Annotated follow-along guide_ Explore linear regression with Python

January 7, 2024

1 Simple linear regression

Throughout the following exercises, you will learn to use Python to build a simple linear regression model. Before starting on this programming exercise, we strongly recommend watching the video lecture and completing the IVQ for the associated topics.

All the information you need for solving this assignment is in this notebook, and all the code you will be implementing will take place within this notebook.

As we move forward, you can find instructions on how to install required libraries as they arise in this notebook. Before we begin with the exercises and analyzing the data, we need to import all libraries and extensions required for this programming exercise. Throughout the course, we will be using pandas and statsmodels for operations, and seaborn for plotting.

1.1 Relevant imports

Begin by importing the relevant packages and data.

[1]:
```
# Import packages
import pandas as pd
import seaborn as sns
```

Note: Recall that the default for `head()` is to show the first 5 rows. If you change the value for `n`, you can show more rows. For example, if you load the sns dataset and call it "penguins," the command `penguins.head(3)` will show 3 rows.

[2]:
```
# Load dataset
penguins = sns.load_dataset("penguins")

# Examine first 5 rows of dataset
penguins.head()
```

[2]:
```
  species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0  Adelie  Torgersen            39.1           18.7              181.0
1  Adelie  Torgersen            39.5           17.4              186.0
2  Adelie  Torgersen            40.3           18.0              195.0
```

```
3  Adelie  Torgersen               NaN            NaN                NaN
4  Adelie  Torgersen              36.7           19.3              193.0

   body_mass_g     sex
0       3750.0    Male
1       3800.0  Female
2       3250.0  Female
3          NaN     NaN
4       3450.0  Female
```

From the first 5 rows of the dataset, we can see that there are several columns available: `species`, `island`, `bill_length_mm`, `bill_depth_mm`, `flipper_length_mm`, `body_mass_g`, and `sex`. There also appears to be some missing data.

1.2 Data cleaning (not shown in videos)

For the purposes of this course, we are focusing our analysis on Adelie and Gentoo penguins, and will be dropping any missing values from the dataset. In a work setting, you would typically examine the data more thoroughly before deciding how to handle missing data (i.e., fill in, drop, etc.). Please refer back to previous program content if you need to review how to handle missing data.

[3]:
```python
# Keep Adelie and Gentoo penguins, drop missing values
penguins_sub = penguins[penguins["species"] != "Chinstrap"]
penguins_final = penguins_sub.dropna()
penguins_final.reset_index(inplace=True, drop=True)
```

You can review the documentation for `dropna()` and `reset_index()`. In short, the `dropna()` function by default removes any rows with any missing values in any of the columns. The `reset_index()` function resets the index values for the rows in the dataframe. Typically, you use `reset_index()` after you've finished manipulating the dataset. By setting `inplace=True`, you will not create a new DataFrame object. By setting `drop=True`, you will not insert a new index column into the DataFrame object.

1.3 Exploratory data analysis

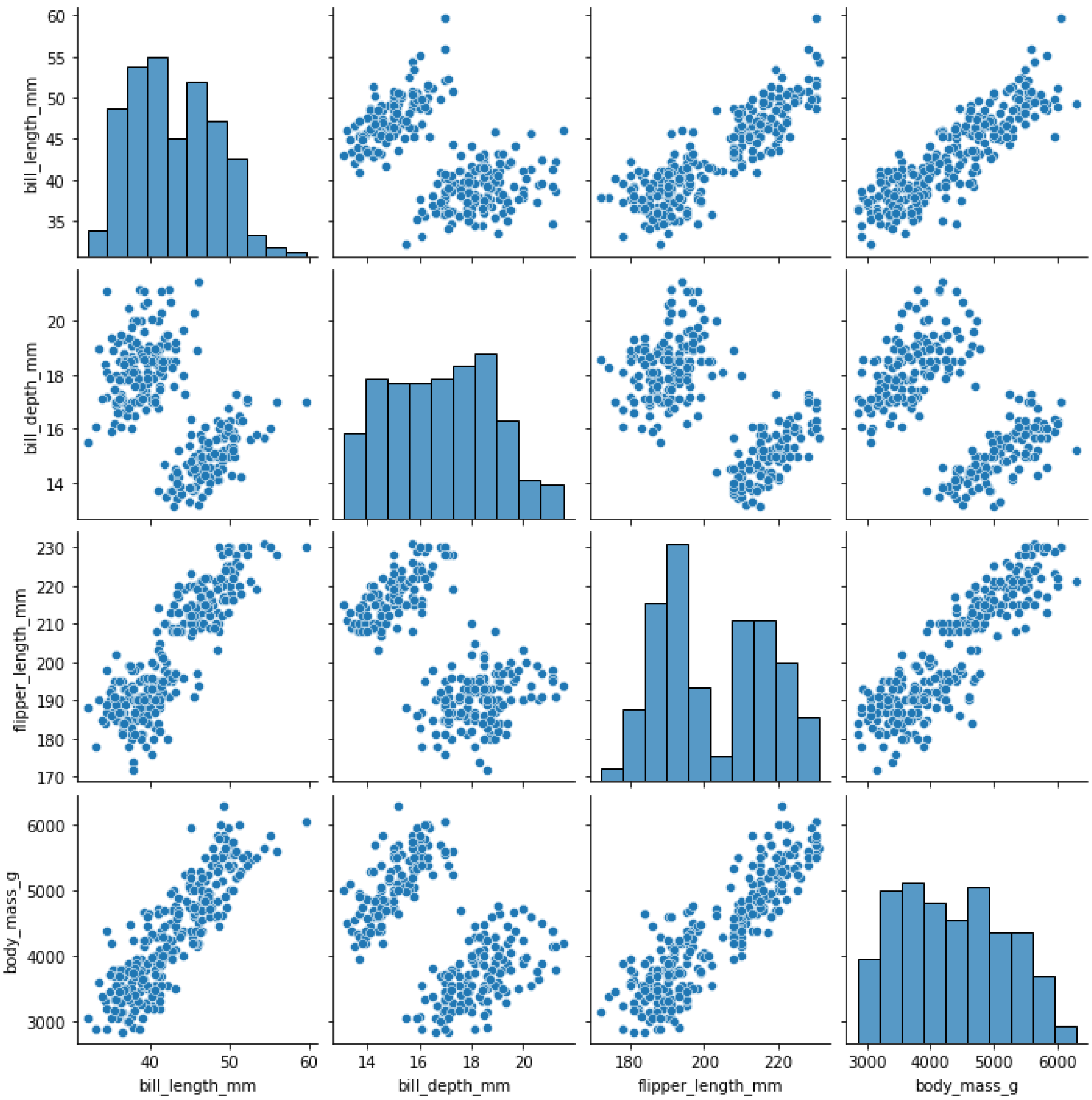
Before you construct any model, it is important to get more familiar with your data. You can do so by performing exploratory data analysis or EDA. Please review previous program materials as needed if you would like to refamiliarize yourself with EDA concepts.

Since this part of the course focuses on simple linear regression, you want to check for any linear relationships among variables in the dataframe. You can do this by creating scatterplots using any data visualization package, for example `matplotlib.plt`, `seaborn`, or `plotly`.

To visualize more than one relationship at the same time, we use the `pairplot()` function from the `seaborn` package to create a scatterplot matrix.

[4]:
```python
# Create pairwise scatterplots of data set
sns.pairplot(penguins_final)
```

[4]:
```
<seaborn.axisgrid.PairGrid at 0x7f7dd01dc4d0>
```



From the scatterplot matrix, you can observe a few linear relationships: * bill length (mm) and flipper length (mm) * bill length (mm) and body mass (g) * flipper length (mm) and body mass (g)

1.4 Model construction

Based on the above scatterplots, you could probably run a simple linear regression on any of the three relationships identified. For this part of the course, you will focus on the relationship between bill length (mm) and body mass (g).

To do this, you will first subset the variables of interest from the dataframe. You can do this by using double square brackets `[[]]`, and listing the names of the columns of interest.

[5]:
```
# Subset Data
ols_data = penguins_final[["bill_length_mm", "body_mass_g"]]
```

Next, you can construct the linear regression formula, and save it as a string. Remember that the y or dependent variable comes before the `~`, and the x or independent variables comes after the `~`.

Note: The names of the x and y variables have to exactly match the column names in the dataframe.

[6]:
```
# Write out formula
ols_formula = "body_mass_g ~ bill_length_mm"
```

Lastly, you can build the simple linear regression model in `statsmodels` using the `ols()` function. You can import the `ols()` function directly using the line of code below.

[7]:
```
# Import ols function
from statsmodels.formula.api import ols
```

Then, you can plug in the `ols_formula` and `ols_data` as arguments in the `ols()` function. After you save the results as a variable, you can call on the `fit()` function to actually fit the model to the data.

[8]:
```
# Build OLS, fit model to data
OLS = ols(formula = ols_formula, data = ols_data)
model = OLS.fit()
```

Lastly, you can call the `summary()` function on the `model` object to get the coefficients and more statistics about the model. The output from `model.summary()` can be used to evaluate the model and interpret the results. Later in this section, we will go over how to read the results of the model output.

[9]:
```
model.summary()
```

[9]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:            body_mass_g   R-squared:                       0.769
Model:                            OLS   Adj. R-squared:                  0.768
Method:                 Least Squares   F-statistic:                     874.3
Date:                Thu, 17 Nov 2022   Prob (F-statistic):           1.33e-85
Time:                        18:34:33   Log-Likelihood:                -1965.8
No. Observations:                 265   AIC:                             3936.
Df Residuals:                     263   BIC:                             3943.
Df Model:                           1
```

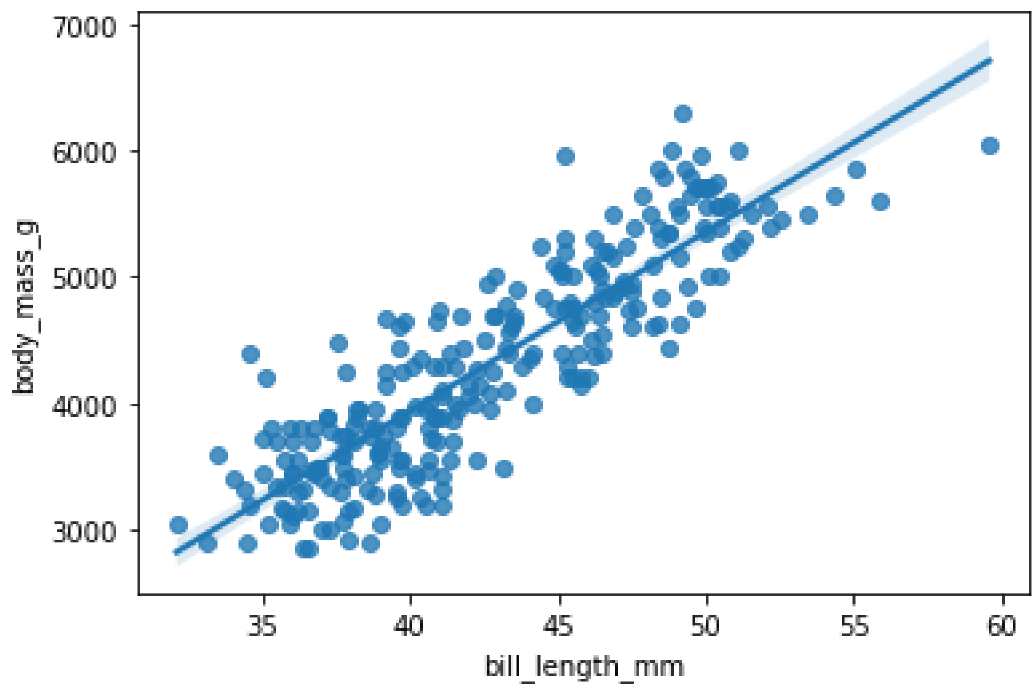
```
Covariance Type:            nonrobust
================================================================================
==
                    coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--
Intercept      -1707.2919    205.640     -8.302      0.000   -2112.202
-1302.382
bill_length_mm   141.1904      4.775     29.569      0.000     131.788
150.592
==============================================================================
Omnibus:                        2.060   Durbin-Watson:                   2.067
Prob(Omnibus):                  0.357   Jarque-Bera (JB):                2.103
Skew:                           0.210   Prob(JB):                        0.349
Kurtosis:                       2.882   Cond. No.                         357.
==============================================================================

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

You can use the `regplot()` function from `seaborn` to visualize the regression line.

[10]:
```
sns.regplot(x = "bill_length_mm", y = "body_mass_g", data = ols_data)
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7dcc2cbb50>
```



1.5 Finish checking model assumptions

As you learned in previous videos, there are four main model assumptions for simple linear regression, in no particular order: 1. Linearity 2. Normality 3. Independent observations 4. Homoscedasticity

You already checked the linearity assumption by creating the scatterplot matrix. The independent observations assumption is more about data collection. There is no reason to believe that one penguin's body mass or bill length would be related to any other penguin's anatomical measurements. So we can check off assumptions 1 and 3.

The normality and homoscedasticity assumptions focus on the distribution of errors. Thus, you can only check these assumptions after you have constructed the model. To check these assumptions, you will check the residuals, as an approximation of the errors.

To more easily check the model assumptions and create relevant visualizations, you can first subset the X variable by isolating just the `bill_length_mm` column. Additionally, you can save the predicted values from the model using the `model.predict(X)` function.

[11]:
```
# Subset X variable
X = ols_data["bill_length_mm"]

# Get predictions from model
fitted_values = model.predict(X)
```

Then, you can save the model residuals as a variable by using the `model.resid` attribute.

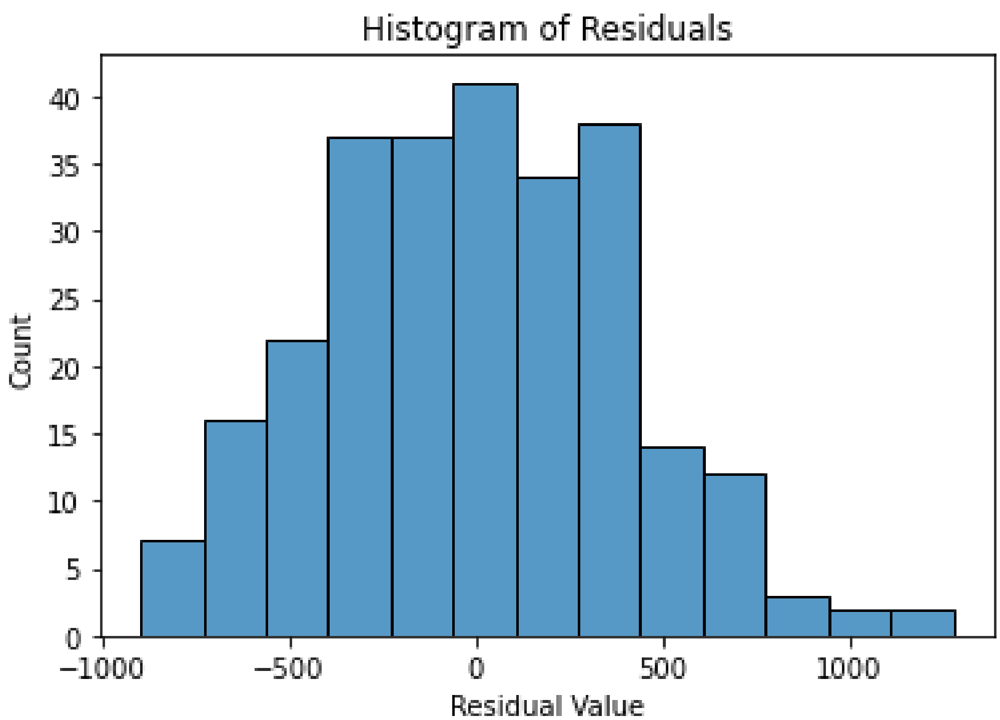
[12]:
```
# Calculate residuals
residuals = model.resid
```

1.5.1 Check the normality assumption

To check the normality assumption, you can create a histogram of the residuals using the `histplot()` function from the `seaborn` package.

From the below histogram, you may notice that the residuals are almost normally distributed. In this case, it is likely close enough that the assumption is met.

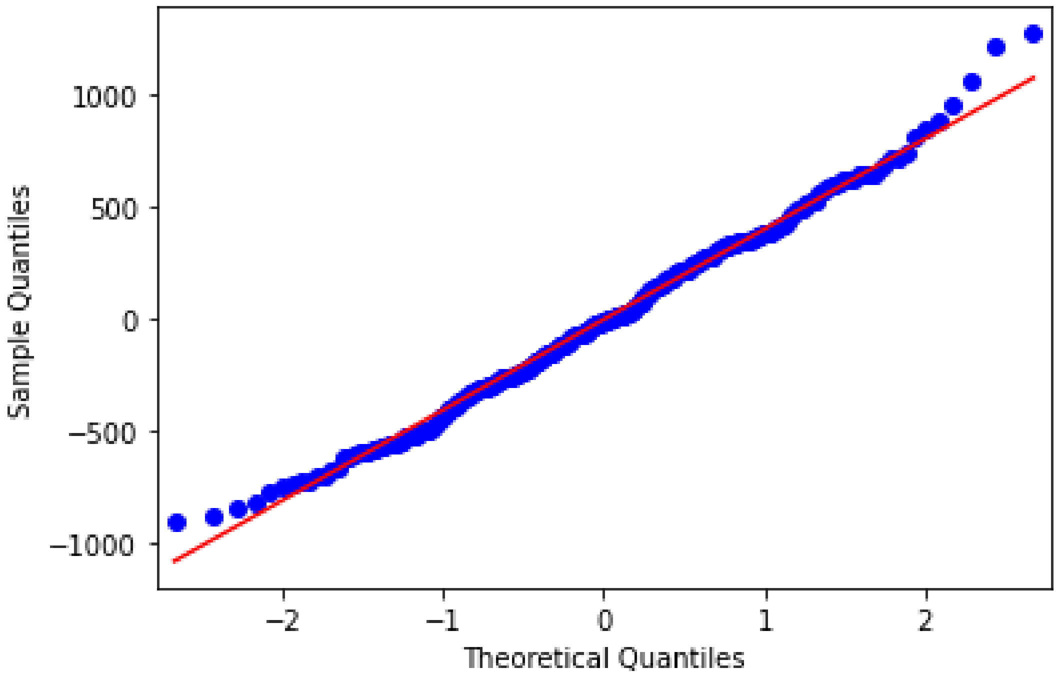
[13]:
```
import matplotlib.pyplot as plt
fig = sns.histplot(residuals)
fig.set_xlabel("Residual Value")
fig.set_title("Histogram of Residuals")
plt.show()
```



Another way to check the normality function is to create a quantile-quantile or Q-Q plot. Recall that if the residuals are normally distributed, you would expect a straight diagonal line going from the bottom left to the upper right of the Q-Q plot. You can create a Q-Q plot by using the `qqplot` function from the `statsmodels.api` package.

The Q-Q plot shows a similar pattern to the histogram, where the residuals are mostly normally distributed, except at the ends of the distribution.

[14]:
```
import matplotlib.pyplot as plt
import statsmodels.api as sm
fig = sm.qqplot(model.resid, line = 's')
plt.show()
```



1.5.2 Check the homoscedasticity assumption

Lastly, we have to check the homoscedasticity assumption. To check the homoscedasticity assumption, you can create a scatterplot of the fitted values and residuals. If the plot resembles a random cloud (i.e., the residuals are scattered randomly), then the assumption is likely met.

You can create one scatterplot by using the `scatterplot()` function from the `seaborn` package. The first argument is the variable that goes on the x-axis. The second argument is the variable that goes on the y-axis.

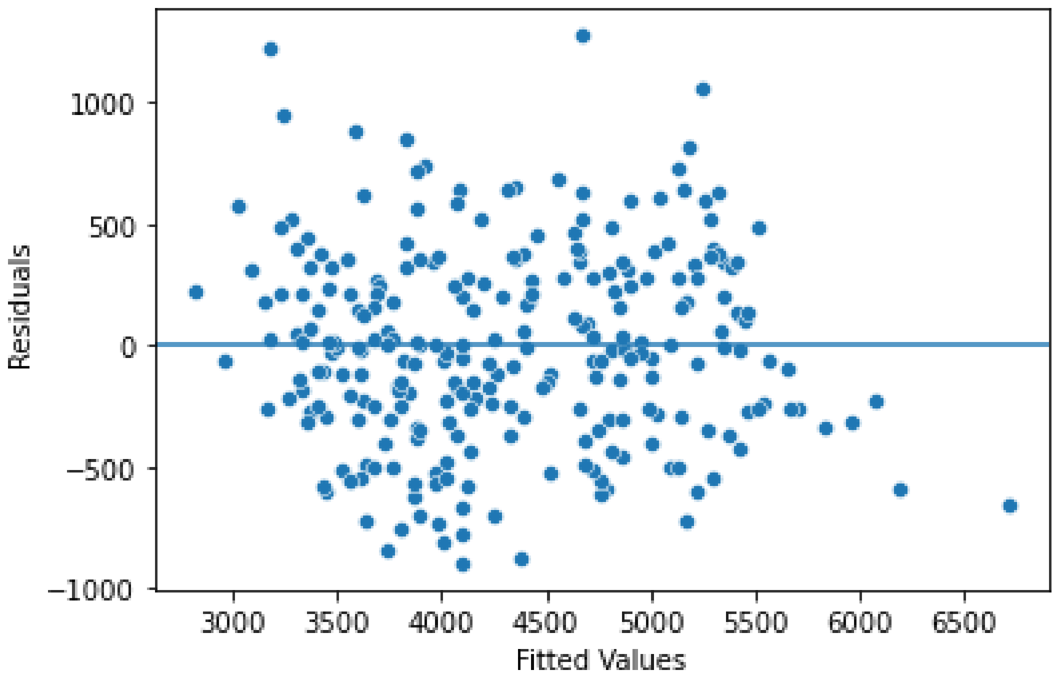
[15]:
```
# Import matplotlib
import matplotlib.pyplot as plt
fig = sns.scatterplot(x=fitted_values, y=residuals)

# Add reference line at residuals = 0
fig.axhline(0)

# Set x-axis and y-axis labels
fig.set_xlabel("Fitted Values")
```

```
fig.set_ylabel("Residuals")

# Show the plot
plt.show()
```



Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.

You now understand how to build a simple linear regression model with Python. Going forward, you can start using simple linear regression models with your own datasets.