Ingeniería de Datos Masivos

HACKATHON

- IRIS
- TELESCOPIO
- POKEMON

PABLO ELÍAS RAMÍREZ ESCALANTE
MARIANA ALMAGUER GONZÁLEZ
IAN EMMANUEL REYES YAN
KAROL ALEJANDRA MAR PALACIOS
ALAN RODRIGO CEBALLOS MORENO

IRIS

LIMPIEZA Y ESTRUCTURA DE DATOS

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
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
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

```
[77] # Separar 3 especies
     df['Species'] = df['Species'].apply(lambda x: 0 if x == 'Iris-setosa' else (1 if x == 'Iris-versicolor' else (2 if x == 'Iris-virginica' else 'Other')))
```

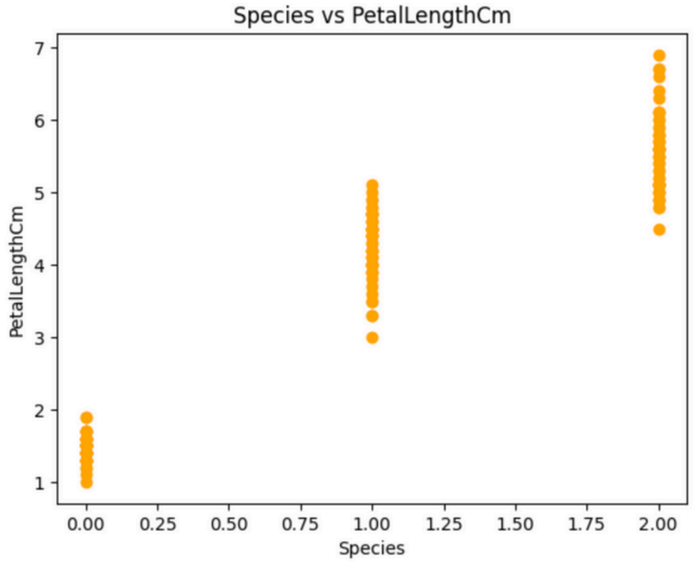
IRIS

OBTENCIÓN DE DUMMIES

	Species	SepalLengthCm_4.3	SepalLengthCm_4.4	SepalLengthCm_4.5	SepalLengthCm_4.6	SepalLengthCm_4.7	SepalLengthCm_4.8	SepalLengthCm_4.9
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	1	0	0
3	0	0	0	0	1	0	0	0
4	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
145	2	0	0	0	0	0	0	0
146	2	0	0	0	0	0	0	0
147	2	0	0	0	0	0	0	0
148	2	0	0	0	0	0	0	0
149	2	0	0	0	0	0	0	0

150 rows × 124 columns

```
# Crear dos DataFrames X con variables indpendientes o features y uno y con variable dependiente , la que vamos a predecir
X = df.drop('Species',1)
Y = df.Species
```

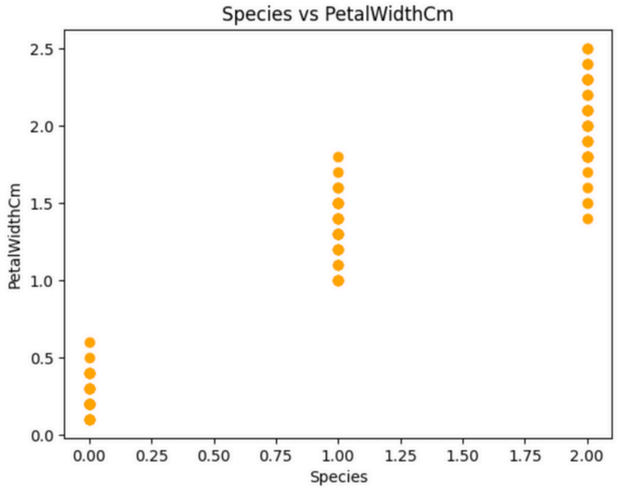


SELECT KBEST = 55

```
from sklearn.feature_selection import SelectKBest, chi2

select = SelectKBest(chi2,k=55) #124 es el total, 62 es la mitad y 42 es la tercera parte
selected_features = select.fit(X, Y)
indices_selected = selected_features.get_support(indices=True)
colnames_selected = [df.columns[i] for i in indices_selected]

X = df[colnames_selected]
X
```

	SepalLengthCm_4.3	SepalLengthCm_4.5	SepalLengthCm_4.7	SepalLengthCm_4.9	SepalLengthCm_5.0	SepalLengthCm_5.3	SepalLengthCm_5.4	SepalLengthCm_5.5
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...
145	0	0	0	0	0	0	0	0
146	0	0	0	0	0	0	0	0
147	0	0	0	0	0	0	0	0
148	0	0	0	0	0	0	0	0
149	0	0	0	0	0	0	0	0

150 rows × 55 columns



IRIS

LIBRERIA SPLIT

```
[46] from sklearn.model_selection import train_test_split
     #Separación los datos de "train" en entrenamiento y prueba para probar los algoritmos
     X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)
     print('Son {} datos para entrenamiento y {} datos para prueba'.format(X_train.shape[0], X_test.shape[0]))
```

```
Son 120 datos para entrenamiento y 30 datos para prueba
```

MODELO DE ÁRBOLES DE DECISIÓN CLASIFICACIÓN

```
[49] from sklearn.tree import DecisionTreeClassifier
     #Modelo de Árboles de Decisión Clasificación
     algoritmo = DecisionTreeClassifier()
     algoritmo.fit(X_train, y_train)
     Y_pred = algoritmo.predict(X_test)
     print('Precisión Árboles de Decisión Clasificación: {}'.format(algoritmo.score(X_train, y_train)))
```

```
Precisión Árboles de Decisión Clasificación: 0.9833333333333333
```

TELESCOPIO

LIMPIEZA Y ESTRUCTURA DE DATOS (Kmeans)

	fPhot	fArrivalTime
0	-0.861328	164.990
1	-1.753910	131.935
2	-1.531250	136.663
3	-2.250000	162.914
4	1.445310	134.650
...	...	...
9965435	-2.367190	141.203
9965436	0.890625	133.252
9965437	0.000000	-1.000
9965438	0.000000	-1.000
9965439	0.000000	-1.000

9965440 rows × 2 columns

```
X = pd.DataFrame()
X.loc[:,0] = df.loc[:,"fArrivalTime"]
X.loc[:,1] = df.loc[:,"fPhot"]

cls = KMeans(init="k-means++", n_clusters=7, n_init = 10)
cls.fit(X)

X["predicted_label"] = cls.labels_.astype(int)
X.columns = ["fArrivalTime","fPhot","label"]
X
```

	fArrivalTime	fPhot	label
0	164.990	-0.861328	2
1	131.935	-1.753910	0
2	136.663	-1.531250	4
3	162.914	-2.250000	2
4	134.650	1.445310	0
...	...	...	...
9965435	141.203	-2.367190	4
9965436	133.252	0.890625	0
9965437	-1.000	0.000000	1
9965438	-1.000	0.000000	1
9965439	-1.000	0.000000	1

9965440 rows × 3 columns

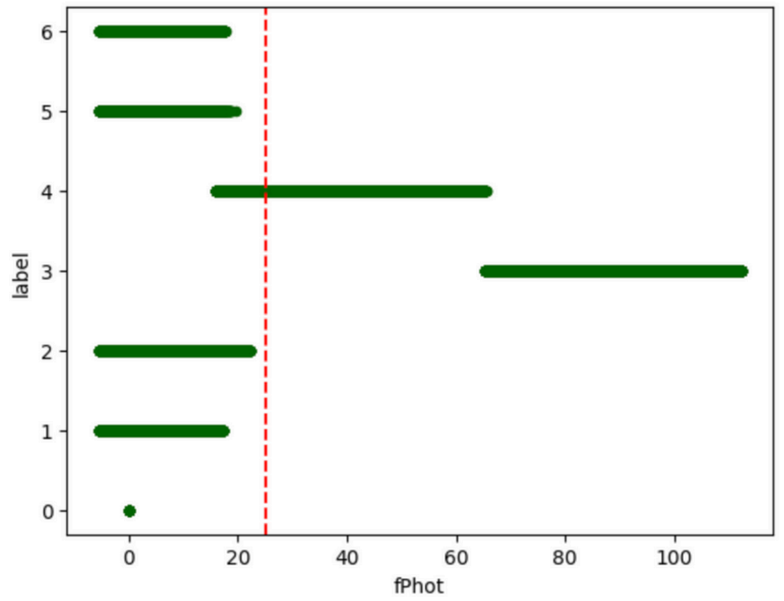
TELESCOPIO

```
# Filtrar filas con fPhot <= 25
ruido = X[X['fPhot'] <= 25]
labels_ruido = ruido['label'].unique()

# Filtrar filas con fPhot > 25
señal = X[X['fPhot'] > 25]
labels_señal = señal['label'].unique()

print("Labels con fPhot <= 25:", labels_ruido)
print("Labels con fPhot > 25:", labels_señal)
```

```
Labels con fPhot <= 25: [2 5 1 6 4 0]
Labels con fPhot > 25: [3 4]
```

OBTENCIÓN DE RUIDO Y SEÑAL

FPHOT VS LABEL



TELESCOPIO

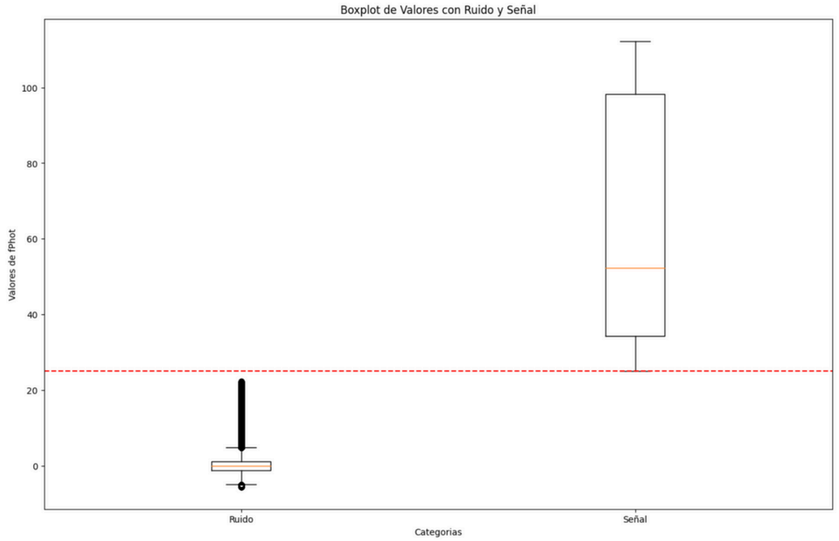
KMEANS

```
label_repetido = [valor for valor in labels_ruido if valor in labels_señal]
label_repetido
```

```
[4]
```

```
# Obtener la cantidad de valores con fPhot <= 25 y fPhot > 25 para label_repetido
label_repetido_ruido = len(X[(X['label'].isin(label_repetido)) & (X['fPhot'] <= 25)])
label_repetido_señal = len(X[(X['label'].isin(label_repetido)) & (X['fPhot'] > 25)])

print("Longitud del ruido en el label repetido{}: {}".format(label_repetido,label_repetido_ruido))
print("Longitud de la señal en el label repetido{}: {}\n".format(label_repetido,label_repetido_señal))

# Actualizar las variables labels_ruido y labels_señal
if label_repetido_ruido > label_repetido_señal:
    labels_señal = labels_señal[labels_señal != label_repetido[0]]  # Remover label_repetido de labels_señal
else:
    labels_ruido = labels_ruido[labels_ruido != label_repetido[0]]  # Remover label_repetido de labels_ruido

print("Labels con fPhot <= 25 (ruido):", labels_ruido)
print("Labels con fPhot > 25 (señal):", labels_señal)
```

```
Longitud del ruido en el label repetido[4]: 66205
Longitud de la señal en el label repetido[4]: 104120

Labels con fPhot <= 25 (ruido): [2 5 1 6 0]
Labels con fPhot > 25 (señal): [3 4]
```




```
grafica_ruido = ruido.copy()
grafica_ruido['label'] = 0
grafica_señal = señal.copy()
grafica_señal['label'] = 1
```

```
ruido = ruido[(ruido['label'].isin(labels_ruido))]
señal = señal[(señal['label'].isin(labels_señal))]
```

TELESCOPIO

Comprobación

```
#Sin Kmeans

# Contar la cantidad de filas con valores <= 25 en la columna 'fPhot'
cantidad_ruido = len(X[X['fPhot'] <= 25])

# Contar la cantidad de filas con valores > 25 en la columna 'fPhot'
cantidad_no_ruido = len(X[X['fPhot'] > 25])

comprobacion_real = cantidad_ruido + cantidad_no_ruido

print(f"Cantidad de valores <= 25 en 'fPhot': {cantidad_ruido}")
print(f"Cantidad de valores > 25 en 'fPhot': {cantidad_no_ruido}")

#Comprobar
print("\nSuma de ruido y señal real (sin usar Kmeans): {}".format(comprobacion_real))
```

```
Cantidad de valores <= 25 en 'fPhot': 9792420
Cantidad de valores > 25 en 'fPhot': 173020

Suma de ruido y señal real (sin usar Kmeans): 9965440
```

```
comprobacion_Kmean = len(ruido)+len(señal)
print("Suma de ruido y señal usando Kmeans: {}".format(comprobacion_Kmean))
```

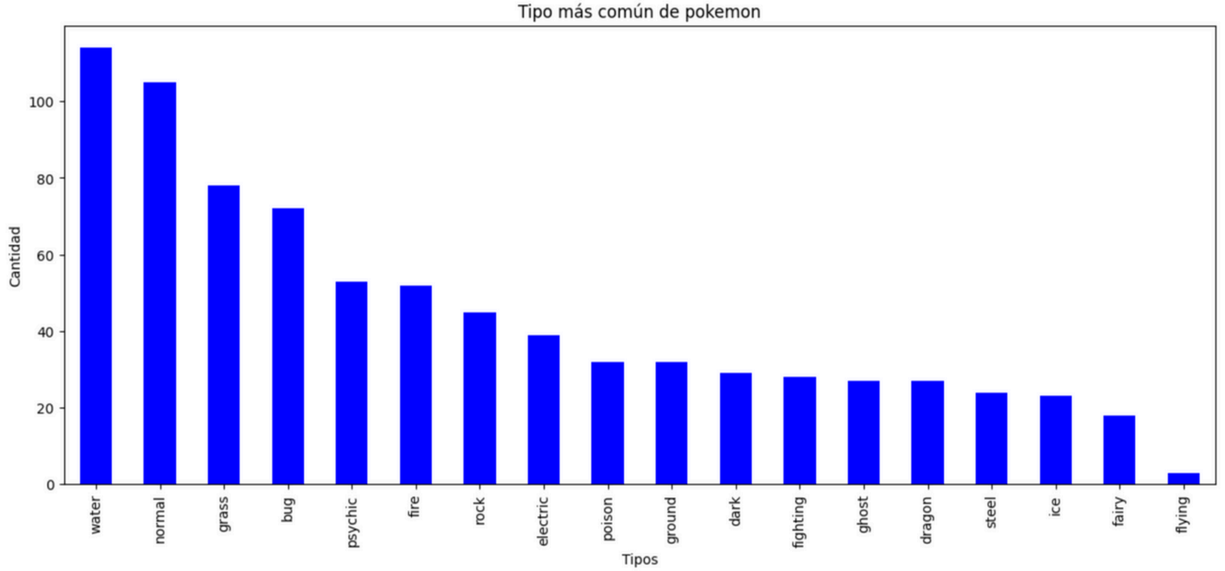
```
Suma de ruido y señal usando Kmeans: 9899235
```

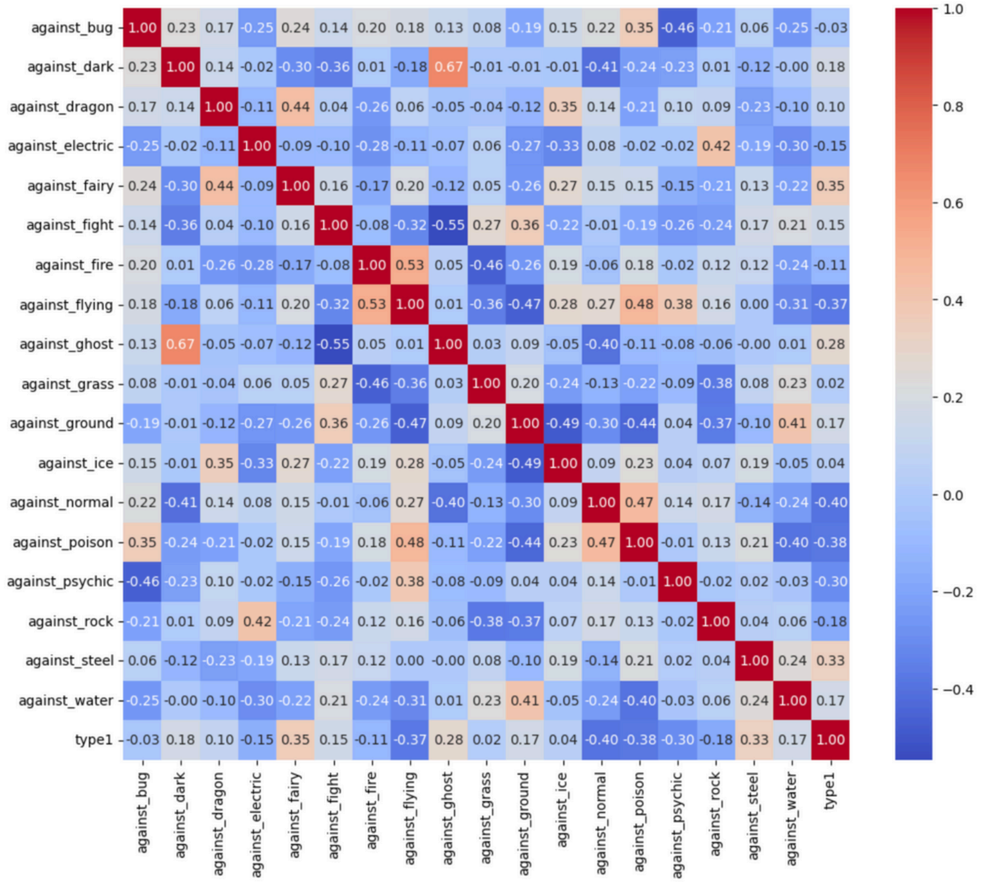
```
average = ((comprobacion_Kmean * 100)/100) / comprobacion_real

print("Average: ",average)
```

```
Average:  0.9933565402029414
```

POKEMON

```
El tipo más comun es: water
Frecuencia: 114 veces
```





https://colab.research.google.com/drive/1l_hJ4pk6z6wzerQWk_H3hY3ok1nhGDET?usp=sharing