Actividad 2.1 – Análisis de observaciones influyentes

1.

A)

Ejercicio: Calcular la media y la mediana antes de realizar la modificación de incluir unos ingresos de 500.000€

```python
media=  np.mean(datos)
mediana=  np.median (datos) # Es el valor central cuando los datos se ordenan
print(f" Media= {media} y Mediana={mediana}")
```

```
Media= 20364.58936071857 y Mediana=21111.15580445307
```

B)

Ejercicio: Aplicar el método de Probabilidad global, para detectar los outliers utilizado en el ejemplo 2_3_Outliers

```python
import scipy.stats as st

# CRITERIO 1: PROBABILIDAD GLOBAL - Explicado en el capítulo 5 - Preprocesamiento (Semana 2), en la pág 10
# Asumiendo que las variables tiene una distribución normal.
# Probabilidad de la muestra de estar dentro de las bandas
p_g=0.95
# probabilidad global
alfa_g=(1-p_g)/2
# probabilidad  para un solo dato
alfa= 1-(1-alfa_g)**(1/len(datos)) # Se realiza este ajuste para ser más precisos.
#alfa = alfa_g # Si no realiza la corrección de la línea anterior entonces hubiese detectado más valores que son outliers cuando re
              # con esta corrección lo que se pretende es obtener bandas lo sificientemente anchas.
#alfa=1/(2*len(datos))
Z_alfa=st.norm.ppf(1-alfa/2)

# Impresión de resultados
alfa=round(alfa,5)
Z_alfa=round(Z_alfa,5)
print(f" Alfa ={alfa}")
print(f" Z_alfa ={Z_alfa}")
```

```
Alfa =0.00025
Z_alfa =3.65906
```

```python
# Utilizaremos el Criterio 1 ó el Criuterio 2 para calcular alfa y Z-alfa en función de la naturaleza de los datos que estamos proc
xL=  round(np.mean(datos)-Z_alfa*  np.std(datos),4)
xU=  round(np.mean(datos)+Z_alfa*  np.std(datos),4)
print(f" Banda= [ {xL},{xU}]")
```

```
Banda= [ -149826.1041,200131.9466]
```

```python
for i in  range(len(datos)):
        if datos[i] < xL  or datos[i]>xU:
          print(f" El dato[{i}]={datos[i]} es un outlier")
```

Python

```
El dato[50]=500000.0 es un outlier
```

C)

Ejercicio: Repetir el mismo procedimiento (método Jackenife) de detectar los outliers pero con la mediana en lugar de con la media: ¿Qué ocurre? ¿observas variaciones?

```python
phi=np.zeros(len(datos)) # phi es un array de la misma longitud que datos pero con el valor cero en cada una de sus posiciones
# En el bucle lo que se realiza es el método Jackknife, es decir, prepara los datos para realizar un análisis estadístico sin el da
# Para ello, calcula la media de todos los elementos excluyendo el elemento de la posición i que está procesando en el bucle.
# De forma que obtiene un índice de calidad del modelo para cada uno de los elementos
for i in  range(len(datos)):
    datos_aux=datos
    datos_sin_i=np.delete(datos_aux, i)
    phi[i]=np.median(datos_sin_i)
phi
```

Python

```
array([21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21207.47121438, 21207.47121438,
       21207.47121438, 21207.47121438, 21055.49865657, 21055.49865657,
       21055.49865657, 21055.49865657, 21055.49865657, 21055.49865657,
       21055.49865657, 21055.49865657, 21055.49865657, 21055.49865657,
       21055.49865657, 21055.49865657, 21055.49865657, 21055.49865657,
       21055.49865657, 21055.49865657, 21055.49865657, 21055.49865657,
       21055.49865657, 21055.49865657, 21055.49865657, 21055.49865657,
       21055.49865657, 21055.49865657, 21055.49865657, 21055.49865657,
```

```python
# Aplicamos el método de la distancia entre cuartiles, en este caso, al aplicar el método Jackknife no obtenemos
# el valor del elemento que es considerado Outlier, sino la posición del mismo.
Q1 = np.quantile(phi,0.25)
Q3 = np.quantile(phi,0.75)
IQR = Q3 - Q1
xL=Q1 - 1.5 * IQR
xU=Q3 + 1.5 * IQR
for i in  range(len(datos)):
        if phi[i] < xL  or phi[i]>xU:
          print(f" El dato {i} es una observación influyente para la media")
```

```
1  pd.DataFrame(phi).describe()
```

	0
count	100.000000
mean	21131.484935
std	76.369084
min	21055.498657
25%	21055.498657
50%	21131.484935
75%	21207.471214
max	21207.471214

En este caso no encuentra el dato influyente.

2.

A)

1. ¿Cuánto vale la media, mediana, la desviación estándar muestral, la varianza muestral y el rango de la variable X?

```python
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Media
media = np.mean(X)

# Mediana
mediana = np.median(X)

# Desviación estándar muestral
desviacion_estandar = np.std(X)

# Varianza muestral
varianza = np.var(X)

# Rango
rango = np.max(X) - np.min(X)

print("Media:", media)
print("Mediana:", mediana)
print("Desviación estándar muestral:", desviacion_estandar)
print("Varianza muestral:", varianza)
print("Rango:", rango)
```

```
Media: 5.5
Mediana: 5.5
Desviación estándar muestral: 2.8722813232690143
Varianza muestral: 8.25
Rango: 9
```

B)

2. Utilizar la función describe() de Panda, para obtener la media, desviación estándar, etc...

```
X = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

print(X.describe());
```

```
count    10.00000
mean      5.50000
std       3.02765
min       1.00000
25%       3.25000
50%       5.50000
75%       7.75000
max      10.00000
dtype: float64
```

C)

3. ¿Por qué el resultado de calcular la desviación estándar con Numpy es diferente a la calculada por describe de Panda? ¿Qué ajuste sería necesario realizar para que los resultados fuesen similares/iguales?

Por defecto, en NumPy, ddof es 0, lo que significa que se está calculando la desviación estándar y la varianza de la población. Sin embargo, en estadísticas, cuando trabajamos con muestras en lugar de poblaciones completas, a menudo queremos ajustar los cálculos para tener en cuenta los grados de libertad perdidos al estimar parámetros poblacionales a partir de la muestra. En este caso, se utiliza ddof=1.

Para que los datos fuesen iguales o parecidos habria que añadir el parametro ddof al metodo de NumPy:

```
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Desviación estándar muestral
desviacion_estandar = np.std(X, ddof=1)

print("Desviación estándar muestral:", desviacion_estandar)
```

Python

```
Desviación estándar muestral: 3.0276503540974917
```

D)

4. Estandarizar la variable (escalamiento) mediante rangos y a continuación calcular la media y la mediana de la variable escalada

$$X_{\text{escalado}} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

```python
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Rango
rango = np.max(X) - np.min(X)

# Estandarizar mediante rangos
X_escalado = (X - np.min(X)) / rango

# Calcular la media y la mediana de la variable escalada
media_escalada = np.mean(X_escalado)
mediana_escalada = np.median(X_escalado)

print("Media de la variable escalada:", media_escalada)
print("Mediana de la variable escalada:", mediana_escalada)
```

Python

```
Media de la variable escalada: 0.5
Mediana de la variable escalada: 0.5
```

E)

5. Repetir el apartado anterior con el escalamiento Z - score

$$Z = \frac{X - \text{media}(X)}{\text{desviación estándar}(X)}$$

```python
X = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Estandarizar mediante Z-score
X_zscore = (X - X.mean()) / X.std()

# Calcular la media y la mediana de la variable estandarizada
media_zscore = np.mean(X_zscore)
mediana_zscore = np.median(X_zscore)

print("Media de la variable estandarizada (Z-score):", media_zscore)
print("Mediana de la variable estandarizada (Z-score):", mediana_zscore)
```

Python

```
Media de la variable estandarizada (Z-score): 4.440892098500626e-17
Mediana de la variable estandarizada (Z-score): 0.0
```

GITHUB