31. Logistic Regression (Practical) (Multiclass Classification)

- It follows **OVR (One versus Rest) method**.
- for exmaple our data is comprises of cat, dog and cow - so to convert it to one-hot-encoding:
- Cat Dog Cow
- 1 0 0
- 0 1 0
- 0 0 1

In [13]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [14]:
```
dataset = pd.read_csv(r'Data/iris.csv')
dataset.head(3)
```

Out[14]:

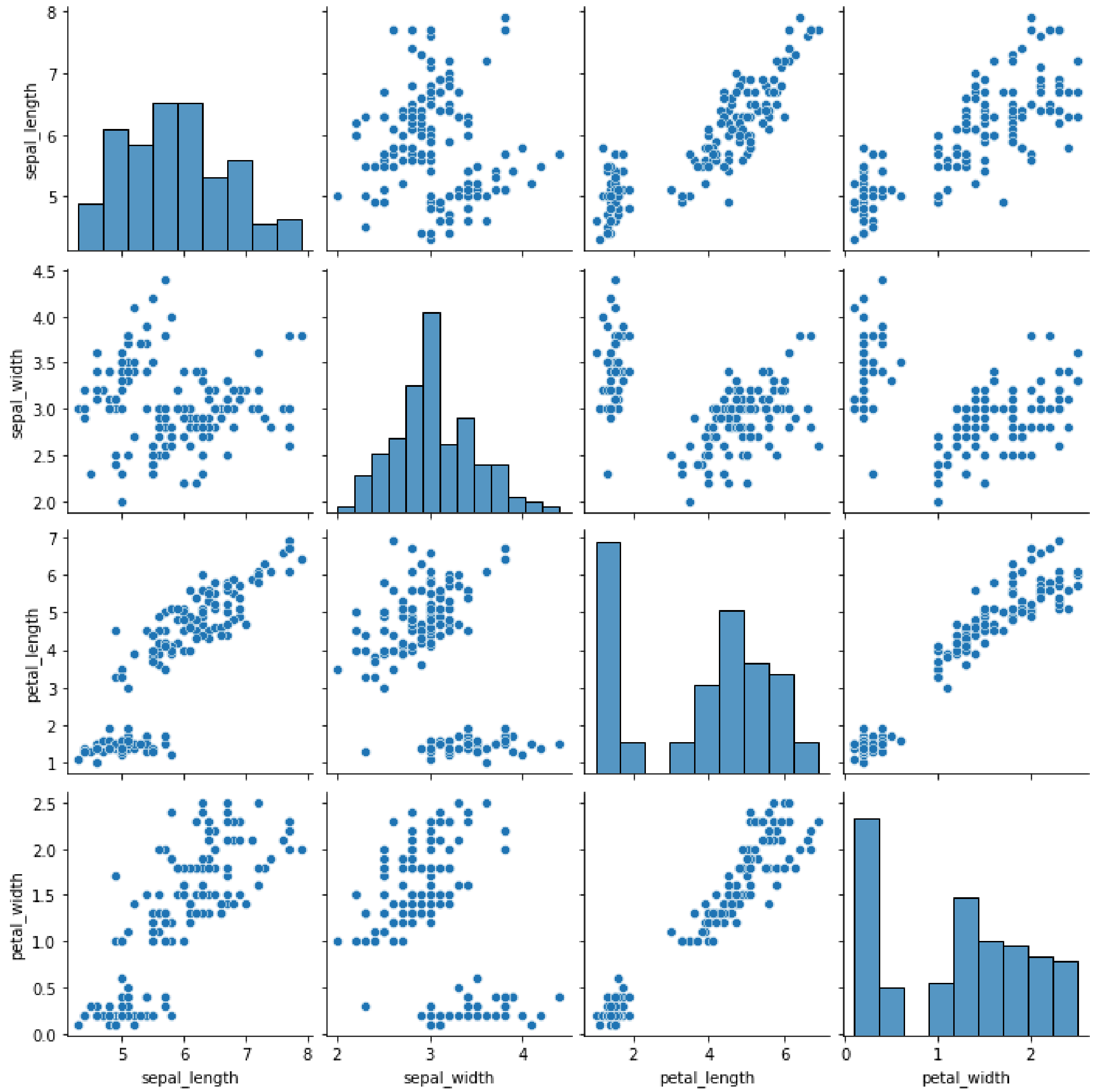
	sepal_length	sepal_width	petal_length	petal_width	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa

In [15]:
```
dataset["Species"].unique()
```

Out[15]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

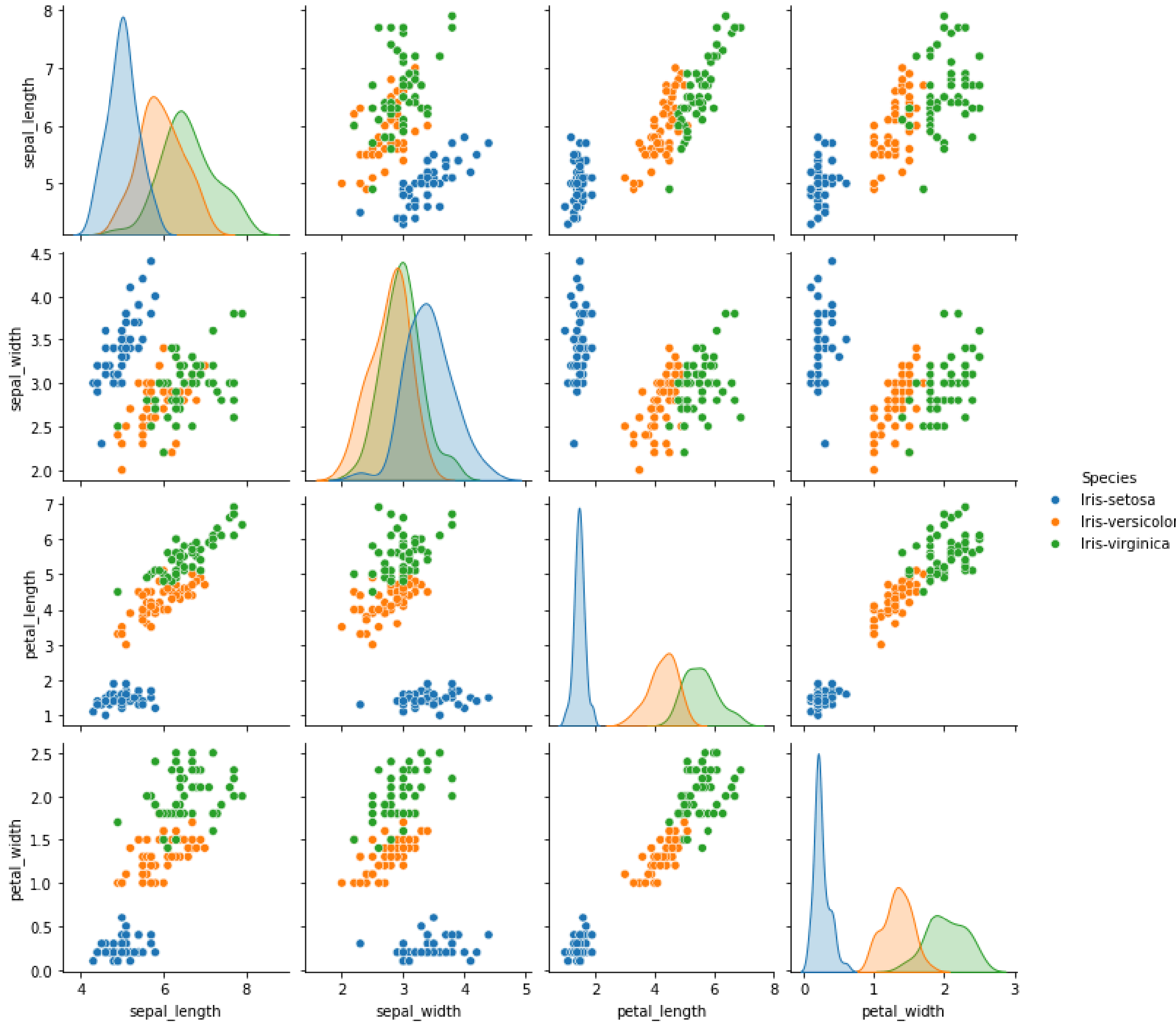
In [16]:
```
sns.pairplot(data=dataset)
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\axis
grid.py:123: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [17]:
```
sns.pairplot(data=dataset, hue='Species')
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\axis
grid.py:123: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Interpretation of above graphs

- sepal length - all three curvatures are overlapping with one another, so to find differential is difficult
- sepal width - all 3 curvatures are more overlapping than even those of seapl length
- petal length - all 3 curvatures are distinct and discrete - so we can find differential easily - so we can easily classify them
- petal_width - all 3 curvatures are distinct and discrete - so we can find differential easily - so we can easily classify them
- From these curvatures, we can take information for feature(s) selection, which feature to take and which to drop for building model so that the built model will be accurately trained
- sepal_width is pretty much overlapped, so we cannot use this features for model training - it should be dropped

- but as for this notebook, feature_selection is not the topic so we will keep all the features for now
- the focus of this exercise is logistic regression with multiclass features selection

Separate dependent (ouput, y) and independent variables (input, x)

In [19]:
```
x = dataset.iloc[:,:-1]
x
```

Out[19]:

	sepal_length	sepal_width	petal_length	petal_width
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
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [20]:
```
y = dataset['Species']
y
```

Out[20]:
```
0          Iris-setosa
1          Iris-setosa
2          Iris-setosa
3          Iris-setosa
4          Iris-setosa
              ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

Split the data into train and test data

In [21]:
```
from sklearn.model_selection import train_test_split
```

In [23]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_st
```

Apply Logistic Regression through OVR Method

multi_class{'auto', 'ovr', 'multinomial'}, default='auto' If the option chosen is 'ovr', then a binary problem is fit for each label. For 'multinomial' the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. 'multinomial' is unavailable when solver='liblinear'. 'auto' selects 'ovr' if the data is binary, or if solver='liblinear', and otherwise selects 'multinomial'.

In [24]:
```
from sklearn.linear_model import LogisticRegression
```

In [29]:
```
lg = LogisticRegression(multi_class="ovr")
lg.fit(x_train, y_train)
```

Out[29]:
```
▾ LogisticRegression
LogisticRegression(multi_class='ovr')
```

Check accuracy of the model

In [30]:
```
lg.score(x_test, y_test)*100
```

Out[30]:
```
96.66666666666667
```

Apply Logistic Regression through Multinomial Method

In [31]:
```
lg1 = LogisticRegression(multi_class="multinomial")
lg1.fit(x_train, y_train)
```

Out[31]:
```
▾ LogisticRegression
LogisticRegression(multi_class='multinomial')
```

Check accuracy of the model

In [32]:
```
lg1.score(x_test, y_test)*100
```

Out[32]:
```
100.0
```

Apply Logistic Regression through Direct (Default) Method

In [36]:
```
lg2 = LogisticRegression()
lg2.fit(x_train, y_train)
```

Out[36]:

```
▾ LogisticRegression
LogisticRegression()
```

Check accuracy of the model

In [35]:

```python
lg2.score(x_test, y_test)*100
```

Out[35]:

```
100.0
```

In [ ]:

```python

```