Practical No. 1

Name: Prathamesh Pawar | Roll No: B-23

```
import pandas as pd

iris_data = pd.read_csv('iris-data.csv')
iris_data.head()
```

```
   sepal_length_cm  sepal_width_cm  petal_length_cm  petal_width_cm  \
0              5.1             3.5              1.4             0.2
1              4.9             3.0              1.4             0.2
2              4.7             3.2              1.3             0.2
3              4.6             3.1              1.5             0.2
4              5.0             3.6              1.4             0.2

         class
0  Iris-setosa
1  Iris-setosa
2  Iris-setosa
3  Iris-setosa
4  Iris-setosa
```

We're in luck! The data seems to be in a usable format.

The first row in the data file defines the column headers, and the headers are descriptive enough for us to understand what each column represents. The headers even give us the units that the measurements were recorded in, just in case we needed to know at a later point in the project.

Each row following the first row represents an entry for a flower: four measurements and one class, which tells us the species of the flower.

One of the first things we should look for is missing data. Thankfully, the field researchers already told us that they put a 'NA' into the spreadsheet when they were missing a measurement.

We can tell pandas to automatically identify missing values if it knows our missing value marker.

```
iris_data = pd.read_csv('iris-data.csv', na_values=['NA'])
```

Voilà! Now pandas knows to treat rows with 'NA' as missing values.

Next, it's always a good idea to look at the distribution of our data — especially the outliers.

Let's start by printing out some summary statistics about the data set.

```
iris_data.describe()
```

```
       sepal_length_cm  sepal_width_cm  petal_length_cm
petal_width_cm
count       150.000000      150.000000       150.000000
```

```
145.000000
mean           5.644627        3.054667        3.758667
1.236552
std            1.312781        0.433123        1.764420
0.755058
min            0.055000        2.000000        1.000000
0.100000
25%            5.100000        2.800000        1.600000
0.400000
50%            5.700000        3.000000        4.350000
1.300000
75%            6.400000        3.300000        5.100000
1.800000
max            7.900000        4.400000        6.900000
2.500000
```

We can see several useful values from this table. For example, we see that five `petal_width_cm` entries are missing.

If you ask me, though, tables like this are rarely useful unless we know that our data should fall in a particular range. It's usually better to visualize the data in some way. Visualization makes outliers and errors immediately stand out, whereas they might go unnoticed in a large table of numbers.

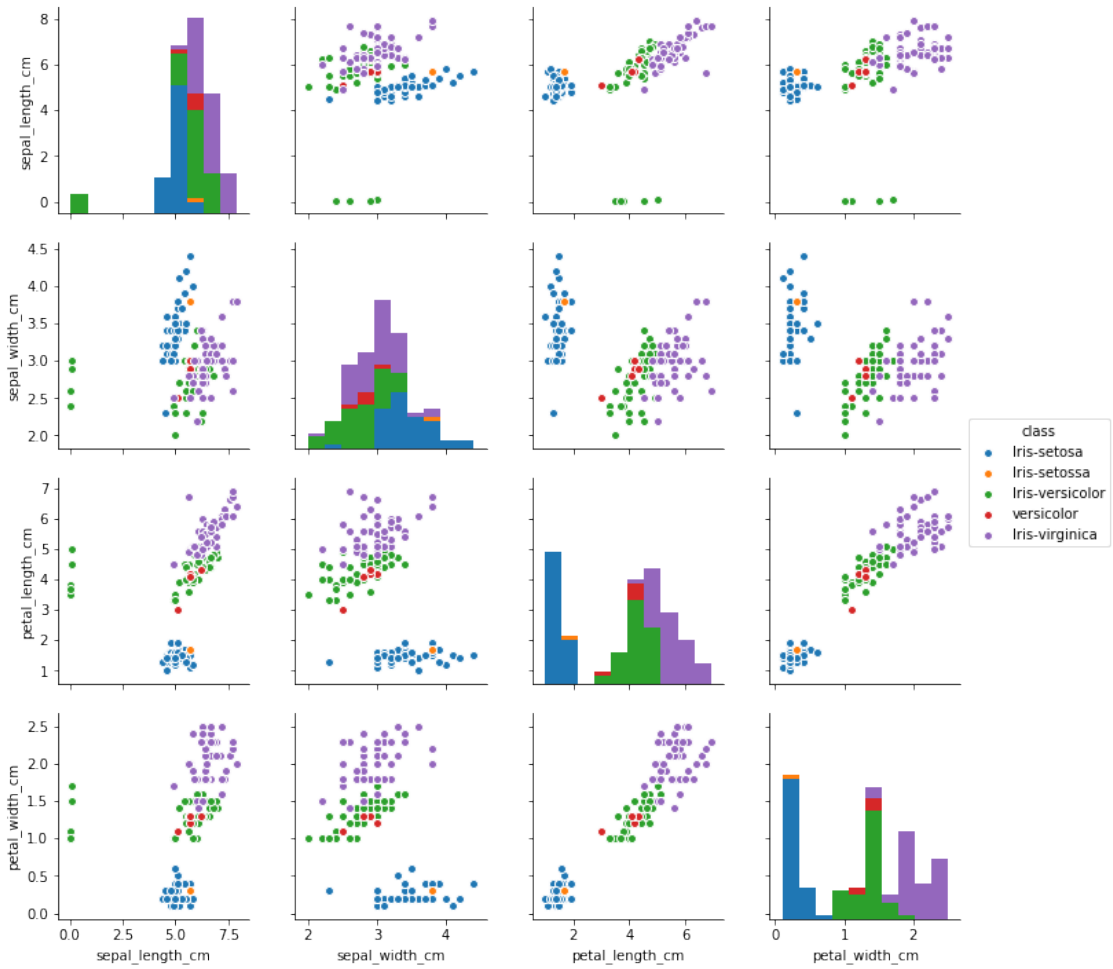
Since we know we're going to be plotting in this section, let's set up the notebook so we can plot inside of it.

```
# This line tells the notebook to show plots inside of the notebook
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sb
```

Next, let's create a **scatterplot matrix**. Scatterplot matrices plot the distribution of each column along the diagonal, and then plot a scatterplot matrix for the combination of each variable. They make for an efficient tool to look for errors in our data.

We can even have the plotting package color each entry by its class to look for trends within the classes.

```
# We have to temporarily drop the rows with 'NA' values
# because the Seaborn plotting function does not know
# what to do with them
sb.pairplot(iris_data.dropna(), hue='class')
;

''
```



From the scatterplot matrix, we can already see some issues with the data set:

1. There are five classes when there should only be three, meaning there were some coding errors.

2. There are some clear outliers in the measurements that may be erroneous: one `sepal_width_cm` entry for `Iris-setosa` falls well outside its normal range, and several `sepal_length_cm` entries for `Iris-versicolor` are near-zero for some reason.

3. We had to drop those rows with missing values.

In all of these cases, we need to figure out what to do with the erroneous data. Which takes us to the next step...

Step 3: Tidying the data

[go back to the top]

Now that we've identified several errors in the data set, we need to fix them before we proceed with the analysis.

Let's walk through the issues one-by-one.

> There are five classes when there should only be three, meaning there were some coding errors.

After talking with the field researchers, it sounds like one of them forgot to add `Iris-` before their `Iris-versicolor` entries. The other extraneous class, `Iris-setossa`, was simply a typo that they forgot to fix.

Let's use the DataFrame to fix these errors.

```
iris_data.loc[iris_data['class'] == 'versicolor', 'class'] = 'Iris-
versicolor'
iris_data.loc[iris_data['class'] == 'Iris-setossa', 'class'] = 'Iris-
setosa'

iris_data['class'].unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'],
dtype=object)
```

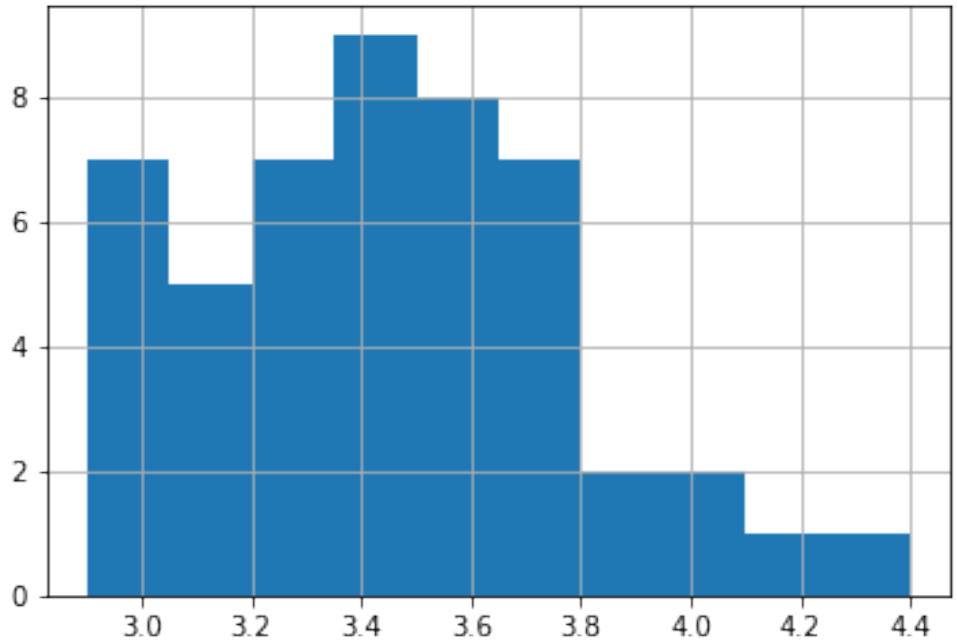
Much better! Now we only have three class types. Imagine how embarrassing it would've been to create a model that used the wrong classes.

> There are some clear outliers in the measurements that may be erroneous: one `sepal_width_cm` entry for `Iris-setosa` falls well outside its normal range, and several `sepal_length_cm` entries for `Iris-versicolor` are near-zero for some reason.

Fixing outliers can be tricky business. It's rarely clear whether the outlier was caused by measurement error, recording the data in improper units, or if the outlier is a real anomaly. For that reason, we should be judicious when working with outliers: if we decide to exclude any data, we need to make sure to document what data we excluded and provide solid reasoning for excluding that data. (i.e., "This data didn't fit my hypothesis" will not stand peer review.)

In the case of the one anomalous entry for `Iris-setosa`, let's say our field researchers know that it's impossible for `Iris-setosa` to have a sepal width below 2.5 cm. Clearly this entry was made in error, and we're better off just scrapping the entry than spending hours finding out what happened.

```
# This line drops any 'Iris-setosa' rows with a separal width less
than 2.5 cm
iris_data = iris_data.loc[(iris_data['class'] != 'Iris-setosa') |
(iris_data['sepal_width_cm'] >= 2.5)]
iris_data.loc[iris_data['class'] == 'Iris-setosa',
'sepal_width_cm'].hist()
;

''
```



Excellent! Now all of our `Iris-setosa` rows have a sepal width greater than 2.5.

The next data issue to address is the several near-zero sepal lengths for the `Iris-versicolor` rows. Let's take a look at those rows.

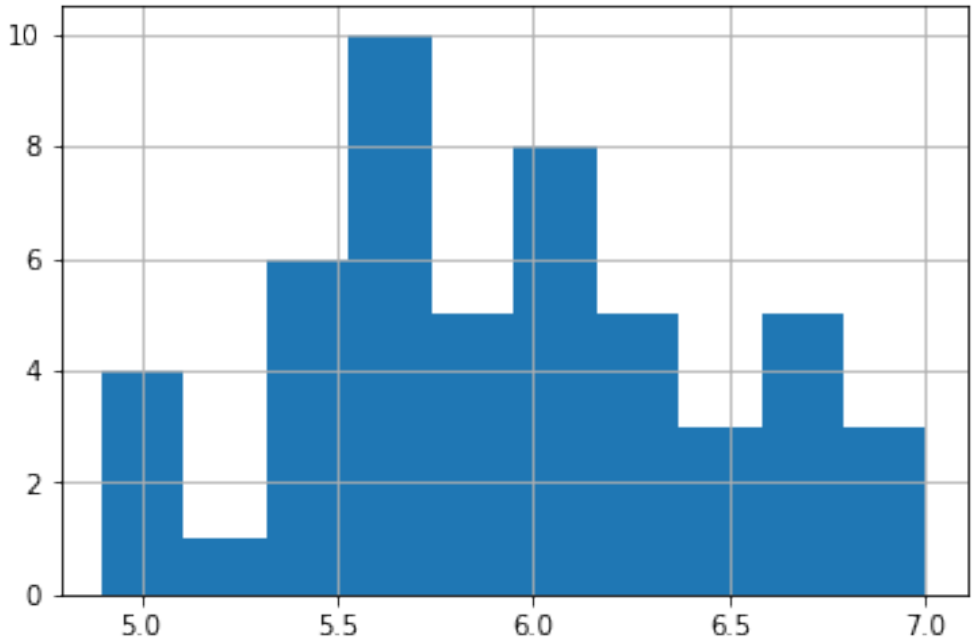
```
iris_data.loc[(iris_data['class'] == 'Iris-versicolor') &
              (iris_data['sepal_length_cm'] < 1.0)]
```

```
    sepal_length_cm  sepal_width_cm  petal_length_cm
petal_width_cm  \
77            0.067             3.0              5.0             1.7

78            0.060             2.9              4.5             1.5

79            0.057             2.6              3.5             1.0

80            0.055             2.4              3.8             1.1

81            0.055             2.4              3.7             1.0


              class
77  Iris-versicolor
78  Iris-versicolor
79  Iris-versicolor
80  Iris-versicolor
81  Iris-versicolor
```

How about that? All of these near-zero `sepal_length_cm` entries seem to be off by two orders of magnitude, as if they had been recorded in meters instead of centimeters.

After some brief correspondence with the field researchers, we find that one of them forgot to convert those measurements to centimeters. Let's do that for them.

```
iris_data.loc[(iris_data['class'] == 'Iris-versicolor') &
              (iris_data['sepal_length_cm'] < 1.0),
              'sepal_length_cm'] *= 100.0

iris_data.loc[iris_data['class'] == 'Iris-versicolor',
'sepal_length_cm'].hist()
;
''
```



Phew! Good thing we fixed those outliers. They could've really thrown our analysis off.

> We had to drop those rows with missing values.

Let's take a look at the rows with missing values:

```
iris_data.loc[(iris_data['sepal_length_cm'].isnull()) |
              (iris_data['sepal_width_cm'].isnull()) |
              (iris_data['petal_length_cm'].isnull()) |
              (iris_data['petal_width_cm'].isnull())]
```

```
    sepal_length_cm  sepal_width_cm  petal_length_cm
petal_width_cm  \
7               5.0             3.4              1.5             NaN
```

```
8               4.4             2.9             1.4             NaN

9               4.9             3.1             1.5             NaN

10              5.4             3.7             1.5             NaN

11              4.8             3.4             1.6             NaN
```

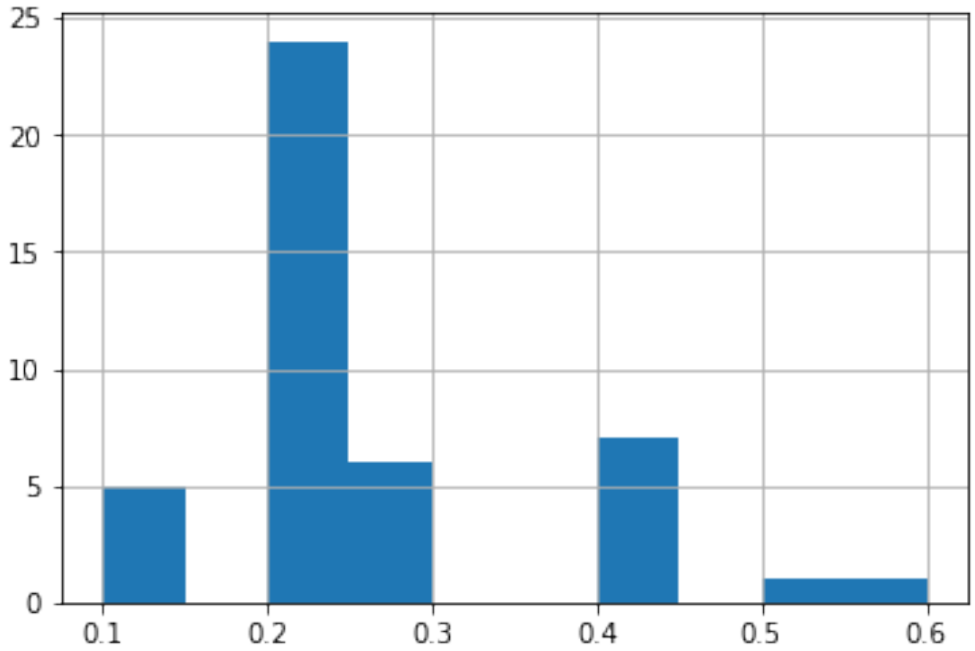
```
          class
7   Iris-setosa
8   Iris-setosa
9   Iris-setosa
10  Iris-setosa
11  Iris-setosa
```

It's not ideal that we had to drop those rows, especially considering they're all `Iris-setosa` entries. Since it seems like the missing data is systematic — all of the missing values are in the same column for the same *Iris* type — this error could potentially bias our analysis.

One way to deal with missing data is **mean imputation**: If we know that the values for a measurement fall in a certain range, we can fill in empty values with the average of that measurement.

Let's see if we can do that here.

```
iris_data.loc[iris_data['class'] == 'Iris-setosa',
'petal_width_cm'].hist()
;
''
```



Most of the petal widths for `Iris-setosa` fall within the 0.2-0.3 range, so let's fill in these entries with the average measured petal width.

```
average_petal_width = iris_data.loc[iris_data['class'] == 'Iris-
setosa', 'petal_width_cm'].mean()

iris_data.loc[(iris_data['class'] == 'Iris-setosa') &
              (iris_data['petal_width_cm'].isnull()),
              'petal_width_cm'] = average_petal_width

iris_data.loc[(iris_data['class'] == 'Iris-setosa') &
              (iris_data['petal_width_cm'] == average_petal_width)]
```

```
    sepal_length_cm  sepal_width_cm  petal_length_cm
petal_width_cm  \
7               5.0             3.4              1.5            0.25

8               4.4             2.9              1.4            0.25

9               4.9             3.1              1.5            0.25

10              5.4             3.7              1.5            0.25

11              4.8             3.4              1.6            0.25


          class
7   Iris-setosa
8   Iris-setosa
```

```
9   Iris-setosa
10  Iris-setosa
11  Iris-setosa
```

```
iris_data.loc[(iris_data['sepal_length_cm'].isnull()) |
              (iris_data['sepal_width_cm'].isnull()) |
              (iris_data['petal_length_cm'].isnull()) |
              (iris_data['petal_width_cm'].isnull())]
```

```
Empty DataFrame
Columns: [sepal_length_cm, sepal_width_cm, petal_length_cm,
petal_width_cm, class]
Index: []
```

Great! Now we've recovered those rows and no longer have missing data in our data set.

Note: If you don't feel comfortable imputing your data, you can drop all rows with missing data with the `dropna()` call:

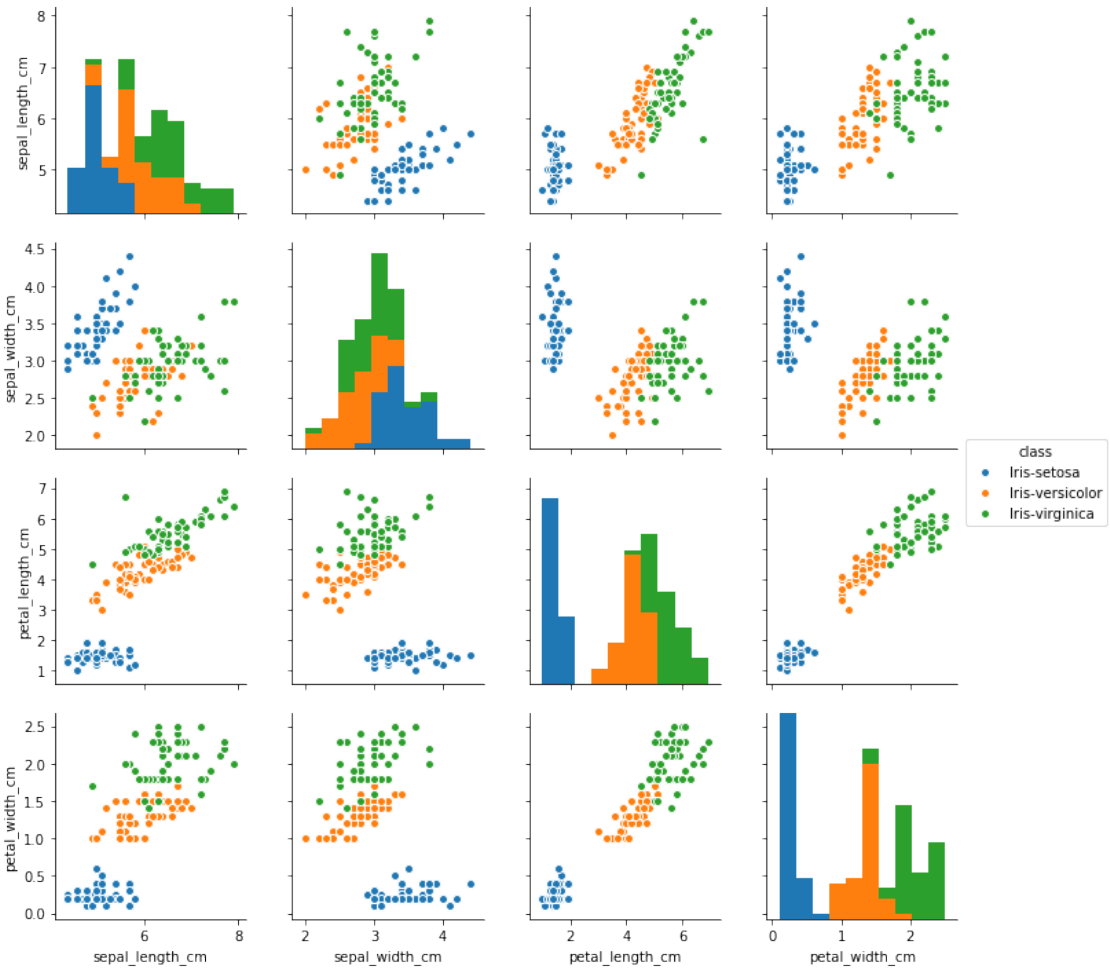
```
iris_data.dropna(inplace=True)
```

After all this hard work, we don't want to repeat this process every time we work with the data set. Let's save the tidied data file *as a separate file* and work directly with that data file from now on.

```
iris_data.to_csv('iris-data-clean.csv', index=False)

iris_data_clean = pd.read_csv('iris-data-clean.csv')
```

Now, let's take a look at the scatterplot matrix now that we've tidied the data.

```
sb.pairplot(iris_data_clean, hue='class')

''
```


sepal_length_cm
sepal_width_cm
petal_length_cm
petal_width_cm
class
Iris-setosa
Iris-versicolor
Iris-virginica