iris

October 14, 2023

1 Performing Unsupervised Machine Learning on the 'Iris' dataset to predict the optimal number of clusters. Utilize Python for the analysis and visually represent the identified clusters.

2 Author: AHMED REKIK.

THE MACHINE LEARNING INTERNSHIP ### Dataset: (https://bit.ly/2kXTdox)

[ ]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
iris = pd.read_csv("/content/IRIS.csv")
iris.head()
```

[ ]:

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

```
<google.colab._quickchart_helpers.SectionTitle at 0x7f31f56e9d50>
```

```
import numpy as np
from google.colab import autoviz

def value_plot(df, y, figscale=1):
  from matplotlib import pyplot as plt
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(_df_0, *['sepal_length'], **{})
chart
```

```
import numpy as np
```

```
from google.colab import autoviz

def value_plot(df, y, figscale=1):
  from matplotlib import pyplot as plt
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(_df_1, *['sepal_width'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def value_plot(df, y, figscale=1):
  from matplotlib import pyplot as plt
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(_df_2, *['petal_length'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7f31f32c79d0>
```

```
import numpy as np
from google.colab import autoviz

def histogram(df, colname, num_bins=20, figscale=1):
  from matplotlib import pyplot as plt
  df[colname].plot(kind='hist', bins=num_bins, title=colname,␣
 ↪figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(_df_3, *['sepal_length'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def histogram(df, colname, num_bins=20, figscale=1):
  from matplotlib import pyplot as plt
  df[colname].plot(kind='hist', bins=num_bins, title=colname,␣
 ↪figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(_df_4, *['sepal_width'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def histogram(df, colname, num_bins=20, figscale=1):
  from matplotlib import pyplot as plt
  df[colname].plot(kind='hist', bins=num_bins, title=colname,␣
 ↪figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(_df_5, *['petal_length'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7f31f3233f40>
```

```
import numpy as np
from google.colab import autoviz

def scatter_plots(df, colname_pairs, figscale=1, alpha=.8):
  from matplotlib import pyplot as plt
  plt.figure(figsize=(len(colname_pairs) * 6 * figscale, 6 * figscale))
  for plot_i, (x_colname, y_colname) in enumerate(colname_pairs, start=1):
    ax = plt.subplot(1, len(colname_pairs), plot_i)
    df.plot(kind='scatter', x=x_colname, y=y_colname, s=(32 * figscale),␣
 ↪alpha=alpha, ax=ax)
    ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = scatter_plots(_df_6, *[[['sepal_length', 'sepal_width'], ['sepal_width',␣
 ↪'petal_length']]], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7f31f32337f0>
```

```
import numpy as np
from google.colab import autoviz

def time_series_multiline(df, timelike_colname, value_colname, series_colname,␣
 ↪figscale=1, mpl_palette_name='Dark2'):
  from matplotlib import pyplot as plt
  import seaborn as sns
  figsize = (10 * figscale, 5.2 * figscale)
  palette = list(sns.palettes.mpl_palette(mpl_palette_name))
  def _plot_series(series, series_name, series_index=0):
    if value_colname == 'count()':
      counted = (series[timelike_colname]
                 .value_counts()
                 .reset_index(name='counts')
                 .rename({'index': timelike_colname}, axis=1)
                 .sort_values(timelike_colname, ascending=True))
      xs = counted[timelike_colname]
      ys = counted['counts']
    else:
      xs = series[timelike_colname]
      ys = series[value_colname]
    plt.plot(xs, ys, label=series_name, color=palette[series_index %␣
 ↪len(palette)])

  fig, ax = plt.subplots(figsize=figsize, layout='constrained')
  df = df.sort_values(timelike_colname, ascending=True)
  if series_colname:
    for i, (series_name, series) in enumerate(df.groupby(series_colname)):
      _plot_series(series, series_name, i)
    fig.legend(title=series_colname, bbox_to_anchor=(1, 1), loc='upper left')
  else:
    _plot_series(df, '')
  sns.despine(fig=fig, ax=ax)
  plt.xlabel(timelike_colname)
  plt.ylabel(value_colname)
  return autoviz.MplChart.from_current_mpl_state()

chart = time_series_multiline(_df_7, *['petal_width', 'sepal_length', None],␣
 ↪**{})
chart
```

```
import numpy as np
from google.colab import autoviz

def time_series_multiline(df, timelike_colname, value_colname, series_colname,␣
 ↪figscale=1, mpl_palette_name='Dark2'):
  from matplotlib import pyplot as plt
  import seaborn as sns
  figsize = (10 * figscale, 5.2 * figscale)
  palette = list(sns.palettes.mpl_palette(mpl_palette_name))
  def _plot_series(series, series_name, series_index=0):
    if value_colname == 'count()':
      counted = (series[timelike_colname]
                 .value_counts()
                 .reset_index(name='counts')
                 .rename({'index': timelike_colname}, axis=1)
                 .sort_values(timelike_colname, ascending=True))
      xs = counted[timelike_colname]
      ys = counted['counts']
    else:
      xs = series[timelike_colname]
      ys = series[value_colname]
    plt.plot(xs, ys, label=series_name, color=palette[series_index % len(palette)])

  fig, ax = plt.subplots(figsize=figsize, layout='constrained')
  df = df.sort_values(timelike_colname, ascending=True)
  if series_colname:
    for i, (series_name, series) in enumerate(df.groupby(series_colname)):
      _plot_series(series, series_name, i)
    fig.legend(title=series_colname, bbox_to_anchor=(1, 1), loc='upper left')
  else:
    _plot_series(df, '')
  sns.despine(fig=fig, ax=ax)
  plt.xlabel(timelike_colname)
  plt.ylabel(value_colname)
  return autoviz.MplChart.from_current_mpl_state()

chart = time_series_multiline(_df_8, *['petal_width', 'sepal_width', None], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def time_series_multiline(df, timelike_colname, value_colname, series_colname, figscale=1, mpl_palette_name='Dark2'):
  from matplotlib import pyplot as plt
  import seaborn as sns
  figsize = (10 * figscale, 5.2 * figscale)
  palette = list(sns.palettes.mpl_palette(mpl_palette_name))
  def _plot_series(series, series_name, series_index=0):
    if value_colname == 'count()':
      counted = (series[timelike_colname]
                 .value_counts()
                 .reset_index(name='counts')
                 .rename({'index': timelike_colname}, axis=1)
                 .sort_values(timelike_colname, ascending=True))
      xs = counted[timelike_colname]
      ys = counted['counts']
    else:
      xs = series[timelike_colname]
      ys = series[value_colname]
    plt.plot(xs, ys, label=series_name, color=palette[series_index % len(palette)])
  fig, ax = plt.subplots(figsize=figsize, layout='constrained')
  df = df.sort_values(timelike_colname, ascending=True)
  if series_colname:
    for i, (series_name, series) in enumerate(df.groupby(series_colname)):
      _plot_series(series, series_name, i)
    fig.legend(title=series_colname, bbox_to_anchor=(1, 1), loc='upper left')
  else:
    _plot_series(df, '')
  sns.despine(fig=fig, ax=ax)
  plt.xlabel(timelike_colname)
  plt.ylabel(value_colname)
  return autoviz.MplChart.from_current_mpl_state()

chart = time_series_multiline(_df_9, *['petal_width', 'petal_length', None],␣
 ↪**{})
chart
```

```
import numpy as np
from google.colab import autoviz

def time_series_multiline(df, timelike_colname, value_colname, series_colname,␣
 ↪figscale=1, mpl_palette_name='Dark2'):
  from matplotlib import pyplot as plt
  import seaborn as sns
  figsize = (10 * figscale, 5.2 * figscale)
  palette = list(sns.palettes.mpl_palette(mpl_palette_name))
  def _plot_series(series, series_name, series_index=0):
    if value_colname == 'count()':
      counted = (series[timelike_colname]
                 .value_counts()
                 .reset_index(name='counts')
                 .rename({'index': timelike_colname}, axis=1)
                 .sort_values(timelike_colname, ascending=True))
      xs = counted[timelike_colname]
      ys = counted['counts']
    else:
      xs = series[timelike_colname]
      ys = series[value_colname]
    plt.plot(xs, ys, label=series_name, color=palette[series_index %␣
 ↪len(palette)])

  fig, ax = plt.subplots(figsize=figsize, layout='constrained')
  df = df.sort_values(timelike_colname, ascending=True)
  if series_colname:
    for i, (series_name, series) in enumerate(df.groupby(series_colname)):
      _plot_series(series, series_name, i)
    fig.legend(title=series_colname, bbox_to_anchor=(1, 1), loc='upper left')
  else:
    _plot_series(df, '')
  sns.despine(fig=fig, ax=ax)
  plt.xlabel(timelike_colname)
  plt.ylabel(value_colname)
  return autoviz.MplChart.from_current_mpl_state()

chart = time_series_multiline(_df_10, *['petal_width', 'count()', None], **{})
chart
```

[ ]:
```
iris.shape
```

[ ]:
```
(150, 5)
```

[ ]:
```
iris.columns
```

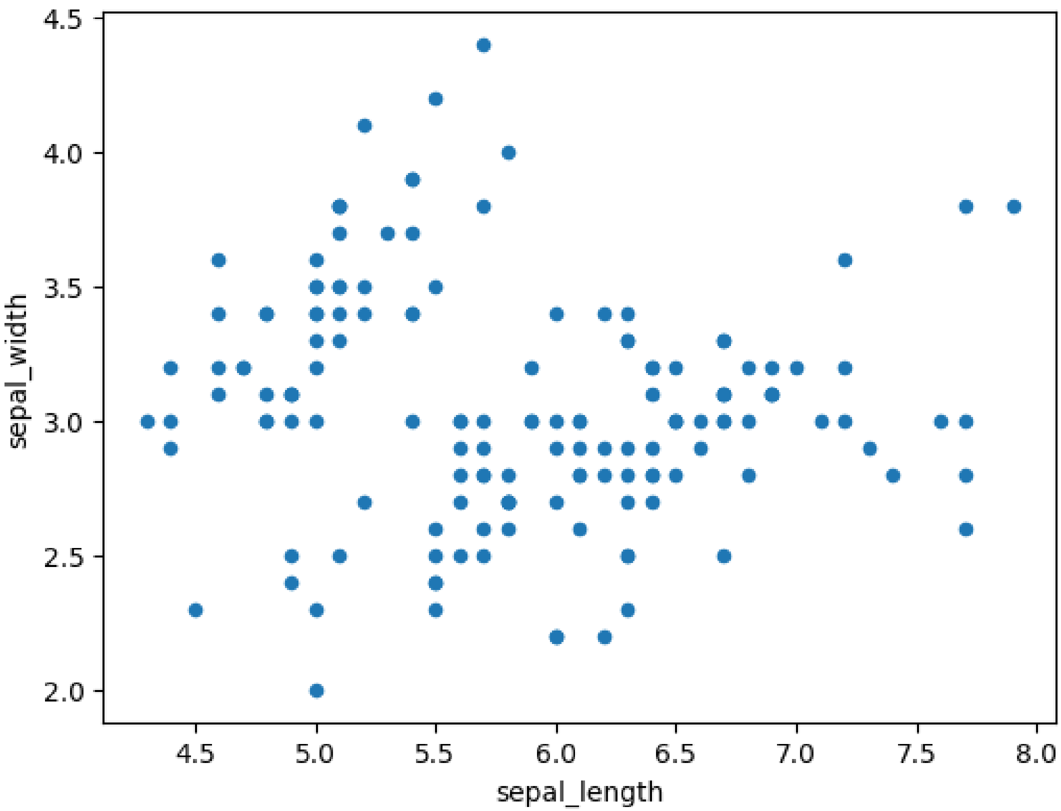
[ ]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

[ ]:
```
iris["species"].value_counts()
```

[ ]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: species, dtype: int64
```

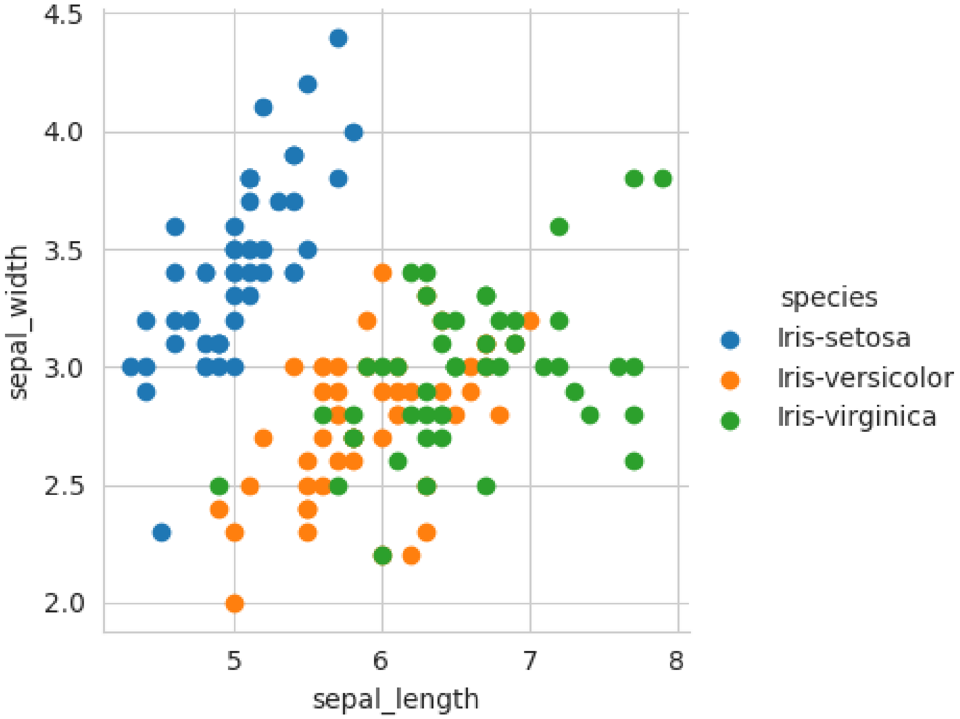
3 Scatter plot (EDA)

[ ]:
```
iris.plot(kind='scatter', x='sepal_length', y= 'sepal_width')
plt.show()
```



[ ]:

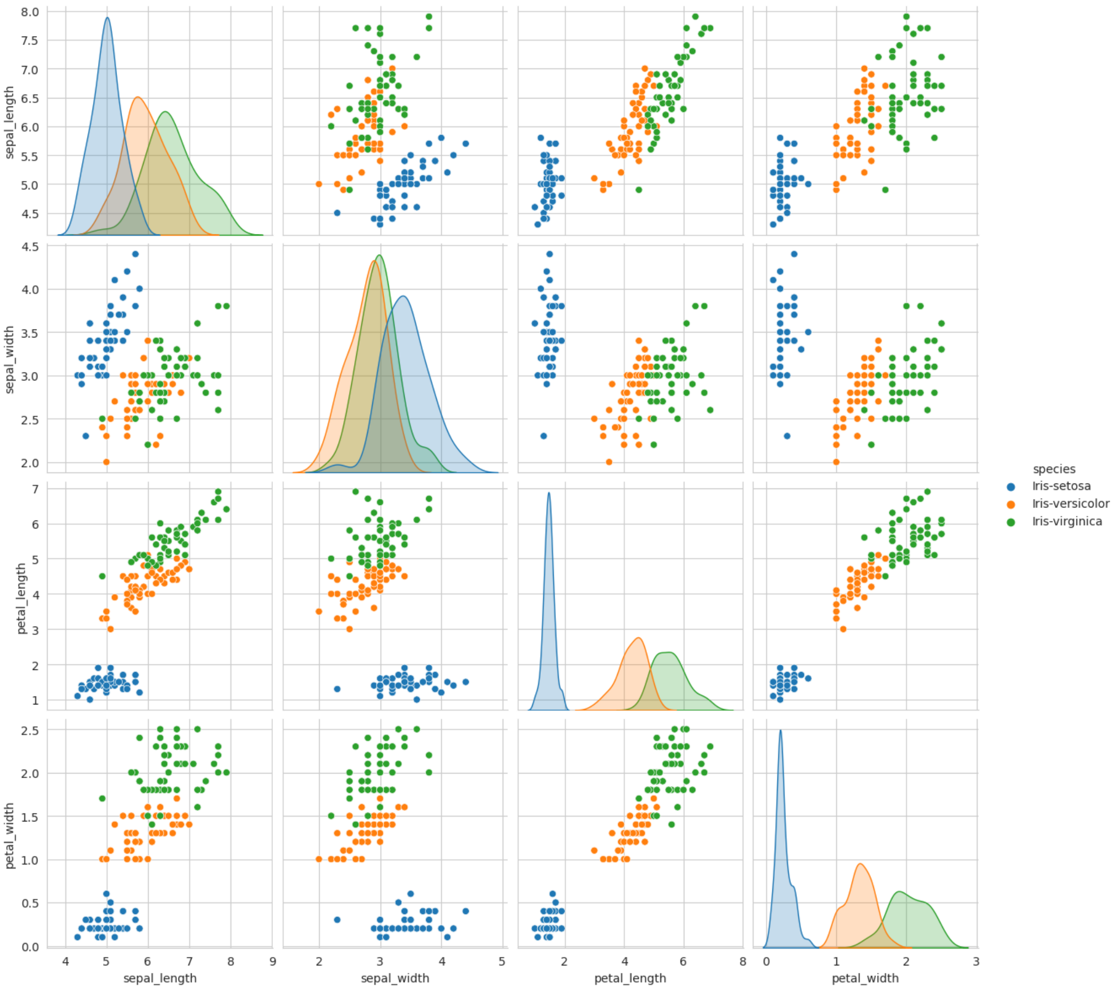
```
sns.set_style('whitegrid')
MF =sns.FacetGrid(iris, hue="species", height =4)
MF.map(plt.scatter,"sepal_length","sepal_width")
MF.add_legend()
plt.show()
```



3.1 Pair Plot (EDA)

[ ]:

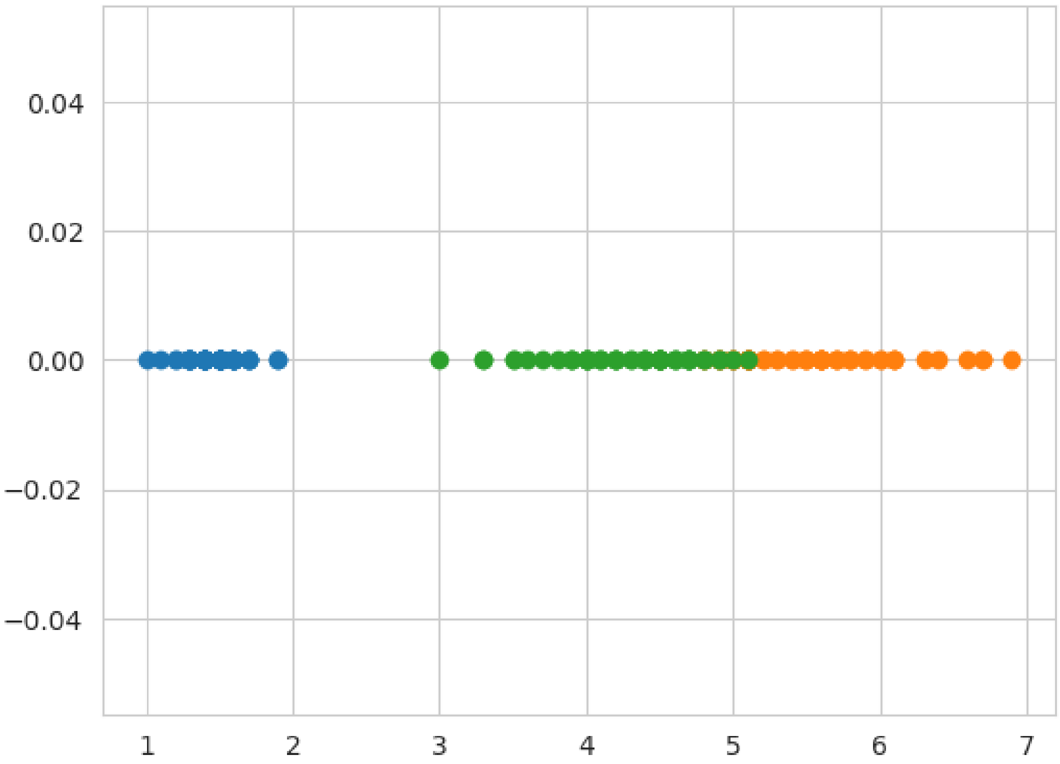
```
sns.set_style("whitegrid")
sns.pairplot(iris, hue="species",height=3)
plt.show()
```



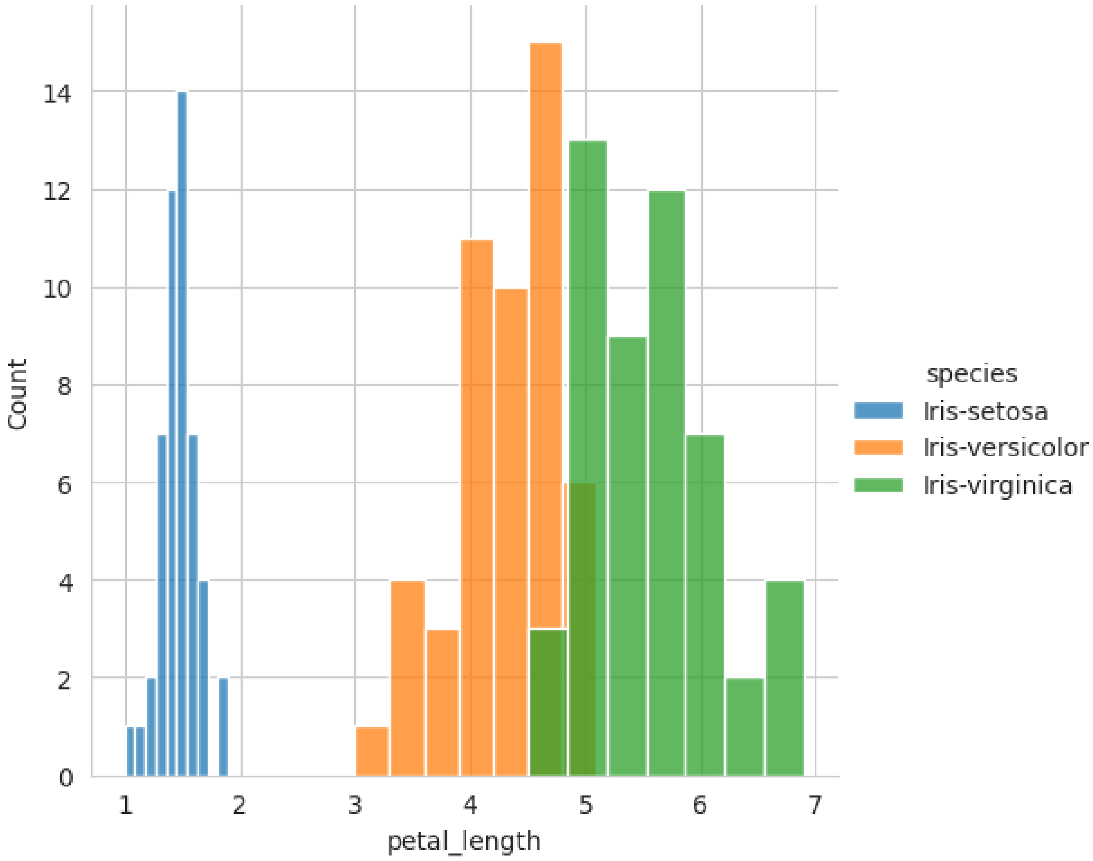
4 Histogram (EDA)

[ ]:

```
iris_setosa = iris.loc[iris['species'] == 'Iris-setosa']
iris_virginica = iris.loc[iris['species']=='Iris-virginica']
iris_versicolor = iris.loc[iris['species']=='Iris-versicolor']

plt.plot(iris_setosa['petal_length'], np.
 ↪zeros_like(iris_setosa['petal_length']),'o')
plt.plot(iris_virginica['petal_length'], np.
 ↪zeros_like(iris_virginica['petal_length']),'o')
plt.plot(iris_versicolor['petal_length'], np.
 ↪zeros_like(iris_versicolor['petal_length']),'o')
plt.show()
```

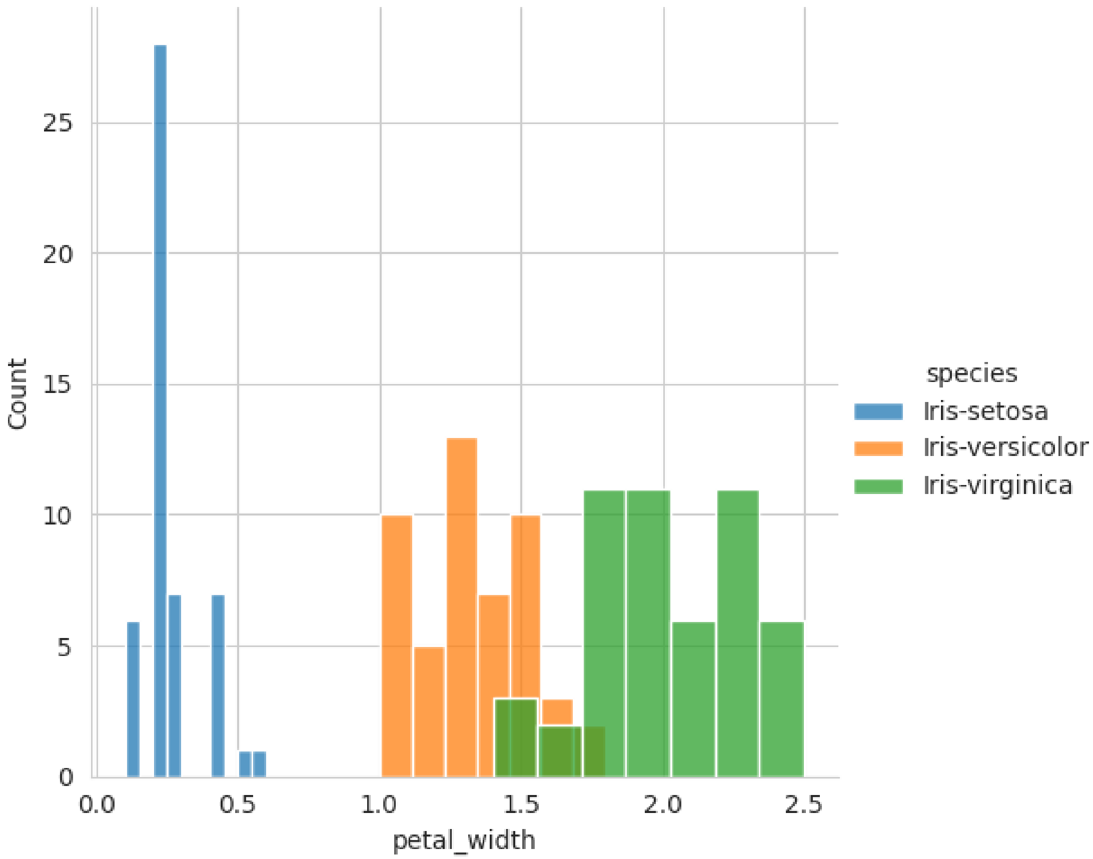


```
[ ]: ## Histogram
     FM = sns.FacetGrid(iris,hue='species',height=5)
     FM.map(sns.histplot,'petal_length')
     FM.add_legend()
     plt.show()
```



[ ]:

```
fm = sns.FacetGrid(iris,hue='species',height =5)
fm.map(sns.histplot,'petal_width')
fm.add_legend()
plt.show()
```

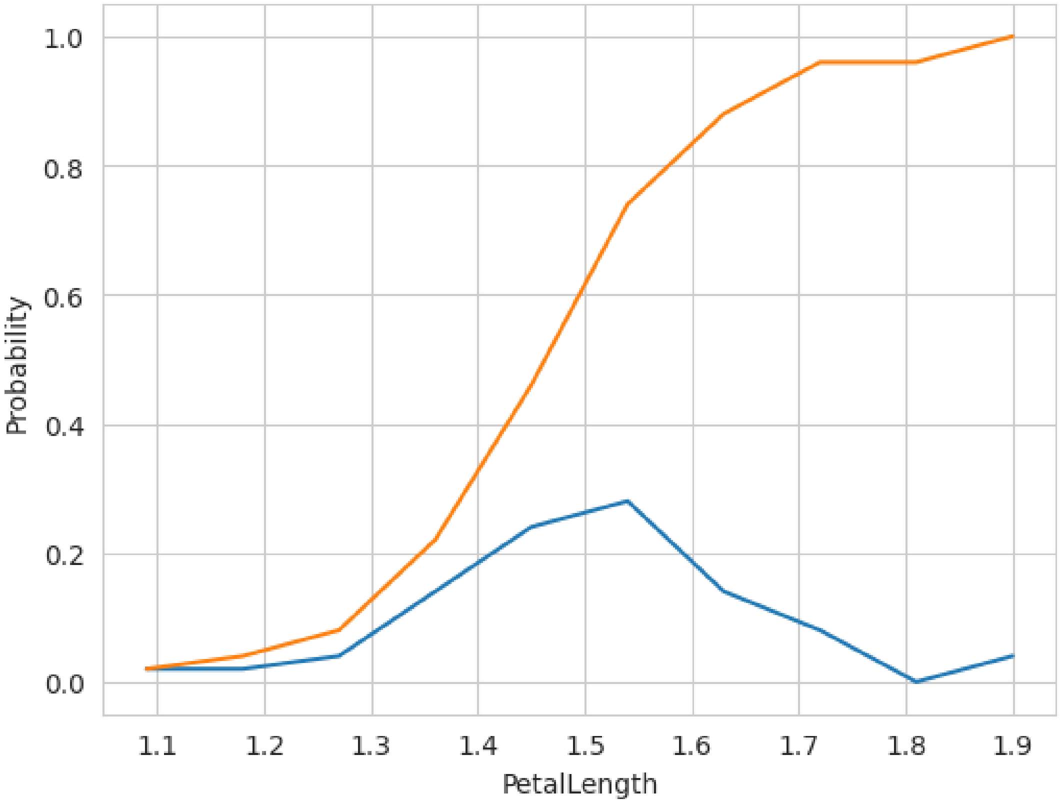


Cumulative Distribution Function CDF

check the probability

[ ]:

```
counts, bin_edges = np.histogram(iris_setosa['petal_length'], bins=10, density=True)
pdf= counts/(sum(counts))
pdf
bin_edges

# compute CDF
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:],cdf)
plt.xlabel("PetalLength")
plt.ylabel("Probability")
plt.show()
```



5 Mean and Standard Deviation

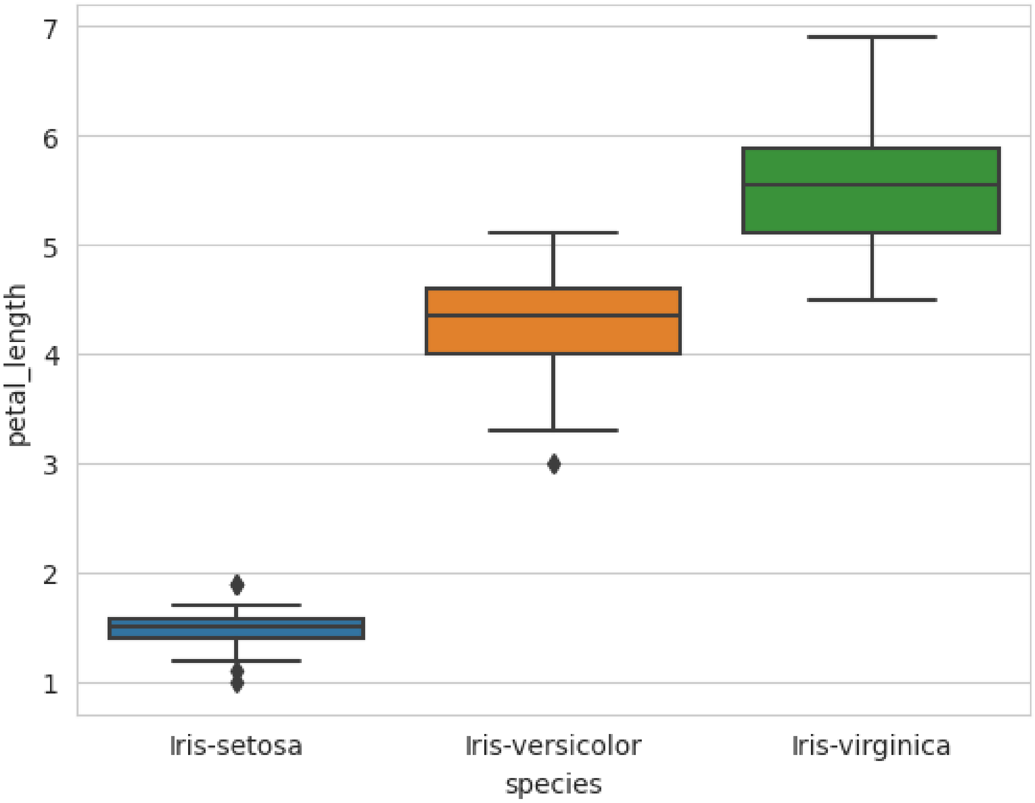
[ ]:
```
print("Means:")
print(np.mean(iris_setosa['petal_length']))
#Mean with an outlier
print(np.mean(np.append(iris_setosa["petal_length"],50)))
print(np.mean(iris_virginica['petal_length']))
print(np.mean(iris_versicolor['petal_length']))

print("\nStd-dev:")
print(np.std(iris_setosa['petal_length']))
print(np.std(iris_virginica['petal_length']))
print(np.std(iris_versicolor['petal_length']))
```

```
Means:
1.464
2.4156862745098038
5.5520000000000005
4.26
```

```
Std-dev:
0.17176728442867112
0.546347874526844
0.4651881339845203
```

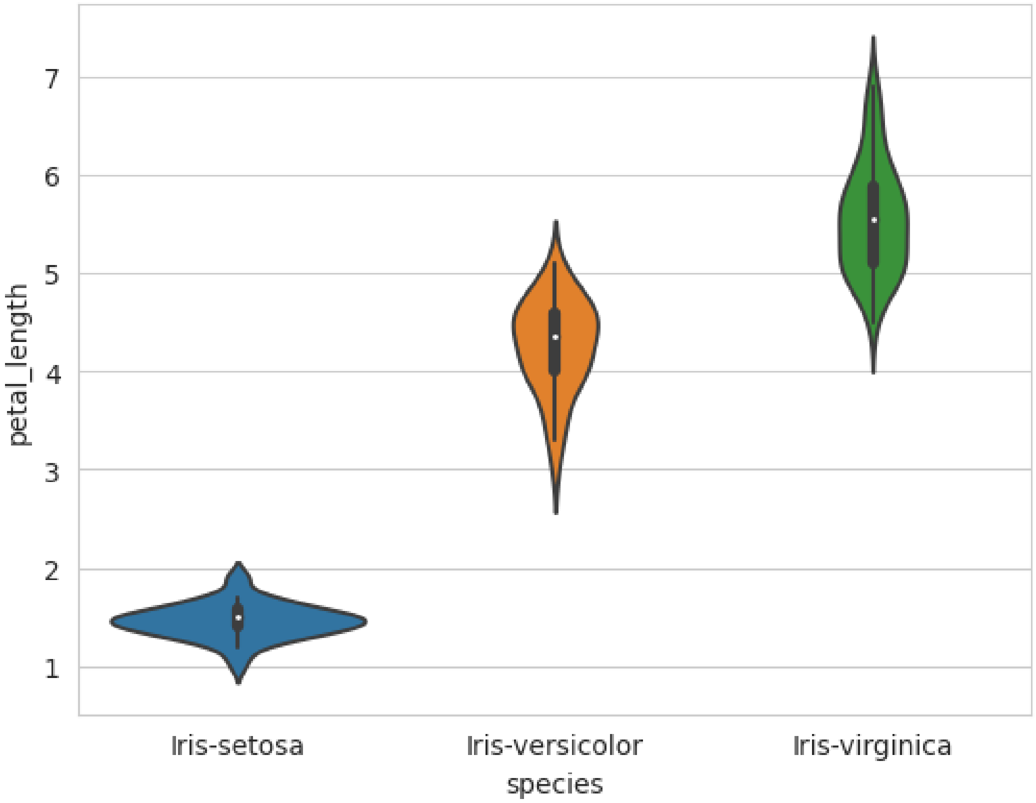
6 box plot

```
[ ]: sns.boxplot(x='species',y='petal_length', data=iris)
     plt.show()
```



7 violin plot

```
[ ]: sns.violinplot(x='species',y='petal_length',data=iris)
     plt.show()
```


7
6
5
4
3
2
1
petal_length
Iris-setosa
Iris-versicolor
Iris-virginica
species