Sheet 1 - Introduction

By: Genivika Mann, Jonah, Fatemeh Salehi, Helia Salimi, Fabrice Beaumont

Ex1 - Descriptive Statistics & Data Visualization

1. Load the Iris dataset into your notebook from Scikit-Learn.

In [1]:
```
from sklearn.datasets import load_iris
import pandas as pd
iris_dataset = load_iris()

### Features & target labels
print( 'Feature Names:', iris_dataset.feature_names )
print( 'Target Names:', iris_dataset.target_names )
```

```
Feature Names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']
Target Names: ['setosa' 'versicolor' 'virginica']
```

2. Report the descriptive statistics of the features of the iris dataset.

In [2]:
```
### The following description returns some statistics, but not all desired ones
# print(data.DESCR)
# ============== ==== ==== ======= ===== ====================
#                 Min  Max   Mean    SD   Class Correlation
# ============== ==== ==== ======= ===== ====================
# sepal length:   4.3  7.9   5.84   0.83    0.7826
# sepal width:    2.0  4.4   3.05   0.43   -0.4194
# petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
# petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
# ============== ==== ==== ======= ===== ====================
```

In [3]:
```
### Lets use Pandas dataframes to get the data more easily
data, target = load_iris(return_X_y=True,as_frame=True)

### Shape of dataset
print( 'Number of samples:\t', data.shape[0] )
print( 'Number of attributes:\t', data.shape[1] )
```

```
Number of samples:       150
Number of attributes:    4
```

In [4]:
```
### Lets get an overview:
data.describe()
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
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

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

a. Mean, Median, Mode

In [5]:
```
### Besides this, we can use many built in functions to receive the desired values
data.mean()
```

Out[5]:
```
sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
dtype: float64
```

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

Out[6]:
```
sepal length (cm)    5.80
sepal width (cm)     3.00
petal length (cm)    4.35
petal width (cm)     1.30
dtype: float64
```

In [7]:
```
### The mode of a set of values is the value that appears most often.
data.mode()
```

Out[7]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.0	3.0	1.4	0.2
1	NaN	NaN	1.5	NaN

b. Variance, MAD, Standard deviation

In [8]:
```
data.var()
```

Out[8]:
```
sepal length (cm)    0.685694
sepal width (cm)     0.189979
petal length (cm)    3.116278
petal width (cm)     0.581006
dtype: float64
```

In [9]:
```
### The MAD is the Mean Absolute Deviation of the values over the requested axis.
data.mad()
```

Out[9]:
```
sepal length (cm)    0.687556
sepal width (cm)     0.336782
petal length (cm)    1.562747
petal width (cm)     0.658133
dtype: float64
```

In [10]:
```
data.std()
```

Out[10]:
```
sepal length (cm)    0.828066
sepal width (cm)     0.435866
petal length (cm)    1.765298
petal width (cm)     0.762238
dtype: float64
```

c. Quantiles, IQR

In [11]:
```
data.quantile()
```

Out[11]:
```
sepal length (cm)    5.80
sepal width (cm)     3.00
petal length (cm)    4.35
petal width (cm)     1.30
Name: 0.5, dtype: float64
```

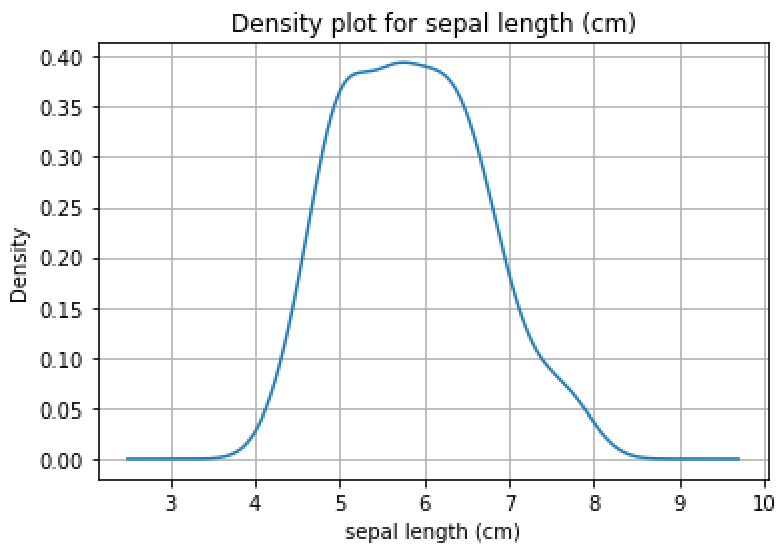
In [12]:
```
### By definition the IQR is the interquartile range and can be computed as
### the difference between 75th and 25th percentiles
data.quantile(0.75) - data.quantile(0.25)
```

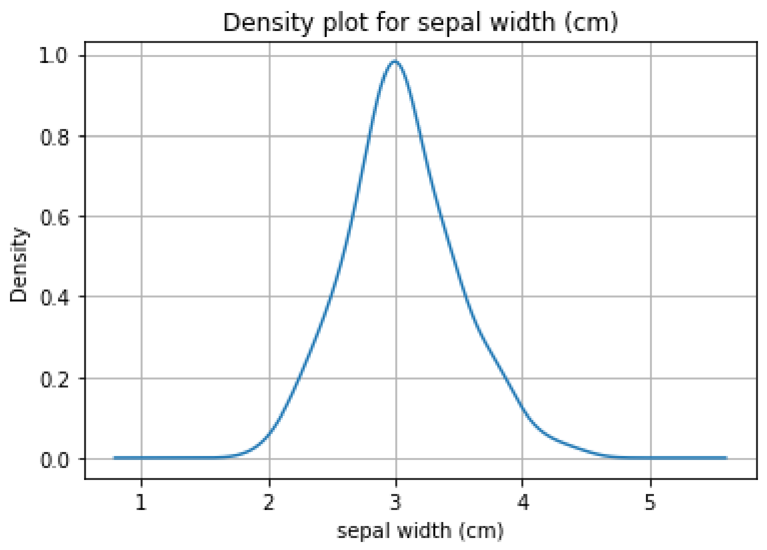
Out[12]:
```
sepal length (cm)    1.3
sepal width (cm)     0.5
petal length (cm)    3.5
petal width (cm)     1.5
dtype: float64
```

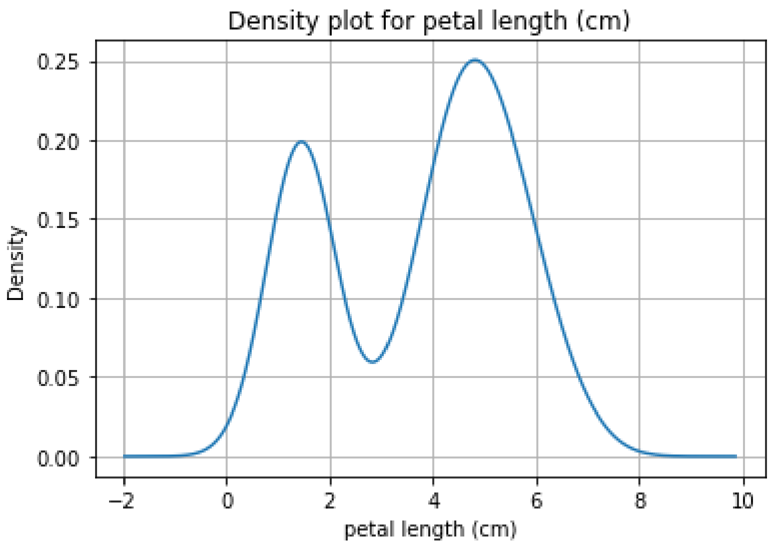
3. Plot a density plot for each of the variables. Interpret the plots.

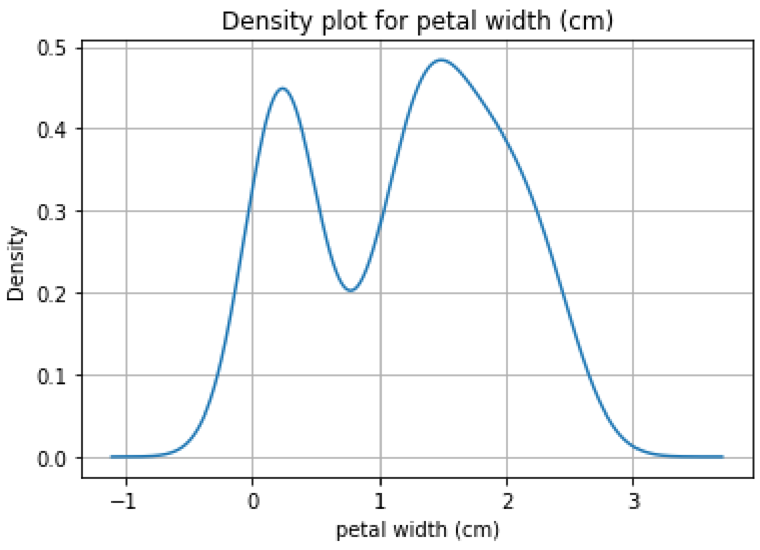
In [13]:
```
import matplotlib.pyplot as plt

for column_name in data.columns:
    feature_dataframe = data[column_name]
    feature_dataframe.plot.density(grid=True)
    plt.title('Density plot for '+ column_name)
    plt.xlabel(column_name)
    plt.show()
```









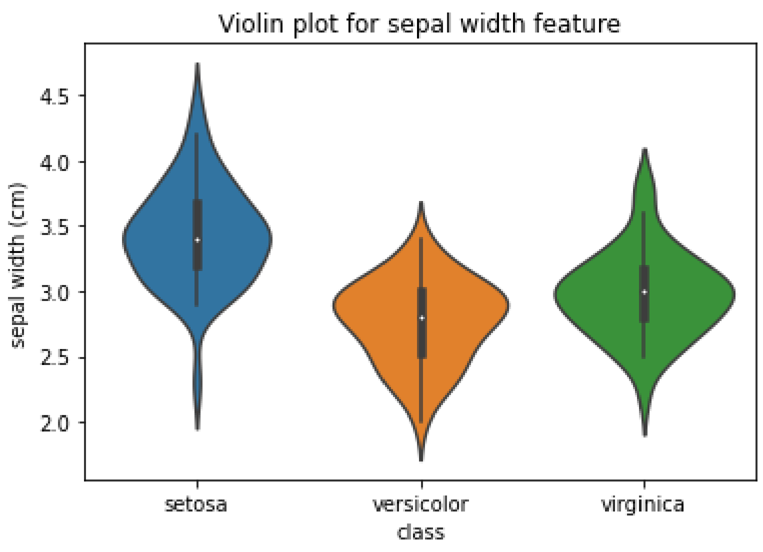
Interpretation: The following can be inferred from the density plots:

- The sepal length and sepal width feature follow normal distribution.
- Petal width and petal length feature have bimodal distribution.
- The majority of sepal lengths lie close to 5.8 cm. Similarly, 3 cm is the most common sepal width in the data. The plots also reveal that most of the petal lengths are 4.3 cm or 1.7 cm. The petal width density plot shows that majority of petal widths are 1.5 cm followed by around 0.3 cm.

4. Create a violin plot for the sepal width feature for each class. What can be seen from the plots?

In [14]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.violinplot( x = target, y = data[ 'sepal width (cm)'] )
plt.title( 'Violin plot for sepal width feature' )
plt.xlabel( 'class' )
plt.xticks( ticks = range(3), labels = iris_dataset.target_names )
plt.show()
```



Interpretation: Infer sepal width violin plot

The following was inferred from the violin plots of sepal width feature of each class:

- The median sepal width is highest for the class 'sentosa', followed by 'virginica' and 'versicolor'.
- The interquartile range is highest for 'versicolor' and least for 'virginica'.
- The 'versicolor' class has the lowest number of outliers for sepal width feature, while 'sentosa' has the highest outliers followed by 'virginica
- For 'senstosa' class, the violin plot is the widest around 3.3cm which indicates that there is a higher probability of sepal widths of this value in the class.
- For 'versicolor' class and 'virginica' class, the plots are widest around 3cm indicating this as the most probable sepal width for these two classes.
- It is also clear from the plots that sepal width feature has a unimodal distribution for all three classes since only a single peak is visible in each of the violin plots.

Ex2 - Data Pre-processing

1. Load the heart dataset from the given heart.csv file. How many rows and columns does the dataset contain?

In [15]:
```
df = pd.read_csv("heart.csv")
nr_rows, nr_cols = df.shape
print(f"The dataset contains {nr_rows} rows and {nr_cols} columns.")
```

```
The dataset contains 312 rows and 14 columns.
```

2. How many unique values does each column contain?

In [16]:
```
print( 'No. of unique values in each column')
print( df.nunique() )
```

```
No. of unique values in each column
age          41
sex           2
cp            4
trestbps     49
chol        152
fbs           2
restecg       3
thalach      91
```

```
exang         2
oldpeak      40
slope         3
ca            5
thal          4
target        2
dtype: int64
```

3. Count the number of duplicate rows in the dataset. How can you remove the duplicate rows?

In [17]:
```python
import numpy as np
### To get the number of duplicates, collect them via 'duplicated'
### And then count them, by summing every 'True' value
nr_duplicated_rows = np.sum(df.duplicated())
print(f"There are {nr_duplicated_rows} duplicated rows.")
# Result: 9
```

```
There are 9 duplicated rows.
```

In [18]:
```python
### We can get rid of the duplicates by calling the respective method.
### Note that the method RETURNS a duplicate free DataFrame and does not change the
df = df.drop_duplicates()
### And checking that zero duplicates are left:
if np.sum(df.duplicated()) == 0: print("No more duplicates present!")
nr_rows, nr_cols = df.shape
print(f"The dataset contains {nr_rows} rows and {nr_cols} columns.")
```

```
No more duplicates present!
The dataset contains 303 rows and 14 columns.
```

4. Count the number of missing values in the dataset.

In [19]:
```python
### We can use the function 'isna' to mark every NaN/missing value with True
### Call 'sum' two times to sum their number over both columns and rows
nr_missing_values = df.isna().sum().sum() # similar function 'isnull'
print(f"There are {nr_missing_values} values missing (or NaN) in all columns and row
# Result: 17
```

```
There are 17 values missing (or NaN) in all columns and rows.
```

In [20]:
```python
print( 'Column-wise missing values', '\n', df.isna().sum(), '\n' )
```

```
Column-wise missing values
 age        0
sex         3
cp          1
trestbps    1
chol        1
fbs         5
restecg     1
thalach     1
exang       0
oldpeak     1
slope       1
ca          1
thal        1
target      0
dtype: int64
```

5. How can you deal with missing values in your dataset? Implement one of the possible methods.

Results from our research:

The following mechanisms can be used to deal with missing values:-

- Deleting the row or column which contains missing values We can delete certain rows or columns in our dataset where the no. of missing values is greater than a threshold limit decided by us. This methos can however lead to loss of information or introduce bias in our dataset. Therefore it is only feasible to opt for this method when we have sufficient data.

- Replacing missing values with a statistic We can replace missing values with a statistic such as mean, median or mode. However this method is only possible for numeric values. This mechanism does not lead to loss of data but might introduce variance or bias in the dataset.

- For a categorical column's missing value, create a new category In a categorical feature that contains missing values, we can create a new category and assign it to all missing values. This method would not lead to loss of data and will introduce lower variance as compared to other techniques. However, algorithms might attempt to model the new category created for missing values which could result in poor performance.

- Predict missing values using machine learning This method predicts the missing values by applying maching learning techniques. The estimated missing values introduce less bias using this method.

- Apply althorithms that can handle missing values This method involves restricting to the use of algorithms that can handle missing values, eg. Random Forest, KNN algorithm.

In [21]:
```
### We can fill NaN/null/missing values in a datasets with the
### 'fillna', 'replace' and 'interpolate' functions.

### The 'interpolate' function is basically used to fill NaN values in the dataframe
### but it uses various interpolation technique to fill the missing values rather th

df = df.fillna(0) # missing -> 0.0
# df.fillna(method ='pad', inplace= True ) # missing -> previous value - will not fi
# df.fillna(method ='bfill', inplace= True ) # missing -> next value - will not fill
# df.interpolate(method ='linear', limit_direction ='forward') # missing -> linear i

### Again, let us check, if this has worked:
print( 'Column-wise missing values', '\n', df.isna().sum(), '\n' )
print(f"There are {df.isna().sum().sum()} values missing (or NaN).")
```

```
Column-wise missing values
 age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64

There are 0 values missing (or NaN).
```

Ex3 - Correlation

1. Load the dataset from the given dataset.tsv file.

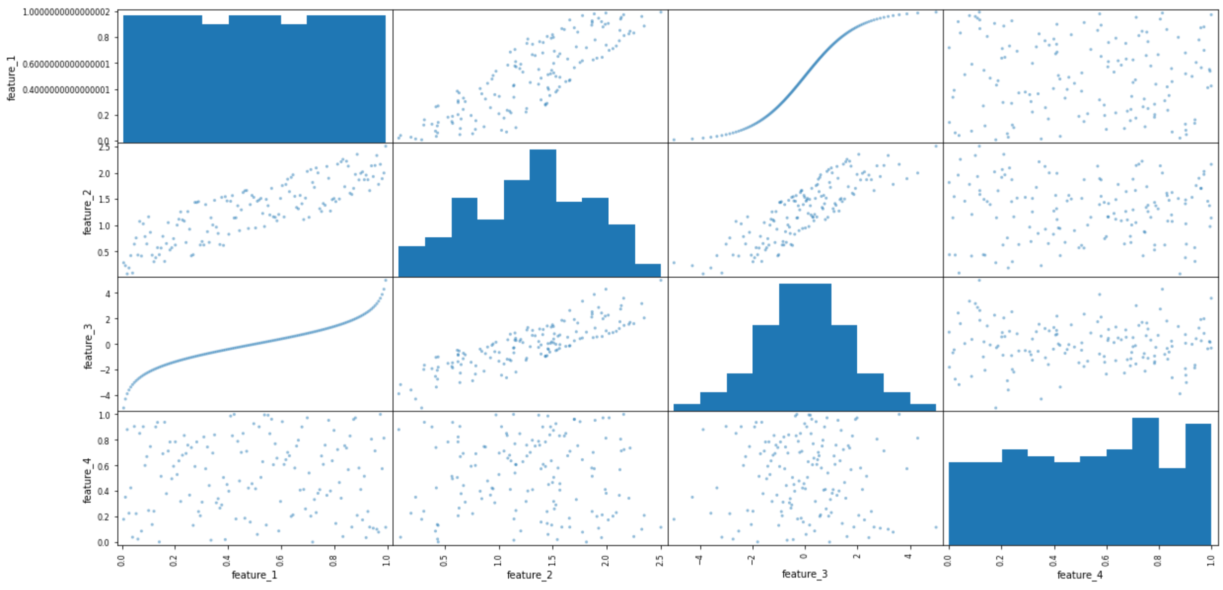
In [22]:
```
### We can use the 'read_csv' function, but since the data is not
### comma-separated but tabular-separated, we need to specify the separator accordin
### We treat the 'unnamed' column as the index column (by setting 'index_col=0'.)
### (Alternative method: df.drop(df.columns[0], axis=1, inplace=True))
df = pd.read_csv("dataset.tsv", sep='\t', index_col = 0)
df.head(4)
```

Out[22]:

	feature_1	feature_2	feature_3	feature_4
0	0.006711	0.286672	-4.997212	0.178739
1	0.013423	0.230586	-4.297285	0.351505
2	0.020134	0.074979	-3.884994	0.879812
3	0.026846	0.187541	-3.590439	0.226149

2. Plot the scatterplot matrix for the given dataset. What can be seen in the scatterplot matrix?

In [23]:
```
# pd.plotting.scatter_matrix(df, alpha=1)
pd.plotting.scatter_matrix(df, figsize=(20, 10 ))
plt.show()
```



Solution 2 - inference from scatterplot matrix

The following can be inferred from scatter plot:

- 'feature_1' and 'feature_2' show a linear relationship. The postive slope indicates that the association is postive. However, it is a weak relationship as the points are widely spread.
- 'feature_1' and 'feature_3' show a postive association and non-linear relationship. The relationship is also strong since all points are concentrated.
- 'feature_4' does not show any relationship with another feature as points are scattered in the plot.
- 'feature_2' and 'feature_3' show a linear relationship. The postive slope indicates that the relationship is postive. However, it is a weak relationship as the points are widely spread.

3. Which correlation would suit the comparison of feature_1 and feature_3? Calculate the relevant correlation coefficient for the 2 features.

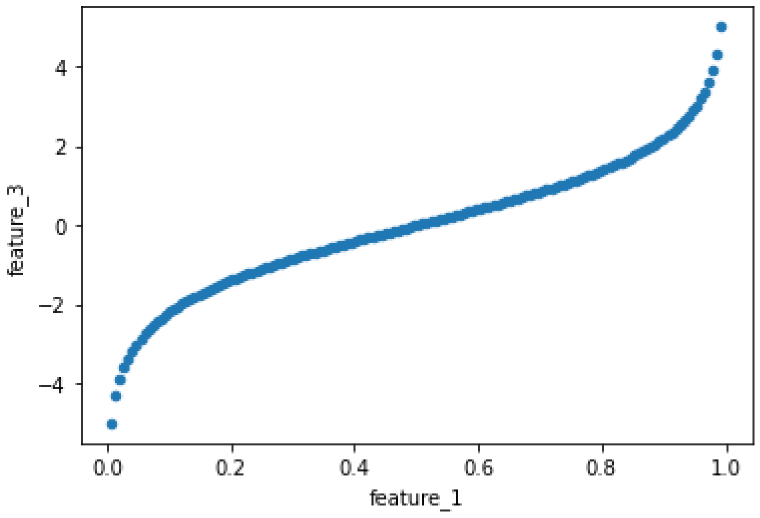
The correlation between the two features is positive and almost linear. The values of feature 1 increase iff the values of feature 2 increase. It seems to resemble a Spearman's rank correlation (non-linear dependency).

In [24]:
```
print( 'Pearson Correlation Coeffiecient:\t', df.corr(method ='pearson')[ 'feature_1
print( 'Spearman Correlation Coeffiecient:\t', df.corr(method ='spearman')[ 'feature
```

```
Pearson Correlation Coeffiecient:        0.9685071155522017
Spearman Correlation Coeffiecient:       1.0
```

In [25]:
```
df.plot.scatter(x='feature_1', y='feature_3')
```

Out[25]:
```
<AxesSubplot:xlabel='feature_1', ylabel='feature_3'>
```



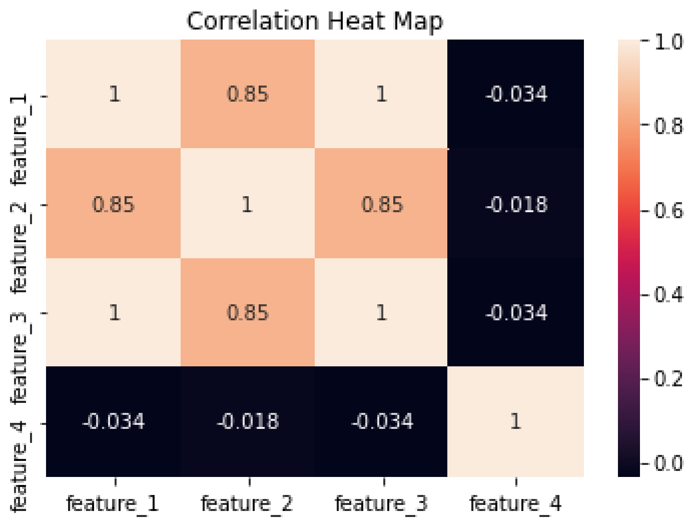
In [26]:
```
correlation = df.corr(method='spearman')
print('Spearman correlation between feature 1 and 3 is', correlation.iloc[0,2])
```

```
Spearman correlation between feature 1 and 3 is 1.0
```

4. Plot the correlation heatmap of the entire dataset.

In [27]:
```
import seaborn as sns

Var_Corr = df.corr(method ='spearman')
# plot the heatmap and annotation on it
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns, yticklabels=Var_Corr.columns, an
plt.title('Correlation Heat Map')
plt.show()
```



Correlation Heat Map
feature_1
feature_2
feature_3
feature_4
1
0.85
1
-0.034
0.85
1
0.85
-0.018
1
0.85
1
-0.034
-0.034
-0.018
-0.034
1
1.0
0.8
0.6
0.4
0.2
0.0