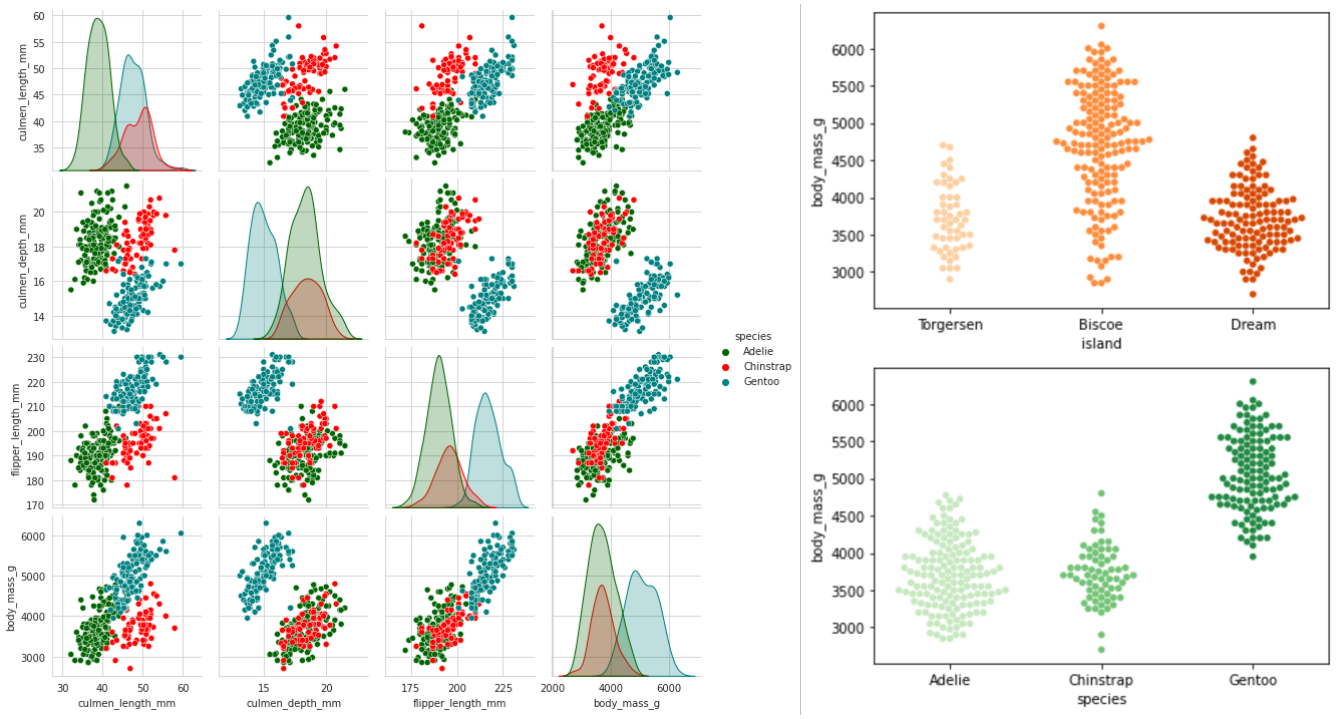
Müge Kuşkon · Follow
Jan 27 · 8 min read · Listen

How to Perform Exploratory Data Analysis?

A Quick Guide to EDA.



Exploratory data analysis or **EDA** is a crucial step which leads to fathom the dataset. Various kind of steps can be conducted in EDA. I will take 4 main steps into consideration and an implementation of these steps will be shown on the **Palmer Archipelago (Antarctica) Penguin Data**.

1. Scrutinize the data

The reason of this step is to figure out the variables and shape of the dataset. It answers questions such as "Is this dat "How many features or rows does it contain?" etc. After loading the dataset, checking the first five rows with the **head()** function would be a nice start to understand the structure of the dataset as seen below.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

#Loading the dataset
penguins_size = pd.read_csv('penguins_size.csv', sep = ",")
penguins_size.head()
```

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN

```
Shape is:  (344, 7)
```

From here, it is understood that the dataset's shape is **(344, 7)** meaning that 7 features and 344 rows are present which emphasizes that the dataset is not large enough. The seven features can be listed as species, island, depth of the culmen, length of the culmen, length of the flipper, body mass and sex.

In order to visualize the data types of the features, **info()** function can be used as seen below. The results let us understand that species, island and sex are objects and remaining features are float variables. The use of **dtypes** is also an alternative to learn the data types of the columns.

```
penguins_size.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   species            344 non-null    object
 1   island             344 non-null    object
 2   culmen_length_mm   342 non-null    float64
 3   culmen_depth_mm    342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                334 non-null    object
```

```
penguins_size.dtypes
```

```
species              object
island               object
culmen_length_mm    float64
culmen_depth_mm     float64
flipper_length_mm   float64
body_mass_g         float64
sex                  object
dtype: object
```

2. Data Cleaning

Finding missing values, removing duplicates etc. are crucial step in exploratory data analysis. These values could lead our models to draw incorrect conclusions at the end. Investigating only **isnull()** is not enough. For instance, in a dataset containing a feature of heart rate, the value of that feature cannot be 0. In this case 0 is also a missing value and needs to be dealt with.

There are various ways to deal with the missing values of the data such as **deleting the rows containing missing values** (if the dataset is large enough and the number of missing values is not too many, that could be an option), **imputation methods** (mean/median of the feature) etc.

```
penguins_size.isnull().sum()
```

```
culmen_length_mm      2
culmen_depth_mm       2
flipper_length_mm     2
body_mass_g           2
sex                  10
dtype: int64
```

As seen above, all features except island and species contain missing values in this dataset. I've chosen to impute the missing values of the float features with the mean of the corresponding feature since the dataset is quite small.

```
penguins_size.value_counts(["sex"])
```

```
sex
MALE      168
FEMALE    165
.           1
dtype: int64
```

```
penguins_size['sex'] = penguins_size['sex'].fillna('MALE')
```

For the sex of the penguin, after checking the count of the values for female and male, the most frequent value will be taken into consideration, in this case missing values will be imputed with **"MALE"**. As seen above, another value as "." is seen which has to be imputed or dropped. I preferred to drop it since it contains only one row, but making equal to "MALE" could've been another solution. The index of the problematic row is 336. Finally, after all the missing values have been imputed or dropped, we check again with the **isna()** function and it is understood that no missing values are left.

```
penguins_size.drop(axis = 0, inplace = True, index = 336)
penguins_size.isna().sum()
```

```
species              0
island               0
culmen_length_mm     0
culmen_depth_mm      0
flipper_length_mm    0
body_mass_g          0
sex                  0
dtype: int64
```

Lastly for this section, the presence of any duplicate rows is checked and in this dataset none was found.

```
duplicated = penguins_size.duplicated()
print(duplicated.sum())
```

3. Statistical insights

This section is also a part of understanding the data. After handling with the missing values, the **describe()** function can be used in order to grasp information such as the mean, maximum, minimum and standard deviation of the data. This method can also be useful to detect missing values such as if the minimum of a feature value is 0 where it shouldn't be, the describe function facilitates the process of handling missing values if any of them are left.

count	343.000000	343.000000	343.000000	343.000000
mean	43.920244	17.155400	200.868310	4199.791571
std	5.451506	1.970337	14.014098	799.950869
min	32.100000	13.100000	172.000000	2700.000000
25%	39.250000	15.600000	190.000000	3550.000000
50%	44.100000	17.300000	197.000000	4050.000000
75%	48.500000	18.700000	213.000000	4750.000000
max	59.600000	21.500000	231.000000	6300.000000

By the use of **value_counts()** function, the count of unique values for the objects is done. In this case, the number of rows belonging to each species are calculated as 152, 123 and 68 meaning that Adélie penguins are the ones who dominate the dataset. Moreover, the mean of body mass for each species can be found by using **groupby().** For continuous features, this function is useful in terms of splitting the data in categories (in this example species) and observing it better.

```
penguins_size['species'].value_counts()
```

```
Adelie       152
Gentoo       123
Chinstrap     68
Name: species, dtype: int64
```

```
# Find body mass mean for each species.
mean_bodymass = penguins_size.groupby('species')['body_mass_g'].mean()
mean_bodymass
```

```
species
Adelie       3703.958910
Chinstrap    3733.088235
Gentoo       5070.542719
Name: body_mass_g, dtype: float64
```

4. Data visualization

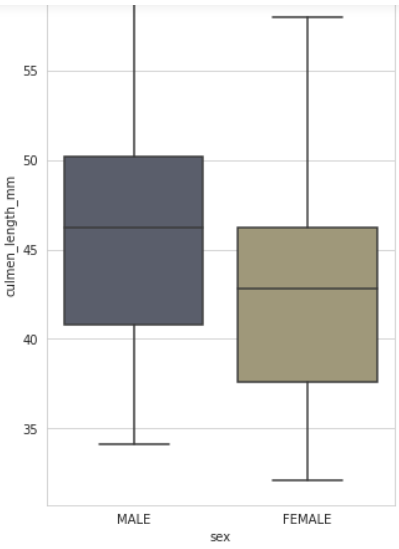
Various plotting techniques can be used in order to better visualize the dataset. In this section only few of these techniques will be explained and shown. Some plots are better in visualizing categorical data and some of them are more suitable for numerical data.

Box Plot

These plots are a good way to check the outliers or understand the relationship between a categorical and continuous feature by showing the distribution of data.

As it can be seen below, no outliers is detected since no data point is seen above or below the maximum and minimum respectively. Furthermore, the median of data points can be easily found since the horizontal line passing inside of the box represents it.

```
#Relationship of the culmen length and sex of the penguins.

fig = plt.figure(figsize=(5,8))
ax= sns.boxplot(x = penguins_size.sex, y=penguins_size['culmen_length_mm'],orient="v", palette = "cividis")
plt.title('Culmen length mm')
```

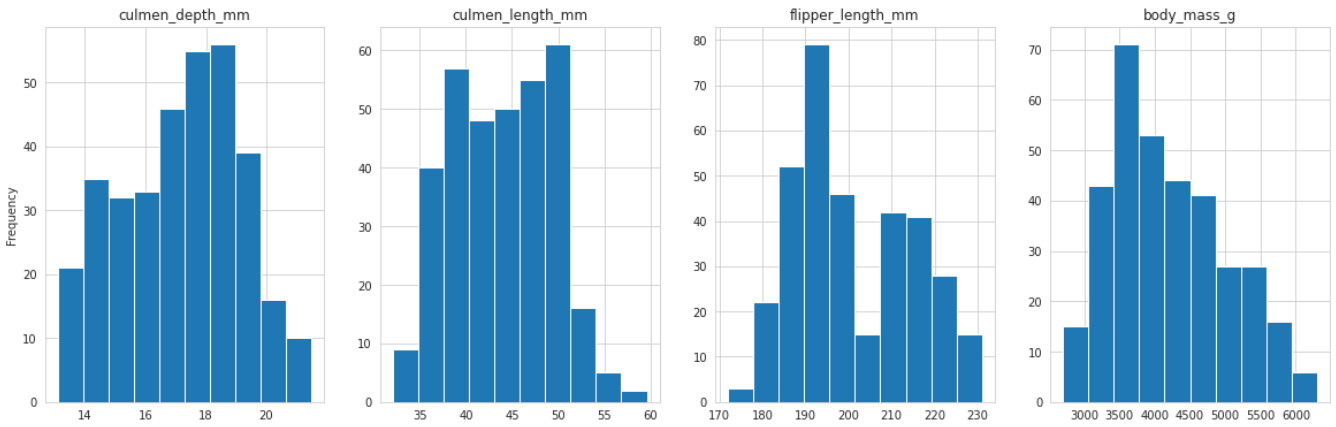


Histogram

Histograms are used to depict the frequency distribution. It can be only used with **numerical data**.

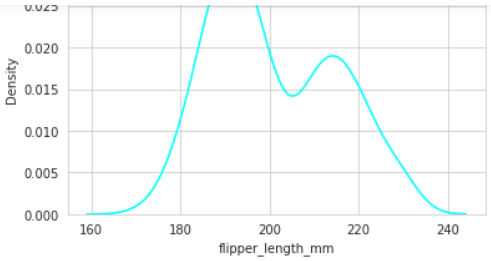
Below, histograms which emphasize the frequencies of depth and length of the culmen, length of flipper and body mass since these features contain numerical data.

```
#Shows us frequency distribution.
fig,axs = plt.subplots(1,4,figsize=(20,6))
axs[0].hist(penguins_size.culmen_depth_mm)
axs[0].set_title('culmen_depth_mm')
axs[0].set_ylabel('Frequency')
axs[1].hist(penguins_size.culmen_length_mm)
axs[1].set_title('culmen_length_mm')
axs[2].hist(penguins_size.flipper_length_mm)
axs[2].set_title('flipper_length_mm')
axs[3].hist(penguins_size.body_mass_g)
axs[3].set_title('body_mass_g')

plt.show()
```



Moreover, **kdeplot** is another way to visualize the distribution of the data. This plot is really similar to histograms but instead of putting the values into bins, it draws a curve. This is smoother than histograms which can lose little information.

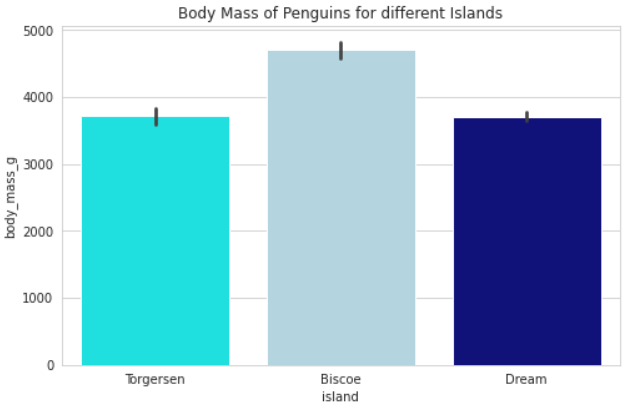
```
#Used for visualizing the probability density of a continuous var.

sns.kdeplot(penguins_size.flipper_length_mm,color='Cyan')
plt.show()
```



Bar Plot

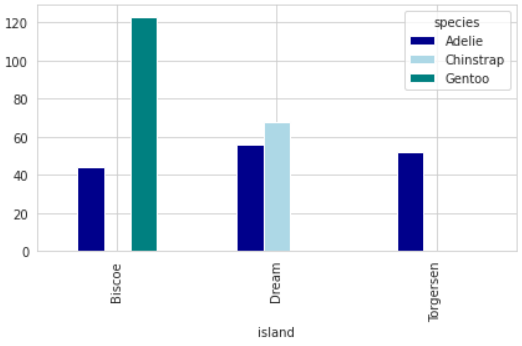
In a bar plot, the x-axis represents a categorical variable while the y-axis is a numerical variable. That is why the bar plot depicts a relationship between these two variables. For instance, below the body masses of the penguins for each islands are seen. The categorical data in the x-axis is the islands whereas the numerical data in y-axis is the body mass of the penguin.

```
plt.figure(figsize=(8,5))
colors = ["cyan","lightblue", "darkblue"]
sns.barplot(x =penguins_size['island'],
y = penguins_size['body_mass_g'], palette = colors)
plt.title('Body Mass of Penguins for different Islands')
plt.show()
```



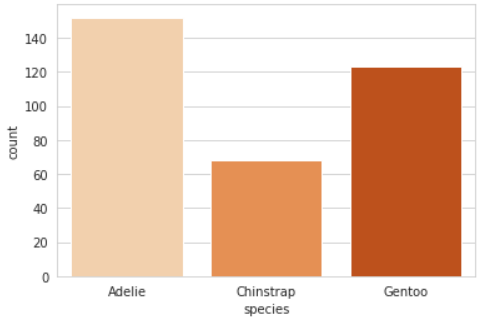
By the use of pandas function **crosstab**, the relationship between two or more variables can be analyzed. As an illustration, the bar plot below underlines the relationship between the number of penguins of specific species living in a particular island. It is seen that in the Torgersen island approximately 50 Adelie penguins live.

```
pd.crosstab(penguins_size['island'], penguins_size['species']).plot.bar(color=('DarkBlue', 'LightBlue',
'Teal'))
plt.tight_layout()
```



Count Plot

```
sns.countplot('species',data=penguins_size, palette = "Oranges")
plt.show()
```

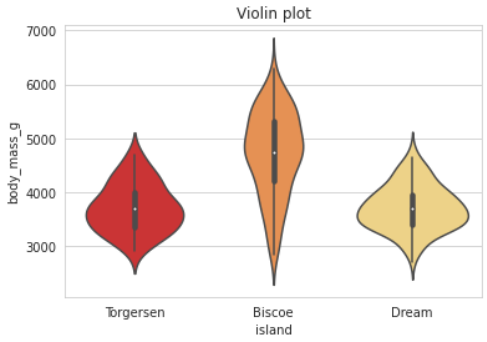


Violin Plot

Violin plots have common properties with box plots and used when the objective is to observe the distribution of numerical data for different categories. Its difference from a box plot is that it depicts the probability density of the dataset. It gives more insights than a box plot, because two different categories might have the same mean but it doesn't mean that they are the same. Their distributions might differ and in that case, violin plots would be more useful to observe.

In the violin plot below, the mean of the body mass of the penguins are clustered between 3000 and 4000 g in Dream island whereas in Biscoe island the mean is between approximately 4500 and 5500 g. Other conclusions can be drawn from these plots, too.

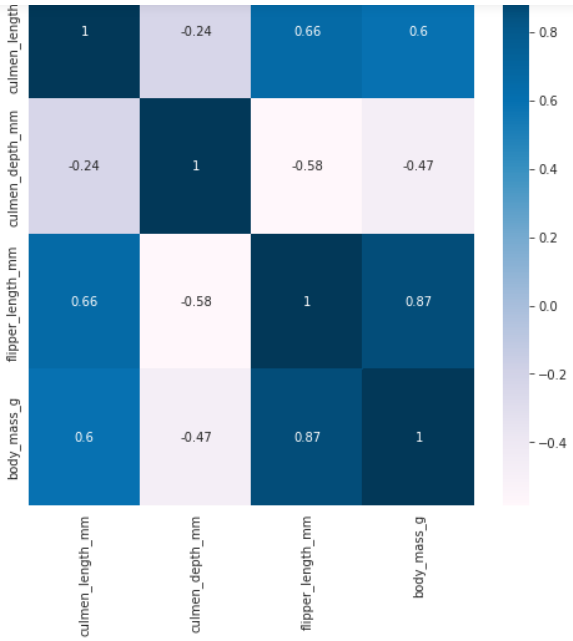
```
sns.violinplot(x = 'island',y = 'body_mass_g',data = penguins_size, palette="YlOrRd_r")
plt.title('Violin plot')
```



Correlation Matrix

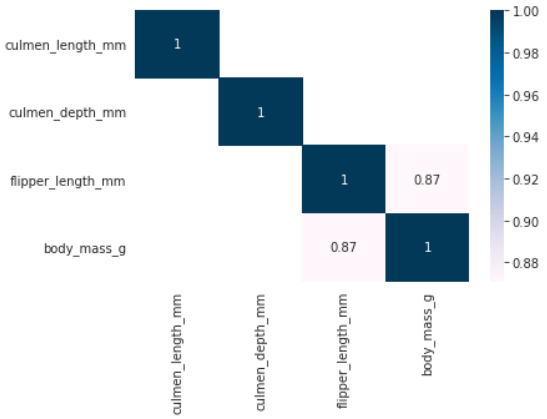
The summarization of our data is finally done with the correlation matrix. This matrix shows the correlation between features. The diagonal values are 1 since the features are correlated with themselves. When the relationship between length of flipper and the body mass is questioned, from the correlation matrix it is understood that their correlation is 0.87 which is quite high.

```
corr = penguins_size.corr()
plt.figure(figsize=(8,8))
sns.heatmap(corr, annot = True, cmap = "PuBu")
plt.title('Correlation Matrix')
plt.show()
```



When lots of features are present, visualizing the heatmap only with high values is more useful. As seen below, features correlated more than 0.8 are show and only one correlation (which is between body mass and flipper length) is seen.

```
sns.heatmap(corr[(corr > 0.8)],annot = True, cmap="PuBu")
```



This is a really short and quick way to perform exploratory data analysis and this dataset is a small and easily understandable one. The full code can be reached from here. I hope you enjoyed reading. Thanks a lot!