Karlijn Willems
March 16th, 2017

PYTHON +4

Python Exploratory Data Analysis Tutorial

Learn the basics of Exploratory Data Analysis (EDA) in Python with Pandas, Matplotlib and NumPy, such as sampling, feature engineering, correlation, etc.

As you will know by now, the Python data manipulation library Pandas is used for data manipulation; For those who are just starting out, this might imply that this package can only be handy when preprocessing data, but much less is true: Pandas is also great to explore your data and to store it after you're done preprocessing the data.

Additionally, for those who have been following DataCamp's Python tutorials or that have already been introduced to the basics of SciPy, NumPy, Matplotlib and Pandas, it might be a good idea to recap some of the knowledge that you have built up.

Today's tutorial will actually introduce you to some ways to explore your data efficiently with all the above packages so that you can start modeling your data:

- You'll first learn how to import data, which is the first step that you need to complete successfully before you can start your analysis.
- If you're not sure what Exploratory Data Analysis (EDA) is and what the exact difference between EDA and Data Mining is, this section will explain it for you before you start the tutorial!
- Then, you'll get a basic description of your data. You'll focus on getting some descriptive statistics, checking out the first and last rows of your DataFrame, retrieving samples

- After gathering some information on your data, it might also be a good idea to also take a deeper look at it by querying or indexing the data. You can use this technique to test some of the basic hypotheses that you might have about the data.
- Now that you have inspected your data, you'll probably already see that there are some features that can be of interest to your analysis: you'll see which ones can influence your analysis positively with feature engineering and feature selection.
- Next, you'll see that an initial exploration is good, but you will also need to have an idea of the challenges that your data can pose, such as missing values or outliers, and, of course, how you can handle those challenges, and
- Lastly, you'll also learn how to discover patterns in your data, by either visualizing your data easily and quickly with the Python data visualization packages Matplotlib and Bokeh, or by using specific functions to compute the correlation between attributes.

Are you interested in taking a Pandas course? Consider taking one of our three courses that we have made in collaboration with Continuum Analytics, the creator and driving force behind Anaconda, such as the Pandas Foundations course!

Importing The Data

To start exploring your data, you'll need to start by actually loading in your data. You'll probably know this already, but thanks to the Pandas library, this becomes an easy task: you import the package as `pd`, following the convention, and you use the `read_csv()` function, to which you pass the URL in which the data can be found and a `header` argument. This last argument is one that you can use to make sure that your data is read in correctly: the first row of your data won't be interpreted as the column names of your DataFrame.

Alternatively, there are also other arguments that you can specify to ensure that your data is read in correctly: you can specify the delimiter to use with the `sep` or `delimiter` arguments, the column names to use with `names` or the column to use as the row labels for the resulting DataFrame with `index_col`.

script.py IPython Shell

```
# Import the `pandas` library as `pd`
import pandas as pd

# Load in the data with `read_csv()`
digits = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-databases
/optdigits/optdigits.tra",
                     header=None)

# Print out `digits`
print(digits)
```

Run

Note that in this case, you made use of `read_csv()` because the data happens to be in a comma-separated format. If you have files that have another separator, you can also consider using other functions to load in your data, such as `read_table()`, `read_excel()`, `read_fwf()` and `read_clipboard`, to read in general delimited files, Excel files, Fixed-Width Formatted data and data that was copied to the Clipboard, respectively.

Also, you'll find `read_sql()` as one of the options to read in an SQL query or a database table into a DataFrame. For even more Input functions, consider this section of the Pandas documentation.

What is Exploratory Data Analysis (EDA)?

can also use it to prepare the data for modeling. The thing that these two probably have in common is a good knowledge of your data to either get the answers that you need or to develop an intuition for interpreting the results of future modeling.

There are a lot of ways to reach these goals: you can get a basic description of the data, visualize it, identify patterns in it, identify challenges of using the data, etc.

One of the things that you'll often see when you're reading about EDA is Data profiling. Data profiling is concerned with summarizing your dataset through descriptive statistics. You want to use a variety of measurements to better understand your dataset. The goal of data profiling is to have a solid understanding of your data so you can afterwards start querying and visualizing your data in various ways. However, this doesn't mean that you don't have to iterate: exactly because data profiling is concerned with summarizing your dataset, it is frequently used to assess the data quality. Depending on the result of the data profiling, you might decide to correct, discard or handle your data differently.

You'll learn more about data profiling in a next post.

EDA And Data Mining (DM)

EDA distinguishes itself from data mining, even though the two are closely related, as many EDA techniques have been adopted into data mining. Also the goals of the two are very similar: EDA indeed makes sure that you explore the data in such a way that interesting features and relationships between features will become more clear. In EDA, you typically explore and compare many different variables with a variety of techniques to search and find systematic patterns. Data mining, on the other hand, is concerned with extracting patterns from the data. Those patterns provide insights into relationships between variables that can be used to improve business decisions. Also, in both cases, you have no a priori expectations or expectations that are not complete about the relations between the variables.

However, in general, Data Mining can be said to be more application-oriented, while EDA is concerned with the basic nature of the underlying phenomena. In other words, Data Mining is relatively less concerned with identifying the specific relations between the involved variables. As a result, Data Mining accepts a "black box" approach to data exploration and doesn't only use techniques that are also used in EDA but also techniques

Basic Description of the Data

Like you read above, EDA is all about getting to know your data. One of the most elementary steps to do this is by getting a basic description of your data. A basic description of your data is indeed a very broad term: you can interpret it as a quick and dirty way to get some information on your data, as a way of getting some simple, easy-to-understand information on your data, to get a basic feel for your data, etc.

This section won't make a distinction between these interpretations: it will indeed introduce you to some of the ways that you can quickly gather information on your DataFrame that is easy to understand.

Describing The Data

For example, you can use the `describe()` function to get various summary statistics that exclude NaN values. Consider this example in which you describe the famous Iris dataset. The data has already been loaded in for you in the DataCamp Light chunk:

script.py IPython Shell

```
1  iris.describe()
```

Run

You see that this function returns the count, mean, standard deviation, minimum and maximum values and the quantiles of the data. Note that, of course, there are many packages available in Python that can give you those statistics, including Pandas itself. Using this function is just one of the ways to get this information.

assess the quality of your data. Then you'll be able to decide whether you need to correct, discard or deal with the data in anohter way. This is usually the data profiling step. This step in the EDA is meant to understand the data elements and its anomalies a bit better and to see how the data matches the documentation on the one hand and accommodates to the business needs on the other hand.

Note that you'll come back to the data profiling step as you go through your exploratory data analysis, as the quality of your data can be impacted by the steps that you'll go through.

First and Last DataFrame Rows

Now that you have got a general idea about your data set, it's also a good idea to take a closer look at the data itself. With the help of the `head()` and `tail()` functions of the Pandas library, you can easily check out the first and last lines of your DataFrame, respectively.

Inspect the first and last five rows of the handwritten digits data with the `head()` and `tail()` functions in the DataCamp Light chunk below. The data has already been loaded in for you in the DataCamp Light chunk:

script.py IPython Shell

```
# Inspect the first 5 rows of `digits`
first = digits._____(5)

# Inspect the last 5 rows
last = digits.____(5)
```

Solution Run

You'll see that the result of the `head()` and `tail()` functions doesn't quite say much when you're not familiar with this kind of data.

relevant information on how the data was collected and also states the number of attributes and rows, which can be handy to check whether you have imported the data correctly.

Additionally, go back to your initial finding: the numerical values in the rows. At first sight, you might not think that there is a problem, as the integer values appear to be correct and don't raise any flags when you're looking at it at first.

But if you would have done all of this on another data set that you had in front of you and that might have had, for example, date time information, a quick glance on the result of these lines of code might have raised the following questions: "Has my data been read in as a DateTime?", "How can I check this?" and "How can I change the data type?".

These are deeper questions that you'll typically address in the data profiling step, which will be addressed in a next post.

Sampling The Data

If you have a large dataset, you might consider taking a sample of your data as an easy way to get a feel for your data quickly. As a first and easy way to do this, you can make use of the `sample()` function that is included in Pandas, just like this:

script.py IPython Shell

```
# Take a sample of 5
digits.sample(5)
```

Solution Run

`random` package that has a module `sample` that will allow you to sample your data, in combination with `range()` and `len()`. Note that you also make use of `ix` to select the exact rows of your DataFrame that you want to include in your sample.

If you don' have an idea of why you use `ix` in this context, DataCamp's more specific tutorial can be of help! It covers these more general topics in detail. Go and check it out by clicking on the link that has been included above!

For now, let's practice our Python skills! Get started on the exercise below:

script.py IPython Shell

```
# import `sample` from `random`
from random import ______

# Create a random index
randomIndex = np.array(sample(range(len(______)), 5))

# Get 5 random rows
digitsSample = digits.ix[__________]

# Print the sample
print(____________)
```

Solution Run

A Closer Look At Your Data: Queries

Now that you have taken a quick look at your data and have seen what it's about, you're ready to dive a little bit deeper: it's time to inspect the data further by querying the data.

This goes easily with the `query()` function, which allows you to test some very simple hypotheses that you have about your data, such as "Is the petal length usually greater than the sepal length?" or "Is the petal length sometimes equal to the sepal length?".

script.py IPython Shell

```
In [1]:
```

Run

You'll see that this hypothesis doesn't hold. You get an empty DataFrame back as a result.

Note that this function can also be expressed as `iris[iris.Petal_length > iris.Sepal_length]`.

The Challenges of Your Data

Now that you've gathered some basic information on your data, it's a good idea to just go a little bit deeper into the challenges that your data might pose. If you have already gone through the data profiling step, you'll be aware of missing values, you'll have an idea of which values might be outliers, etc.

This section will describe some of the basic ways to already get an idea of these things and will describe how you can handle the data in case you do find irregularities in your data. Note once again that you will go or have gone deeper into identifying these irregularities in the data profiling step, and that it's normal to return to this step once you have handled some of the challenges that your data poses.

Missing Values

Something that you also might want to check when you're exploring your data is whether or not the data set has any missing values.

Examining this is important because when some of your data is missing, the data set can lose expressiveness, which can lead to weak or biased analyses. Practically, this means that when you're missing values for certain features, the chances of your classification or predictions for the data being off only increase.

Of course, the cause of you missing data in your data set can be the result of a faulty extraction or import of the data, or it might be the result of the collection process. The systems that give you the data might malfunction or the survey that you sent out might

can be useful. Remember that you can use data profiling to get a better idea of your data quality. You can read more about how you can discover patterns of missing data in a follow-up post.

In short, the causes of missing data can be various and largely depend on the data context, but can also depend on yourself. That's why you have first inspected your data when you imported it in one of the previous steps!

To identify the rows that contain missing values, you can use `isnull()`. In the result that you'll get back, you'll see `True` or `False` appearing in each cell: `True` will indicate that the value contained within the cell is a missing value, `False` means that the cell contains a 'normal' value.

script.py IPython Shell

```
# Identifiy missing values
pd.isnull(_____)
```

Solution Run

In this case, you see that the data is quite complete: there are no missing values.

Note that you could have also read this in the data set description of the UCI Machine Learning Repository which was linked above, where you'll have seen that there are no missing values listed for the data.

However, this will not be the case in every data set that you'll come across. That's why it's good to know what you can do when you do run across a situation where you need to think about what you want to be doing with the missing data.

to be careful with this procedure, as deleting data might also bias your analysis. That's why you should ask yourself the question of whether the probability of certain data that is missing for a record is the same as for all other records. If the probability doesn't vary record-per-record, deleting the missing data is a valid option.

- Besides deletion, there are also methods that you can use to fill up cells if they contain missing values with so-called "imputation methods". If you already have a lot of experience with statistics, you'll know that imputation is the process of replacing missing data with substituted values. You can either fill in the mean, the mode or the median. Of course, here you need to think about whether you want to take, for example, the mean or median for all missing values of a variable, or whether you want to replace the missing values based on another variable. For example, for data in which you have records that have features with categorical variables such as "male" or "female", you might also want to consider those before replacing the missing values, as the observations might differ from males and females. If this is the case, you might just calculate the average of the female observations and then fill out the missing values for other "female" records with this average.
- Estimate the value with the help of regression, ANOVA, logistic regression or another modelling technique. This is by far the most complex way to fill in the values.
- You fill in the cells with values of records that are most similar to the one that has missing values. You can use KNN or K-Nearest Neighbors in cases such as these.

Note that there are advantages and drawbacks to every one of the above ways to fill in missing data! You'll want to consider things such as time, expense, the nature of your data, etc. before making a final decision on this.

When you have made a final decision on what you're going to do with the missing data, read on to see how you can implement the changes that you want to see in your data.

Filling Missing Values

If you do decide to fill in the values with imputation, you can still choose how you want to make this happen!

the results of a survey that asks for people's salary. Assuming a context where your audience are all from the same class in society and the likelihood of the respondents answering to the question is the same for every person, you can opt to calculate the mean of the people that did answer the question and use that mean to fill in the values of people that didn't answer.

```
# Import NumPy
import numpy as np

# Calculate the mean
mean = np.mean(df.Salary)

# Replace missing values with the mean
df. = df.Salary.fillna(mean)
```

Of course, you don't necessarily need to pass in a value to `fillna()`. You can also propagate non-null values forward or backward by adding the argument `method` to the `fillna()` function. Pass in `ffill` or `bfill` to specify you want to fill the values backward or forward.

Drop Labels With Missing Values

To exclude columns or rows that contain missing values, you can make use of Pandas' `dropna()` function:

```
# Drop rows with missing values
df.dropna(axis=0)

# Drop columns with missing values
df.dropna(axis=1)
```

Interpolation

Alternatively, you can also choose to interpolate missing values: the `interpolate()` function will perform a linear interpolation at the missing data points to "guess" the value that is most likely to be filled in.

You can also add the `method` argument to gain access to fancier interpolation methods, such as polynomial interpolation or cubic interpolation, but when you want to use these types of interpolation, you'll need to have SciPy installed.

Of course, there are limits to the interpolation, especially if the NaN values for interpolation are too far from the last valid observation. In such cases, you want to add a `limit` argument to the original code. You pass a positive integer to it and this number will determine how many values after a non-NaN value will be filled out. The default limit direction is forward, but also this you can change by adding `limit_direction`

Outliers

Just like missing values, your data might also contain values that diverge heavily from the big majority of your other data. These data points are called "outliers". To find them, you can check the distribution of your single variables by means of a box plot or you can make a scatter plot of your data to identify data points that don't lie in the "expected" area of the plot.

The causes for outliers in your data might vary, going from system errors to people interfering with the data through data entry or data processing, but it's important to consider the effect that they can have on your analysis: they will change the result of statistical tests such as standard deviation, mean or median, they can potentially decrease the normality and impact the results of statistical models, such as regression or ANOVA.

To deal with outliers, you can either delete, transform, or impute them: the decision will again depend on the data context. That's why it's again important to understand your data and identify the cause for the outliers:

- If the outlier value is due to data entry or data processing errors, you might consider deleting the value.
- You can transform the outliers by assigning weights to your observations or use the natural log to reduce the variation that the outlier values in your data set cause.
- Just like the missing values, you can also use imputation methods to replace the extreme values of your data with median, mean or mode values.

You can use the functions that were described in the above section to deal with outliers in your data.

Your Data's Features

Note that this step is one that you do iteratively with other data science tasks: you'll build your models and validate, but after, you might decide to adjust the features and iterate on building the model again, etc.

Feature Engineering

You can use feature engineering as a way to increase the predictive power of learning algorithms by creating features from raw data that will help the learning process. You'll do this by creating additional relevant features from the existing raw features in the data.

Feature engineering is something that will cost some time to get the hang of; It's not always clear what you can do with the raw data so that you can help the predictive power of the data. But maybe the following list can provide some help when you're looking for ways to engineer features for your dataset:

- Factorize a feature: encode categorical variables into numerical ones with `factorize()`, like in this example:

script.py IPython Shell

```
# Factorize the values
labels,levels = pd.factorize(iris.Class)

# Save the encoded variables in `iris.Class`
iris.Class = labels

# Print out the first rows
iris.Class.head()
```

Run

- Bin continuous variables in groups: use `cut()` to cut the values for a column in bins

```
# Define your own bins
mybins = range(0, df.age.max(), 10)

# Cut the data with the help of the bins
df['age_bucket'] = pd.cut(df.age, bins=mybins)

# Count the number of values per bucket
df['age_bucket'].value_counts()
```

Run

- Scale features: center your data around 0. You can make use of Scikit-Learn's preprocessing module:

script.py IPython Shell

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(X)

rescaledX = scaler.transform(X)
```

Run

Tip: look into the preprocessing module of Scikit-Learn more closely, you'll see that it contains some handy functions that will help you to come up with new features for your data!

Note that these are just some of the ways in which you can engineer new features to make your data more predictive! The key is to brainstorm new features or combine old features together and try to test their effectiveness with Scikit-Learn.

When you select features, you select the key subset of original data features in an attempt to reduce the dimensionality of the training problem. This seems very similar to other dimensionality reduction techniques that you might already know, such as PCA. Yet, there is a difference: PCA combines similar (correlated) attributes and creates new ones that are considered superior to the original attributes of the dataset. Feature selection doesn't combine attributes: it evaluates the quality and predictive power and selects the best set.

To find important features, you can make use of the RandomForest algorithm: it randomly generates thousands of decision trees and takes turns leaving out each variable in fitting the model. This way, you can calculate how much better or worse a model does when you leave one variable out of the equation. You can use the Scikit-Learn Python library to implement this algorithm:

script.py IPython Shell

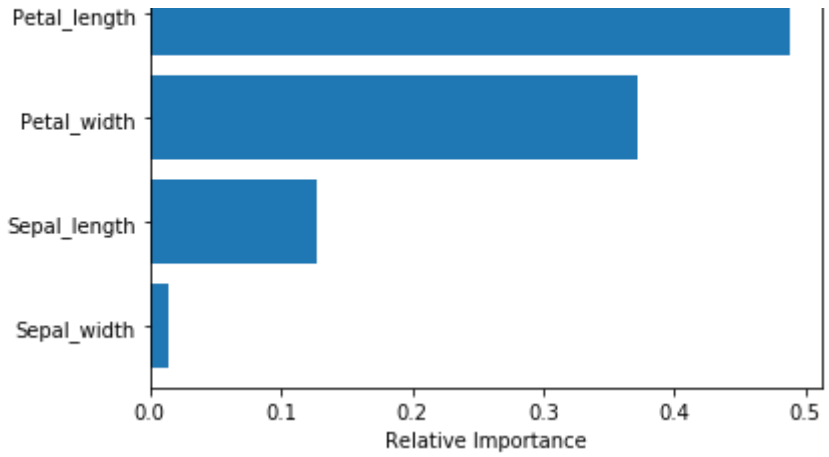
```
# Import `RandomForestClassifier`
from sklearn.ensemble import RandomForestClassifier

# Isolate Data, class labels and column values
X = iris.iloc[:,0:4]
Y = iris.iloc[:,-1]
names = iris.columns.values

# Build the model
rfc = RandomForestClassifier()

# Fit the model
rfc.fit(X, Y)

# Print the results
print("Features sorted by their score:")
print(sorted(zip(map(lambda x: round(x, 4), rfc.feature_importances_), names),
reverse=True))
```

Run

You'll see that the best feature set is one that includes the petal length and petal width data.

Note that you can also visualize the results of the feature selection with Matplotlib:

```
# Import `pyplot` and `numpy`
import matplotlib.pyplot as plt
import numpy as np

# Isolate feature importances
importance = rfc.feature_importances_

# Sort the feature importances
sorted_importances = np.argsort(importance)

# Insert padding
padding = np.arange(len(names)-1) + 0.5

# Plot the data
plt.barh(padding, importance[sorted_importances], align='center')

# Customize the plot
plt.yticks(padding, names[sorted_importances])
plt.xlabel("Relative Importance")
plt.title("Variable Importance")

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



Patterns In Your Data

One of the next steps that you can take in the exploration of your data is the identification of patterns in your data, which includes correlation between data attributes or between missing data. One of the things that can help in doing this is the visualization of your data; And this doesn't need to be static: dare to go for interactive visualizations of your data with the Python libraries Bokeh or Plotly.

Correlation Identification With Matplotlib

Now that you have looked at the numbers and analyzed your data in a quantitative way, you'll also find it useful to consider you data in a visual way. It's time to also explore the data visually.

To easily and quickly do this, you can make use of the Python data visualization library Matplotlib. The only thing that stands in your way is, ironically, your data: as you are well aware, your data has 64 columns or features. When you have so many features, it's said that you're working with high dimensional data.

What dimensional data exactly is, you'll learn in our machine learning tutorial, but for now it's good to understand that, if you want to visualize your data in a 2D or 3D plot, you'll need your data to only have two or three dimensions. This means that you'll need to reduce your data's dimensions.

This means that you'll have to make use of Dimensionality Reduction techniques, such as Principal Component Analysis (PCA):

```
from sklearn.decomposition import ___

# Build the model
pca = PCA(n_components=2)

# Reduce the data, output is ndarray
reduced_data = pca.fit_transform(_______)

# Inspect the shape of `reduced_data`
reduced_data._____

# print out the reduced data
print(_______________)
```

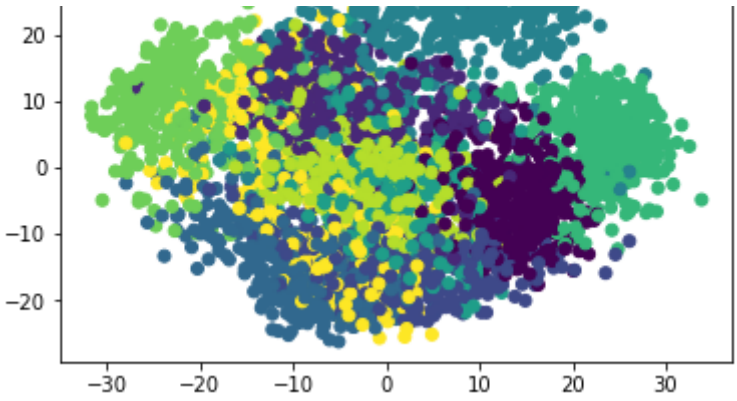
Solution Run

When you inspect the reduced data, that the columns or features have now been reduced to only two. The number of rows or observations is still the same, namely, 3823. And now that your data is in the right format, it's time to get to the plotting!

The choice of the right plot is already a great start, but what will you choose?

In this case, you're exploring the data, so you probably want to discover possible correlations between the attributes of your data. A scatter plot is probably a good way to visualize this: it allows you to identify a relationship between the two features that you have gained from the dimensionality reduction.

```
import matplotlib.pyplot as plt

plt.scatter(reduced_data[:,0], reduced_data[:,1], c=labels, cmap = 'viridis')

plt.show()
```



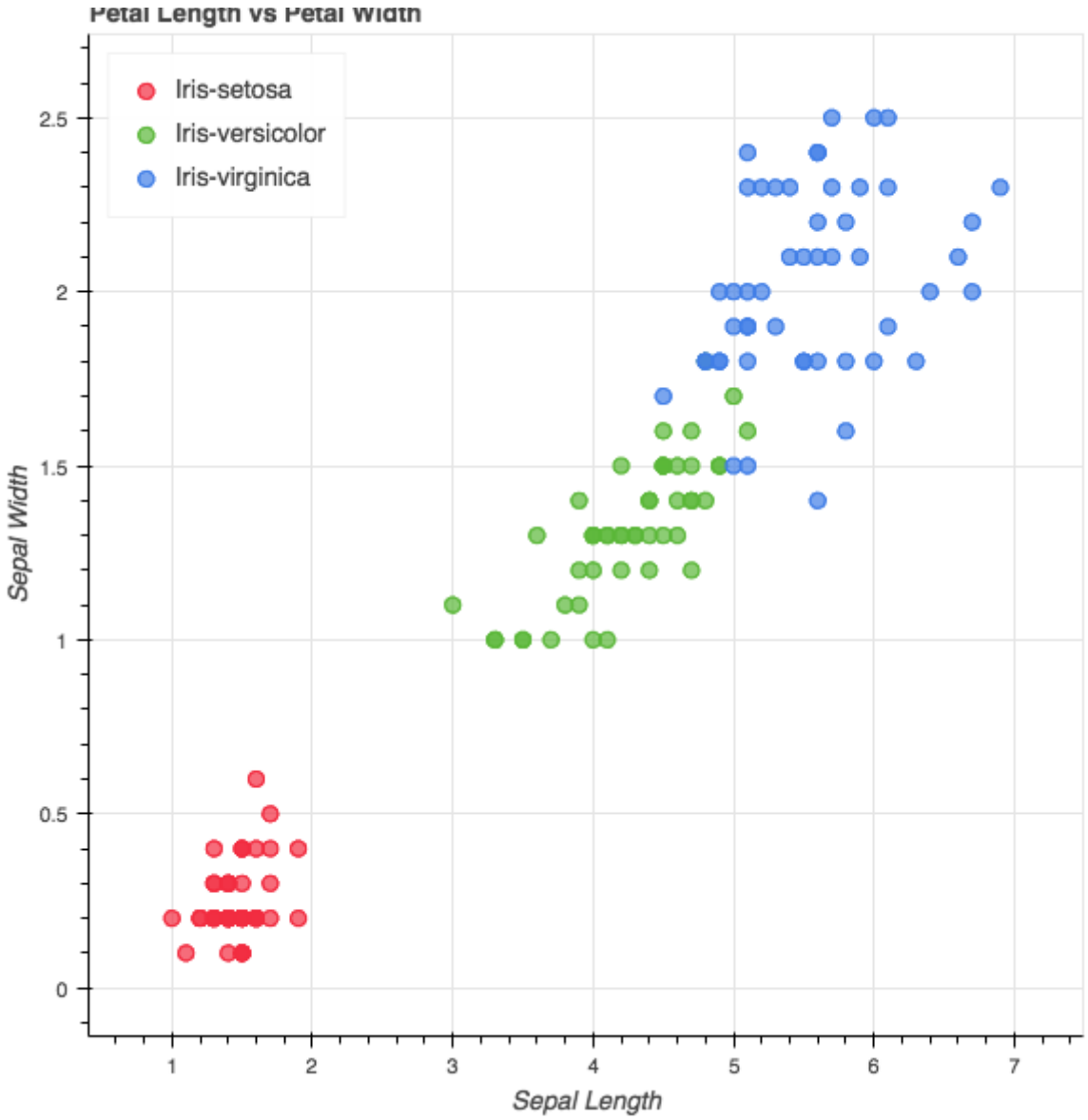
Correlation Identification With Bokeh

Secondly, you can also consider using Bokeh to construct an interactive plot to discover correlations between the attributes in your data. The Bokeh library is a Python interactive visualization library that targets modern web browsers for presentation. It's ideal if you're working with large or streaming datasets, but as you can see in the following example, you can also use it for "regular" data.

The code is very simple: you import the necessary modules, construct the scatter plot, configure the default output state to generate output saved to a file when show() is called. Finally, you call `show()` to see the scatter plot that you have constructed!

```
# Import the necessary modules
from bokeh.charts import Scatter, output_file, show

# Construct the scatter plot
p = Scatter(iris, x='Petal_length', y='Petal_width', color="Class", title="Petal Length vs
            xlabel="Sepal Length", ylabel="Sepal Width")

# Output the file
output_file('scatter.html')

# Show the scatter plot
show(p)
```

The result is elegant:



Note that this is a static, saved image of the plot but that the resulting plot in your notebook or terminal will be interactive! Of course, this is just one simple example of how you can use Bokeh to make interactive graphs. Make sure to check out the Bokeh Gallery for more inspiration or take DataCamp's Interactive Data Visualization with Bokeh course.

Correlation Identification With Pandas

The plots that you have seen in the previous sections are a visual way of exploring correlation between the attributes of your data. But that doesn't mean that you can not explore this measure in a quantitative way! And when you do decide to do this, make use of Pandas' `corr()` function. But do note that the NaN or null values are excluded in this computation!

```
3
4  # Kendall Tau correlation
5  iris.corr('kendall')
6
7  # Spearman Rank correlation
8  iris.corr('spearman')
```

Run

Note that the two last correlation measures require you to rank the data before calculating the coefficients. You can easily do this with `rank()` : for the exercise above, the `iris` data was ranked by executing `iris.rank()` for you.

Furthermore, there are some assumptions that these correlations work with: the Pearson correlation assumes that your variables are normally distributed, that there is a straight line relationship between each of the variables and that the data is normally distributed about the regression line. The Spearman correlation, on the other hand, assumes that you have two ordinal variables or two variables that are related in some way, but not linearly.

The Kendall Tau correlation is a coefficient that represents the degree of concordance between two columns of ranked data. You can use the Spearman correlation to measure the degree of association between two variables. These seem very similar to each other, don't they?

Even though the Kendal and the Spearman correlation measures seem similar, but they do differ: the exact difference lies in the fact that the calculations are different. The Kendal Tau coefficient is calculated by the number of concordant pairs minus the number of discordant pairs divided by the total number of pairs. The Spearman coefficient is the sum of deviation squared by `n` times `n` minus 1.

Spearman's coefficient will usually be larger than the Kendall's Tau coefficient, but this is not always the case: you'll get a smaller Spearman's coefficient when the deviations are huge among the observations of your data. The Spearman correlation is very sensitive to this and this might come in handy in some cases!

concordant pairs relative to discordant pairs and the Spearman's coefficient doesn't do that. You can also argue that the Kendall Tau correlation has a more intuitive interpretation and easier to calculate, that it gives a better estimate of the corresponding population parameter and that the p values are more accurate in small sample sizes.

Tip add the `print()` function to see the results of the specific pairwise correlation compututations of columns.

Onwards!

Congrats, you have made it to the end of our Pandas tutorial! You now have some mastered some of the basic techniques that you can use to explore your data with Python.

If you want to deep dive into the topic even further, our Pandas course series is perfect: check out our Pandas Foundations, Merging DataFrames with Pandas or Manipulating DataFrames with Pandas courses.

If, however, you're ready to move on from Pandas and explore the Matplotlib package some more, consider taking DataCamp's Python data visualization tutorial or start modelling your data! Continue to our machine learning tutorial to find out how you can build a machine learning model to recognize the handwritten digits automatically!

PYTHON +2

Hierarchical indices, groupby and pandas

Hugo Bowne-Anderson
October 3rd, 2017

DATA MANIPULATION +3

Exploratory Data Analysis of Craft Beers: Data Profiling

Jean-Nicholas Hould
April 13th, 2017

COMMENTS

Pito Salas
10/05/2018 07:30 PM
bokeh.charts is no longer supported!

▲ 3

Bin (Vince) Wang
12/09/2018 01:42 AM
The .ix indexer is deprecated starting in Pandas 0.20.0.

▲ 2

Grant Wilson
14/01/2019 06:12 PM
I've been searching for some relevant information on feature engineering. Is there a full guide with the examples to try? I've also sent a request to hire essay writers on training courses. The

▲ 2

Ashley Blossom
09/01/2020 12:58 PM

There are so many approaches to analyze data. Il used EDA a few weeks ago but faced some difficulties so I got help cheap London assignment writers by Ace My Assignment which provides different assignment help. EDA is one of the approach which is used to analyze data it happens when we have some values in the observation.

▲ 1

Marjan Radfar
04/02/2019 08:51 AM

Thanks for your useful article :)

▲ 2

Alexa Warnner
09/07/2019 01:01 PM

data Sources. House Prices: Advanced Regression Techniques. House Prices: Advanced ... Thanks for your explanations, this is great path to *exploratory data analysis*. i have also wrote some blog on data analysis in my website on apple itunes support

▲ 1

Carina Cruz
29/07/2019 09:33 PM

Great tutorial!

▲ 1

Neela Blore
31/07/2019 07:59 PM

Plant Irrigation Water Sprinkler Robot
PLC Based Paper Cutting Machine
PLC Based Sorting System Using Metal Detection
Pneumatic Power Steering System
Pneumatic Powered Metal Pick and Place Arm
Pneumatic Powered Wall Climbing Robot
Power Generation Using Electromagnetic Suspension
Power Generation using Speed Breakers
Power Generator Forearms Machine

Push Based Box Transport Mechanism
Regenerative Braking System Project
Regenerative Breaking With Power Monitor
Remote Controlled Automobile Using Rf
Remote Controlled Mini Forklift
Remote Controlled Pick and Place Robotic Vehicle
Remote Controlled Robotic Arm Using Rf
RF Controlled Beach Cleaner Robotic Vehicle
RF Controlled Robotic Vehicle
RF Controlled Robotic Vehicle With Metal Detection Project
Rf Controlled Spy Robot With Night Vision Camera
Rough Terrain Beetle Robot
Rough Terrain Vehicle Using Rocker Bogie Mechanism
Sand Filter and Separator Project
Six Legged Spider Bot using Klann Mechanism
Smart Solar Grass Cutter With Lawn Coverage
Staircase Climbing Trolley
Steering Mechanism Vehicle With Joystick Control
Table Saw Project
Three Wheel Handicapped Steering Propulsion Cycle
Voice Controlled Robotic Vehicle
Zero Friction Electromagnetic Braking System Project

▲ 1