PRÁCTICA 2 - ¿CÓMO REALIZAR LA LIMPIEZA Y ANÁLISIS DE DATOS?

AUTOR:

OSCAR JAVIER VÁSQUEZ CASALLAS

DOCENTE:

JOSE MOREIRA SANCHEZ

TIPOLOGÍA Y CICLO DE VIDA DE LOS DATOS

MESTRIA EN CIENCIA DE DATOS

UNIVERSITAT OBERTA DE CATALUNYA

13 DE ENERO DE 2022

Contenido

1. Descripción del Dataset

El dataset seleccionado para el desarrollo de la practica tiene por título **Heart Failure Prediction Dataset** (Conjunto de datos de predicción de insuficiencia cardíaca). Las enfermedades cardiovasculares (ECV) son la principal causa de muerte en todo el mundo y se cobran aproximadamente 17,9 millones de vidas cada año, lo que representa el 31 % de todas las muertes en todo el mundo. Cuatro de cada 5 muertes por ECV se deben a ataques cardíacos y accidentes cerebrovasculares, y un tercio de estas muertes ocurren prematuramente en personas menores de 70 años. La insuficiencia cardíaca es un evento común causado por ECV y este conjunto de datos contiene 11 atributos que pueden usarse para predecir una posible enfermedad cardíaca.

Las personas con enfermedades cardiovasculares o que tienen un alto riesgo cardiovascular (por la presencia de uno o más factores de riesgo como hipertensión, diabetes, hiperlipidemia o enfermedad ya establecida) necesitan una detección y manejo temprano en el que un modelo de aprendizaje automático puede ser de gran ayuda. El siguiente ejercicio pretende a través de métodos de clasificación y basándonos en el conjunto de datos seleccionado, predecir si un paciente es propenso a una insuficiencia cardiaca.

Este conjunto de datos se creó combinando diferentes conjuntos de datos ya disponibles de forma independiente pero que no se habían combinado antes. En este dataset, 5 conjuntos de datos cardíacos se combinan en 11 características comunes. Los cinco conjuntos de datos utilizados para su curación son:

Cleveland: 303 observaciones

Hungría: 294 observaciones

Suiza: 123 observaciones

Long Beach VA: 200 observaciones

Conjunto de datos de Stalog (corazón): 270 observaciones

Total: 1190 observaciones

Duplicado: 272 observaciones

Conjunto de datos final: 918 observaciones

El dataset tiene 918 filas y está compuesto por 12 columnas las cuales se indican a continuación:

- Age: Edad
- Sex: Sexo (M – F)
- ChestPainType: Tipo de dolor en el pecho

- RestingBP: Presión arterial en reposo
- Cholesterol: Colesterol sérico
- FastingBS: Glucemia en ayunas
- RestingECG: Resultados del electrocardiograma en reposo
- MaxHR: Frecuencia cardíaca máxima alcanzada
- ExerciseAngina: Angina inducida por el ejercicio
- Oldpeak: Pico antiguo = ST
- ST_Slope: La pendiente del segmento ST de ejercicio máximo
- HeartDisease: Objetivo

El dataset se puede encontrar en: https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction?select=heart.csv

Para la realización de la practica el código se desarrolló en lenguaje Python, ya que se tiene más experiencia en el manejo de este, y se tomó como base ejemplos encontrados en la plataforma Kaggle.

Para conocer y verificar los datos del dataset se utilizaron una serie de funciones y se generaron algunos gráficos que nos permiten entenderlos de una mejor forma.

A través de la función ***info()*** de la librería ***pandas*** obtenemos la información de los tipos de datos de las columnas del dataset.

[59]:
```
# obtenemos los tipos de datos del dataset
df_heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
 11  HeartDisease    918 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

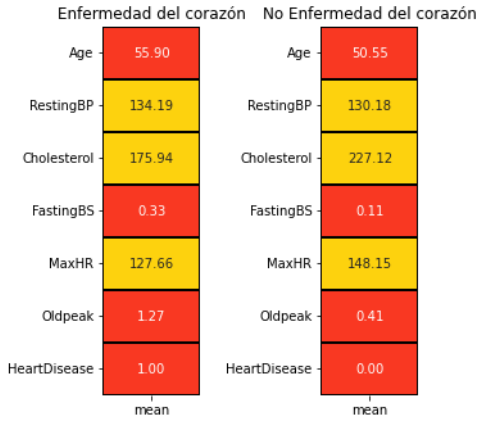
[62]:
```
# Generación de estadísticas descriptivas
df_heart.describe().T
```

[62]:

	count	mean	std	min	25%	50%	75%	max
Age	918.0	53.510893	9.432617	28.0	47.00	54.0	60.0	77.0
RestingBP	918.0	132.396514	18.514154	0.0	120.00	130.0	140.0	200.0
Cholesterol	918.0	198.799564	109.384145	0.0	173.25	223.0	267.0	603.0
FastingBS	918.0	0.233115	0.423046	0.0	0.00	0.0	0.0	1.0
MaxHR	918.0	136.809368	25.460334	60.0	120.00	138.0	156.0	202.0
Oldpeak	918.0	0.887364	1.066570	-2.6	0.00	0.6	1.5	6.2
HeartDisease	918.0	0.553377	0.497414	0.0	0.00	1.0	1.0	1.0

Luego a través de la función ***describe()*** se obtienen las estadísticas descriptivas del dataste incluyendo aquellas que resumen la tendencia central, la dispersión y la forma de la distribución de un conjunto de datos y excluyendo los valores nulos.

```
[63]: # Obtención de la media de columnas del dataset
yes = df_heart[df_heart['HeartDisease'] == 1].describe().T
no = df_heart[df_heart['HeartDisease'] == 0].describe().T
colors = ['#F93822','#FDD20E']

fig,ax = plt.subplots(nrows = 1,ncols = 2,figsize = (5,5))
plt.subplot(1,2,1)
sns.heatmap(yes[['mean']],annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black',cbar = False,fmt = '.2f',)
plt.title('Enfermedad del corazón');

plt.subplot(1,2,2)
sns.heatmap(no[['mean']],annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black',cbar = False,fmt = '.2f')
plt.title('No Enfermedad del corazón');

fig.tight_layout(pad = 2)
```



Con la función anterior se obtuvieron las medias de todas los atributos para los casos de enfermedades del corazón y no enfermedades del corazón a través del valor de la columna **HeartDisease.**

2. Integración y selección

Como fue mencionado anteriormente este conjunto de datos fue generado a través de procesos de integración, combinando cinco diferentes conjuntos de datos ya disponibles de forma independiente; este dataset combinan en 11 características comunes.

En cuanto a la selección, para efectos de la práctica se divide las columnas del conjunto de datos en numéricas y categóricas.

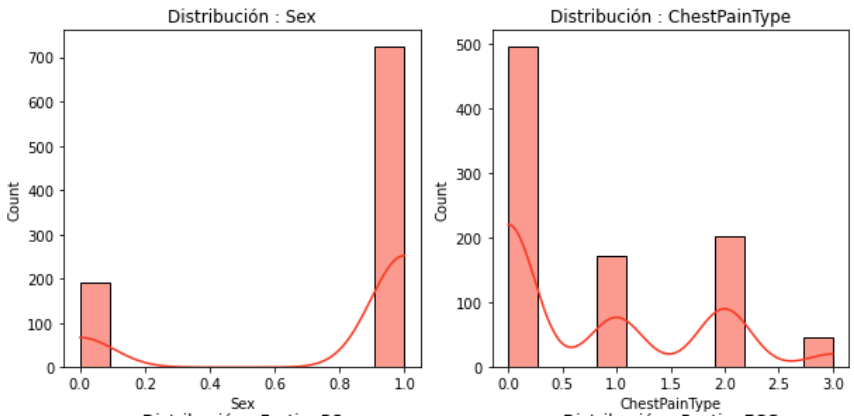
[9]:
```
# División del dataset en columnas categoricas y numéricas
col = list(df_heart.columns)
columnas_categoricas = []
columnas_numericas = []
for i in col:
    if len(df_heart[i].unique()) > 6:
        columnas_numericas.append(i)
    else:
        columnas_categoricas.append(i)

print('Columnas categoricas:',*columnas_categoricas)
print('Columnas numericas:',*columnas_numericas)
```

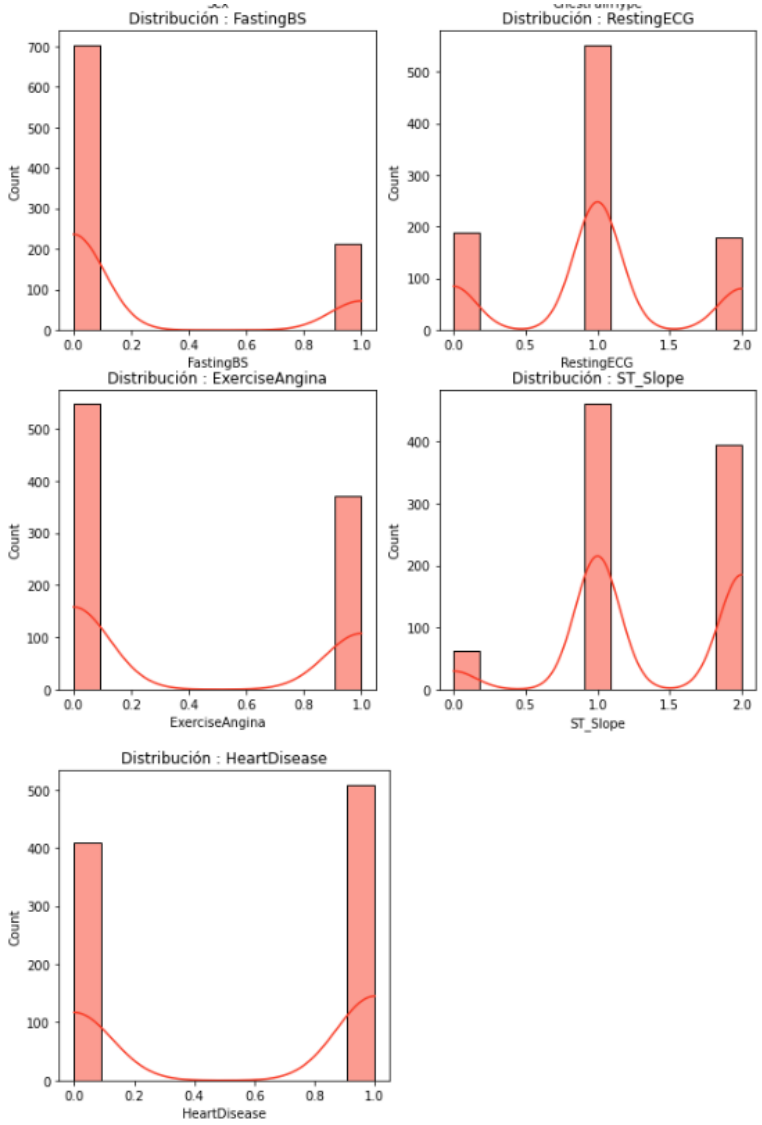
```
Columnas categoricas: Sex ChestPainType FastingBS RestingECG ExerciseAngina ST_Slope HeartDisease
Columnas numericas: Age RestingBP Cholesterol MaxHR Oldpeak
```

Se crea una copia profunda del conjunto de datos original y se codifica la etiqueta de los datos de texto de los atributos categóricos, la cual se utilizará para fines de visualización y modelado ya que se transforman los datos en valores numéricos, esto sin que afecte el dataset original.

[75]:
```
# Transformación de datos categóricos a numéricos
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df_heart1 = df_heart.copy(deep = True)

df_heart1['Sex'] = le.fit_transform(df_heart1['Sex'])
df_heart1['ChestPainType'] = le.fit_transform(df_heart1['ChestPainType'])
df_heart1['RestingECG'] = le.fit_transform(df_heart1['RestingECG'])
df_heart1['ExerciseAngina'] = le.fit_transform(df_heart1['ExerciseAngina'])
df_heart1['ST_Slope'] = le.fit_transform(df_heart1['ST_Slope'])
```

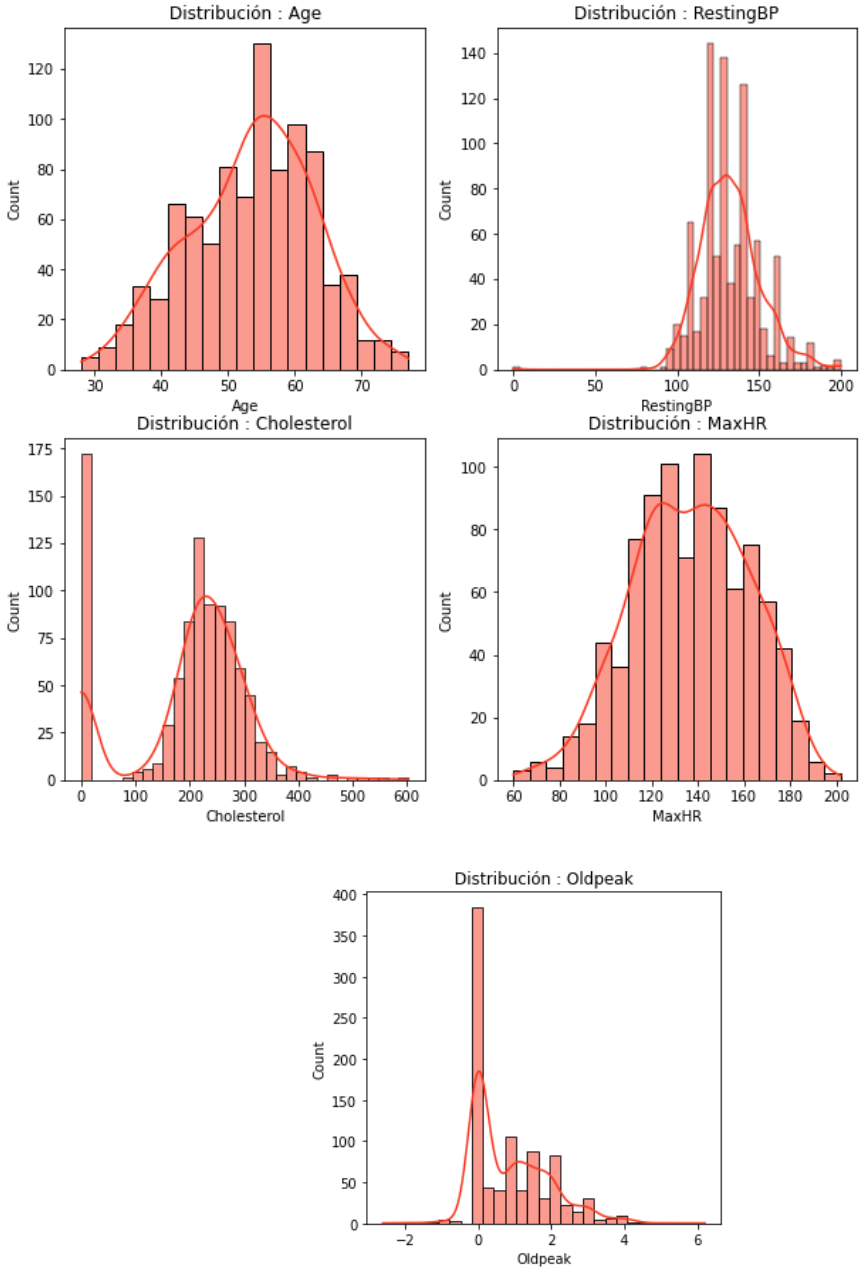
[81]:
```
# Distribución de atributos categóricos
fig, ax = plt.subplots(nrows = 3,ncols = 2,figsize = (10,15))
for i in range(len(columnas_categoricas) - 1):

    plt.subplot(3,2,i+1)
    sns.histplot(df_heart1[columnas_categoricas[i]], kde = True,color = colors[0]);
    title = 'Distribución : ' + columnas_categoricas[i]
    plt.title(title)

plt.figure(figsize = (4.75,4.55))
sns.histplot(df_heart1[columnas_categoricas[len(columnas_categoricas) - 1]], kde = True,color = colors[0])
title = 'Distribución : ' + columnas_categoricas[len(columnas_categoricas) - 1]
plt.title(title);
```





Se realiza la distribución de los atributos categóricos y en los gráficos generados se aprecia que están cerca de una distribución normal.

[83]:
```
# Distribución de atributos numéricos
fig, ax = plt.subplots(nrows = 2,ncols = 2,figsize = (10,9.75))
for i in range(len(columnas_numericas) - 1):
    plt.subplot(2,2,i+1)
    sns.histplot(df_heart[columnas_numericas[i]], kde = True, color = colors[0])
    title = 'Distribución : ' + columnas_numericas[i]
    plt.title(title)
plt.show()

plt.figure(figsize = (4.75,4.55))
sns.histplot(df_heart1[columnas_numericas[len(columnas_numericas) - 1]], kde = True,color = colors[0])
title = 'Distribución : ' + columnas_numericas[len(columnas_numericas) - 1]
plt.title(title);
```



Se realiza la distribución de datos de los atributos numéricos y se aprecia que la de la variable **Oldpeak** está correctamente sesgada, la variable **Cholesterol** tiene una distribución bimodal.

3. Limpieza de los datos.

3.1. ¿Los datos contienen ceros o elementos vacíos? Gestiona cada uno de estos casos.

Es importantes determinar los datos perdidos del conjunto de datos y gestionarlos, ya que para la aplicación de cualquier modelo de análisis nos puede generar problemas la existencia de ellos.

Para la gestión de los datos nulos se ejecuta la siguiente instrucción en Python la cual da como resultado la suma de los valores nulos por las diferentes columnas del dataset, en Python los valores nulos se representan con NaN.

```
[8]: # Comprobación de datos nulos
     df_heart.isnull().sum()
```

```
[8]: Age              0
     Sex              0
     ChestPainType    0
     RestingBP        0
     Cholesterol      0
     FastingBS        0
     RestingECG       0
     MaxHR            0
     ExerciseAngina   0
     Oldpeak          0
     ST_Slope         0
     HeartDisease     0
     dtype: int64
```

Esto nos da como resultado que ninguna de la columnas del dataset tiene valores nulos.

La gestión de datos con valor cero se presentará su tratamiento en el punto 3.2, donde se analicen los valores extremos.

3.2. Identifica y gestiona los valores extremos.

Para la identificación visual de los valores extremos se utilizó un diagrama de caja y bigotes el cual muestra la distribución de datos cuantitativos de una manera que facilita las comparaciones entre variables. El cuadro muestra los cuartiles del conjunto de datos mientras que los bigotes se extienden para mostrar el resto de la distribución; es una forma estandarizada de mostrar la distribución de datos basada en el resumen de cinco números:

Mínimo

Primer cuartil

Mediana

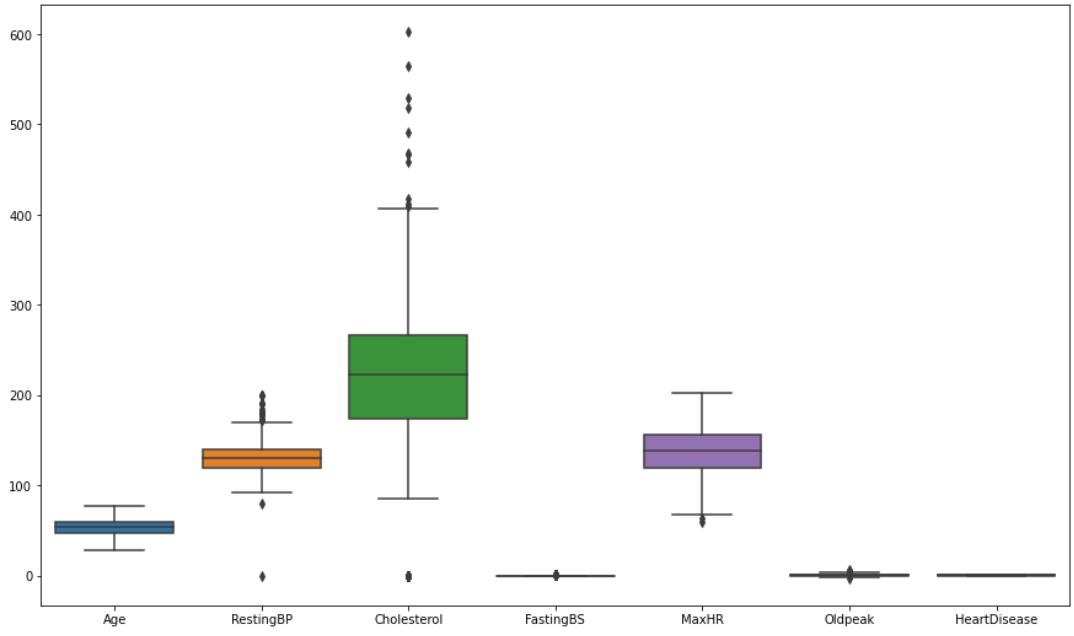
Tercer cuartil

Máximo.

En el diagrama de caja más simple, el rectángulo central abarca desde el primer cuartil hasta el tercer cuartil (el rango intercuartílico o IQR). Un segmento dentro del rectángulo muestra la mediana y los "bigotes" arriba y abajo del cuadro muestran las ubicaciones del mínimo y el máximo.

Este es el diagrama de caja general de las columnas con datos cuantitativos, generado con la siguiente instrucción:

```
•[9]: # Generación de un diagrama de caja general para la detección de valores extremos
      plt.figure(figsize=(15,9))
      sns.boxplot(data=df_heart)
```



Analizando el diagrama de caja, se aprecia que existen valores atípicos en las columnas **Age**, **RestingBP, Cholesterol, MaxHR y Oldpeak**, ante esto se crea una lista y se realiza una función para detectar los outliers:

[11]:
```
# Función para detectar los outliers

df_num_nombre = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']

def find_outliers_IQR(df_heart):
   q1=df_heart.quantile(0.25)
   q3=df_heart.quantile(0.75)
   IQR=q3-q1
   outliers = df_heart[((df_heart<(q1-1.5*IQR)) | (df_heart>(q3+1.5*IQR)))]
   return outliers
```

[13]:
```
# Obtener e imprimir valores atipicos por columna

for char in df_num_nombre :
    outliers = find_outliers_IQR(df_heart[str(char)])
    print(f"-Obtener {char} outliers ")
    print('Número de outliers: '+ str(len(outliers)))
    print('Valor máximo de outlier: '+ str(outliers.max()))
    print('Valor mínimo de outlier: '+ str(outliers.min())+ '\n')
```

```
-Obtener Age outliers
Número de outliers: 0
Valor máximo de outlier: nan
Valor mínimo de outlier: nan

-Obtener RestingBP outliers
Número de outliers: 28
Valor máximo de outlier: 200
Valor mínimo de outlier: 0

-Obtener Cholesterol outliers
Número de outliers: 183
Valor máximo de outlier: 603
Valor mínimo de outlier: 0

-Obtener MaxHR outliers
Número de outliers: 2
Valor máximo de outlier: 63
Valor mínimo de outlier: 60

-Obtener Oldpeak outliers
Número de outliers: 16
Valor máximo de outlier: 6.2
Valor mínimo de outlier: -2.6
```

De esta forma encontramos el número de valores atípicos y el máximo y mínimo de ellos para poder detectarlos claramente.

[18]:
```
# Imprimir valores de 0 y no 0 en la columna Cholesterol

print(f" Forma original de la columna Cholesterol: {df_heart.Cholesterol.shape}")
cero_Cholesterol = df_heart[df_heart['Cholesterol'] == 0]
print(f" Valores de 0 en la columna Cholesterol : {cero_Cholesterol.shape}")
no_cero_Cholesterol = df_heart[df_heart['Cholesterol'] != 0]
print(f" Valores diferentes de 0 en la columna Cholesterol : {no_cero_Cholesterol.shape}")
```

```
 Forma original de la columna Cholesterol: (918,)
 Valores de 0 en la columna Cholesterol : (172, 12)
 Valores diferentes de 0 en la columna Cholesterol : (746, 12)
```

[19]:
```
# Reemplazar valores 0 con dato vacio
df_heart.loc[df_heart['Cholesterol'] == 0,'Cholesterol'] = np.nan
```

[20]:
```
# Reemplazar datos vacios con la media de la columna
df_heart["Cholesterol"] = df_heart["Cholesterol"].fillna(df_heart["Cholesterol"].median())
```

La columna **Cholesterol** es la que más datos atípicos tienen, adicionalmente tiene datos con el valor 0, el cual por el tipo de columna y la característica de esta no los debería tener, por eso debemos

gestionarlos. Lo primero que hice es filtrar e imprimir en pantalla el número de filas que tienen 0 en la columna **Cholesterol** y compararlas con el valor total de datos del conjunto. Luego busco y reemplazo esos valores de 0 con np.nan; para luego reemplazar estos valores vacíos con la media de esta columna.

```
[29]: # Filtrar valores con 0 de la columna RestingBP
      cero_RestingBP = df_heart[df_heart['RestingBP'] == 0]
      cero_RestingBP
```

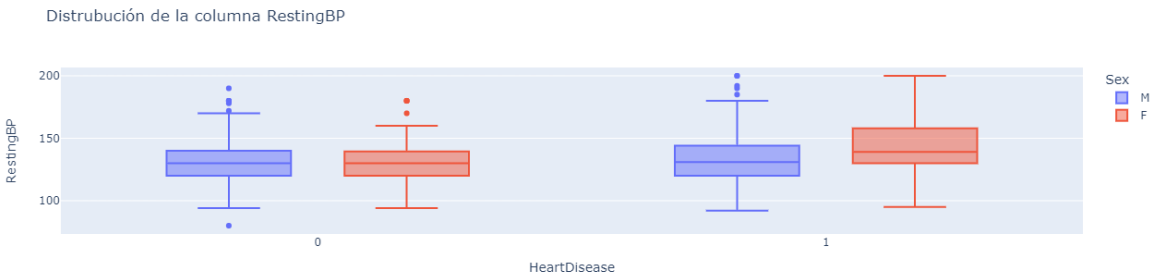
[29]:

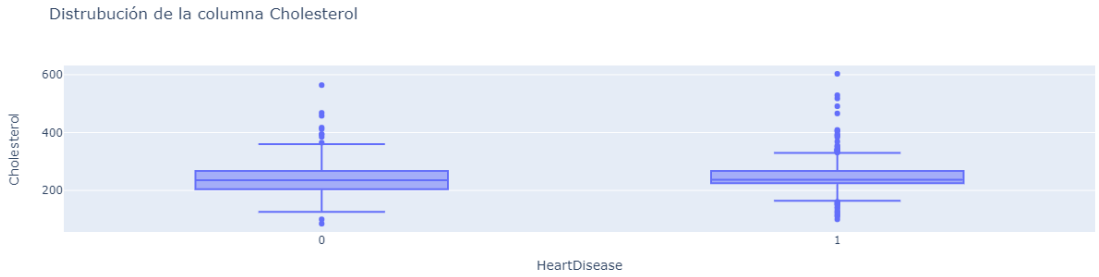
	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
449	55	M	NAP	0	237.0	0	Normal	155	N	1.5	Flat	1

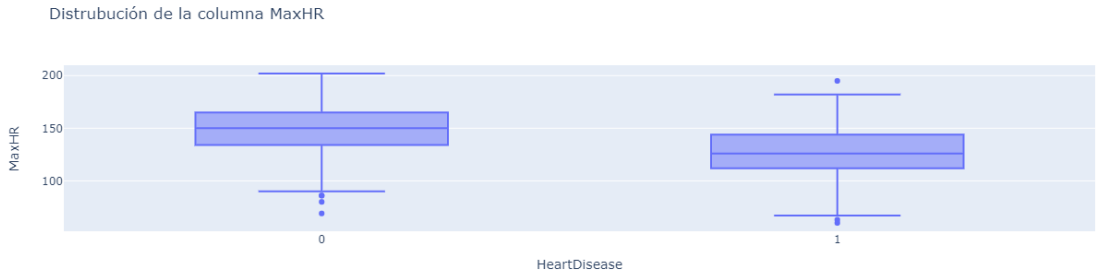
```
[30]: # Eliminar fila con valor 0 de la columna RestingBP del conjunto de datos
      df_heart = df_heart.drop(df_heart[(df_heart['RestingBP'] == 0)].index)
```

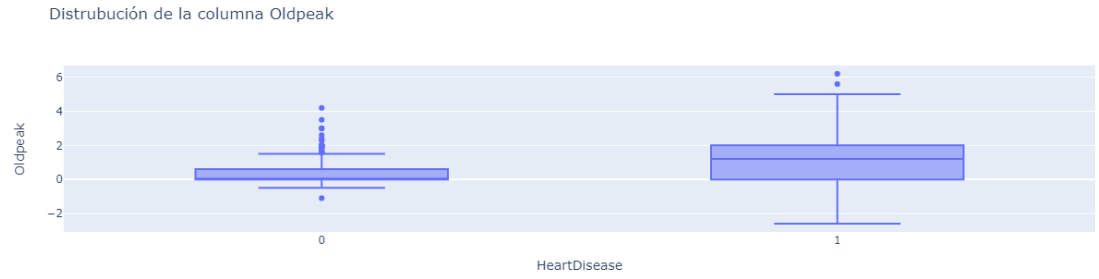
Para la columna **RestingBP** se realiza el filtro para determinar las filas que tengan el valor de 0, para efectos de la práctica esta fila fue eliminada.

Por otra parte, se generaron diagramas de caja independientes por cada columna para determinar la distribución respecto a la variable objetivo **HeartDisease**.









4. Análisis de los datos.

4.1. Selección de los grupos de datos que se quieren analizar/comparar (p. ej., si se van a comparar grupos de datos, ¿cuáles son estos grupos y qué tipo de análisis se van a aplicar?)

Es necesario realizar procesos de normalización y estandarización de los datos, ya que los modelos de aprendizaje automático así lo requieren. Como la mayoría de los algoritmos asumen que los datos tienen una distribución normal (gaussiana), la normalización se realiza para las características cuyos datos no muestran una distribución normal y la estandarización se lleva a cabo para las atributos que se distribuyen normalmente donde sus valores son grandes o muy pequeños en comparación con otros atributos.

```
[55]: # Normalización y estandarización de atributos
from sklearn.preprocessing import MinMaxScaler,StandardScaler
mms = MinMaxScaler() # Normalization
ss = StandardScaler() # Standardization

df_heart1['Oldpeak'] = mms.fit_transform(df_heart1[['Oldpeak']])
df_heart1['Age'] = ss.fit_transform(df_heart1[['Age']])
df_heart1['RestingBP'] = ss.fit_transform(df_heart1[['RestingBP']])
df_heart1['Cholesterol'] = ss.fit_transform(df_heart1[['Cholesterol']])
df_heart1['MaxHR'] = ss.fit_transform(df_heart1[['MaxHR']])
df_heart1.head()
```

[55]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	-1.433140	1	1	0.410909	0.825070	0	1	1.382928	0	0.295455	2	0
1	-0.478484	0	2	1.491752	-0.171961	0	1	0.754157	0	0.409091	1	1
2	-1.751359	1	1	-0.129513	0.770188	0	2	-1.525138	0	0.295455	2	0
3	-0.584556	0	0	0.302825	0.139040	0	1	-1.132156	1	0.465909	1	1
4	0.051881	1	2	0.951331	-0.034755	0	1	-0.581981	0	0.295455	2	0

El atributo **Oldpeak** se normaliza con la función ***MinMaxScaler()***, ya que había mostrado una distribución de datos sesgada hacia la derecha. MinMaxSchaler transforma los datos escalando cada valor a un rango determinado entre cero y uno.

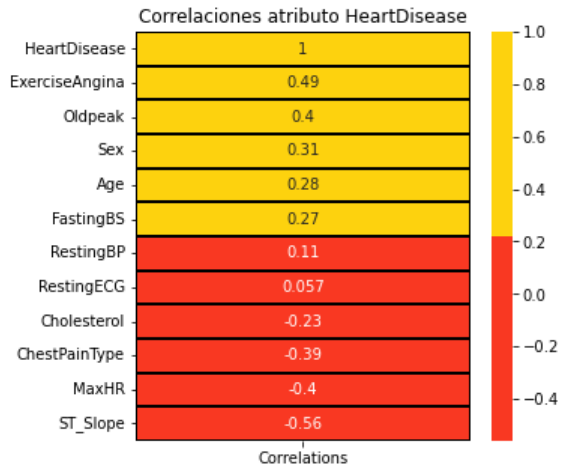
Los atributos **Age**, **RestingBP**, **Cholesterol** y **MaxHR**, se estandarizan ya que se distribuyen normalmente, se utiliza la función ***StandardScaler()***, la cual estandariza los valores eliminando la media y escalando a la varianza de la unidad.

Se genera la matriz de correlación la cual nos permite representar gráficamente las correlaciones entre pares de variables en un conjunto de datos.

```
•[33]: # Generación de la matriz de correlación
plt.figure(figsize = (20,5))
sns.heatmap(df_heart1.corr(),cmap = colors,annot = True);
```

Se aprecia que se genera una matriz con demasiadas características, ante esto nos centraremos en verificar únicamente la correlación con respecto a la columna **HeartDisease**.

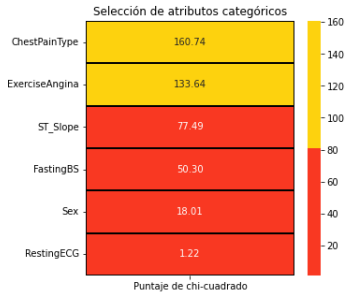
```
[58]: # Generación de las correlaciones con el atributo HeartDisease
corr = df_heart1.corrwith(df_heart1['HeartDisease']).sort_values(ascending = False).to_frame()
corr.columns = ['Correlations']
plt.subplots(figsize = (5,5))
sns.heatmap(corr,annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black');
plt.title('Correlaciones atributo HeartDisease');
```



Analizando los resultados generados apreciamos que a excepción de los atributos **RestingBP** y **RestingECG**, todos los demás muestran una relación positiva o negativa con **HeartDisease.**

Para la selección de las propiedades categóricas se realiza la prueba de chi-cuadrado.

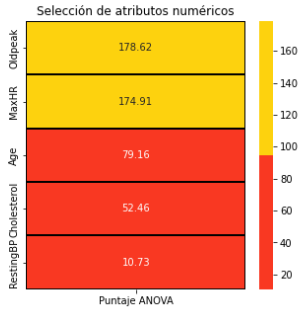
```
*[59]: # Generación de la prueba de chi-cuadrado para la selección de propiedades categóricas
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
features = df_heart1.loc[:,columnas_categoricas[:-1]]
target = df_heart1.loc[:,columnas_categoricas[-1]]

best_features = SelectKBest(score_func = chi2,k = 'all')
fit = best_features.fit(features,target)

featureScores = pd.DataFrame(data = fit.scores_,index = list(features.columns),columns = ['Puntaje de chi-cuadrado'])

plt.subplots(figsize = (5,5))
sns.heatmap(featureScores.sort_values(ascending = False,by = 'Puntaje de chi-cuadrado'),annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black',fmt = '.2f');
plt.title('Selección de atributos categóricos');
```



Según los resultados de la prueba se aprecia que a excepción de la propiedad **RestingECG**, todas las demás propiedades categóricas son bastante importantes para predecir enfermedades del corazón.

Para la selección de las propiedades numéricas se realiza la prueba ANOVA.

```
[61]: # Generación de la prueba de ANOVA para la selección de propiedades numéricas
from sklearn.feature_selection import f_classif

features = df_heart1.loc[:,columnas_numericas]
target = df_heart1.loc[:,columnas_categoricas[-1]]

best_features = SelectKBest(score_func = f_classif,k = 'all')
fit = best_features.fit(features,target)

featureScores = pd.DataFrame(data = fit.scores_,index = list(features.columns),columns = ['Puntaje ANOVA'])

plt.subplots(figsize = (5,5))
sns.heatmap(featureScores.sort_values(ascending = False,by = 'Puntaje ANOVA'),annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black',fmt = '.2f');
plt.title('Selección de atributos numéricos');
```



Según los resultados de la prueba se aprecia que a excepción de la propiedad **RestingBP**, todas las demás propiedades numéricas son importantes para predecir enfermedades del corazón.

4.2. Comprobación de la normalidad y homogeneidad de la varianza.

```
[ ]: from sklearn.preprocessing import StandardScaler
```

```
[63]: from scipy.stats import shapiro

num_cols=df_heart1.select_dtypes(include=['int64','float64']).columns.tolist()
for feature in num_cols:
    data = df_heart1.copy()
    DataToTest = data[feature]
    stat, p = shapiro(DataToTest)
    print(feature)
    print('stat = %.2f, p = %.30f' % (stat, p))

    if p > 0.05:
        print('Distribución Normal')
        print()
    else:
        print('No Distribución Normal')
        print()
```

```
Age
stat = 0.99, p = 0.000021732857931056059896945953
No Distribución Normal

RestingBP
stat = 0.96, p = 0.000000000000001493837999568643
No Distribución Normal

Cholesterol
stat = 0.87, p = 0.000000000000000000000000006975
No Distribución Normal

FastingBS
stat = 0.52, p = 0.000000000000000000000000000000
No Distribución Normal

MaxHR
stat = 0.99, p = 0.000168283208040520548820495605
No Distribución Normal

Oldpeak
stat = 0.86, p = 0.000000000000000000000000000826
No Distribución Normal

HeartDisease
stat = 0.63, p = 0.000000000000000000000000000000
No Distribución Normal
```

Se aplica el test de Shapiro y se comprueba que ninguna de las propiedades numéricas sigue una distribución normal.

4.3. Aplicación de pruebas estadísticas para comparar los grupos de datos. En función de los datos y el objetivo del estudio, aplicar pruebas de contraste de hipótesis, correlaciones, regresiones, etc. Aplicar al menos tres métodos de análisis diferentes.

Se seleccionan las propiedades de las pruebas realizadas anteriormente y se dividen los datos en 80 para entrenamiento y 20 para prueba.

[123]:
```
# Selección de los grupos de prueba
features = df_heart1[df_heart1.columns.drop(['HeartDisease','RestingBP','RestingECG'])].values
target = df_heart1['HeartDisease'].values
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size = 0.20, random_state = 2)
```

[124]:
```
#Función para graficar la curva roc del modelo
def model(classifier):

    classifier.fit(x_train,y_train)
    prediction = classifier.predict(x_test)
    cv = RepeatedStratifiedKFold(n_splits = 10,n_repeats = 3,random_state = 1)
    print("Precisión : ",'{0:.2%}'.format(accuracy_score(y_test,prediction)))
    print("Puntaje de validación cruzada : ",'{0:.2%}'.format(cross_val_score(classifier,x_train,y_train,cv = cv,scoring = 'roc_auc').mean()))
    print("Puntaje - Área bajo la curva: ",'{0:.2%}'.format(roc_auc_score(y_test,prediction)))
    metrics.RocCurveDisplay.from_estimator(classifier, x_test,y_test)
    plt.title('Curvas ROC')
    plt.show()

def model_evaluation(classifier):

    # Generación de la amtriz de confusión
    cm = confusion_matrix(y_test,classifier.predict(x_test))
    names = ['True Neg','False Pos','False Neg','True Pos']
    counts = [value for value in cm.flatten()]
    percentages = ['{0:.2%}'.format(value) for value in cm.flatten()/np.sum(cm)]
    labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(names,counts,percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(cm,annot = labels,cmap = colors,fmt ='')

    # Imprimir reporte de clasificación
    print(classification_report(y_test,classifier.predict(x_test)))
```

Se crea la función para graficar las curvas ROC de los modelos de clasificación que se van a comparar y en el cual se presenta el área bajo la curva que nos permite comprobar la calidad del modelo con base en la precisión de este.

Adicionalmente se genera y visualiza la matriz de confusión la cual es una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real.

A continuación, se aplican los tres modelos de clasificación:

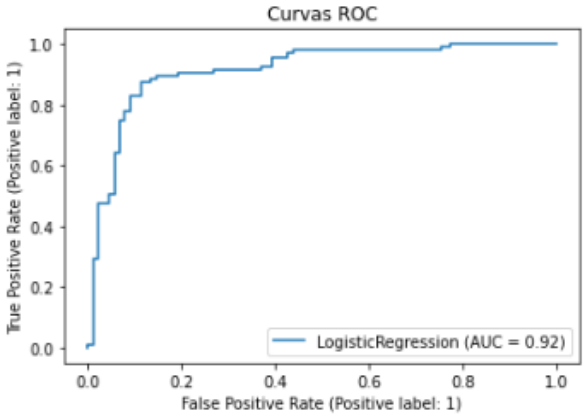
- **Regresión logística**

```
[120]: from sklearn.linear_model import LogisticRegression
```

```
[125]: # Aplicación del modelo de regresión logística
classifier_lr = LogisticRegression(random_state = 0,C=10,penalty= 'l2')
```

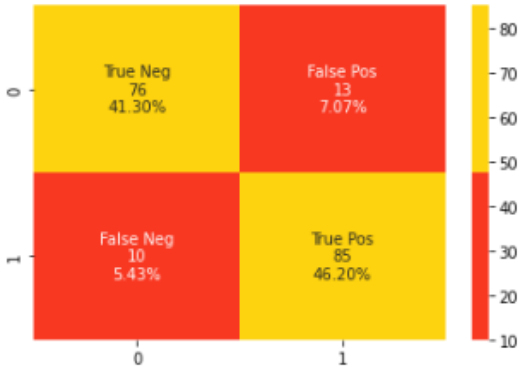
```
[122]: model(classifier_lr)
```

```
Accuracy :  87.50%
Cross Validation Score :  91.12%
Puntaje - Área bajo la curva:  87.43%
```




```
[108]: model_evaluation(classifier_lr)
```

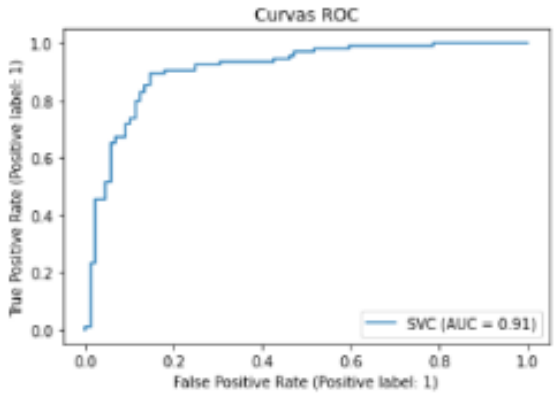
	precision	recall	f1-score	support
0	0.88	0.85	0.87	89
1	0.87	0.89	0.88	95
accuracy			0.88	184
macro avg	0.88	0.87	0.87	184
weighted avg	0.88	0.88	0.87	184



- **Vectores de soporte**

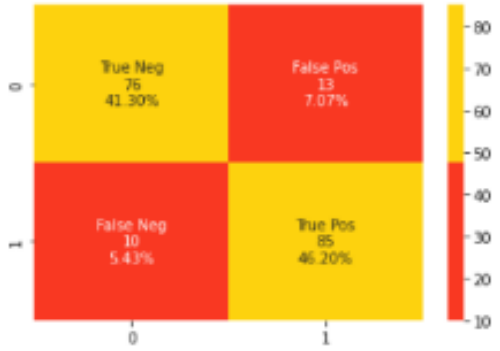
```
[127]: from sklearn.svm import SVC

[128]: # Aplicación del modelo de SVC
       classifier_svc = SVC(kernel = 'linear',C = 0.1)

[129]: model(classifier_svc)
```

```
Precisión :  87.50%
Puntaje de validación cruzada :  90.53%
Puntaje - Área bajo la curva:  87.43%
```




```
[112]: model_evaluation(classifier_svc)
```

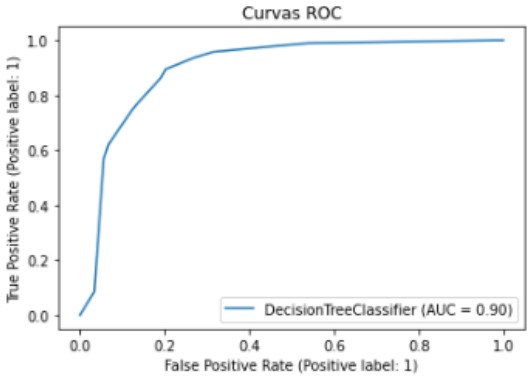
	precision	recall	f1-score	support
0	0.88	0.85	0.87	89
1	0.87	0.89	0.88	95
accuracy			0.88	184
macro avg	0.88	0.87	0.87	184
weighted avg	0.88	0.88	0.87	184



- **Arboles de decisión**

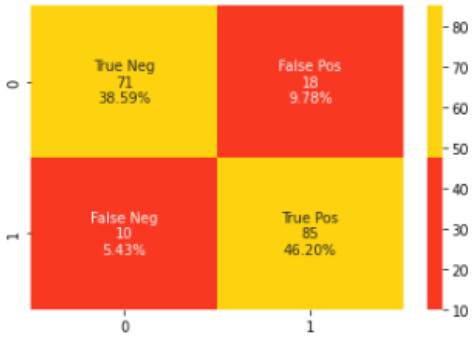
```
[130]: from sklearn.tree import DecisionTreeClassifier

[131]: # Aplicación del modelo de arboles de decisión
classifier_dt = DecisionTreeClassifier(random_state = 1000,max_depth = 4,min_samples_leaf = 1)

[132]: model(classifier_dt)
```

```
Precisión :  84.78%
Puntaje de validación cruzada :  89.09%
Puntaje - Área bajo la curva:  84.62%
```




```
[116]: model_evaluation(classifier_dt)
```

	precision	recall	f1-score	support
0	0.88	0.80	0.84	89
1	0.83	0.89	0.86	95
accuracy			0.85	184
macro avg	0.85	0.85	0.85	184
weighted avg	0.85	0.85	0.85	184



A continuación, se presenta la tabla de resultados de la aplicación de los algoritmos:

Algoritmo	Precisión	Porcentaje de validación cruzada	Porcentaje de área bajo la curva
Regresión logística	87.50%	91.12%	87.43%
Vectores de soporte	87.50%	90.53%	87.43%
Árboles de decisión	84.78%	89.09%	84.62%

5. Representación de los resultados

La representación gráfica de los resultados se presenta a lo largo de la práctica con la generación y visualización de las gráficos que apoyan el entendimiento del desarrollo.

6. Resolución del problema

A continuación, se presentan las conclusiones de la práctica realizada:

- Este conjunto de datos es excelente para comprender cómo manejar los problemas de clasificación binaria con la combinación de características numéricas y categóricas, de esta forma se puede resolver el problema propuesto ya que podemos predecir a través de algoritmos de clasificación si un paciente es propenso a una enfermedad de insuficiencia cardiaca. Se analizan tres métodos de clasificación y se obtiene que el que mejor resultados presenta es el de regresión logística.

- Para esta práctica se realizó la detección y gestión de valores atípicos, sin embargo, para la aplicación de los algoritmos de clasificación se decidió realizar con el dataset original, ya que su eliminación implica tener un cierto nivel de experticia en el tema médico.

- La visualización de los diferentes procesos que se realiza al conjunto de datos es muy importante ya que se entiende de una mejor manera su comportamiento y hace que los datos sean mucho más fáciles de comunicar. Mostrar la información actual y los resultados de cualquier prueba o salida a través de la visualización se vuelve crucial ya que facilita la comprensión.

7. Código

El código con el que se desarrolló la practica se construyó en lenguaje Python y se encuentra en la carpeta source del repositorio de GitHub bajo el nombre PracticaAnalisisDatos.ipynb (https://github.com/oscajvasquez/practica2_analisis_datos)

8. Vídeo

Se realiza y sube el vídeo en el Google Drive de la Universidad en el siguiente enlace
https://drive.google.com/file/d/1KLCghBcoXQ_UrqYmUiy7lXV7MUvaRS3M/view

CONTRIBUCIONES	FIRMA
Investigación previa	OJVC
Redacción de las respuestas	OJVC
Desarrollo del código Integrante	OJVC
Participación en el vídeo	OJVC