Report about the iris flower data

Link to the Github page :

https://gitlab.com/python7963908/cassie_doguet_iris_dataset_dia

Description of the algorithm used for each of the solution

With the iris flower dataset we had to classify images of flowers based on their features. I decided to apply the model random forest to classify the flowers.

Here are the three first rows of the data set :

```
data_iris = pd.read_csv("...")
data_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

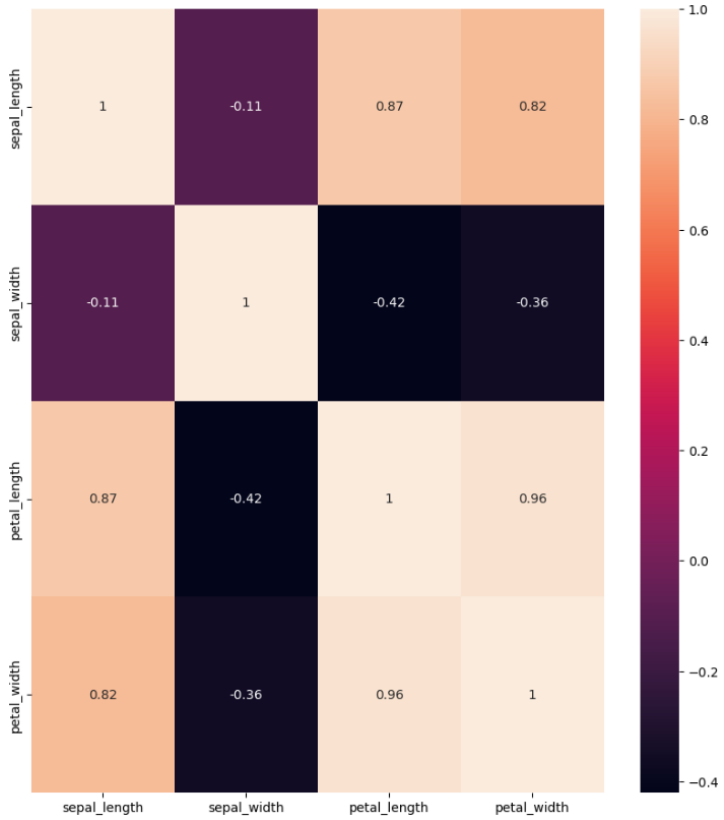
And here is a description of the data :

```
data_iris.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Then we wanted to see the correlation of between the features :

```
plt.figure(figsize=(10,11))
sns.heatmap(data_iris.corr(),annot=True)
plt.plot()
```



So here we can see that the most correlated parameters are :

- Petal length and petal width
- Petal length and sepal length
- Sepal width and petal width

So all the features are important. Then we have to clean the data and see if there are any lack of information:

```
print("Missing values distribution: ")
print(data_iris.isnull().mean())
```

```
Missing values distribution:
sepal_length    0.0
sepal_width     0.0
petal_length    0.0
petal_width     0.0
species         0.0
dtype: float64
```

As we can see there is no lack of information, so we have nothing to clean.

Then we must see what the data type of our dataset are:

```
print("Column datatypes: ")
print(data_iris.dtypes)
```

```
Column datatypes:
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

The column species is a type object. It must be a type float or int to fit into the model. Then we use this method to encode the species. We will see in the next project that there are other solutions to do it.

```
df_species = data_iris['species'].to_numpy()
for i in range(0,len(df_species)) :
    if df_species[i] == 'Iris-setosa' :
        df_species[i] = 1
    elif df_species[i] == 'Iris-versicolor' :
        df_species[i] = 2
    elif df_species[i] == 'Iris-virginica' :
        df_species[i] = 3
target = pd.Series(df_species)
target = target.astype('int')
```

So now that we have our target data (what the model must predict). We now need to separate our dataset into 4 part :

- Two training parts (one for the features and one for the targets), respectively x_train and y_train
- Two testing parts (again on for the features and one for the targets), respectively x_test and y_test

We choose to use 75% of out dataset for the training and 25% for the testing.

```
feature = data_iris.drop(['species'],axis = 1)

x_train,x_test,y_train,y_test = train_test_split(feature,target,train_size = 0.75)
```

Then we want to normalise the data before fitting it into the machine learning model. The function *StandardScaler()* standardize feature by removing the mean and scaling to unit variance. *fit_transform()* is used to fit the *StandardScaler* to the training data and then transform the data by centring and scaling it. *transform()* applies the same scaling data to the testing data set so that it's transformed in the same way as the training data.

We have now normalized the data and improved the performance of our machine learning model.

```
st_x = StandardScaler()
x_train = st_x.fit_transform(x_train)
x_test=st_x.transform(x_test)
```

How the model performance was improved

To begin with, random forest works with a large number of individual decision trees. Each make a decision and the most voted prediction is outputted. It protects each tree from other tree's individual errors.

At first I tried to improve the model performance by using the n_estimator hyper parameter (the methods are still in the notebook). But then I decided to use the Grid Search to find the best hyperparameters for our model.

First of all, hyperparameters are settings that are not learned by the marchine learning algorithm, but are set before training the model. The Grid Search algorithm works by exhaustively searching over a pre-definded set of hyperparameters for our machine learning model. It will then give us the best final model and its accuracy.

```
iris = load_iris()

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 15, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rfc = RandomForestClassifier()
grid_search = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)
best_params = grid_search.best_params_
final_model = RandomForestClassifier(**best_params)
final_model.fit(x_train, y_train)
y_pred = final_model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9736842105263158
```

As we can see previously the final model training as an accuracy of 97%.

The errors are often between the species versicolor (orange) and virginica (green). As we can see on theses graphs their features are verry similar.

```
sns.pairplot(data_iris,hue='species',height=4)
```

```
<seaborn.axisgrid.PairGrid at 0x19e97a468e0>
```