Hoja de trabajo 4: Clustering

Segmentación de Especies con Cluster Analysis

Repositorio

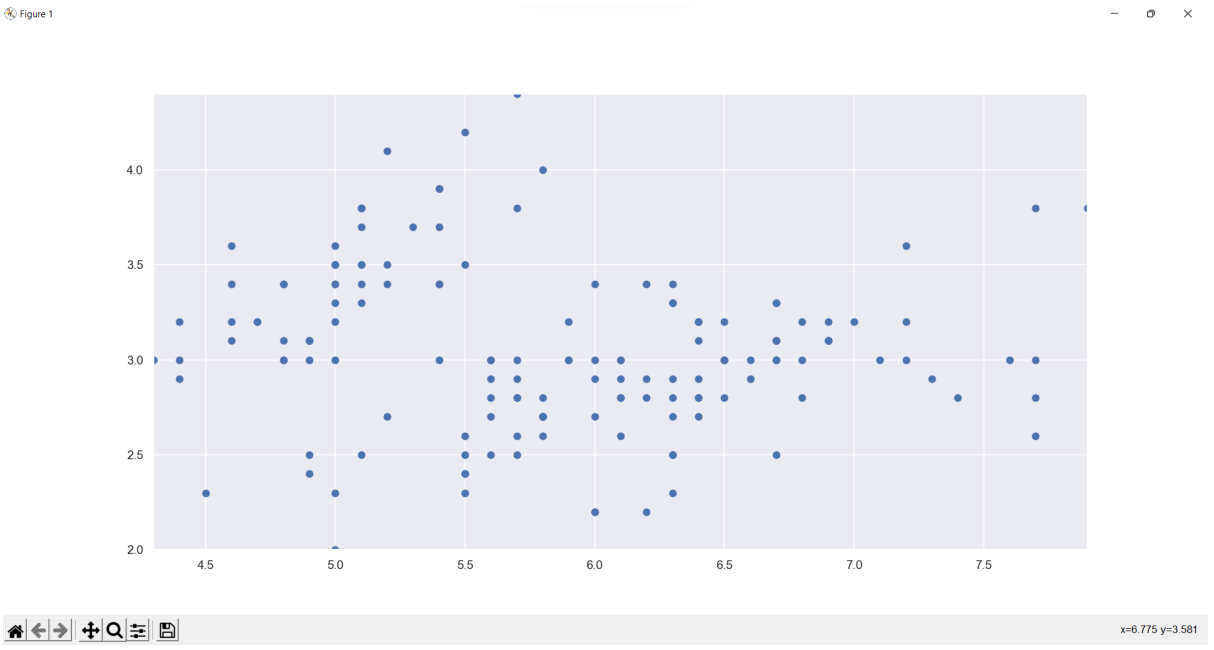
https://github.com/LPELCRACK896/DM-hdt4-iris-clustering.git // no me dejó subir el zip.

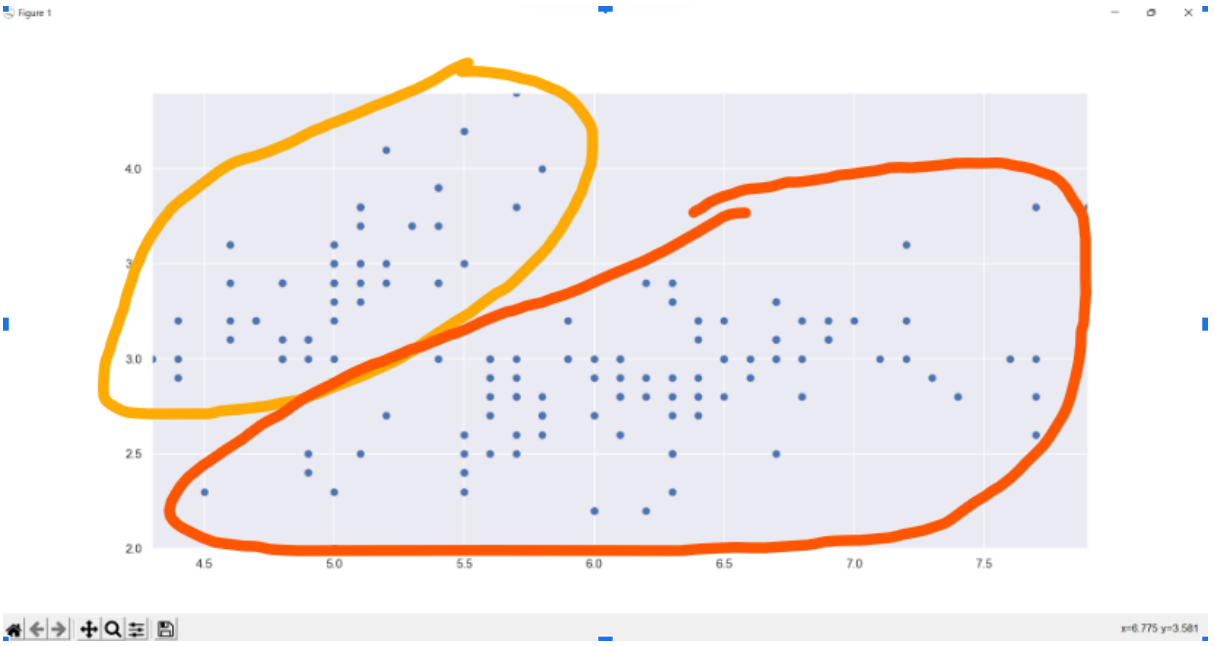
Resumen

Sección 1

1. Visualicen los datos para ver si pueden detectar algunos grupos. **Ayuda:** utilicen la forma del sépalo.

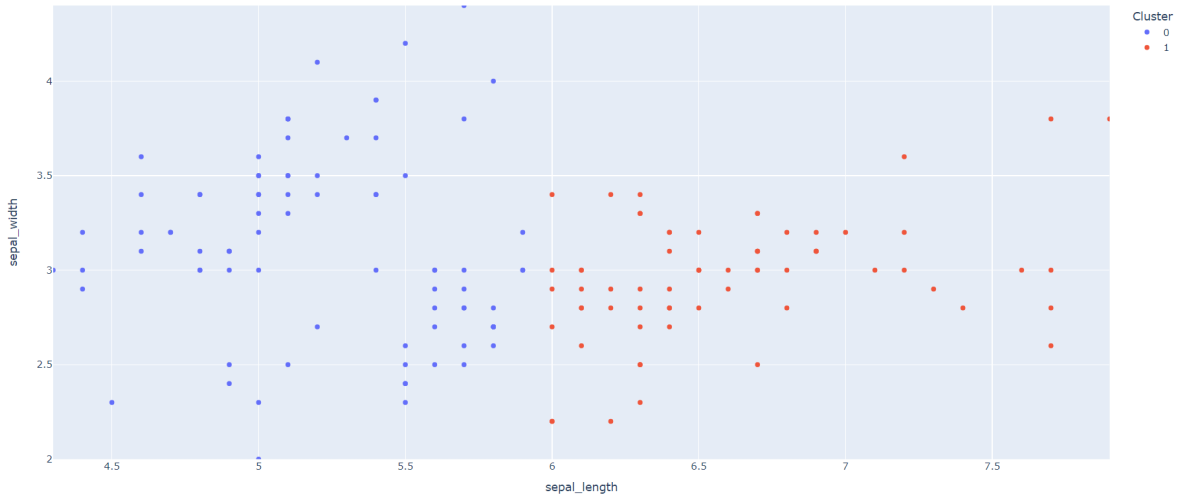
```
plt.scatter(data['sepal_length'], data['sepal_width'])
plt.xlim(data['sepal_length'].min(), data['sepal_length'].max())
plt.ylim(data['sepal_width'].min(), data['sepal_width'].max())
plt.show()
```





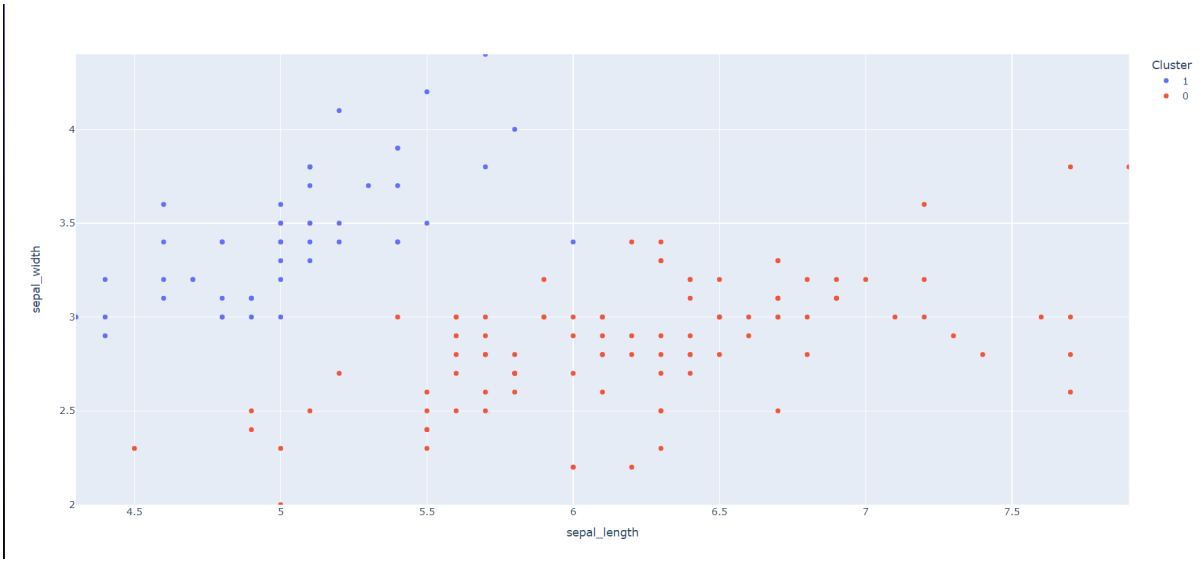
2. Creen 2 "clusters" utilizando K_Means Clustering y grafiquen los resultados.

```
# 2. Visualicen los datos para ver si pueden detectar algunos grupos. Ayuda: utilicen la forma del sépalo
X = data.filter(["sepal_length", "sepal_width"])
kmeans = KMeans(2)
kmeans.fit(X)
clusters = kmeans.fit_predict(X)
data_sec1 = data.copy()
data_sec1["Cluster"] = clusters
data_sec1["Cluster"] =data_sec1["Cluster"].astype("category")
fig = px.scatter(data_sec1, x = "sepal_length", y = "sepal_width", color = "Cluster", hover_data = ['Cluster'])
fig.update_xaxes(range=[min_sepal_length, max_sepal_length])
fig.update_yaxes(range=[min_sepal_width, max_sepal_width])
fig.show()
```



3. Estandaricen los datos e intenten el paso 2, de nuevo. ¿Qué diferencias hay, si es que lo hay?

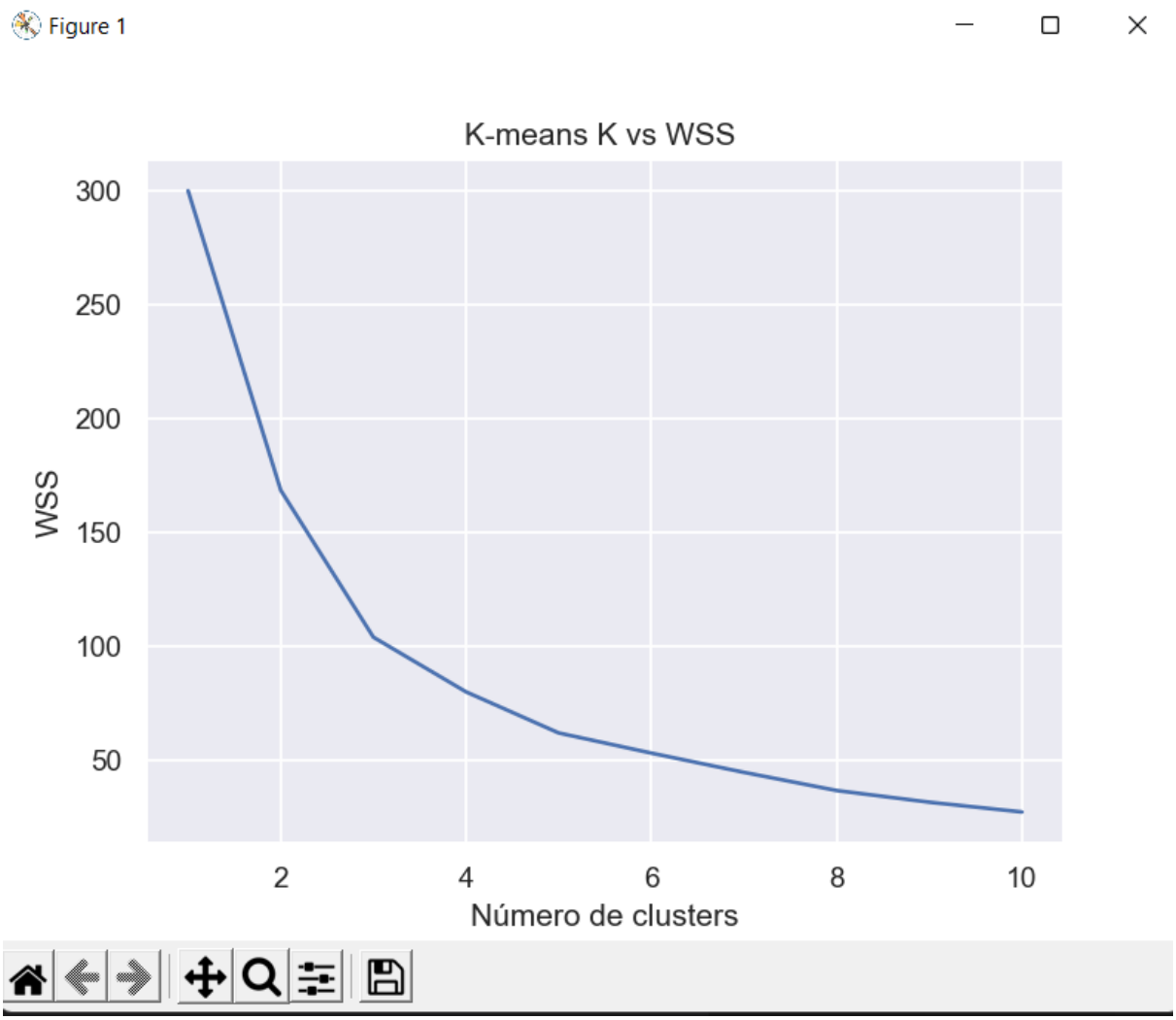
```python
# 3 Estandaricen los datos e intenten el paso 2, de nuevo. ¿Qué diferencias hay, si es que lo hay?
scaler = StandardScaler()
X_standarized = X.copy()
X_standarized[X_standarized.columns] = scaler.fit_transform(X_standarized[X_standarized.columns])
kmeans = KMeans(2)
kmeans.fit(X_standarized)
clusters = kmeans.fit_predict(X_standarized)
data_sec1_3 = data.copy()
data_sec1_3["Cluster"] = clusters
data_sec1_3["Cluster"] =data_sec1_3["Cluster"].astype("category")
fig = px.scatter(data_sec1_3, x = "sepal_length", y = "sepal_width", color = "Cluster", hover_data = ['Cluster'])
fig.update_xaxes(range=[min_sepal_length, max_sepal_length])
fig.update_yaxes(range=[min_sepal_width, max_sepal_width])
fig.show()
```



Si cambió el cluster. Esto debido a que el escalamiento altera las distancias calculadas hacia los centroides. En este caso puede funcionar bien porque permite comparar las columnas de los datos sobre una comparación más uniforme entre columnas.

4. Utilicen el método del "codo" para determinar cuantos "clusters" es el ideal. (prueben un rango de 1 a 10)

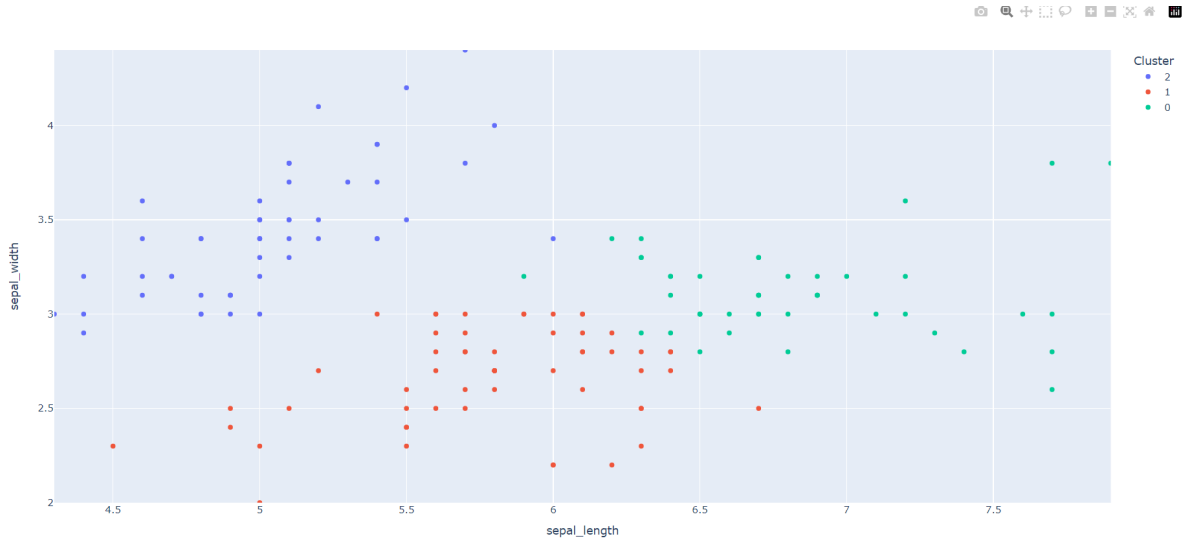
```python
# 4 Utilicen el método del "codo" para determinar cuantos "clusters" es el ideal. (prueben un rango de 1 a 10)
wss = []
x_graph = range(1, 11)
for i in range(1, 11):
    kmeans = KMeans(i)
    kmeans.fit(X_standarized)
    wss.append(kmeans.inertia_)
plt.plot(x_graph, wss)
plt.title('K-means K vs WSS ')
plt.xlabel('Número de clusters')
plt.ylabel('WSS')
plt.show()
```

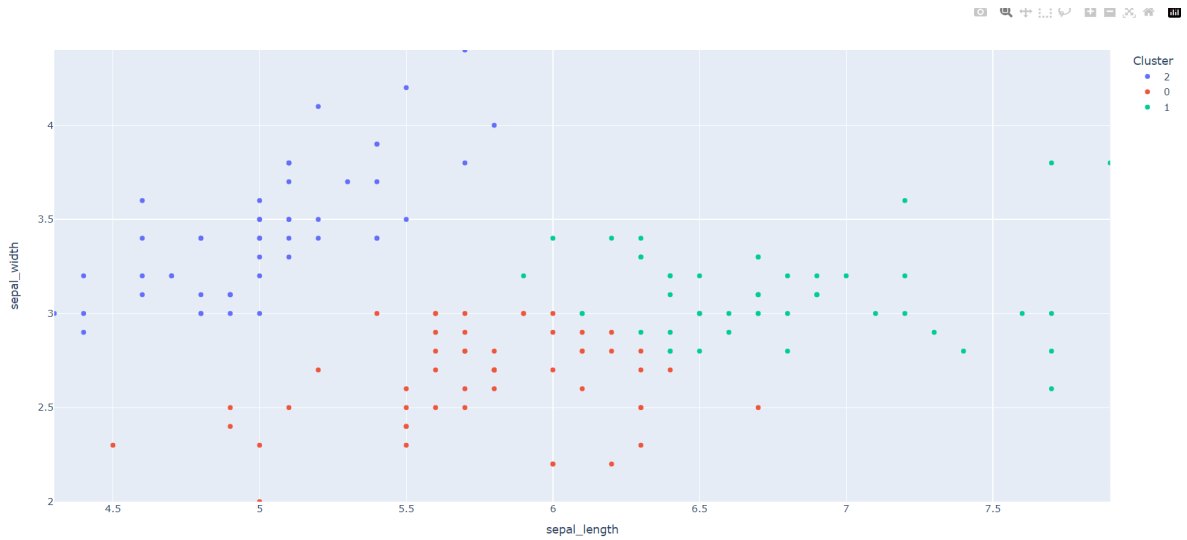


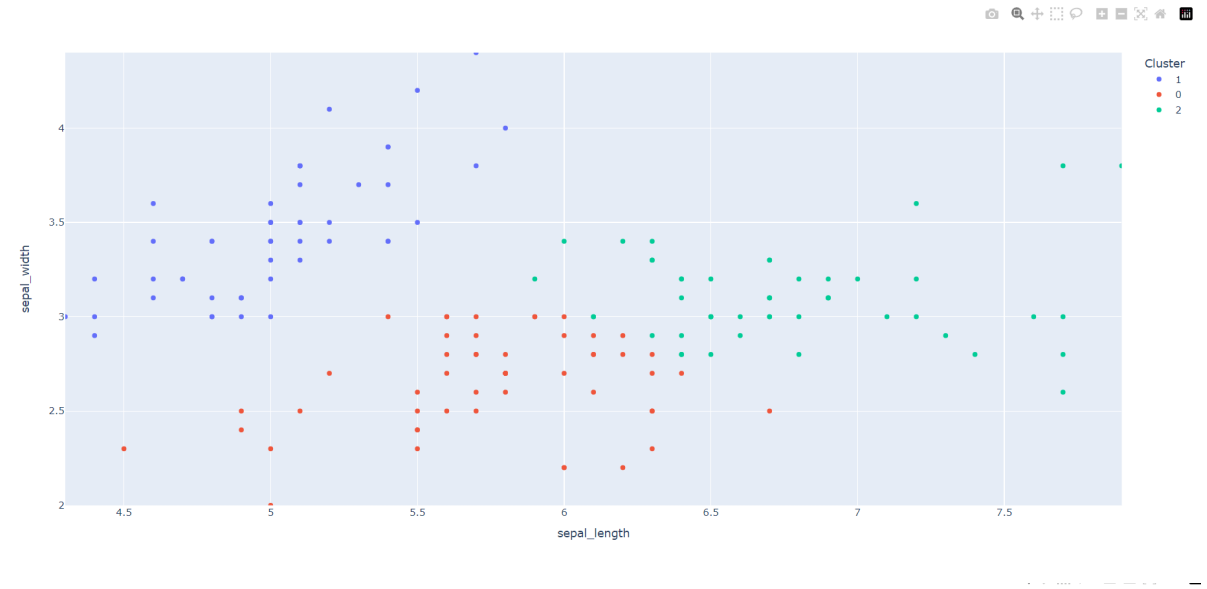
El punto donde hay más cambio de una pendiente a otra es con k=3, por tanto el número de clusters ideal es 3.

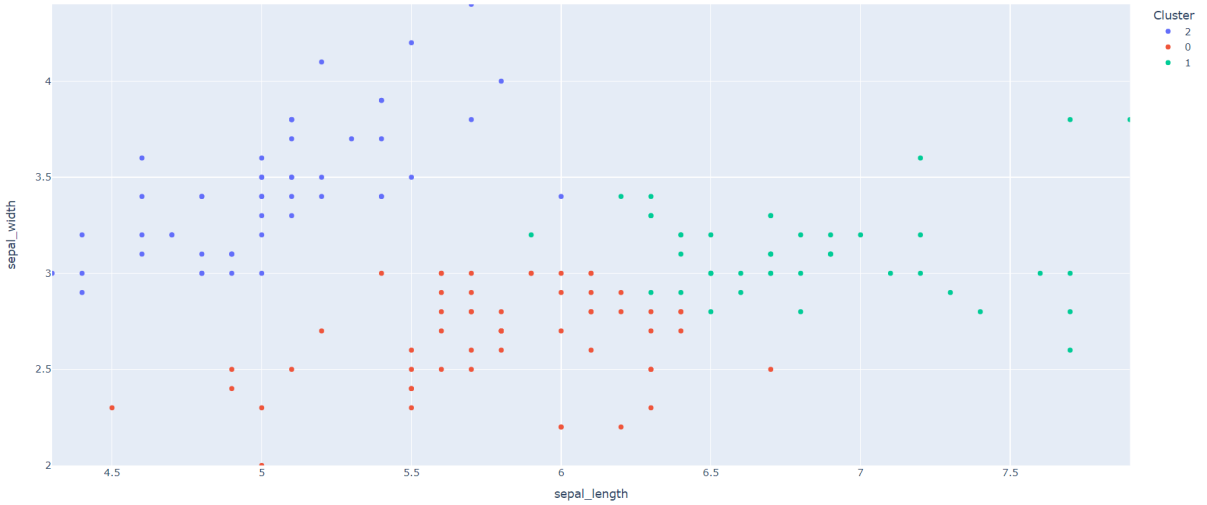
5. Basado en la gráfica del "codo" realicen varias gráficas con el número de clusters (unos 3 o 4 diferentes) que Uds creen mejor se ajusten a los datos.

```
#5 Basado en la gráfica del "codo" realicen varias gráficas con el número de clusters (unos 3 o 4 diferentes) que Uds cre
k = 3
for _ in range(4):
    kmeans = KMeans(k)
    kmeans.fit(X_standarized)
    clusters = kmeans.fit_predict(X_standarized)
    data_sec1_5 = data.copy()
    data_sec1_5["Cluster"] = clusters
    data_sec1_5["Cluster"] =data_sec1_5["Cluster"].astype("category")
    fig = px.scatter(data_sec1_5, x = "sepal_length", y = "sepal_width", color = "Cluster", hover_data = ['Cluster'])
    fig.update_xaxes(range=[min_sepal_length, max_sepal_length])
    fig.update_yaxes(range=[min_sepal_width, max_sepal_width])
    fig.show()
```

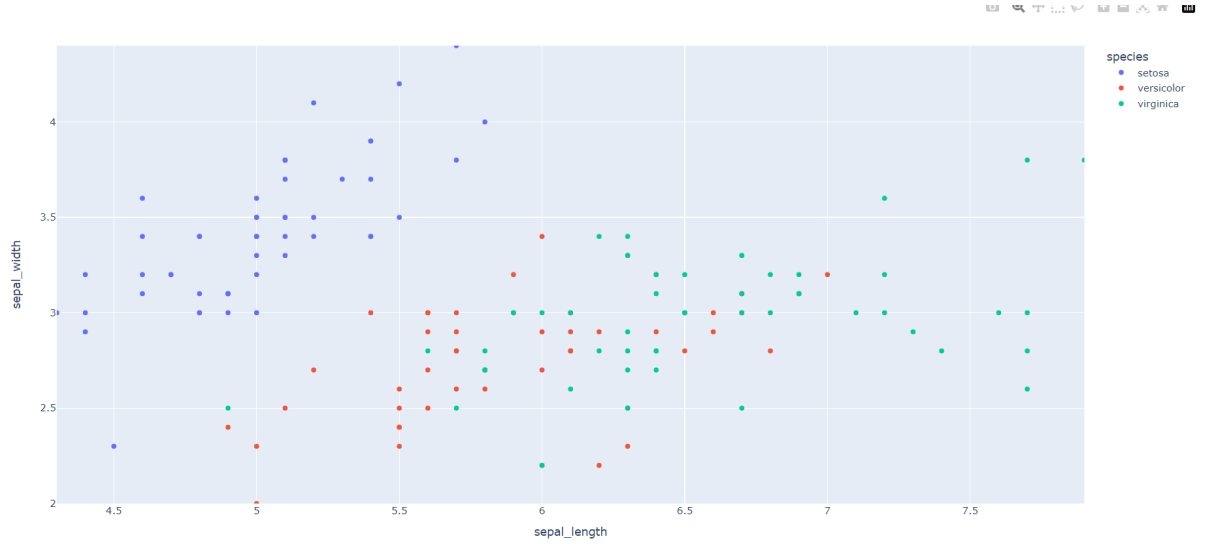

Cluster
2
1
0
sepal_width
sepal_length
4
3.5
3
2.5
2
4.5
5
5.5
6
6.5
7
7.5


Cluster
2
0
1
sepal_width
sepal_length
4
3.5
3
2.5
2
4.5
5
5.5
6
6.5
7
7.5





6. Comparen sus soluciones con los datos reales, archivo: iris-con-respuestas.csv

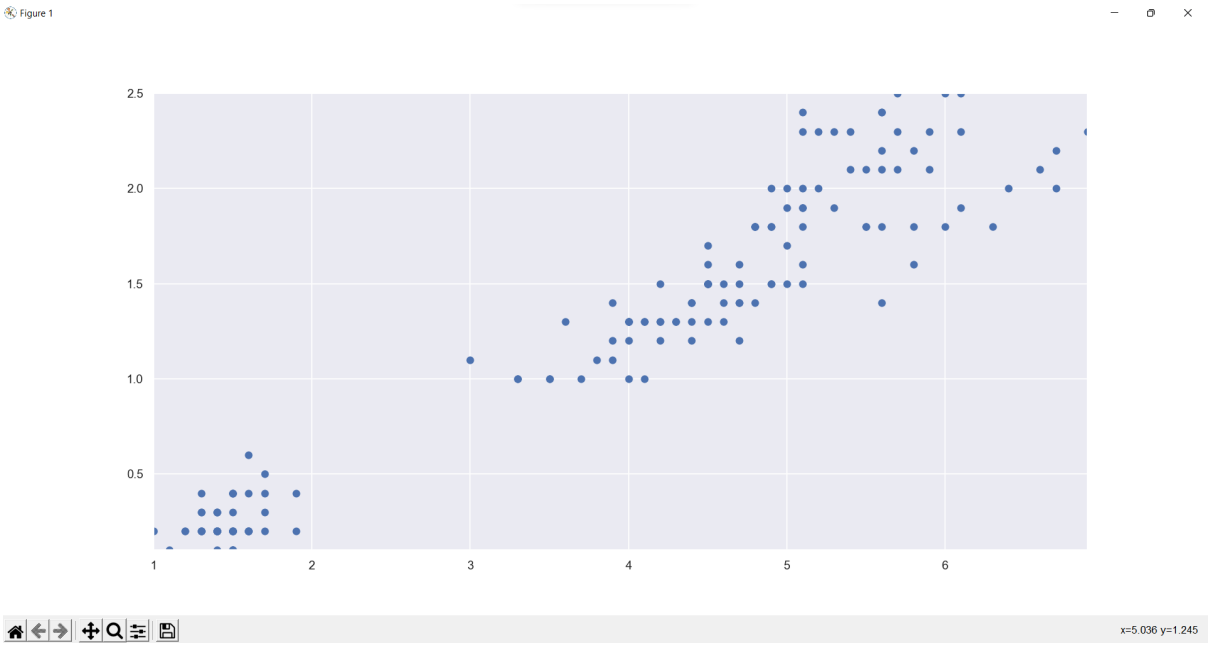


Omitiendo uno que otro dato atípico de la especie virginica, sin embargo, no fue posible predecir la frontera tan difusa e incierta que existe entre versicolor y virginica.

Sección 2

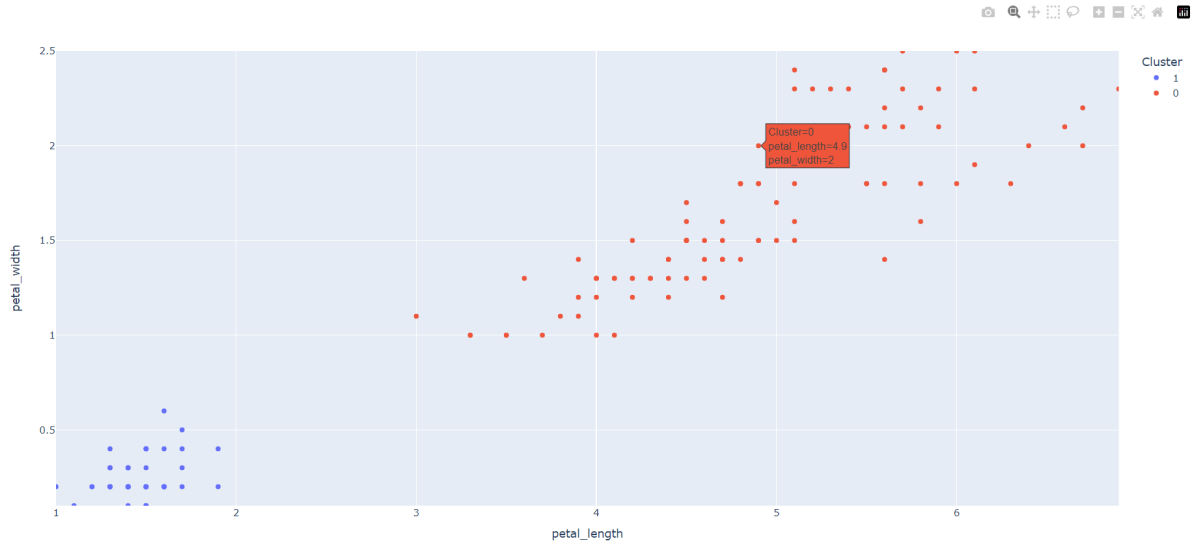
Para no repetir código se transformó el código a una función que recibe como parámetros los features sobre los cuales se desea calcular los clusters.

1. Visualicen los datos para ver si pueden detectar algunos grupos.



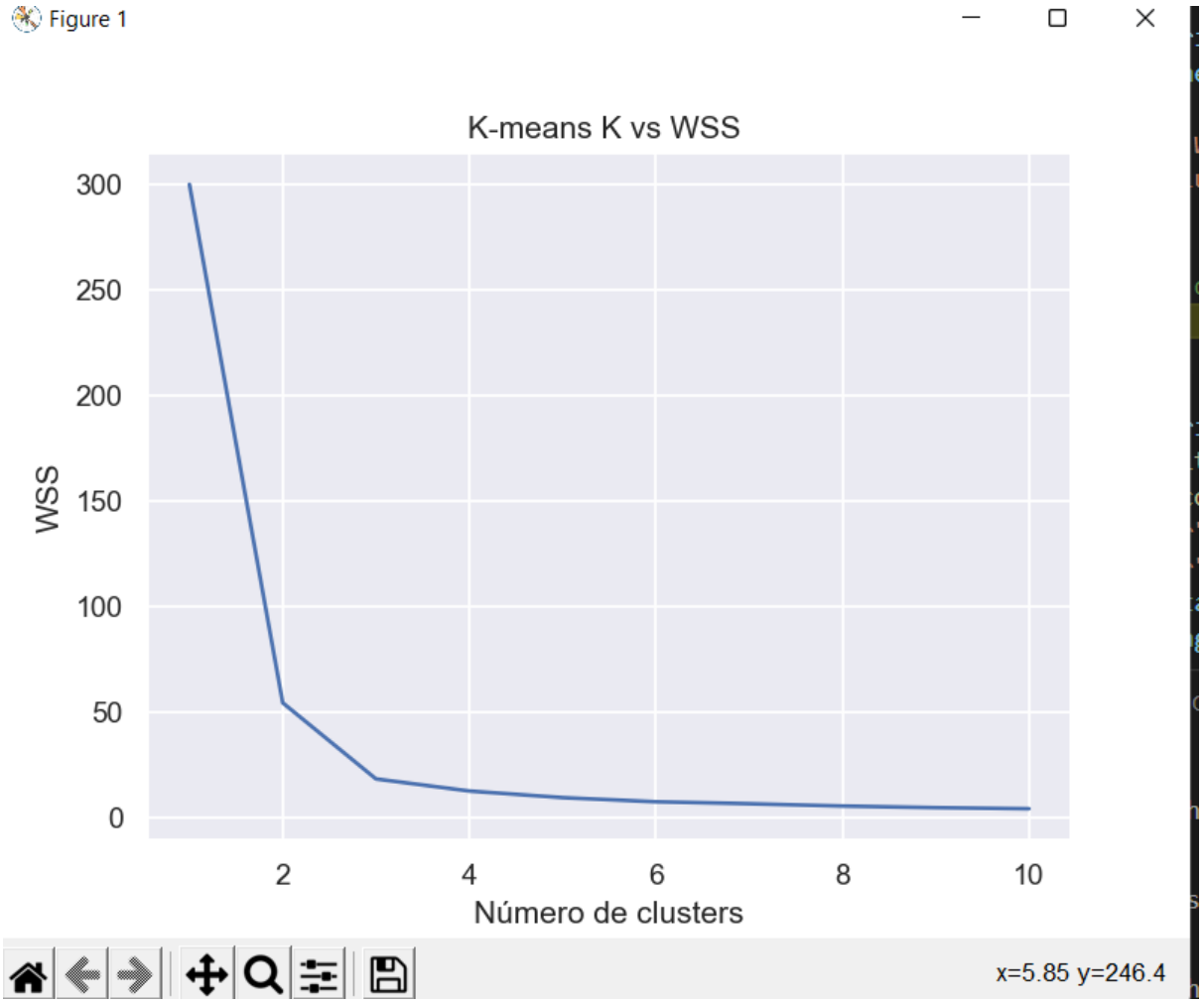
2. Creen 2 "clusters" utilizando K_Means Clustering y grafiquen los resultados.

3. Estandaricen los datos e intenten el paso 2, de nuevo. ¿Qué diferencias hay, si es que lo hay?



Los clusters permanecieron similares pero hubo una mejor con la asignación de algunos puntos en frontera; aunque esto podría atribuirse a las condiciones iniciales, no necesariamente a la estandarización.

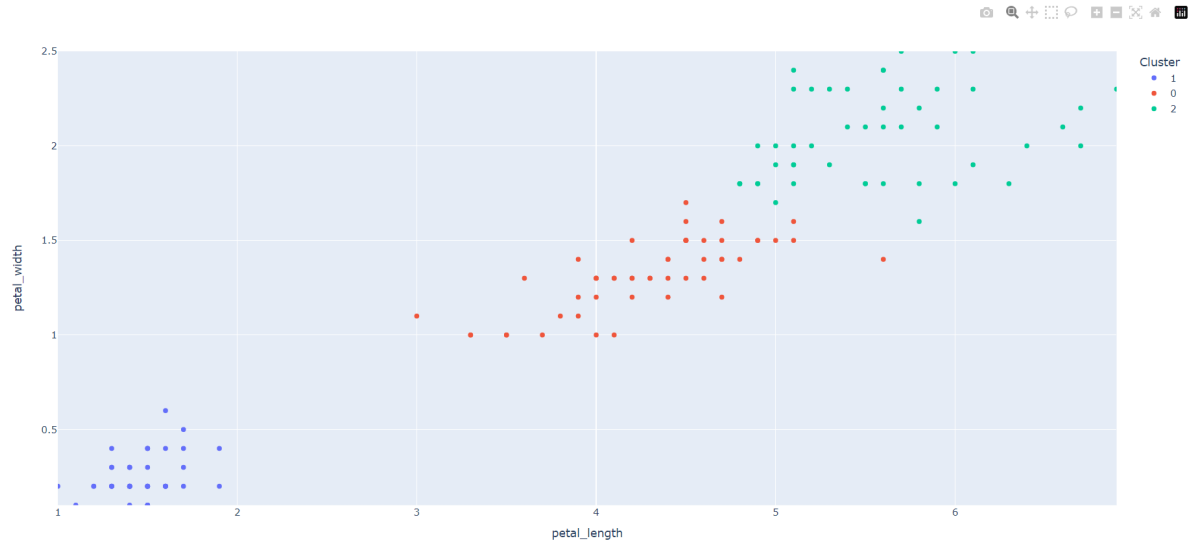
4. Utilicen el método del "codo" para determinar cuantos "clusters" es el ideal. (prueben un rango de 1 a 10)

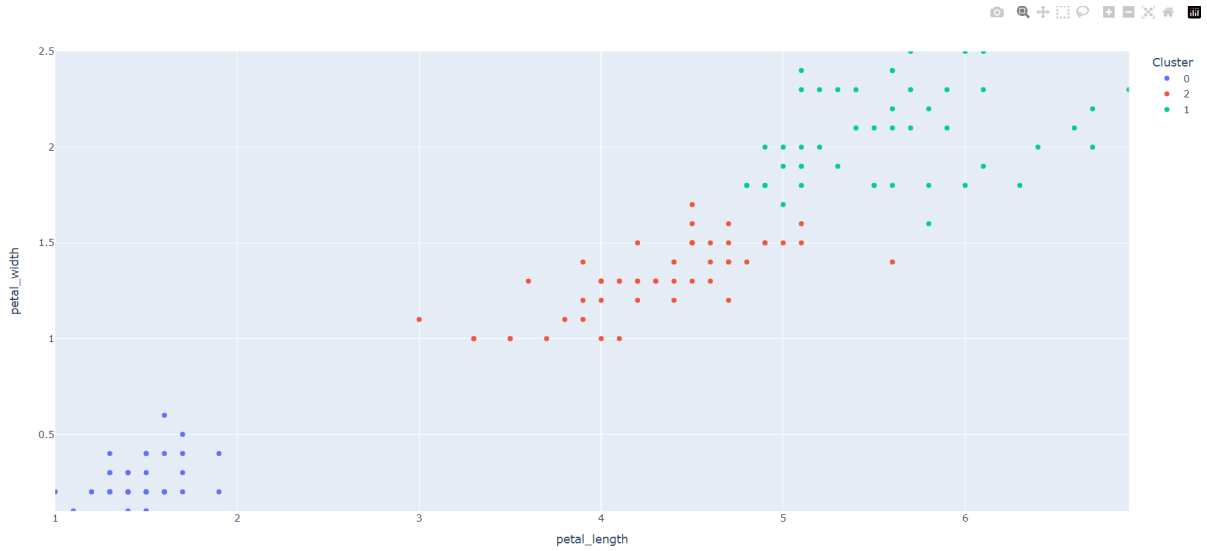


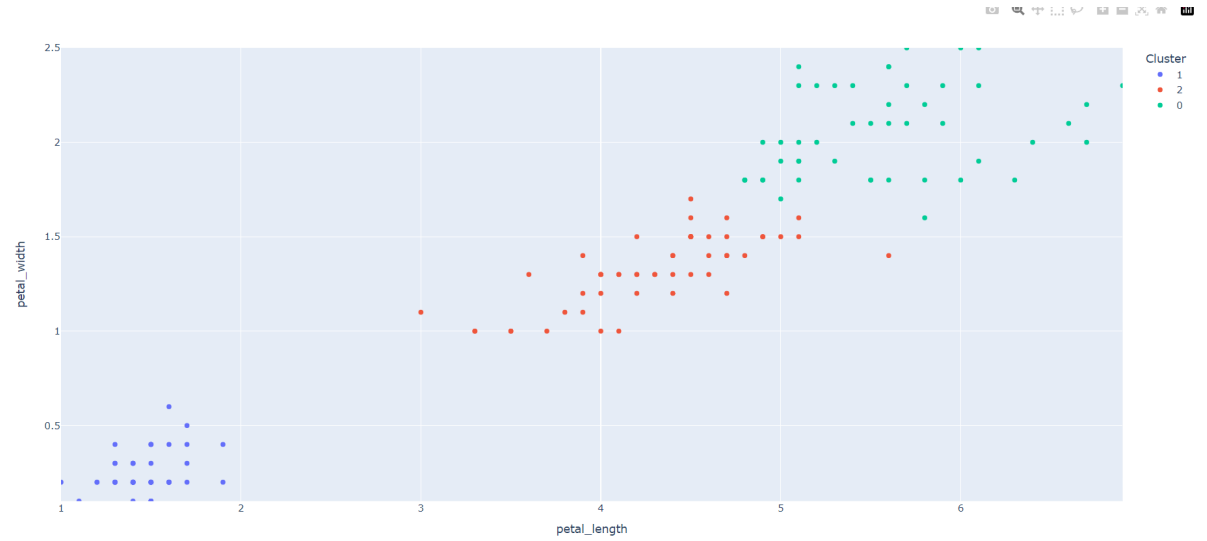
El codo indica que la k ideal está en 3 .

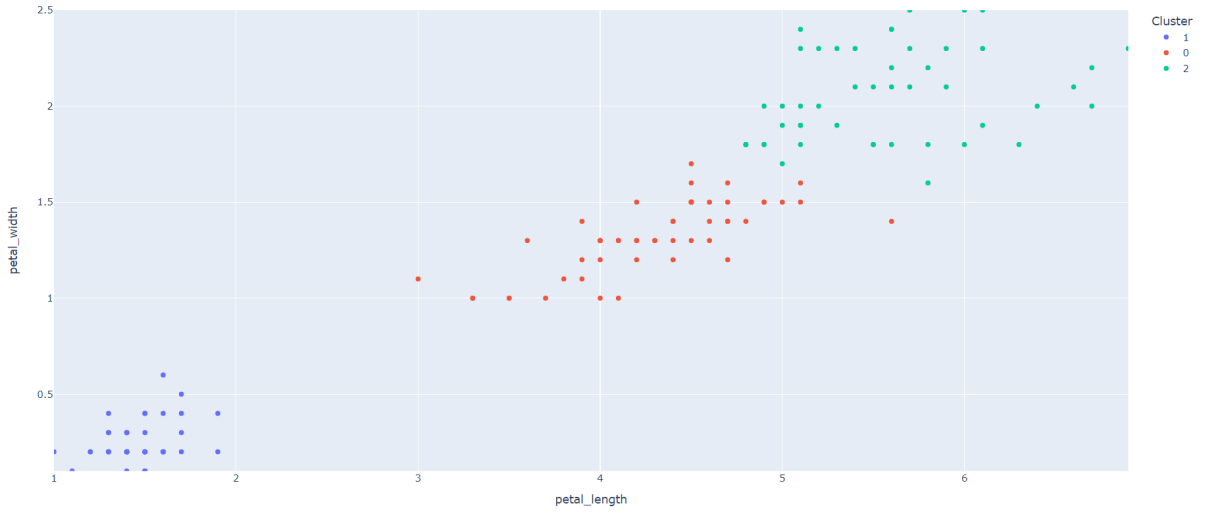
5. Basado en la gráfica del "codo" realicen varias gráficas con el número de clusters (unos 3 o 4 diferentes) que Uds creen mejor se ajusten a los datos.

con k = 3



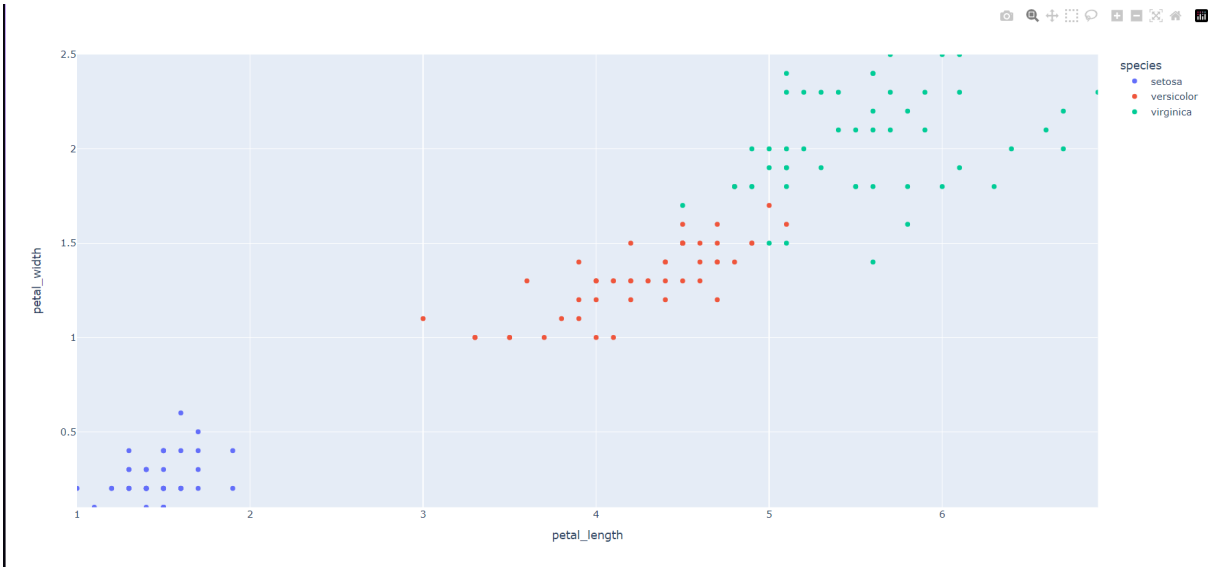






Practicamente mismos clusters.

6. Comparen sus soluciones con los datos reales, archivo: iris-con-respuestas.csv



El cluster queda aún mejor que utilizando la forma del sépalo. Una mejor forma de diferencias entre estas categorias esta en clasificarlas por el la forma del petalo. El cluster queda prácticamente idéntico, mejorando el caso de las fronteras y separando los clusters sin entrelazarse prácticamente nada.

Sección 3

```python
def get_optimal_k(feature_1, feature_2):
    DATA_PATH = "./data"
    data = pd.read_csv(f"{DATA_PATH}/iris.csv")
    X = data.filter([feature_1, feature_2])
    scaler = StandardScaler()
    X_standarized = X.copy()
    X_standarized[X_standarized.columns] = scaler.fit_transform(X_standarized[X_standarized.columns])

    wss = []
    x_graph = range(1, 11)
    for i in range(1, 11):
        kmeans = KMeans(i)
        kmeans.fit(X_standarized)
        wss.append(kmeans.inertia_)
    kl = KneeLocator(x_graph, wss, curve="convex", direction="decreasing")
    optimal_k = kl.elbow
    print(f"El k optimo encontrado para el clustering con {feature_1} y {feature_2}: {optimal_k}")
```

```
rning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
o suppress the warning
  warnings.warn(
El k optimo encontrado para el clustering con petal_length y petal_width: 3
```

El algoritmo mostró el mismo resultado de clusters ideales. La conclusión es que kmeans , apoyado del método del coso presenta una forma eficiente de clasificar datos, sin embargo la elección de las features para la construcción de estos clusters son un paso fundamental en alcanzar la mayor precisión con los clusters.