17.2.3

Linear Regression Example

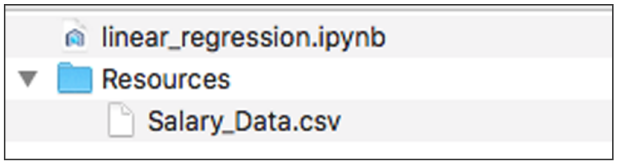
Jill is impressed with your clear explanation of the two main uses of supervised learning: regression and classification. She would like you to now learn how to implement a machine learning model in Python. You will use Scikit-learn, a Python machine learning library. Since you are already familiar with using linear regression, Jill suggests that you implement a linear regression model with Scikit-learn.

To get started, download the CSV file with salary data.

Download Salary_Data.csv (https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_17/Salary_Data.csv)

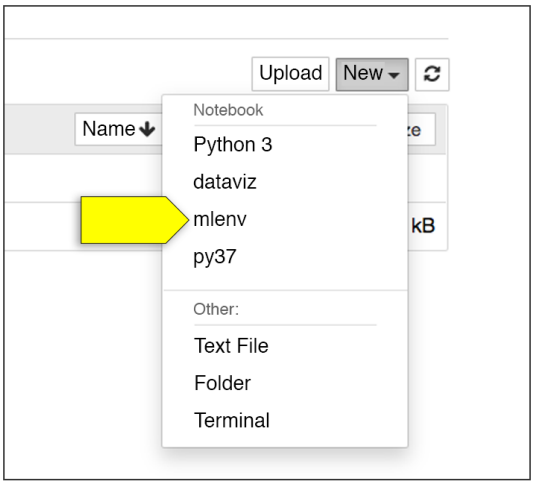
As we go through the following example, you'll learn the basic procedure for implementing a supervised learning model: create a model, train the model, and then create predictions.

Create a directory named "linear_regression_salary." In this directory, create another directory called "Resources," into which you'll copy the CSV file that you downloaded:



Navigate to the directory and start the Jupyter Notebook.

Create a new notebook, and be sure to select the mlenv kernel for the notebook:



Paste the following code into the first cell and run it. (Each code block from this point on will be a separate cell in your notebook.)

```
import pandas as pd
from pathlib import Path
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

`sklearn` is the Scikit-learn machine learning library for Python. It has many modules, including one for linear regression, which we use here.

In the next cell, load the CSV file as a Pandas DataFrame and preview the DataFrame:

```
df = pd.read_csv(Path('./Resources/Salary_Data.csv'))
df.head()
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

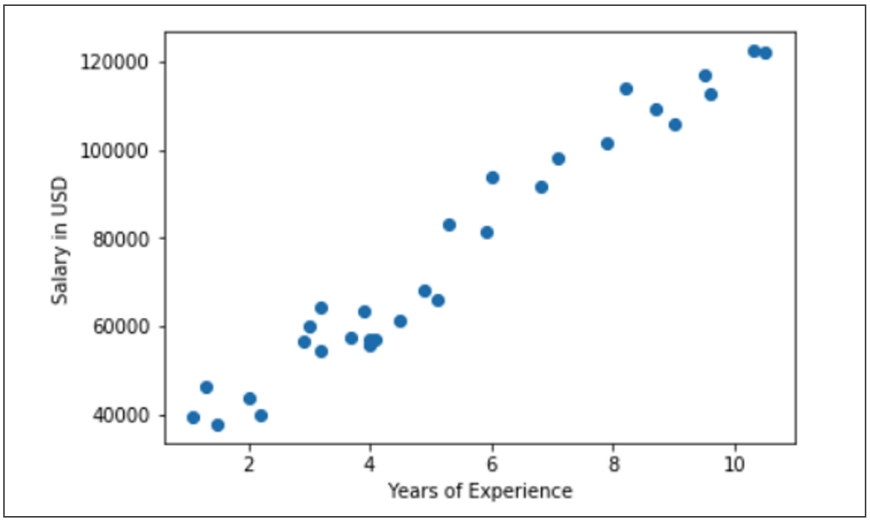
Use the Path library to locate the data file. Its use is not required, but it does make it easier to identify the file path whether you use a Mac or a Windows machine.

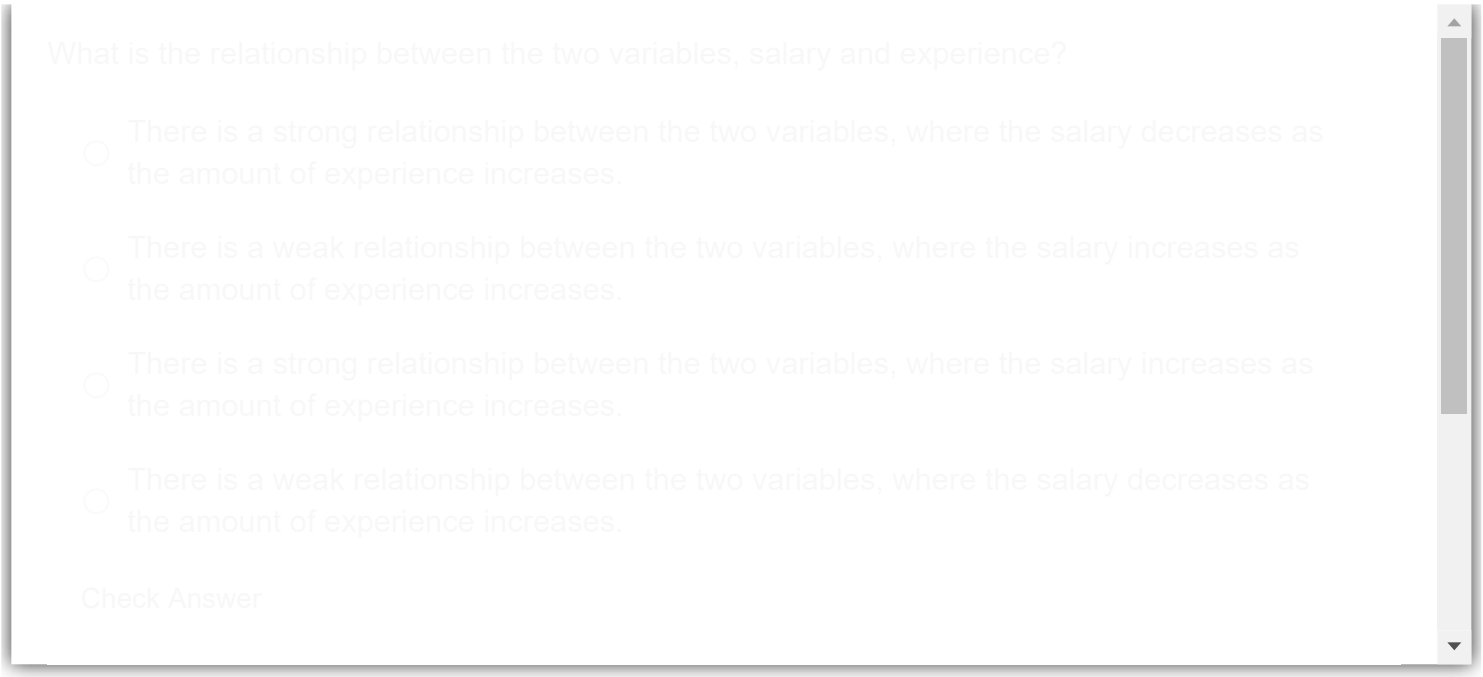
Each row in the DataFrame represents an employee and has two columns containing that employee's years of work experience and salary.

The target variable is `Salary`, meaning that the goal of the linear regression model is to predict a person's salary based on years of work experience.

First, let's visually inspect the relationship between `Years of Experience` and `Salary`:

```
plt.scatter(df.YearsExperience, df.Salary)
plt.xlabel('Years of Experience')
plt.ylabel('Salary in USD')
plt.show()
```





However, we want to do better than merely establish that there's a relationship. We want to use the available data to make the most accurate predictions possible.

The next line of code formats the data to meet the requirements of the Scikit-learn library:

```
X = df.YearsExperience.values.reshape(-1, 1)
```

In the scatter plot, the `YearsExperience` was placed on the x-axis. Conventionally, the independent variable is placed on the x-axis, and the dependent variable is placed on the y-axis. The years of experience is the independent variable here because we assume that employee salary depends on the years of experience.

The values from the `YearsExperience` column are modified by the `reshape()` method. This method allows data needs to be formatted, or shaped, to follow Scikit-learn's specifications as follows:

- The first argument of `reshape()` specifies the number of rows. Here, the argument is -1, and means that the number of rows is unspecified. Accordingly, the NumPy library will automatically identify the number of rows in the dataset.
- The second argument of `reshape()` refers to the number of columns. Here, the argument is 1, meaning that there is only one column of independent variables.

When we examine the first five entries in X, we see that the output is a two-dimensional NumPy array:

```
X[:5]
```

```
X[:5]
```

```
array([[1.1],
       [1.3],
       [1.5],
       [2. ],
       [2.2]])
```

When we examine the shape of X, we see that there are 30 rows and 1 column:

```
X.shape
```

```
# The shape of X is 30 samples, with a single feature (column)
X.shape
```

```
(30, 1)
```

IMPORTANT

Using a dataset with an incorrect shape, or formatting, can cause errors in Scikit-learn. Remember that you may have to use `reshape()` to format your data properly.

Next, we assign the target variable, or the `Salary` column, to y. Although it's possible to reshape this column, as we did with X, it's not required in this instance:

```
y = df.Salary
```

To summarize, the dataset has been separated into X and y components: X is the input data, and y is the output. You also may have noticed that X is capitalized, while y is not. Although you aren't required to stick to this convention, it's a common practice when using Scikit-learn.

The next step is to create an instance of the linear regression model. An object is instantiated, or created, from `sklearn.linear_model`'s `LinearRegression` class. Instantiation here means that the `LinearRegression` class is like a template that contains the algorithms required to perform linear regression. From this template, a specific object called `model` is created that will analyze the data and store information specific to this dataset:

```
model = LinearRegression()
```

After a model is instantiated, it will analyze the data and attempt to learn patterns in the data. This learning stage is alternatively called **fitting** or **training:**

```
model.fit(X, y)
```

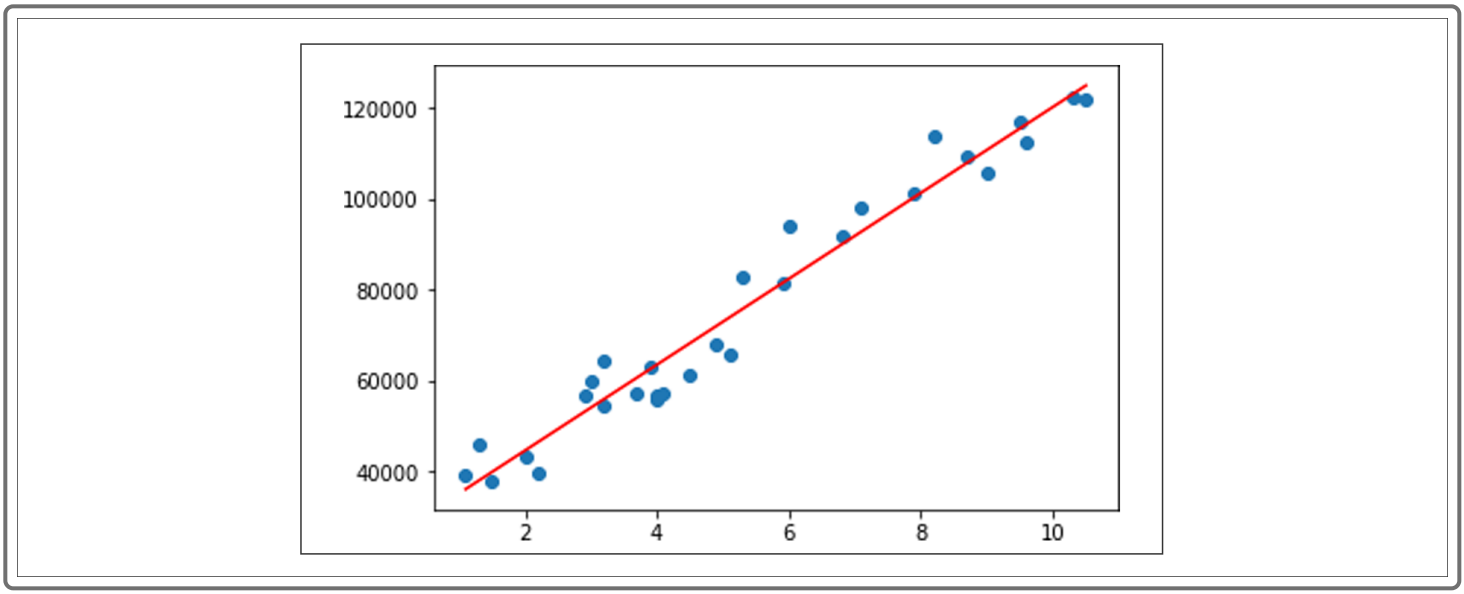
After the learning stage, the `predict()` method is used to generate predictions: given any number of a person's years of experience, the model will predict the salary:

```
y_pred = model.predict(X)
print(y_pred.shape)
```

Printing the shape of `y_pred` returns `(30,)`. Recall that a linear regression model generates a straight line that best fits the overall trend of the data. Based on the 30 available data points in X, `model.predict(X)` returned 30 predictions through which a straight line can be drawn.

Furthermore, the model can make predictions for years of experience beyond the range in the current data. Let's plot the predictions as a red line against the data points:

```
plt.scatter(X, y)
plt.plot(X, y_pred, color='red')
plt.show()
```



The best fit line is in red, drawn through the predictions. The maximum value of years of experience in the current dataset is 10.5, but the linear regression model can extrapolate beyond it. That is, if given 12 years of experience, it will be able to guess the associated salary level.

Finally, we can examine the specific parameters of our model: the slope and the y-intercept. The slope is represented by `model.coef_`, and `model.intercept_` is the y-intercept:

```
print(model.coef_)
print(model.intercept_)
```

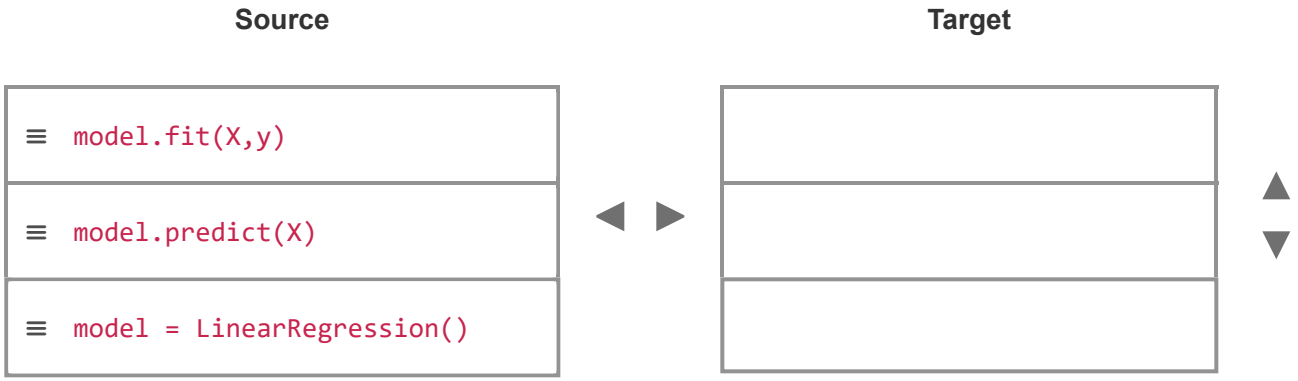
Let's summarize the basic pattern for supervised learning we used in this linear regression example:

1. Split the data into input (X) and output (y).
2. Create an instance of the model with `model = LinearRegression()`.
3. Train the model with the dataset with `model.fit(X,y)`.

4. Create predictions with `y_pred = model.predict(X)`.

You'll encounter this pattern repeatedly in machine learning, and you'll expand on it to perform additional tasks, such as evaluating how well your model performs.

Assuming that you have data properly formatted for Scikit-learn, what is the order you would perform supervised learning in using linear regression?



Check Answer

Finish

SKILL DRILL

To gain familiarity with the basics of supervised machine learning, try running the notebook again from the beginning. You may download the following Jupyter Notebook to check your work.

Download 17-2-3-linear_regression_salary.zip (https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-online/module_17/17-2-3-linear_regression_salary.zip)