Unsupervised machine learning: clustering project by Sisay Menji

In [303]:
```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
%matplotlib inline
```

Data Description

In this project I will use the popular classification dataset, IRIS from UCI machine learnig repository (http://archive.ics.uci.edu/ml/datasets/Iris/). The dataset contains 3 classes of 50 instances each. Each class represents the type of iris plant. The data contains 5 variables and 150 observations. The variables are

1. sepal_length: sepal length in cm
2. sepal_width: sepal width in cm
3. petal_length: petal length in cm
4. petal_width: petal width in cm
5. class: which have 3 types Iris Setosa, Iris Versicolour and Iris Virginica

In [304]:
```
df = pd.read_csv("iris.data", names=['sepal_length', 'sepal_width','petal_length','petal_width','class'])
```

In [305]:
```
df.head()
```

Out[305]:

	sepal_length	sepal_width	petal_length	petal_width	class
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

Objectives of the analysis

In this project I will use different clustering alogrithms to predict the three different classes of the iris plant. The analysis plans to achieve the following:

- Use KMeans, Agglomerative and DB scal clustering algorithms to find the class labels based on the four numeric variables
- Select the best cluster class size using inertia for the KMeans model and use the size for the other models
- Compare which of the three clustering algorithsm fit the data well

In [306]:
```
cols = df.columns
df['class'].value_counts()
```

Out[306]:
```
Iris-virginica     50
Iris-versicolor    50
Iris-setosa        50
Name: class, dtype: int64
```

Descriptive statistics

In [307]:
```
feature_cols = [c for c in cols if c!='class']
feature_cols
```

Out[307]:
```
['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
```

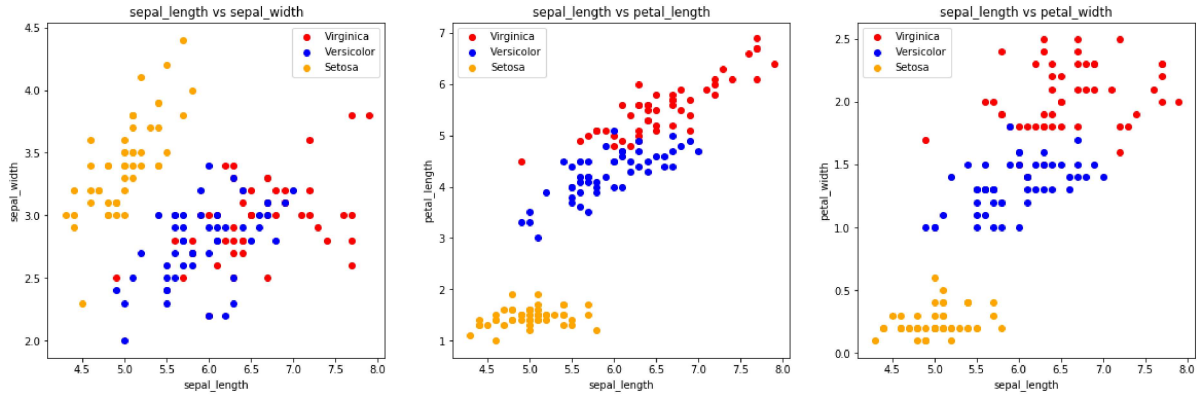
In [308]:
```
# separate dfs to make charing easier
c1, c2, c3 = "Iris-virginica", "Iris-versicolor","Iris-setosa"
df1 = df.loc[df['class']==c1,:]
df2 = df.loc[df['class']==c2,:]
df3 = df.loc[df['class']==c3,:]
df1.shape, df2.shape, df3.shape
```

Out[308]:
```
((50, 5), (50, 5), (50, 5))
```

In [310]:
```
# function creates scatter plots by one variable for all numeric feature
def plot_scatters(v1):
    fig, axs = plt.subplots(1,3, figsize=(20,6))
    fcols = [c for c in feature_cols if c !=v1]
    for i in range(len(fcols)):
        axs[i].scatter(df1.loc[:,v1],df1.loc[:,fcols[i]], c='red')
        axs[i].scatter(df2.loc[:,v1],df2.loc[:,fcols[i]], c='blue')
        axs[i].scatter(df3.loc[:,v1],df3.loc[:,fcols[i]], c='orange')
        axs[i].set_xlabel(v1)
        axs[i].set_ylabel(fcols[i])
        axs[i].set_title(v1 + " vs " + fcols[i])
        axs[i].legend(["Virginica", "Versicolor", "Setosa"]);
```

Sepal length and other feature columns in identifying the clusters

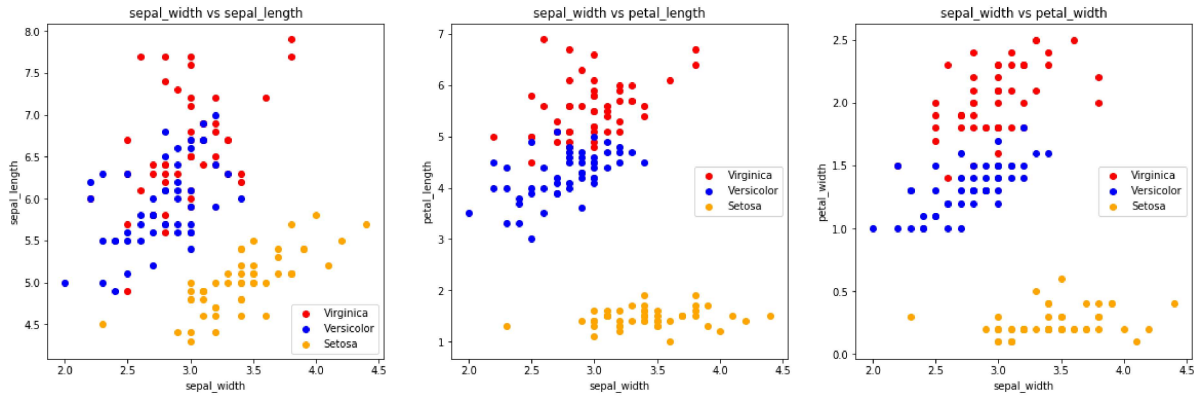
In [311]:
```
plot_scatters("sepal_length")
```



As the above charts show sepal length and sepal width are good in separating setosa from the other iris types but they are not sufficient to clearly separate virginica and versicolor. Combining sepal length with petal length and petal width gives better results in separating virginica and versicolor but the cluster is not linearly separable as can be seen from the plot above.

Sepal width and other feature columns in identifying the clusters

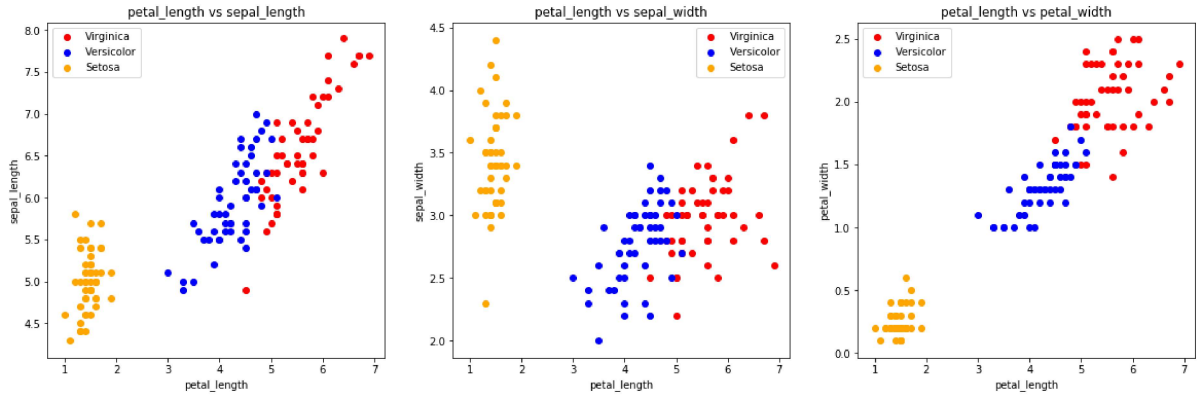
In [312]:
```
plot_scatters("sepal_width")
```



As the above charts shows that similar to the scatters for sepal length, it is easier to separate setosa while the two classes (virginica and versicolor) are different to separate.

Petal length, petal width in identifying the clusters

In [313]:
```
plot_scatters("petal_length")
```



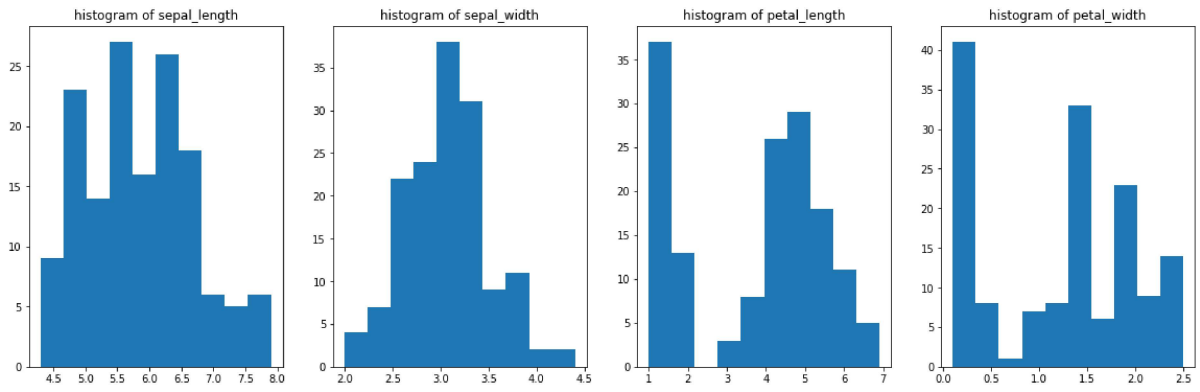
The scatter plots above clearly show that **it is easier to separate setosa linearly while versicolor and virginica are not linearly separable**. This implies that linear clustering algorithms will perform poorly in getting the 3 clusters.

In [314]:
```
## summary statistics for the variables
df.describe()
```

Out[314]:

	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

In [315]:
```
fig, axs = plt.subplots(1,4, figsize=(20,6))
for i in range(len(feature_cols)):
    axs[i].hist(df[feature_cols[i]])
    axs[i].set_title("histogram of " + feature_cols[i])
```



As the aboe figures show all the variables do not look normally distributed and they are skewed which can be checked with the skewness values. The values below show that the skewness are not further from zero and hence no scaling to change the data to normal is not done

In [316]:
```
df.skew()
```

Out[316]:
```
sepal_length    0.314911
sepal_width     0.334053
petal_length   -0.274464
petal_width    -0.104997
dtype: float64
```

Data preparation

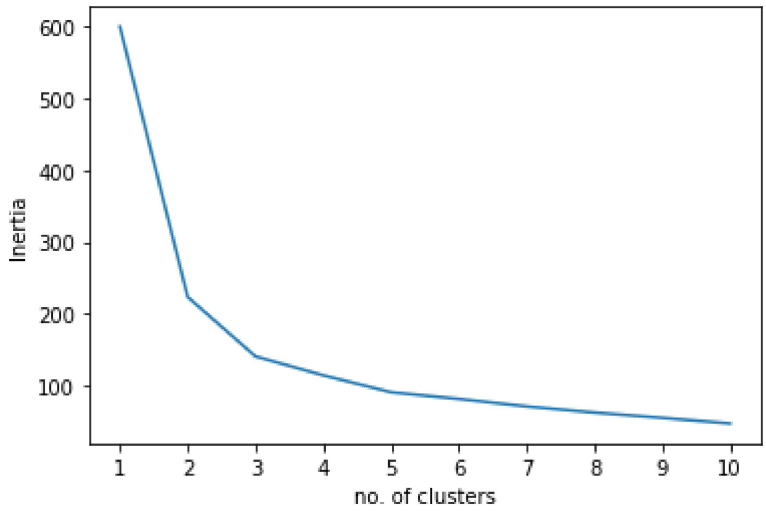
In this section, I will use standard scaler to transform the feature columns into same scale as it is important for the clustering algorithms if the data have similar range

In [317]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X, y = df[feature_cols], df['class']
x = scaler.fit_transform(X)
```

Clustering algorithms - KMeans

In this section I will apply KMeans algorithm with clusters of 1-10 and chose the best KMeans model based on inertia

In [318]:
```
from sklearn.cluster import KMeans
inertia = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(x)
    inertia.append(kmeans.inertia_)
plt.plot(range(1,11), inertia);
plt.xlabel("no. of clusters")
plt.ylabel("Inertia")
plt.xticks(range(1,11));
```



The elbow model above shows that kmeans of cluster 3 provides a good solution. The best kmeans solution will be saved and later used to compare the different clustering algoritms

In [327]:
```
kmeans = KMeans(n_clusters=3, random_state=100)
```

Clustering algorithms - Agglomerative clustering

In this section I will apply Agglomerative clustering algorithm with clusters of size 3 using the best size based on KMEANS.

In [326]:
```
from sklearn.cluster import AgglomerativeClustering
agg = AgglomerativeClustering(n_clusters=3)
```

Clustering algorithms - DBSCAN

In this section I will apply DBSCAN clustering algorithm. The best cluster size will be chosen by usng different combinations of eps and min_samples and chosing the one where there are no noise (no -1 label classification) and the cluster size is 3

In [321]:
```
from sklearn.cluster import DBSCAN
eps = set(np.linspace(0.5,5,30))
nsamp = set(np.linspace(1,20,20))
alg = set(["ball_tree", "kd_tree", "brute"])
from itertools import product
comb = list(product(eps, nsamp, alg));
dbms = []
for e, n, a in comb:
    db = DBSCAN(eps=e, min_samples=n,algorithm=a, n_jobs=-1)
    db.fit(x)
    uc = pd.DataFrame(db.labels_).nunique()[0]
    if (-1 not in db.labels_) and (uc == 3):
        dbms.append(db)
        print("converged solution: {},{},{}".format(e,n,a, uc))
```

```
converged solution: 0.9655172413793104,1.0,ball_tree
converged solution: 0.9655172413793104,1.0,kd_tree
converged solution: 0.9655172413793104,1.0,brute
converged solution: 1.1206896551724137,1.0,ball_tree
converged solution: 1.1206896551724137,1.0,kd_tree
converged solution: 1.1206896551724137,1.0,brute
converged solution: 1.2758620689655173,1.0,ball_tree
converged solution: 1.2758620689655173,1.0,kd_tree
converged solution: 1.2758620689655173,1.0,brute
```

In [322]:
```
dbscan = dbms[0]
```

Comparing clustering algoritms

In this section I will compare the clustering algorithms using adjuted rand score of the selected models

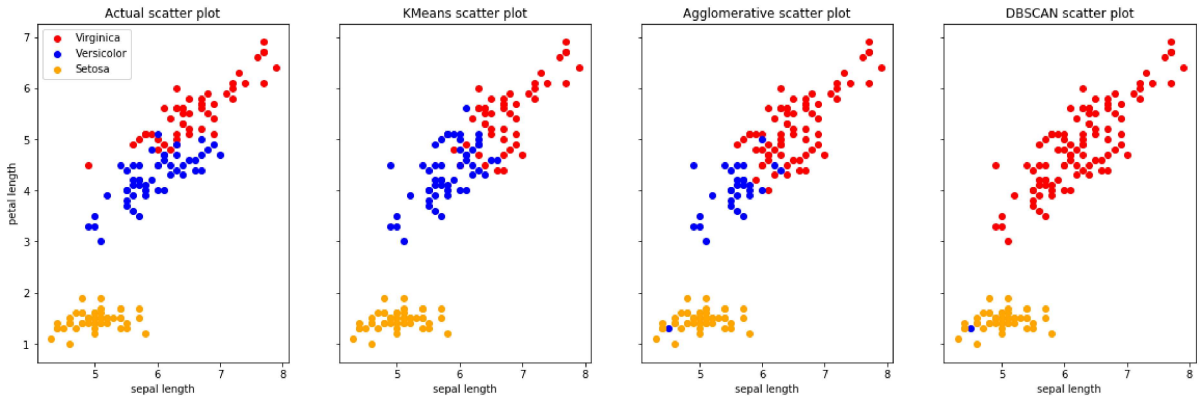
In [328]:
```
from sklearn.metrics.cluster import adjusted_rand_score, normalized_mutual_inf
o_score
for model, name in [(kmeans, 'KMeans'), (agg, 'Agglomerative'), (dbscan, 'DBSC
AN')]:
    y_pred = model.fit_predict(x)
    print(name, "Adjusted rand score: ", "{:.2f}".format(adjusted_rand_score(y
,y_pred)),
          "Normalized score: ", "{:.2f}".format(normalized_mutual_info_score(y
,y_pred)))
```

```
KMeans Adjusted rand score:  0.62 Normalized score:  0.66
Agglomerative Adjusted rand score:  0.62 Normalized score:  0.68
DBSCAN Adjusted rand score:  0.56 Normalized score:  0.72
```

The results above shows that KMeans and Agglomerative clustering model provides better solution compared to the other two models. Next I will plot the scatter plot of the results of the algorithms against the predicted labels.I will use scatter plot of sepal length and petal length to show the clusters and compare them with the predicted clusters.

In [329]:
```
kmeans_ = kmeans.predict(x)
agg_ = agg.fit_predict(x)
db_ = dbscan.fit_predict(x)
```

In [333]:
```
fig, axs = plt.subplots(1,4, figsize=(20,6),sharey=True)
# first plot normal
c1, c2, c3 = "Iris-virginica", "Iris-versicolor","Iris-setosa"
df1 = df.loc[df['class']==c1,:]
df2 = df.loc[df['class']==c2,:]
df3 = df.loc[df['class']==c3,:]
axs[0].scatter(df1.loc[:,'sepal_length'],df1.loc[:,'petal_length'], c='red')
axs[0].scatter(df2.loc[:,'sepal_length'],df2.loc[:,'petal_length'], c='blue')
axs[0].scatter(df3.loc[:,'sepal_length'],df3.loc[:,'petal_length'], c='orange'
)
axs[0].set_xlabel('sepal length')
axs[0].set_ylabel('petal length')
axs[0].set_title("Actual scatter plot")
axs[0].legend(["Virginica", "Versicolor", "Setosa"]);
# second plot: KMeans
df1 = df.iloc[kmeans_==0,:]
df2 = df.iloc[kmeans_==2,:]
df3 = df.iloc[kmeans_==1,:]
axs[1].scatter(df1.loc[:,'sepal_length'],df1.loc[:,'petal_length'], c='red')
axs[1].scatter(df2.loc[:,'sepal_length'],df2.loc[:,'petal_length'], c='blue')
axs[1].scatter(df3.loc[:,'sepal_length'],df3.loc[:,'petal_length'], c='orange'
)
axs[1].set_xlabel('sepal length')
axs[1].set_title("KMeans scatter plot")
# third plot: Agglomerative clustering
df1 = df.iloc[agg_==0,:]
df2 = df.iloc[agg_==2,:]
df3 = df.iloc[agg_==1,:]
axs[2].scatter(df1.loc[:,'sepal_length'],df1.loc[:,'petal_length'], c='red')
axs[2].scatter(df2.loc[:,'sepal_length'],df2.loc[:,'petal_length'], c='blue')
axs[2].scatter(df3.loc[:,'sepal_length'],df3.loc[:,'petal_length'], c='orange'
)
axs[2].set_xlabel('sepal length')
axs[2].set_title("Agglomerative scatter plot")
# fourth plot: DBSCAN clustering
df1 = df.iloc[db_==2,:]
df2 = df.iloc[db_==1,:]
df3 = df.iloc[db_==0,:]
axs[3].scatter(df1.loc[:,'sepal_length'],df1.loc[:,'petal_length'], c='red')
axs[3].scatter(df2.loc[:,'sepal_length'],df2.loc[:,'petal_length'], c='blue')
axs[3].scatter(df3.loc[:,'sepal_length'],df3.loc[:,'petal_length'], c='orange'
)
axs[3].set_xlabel('sepal length')
axs[3].set_title("DBSCAN scatter plot");
```



Summary

As the above scatter plots show KMeans and Agglomerative clustering perform much better than DBSCAN which identified only two clusters clearly and couldn't separate the virginica and versicolor types. The inability of DBSCAN in separating virginica and versicolor comes from the model limitation that it doesn't adapt to varying density of the clusters.

Next stpes

In this project three clustering algorithms were compared to predict the labels of the Iris dataset. The results show that KMeans and Agglomerative clustering performed better. Though not considered in this project, the following activities might affect the results and can be areas for future study:

- considering different scaling algorithms. I used StandardScaler but it is good to see if results can be affected when MinMax scaler is used
- Checking if dimensionality reduction can help DBSCAN perform better. Reducing the number of features from 4 to say 2 might improve DBSCAN performance to separate the two classes

---------- Thank you!----------