Bentley University MA252 in Python

Content extracted from the How to Data Website

This PDF generated on 07 December 2021

Contents

MA252 is an undergraduate statistics course at Bentley University that focuses on model building using regression. The description from the course catalog can be found here.

It covers simple linear regression, multivariate linear regression, logistic linear regression, model building, transformations, and interactions.

Content last modified on 07 December 2021.

How to fit a linear model to two columns of data

Description

Let's say we have two columns of data, one for a single independent variable x and the other for a single dependent variable y. How can I find the best fit linear model that predicts y based on x?

In other words, what are the model coefficients β_0 and β_1 that give me the best linear model $\hat{y} = \beta_0 + \beta_1 x$ based on my data?

Related tasks:

- How to compute R-squared for a simple linear model
- How to fit a multiple linear regression model
- How to predict the response variable in a linear model

Solution in Python

This solution uses a pandas DataFrame of fake example data. When using this code, replace our fake data with your real data.

Although the solution below uses plain Python lists of data, it also works if the data are stored in NumPy arrays or pandas Series.

```python
# Here is the fake data you should replace with your real data.
xs = [ 393, 453, 553, 679, 729, 748, 817 ]
ys = [  24,  25,  27,  36,  55,  68,  84 ]

# We will use SciPy to build the model
import scipy.stats as stats

# If you need the model coefficients stored in variables for later use, do:
model = stats.linregress( xs, ys )
beta0 = model.intercept
beta1 = model.slope

# If you just need to see the coefficients (and some other related data),
# do this alone:
stats.linregress( xs, ys )
```

```
LinregressResult(slope=0.1327195637885226, intercept=-37.32141898334582, rvalue=0.8949574425541466, pvalue=0.006
```

The linear model in this example is approximately $y = 0.133x - 37.32$.

Content last modified on 28 May 2021.

See a problem? Tell us or edit the source.

How to compute a confidence interval for the expected value of a response variable

Description

If we have a simple linear regression model, $y = \beta_0 + \beta_1 x + \epsilon$, where ϵ is some random error, then given any x input, y can be veiwed as a random variable because of ϵ. Let's consider its expected value. How do we construct a confidence interval for that expected value, given a value for the predictor x?

Related tasks:

- How to compute a confidence interval for a mean difference (matched pairs) (on website)
- How to compute a confidence interval for a regression coefficient
- How to compute a confidence interval for a population mean (on website)
- How to compute a confidence interval for a single population variance (on website)
- How to compute a confidence interval for the difference between two means when both population variances are known (on website)
- How to compute a confidence interval for the difference between two means when population variances are unknown (on website)
- How to compute a confidence interval for the difference between two proportions (on website)
- How to compute a confidence interval for the population proportion (on website)
- How to compute a confidence interval for the ratio of two population variances (on website)

Solution in Python

How to Data does not yet contain a solution for this task in Python.

How to compute R-squared for a simple linear model

Description

Let's say we have fit a linear model to two columns of data, one for a single independent variable x and the other for a single dependent variable y. How can we compute R^2 for that model, to measure its goodness of fit?

Related tasks:

- How to fit a linear model to two columns of data
- How to compute adjusted R-squared

Solution in Python

We assume you have already fit a linear model to the data, as in the code below, which is explained fully in a separate task, how to fit a linear model to two columns of data.

```
import scipy.stats as stats
xs = [ 393, 453, 553, 679, 729, 748, 817 ]
ys = [  24,  25,  27,  36,  55,  68,  84 ]
model = stats.linregress( xs, ys )
```

The R value is part of the model object that `stats.linregress` returns.

```
model.rvalue
```

```
0.8949574425541466
```

You can compute R^2 just by squaring it.

```
model.rvalue ** 2
```

```
0.8009488239830586
```

Content last modified on 01 June 2021.

See a problem? Tell us or edit the source.

How to predict the response variable in a linear model

Description

If we have a linear model and a value for each explanatory variable, how do we predict the corresponding value of the response variable?

Related tasks:

- How to fit a linear model to two columns of data
- How to fit a multiple linear regression model

Solution in Python

Let's assume that you've already built a linear model. We do an example below with fake data, but you can use your own actual data. For more information on the following code, see how to fit a multivariate linear model.

```
import pandas as pd
df = pd.DataFrame( {
    'x1' : [ 2,  7,  4,  3, 11, 18,   6, 15,   9,  12],
    'x2' : [ 4,  6, 10,  1, 18, 11,   8, 20,  16,  13],
    'x3' : [11, 16, 20,  6, 14,  8,   5, 23,  13,  10],
    'y'  : [24, 60, 32, 29, 90, 45, 130, 76, 100, 120]
} )

import statsmodels.api as sm
model = sm.OLS( df['y'], sm.add_constant( df[['x1','x2','x3']] ) ).fit()
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of panc
  x = pd.concat(x[::order], 1)
```

Let's say we want to estimate y given that $x_1 = 5$, $x_2 = 12$, and $x_3 = 50$. We can use the model's `predict()` function as shown below, but we must add an entry for the constant term in the model—we can use any value, but we choose 1.

```
model.predict( [ 1, 5, 12, 50 ] )
```

```
array([-91.71014402])
```

For the given values of the explanatory variables, our predicted response variable is -91.71014402.

Note that if you want to compute the predicted values for all the data on which the model was trained, simply call `model.predict()` with no arguments, and it defaults to using the training data.

```
model.predict()
```

```
array([ 47.5701159 ,  24.35988296,  42.21531274,  47.27613825,
       110.86526185,  70.03097584,  95.12689978,  70.91290879,
       106.52986696,  91.11263692])
```

Content last modified on 07 December 2021.

How to fit a multivariate linear model

Description

Let's say we have several independent variables, $x_1, x_2, \ldots, x_k$, and a dependent variable y. How can I fit a linear model that uses these independent variables to best predict the dependent variable?

In other words, what are the model coefficients $\beta_0, \beta_1, \beta_2, \ldots, \beta_k$ that give me the best linear model $\hat{y} = \beta_0 + \beta_1 x + \beta_2 x + \cdots + \beta_k x$ based on my data?

Related tasks:

- How to fit a linear model to two columns of data
- How to predict the response variable in a linear model

Solution in Python

We're going to use fake data here for illustrative purposes. You can replace our fake data with your real data in the code below.

We'll put the data into a dataframe and then make a variable with a list of the independent variables and a variable with the outcome variable.

```
import pandas as pd

# Replace this fake data with your real data
df = pd.DataFrame( {
    'x1':[2, 7, 4, 3, 11, 18, 6, 15, 9, 12],
    'x2':[4, 6, 10, 1, 18, 11, 8, 20, 16, 13],
    'x3':[11, 16, 20, 6, 14, 8, 5, 23, 13, 10],
    'y':[24, 60, 32, 29, 90, 45, 130, 76, 100, 120]
} )

xs = df[['x1', 'x2', 'x3']]  # list of independent variables
y = df['y']                  # dependent variable
```

We can use StatsModels' `OLS` to build our multivariate linear model. We'll print out the coefficients and the intercept, and the coefficients will be in the form of an array when we print them.

```
import statsmodels.api as sm

# Add a constant to the dependent variables first
xs = sm.add_constant(xs)

# Build the model
model = sm.OLS(y, xs).fit()

# Show the model summary to get the coefficients and the intercept
model.summary()
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of pand
  x = pd.concat(x[::order], 1)
/opt/conda/lib/python3.9/site-packages/scipy/stats/stats.py:1541: UserWarning: kurtosistest only valid for n>=20
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

OLS Regression Results

Dep. Variable:

```
      <td>y</td>        <th>  R-squared:         </th> <td>   0.594</td>
```

Model:

```
             <td>OLS</td>       <th>  Adj. R-squared:    </th> <td>   0.390</td>
```

Method:

```
       <td>Least Squares</td>  <th>  F-statistic:       </th> <td>   2.921</td>
```

Date:

```
       <td>Tue, 07 Dec 2021</td> <th>  Prob (F-statistic):</th>  <td> 0.122</td>
```

Time:

```
           <td>15:07:00</td>     <th>  Log-Likelihood:    </th> <td> -45.689</td>
```

No. Observations:

```
  <td>    10</td>      <th>  AIC:               </th> <td>   99.38</td>
```

Df Residuals:

```
      <td>     6</td>      <th>  BIC:               </th> <td>   100.6</td>
```

Df Model:

```
          <td>     3</td>      <th>                     </th>     <td> </td>
```

Covariance Type:

```
  <td>nonrobust</td>    <th>                     </th>     <td> </td>
```

coef

std err

t

P>|t|

[0.025]

const

77.2443

27.366

2.823

0.030

10.282

144.206

x1

-2.7009

2.855

-0.946

0.381

-9.686

4.284

x2

7.2989

2.875

2.539

0.044

0.265

14.333

x3

-4.8607

2.187

-2.223

0.068

-10.211

0.490

Omnibus:

```
  <td> 2.691</td> <th>  Durbin-Watson:     </th> <td>   2.123</td>
```

Prob(Omnibus):

0.260

Jarque-Bera (JB):

1.251

Skew:

```
     <td> 0.524</td> <th>  Prob(JB):          </th> <td>   0.535</td>
```

Kurtosis:

```
<td> 1.620</td> <th>  Cond. No.          </th> <td>    58.2</td>
```

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

The coefficients and intercept appear on the left hand side of the output, about half way down, under the heading "coef."

Thus the multivariate linear model from the example data is $\hat{y} = 77.2443 - 2.7009x_1 + 7.2989x_2 - 4.8607x_3$.

Content last modified on 07 December 2021.

See a problem? Tell us or edit the source.

How to compute a confidence interval for a regression coefficient

Description

Say we have a linear regression model, either single variable or multivariate. How do we compute a confidence interval for the coefficient of one of the explanatory variables in the model?

Related tasks:

- How to compute a confidence interval for a mean difference (matched pairs) (on website)
- How to compute a confidence interval for a population mean (on website)
- How to compute a confidence interval for a single population variance (on website)
- How to compute a confidence interval for the difference between two means when both population variances are known (on website)
- How to compute a confidence interval for the difference between two means when population variances are unknown (on website)
- How to compute a confidence interval for the difference between two proportions (on website)
- How to compute a confidence interval for the expected value of a response variable
- How to compute a confidence interval for the population proportion (on website)
- How to compute a confidence interval for the ratio of two population variances (on website)

Solution in Python

How to Data does not yet contain a solution for this task in Python.

How to compute adjusted R-squared

Description

If we have fit a multivariate linear model, how can we compute the Adjusted R^2 for that model, to measure its goodness of fit?

Related tasks:

- How to compute R-squared for a simple linear model

Solution in Python

We assume you have already fit a multivariate linear model to some data, as in the code below. (If you're unfamiliar with how to do so, see how to fit a multivariate linear model.) The data shown below is fake, and we assume you will replace it with your own real data if you use this code.

```
import pandas as pd
import statsmodels.api as sm
df = pd.DataFrame( {
    'x1':[2, 7, 4, 3, 11, 18, 6, 15, 9, 12],
    'x2':[4, 6, 10, 1, 18, 11, 8, 20, 16, 13],
    'x3':[11, 16, 20, 6, 14, 8, 5, 23, 13, 10],
    'y':[24, 60, 32, 29, 90, 45, 130, 76, 100, 120]
} )
xs = df[['x1', 'x2', 'x3']]
y = df['y']
xs = sm.add_constant(xs)
model = sm.OLS(y, xs).fit()
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of panc
  x = pd.concat(x[::order], 1)
```

You can get a lot of information about your model from its summary.

```
model.summary()
```

```
/opt/conda/lib/python3.9/site-packages/scipy/stats/stats.py:1541: UserWarning: kurtosistest only valid for n>=20
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

OLS Regression Results

Dep. Variable:

```
      <td>y</td>        <th>  R-squared:         </th> <td>   0.594</td>
```

Model:

```
             <td>OLS</td>       <th>  Adj. R-squared:    </th> <td>   0.390</td>
```

Method:

```
        <td>Least Squares</td>  <th>  F-statistic:       </th> <td>   2.921</td>
```

Date:

```
        <td>Tue, 07 Dec 2021</td> <th>  Prob (F-statistic):</th>  <td> 0.122</td>
```

Time:

```
            <td>15:06:52</td>     <th>  Log-Likelihood:    </th> <td> -45.689</td>
```

No. Observations:

```
  <td>    10</td>      <th>  AIC:               </th> <td>   99.38</td>
```

Df Residuals:

```
      <td>     6</td>      <th>  BIC:               </th> <td>   100.6</td>
```

Df Model:

```
          <td>     3</td>      <th>                     </th>     <td> </td>
```

Covariance Type:

```
  <td>nonrobust</td>    <th>                     </th>     <td> </td>
```

coef

std err

t

P>|t|

[0.025]

const

77.2443

27.366

2.823

0.030

10.282

144.206

x1

-2.7009

2.855

-0.946

0.381

-9.686

4.284

x2

7.2989

2.875

2.539

0.044

0.265

14.333

x3

-4.8607

2.187

-2.223

0.068

-10.211

0.490

Omnibus:

```
    <td> 2.691</td> <th>  Durbin-Watson:     </th> <td>   2.123</td>
```

Prob(Omnibus):

0.260

Jarque-Bera (JB):

1.251

Skew:

```
       <td> 0.524</td> <th>  Prob(JB):          </th> <td>   0.535</td>
```

Kurtosis:

```
  <td> 1.620</td> <th>  Cond. No.          </th> <td>    58.2</td>
```

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

In particular, that printout contains the Adjusted R^2 value; it is the second value in the right-hand column, near the top.

You can also obtain it directly, as follows:

```
model.rsquared_adj
```

```
0.390392407508503
```

In this case, the Adjusted R^2 is 0.3904.

How to do a test of joint significance

Description

If we have a multivariate linear model, how do we test the joint significance of all the variables in the model? In other words, how do we test the overall significance of the regression model?

Solution in Python

Let's assume that you already made your multivariate linear model, similar to the one shown below. If you still need to create one, first see how to fit a multiple linear regression model (on website).

We use example data here, but you would use your own data instead.

```
import pandas as pd
import statsmodels.api as sm
data = {
    'x1' : [ 2,  7,  4,  3, 11, 18,   6, 15,   9,  12],
    'x2' : [ 4,  6, 10,  1, 18, 11,   8, 20,  16,  13],
    'x3' : [11, 16, 20,  6, 14,  8,   5, 23,  13,  10],
    'y' :  [24, 60, 32, 29, 90, 45, 130, 76, 100, 120]
}
```

The following code fits the model to the data.

```
df = pd.DataFrame(data)
xs = df[['x1', 'x2', 'x3']]
y = df['y']
xs = sm.add_constant(xs)
model = sm.OLS(y, xs).fit()
```

```
/opt/conda/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142: FutureWarning: In a future version of pan
  x = pd.concat(x[::order], 1)
```

Now we want to test whether the model is significant. We will use a null hypothesis that states that all of the model's coefficients are equal to zero, that is, they are not jointly significant in predicting y. We can write $H_0 : \beta_0 = \beta_1 = \beta 2 = \beta_3 = 0$.

We also choose a value $0 \leq \alpha \leq 1$ as our Type 1 error rate. Herer we'll use $\alpha = 0.05$.

The summary output for the model will give us both the F-statistic and the p-value.

```
model.summary()
```

```
/opt/conda/lib/python3.9/site-packages/scipy/stats/stats.py:1541: UserWarning: kurtosistest only valid for n>=20
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

OLS Regression Results

Dep. Variable:

```
      <td>y</td>        <th>  R-squared:         </th> <td>   0.594</td>
```

Model:

```
              <td>OLS</td>       <th>  Adj. R-squared:    </th> <td>   0.390</td>
```

Method:

```
        <td>Least Squares</td>  <th>  F-statistic:       </th> <td>   2.921</td>
```

Date:

```
        <td>Tue, 05 Oct 2021</td> <th>  Prob (F-statistic):</th>  <td> 0.122</td>
```

Time:

```
            <td>20:20:04</td>     <th>  Log-Likelihood:    </th> <td> -45.689</td>
```

No. Observations:

```
  <td>    10</td>      <th>  AIC:               </th> <td>   99.38</td>
```

Df Residuals:

```
      <td>     6</td>      <th>  BIC:               </th> <td>   100.6</td>
```

Df Model:

```
          <td>     3</td>      <th>                     </th>     <td> </td>
```

Covariance Type:

```
  <td>nonrobust</td>    <th>                     </th>     <td> </td>
```

coef

std err

t

P>|t|

[0.025]

const

77.2443

27.366

2.823

0.030

10.282

144.206

x1

-2.7009

2.855

-0.946

0.381

-9.686

4.284

x2

7.2989

2.875

2.539

0.044

0.265

14.333

x3

-4.8607

2.187

-2.223

0.068

-10.211

0.490

Omnibus:

```
   <td> 2.691</td> <th>  Durbin-Watson:     </th> <td>   2.123</td>
```

Prob(Omnibus):

0.260

Jarque-Bera (JB):

1.251

Skew:

```
      <td> 0.524</td> <th>  Prob(JB):          </th> <td>   0.535</td>
```

Kurtosis:

```
  <td> 1.620</td> <th>  Cond. No.          </th> <td>    58.2</td>
```

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

Near the top right of the output, we can see that the F-statistic is 2.921. The corresponding p-value immediately below it is 0.1222, which is greater than α, so we do not have sufficient evidence to reject the null hypothesis.

We cannot conclude that the independent variables in our model are jointly significant in predicting the response variable.

Content last modified on 05 October 2021.

How to compute covariance and correlation coefficients

Description

Covariance is a measure of how much two variables "change together." It is positive when the variables tend to increase or decrease together, and negative when they upward motion of one variable is correlated with downward motion of the other. Correlation normalizes covariance to the interval $[-1, 1]$.

Solution in Python

We will construct some random data here, but when applying this, you would use your own data, of course.

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.rand(10,5))
df.columns = [ 'col1','col2','col3','col4','col5' ]
df.head()
```

	col1	col2	col3	col4	col5
0	0.263520	0.803570	0.514936	0.925995	0.678391
1	0.554981	0.444687	0.742719	0.882195	0.719853
2	0.272787	0.527744	0.299307	0.324882	0.446657
3	0.421805	0.000464	0.769808	0.318595	0.383792
4	0.005127	0.031319	0.233333	0.288079	0.273122

If you have two pandas Series, you can compute the covariance of just those two variables. Note that every column in a DataFrame is a pandas series.

```
np.cov( df['col1'], df['col2'] )
```

```
array([[ 0.03876363, -0.01118424],
       [-0.01118424,  0.10323356]])
```

You can also compare all of a DataFrame's columns among one another, each as a separate variable.

```
df.cov()
```

	col1	col2	col3	col4	col5
col1	0.038764	-0.011184	0.036492	0.010443	0.000424
col2	-0.011184	0.103234	-0.017756	0.023502	0.031411
col3	0.036492	-0.017756	0.097690	0.005633	-0.010821
col4	0.010443	0.023502	0.005633	0.073091	0.056259
col5	0.000424	0.031411	-0.010821	0.056259	0.063866

The Pearson correlation coefficient can be computed with `np.corrcoef` in place of `np.cov`.

```
np.corrcoef( df['col1'], df['col2'] )
```

```
array([[ 1.        , -0.1768007],
       [-0.1768007,  1.        ]])
```

And pandas DataFrames have a built in method to do this for all numeric columns.

```
df.corr()
```

	col1	col2	col3	col4	col5
col1	1.000000	-0.176801	0.593003	0.196192	0.008517
col2	-0.176801	1.000000	-0.176814	0.270565	0.386852
col3	0.593003	-0.176814	1.000000	0.066660	-0.136993
col4	0.196192	0.270565	0.066660	1.000000	0.823433
col5	0.008517	0.386852	-0.136993	0.823433	1.000000

Content last modified on 23 July 2021.

See a problem? Tell us or edit the source.

How to compute the standard error of the estimate for a model

Description

One measure of the goodness of fit of a model is the standard error of its estimates. If the actual values are y_i and the estimates are $\hat{y}_i$, the definition of this quantity is as follows, for n data points.

$$\sigma_{\text{est}} = \sqrt{\frac{\sum (y_i - \hat{y}_i)^2}{n}}$$

If we've fit a linear model, how do we compute the standard error of its estimates?

Solution in Python

How to Data does not yet contain a solution for this task in Python.

How to do a hypothesis test of a coefficient's significance

Description

Let's say we have a linear model, either one variable or many. How do we conduct a test of significance for the coefficient of a single explanatory variable in the model? Similarly, how can we determine if an explanatory variable has a significant impact on the response variable?

Related tasks:

- How to compute a confidence interval for the difference between two proportions (on website)
- How to do a hypothesis test for a mean difference (matched pairs) (on website)
- How to do a hypothesis test for a population proportion (on website)
- How to do a hypothesis test for population variance (on website)
- How to do a hypothesis test for the difference between means when both population variances are known (on website)
- How to do a hypothesis test for the difference between two proportions (on website)
- How to do a hypothesis test for the mean with known standard deviation (on website)
- How to do a hypothesis test for the ratio of two population variances (on website)
- How to do a one-sided hypothesis test for two sample means (on website)
- How to do a two-sided hypothesis test for a sample mean (on website)
- How to do a two-sided hypothesis test for two sample means (on website)

Solution in Python

How to Data does not yet contain a solution for this task in Python.

How to do a Spearman rank correlation test

Description

When we want to determine whether there is a relationship between two variables, but our samples do not come from normally distributed populations, we can use the Spearman Rank Correlation Test. How do we conduct it?

Solution in Python

We will use some fake data about height and weight measurements for people. You can replace it with your real data.

Our data should be NumPy arrays, as in the example below. (Recall that pandas DataFrame columns are also NumPy arrays.)

```
import numpy as np
heights = np.array([60, 76, 57, 68, 70, 62, 63])
weights = np.array([145, 178, 120, 143, 174, 130, 137])
```

Let's say we want to test the correlation between height (inches) and weight (pounds). Our null hypothesis would state that the Pearson correlation coefficient is equal to zero, or that there is no relationship between height and weight, $H_0 : \rho_s = 0$. We choose α, or the Type I error rate, to be 0.05 and carry out the Spearman Rank Correlation Test to get the test-statistic and p-value.

```
from scipy import stats
from scipy.stats import spearmanr
spearmanr(heights, weights)
```

```
SpearmanrResult(correlation=0.7857142857142859, pvalue=0.03623846267982713)
```

Our p-value is 0.03624, which is less than $\alpha = 0.05$, so we reject the null hypothesis. There does appear to be a relationship between height and weight.

(This p-value is different than the one computed in the solution using R, because different approximation methods are used by the two software packages when the sample size is small.)

Note that for right- or left-tailed tests, the following syntax can be used.

```
spearmanr(heights, weights, alternative="greater")  # right-tailed
spearmanr(heights, weights, alternative="less")     # left-talied
```

Content last modified on 05 October 2021.

See a problem? Tell us or edit the source.

How to compute the residuals of a linear model

Description

If a model has been fit to a dataset, the *residuals* are the differences between the actual data points and the results the model would predict. Given a linear model and a dataset, how can we compute those residuals?

Solution in Python

How to Data does not yet contain a solution for this task in Python.