In [1]:
```
import pandas as pd
import numpy as np
```

In [8]:
```
read=pd.read_csv('iris.csv')
```

In [9]:
```
read.head()
```

Out[9]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [11]:
```
read.isnull().sum()
```

Out[11]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [12]:
```
read.describe()
```

Out[12]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [18]:
```
read.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
```

```
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 6.5+ KB
```

In [ ]:

In [ ]:

Assignment - 2

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import seaborn as sns
```

In [2]:
```
data = pd.read_csv("acad.csv")
```

In [4]:
```
data.head()
```

Out[4]:

	gender	NationalITy	PlaceofBirth	StageID	GradeID	SectionID	Topic	Semester	Relation	raisedh
0	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	Father	
1	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	Father	
2	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	Father	
3	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	Father	
4	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	Father	

In [5]:
```
data.isnull().sum()
```

Out[5]:
```
gender                      0
NationalITy                 0
PlaceofBirth                0
StageID                     0
GradeID                     0
SectionID                   0
Topic                       0
Semester                    0
Relation                    0
raisedhands                 0
VisITedResources            0
AnnouncementsView           0
Discussion                  0
ParentAnsweringSurvey       0
ParentschoolSatisfaction    0
StudentAbsenceDays          0
Class                       0
dtype: int64
```

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 480 entries, 0 to 479
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   gender                    480 non-null    object
 1   NationalITy               480 non-null    object
 2   PlaceofBirth              480 non-null    object
 3   StageID                   480 non-null    object
 4   GradeID                   480 non-null    object
 5   SectionID                 480 non-null    object
```

```
 6   Topic                     480 non-null    object
 7   Semester                  480 non-null    object
 8   Relation                  480 non-null    object
 9   raisedhands               480 non-null    int64
 10  VisITedResources          480 non-null    int64
 11  AnnouncementsView         480 non-null    int64
 12  Discussion                480 non-null    int64
 13  ParentAnsweringSurvey     480 non-null    object
 14  ParentschoolSatisfaction  480 non-null    object
 15  StudentAbsenceDays        480 non-null    object
 16  Class                     480 non-null    object
dtypes: int64(4), object(13)
memory usage: 39.4+ KB
```

In [7]:
```
data.describe()
```

Out[7]:

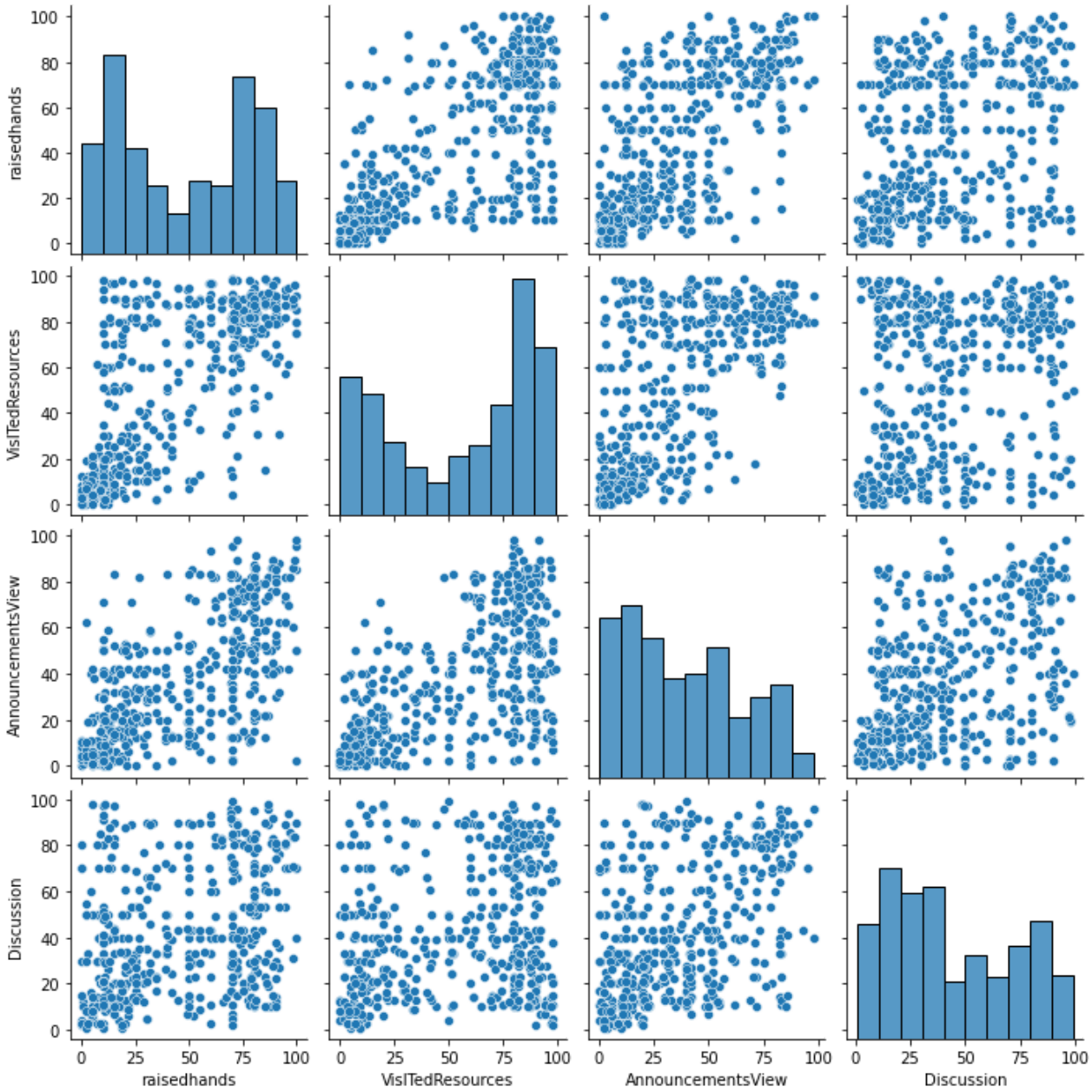
	raisedhands	VisITedResources	AnnouncementsView	Discussion
count	480.000000	480.000000	480.000000	480.000000
mean	46.775000	54.797917	37.918750	43.283333
std	30.779223	33.080007	26.611244	27.637735
min	0.000000	0.000000	0.000000	1.000000
25%	15.750000	20.000000	14.000000	20.000000
50%	50.000000	65.000000	33.000000	39.000000
75%	75.000000	84.000000	58.000000	70.000000
max	100.000000	99.000000	98.000000	99.000000

In [8]:
```
data.shape
```

Out[8]:
```
(480, 17)
```

In [10]:
```
sns.pairplot(data)
```

Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x64cb8f8>
```



In [11]:
```
data['Relation'].unique()
```

Out[11]:
```
array(['Father', 'Mum'], dtype=object)
```

In [16]:
```
data.loc[data['Relation']=="Father","Relation"]="f"
```

In [18]:
```
data.loc[data['Relation']=="Mum","Relation"]="M"
```

In [19]:
```
data
```

Out[19]:

	gender	NationalITy	PlaceofBirth	StageID	GradeID	SectionID	Topic	Semester	Relatior
0	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	I
1	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	I
2	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	I
3	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	I

	gender	NationalITy	PlaceofBirth	StageID	GradeID	SectionID	Topic	Semester	Relatior
4	M	KW	KuwaIT	lowerlevel	G-04	A	IT	F	I
...	...	...	...	...	...	...	...	...	..
475	F	Jordan	Jordan	MiddleSchool	G-08	A	Chemistry	S	I
476	F	Jordan	Jordan	MiddleSchool	G-08	A	Geology	F	I
477	F	Jordan	Jordan	MiddleSchool	G-08	A	Geology	S	I
478	F	Jordan	Jordan	MiddleSchool	G-08	A	History	F	I
479	F	Jordan	Jordan	MiddleSchool	G-08	A	History	S	I

480 rows × 17 columns

In [ ]:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df  = pd.read_csv('iris.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [4]:
```
df.describe()
```

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [5]:
```
df['Species'].unique()
```

Out[5]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

In [6]:
```
setosa = df['Species'] == 'Iris-setosa'
print("**************************Iris-setosa****************************")
print(df[setosa].describe())
```

```
**************************Iris-setosa****************************
             Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  50.00000       50.00000     50.000000      50.000000      50.00000
mean   25.50000        5.00600      3.418000       1.464000       0.24400
std    14.57738        0.35249      0.381024       0.173511       0.10721
min     1.00000        4.30000      2.300000       1.000000       0.10000
25%    13.25000        4.80000      3.125000       1.400000       0.20000
50%    25.50000        5.00000      3.400000       1.500000       0.20000
75%    37.75000        5.20000      3.675000       1.575000       0.30000
max    50.00000        5.80000      4.400000       1.900000       0.60000
```

In [7]:
```
versicolor = df['Species'] == 'Iris-versicolor'
print("**************************Iris-versicolor*****************************")
print(df[versicolor].describe())
```

```
**************************Iris-versicolor*****************************
              Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count   50.00000      50.000000     50.000000      50.000000     50.000000
mean    75.50000       5.936000      2.770000       4.260000      1.326000
std     14.57738       0.516171      0.313798       0.469911      0.197753
min     51.00000       4.900000      2.000000       3.000000      1.000000
25%     63.25000       5.600000      2.525000       4.000000      1.200000
50%     75.50000       5.900000      2.800000       4.350000      1.300000
75%     87.75000       6.300000      3.000000       4.600000      1.500000
max    100.00000       7.000000      3.400000       5.100000      1.800000
```

In [8]:
```
virginica = df['Species'] == 'Iris-virginica'
print("**************************Iris-virginica*****************************")
print(df[versicolor].describe())
```

```
**************************Iris-virginica*****************************
              Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count   50.00000      50.000000     50.000000      50.000000     50.000000
mean    75.50000       5.936000      2.770000       4.260000      1.326000
std     14.57738       0.516171      0.313798       0.469911      0.197753
min     51.00000       4.900000      2.000000       3.000000      1.000000
25%     63.25000       5.600000      2.525000       4.000000      1.200000
50%     75.50000       5.900000      2.800000       4.350000      1.300000
75%     87.75000       6.300000      3.000000       4.600000      1.500000
max    100.00000       7.000000      3.400000       5.100000      1.800000
```

Assignment - 4

In [1]:
```
#MEDV - Median value of owner-occupied homes in $1000's
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import r2_score, mean_squared_error
import klib
```

In [2]:
```
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
df = pd.read_csv('housing.csv', header=None, delimiter=r"\s+", names=column_names)
```

In [3]:
```
df
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MI
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	9.67	
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	9.08	
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	5.64	
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	6.48	
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	7.88	

506 rows × 14 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64  
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    float64
```

```
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

In [5]:
```
df.describe()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500

In [6]:
```
klib.data_cleaning(df) # performs datacleaning (drop duplicates & empty rows/cols, adju
klib.clean_column_names(df) # cleans and standardizes column names, also called inside
klib.convert_datatypes(df) # converts existing to more efficient dtypes, also called in
klib.drop_missing(df) # drops missing values, also called in data_cleaning()
klib.mv_col_handling(df) # drops features with high ratio of missing vals based on info
klib.pool_duplicate_subsets(df)
```

```
Shape of cleaned data: (506, 14)Remaining NAs: 0

Changes:
Dropped rows: 0
     of which 0 duplicates. (Rows: [])
Dropped columns: 0
     of which 0 single valued.     Columns: []
Dropped missing values: 0
Reduced memory by at least: 0.02 MB (-40.0%)
```

Out[6]:

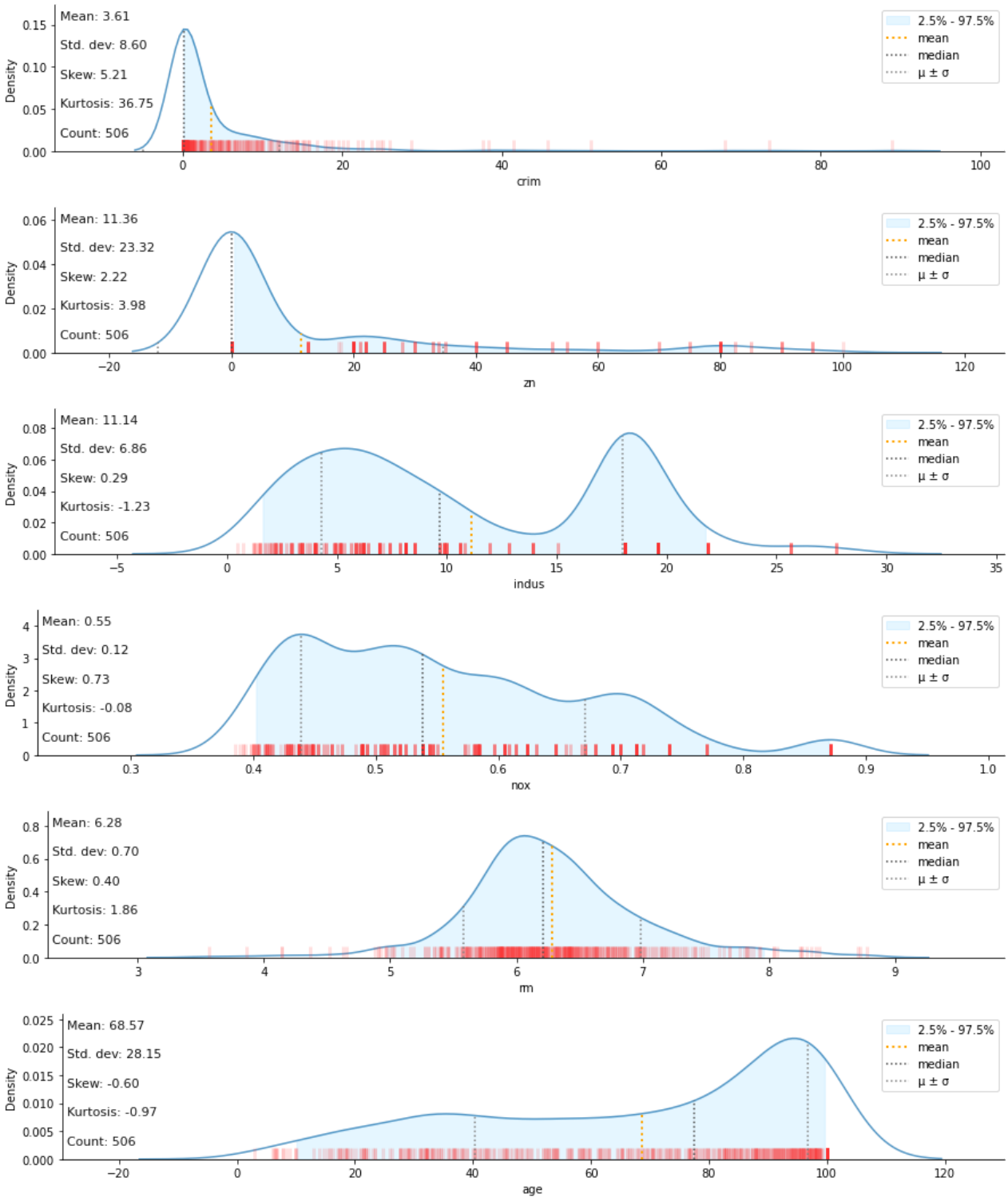
	crim	rm	age	dis	b	lstat	medv	pooled_vars
0	0.00632	6.575	65.2	4.0900	396.90	4.98	24.0	0
1	0.02731	6.421	78.9	4.9671	396.90	9.14	21.6	1
2	0.02729	7.185	61.1	4.9671	392.83	4.03	34.7	1
3	0.03237	6.998	45.8	6.0622	394.63	2.94	33.4	3
4	0.06905	7.147	54.2	6.0622	396.90	5.33	36.2	3
...	...	...	...	...	...	...	...	...
501	0.06263	6.593	69.1	2.4786	391.99	9.67	22.4	501
502	0.04527	6.120	76.7	2.2875	396.90	9.08	20.6	501
503	0.06076	6.976	91.0	2.1675	396.90	5.64	23.9	501

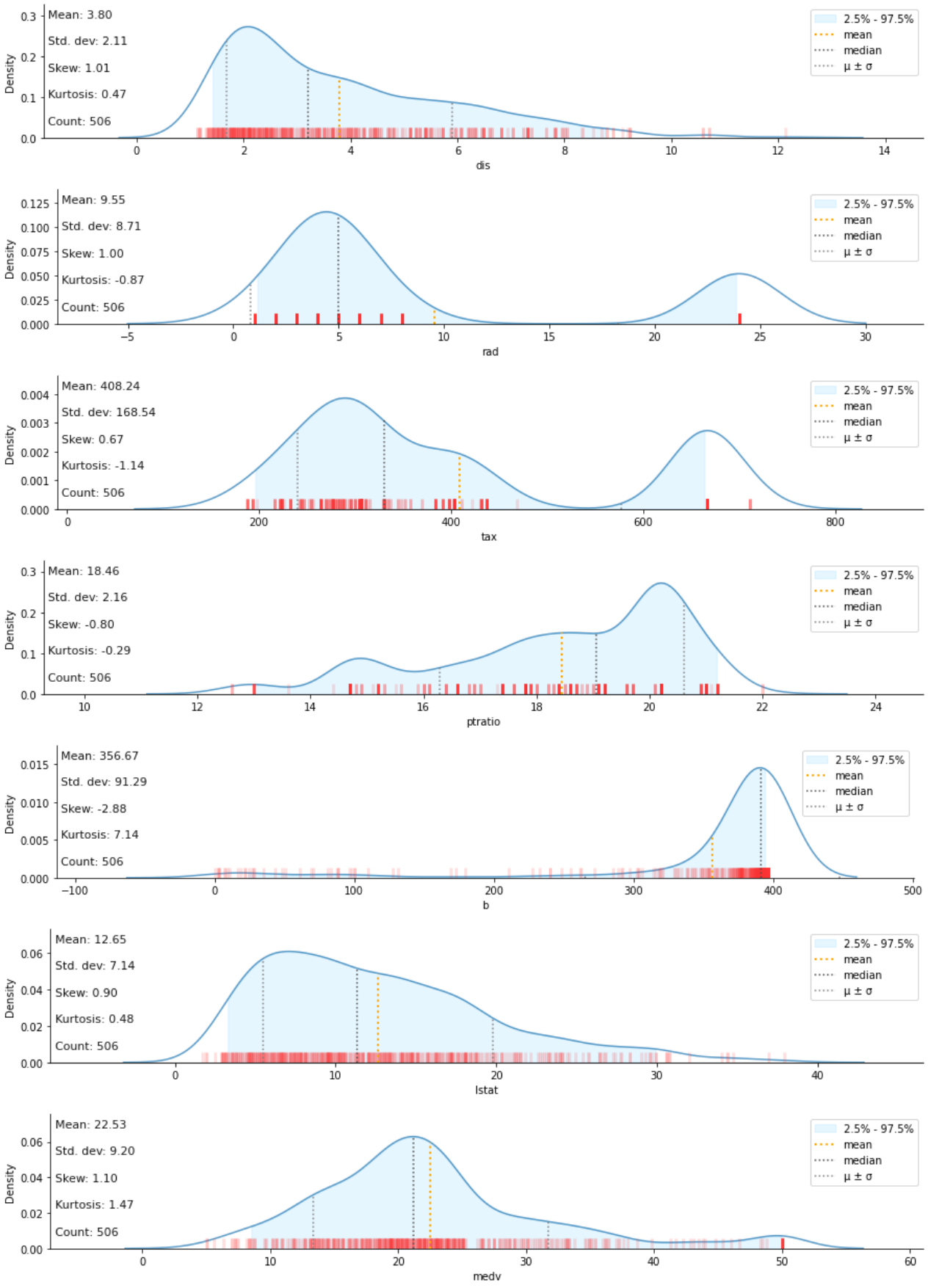
	crim	rm	age	dis	b	lstat	medv	pooled_vars
504	0.10959	6.794	89.3	2.3889	393.45	6.48	22.0	501
505	0.04741	6.030	80.8	2.5050	396.90	7.88	11.9	501

506 rows × 8 columns

In [7]:
```
klib.dist_plot(df)
```

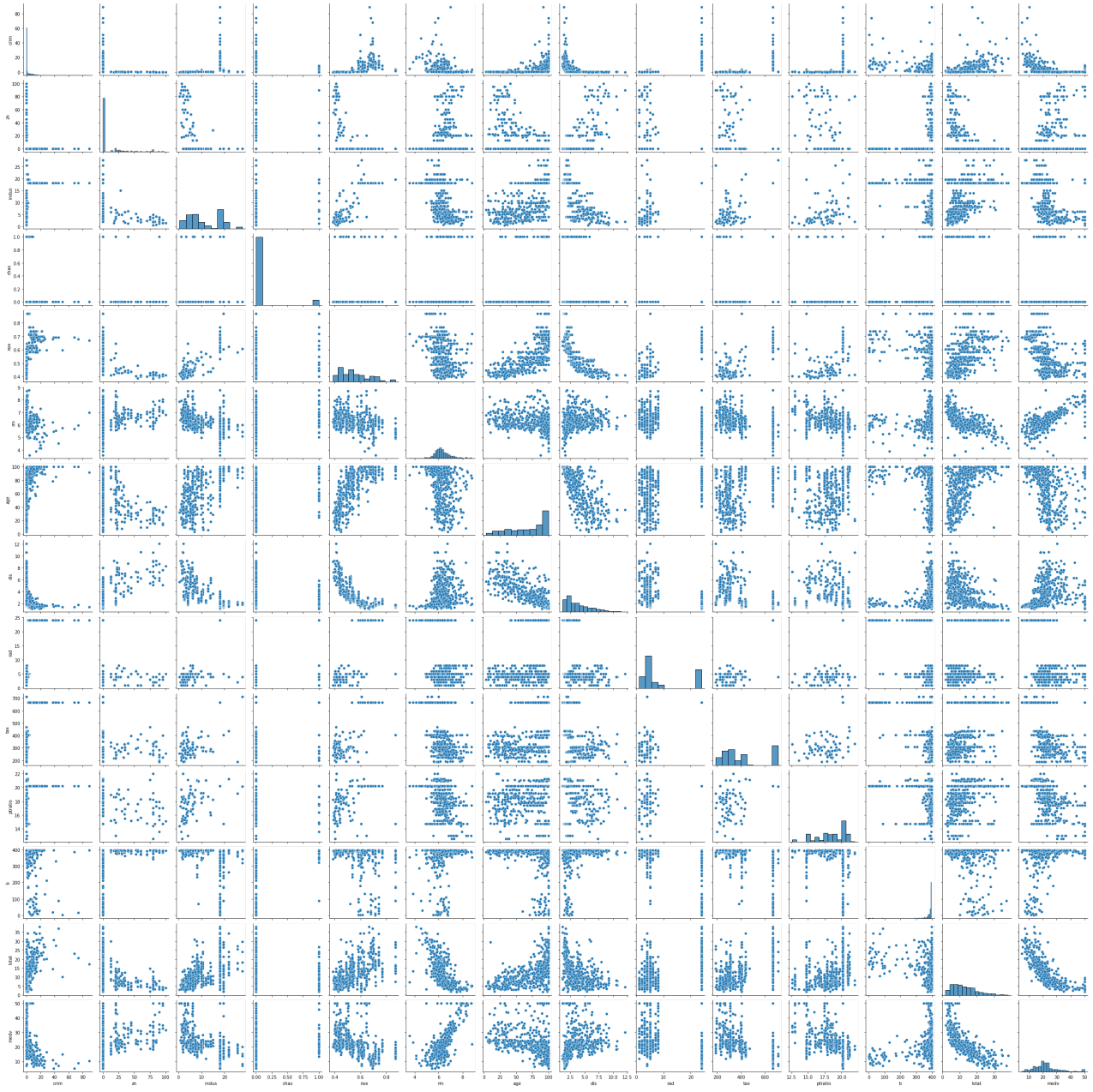
Out[7]: <AxesSubplot:xlabel='medv', ylabel='Density'>





In [8]:
```
sns.pairplot(data=df)
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x1e935904a90>
```



In [9]:
```
X = df.drop(['medv'], axis = 1)
y = df['medv']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state
```

In [10]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
y_pred_train = model.predict(X_train)
print("R2_org:", r2_score(y_train, y_pred_train))
print("MSE_org:", mean_squared_error(y_train, y_pred_train))
print("Accurcy:", model.score(X_train, y_train))

y_pred_test = model.predict(X_test)
print("R2_org:", r2_score(y_test, y_pred_test))
print("MSE_org:", mean_squared_error(y_test, y_pred_test))
print("Accurcy:", model.score(X_test, y_test))
```

```
R2_org: 0.7103879080674731
```

```
MSE_org: 23.513334449327022
Accurcy: 0.7103879080674731
R2_org: 0.7836295385076278
MSE_org: 19.831323672063256
Accurcy: 0.7836295385076278
```

In [11]:
```
test = np.array([0.02729,0.0,7.07,0,0.469,7.185,61.1,4.9671,2,242.0,17.8,392.83,4.03])
```

In [12]:
```
test = test.reshape((1,-1))
```

In [13]:
```
model.predict(test)
```

```
C:\Users\sanke\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.p
y:450: UserWarning: X does not have valid feature names, but LinearRegression was fitted
with feature names
  warnings.warn(
```

Out[13]:
```
array([29.74702904])
```

In [ ]:

In [ ]:

assignment - 5

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
```

In [2]:
```
data = pd.read_csv("Social_Network_Ads.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [4]:
```
# Here User ID is not suitable to predict the results,so we are ignore this coloumn.

data = data[['Gender','Age','EstimatedSalary','Purchased']]
```

In [5]:
```
print(data.head())
```

```
   Gender  Age  EstimatedSalary  Purchased
0    Male   19            19000          0
1    Male   35            20000          0
2  Female   26            43000          0
3  Female   27            57000          0
4    Male   19            76000          0
```

In [6]:
```
data.isnull().sum()
```

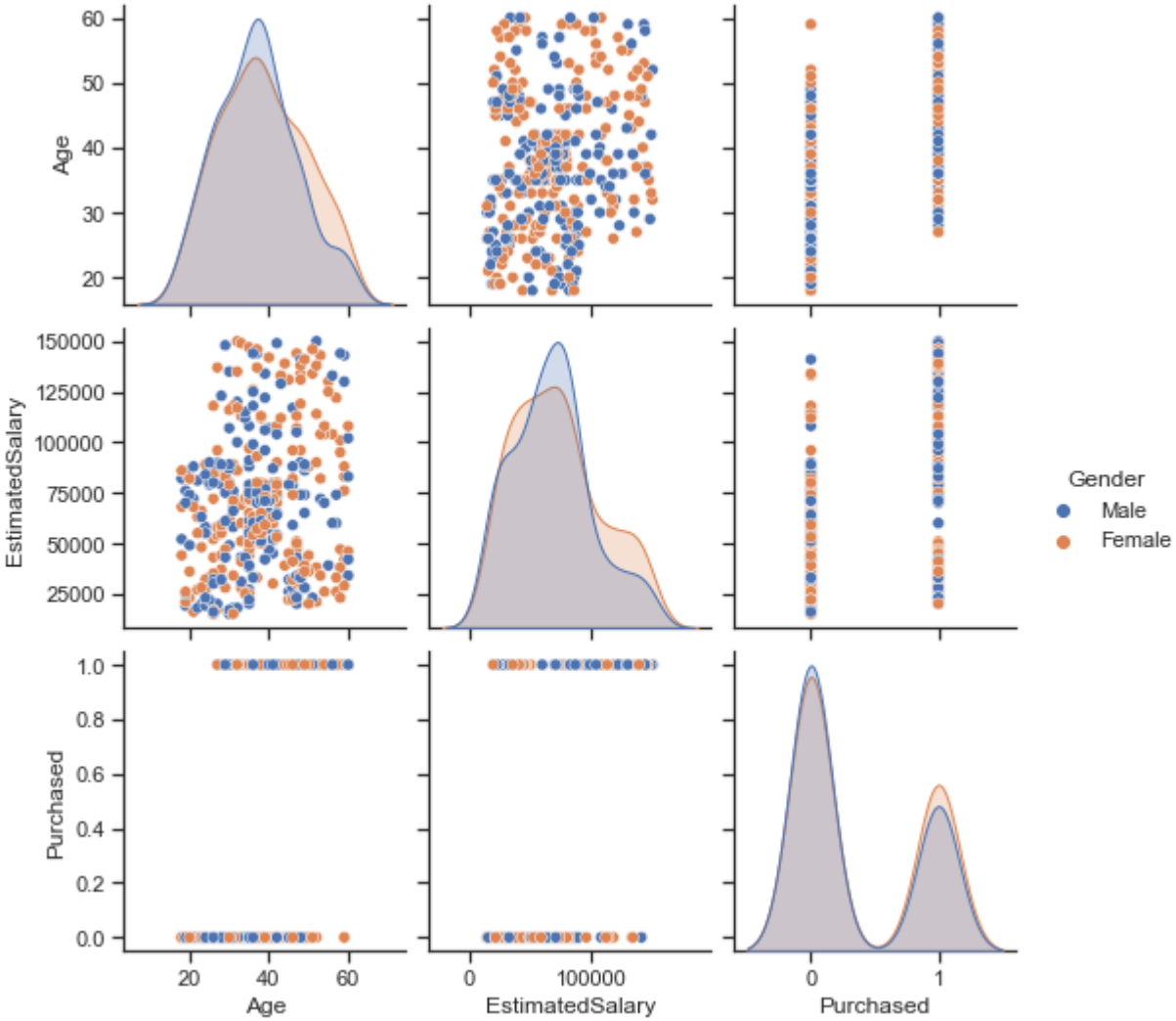
Out[6]:
```
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

In [7]:
```
data.plot()
```

Out[7]:
```
<AxesSubplot:>
```

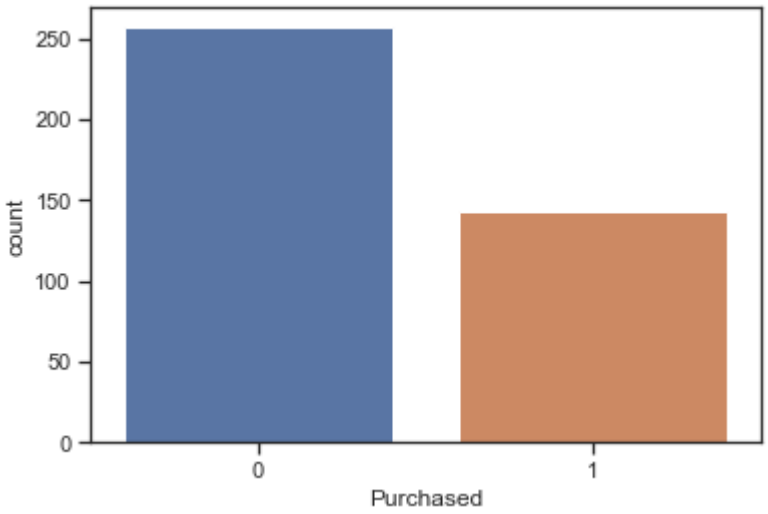
In [8]:
```
import seaborn as sns
sns.set(style="ticks")

sns.pairplot(data, hue="Gender")
```

Out[8]: <seaborn.axisgrid.PairGrid at 0x11d82838>



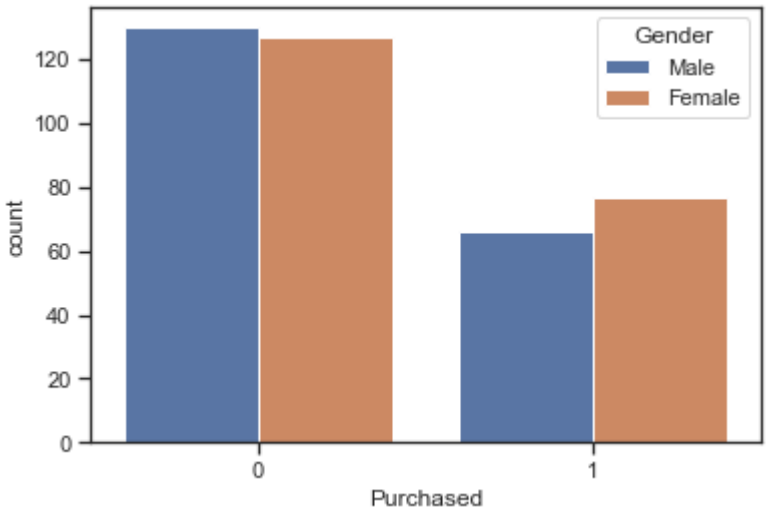
In [9]:
```
# Here We Check The Total No. Who Purchased or Not Purchased
sns.countplot(x="Purchased",data=data)
```

Out[9]: <AxesSubplot:xlabel='Purchased', ylabel='count'>



In [10]:
```
# As We See here mostly female's like to buy product then male's
sns.countplot(x="Purchased",hue="Gender",data=data)
```

Out[10]: <AxesSubplot:xlabel='Purchased', ylabel='count'>



In [11]:
```
# Now Lets Convert The Variables into dummy variabels for our ML model.
# If the Value of 1 in Male Then i.e male if value is 1 in Female then i.e Female
pd.get_dummies(data['Gender'])
```

Out[11]:

	Female	Male
0	0	1
1	0	1
2	1	0
3	1	0
4	0	1
...	...	...
395	1	0
396	0	1

	Female	Male
397	1	0
398	0	1
399	1	0

400 rows × 2 columns

In [12]:
```
sex = pd.get_dummies(data['Gender'],drop_first=True)
sex.head()
```

Out[12]:

	Male
0	1
1	1
2	0
3	0
4	1

In [13]:
```
data_p = pd.concat([data,sex],axis=1)
```

In [14]:
```
data_p.head()
```

Out[14]:

	Gender	Age	EstimatedSalary	Purchased	Male
0	Male	19	19000	0	1
1	Male	35	20000	0	1
2	Female	26	43000	0	0
3	Female	27	57000	0	0
4	Male	19	76000	0	1

In [15]:
```
# Now There is a Gender Column which we do neet further because we have converted into
data_p = data_p.drop(['Gender'],axis=1)
```

In [16]:
```
data_p.head()
```

Out[16]:

	Age	EstimatedSalary	Purchased	Male
0	19	19000	0	1
1	35	20000	0	1
2	26	43000	0	0
3	27	57000	0	0
4	19	76000	0	1

Now Lets Split The Data

In [17]:
```
X = data_p[['Age','EstimatedSalary','Male']].values
y = data_p['Purchased'].values
```

In [18]:
```
# Now Train our Data
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=0,test_size=0.3)
```

In [19]:
```
# Preprocessing
from sklearn.preprocessing import StandardScaler
s = StandardScaler()
X_train = s.fit_transform(X_train)
X_test = s.fit_transform(X_test)
```

In [20]:
```
from sklearn.linear_model import LogisticRegression
```

In [21]:
```
log = LogisticRegression()
log.fit(X_train,y_train)
predict = log.predict(X_test)
```

Now Lets See The Accuracy Of our model

In [22]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,predict))
```

```
[[74  5]
 [ 8 33]]
```

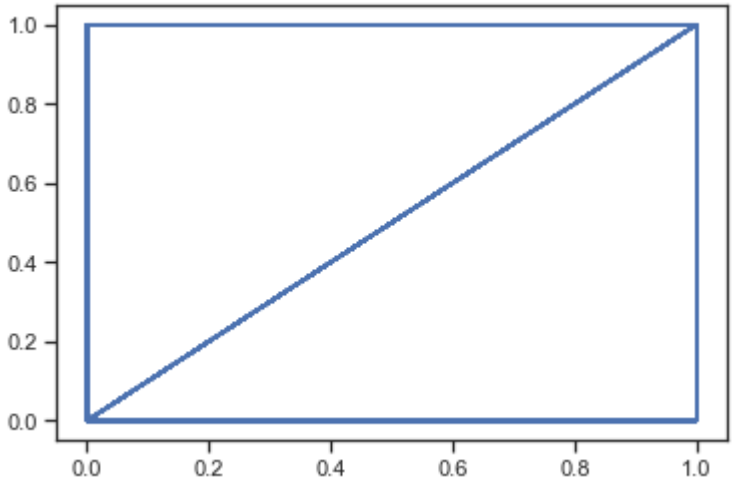
In [23]:
```
print(classification_report(y_test,predict))
```

```
              precision    recall  f1-score   support

           0       0.90      0.94      0.92        79
           1       0.87      0.80      0.84        41

    accuracy                           0.89       120
   macro avg       0.89      0.87      0.88       120
weighted avg       0.89      0.89      0.89       120
```

In [24]:
```
tn, fp, fn, tp = confusion_matrix(y_test,predict).ravel()
print(tn, fp, fn, tp)
```

```
74 5 8 33
```

In [25]:
```
plt.plot(y_test,predict)
plt.show()
```



In [ ]:

Assignment - 6

Importing pandas,numpy,matplotlib and Seaborn module

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing Iris data set

In [2]:
```
iris=pd.read_csv('Iris.csv')
```

Displaying data

In [3]:
```
iris.head()
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [4]:
```
iris['Species'].unique()
```

Out[4]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

This data set has three varities of Iris plant.

In [5]:
```
iris.describe(include='all')
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
count	150.000000	150.000000	150.000000	150.000000	150.000000	150
unique	NaN	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	NaN	Iris-setosa
freq	NaN	NaN	NaN	NaN	NaN	50
mean	75.500000	5.843333	3.054000	3.758667	1.198667	NaN
std	43.445368	0.828066	0.433594	1.764420	0.763161	NaN
min	1.000000	4.300000	2.000000	1.000000	0.100000	NaN
25%	38.250000	5.100000	2.800000	1.600000	0.300000	NaN
50%	75.500000	5.800000	3.000000	4.350000	1.300000	NaN
75%	112.750000	6.400000	3.300000	5.100000	1.800000	NaN
max	150.000000	7.900000	4.400000	6.900000	2.500000	NaN

In [6]:
```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

Removing the unneeded column

In [7]:
```
iris.drop(columns="Id",inplace=True)
```

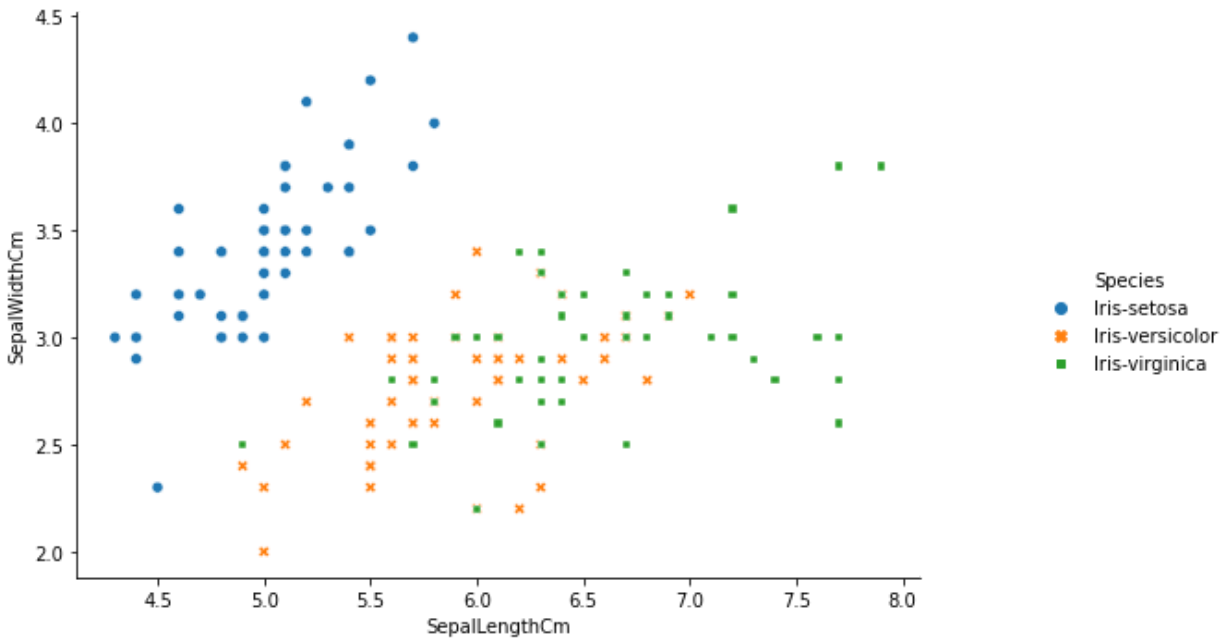
In [8]:
```
iris.isnull().sum()
```

Out[8]:
```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

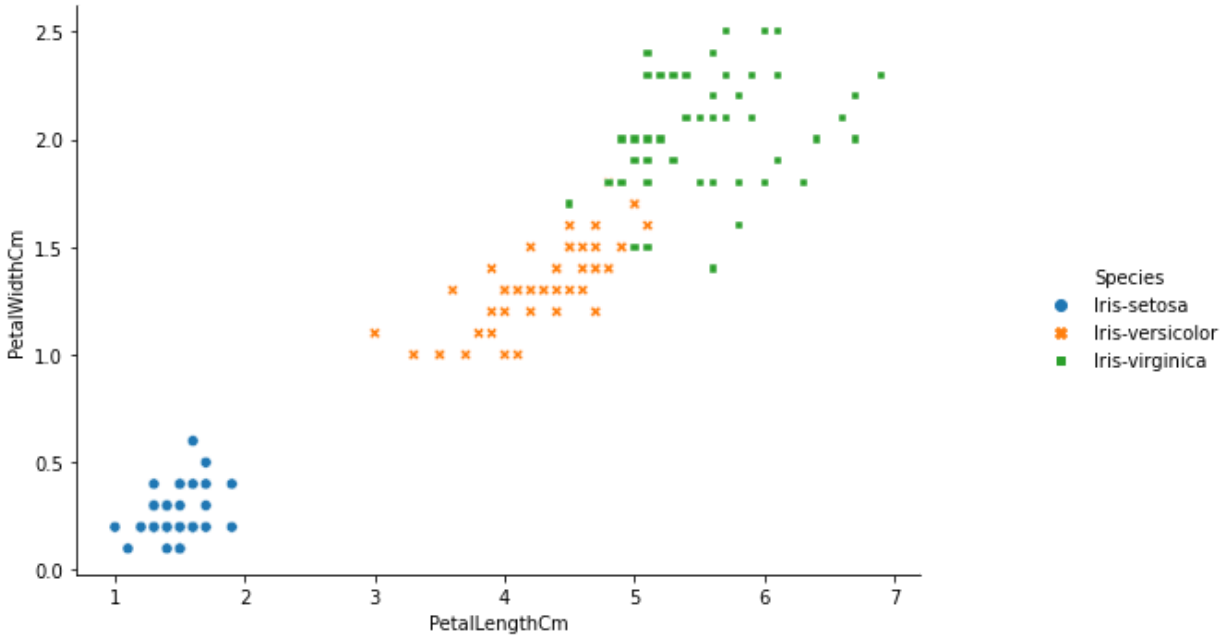
There is not any missing value in this dataset

2. Data Visualization

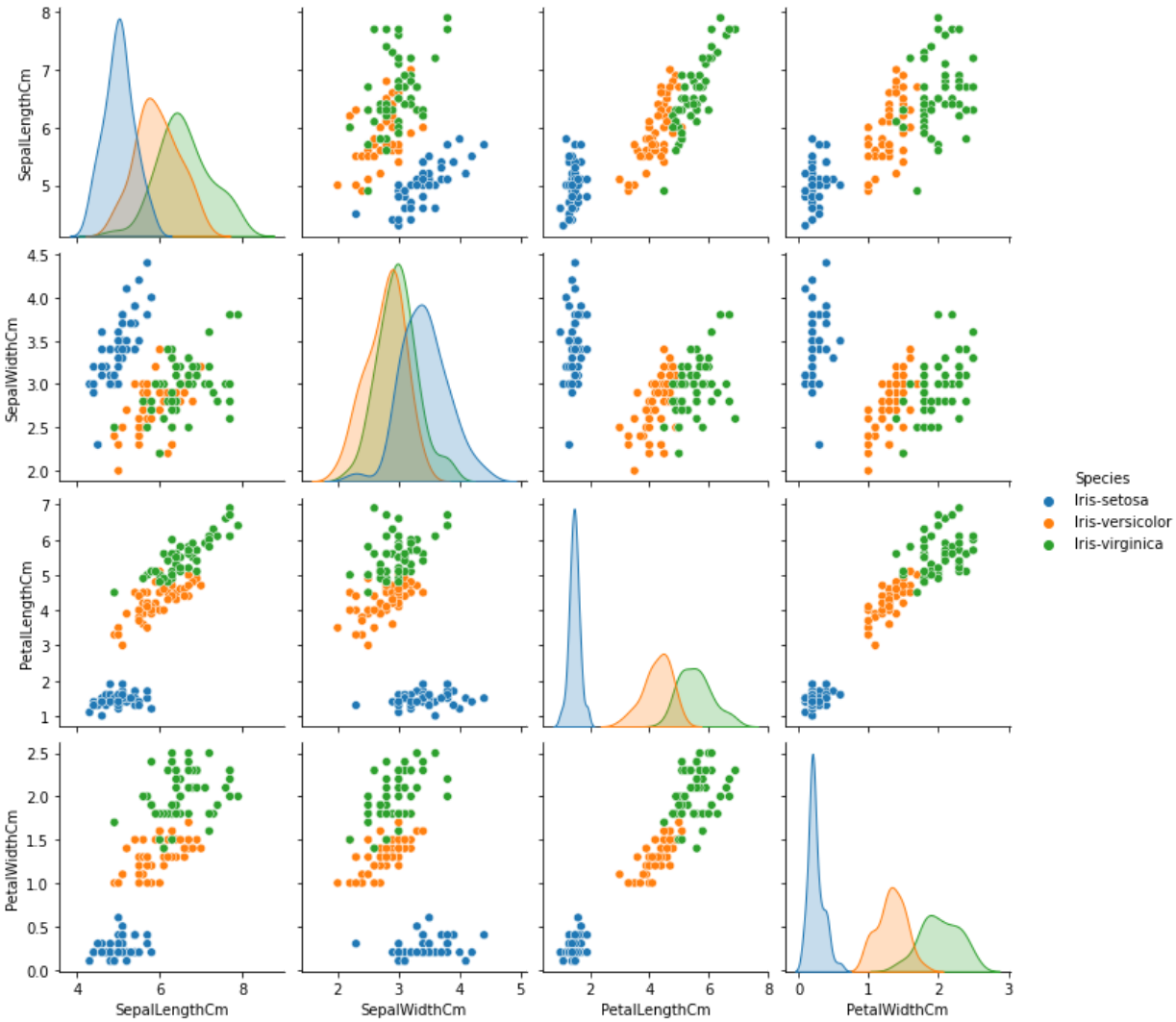
In [9]:
```
g=sns.relplot(x='SepalLengthCm',y='SepalWidthCm',data=iris,hue='Species',style='Specie
g.fig.set_size_inches(10,5)
plt.show()
```



In [10]:
```
g=sns.relplot(x='PetalLengthCm',y='PetalWidthCm',data=iris,hue='Species',style='Specie
g.fig.set_size_inches(10,5)
plt.show()
```



In [11]:
```
sns.pairplot(iris,hue="Species")
plt.show()
```

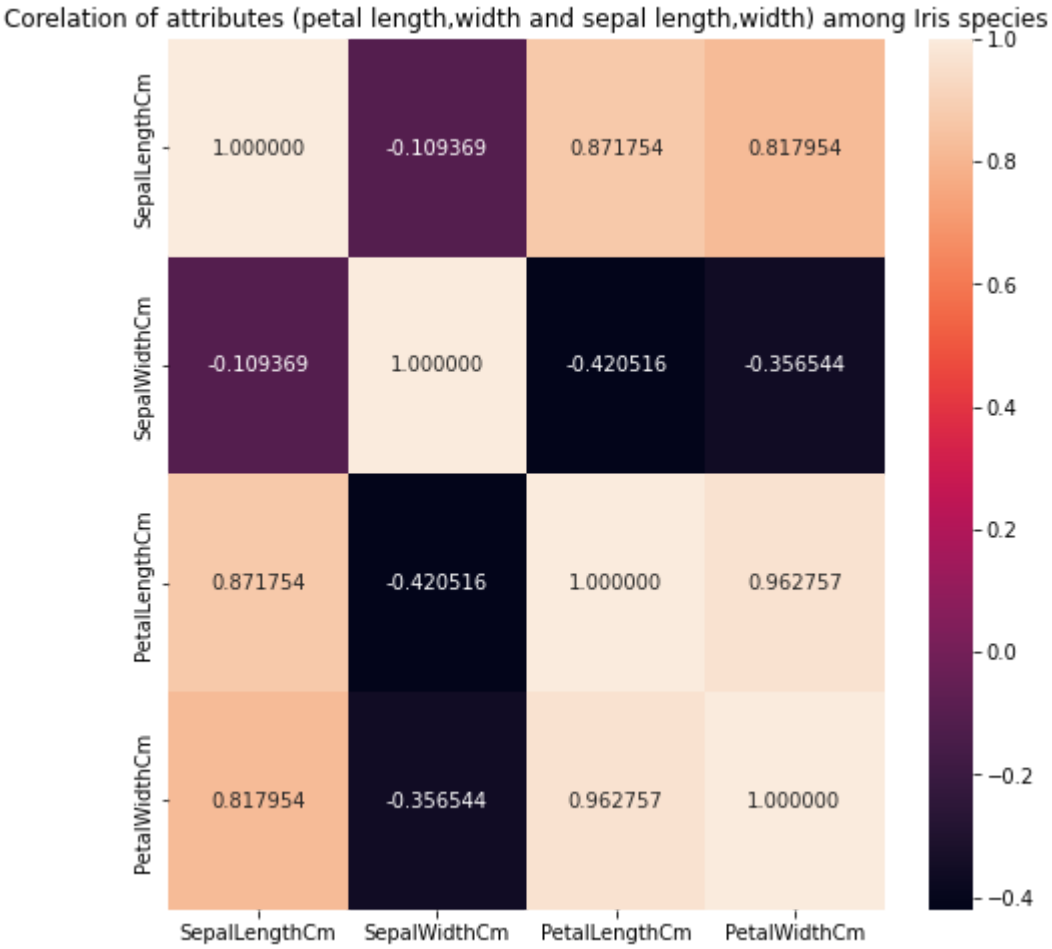


In [13]:
```
iris.corr()
```

Out[13]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

In [14]:
```
plt.subplots(figsize = (8,8))
sns.heatmap(iris.corr(),annot=True,fmt="f").set_title("Corelation of attributes (petal
plt.show()
```



Observation--->

The Sepal Width and Length are not correlated The Petal Width and Length are highly correlated

We will use all the features for training the algorithm and check the accuracy.

In [15]:
```
X=iris.iloc[:,0:4].values
y=iris.iloc[:,4].values
```

In [16]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

In [20]:
```
#Metrics
from sklearn.metrics import make_scorer, accuracy_score,precision_score ,confusion_mat
from sklearn.metrics import accuracy_score ,precision_score,recall_score,f1_score

#Model Select
from sklearn.model_selection import KFold,train_test_split,cross_val_score
from sklearn.naive_bayes import GaussianNB
```

In [21]:
```
#Train and Test split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

Gaussian Naive Bayes:

In [27]:
```
gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
Y_pred = gaussian.predict(X_test)
accuracy_nb=round(accuracy_score(y_test,Y_pred)* 100, 2)
acc_gaussian = round(gaussian.score(X_train, y_train) * 100, 2)

cm = confusion_matrix(y_test, Y_pred)
accuracy = accuracy_score(y_test,Y_pred)
precision =precision_score(y_test, Y_pred,average='micro')
recall =  recall_score(y_test, Y_pred,average='micro')
f1 = f1_score(y_test,Y_pred,average='micro')
print('Confusion matrix for Naive Bayes\n',cm)
print('accuracy_Naive Bayes: %.3f' %accuracy)
print('precision_Naive Bayes: %.3f' %precision)
print('recall_Naive Bayes: %.3f' %recall)
print('f1-score_Naive Bayes : %.3f' %f1)
```

```
Confusion matrix for Naive Bayes
 [[16  0  0]
 [ 0 18  0]
 [ 0  0 11]]
accuracy_Naive Bayes: 1.000
precision_Naive Bayes: 1.000
recall_Naive Bayes: 1.000
f1-score_Naive Bayes : 1.000
```

In [31]:
```
tn, fp, fn, tp = confusion_matrix(list(y_test), list(Y_pred), labels=[0, 1]).ravel()
```

In [34]:
```
print("tn :",tn)
print("fp :",fp)
print("fn :",fn)
print("tp :",tp)
```

```
tn : 16
fp : 0
fn : 0
tp : 18
```

In [ ]:
```

```

In [1]:

```
import pandas as pd
import sklearn as sk
import math
```

In [2]:

```
import docx
```

In [3]:

```
pwd
```

Out[3]:

```
'C:\\Users\\Tej'
```

In [4]:

```
cd E:\
```

```
E:\
```

In [5]:

```
pip install python-docx
```

```
Requirement already satisfied: python-docx in c:\users\tej\anaconda3\lib\sit
e-packages (0.8.11)
Requirement already satisfied: lxml>=2.3.2 in c:\users\tej\anaconda3\lib\sit
e-packages (from python-docx) (4.5.2)
Note: you may need to restart the kernel to use updated packages.
```

In [6]:

```
document = docx.Document('Sample.docx')
```

In [7]:

```
print(document.paragraphs[0].text)
```

```
Hello this is class of TE student Div1
```

In [8]:

```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Error loading punkt: <urlopen error [WinError 10054] An
[nltk_data]     existing connection was forcibly closed by the remote
[nltk_data]     host>
```

Out[8]:

```
False
```

In [9]:

```
word = "It originated from the idea that there are readers who prefer learning new skills f
nltk_tokens = nltk.word_tokenize(word)
print(nltk_tokens)
```

```
['It', 'originated', 'from', 'the', 'idea', 'that', 'there', 'are', 'reader
s', 'who', 'prefer', 'learning', 'new', 'skills', 'from', 'the', 'comforts',
'of', 'their', 'drawing', 'rooms']
```

In [10]:

```
word.split()
```

Out[10]:

```
['It',
 'originated',
 'from',
 'the',
 'idea',
 'that',
 'there',
 'are',
 'readers',
 'who',
 'prefer',
 'learning',
 'new',
 'skills',
 'from',
 'the',
 'comforts',
 'of',
 'their',
 'drawing',
 'rooms']
```

In [11]:

```
from nltk import pos_tag
from nltk import RegexpParser
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Tej\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[11]:

```
True
```

In [12]:

```
word1 = "Learn from IIT and make Life easy".split()
print("After Split:- ", word1)
```

```
After Split:-  ['Learn', 'from', 'IIT', 'and', 'make', 'Life', 'easy']
```

In [13]:

```
token_tags = pos_tag(word1)
print("After Tokenization:- ", token_tags)
```

```
After Tokenization:-  [('Learn', 'NNP'), ('from', 'IN'), ('IIT', 'NNP'), ('a
nd', 'CC'), ('make', 'VB'), ('Life', 'NNP'), ('easy', 'JJ')]
```

In [14]:

```
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Tej\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [15]:

```
text = "Nick likes to play football, however he is not too fond of tennis."
text_tokens = word_tokenize(text)

tokens_without_sw = [word for word in text_tokens if not word in stopwords.words()]

print(tokens_without_sw)
```

```
['Nick', 'likes', 'play', 'football', ',', 'however', 'fond', 'tennis', '.']
```

In [16]:

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
```

In [17]:

```
ps = PorterStemmer()

sentence = "Programmers program with programming languages"

words = word_tokenize(sentence)

for w in words:
    print(w, " : ", ps.stem(w))
```

```
Programmers  :  programm
program  :  program
with  :  with
programming  :  program
languages  :  languag
```

In [18]:

```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Tej\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[18]:

```
True
```

In [19]:

```
word3 = WordNetLemmatizer()

print("rocks :", word3.lemmatize("rocks"))
print("corpora :", word3.lemmatize("corpora"))

print("better :", word3.lemmatize("better", pos ="a"))
```

```
rocks : rock
corpora : corpus
better : good
```

In [20]:

```
import pandas as pd
import sklearn as sk
import math
```

In [21]:

```
first_sentence = "Data Science is the sexiest job of the 21st century"
second_sentence = "machine learning is the key for data science"

first_sentence = first_sentence.split(" ")

second_sentence = second_sentence.split(" ")

total= set(first_sentence).union(set(second_sentence))
print(total)
```

```
{'Science', 'Data', 'data', 'key', '21st', 'the', 'of', 'sexiest', 'centur
y', 'is', 'machine', 'science', 'for', 'job', 'learning'}
```

In [22]:

```
wordDictA = dict.fromkeys(total, 0)
wordDictB = dict.fromkeys(total, 0)

for word in first_sentence:
    wordDictA[word]+=1

for word in second_sentence:
    wordDictB[word]+=1
```

In [23]:

```
pd.DataFrame([wordDictA, wordDictB])
```

Out[23]:

	Science	Data	data	key	21st	the	of	sexiest	century	is	machine	science	for	job	le
0	1	1	0	0	1	2	1	1	1	1	0	0	0	1	
1	0	0	1	1	0	1	0	0	0	1	1	1	1	0	

In [25]:

```
def computeTF(wordDict, doc):
    tfDict = {}
    corpusCount = len(doc)

    for word, count in wordDict.items():
        tfDict[word] = count/float(corpusCount)
        return(tfDict)

tfFirst = computeTF(wordDictA, first_sentence)
tfSecond = computeTF(wordDictB, second_sentence)

pd.DataFrame([tfFirst, tfSecond])
```

Out[25]:

	Science
0	0.1
1	0.0

In [29]:

```
def computeIDF(docList):
    idfDict = {}
    N = len(docList)

    idfDict = dict.fromkeys(docList[0].keys(), 0)

    for word, val in idfDict.items():
        idfDict[word] = math.log10(N / (float(val) + 1))

    return(idfDict)


idfs = computeIDF([wordDictA, wordDictB])
```

In [30]:

```
def computeTFIDF(tfBow, idfs):
    tfidf = {}

    for word, val in tfBow.items():
        tfidf[word] = val*idfs[word]
    return(tfidf)

idfFirst = computeTFIDF(tfFirst, idfs)
idfSecond = computeTFIDF(tfSecond, idfs)

idf= pd.DataFrame([idfFirst, idfSecond])

print(idf)
```

```
    Science
0  0.030103
1  0.000000
```

Assignment - 10

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv('iris.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [4]:
```
column = len(list(df))
column
```

Out[4]:
```
6
```

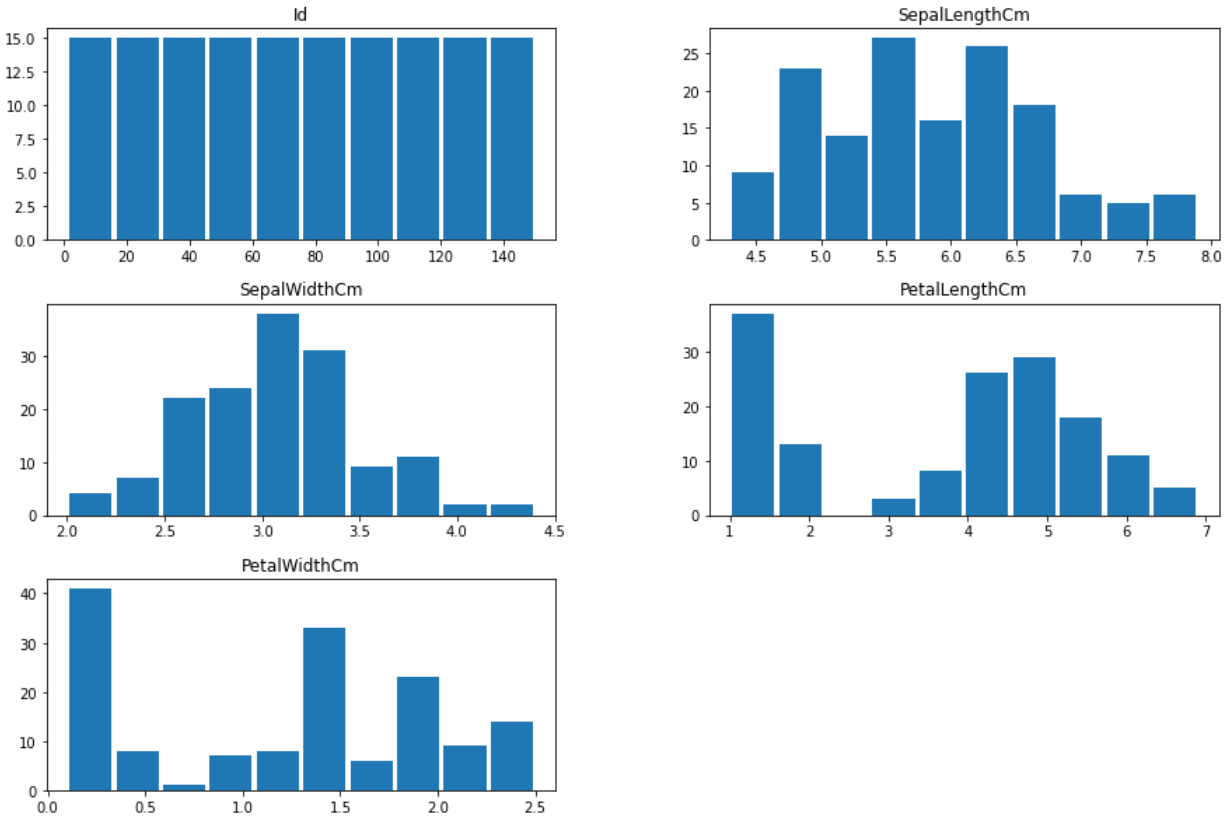
In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

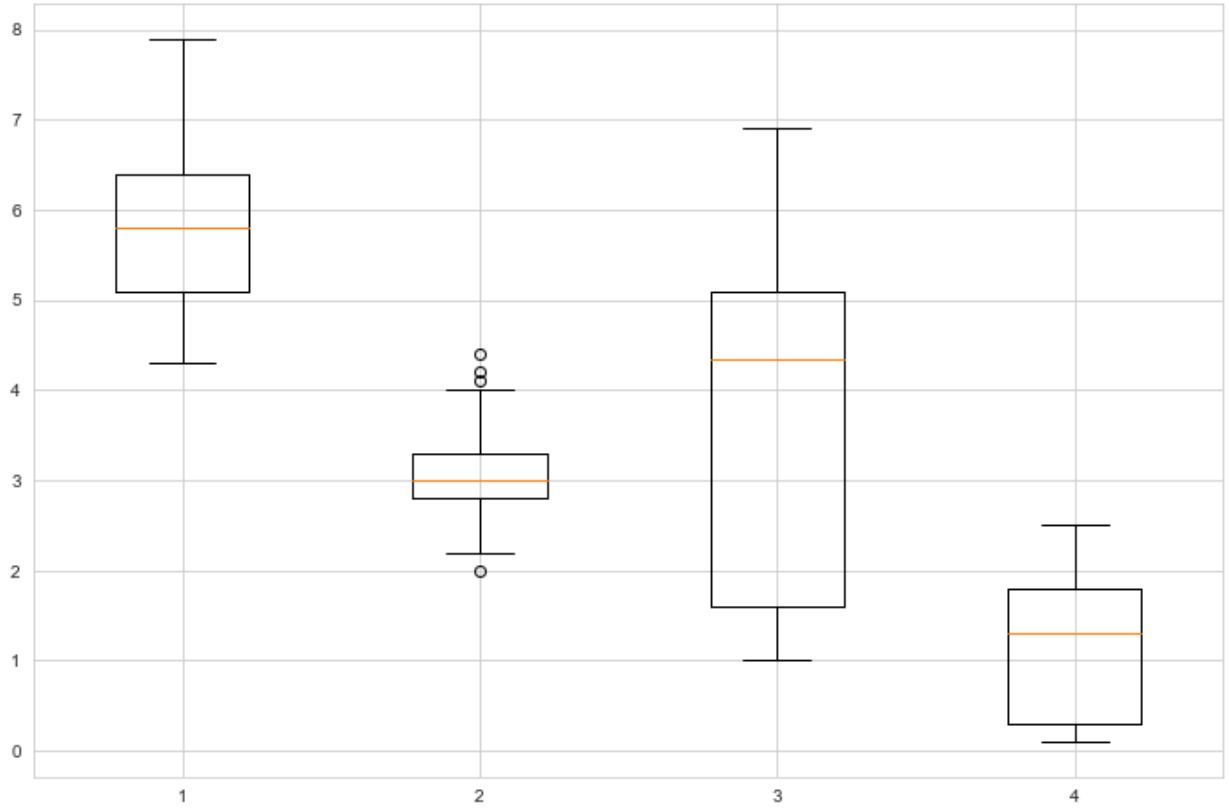
Visualisation

In [6]:
```
df.hist(figsize=(15,10),grid=False,zorder=2, rwidth=0.9)
```

Out[6]:
```
array([[<AxesSubplot:title={'center':'Id'}>,
        <AxesSubplot:title={'center':'SepalLengthCm'}>],
       [<AxesSubplot:title={'center':'SepalWidthCm'}>,
        <AxesSubplot:title={'center':'PetalLengthCm'}>],
       [<AxesSubplot:title={'center':'PetalWidthCm'}>, <AxesSubplot:>]],
      dtype=object)
```



In [7]:
```
data_to_plot = [df["SepalLengthCm"],df["SepalWidthCm"],df["PetalLengthCm"],df["PetalWi

sns.set_style("whitegrid")
# Creating a figure instance
fig = plt.figure(1, figsize=(12,8))

# Creating an axes instance
ax = fig.add_subplot(111)

# Creating the boxplot
bp = ax.boxplot(data_to_plot);
```



If we observe closely. for the box 2, interquartile distance is roughly around 0.75 hence the values lying beyond this range of (third quartile + interquartile distance) i.e. roughly around 4.05 will be considered as outliers. Similarly outliers with other boxplots can be found.