CAP5768_Assignment3_Corbin_Adam

October 14, 2019

1 CAP 5768 - Data Science - Adam Corbin - Fall 2019

1.1 Assignment 3: Statistical analysis - Part 1

1.1.1 Goals

- To transition from data analytics to basic statistical analysis.
- To practice the computation and displaying of summary statistics, percentiles, PMFs and (E)CDFs.
- To expand upon the prior experience of manipulating, summarizing, and visualizing small datasets.
- To display and interpret bee swarm plots and box-and-whisker plots.
- To visualize and compute pairwise correlations among variables in the dataset.

1.1.2 Instructions

- This assignment is structured as a single block, using the same dataset throughout.
- As usual, there will be some Python code to be written and questions to be answered.
- At the end, you should export your notebook to PDF format; it will "automagically" become your report.
- Submit the report (PDF), notebook (.ipynb file), and (optionally) link to the "live" version of your solution on Google Colaboratory via Canvas.
- The total number of points is 107 (plus up to 25 bonus points).

1.1.3 Important

- It is OK to attempt the bonus points, but please **do not overdo it!**

1.2 The Iris dataset

Iris.png

The Python code below will load a dataset containing information about three types of Iris flowers that had the size of its petals and sepals carefully measured.

The Fisher's Iris dataset contains 150 observations with 4 features each: - sepal length in cm; - sepal width in cm; - petal length in cm; and - petal width in cm.

The class for each instance is stored in a separate column called "species". In this case, the first 50 instances belong to class Setosa, the following 50 belong to class Versicolor and the last 50 belong to class Virginica.

See: https://archive.ics.uci.edu/ml/datasets/Iris for additional information.

[2]:
```
import numpy as np
from scipy.stats import poisson
import matplotlib.pyplot as plt
import seaborn as sns
iris = sns.load_dataset("iris")
iris.head()
```

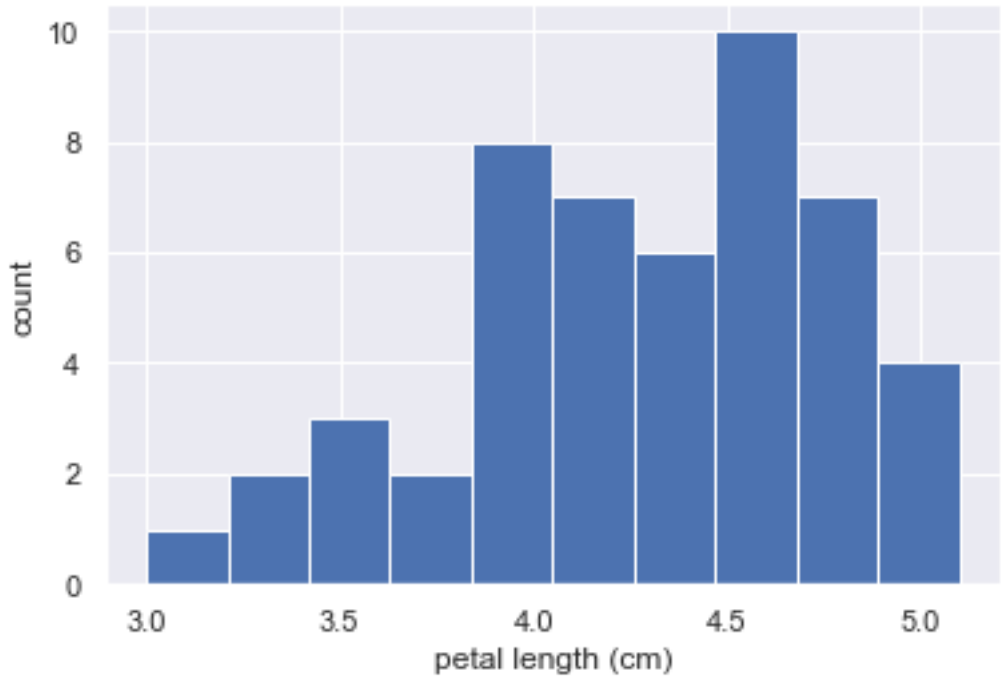
[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

1.3 Histogram and summary statistics

The code below can be used to display the histogram of versicolor petal lengths (with meaningful labels for the axes and default option for number of bins).

[3]:
```
# Set default Seaborn style
sns.set()

# Plot histogram of versicolor petal lengths
versicolor_petal_length = iris[iris.species == 'versicolor'].petal_length
plt.hist(versicolor_petal_length)

# Label axes
plt.xlabel('petal length (cm)')
plt.ylabel('count')

# Show histogram
plt.show()
```



1.4 Your turn! (12 points)

Write code to:

1. Modify the histogram above, this time using the "square root rule" for the number of bins. (4 pts)

The "square root rule" is a commonly-used rule of thumb for choosing number of bins: choose the number of bins to be the square root of the number of samples.

2. Modify the histogram above, such that the y axis shows probability/proportion (rather than absolute count), i.e., a proper PMF. (4 pts)

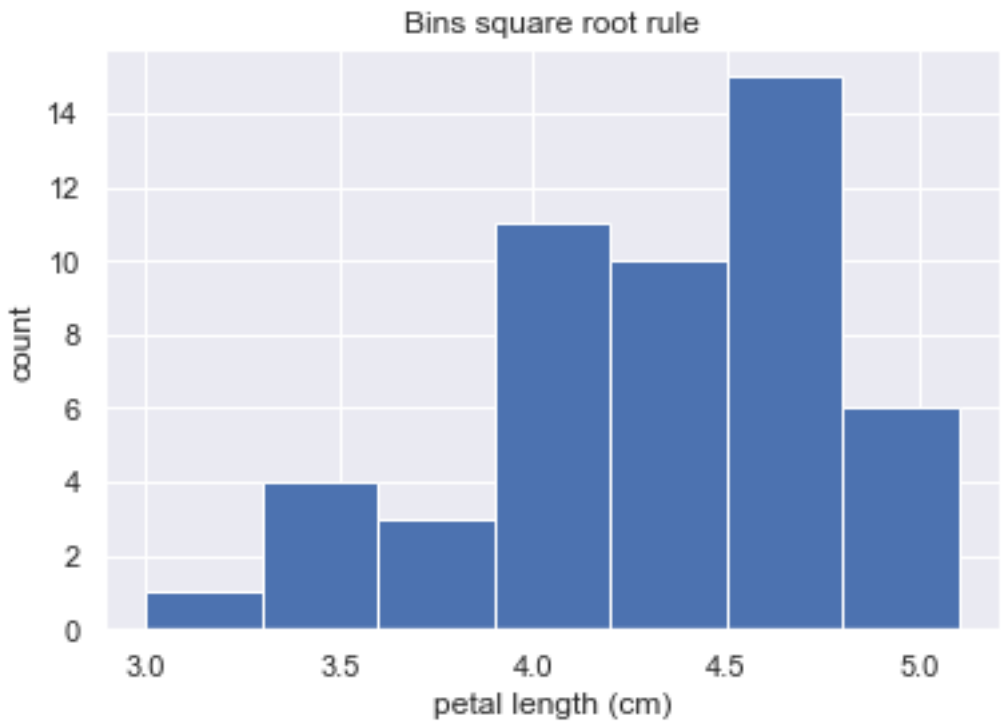
3. Compute summary statistics (2 pts each): mean and standard deviation

1.5 Solution

[4]:
```
print("#1")
n_bins= int(np.sqrt(len(versicolor_petal_length)))
plt.hist(versicolor_petal_length, bins=n_bins)

# Label axes
plt.xlabel('petal length (cm)')
plt.ylabel('count')
plt.title("Bins square root rule")
```

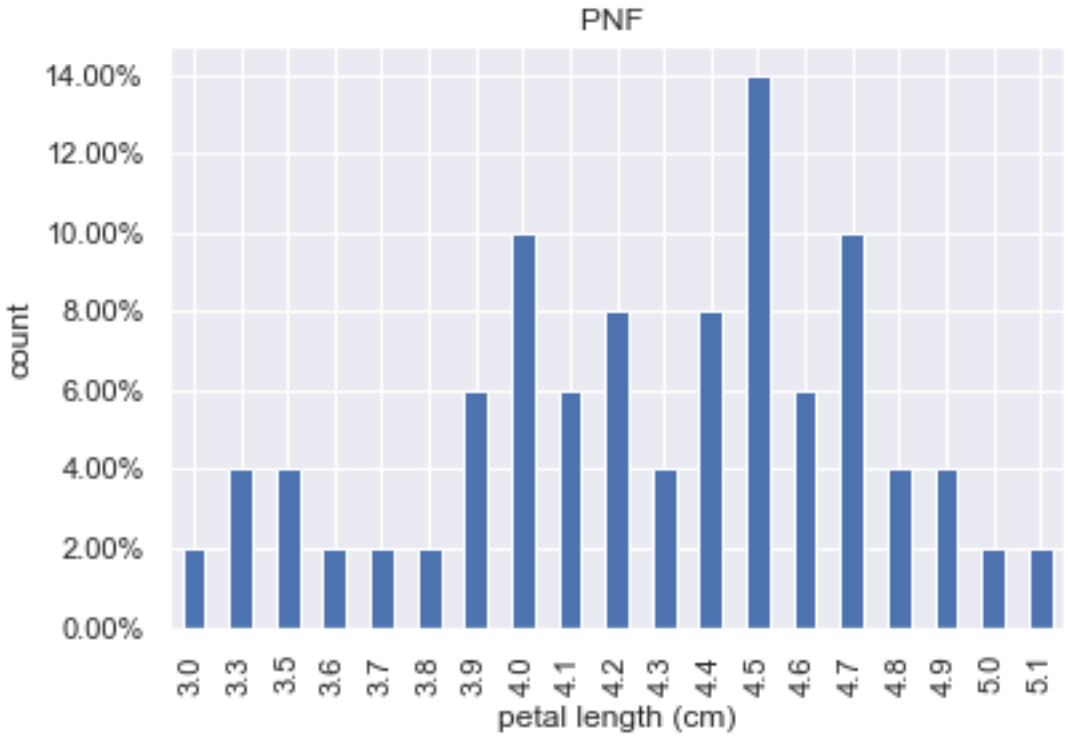
```
# Show histogram
plt.show()
```

#1



[5]:
```
print("#2")
import matplotlib.ticker as mtick
#Get total
# create dictionary where you have frequency/total
n = len(versicolor_petal_length)
pmf = versicolor_petal_length.value_counts()/n

ax = pmf.sort_index().plot(kind='bar')
# manipulate
vals = ax.get_yticks()
ax.set_yticklabels(['{:,.2%}'.format(x) for x in vals])


# Label axes
plt.xlabel('petal length (cm)')
plt.ylabel('count')
plt.title("PNF ")
# Show histogram
plt.show()
```

#2



[26]:
```
print("#3")
mean = versicolor_petal_length.mean()
std = versicolor_petal_length.std()
print("Mean:", mean)
print("Standard Deviation:", round(std,5))
versicolor_petal_length.describe()
```

```
#3
Mean: 4.26
Standard Deviation: 0.46991
```

[26]:
```
count    50.000000
mean      4.260000
std       0.469911
min       3.000000
25%       4.000000
50%       4.350000
75%       4.600000
max       5.100000
Name: petal_length, dtype: float64
```

1.6 Your turn! (6 points)

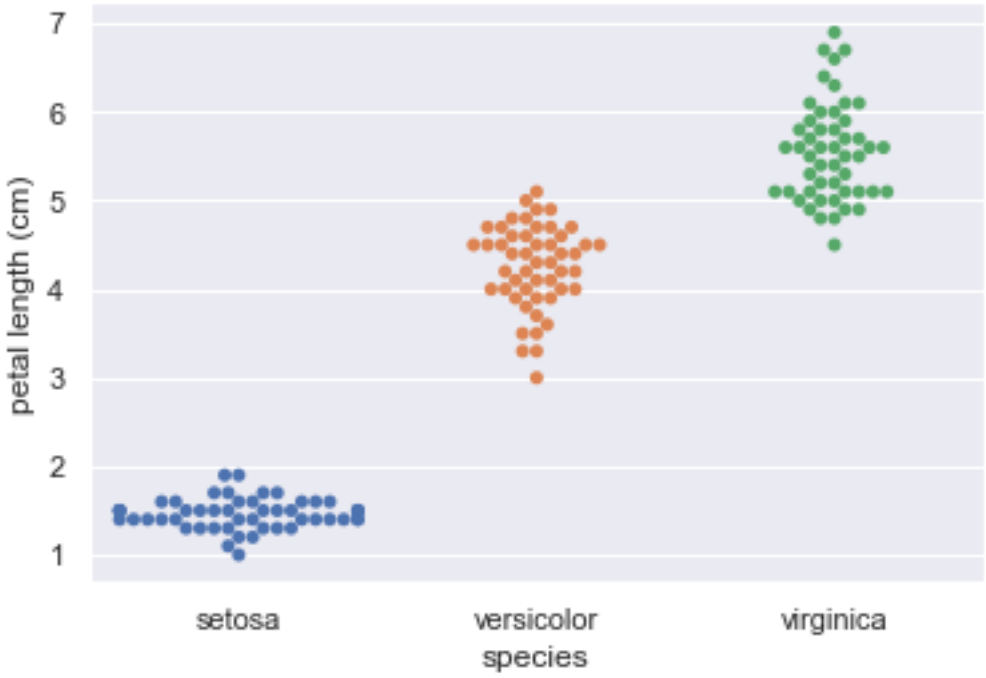
Make a bee swarm plot of the iris petal lengths. Your x-axis should contain each of the three species (properly labeld), and the y-axis the petal lengths.

Your plot should look like this:

iris_beeswarm.png

1.7 Solution

[57]:
```
sns.swarmplot(x='species', y='petal_length', data=iris)
plt.xlabel('species')
plt.ylabel('petal length (cm)')
plt.show()
```




```
<Figure size 864x720 with 0 Axes>
```

1.8 Questions 1-2 (12 points, i.e. 6 pts each)

1. Explain the "binning bias" associated with histogram plots.
2. What is a bee swarm plot and in which situations should you (not) use it?

1.9 Solution

1. Binning bias is selecting the number of bins, the histogram can look dramatically different and could change a humans perception. This could mean telling a different story depending on the level of granularity was chosen for the bins.
2. A bee swarm plot is used to show the density and actual values of the data set which can be a little more optimal than a histogram. It also can show more than 1 type flower of the dataset. It should not be used for large data sets with many values as the graph will become very large and unreadable.

1.10 Empirical Cumulative Distribution Function (ECDF)

The function below takes as input a 1D array of data and then returns the x and y values of the ECDF.

[7]:
```
def ecdf(data):
    """Compute ECDF for a one-dimensional array of measurements."""
    # Number of data points: n
    n = len(data)

    # x-data for the ECDF: x
    x = np.sort(data)

    # y-data for the ECDF: y
    y = np.arange(1, n+1) / n

    return x, y
```

1.11 Your turn! (12 points)

Use the `ecdf()` function above to compute the ECDF for the petal lengths of the Iris versicolor flowers (6 pts) and plot the resulting ECDF (6 pts).

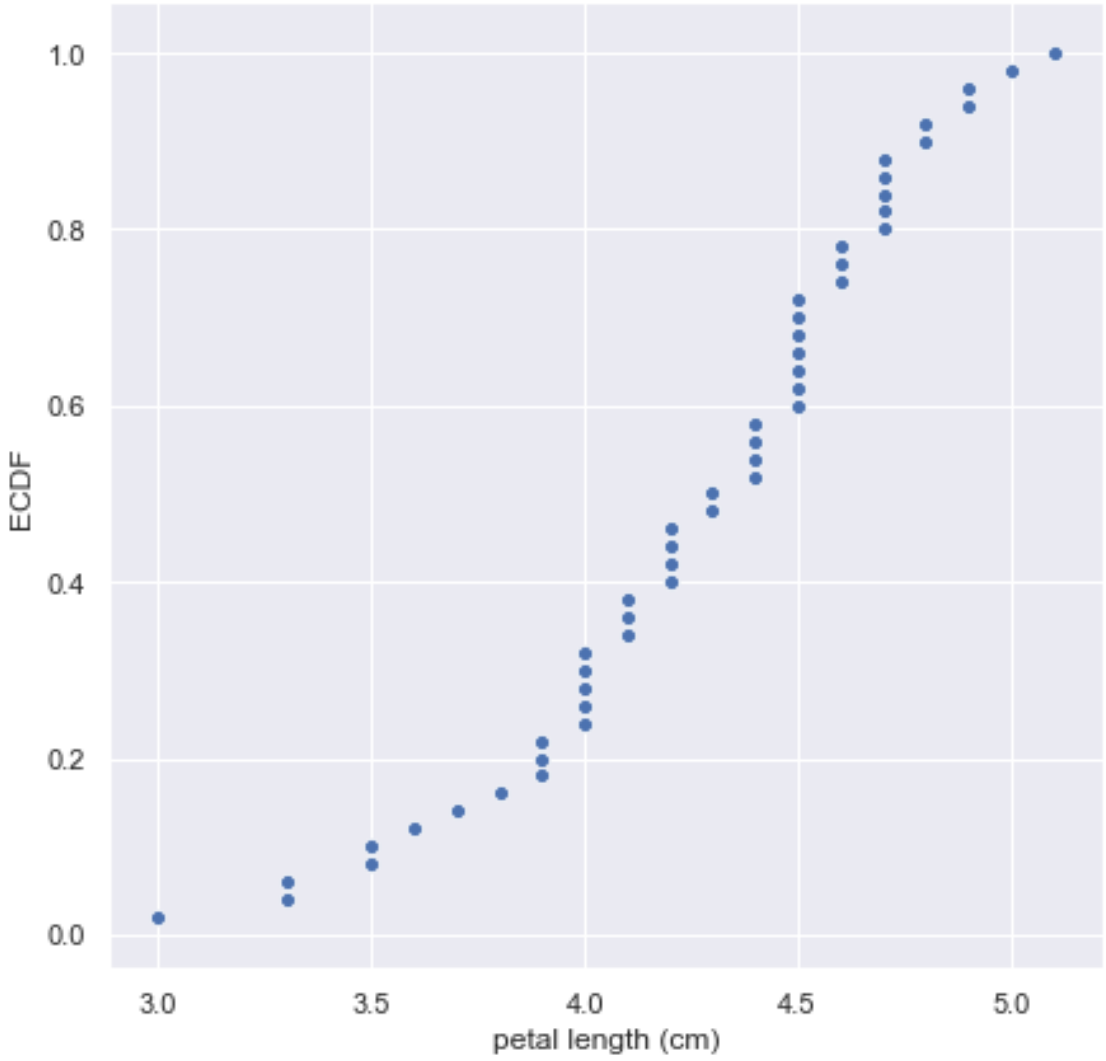
Your plot should look like this:

ecdf_versicolorpng.png

1.12 Solution

[8]:
```
ecdf_versicolor = ecdf(versicolor_petal_length)
x = ecdf_versicolor[0]
y = ecdf_versicolor[1]
plt.figure(figsize=(7,7))
ax = sns.scatterplot(x=x, y=y)
ax.set_ylabel("ECDF")
ax.set_xlabel("petal length (cm)")
ax
```

[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1783ee8df48>
```



1.13 Your turn! (8 points)

Write code to plot ECDFs for the petal lengths of all three iris species.

Your plot should look like this:

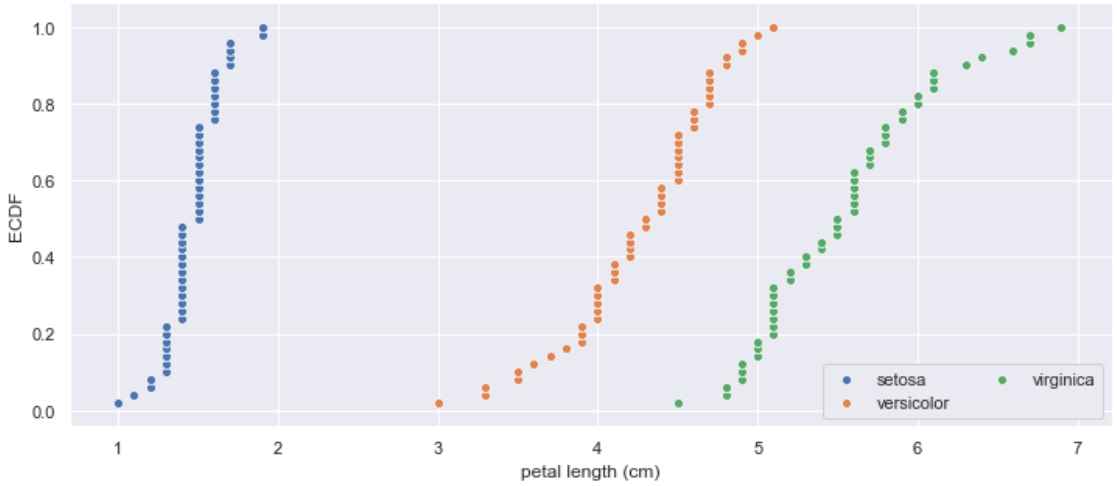
ECDFs_Iris.png

1.14 Solution

[9]:
```
setosa_petal_length = iris[iris.species == 'setosa'].petal_length
virginica_petal_length = iris[iris.species == 'virginica'].petal_length
def generate_ecdf_graphs():

```

```
    plt.figure(figsize=(12,5))
    _ecdf = ecdf(setosa_petal_length)
    x = _ecdf[0]
    y = _ecdf[1]
    ax = sns.scatterplot(x=x, y=y)

    ecdf_versicolor = ecdf(versicolor_petal_length)
    x = ecdf_versicolor[0]
    y = ecdf_versicolor[1]

    ax = sns.scatterplot(x=x, y=y)

    _ecdf = ecdf(virginica_petal_length)
    x = _ecdf[0]
    y = _ecdf[1]
    ax = sns.scatterplot(x=x, y=y)


    ax.legend(labels=['setosa', 'versicolor',"virginica"],ncol=2)
    ax.set_ylabel("ECDF")
    ax.set_xlabel("petal length (cm)")
    return ax
ax = generate_ecdf_graphs()
ax
```

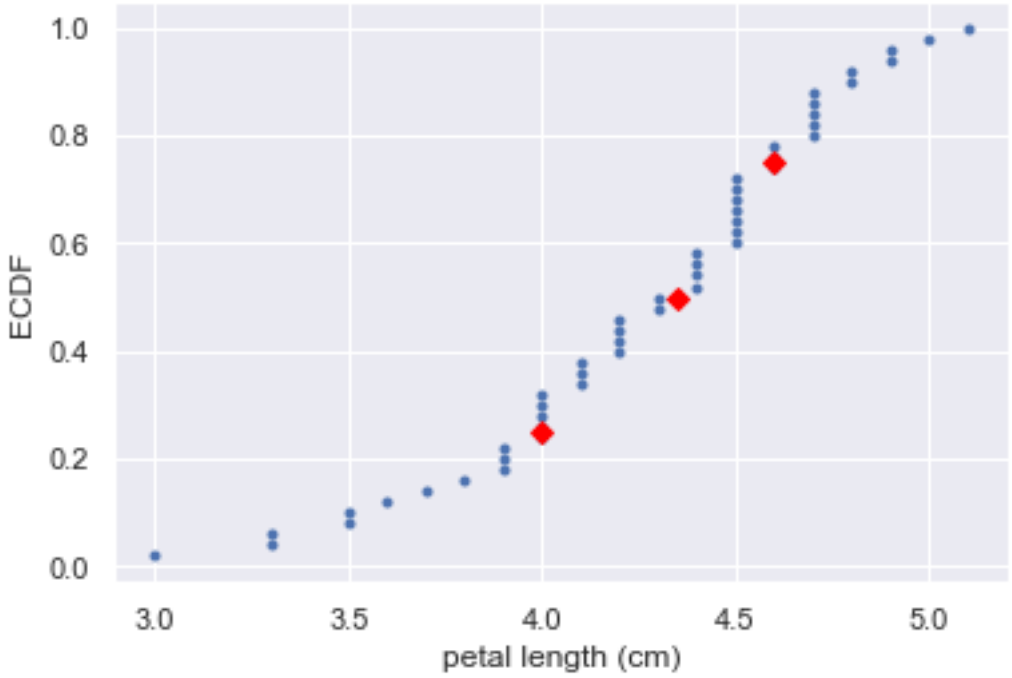
[9]: `<matplotlib.axes._subplots.AxesSubplot at 0x1783f0535c8>`



1.15 Percentiles

The code below computes the 25th, 50th, and 75th percentiles for the petal lengths of the Iris versicolor species and overlays the results on top of the ECDF.

[84]:
```
# Specify array of percentiles: percentiles
percentiles = np.array([25, 50, 75])

# Compute percentiles
ptiles_versicolor = np.percentile(versicolor_petal_length, percentiles)

# Compute ECDF
x_vers, y_vers = ecdf(versicolor_petal_length)
# Plot the ECDF
_ = plt.plot(x_vers, y_vers, '.')
_ = plt.xlabel('petal length (cm)')
_ = plt.ylabel('ECDF')

# Overlay percentiles as red diamonds.
_ = plt.plot(ptiles_versicolor, percentiles/100, marker='D', color='red',
         linestyle='none')

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



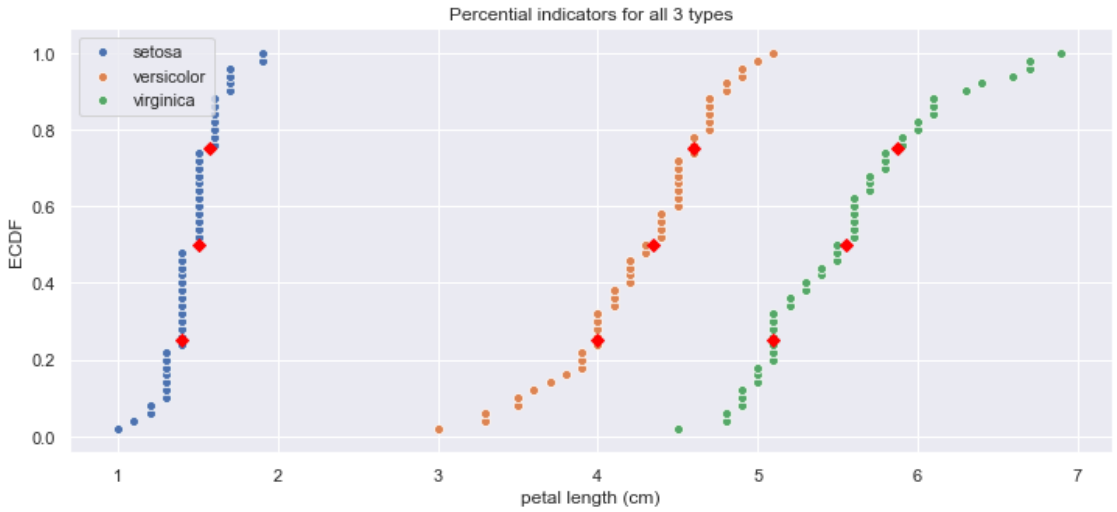
1.16 Your turn! (10 points)

Write code to compute the 25th, 50th, and 75th percentiles for the petal lengths of and plot the resulting values overlaid with the corresponding ECDFs for all three iris species.

1.17 Solution

[103]:
```
ax = generate_ecdf_graphs()
ptiles_versicolor = np.percentile(versicolor_petal_length, percentiles)
ptiles_setosa = np.percentile(setosa_petal_length, percentiles)
ptiles_virginica = np.percentile(virginica_petal_length, percentiles)
_ = plt.plot(ptiles_versicolor, percentiles/100, marker='D', color='red',
         linestyle='none')
_ = plt.plot(ptiles_setosa, percentiles/100, marker='D', color='red',
         linestyle='none')
_ = plt.plot(ptiles_virginica, percentiles/100, marker='D', color='red',
         linestyle='none')
plt.title("Percential indicators for all 3 types")
plt
```

[103]:
```
<module 'matplotlib.pyplot' from
'C:\\Users\\C0rbin\\Anaconda3\\envs\\assignment_1\\lib\\site-
packages\\matplotlib\\pyplot.py'>
```



1.18 Box-and-whisker plots

Box-and-whisker plots (or simply box plots) show the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be "outliers" using a method that is a function of the inter-quartile range.

Box_plot.png

1.19 Your turn! (10 points)

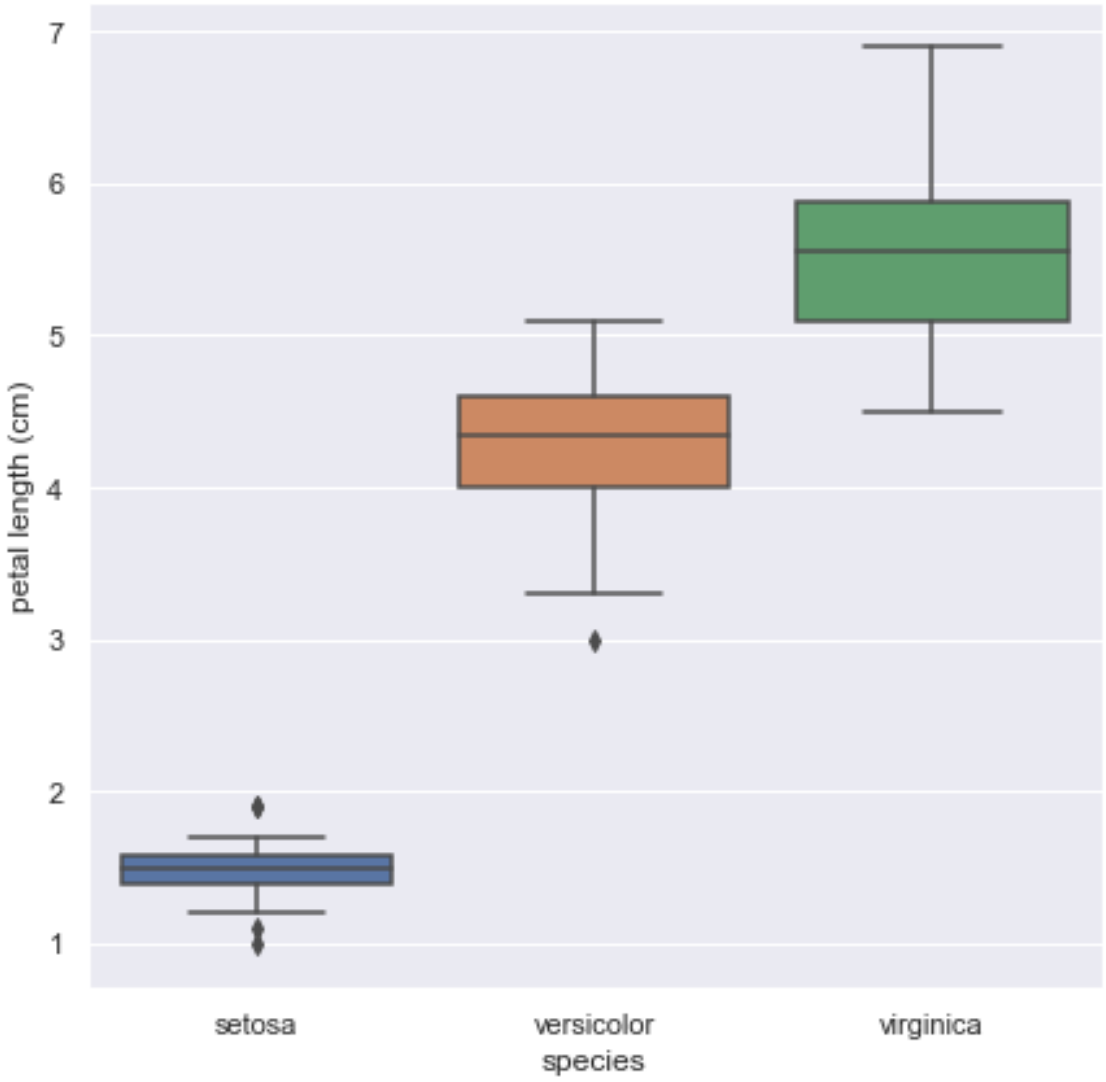
Write code to display the box-and-whisker plot for the petal lengths of all three iris species.
Your plot should look like this:

Iris_boxplot.png

1.20 Solution

[10]:
```
plt.figure(figsize=(7,7))
sns.boxplot(x="species", y="petal_length", data=iris)
plt.ylabel('petal length (cm)')
plt
```

[10]:
```
<module 'matplotlib.pyplot' from
'C:\\Users\\C0rbin\\Anaconda3\\envs\\assignment_1\\lib\\site-
packages\\matplotlib\\pyplot.py'>
```



1.21 Questions 3-4 (6 points, i.e. 3 pts each)

3. Which species has the largest/smallest standard deviation?
4. Which species has the largest/smallest number of outliers?

1.22 Solution

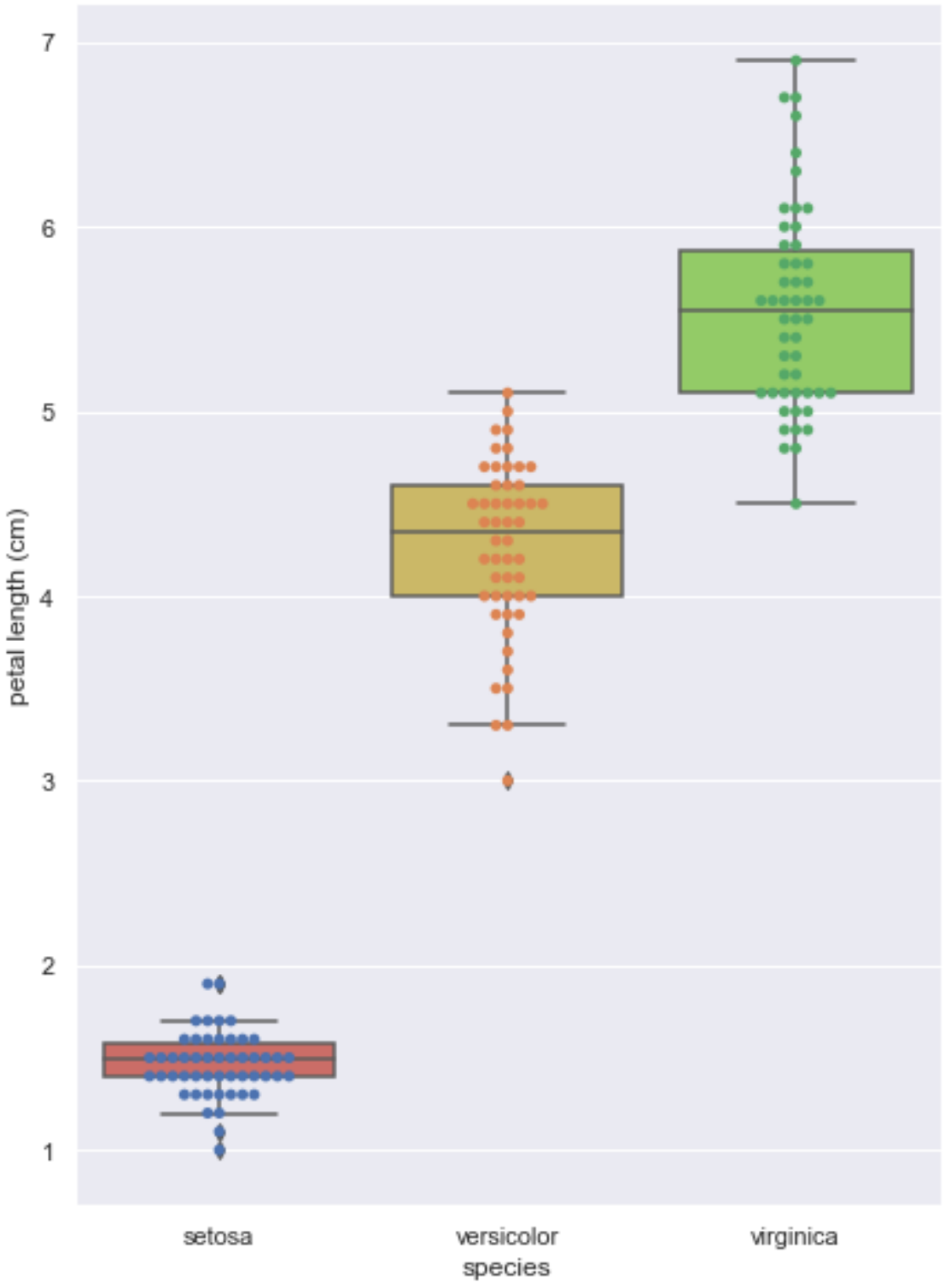
3. Setosa has the smalles standard deviation since the between the wiskers are the smallest. Virginica has the largest since it has the length between the wiskers
4. Setosa appears to have at least 3 outliers with the most and Virginica has the least with zero outliers.

1.23 Bonus! (10 points)

Write code to display the box-and-whisker plot combined with the bee swarm plot for the petal lengths of all three iris species.

1.24 Solution

[125]:
```
plt.figure(figsize=(7,10))
sns.swarmplot(x='species', y='petal_length', data=iris)
sns.boxplot(x="species", y="petal_length", data=iris, palette=sns.
 ↪color_palette("hls", 8))
plt.ylabel('petal length (cm)')
plt.show()
```


7
6
5
4
3
2
1
petal length (cm)
setosa
versicolor
virginica
species

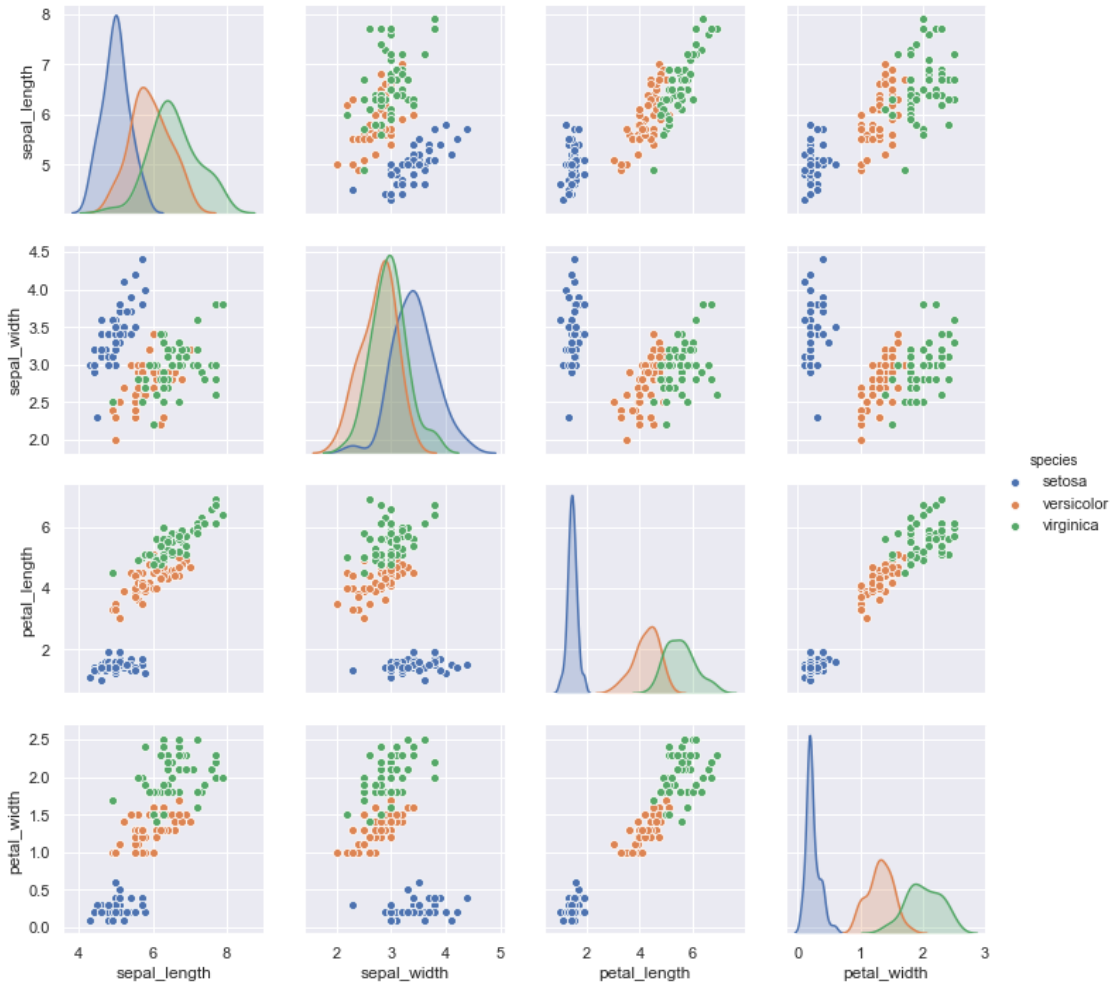
1.25 Scatter plots, pair plots, and correlation between two variables

The code below:

1. Displays the pair plots for all (4) attributes for all (3) categories/species/classes in the Iris dataset.
2. Computes the covariance matrix for the versicolor species.
3. Computes the Pearson correlation coefficient between petal length and petal width for the versicolor species.

[126]:
```
# Display pair plot
sns.pairplot(iris, hue='species', height=2.5);

# Compute 1D arrays for petal length and width
versicolor_petal_width = iris[iris.species == 'versicolor'].petal_width
versicolor_petal_length = iris[iris.species == 'versicolor'].petal_length

def pearson_r(x, y):
    """Compute Pearson correlation coefficient between two arrays."""
    # Compute correlation matrix: corr_mat
    corr_mat = np.corrcoef(x, y)

    # Return entry [0,1]
    return corr_mat[0,1]

# Compute Pearson correlation coefficient for I. versicolor: r
r = pearson_r(versicolor_petal_length, versicolor_petal_width)
print('Pearson correlation coefficient between petal length and petal width for␣
 ↪versicolor species: {:.5f}'.format(r))
```

```
Pearson correlation coefficient between petal length and petal width for
versicolor species: 0.78667
```



1.26 Bonus! (15 points)

Extend the code above to compute the Pearson correlation coeficients for all pair-wise combinations of all three Iris species and display the results in a table format.

1.27 Solution

[163]:
```
from itertools import combinations
# p_table = iris.pivot_table(index = "species",columns='petal_length')
# p_table.head()
group_table = iris.groupby("species")


# group_table.head()
columns = list(iris.columns)
columns.remove("species")
```

```
combos = list(combinations(columns,2))
list_of_species = list(iris["species"].unique())
list_of_species
print("Pearson Correlation coeficients for all pair-wise combinations")
print("Species\t\t|Col1\t\t|Col2\t\t|Pearson Coefficient")
for species in list_of_species:
    for combo in combos:
        attribute_1 = iris[iris["species"] == species][combo[0]]
        attribute_2 = iris[iris["species"] == species][combo[1]]
        r = pearson_r(attribute_1,attribute_2)
        if(species == "setosa"):
            update_species = species + "\t"
            print(update_species,combo[0],combo[1], r, sep="\t|")
        else:
            print(species,combo[0],combo[1], r, sep="\t|")
print("\nCovariance table")
group_table.cov()
```

```
Species         |Col1           |Col2           |Pearson Coefficient
setosa          |sepal_length   |sepal_width    |0.7425466856651597
setosa          |sepal_length   |petal_length   |0.2671757588687571
setosa          |sepal_length   |petal_width    |0.2780983529359696
setosa          |sepal_width    |petal_length   |0.17769996678227068
setosa          |sepal_width    |petal_width    |0.2327520113628792
setosa          |petal_length   |petal_width    |0.33163004080411856
versicolor      |sepal_length   |sepal_width    |0.5259107172828243
versicolor      |sepal_length   |petal_length   |0.754048958592016
versicolor      |sepal_length   |petal_width    |0.5464610715986299
versicolor      |sepal_width    |petal_length   |0.560522091692982
versicolor      |sepal_width    |petal_width    |0.6639987200241115
versicolor      |petal_length   |petal_width    |0.7866680885228169
virginica       |sepal_length   |sepal_width    |0.45722781639411303
virginica       |sepal_length   |petal_length   |0.8642247329355761
virginica       |sepal_length   |petal_width    |0.28110770915731925
virginica       |sepal_width    |petal_length   |0.4010445773427853
virginica       |sepal_width    |petal_width    |0.5377280262661887
virginica       |petal_length   |petal_width    |0.3221082159003183

Covariance table
```

[163]:

species		sepal_length	sepal_width	petal_length	petal_width
setosa	sepal_length	0.124249	0.099216	0.016355	0.010331
	sepal_width	0.099216	0.143690	0.011698	0.009298
	petal_length	0.016355	0.011698	0.030159	0.006069
	petal_width	0.010331	0.009298	0.006069	0.011106
versicolor	sepal_length	0.266433	0.085184	0.182898	0.055780

	sepal_width	0.085184	0.098469	0.082653	0.041204
	petal_length	0.182898	0.082653	0.220816	0.073102
	petal_width	0.055780	0.041204	0.073102	0.039106
virginica	sepal_length	0.404343	0.093763	0.303290	0.049094
	sepal_width	0.093763	0.104004	0.071380	0.047629
	petal_length	0.303290	0.071380	0.304588	0.048824
	petal_width	0.049094	0.047629	0.048824	0.075433

1.28 Question 5 (6 points)

5. Should the Pearson correlation coefficient be replaced with the Spearman rank-order correlation coefficient in this case? Why (not)?

1.29 Solution

The Spearman rank-order takes the index of a sorted list of values. If we have a natural distribution then this might not be as helpful. Below we have printing both spearman nad pearson correlations and they dont appear to be too different.

[183]:
```
print(group_table.corr(method='spearman'))
print(group_table.corr(method='pearson'))
```

		sepal_length	sepal_width	petal_length	petal_width
species					
setosa	sepal_length	1.000000	0.755337	0.278885	0.299499
	sepal_width	0.755337	1.000000	0.179911	0.286536
	petal_length	0.278885	0.179911	1.000000	0.271141
	petal_width	0.299499	0.286536	0.271141	1.000000
versicolor	sepal_length	1.000000	0.517606	0.736625	0.548679
	sepal_width	0.517606	1.000000	0.574727	0.659983
	petal_length	0.736625	0.574727	1.000000	0.787010
	petal_width	0.548679	0.659983	0.787010	1.000000
virginica	sepal_length	1.000000	0.426517	0.824323	0.315772
	sepal_width	0.426517	1.000000	0.387359	0.544310
	petal_length	0.824323	0.387359	1.000000	0.362913
	petal_width	0.315772	0.544310	0.362913	1.000000
		sepal_length	sepal_width	petal_length	petal_width
species					
setosa	sepal_length	1.000000	0.742547	0.267176	0.278098
	sepal_width	0.742547	1.000000	0.177700	0.232752
	petal_length	0.267176	0.177700	1.000000	0.331630
	petal_width	0.278098	0.232752	0.331630	1.000000
versicolor	sepal_length	1.000000	0.525911	0.754049	0.546461
	sepal_width	0.525911	1.000000	0.560522	0.663999
	petal_length	0.754049	0.560522	1.000000	0.786668
	petal_width	0.546461	0.663999	0.786668	1.000000
virginica	sepal_length	1.000000	0.457228	0.864225	0.281108
	sepal_width	0.457228	1.000000	0.401045	0.537728

```
petal_length      0.864225      0.401045      1.000000      0.322108
petal_width       0.281108      0.537728      0.322108      1.000000
```

1.30 Conclusions (25 points)

Write your conclusions and make sure to address the issues below: - What have you learned from this assignment? - Which parts were the most fun, time-consuming, enlightening, tedious? - What would you do if you had an additional week to work on this?

1.31 Solution

1. A better understanding of histograms and how the bee swarm plots can help show the values when the data sets are smaller. How the box and wisker plots can be useful when looking at the percentiles. It was really fun calculating the percentiles too!
2. The most fun part of this assignment was working displaying different types of data in the different graphs. Its great getting use to these libraries. Google is still my friend as i dont have everything in my mind that I want to do. Creating the legends was a little tricky on the percentile graphs but I got that working.
3. Probably look into how to programmatically find the outlines on the wisker and box plots. This assignment was a little on the lighter side but it was still a good assignment.