Bentley University MA252 in Python

Content extracted from the How to Data Website

This PDF generated on 03 August 2023

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.

Simple Linear Regression

- How to fit a linear model to two columns of data
- How to compute a confidence interval for the expected value of a response variable
- How to compute R-squared for a simple linear model
- How to predict the response variable in a linear model

Multivariate Linear Regression

- How to fit a multivariate linear model
- How to add an interaction term to a model
- How to add a polynomial term to a model
- How to add a transformed term to a model
- How to compute a confidence interval for a regression coefficient
- How to compute adjusted R-squared

Model Building

- How to compute covariance and correlation coefficients
- How to compute the standard error of the estimate for a model
- How to do a hypothesis test of a coefficient's significance
- How to do a test of joint significance
- How to do a Spearman rank correlation test

Residual Analysis

- How to compute the residuals of a linear model

Content last modified on 03 August 2023.

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 multivariate linear model
- How to predict the response variable in a linear model

Solution in Python using SciPy

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.

```
# 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 $\hat{y} = 0.133x - 37.32$.

Content last modified on 24 July 2023.

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 using statsmodels and sklearn

Let's assume that you already have a linear model. We construct an example one here from some fabricated data. For a review of how this preparatory code works, see how to fit a linear model to two columns of data.

```
import statsmodels.api as sm

# Replace the following fake data with your actual data:
xs = [  34,   9,  78,  60,  22,  45,  83,  59,  25 ]
ys = [ 126, 347, 298, 309, 450, 187, 266, 385, 400 ]

# Create and fit a linear model to the data:
xs = sm.add_constant( xs )
model = sm.OLS( ys, xs ).fit()
```

Ask the model to do a prediction of one particular input, in this example $x = 40$, with a 95% confidence interval included ($\alpha = 0.05$). You can replce the 40 with your chosen x value, or an array of them, and you can replace the 0.05 with your chosen value of α.

(The extra `1` in the input to `get_prediction` is a placeholder, required because the model has been expanded to include a constant term.)

```
model.get_prediction( [1,40] ).summary_frame( alpha=0.05 )
```

	mean	mean_se	mean_ci_lower	mean_ci_upper	obs_ci_lower	obs_ci_upper
0	313.721744	36.823483	226.648043	400.795444	45.876725	581.566762

Our 95% confidence interval is $[226.648, 400.7954]$. We can be 95% confident that the true average value of y, given that x is 40, is between 226.648 and 400.7954.

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

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 using SciPy

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 24 July 2023.

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 multivariate linear model

Solution in Python using statsmodels

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()
```

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 24 July 2023.

See a problem? Tell us or edit the source.

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 using statsmodels

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/python3.10/site-packages/scipy/stats/_stats_py.py:1736: UserWarning: kurtosistest only valid for n
  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>Mon, 24 Jul 2023</td> <th>  Prob (F-statistic):</th>  <td> 0.122</td>
```

Time:

```
            <td>20:46:50</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 24 July 2023.

See a problem? Tell us or edit the source.

How to add an interaction term to a model

Description

Sometimes, a simple linear model isn't sufficient for our data, and we need more complex terms or transformed variables in the model to make adequate predictions. How do we include these complex and transformed terms in a regression model?

Related tasks:

- How to add a polynomial term to a model
- How to add a transformed term to a model

Solution in Python

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

How to add a polynomial term to a model

Description

Sometimes, a simple linear model isn't sufficient to describe the data. How can we include a higher-order term in a regression model, such as the square or cube of one of the predictors?

Related tasks:

- How to add a transformed term to a model
- How to add an interaction term to a model

Solution in Python using sklearn

We begin with a fabricated dataset of 20 points. You can replace the code below with your own, real, data.

```
import numpy as np
import pandas as pd

x = np.arange(0,20)                                                    # List of integers from 0 to 19
y = [3,4,5,7,9,20,31,50,70,75,80,91,101,120,135,160,179,181,190,193] # List of 20 integers
```

We extend our dataset with a new column (or "feature"), containing x^2.

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures( degree=2, include_bias=False )
x_matrix = x.reshape( -1, 1 )                    # make x a matrix so that we can add columns
poly_features = poly.fit_transform( x_matrix )  # add a second column, so we now have x and x^2
```

Next, fit a regression model to the new features, which are x and x^2.

```
from sklearn.linear_model import LinearRegression
poly_reg_model = LinearRegression()     # Our model will be linear in the features x and x^2
poly_reg_model.fit( poly_features, y )  # Use regression to create the model
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

LinearRegression

Finally, get the coefficients and intercept of the model.

```
poly_reg_model.intercept_, poly_reg_model.coef_
```

```
(-8.384415584415635, array([6.28628389, 0.27420825]))
```

Thus the equation for our model of degree two is $\hat{y} = -8.38 + 6.28x + 0.27x^2$

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

How to add a transformed term to a model

Description

Sometimes, a simple linear model isn't sufficient for our data, and we need more complex terms or transformed variables in the model to make adequate predictions. How do we include these complex and transformed terms in a regression model?

Related tasks:

- How to add a polynomial term to a model
- How to add an interaction term to a model

Solution in Python using NumPy and sklearn

We're going to create the `Pressure` dataset as example data. It contains observations of pressure and temperature. You would use your own data instead.

```
import pandas as pd
pressure = pd.DataFrame( {
    'temperature': [0,20,40,60,80,100,120,140,160,180,200,
                    220,240,260,280,300,320,340,360],
    'pressure':    [0.0002,0.0012,0.0060,0.0300,0.0900,0.2700,0.7500,
                    1.8500,4.2000,8.8000,17.3000,32.1000,57.0000,96.0000,
                    157.0000,247.0000,376.0000,558.0000,806.0000]
} )
pressure
```

	temperature	pressure
0	0	0.0002
1	20	0.0012
2	40	0.0060
3	60	0.0300
4	80	0.0900
5	100	0.2700
6	120	0.7500
7	140	1.8500
8	160	4.2000
9	180	8.8000
10	200	17.3000
11	220	32.1000
12	240	57.0000
13	260	96.0000
14	280	157.0000
15	300	247.0000
16	320	376.0000
17	340	558.0000
18	360	806.0000

Let's model temperature as the dependent variable with the logarithm of pressure as the independent variable. To transform the independent variable pressure, we use NumPy's `np.log` function, as shown below. It uses the natural logarithm (base e).

```
import numpy as np

# Compute the logarithm of pressure
X = pressure[['pressure']]
log_X = np.log(X)

# Build the linear model using Scikit-Learn
from sklearn.linear_model import LinearRegression
y = pressure['temperature']
log_model = LinearRegression()
log_model.fit(log_X, y)

# Display regression coefficients and R-squared value of the model
log_model.intercept_, log_model.coef_, log_model.score(log_X, y)
```

```
(153.97045660511063, array([23.78440995]), 0.9464264282083346)
```

The model is $\hat{t} = 153.97 + 23.784 \log p$, where t stands for temperature and p for pressure.

Another example transformation is the square root transformation. As with `np.log`, just apply the `np.sqrt` function to the appropriate term when defining the model.

```
# Compute the square root of pressure
X = pressure[['pressure']]
sqrt_X = np.sqrt(X)

# Build the linear model using Scikit-Learn
from sklearn.linear_model import LinearRegression
y = pressure['temperature']
sqrt_model = LinearRegression()
sqrt_model.fit(sqrt_X, y)

# Display regression coefficients and R-squared value of the model
sqrt_model.intercept_, sqrt_model.coef_, sqrt_model.score( log_X, y )
```

```
(98.56139249917803, array([11.44621468]), 0.29600246256782614)
```

The model is $\hat{t} = 98.561 + 11.446\sqrt{p}$, with t and p having the same meanings as above.

Content last modified on 24 July 2023.

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 using statsmodels

We'll assume that you have fit a single linear model to your data, as in the code below, which uses fake example data. You can replace it with your actual data.

```
import statsmodels.api as sm

xs = [  34,   9,  78,  60,  22,  45,  83,  59,  25 ]
ys = [ 126, 347, 298, 309, 450, 187, 266, 385, 400 ]

xs = sm.add_constant( xs )
model = sm.OLS( ys, xs )
results = model.fit()
```

We can use Python's `conf_int()` function to find the confidence interval for the model coefficients. You can change the `alpha` parameter to specify a different significance level. Note that if you have a multiple regression model, it will make confidence intervals for all of the coefficient values.

```
results.conf_int( alpha=0.05 )
```

```
array([[172.63807531, 535.52642049],
       [ -4.49196063,   2.47393542]])
```

Each list in the array represents the 95% confidence interval for the corresponding coefficient in the model beginning with the intercept and each regression coefficient thereafter. Accordingly, the 95% confidence interval for the regression coefficient is $[-4.49196063, 2.47393542]$.

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

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 using statsmodels

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()
```

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

```
model.summary()
```

```
/opt/conda/lib/python3.11/site-packages/scipy/stats/_stats_py.py:1806: UserWarning: kurtosistest only valid for n
  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>Mon, 24 Jul 2023</td> <th>  Prob (F-statistic):</th>  <td> 0.122</td>
```

Time:

```
            <td>17:47:21</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.

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

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 using pandas and NumPy

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	0.488293	0.151749	0.485939	0.278562	0.998647
1	0.405459	0.766983	0.915349	0.099784	0.518523
2	0.312085	0.498104	0.526030	0.745883	0.292882
3	0.313217	0.826840	0.254793	0.942009	0.456271
4	0.657147	0.024847	0.769884	0.140779	0.427270

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([[ 0.04524431, -0.02545402],
       [-0.02545402,  0.12901528]])
```

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	0.045244	-0.025454	0.005095	-0.015552	-0.006827
col2	-0.025454	0.129015	0.009857	0.062661	-0.013753
col3	0.005095	0.009857	0.084701	-0.048114	0.014510
col4	-0.015552	0.062661	-0.048114	0.087198	-0.023934
col5	-0.006827	-0.013753	0.014510	-0.023934	0.057866

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

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

```
array([[ 1.        , -0.33316075],
       [-0.33316075,  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	-0.333161	0.082300	-0.247604	-0.133423
col2	-0.333161	1.000000	0.094296	0.590780	-0.159177
col3	0.082300	0.094296	1.000000	-0.559850	0.207259
col4	-0.247604	0.590780	-0.559850	1.000000	-0.336937
col5	-0.133423	-0.159177	0.207259	-0.336937	1.000000

Content last modified on 24 July 2023.

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 using statsmodels

Let's assume that you already fit the linear model, as shown in the code below. This one uses a small amount of fake data, but it's just an example. See also how to fit a linear model to two columns of data.

```
# Below is the fake data as an example. You can replace with your real data.
x = [  34,   9,  78,  60,  22,  45,  83,  59,  25 ]
y = [ 126, 347, 298, 309, 450, 187, 266, 385, 400 ]

# Use statsmodels to build a linear regression model
import statsmodels.api as sm
x = sm.add_constant( x )
model = sm.OLS( y, x ).fit()
```

The standard error is shown as part of the model summary, reported by statsmodels's built-in `summary` function. See the column entitled "std err" in the output below.

```
model.summary()
```

```
/opt/conda/lib/python3.10/site-packages/scipy/stats/_stats_py.py:1736: UserWarning: kurtosistest only valid for n
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

OLS Regression Results

Dep. Variable:

```
      <td>y</td>        <th>  R-squared:         </th> <td>   0.063</td>
```

Model:

```
            <td>OLS</td>       <th>  Adj. R-squared:    </th> <td>  -0.071</td>
```

Method:

```
       <td>Least Squares</td>  <th>  F-statistic:       </th> <td>  0.4693</td>
```

Date:

```
        <td>Mon, 24 Jul 2023</td> <th>  Prob (F-statistic):</th>  <td> 0.515</td>
```

Time:

```
            <td>20:38:01</td>     <th>  Log-Likelihood:    </th> <td> -53.705</td>
```

No. Observations:

```
  <td>     9</td>      <th>  AIC:               </th> <td>   111.4</td>
```

Df Residuals:

```
      <td>     7</td>      <th>  BIC:               </th> <td>   111.8</td>
```

Df Model:

```
         <td>     1</td>      <th>                     </th>     <td> </td>
```

Covariance Type:

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

coef

std err

t

P>|t|

[0.025]

const

354.0822

76.733

4.614

0.002

172.638

535.526

x1

-1.0090

1.473

-0.685

0.515

-4.492

2.474

Omnibus:

```
  <td> 2.324</td> <th>  Durbin-Watson:     </th> <td>   1.618</td>
```

Prob(Omnibus):

0.313

Jarque-Bera (JB):

1.079

Skew:

```
     <td>-0.832</td> <th>  Prob(JB):          </th> <td>   0.583</td>
```

Kurtosis:

```
  <td> 2.674</td> <th>  Cond. No.          </th> <td>    112.</td>
```

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

If we need to extract just the estimates or their standard errors, we can use code like the following.

```
model.params # just the model coefficients
```

```
array([354.0822479 ,  -1.00901261])
```

```
model.bse # just the standard errors of those estimates
```

```
array([76.73277161,  1.47293931])
```

The standard error of the estimate for the intercept is is 76.73277161 and the standard error of the estimate for the slope is 1.47293931.

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.

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 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 using statsmodels

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 multivariate linear model.

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()
```

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/python3.10/site-packages/scipy/stats/_stats_py.py:1736: UserWarning: kurtosistest only valid for n
  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>Mon, 24 Jul 2023</td> <th>  Prob (F-statistic):</th>  <td> 0.122</td>
```

Time:

```
            <td>20:42:37</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 24 July 2023.

See a problem? Tell us or edit the source.

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 using SciPy

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)
```

```
SignificanceResult(statistic=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 24 July 2023.

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 using statsmodels

Let's assume that you've already built a linear model similar to the one below. This one uses a small amount of fake data, but it's just an example.

```
import statsmodels.api as sm

xs = [ 393, 453, 553, 679, 729, 748, 817 ]
ys = [  24,  25,  27,  36,  55,  68,  84 ]

xs = sm.add_constant( xs )
reg = sm.OLS( ys, xs ).fit()
```

We can extract the residuals of the model by calling the model's `resid` attribute.

```
reg.resid
```

```
array([  9.16263041,   2.19945659,  -9.07249979, -16.79516483,
        -4.43114302,   6.04718527,  12.88953537])
```

The result is an array of the residuals for every value in the data set.

Content last modified on 24 July 2023.

See a problem? Tell us or edit the source.