K_Means Clustering

Import Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
plt.rcParams["figure.figsize"] = (10,6)
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df = pd.read_csv("iris.csv")
df.head()
```

Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [3]:

```
df.species.value_counts()
```

Out[3]:

```
setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

null value yok

In [5]:

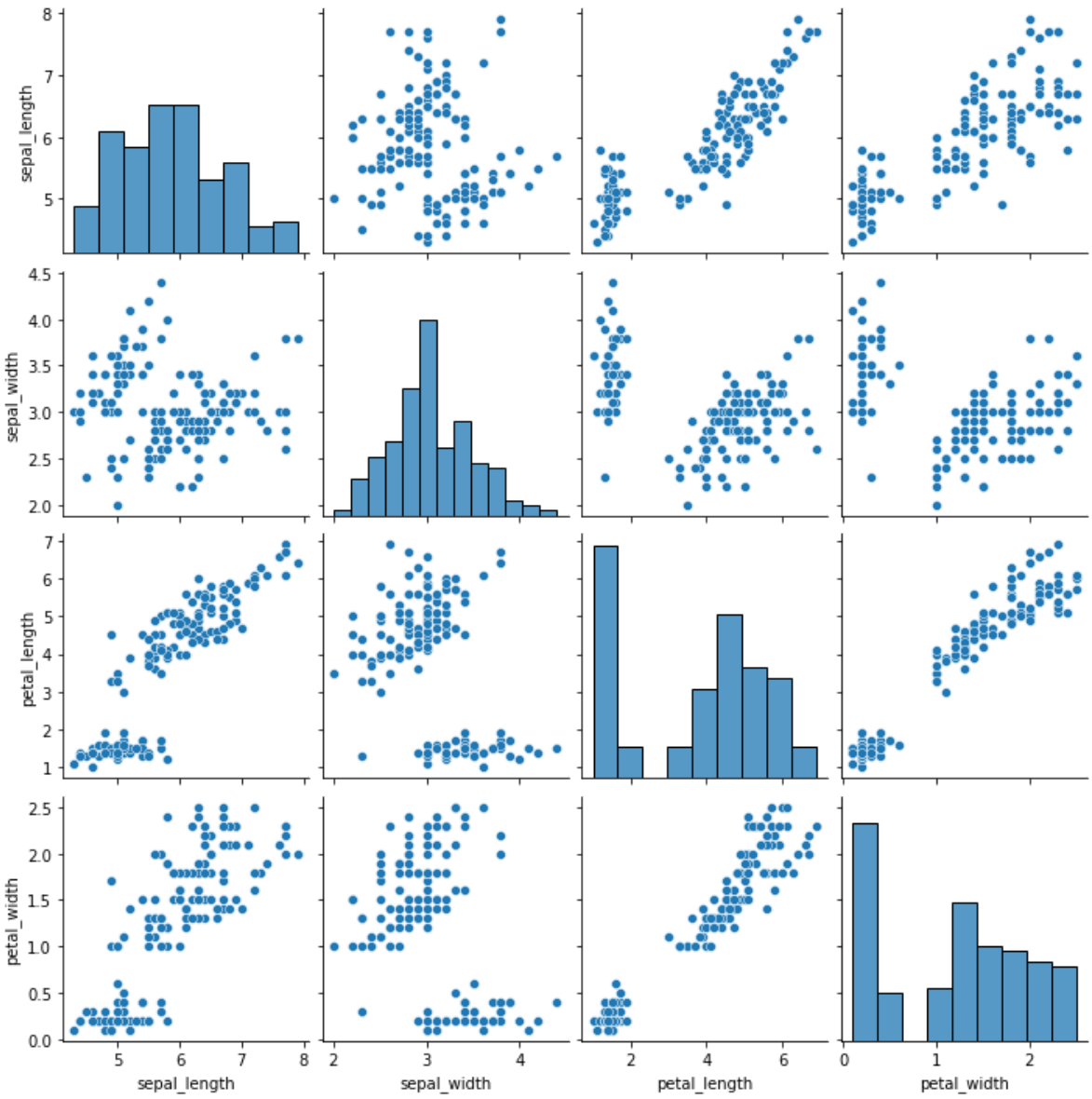
```
df.describe()
```

Out[5]:

	sepal_length	sepal_width	petal_length	petal_width
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
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

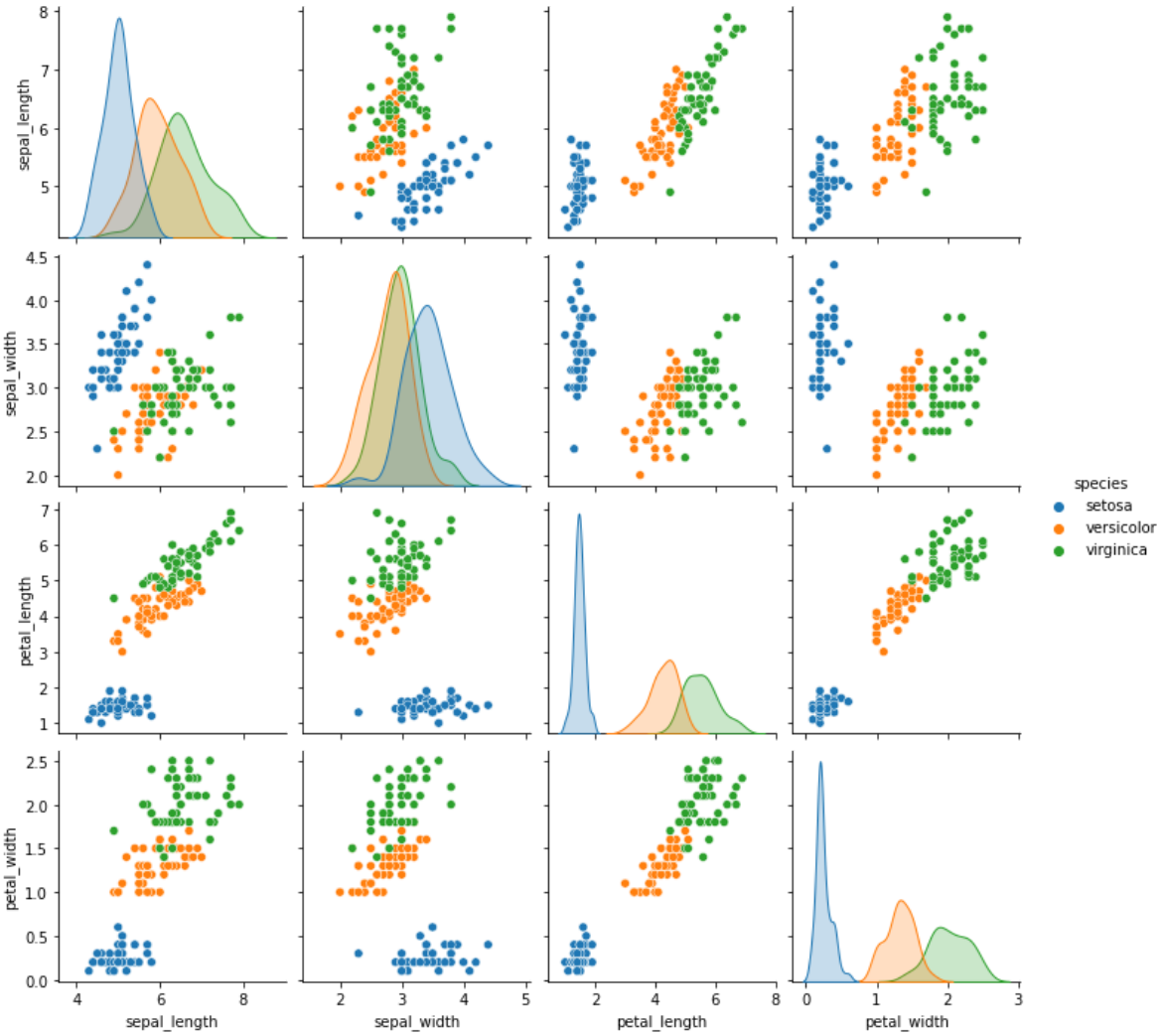
In [6]:

```
sns.pairplot(df)
plt.show()
```



In [7]:

```
sns.pairplot(data=df, hue="species")
plt.show()
```



K_Means Clustering

In [8]:

```
Z = df.copy()
```

In [9]:

```
X = Z.drop("species", axis = 1)
```

In [10]:

```
X.head()
```

Out[10]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [11]:

```
X.shape
```

Out[11]:

```
(150, 4)
```

In [12]:

```
?KMeans
```

In [13]:

```
from sklearn.cluster import KMeans

K_means_model = KMeans(n_clusters=5, random_state=42)
```

In [14]:

```
K_means_model.fit(X)
```

Out[14]:

```
KMeans(n_clusters=5, random_state=42)
```

In [15]:

```
K_means_model.labels_
```

Out[15]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 3, 3, 3, 2, 3, 3, 3, 2, 3, 2, 2, 3, 2, 3, 2, 3,
       3, 2, 3, 2, 3, 2, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 3, 2, 3, 3, 3,
       2, 2, 2, 3, 2, 2, 2, 2, 2, 3, 2, 2, 4, 3, 0, 4, 4, 0, 2, 0, 4, 0,
       4, 4, 4, 3, 4, 4, 4, 0, 0, 3, 4, 3, 0, 3, 4, 0, 3, 3, 4, 0, 0, 0,
       4, 3, 3, 0, 4, 4, 3, 4, 4, 4, 3, 4, 4, 4, 3, 4, 4, 3])
```

In [16]:

```
X["Classes"] = K_means_model.labels_
```

In [17]:

```
X
```

Out[17]:

	sepal_length	sepal_width	petal_length	petal_width	Classes
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	4
146	6.3	2.5	5.0	1.9	3
147	6.5	3.0	5.2	2.0	4
148	6.2	3.4	5.4	2.3	4

Choosing The Optimal Number of Clusters

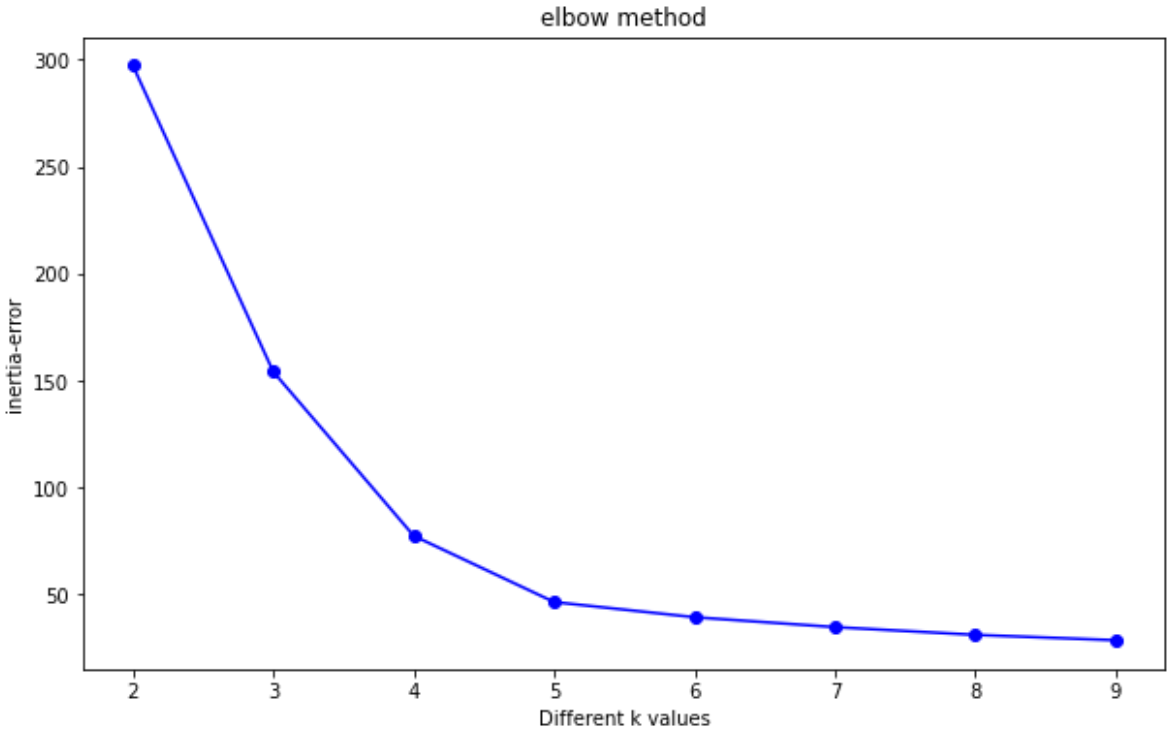
Elbow metod

In [18]:

```
ssd = []                        # sum of squared distance

K = range(2,10)

for k in K:
    model = KMeans(n_clusters =k, random_state=42)
    model.fit(X)
    ssd.append(model.inertia_)
```

In [19]:

```
plt.plot(K, ssd, "bo-")
plt.xlabel("Different k values")
plt.ylabel("inertia-error")
plt.title("elbow method")
plt.show()
```



In [20]:

```
ssd
```

Out[20]:

```
[297.21519904931665,
 154.0945807978364,
 77.4108126984127,
 46.44618205128205,
 39.35425513506102,
 34.735520984278885,
 31.117176989676995,
 28.60224837662338]
```

In [21]:

```
pd.Series(ssd).diff()
```

Out[21]:

```
0           NaN
1   -143.120618
2    -76.683768
3    -30.964631
4     -7.091927
5     -4.618734
6     -3.618344
7     -2.514929
dtype: float64
```

In [22]:

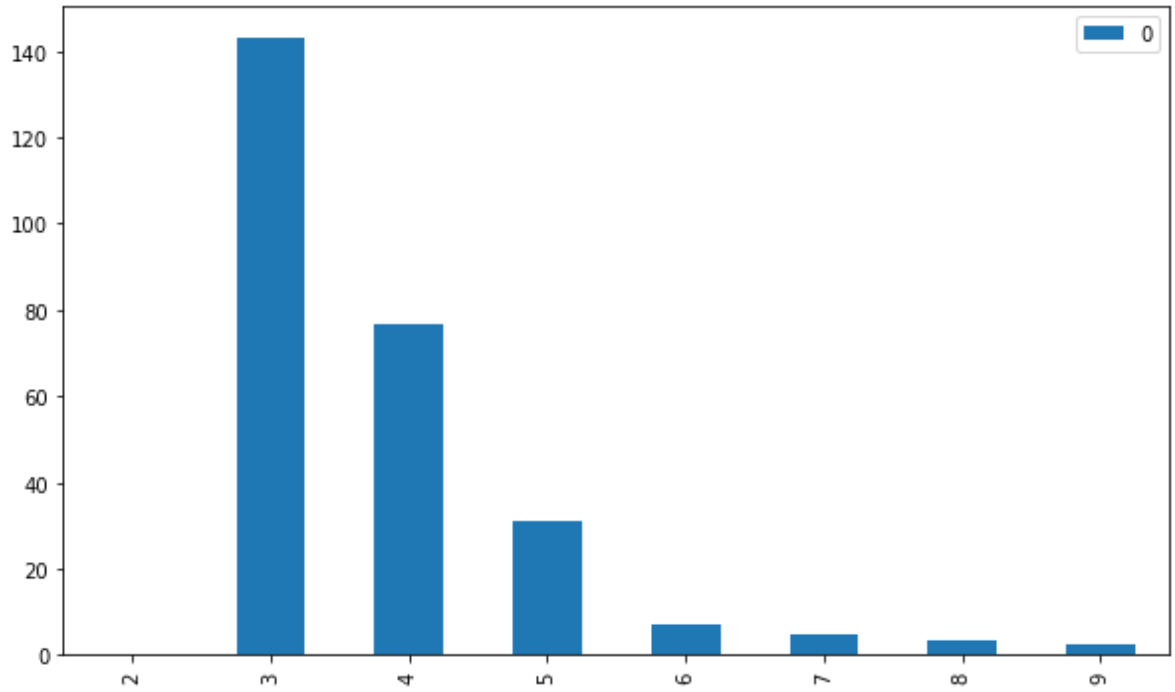
```
df_diff =pd.DataFrame(-pd.Series(ssd).diff()).rename(index = lambda x : x+2)
df_diff
```

Out[22]:

	0
2	NaN
3	143.120618
4	76.683768
5	30.964631
6	7.091927
7	4.618734
8	3.618344
9	2.514929

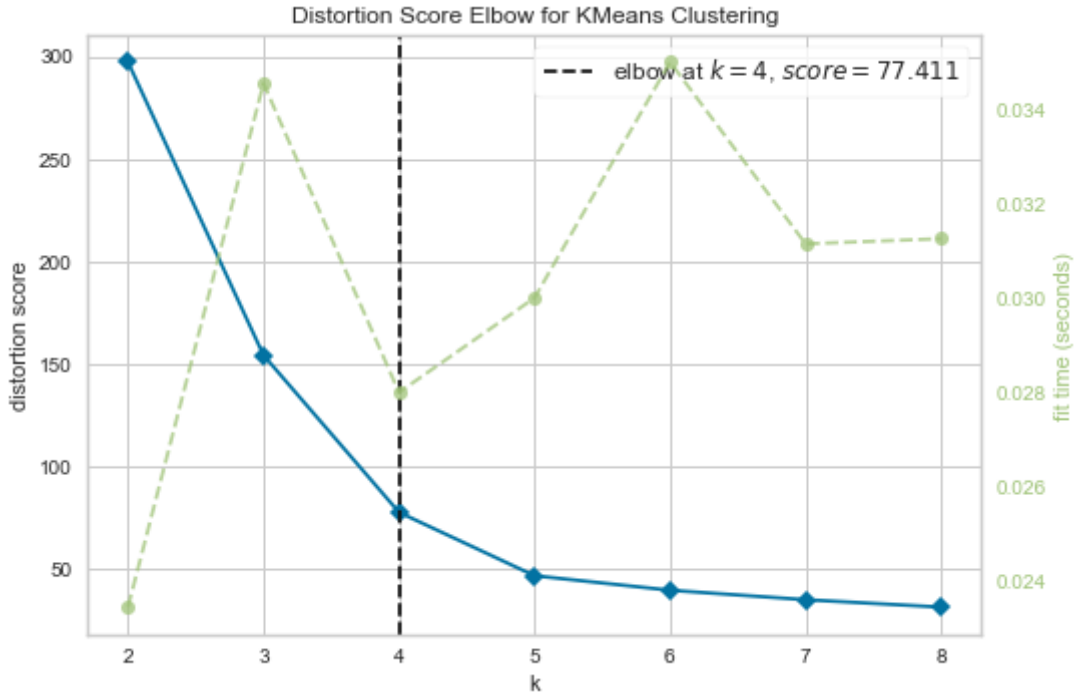
In [23]:

```
df_diff.plot(kind='bar');
```



In [24]:

```
from yellowbrick.cluster import KElbowVisualizer

model_ = KMeans(random_state=42)
visualizer = KElbowVisualizer(model_, k=(2,9))

visualizer.fit(X)
visualizer.show();
```



Silhouette analysis

In [25]:

```
from sklearn.metrics import silhouette_score
```

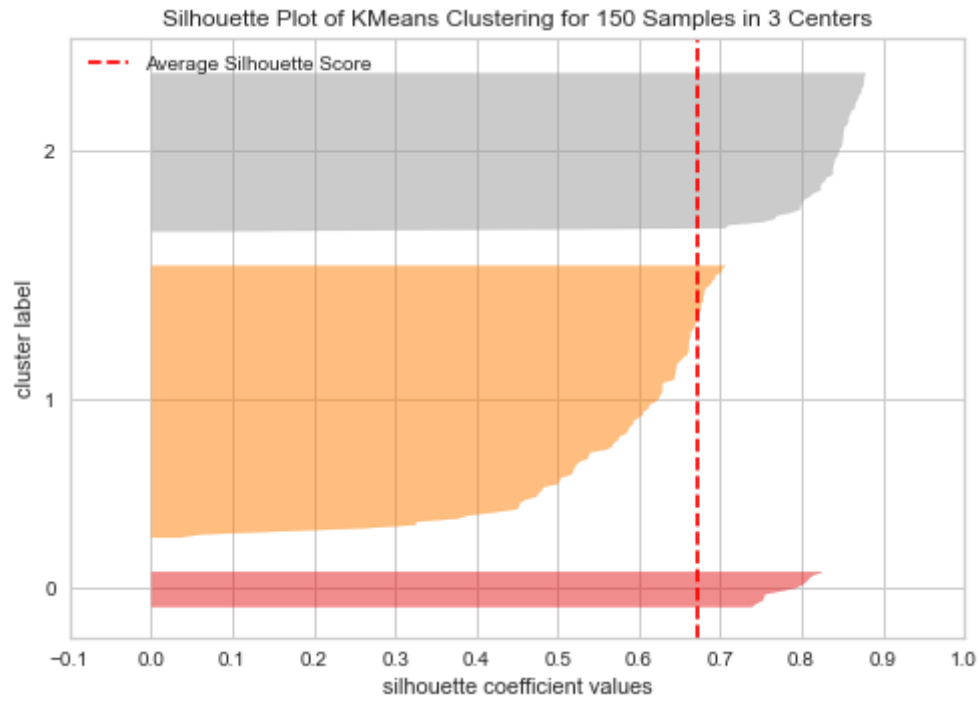
In [26]:

```
range_n_clusters = range(2,9)
for num_clusters in range_n_clusters:
    # intialise kmeans
    kmeans = KMeans(n_clusters=num_clusters, random_state=42)
    kmeans.fit(X)
    cluster_labels = kmeans.labels_
    # silhouette score
    silhouette_avg = silhouette_score(X, cluster_labels)
    print(f"For n_clusters={num_clusters}, the silhouette score is {silhouette_avg}")
```

```
For n_clusters=2, the silhouette score is 0.6302043128172005
For n_clusters=3, the silhouette score is 0.6715780817387732
For n_clusters=4, the silhouette score is 0.6117353392436371
For n_clusters=5, the silhouette score is 0.6314166907607778
For n_clusters=6, the silhouette score is 0.4981312302377309
For n_clusters=7, the silhouette score is 0.445058141055753
For n_clusters=8, the silhouette score is 0.4357498465448826
```

In [27]:

```
from sklearn.cluster import KMeans

from yellowbrick.cluster import SilhouetteVisualizer

model3 = KMeans(n_clusters=3, random_state=42)
visualizer = SilhouetteVisualizer(model3)

visualizer.fit(X)    # Fit the data to the visualizer
visualizer.poof();
```



Building the model based on the optimal number of clusters

In [28]:

```
model = KMeans(n_clusters =3, random_state=42)
model.fit_predict(X)
```

Out[28]:

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [29]:

```
model.labels_
```

Out[29]:

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [30]:

```
clusters = model.labels_
```

In [31]:

```
X.head()
```

Out[31]:

	sepal_length	sepal_width	petal_length	petal_width	Classes
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1

In [32]:

```
X["predicted_clusters"] = clusters
```

In [33]:

```
X
```

Out[33]:

	sepal_length	sepal_width	petal_length	petal_width	Classes	predicted_clusters
0	5.1	3.5	1.4	0.2	1	2
1	4.9	3.0	1.4	0.2	1	2
2	4.7	3.2	1.3	0.2	1	2
3	4.6	3.1	1.5	0.2	1	2
4	5.0	3.6	1.4	0.2	1	2
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	4	1
146	6.3	2.5	5.0	1.9	3	1
147	6.5	3.0	5.2	2.0	4	1
148	6.2	3.4	5.4	2.3	4	1
149	5.9	3.0	5.1	1.8	3	1

150 rows × 6 columns

In [34]:

```
df.head()
```

Out[34]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [35]:

```
df["predicted_clusters"] = clusters
```

In [36]:

```
df.head()
```

Out[36]:

	sepal_length	sepal_width	petal_length	petal_width	species	predicted_clusters
0	5.1	3.5	1.4	0.2	setosa	2
1	4.9	3.0	1.4	0.2	setosa	2
2	4.7	3.2	1.3	0.2	setosa	2
3	4.6	3.1	1.5	0.2	setosa	2
4	5.0	3.6	1.4	0.2	setosa	2

In [37]:

```
ct = pd.crosstab(df.predicted_clusters, df.species)
ct
```

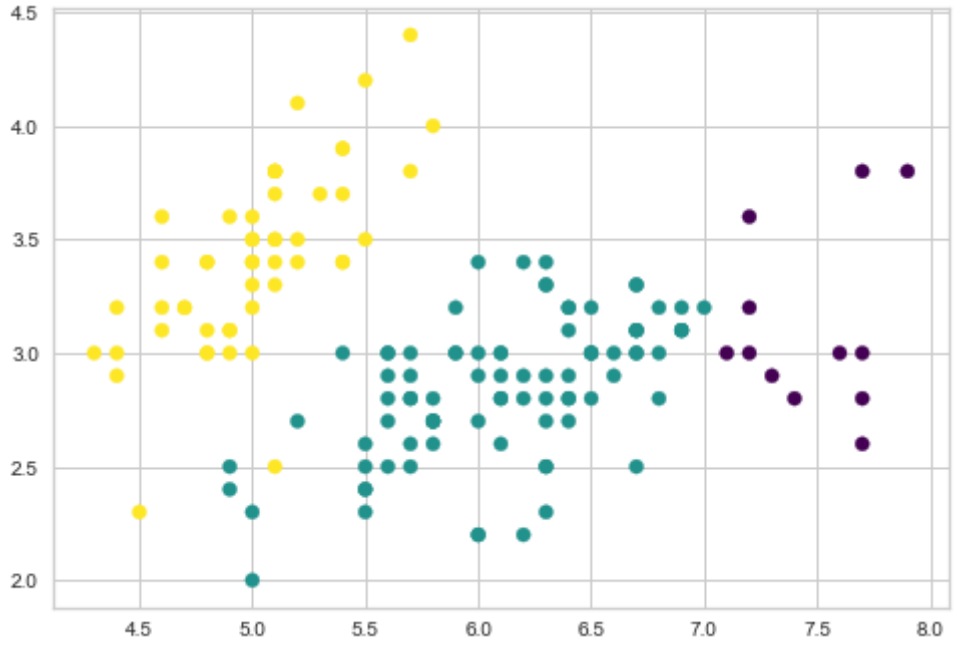
Out[37]:

species	setosa	versicolor	virginica
predicted_clusters			
0	0	0	12
1	0	49	38
2	50	1	0

Visualization Clusters

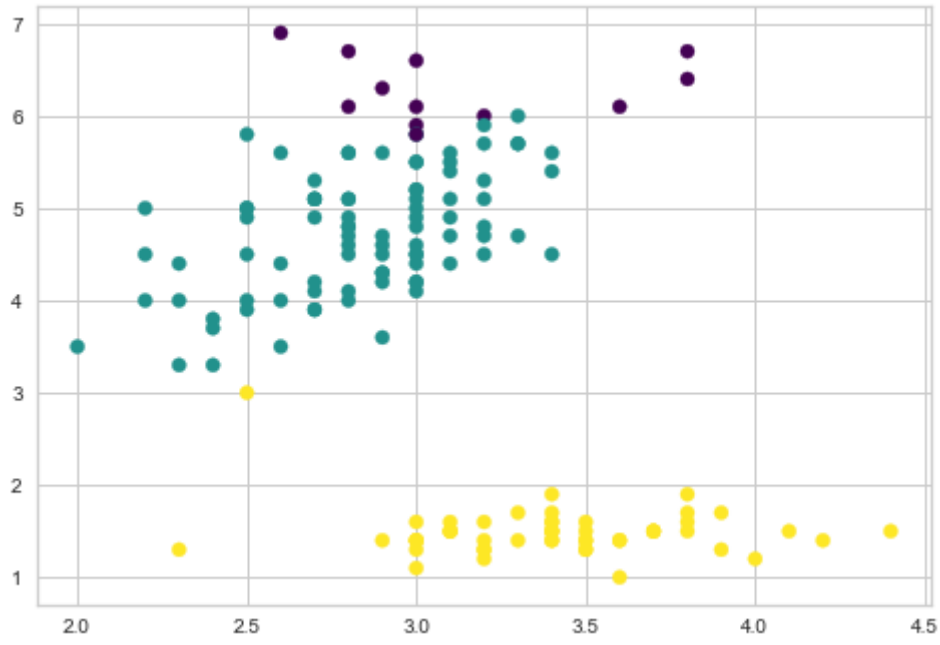
In [38]:

```
plt.scatter(df["sepal_length"], df["sepal_width"], c = df.predicted_clusters, cmap = "virid
```



In [39]:

```
plt.scatter(df["sepal_width"], df["petal_length"], c = df.predicted_clusters, cmap = "virid
```



In [40]:

```
centers = model.cluster_centers_
centers
```

Out[40]:

```
array([[7.47500000e+00, 3.12500000e+00, 6.30000000e+00, 2.05000000e+00,
        4.44089210e-16],
       [6.10804598e+00, 2.84137931e+00, 4.73563218e+00, 1.63103448e+00,
        3.00000000e+00],
       [5.00784314e+00, 3.40980392e+00, 1.49215686e+00, 2.62745098e-01,
        1.01960784e+00]])
```

In [41]:

```
centers[:,0] # centers of sepal_length feature
```

Out[41]:

```
array([7.475     , 6.10804598, 5.00784314])
```

In [42]:

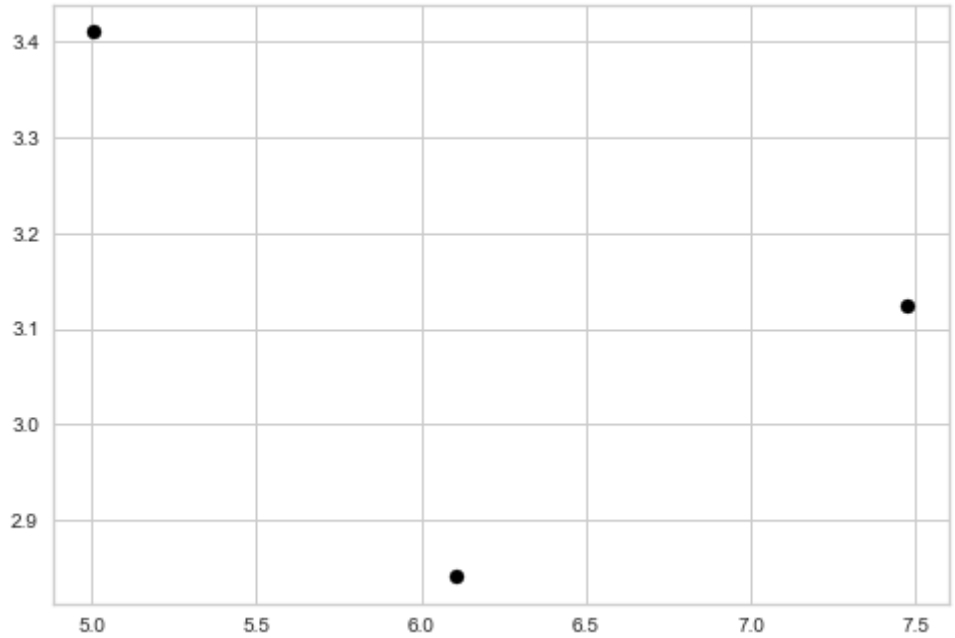
```
centers[:,1] # centers of sepal_width feature
```

Out[42]:

```
array([3.125     , 2.84137931, 3.40980392])
```

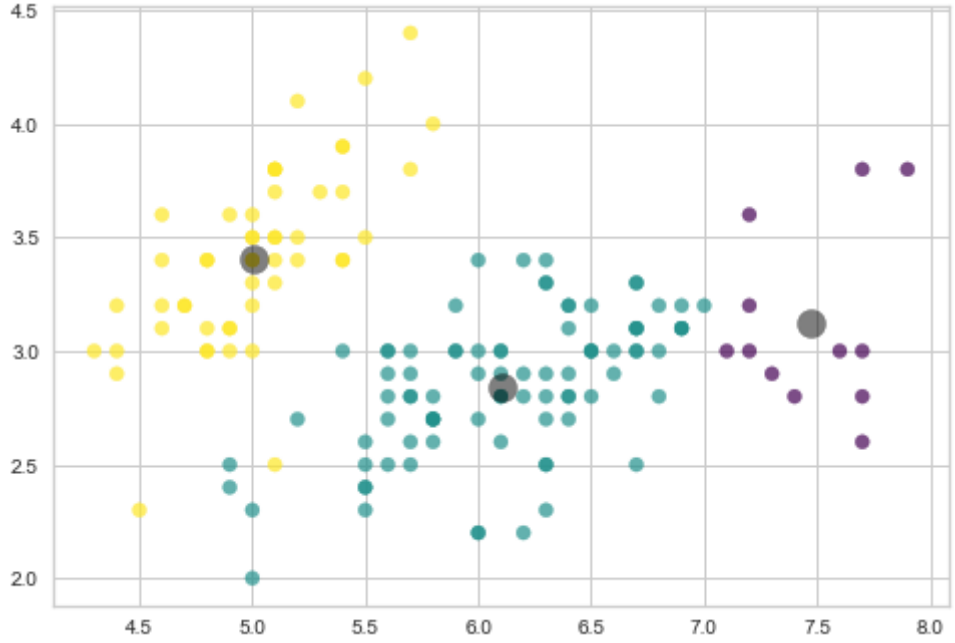
In [43]:

```
plt.scatter(centers[:,0], centers[:,1], c = "black");
```



In [44]:

```
plt.scatter(X["sepal_length"], X["sepal_width"], c = X.predicted_clusters, cmap = "viridis"
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```



Remodeling according to discriminating features

In [45]:

```
df.head()
```

Out[45]:

	sepal_length	sepal_width	petal_length	petal_width	species	predicted_clusters
0	5.1	3.5	1.4	0.2	setosa	2
1	4.9	3.0	1.4	0.2	setosa	2
2	4.7	3.2	1.3	0.2	setosa	2
3	4.6	3.1	1.5	0.2	setosa	2
4	5.0	3.6	1.4	0.2	setosa	2

In [46]:

```
df.iloc[:, [0,1,2,3,5]].head()
```

Out[46]:

	sepal_length	sepal_width	petal_length	petal_width	predicted_clusters
0	5.1	3.5	1.4	0.2	2
1	4.9	3.0	1.4	0.2	2
2	4.7	3.2	1.3	0.2	2
3	4.6	3.1	1.5	0.2	2
4	5.0	3.6	1.4	0.2	2

In [47]:

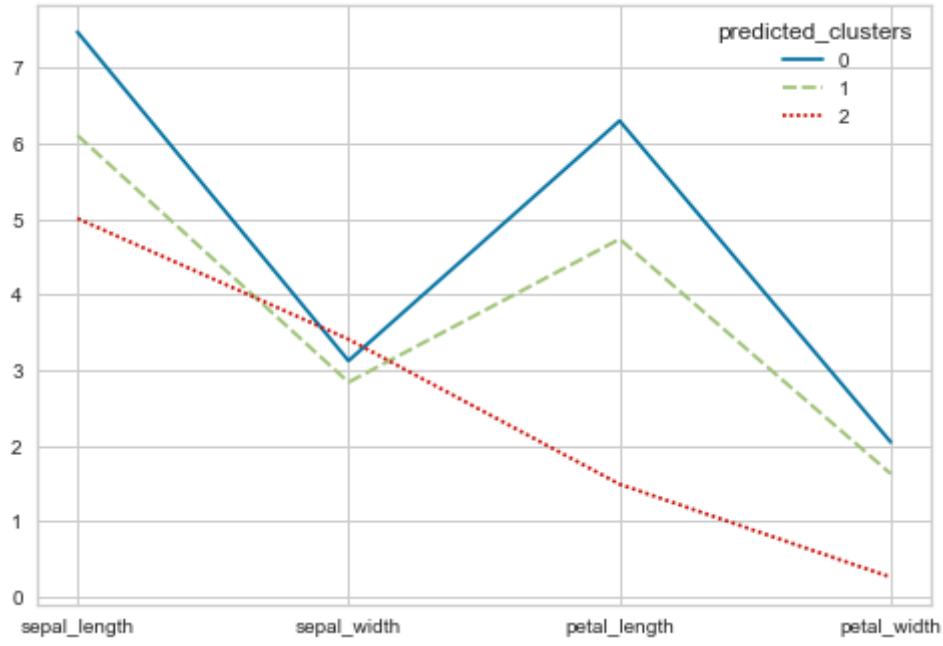
```
clus_pred = df.iloc[:, [0,1,2,3,5]].groupby("predicted_clusters").mean().T
clus_pred
```

Out[47]:

predicted_clusters	0	1	2
sepal_length	7.475	6.108046	5.007843
sepal_width	3.125	2.841379	3.409804
petal_length	6.300	4.735632	1.492157
petal_width	2.050	1.631034	0.262745

In [48]:

```
sns.lineplot(data = clus_pred);
```

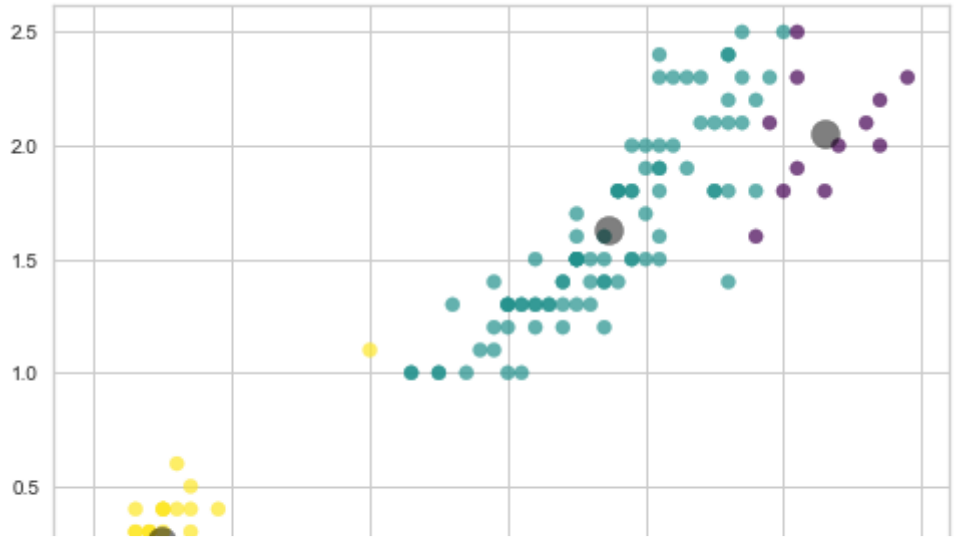


In [49]:

```
plt.scatter(X["petal_length"], X["petal_width"], c = X.predicted_clusters, cmap = "viridis"
plt.scatter(centers[:, 2], centers[:, 3], c='black', s=200, alpha=0.5)
```

Out[49]:

```
<matplotlib.collections.PathCollection at 0x1cce80a31f0>
```



In [50]:

```
X2 = X.iloc[:, [2,3]]
X2
```

Out[50]:

	petal_length	petal_width
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2
4	1.4	0.2
...	...	...
145	5.2	2.3
146	5.0	1.9
147	5.2	2.0
148	5.4	2.3
149	5.1	1.8

150 rows × 2 columns

Elbow metod

In [51]:

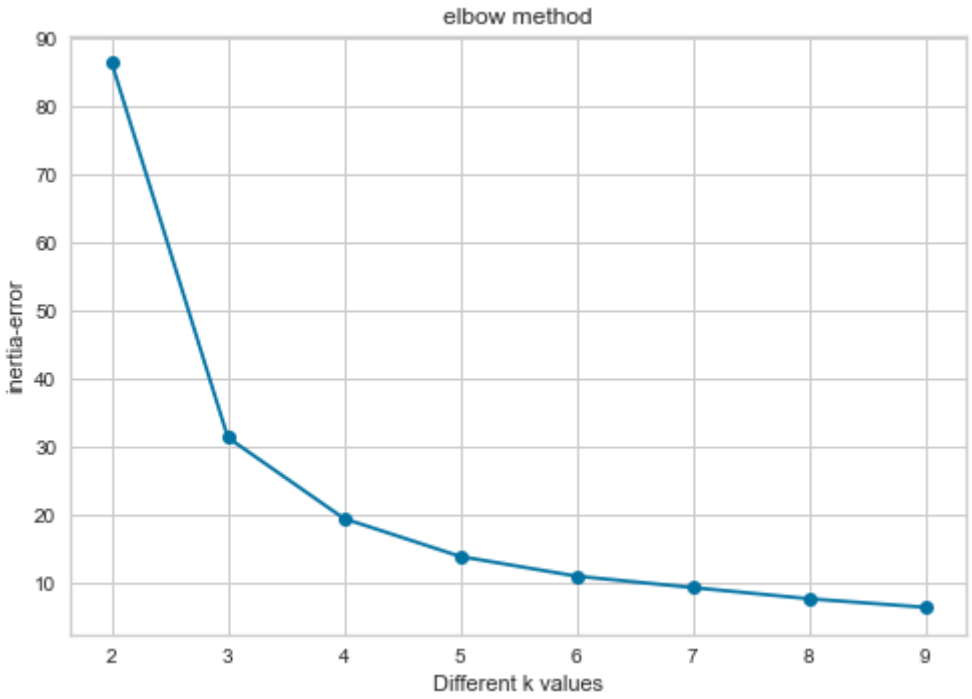
```
ssd = []

K = range(2,10)

for k in K:
    model3 = KMeans(n_clusters =k)
    model3.fit(X2)
    ssd.append(model3.inertia_)
```

In [52]:

```
plt.plot(K, ssd, "bo-")
plt.xlabel("Different k values")
plt.ylabel("inertia-error")
plt.title("elbow method")
```

Out[52]:

```
Text(0.5, 1.0, 'elbow method')
```



In [53]:

```
df_diff = pd.DataFrame(-pd.Series(ssd).diff()).rename(index = lambda x : x+1)
df_diff
```

Out[53]:

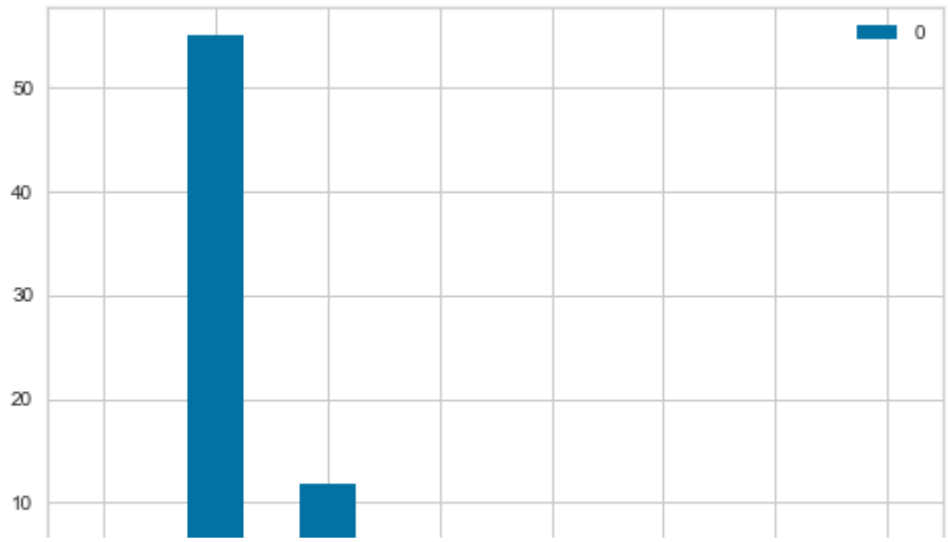
	0
1	NaN
2	55.018861
3	11.888358
4	5.566092
5	2.891764
6	1.675506
7	1.642246
8	1.250899

In [54]:

```
df_diff.plot(kind='bar')
```

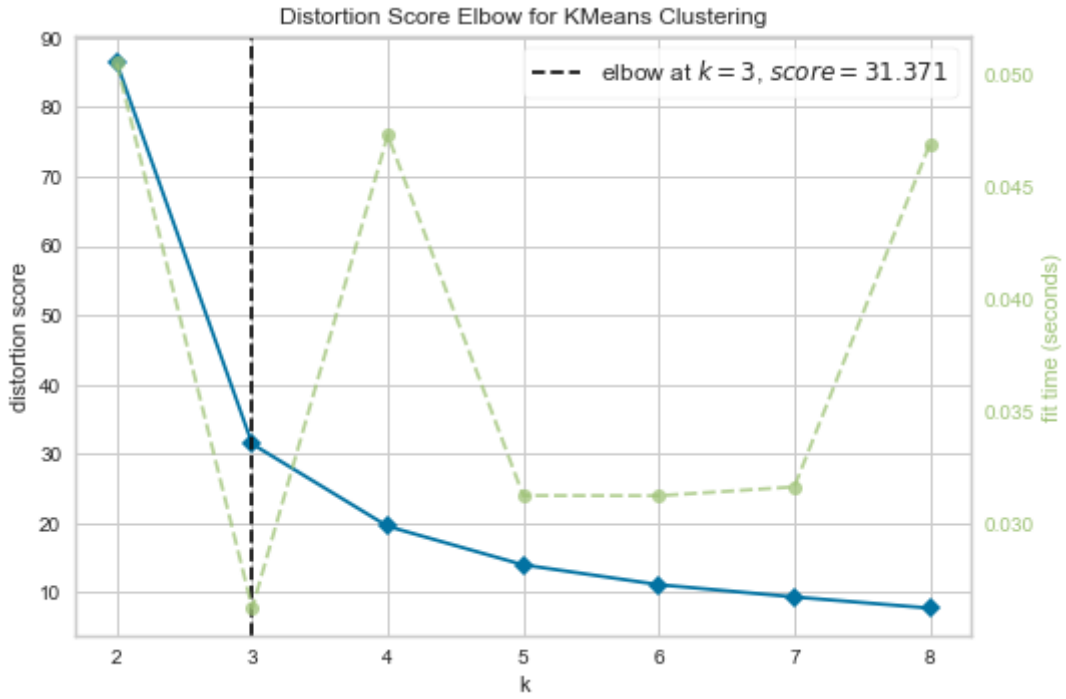
Out[54]:

```
<AxesSubplot:>
```



In [55]:

```
from yellowbrick.cluster import KElbowVisualizer

model_ = KMeans(random_state=42)
visualizer = KElbowVisualizer(model_, k=(2,9))

visualizer.fit(X2)        # Fit the data to the visualizer
visualizer.show();
```



Silhouette analysis

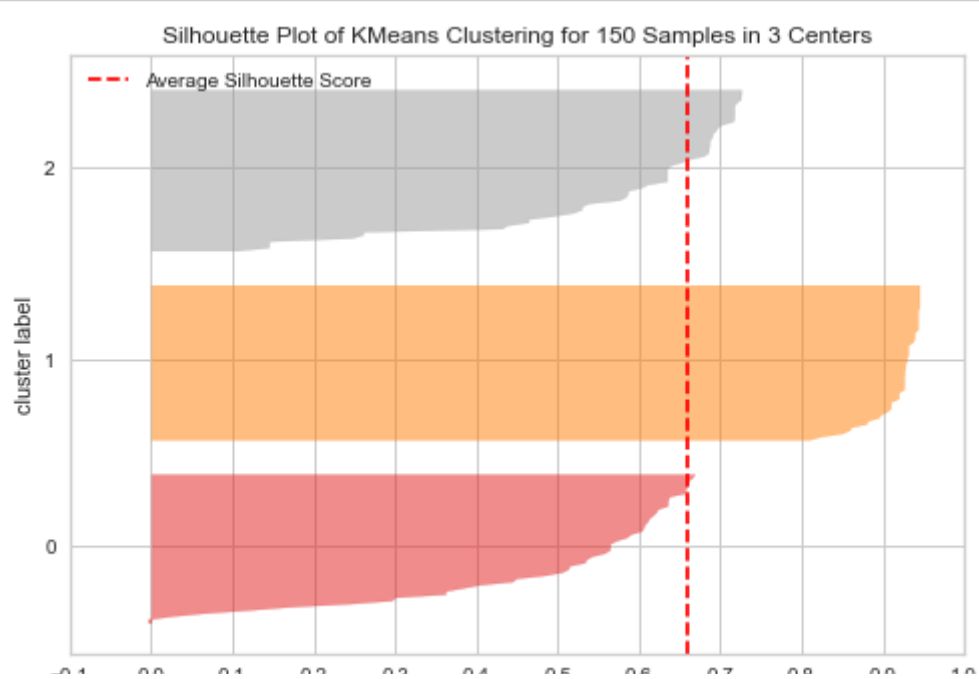
In [56]:

```
range_n_clusters = range(2,9)
for num_clusters in range_n_clusters:
    # intialise kmeans
    kmeans = KMeans(n_clusters=num_clusters, random_state=42)
    kmeans.fit(X2)
    cluster_labels = kmeans.labels_
    # silhouette score
    silhouette_avg = silhouette_score(X2, cluster_labels)
    print(f"For n_clusters={num_clusters}, the silhouette score is {silhouette_avg}")
```

```
For n_clusters=2, the silhouette score is 0.7653904101258123
For n_clusters=3, the silhouette score is 0.6604800083974887
For n_clusters=4, the silhouette score is 0.6127580794464402
For n_clusters=5, the silhouette score is 0.5883732712110276
For n_clusters=6, the silhouette score is 0.576292818723561
For n_clusters=7, the silhouette score is 0.5640984340524555
For n_clusters=8, the silhouette score is 0.5902255624998718
```

In [57]:

```
from sklearn.cluster import KMeans

from yellowbrick.cluster import SilhouetteVisualizer

model3 = KMeans(3, random_state=42)
visualizer = SilhouetteVisualizer(model3)

visualizer.fit(X2)    # Fit the data to the visualizer
visualizer.poof();
```



Building the model based on the optimal number of clusters

In [58]:

```
final_model = KMeans(n_clusters =3, random_state=42)
final_model.fit_predict(X2)
```

Out[58]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [59]:

```
X2["species"] = df["species"]
```

In [60]:

```
X2["predicted_clusters"] = final_model.labels_
```

In [61]:

```
X2.head()
```

Out[61]:

	petal_length	petal_width	species	predicted_clusters
0	1.4	0.2	setosa	1
1	1.4	0.2	setosa	1
2	1.3	0.2	setosa	1
3	1.5	0.2	setosa	1
4	1.4	0.2	setosa	1

Compare results

In [62]:

```
# ct for 2 features
pd.crosstab(X2.predicted_clusters, X2.species)
```

Out[62]:

species	setosa	versicolor	virginica
predicted_clusters			
0	0	2	46
1	50	0	0
2	0	48	4

In [63]:

```
# ct for all features
ct
```

Out[63]:

species	setosa	versicolor	virginica
predicted_clusters			
0	0	0	12
1	0	49	38
2	50	1	0

Prediction cluster of new data

In [64]:

```
new_data = [[1.7, 0.2]]
```

In [65]:

```
final_model.predict(new_data)
```

Out[65]:

```
array([1])
```