	14.00	12.25
3	13.50	12.50	13.50	13.59	13.90	12.00
4	13.00	12.50	13.50	13.25	13.75	12.00

Now that we have successfully pivoted the data, we can perform the test. The `f_oneway`function requires us to specify each column that is passed into the function (rather than passing the entire dataframe)

```
f_oneway(rate_pivot.City_1,rate_pivot.City_2,rate_pivot.City_3,rate_pivot.City_4,rate_pivot.City_5,rate_pivot.City_6)
F_onewayResult(statistic=4.8293848737024, pvalue=0.001174551414504048)
```

The p-value is 0.001174. This value is very small, certainly smaller than 0.05. Therefore, we reject the null hypothesis and conclude that the rates differ by city.

Using statsmodels

`statsmodels` is a Python library aimed specifically at performing statistical tests and hypothesis testing. The output from this library tends to be more detailed.

The function for generating an ANOVA in `statsmodels` is called `anova_lm` and it generates an ANOVA table. As a first step, we define a model and then generate the ANOVA table. In this case, we prefer not to pivot our data since the library will do it for us.

```
import statsmodels.api as sm
from statsmodels.formula.api import ols


model = ols('Rate ~ C(City)', data=rate).fit()
anova_table = sm.stats.anova_lm(model, typ=2)
anova_table
```

	sum_sq	df	F	PR(>F)
C(City)	10.945667	5.0	4.829385	0.001175
Residual	21.758133	48.0	NaN	NaN

In the code below, we defined a model of rate and city. The pivoting is performed internally by using the `C` function. Our result is the same p-value and our conclusion to reject remains the same.

Linear Regression

As we have previously seen, linear regression is a technique for modelling the relationship between one or more predictor (or independent) variables and one or more response (or dependent) variables. Our goal using linear regression is to explain the relationship using a linear equation of the form:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

Using linear regression means having a simple and interpretable model at the cost of losing granular information and potentially oversimplifying and increasing our error.

Linear Regression in Python

Linear Regression using SciPy

There are many ways to perform regression in Python and this lesson will discuss both SciPy and statsmodels. We perform linear regression in SciPy using the `linregress` function. This function returns the slope, the intercept, the r-value (which we will square to find r squared), the p-value (this test checks whether the slope is significantly different from zero), and the standard error of the estimated gradient.
In the example below, we will create a linear model that predicts MPG using acceleration in the `auto-mpg` dataset. Note that `linregress` only supports linear regression with one variable for x and one for y.
The dataset `auto-mpg.csv` can be obtained here.

```
from scipy.stats import linregress


auto = pd.read_csv('auto-mpg.csv')
auto.head()
```

	mpg	cylinders	displacement	horse_power	weight	acceleration	model_year	car_name
0	18.0	8	307.0	130.0	3504	12.0	70	\t"chevrolet chevelle malibu"
1	15.0	8	350.0	165.0	3693	11.5	70	\t"buick skylark 320"
2	18.0	8	318.0	150.0	3436	11.0	70	\t"plymouth satellite"
3	16.0	8	304.0	150.0	3433	12.0	70	\t"amc rebel sst"
4	17.0	8	302.0	140.0	3449	10.5	70	\t"ford torino"

```
slope, intercept, r_value, p_value, std_err = linregress(auto.acceleration, auto.mpg)
slope, intercept, r_value, p_value, std_err
(1.1912045293502271,
 4.969793004253912,
 0.17664276963558906,
 1.8230915350787203e-18,
 0.12923643283101396)
```

This means that our regression equation is:

mpg = 4.9698 + 1.1912 * acceleration

The r squared is 0.1766 which is relatively small. This means that our model only captures 17% of the variation in the data.

The p-value is very small, this means that the slope is significantly different from zero.

Linear Regression using statsmodels

Unlike SciPy, the output we get with `statsmodels` is more detailed. Below, we will repeat the same example but using `statsmodels`.

```
import statsmodels.api as sm


X = sm.add_constant(auto.acceleration) # We must add the intercept using the add_constant function
Y = auto.mpg


model = sm.OLS(Y, X).fit()
predictions = model.predict(X)
```

```
print_model = model.summary()
print(print_model)

                            OLS Regression Results
==============================================================================
=====
Dep. Variable:                    mpg   R-squared:                       0.177
Model:                            OLS   Adj. R-squared:                  0.175
Method:                 Least Squares   F-statistic:                     84.96
Date:                Thu, 31 Jan 2019   Prob (F-statistic):           1.82e-18
Time:                        13:29:47   Log-Likelihood:                -1343.9
No. Observations:                 398   AIC:                             2692.
Df Residuals:                     396   BIC:                             2700.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=======
                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const            4.9698      2.043      2.432      0.015       0.953       8.987
acceleration     1.1912      0.129      9.217      0.000       0.937       1.445
==============================================================================
=====
Omnibus:                       17.459   Durbin-Watson:                   0.677
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               18.214
Skew:                           0.497   Prob(JB):                     0.000111
Kurtosis:                       2.670   Cond. No.                         91.1
==============================================================================
=====
```

Here, we are not limited to only one predictor variable. Let’s try this regression with more than one predictor.

```
X = sm.add_constant(auto[['cylinders', 'weight', 'acceleration']]) # adding a constant
Y = auto.mpg

model = sm.OLS(Y, X).fit()
predictions = model.predict(X)

print_model = model.summary()
print(print_model)
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:                    mpg   R-squared:                       0.700
Model:                            OLS   Adj. R-squared:                  0.698
Method:                 Least Squares   F-statistic:                     306.7
Date:                Thu, 31 Jan 2019   Prob (F-statistic):          1.14e-102
Time:                        13:33:47   Log-Likelihood:                -1142.9
No. Observations:                 398   AIC:                             2294.
Df Residuals:                     394   BIC:                             2310.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
=======
```

```
                              coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const          42.3811      1.960     21.627      0.000      38.528      46.234
cylinders      -0.4827      0.302     -1.599      0.111      -1.076       0.111
weight         -0.0065      0.001    -11.342      0.000      -0.008      -0.005
acceleration    0.2034      0.091      2.236      0.026       0.025       0.382
==============================================================================
Omnibus:                       34.469   Durbin-Watson:                   0.816
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               45.516
Skew:                           0.654   Prob(JB):                     1.31e-10
Kurtosis:                       4.016   Cond. No.                     2.82e+04
==============================================================================
```

Conclusion

In this lesson we learned how to create and evaluate an ANOVA table in both SciPy and `statsmodels`. We learned the proper use of an F test and what hypothesis is tested using this test. We also looked at linear regression in both SciPy and `statsmodels`. We were able to compare the more succinct output from SciPy with the detailed tables in `statsmodels`. Both outputs serve a different purpose and have value in different scenarios. Hopefully, this lesson will empower you to use your statistics chops to make business decisions.