Machine Learning Assignment 1 (group 24)

I. Iris Species

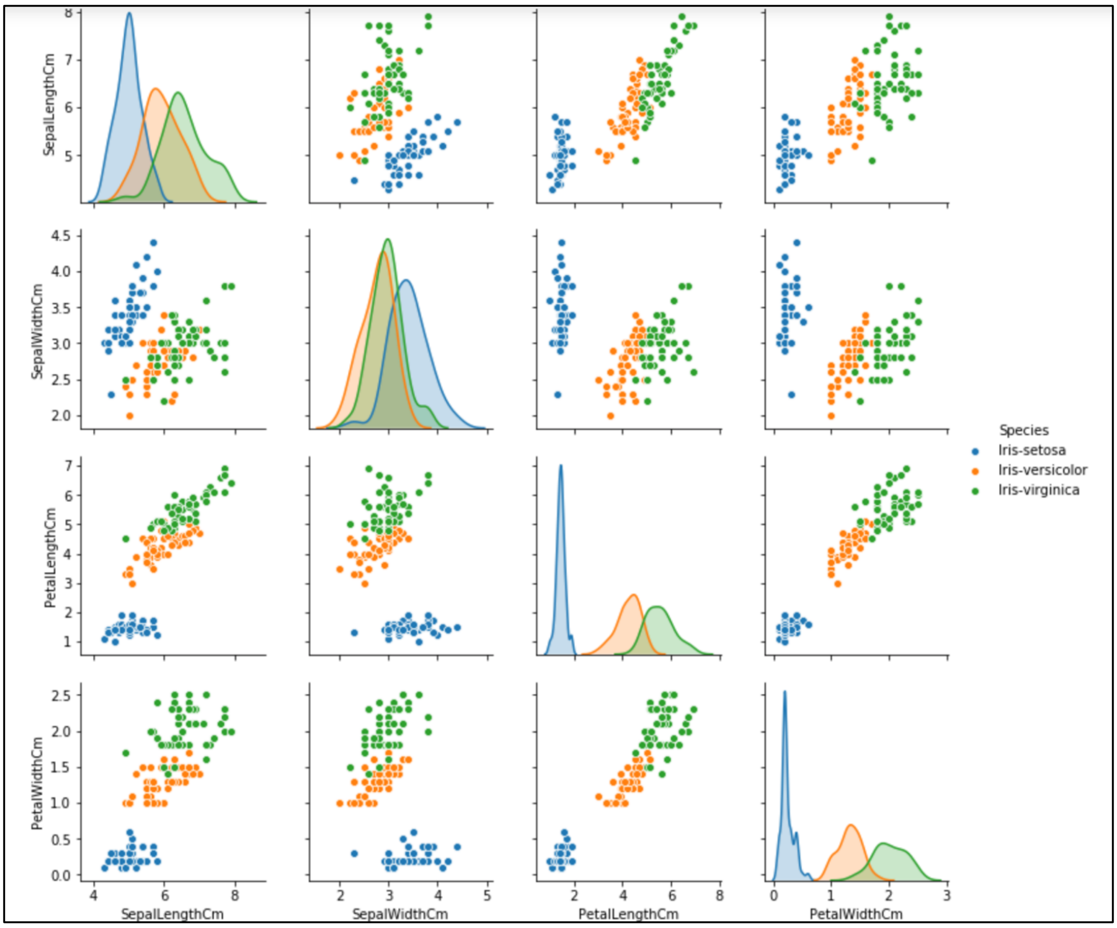
(I) Working environment

	張翔中	劉昱劭	彭敬樺	周才錢
OS	macOS	macOS	Windows	Windows
IDE	Jupyter Notebook	Jupyter Notebook	Jupyter Notebook	Jupyter Notebook

We use Jupyter Notebook public at **140.113.215.82:8888**

(II) Basic visualization

➔ Use seaborn package



Relation between paired features

➔ Data.decribe()

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
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
50%	5.800000	3.000000	4.350000	1.300000
max	7.900000	4.400000	6.900000	2.500000

(III) Data preprocessing

1. Split data and targets
   ➔ Data: 'Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width'
   ➔ Target: 'Species'
2. Split training data and testing data
   ➔ train_test_split()
3. Index targets with numbers
   ➔ {'Iris-setosa' : 0 , 'Iris-versicolor' : 1 , 'Iris-virginica' : 2}

(IV) Decision tree & random forest

1. Training:
   Select two columns (features) of the data each time to train a decision tree
   ➔ sklearn.tree.DecisionTreeClassifier()
   ➔ Use function fit() for training
2. Validation:
   (1) Resubstitution
      ➔ Self-validation: use training data as validation data
      ➔ Should be almost 100% accuracy
   (2) K-fold CV
      ➔ Split training data into K parts
      ➔ Use one of them to do validation and use the other to train the tree
      ➔ Repeat K times with different pieces of training data and choose the tree with best score (accuracy)
3. Testing:
   ➔ Random forest: iris dataset contains 4 columns => 6 trees
   ➔ Predictions with most vote become final result
   ➔ If don't exist most vote => prediction failed

(V) Performance

1. Confusion matrix for K-fold: (p -> prediction, t -> target)

	Setosa (p)	Versicolor (p)	Virginica (p)
Setosa (t)	19	0	0
Versicolor (t)	0	12	2
Virginica (t)	0	2	9

2. Precision & recall for K-fold

	precision	recall
Setosa	1	1
Versicolor	0.857143	0.857143
Virginica	0.818182	0.818182

3. Confusion matrix for Resubstitution : (p -> prediction, t -> target)

	Setosa (p)	Versicolor (p)	Virginica (p)
Setosa (t)	12	0	0
Versicolor (t)	0	12	1
Virginica (t)	0	3	17

4. Precision & recall for Resubstitution:

	precision	recall
Setosa	1	1
Versicolor	0.8	0.923077
Virginica	0.944444	0.85

(VI) Conclusion

1. Outcome differs in every execution, due to the random split between testing data and training data
2. Not much preprocessing need to be done due to clean dataset

II. Google Play Store Apps

(I) Working environment: same as above

(II) Basic visualization

In [2]:
```
# read csv
# 直接把目前用不掉的幾個 column drop 掉
store = pd.read_csv('googleplaystore.csv').drop(['Size', 'Current Ver', 'Android Ver'], axis=1)
store.describe(include='all')
```

Out[2]:

	App	Category	Rating	Reviews	Installs	Type	Price	Content Rating	Genres	Last Updated
count	10841	10841	9367.000000	10841	10841	10840	10841	10840	10841	10841
unique	9660	34	NaN	6002	22	3	93	6	120	1378
top	ROBLOX	FAMILY	NaN	0	1,000,000+	Free	0	Everyone	Tools	August 3, 2018
freq	9	1972	NaN	596	1579	10039	10040	8714	842	326
mean	NaN	NaN	4.193338	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	NaN	NaN	0.537431	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	NaN	NaN	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	NaN	NaN	4.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	NaN	NaN	4.300000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	NaN	NaN	4.500000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	NaN	NaN	19.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN

(III) Data preprocessing

1. Drop flaw data

```
## 這筆資料有問題，直接丟掉
for x in store.loc[store['Type']=='0'].index:
    store = store.drop([x])
```

2. Deal with unusual data

```
# find dirty 'Type'
print(store['Type'].unique())
store.loc[(store['Type']!='Free') & (store['Type']!='Paid')].assign()
```

```
['Free' 'Paid' nan '0']
```

(IV) Decision tree & random forest

1. Training:

   Select 3 columns (features) of the data each time to train a decision tree

   Use 'Reviews', 'Rating', 'Price' to predict 'Installs'

   ➔ sklearn.tree.DecisionTreeClassifier()

   ➔ Use function fit() for training

2. Validation:

   (1) Resubstitution

   ➔ Self-validation: use training data as validation data

   ➔ Should be almost 100% accuracy at large 'Installs' because small 'Installs' are intensely smaller than others, which is hard to predict.

(2) K-fold CV

- ➔ Split training data into K parts
- ➔ Use one of them to do validation and use the other to train the tree
- ➔ Repeat K times with different pieces of training data and choose the tree with best score (accuracy)

3. Testing:
   - ➔ Random forest: 3 trees
   - ➔ Predictions with most vote become final result
   - ➔ If don't exist most vote => prediction failed

(V) Performance

1. Confusion matrix for K-fold

	1+	5+	10+	50+	100+	500+	1,000+	5,000+	10,000+	50,000+	100,000+	500,000+	1,000,000+	5,000,000+	10,000,000+	50,000,000+	100,000,000+	500,000,000+	1,000,000,000+
1+	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5+	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10+	0	0	6	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50+	0	0	0	0	12	0	1	0	0	0	0	0	0	0	0	0	0	0	0
100+	0	0	5	0	36	0	35	2	0	0	0	0	0	0	0	0	0	0	0
500+	0	0	0	0	15	0	30	1	1	0	0	0	0	0	0	0	0	0	0
1,000+	0	0	0	0	14	3	95	4	14	0	1	0	0	0	0	0	0	0	0
5,000+	0	0	0	0	1	0	39	12	25	0	0	0	0	0	0	0	0	0	0
10,000+	0	0	0	0	0	0	22	7	114	11	16	0	0	0	0	0	0	0	0
50,000+	0	0	0	0	0	0	1	0	30	9	32	0	2	0	0	0	0	0	0
100,000+	0	0	0	0	0	0	0	0	14	19	107	18	27	0	0	0	0	0	0
500,000+	0	0	0	0	0	0	0	0	1	2	18	26	41	0	2	0	0	0	0
1,000,000+	0	0	0	0	0	0	0	0	0	0	18	15	271	31	32	1	0	0	0
5,000,000+	0	0	0	0	0	0	0	0	0	0	0	2	66	72	54	0	0	0	0
10,000,000+	0	0	0	0	0	0	0	0	0	0	1	1	29	28	259	4	1	1	0
50,000,000+	0	0	0	0	0	0	0	0	0	0	0	0	0	3	28	35	13	0	0
100,000,000+	0	0	0	0	0	0	0	0	0	0	0	0	1	0	8	1	109	0	0
500,000,000+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	15	1
1,000,000,000+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12

2. Confusion matrix for Resubsitution

	1+	5+	10+	50+	100+	500+	1,000+	5,000+	10,000+	\ 50,000+	100,000+	500,000+	1,000,000+	5,000,000+	\ 10,000,000+	50,000,000+	100,000,000+	500,000,000+	1,000,000,000+
1+	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5+	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10+	0	0	3	1	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50+	0	0	3	0	11	0	1	1	0	0	0	0	0	0	0	0	0	0	0
100+	0	0	5	4	49	8	22	3	0	0	0	0	0	0	0	0	0	0	0
500+	0	0	2	2	21	9	21	4	2	0	0	0	0	0	0	0	0	0	0
1,000+	0	0	3	4	50	28	76	22	40	0	1	0	1	0	0	0	0	0	0
5,000+	0	0	1	0	4	11	33	40	39	6	3	0	1	0	1	0	0	0	0
10,000+	0	0	0	0	5	3	36	40	137	31	31	5	1	0	1	0	0	0	0
50,000+	0	0	0	0	0	0	4	7	50	33	48	11	6	0	0	0	0	0	0
100,000+	0	0	0	0	0	0	1	2	38	48	170	49	41	0	0	0	0	0	0
500,000+	0	0	0	0	0	0	0	1	8	4	44	42	46	4	1	0	0	0	0
1,000,000+	0	0	0	0	0	0	1	0	1	2	43	54	274	44	27	1	3	0	0
5,000,000+	0	0	0	0	0	0	0	0	0	0	1	4	64	108	42	2	2	0	0
10,000,000+	0	0	0	0	0	0	0	0	0	0	2	3	32	50	269	20	5	5	0
50,000,000+	0	0	0	0	0	0	0	0	0	0	0	1	0	0	19	47	13	2	1
100,000,000+	0	0	0	0	0	0	0	0	0	0	0	0	0	1	5	5	113	0	1
500,000,000+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	22	0
1,000,000,000+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22
										0	0	0	0	0					

3. Precision & recall for K-fold

	precision	recall
1+	NaN	NaN
5+	NaN	0.000000
10+	0.250000	0.333333
50+	0.000000	0.000000
100+	0.364286	0.586207
500+	0.081633	0.075472
1,000+	0.429245	0.427230
5,000+	0.282609	0.309524
10,000+	0.487179	0.433225
50,000+	0.228571	0.237037
100,000+	0.506135	0.471429
500,000+	0.282051	0.295302
1,000,000+	0.617587	0.645299
5,000,000+	0.492891	0.454148
10,000,000+	0.714660	0.707254
50,000,000+	0.658824	0.565657
100,000,000+	0.817518	0.861538
500,000,000+	0.724138	0.954545
1,000,000,000+	1.000000	1.000000

4. Precision & recall for Resubstitution

	precision	recall
1+	NaN	0.000000
5+	NaN	0.000000
10+	0.176471	0.230769
50+	0.000000	0.000000
100+	0.324503	0.538462
500+	0.155172	0.147541
1,000+	0.388601	0.333333
5,000+	0.330709	0.302158
10,000+	0.434084	0.465517
50,000+	0.271318	0.220126
100,000+	0.493976	0.469914
500,000+	0.248588	0.293333
1,000,000+	0.593952	0.611111
5,000,000+	0.509709	0.470852
10,000,000+	0.734247	0.694301
50,000,000+	0.592593	0.578313
100,000,000+	0.822222	0.888000
500,000,000+	0.733333	0.846154
1,000,000,000+	0.916667	1.000000

(VI) Conclusion

1. Outcome differs in every execution, due to the random split between testing data and training data
2. Dataset preprocessing is important for outlier data