4.0.outliers

December 9, 2024

https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/

1 Outliers

Les outliers, qui s'écartent significativement de la norme, peuvent fausser les mesures de tendance centrale et influencer les analyses statistiques. Cet article explore les causes courantes des outliers, qu'il s'agisse d'erreurs ou d'introductions intentionnelles, et met en lumière leur importance dans l'identification des anomalies lors de l'analyse des données.

1.1 Qu'est-ce qu'un outlier ?

Un outlier est un élément ou un objet de données qui s'écarte significativement des autres objets (considérés comme normaux). Identifier les outliers est crucial en statistique et en analyse de données, car ils peuvent avoir un impact considérable sur les résultats des analyses statistiques. L'analyse visant à détecter les outliers est appelée outlier mining (extraction des outliers).

Les outliers peuvent biaiser la moyenne (ou moyenne arithmétique) et affecter les mesures de tendance centrale, tout en influençant les résultats des tests de significativité statistique.

1.2 Comment les outliers sont-ils causées ?

Les outliers peuvent être causés par une variété de facteurs, résultant souvent d'une variabilité inhérente aux données ou d'erreurs dans la collecte, la mesure ou l'enregistrement des données. Voici quelques causes courantes des outliers :

- **Erreurs de mesure :** Des erreurs dans les processus de collecte ou de mesure des données peuvent entraîner des outliers.
- **Erreurs d'échantillonnage :** Dans certains cas, des problèmes liés au processus d'échantillonnage peuvent engendrer des outliers.
- **Variabilité naturelle :** Une variabilité inhérente à certains phénomènes peut également donner lieu à des valeurs extrêmes. Certains systèmes peuvent présenter des valeurs extrêmes en raison de la nature du processus étudié.
- **Erreurs de saisie des données :** Les erreurs humaines lors de la saisie des données peuvent introduire des outliers.
- **Erreurs expérimentales :** Dans les contextes expérimentaux, des anomalies peuvent survenir en raison de facteurs incontrôlés, de dysfonctionnements d'équipement ou d'événements inattendus.
- **Échantillonnage de plusieurs populations :** Les données provenant de populations multiples aux caractéristiques différentes peuvent être combinées par inadvertance.

- **Outliers intentionnels :** Des outliers peuvent être introduits intentionnellement pour tester la robustesse des méthodes statistiques.

[3]:
```
# Importing
import sklearn
from sklearn.datasets import load_diabetes
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
diabetics = load_diabetes()

# Create the dataframe
column_name = diabetics.feature_names
df_diabetics = pd.DataFrame(diabetics.data)
df_diabetics.columns = column_name
df_diabetics.head()
```

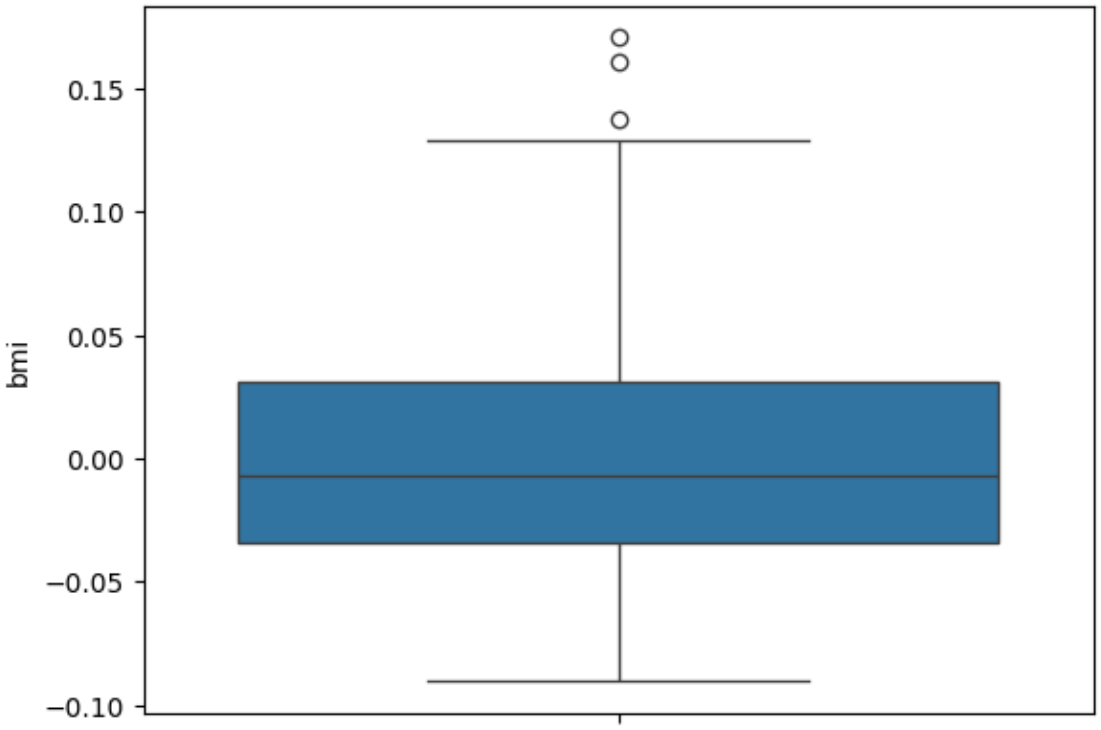
[3]:

	age	sex	bmi	bp	s1	s2	s3	\
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	

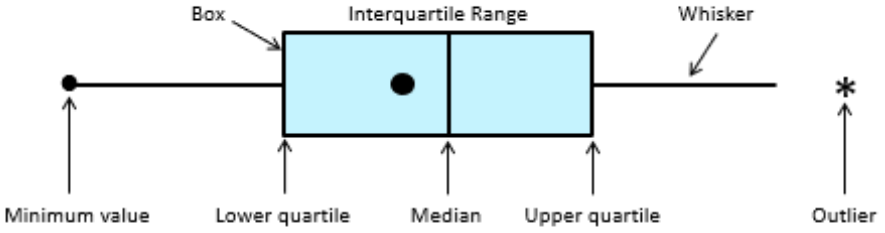
	s4	s5	s6
0	-0.002592	0.019907	-0.017646
1	-0.039493	-0.068332	-0.092204
2	-0.002592	0.002861	-0.025930
3	0.034309	0.022688	-0.009362
4	-0.002592	-0.031988	-0.046641

[4]:
```
# Box Plot
import seaborn as sns
sns.boxplot(df_diabetics['bmi'])
```

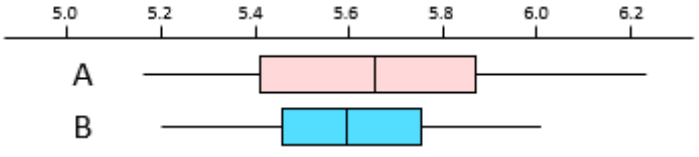
[4]:
```
<Axes: ylabel='bmi'>
```



- A box plot is made up of a box and two whiskers.
- The data is plotted in such away that the bottom 25% and the top 25% of the data points are represented by the two whiskers, whereas the middle 50% of the data points are represented by the box.



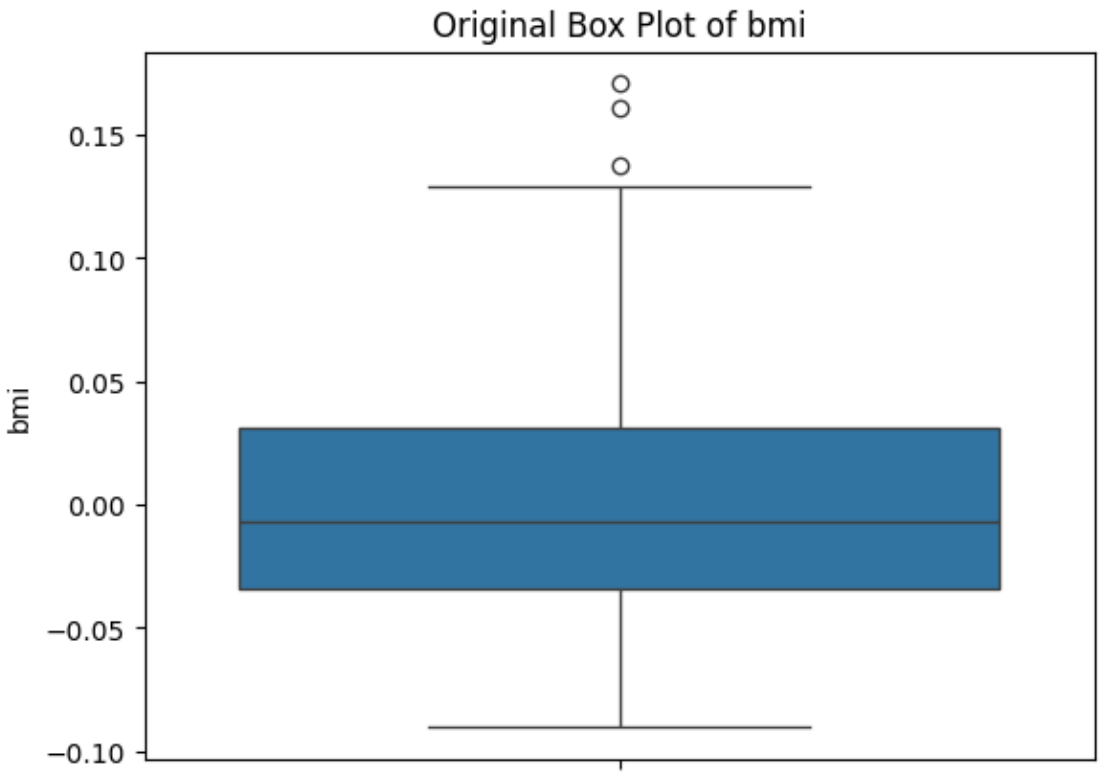
Example of How to Compare Between Multiple Data Sets

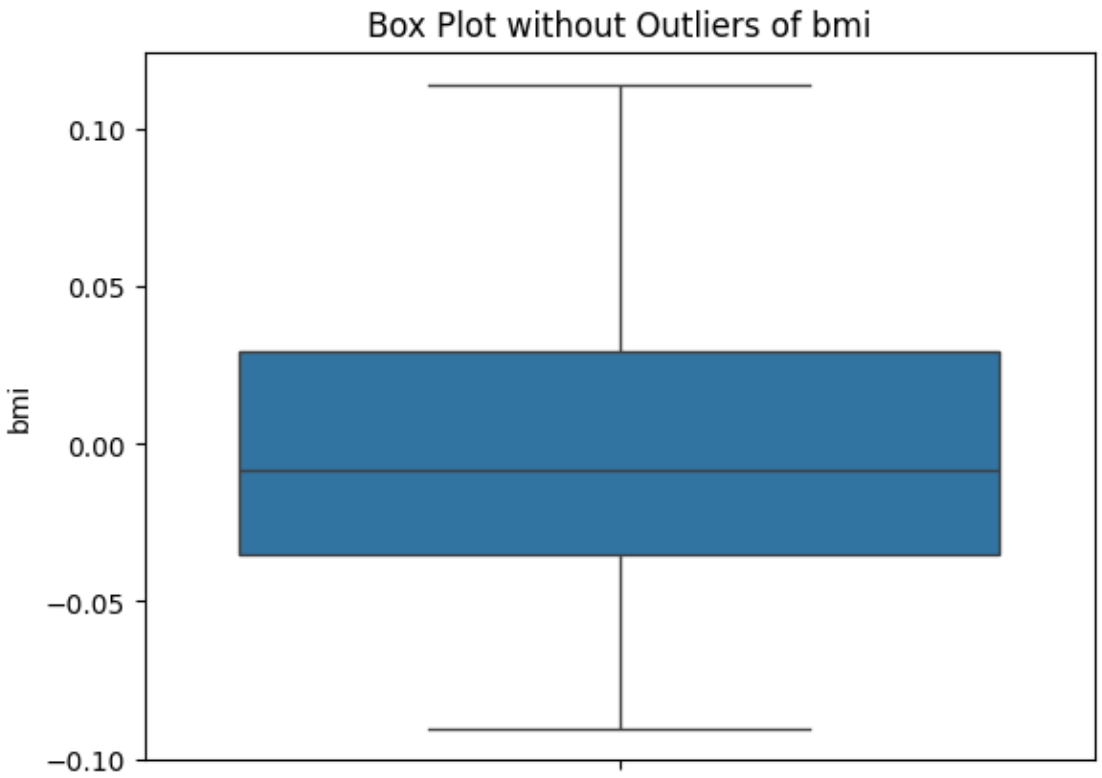


'A' appears to have higher median and higher variability than 'B'.

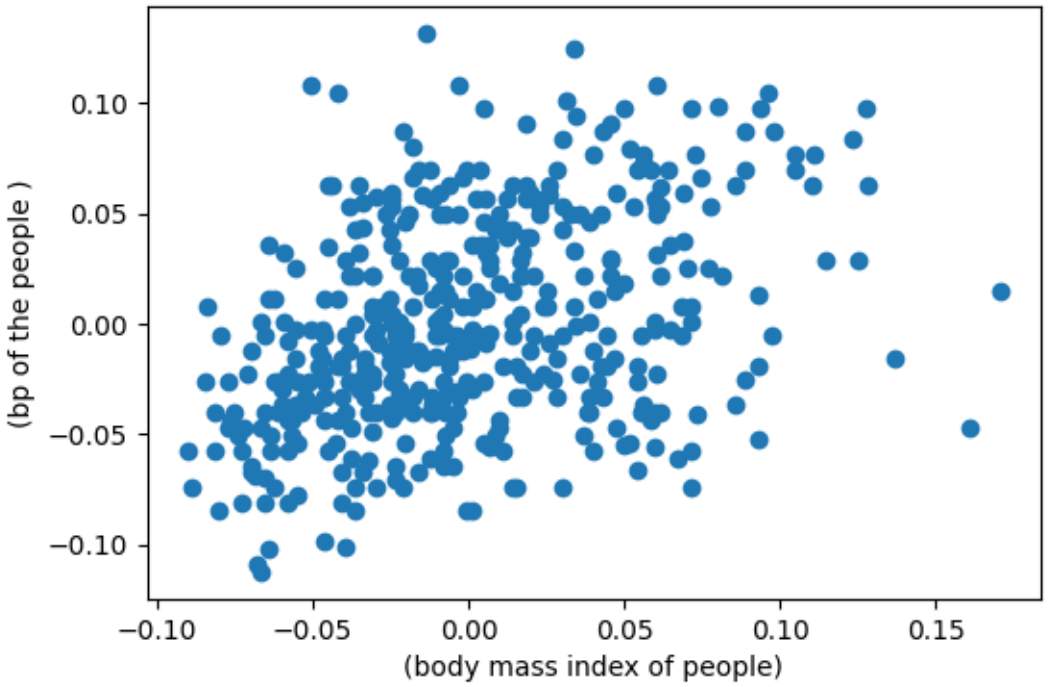
Box plots are ideal to represent moderate to large amount of data. The size of the box plot can vary significantly if the data size is small.

[5]:
```
import seaborn as sns
import matplotlib.pyplot as plt


def removal_box_plot(df, column, threshold):
    sns.boxplot(df[column])
    plt.title(f'Original Box Plot of {column}')
    plt.show()

    removed_outliers = df[df[column] <= threshold]

    sns.boxplot(removed_outliers[column])
    plt.title(f'Box Plot without Outliers of {column}')
    plt.show()
    return removed_outliers


threshold_value = 0.12

no_outliers = removal_box_plot(df_diabetics, 'bmi', threshold_value)
```





[6]:
```
fig, ax = plt.subplots(figsize=(6, 4))
ax.scatter(df_diabetics['bmi'], df_diabetics['bp'])
ax.set_xlabel('(body mass index of people)')
ax.set_ylabel('(bp of the people )')
plt.show()
```

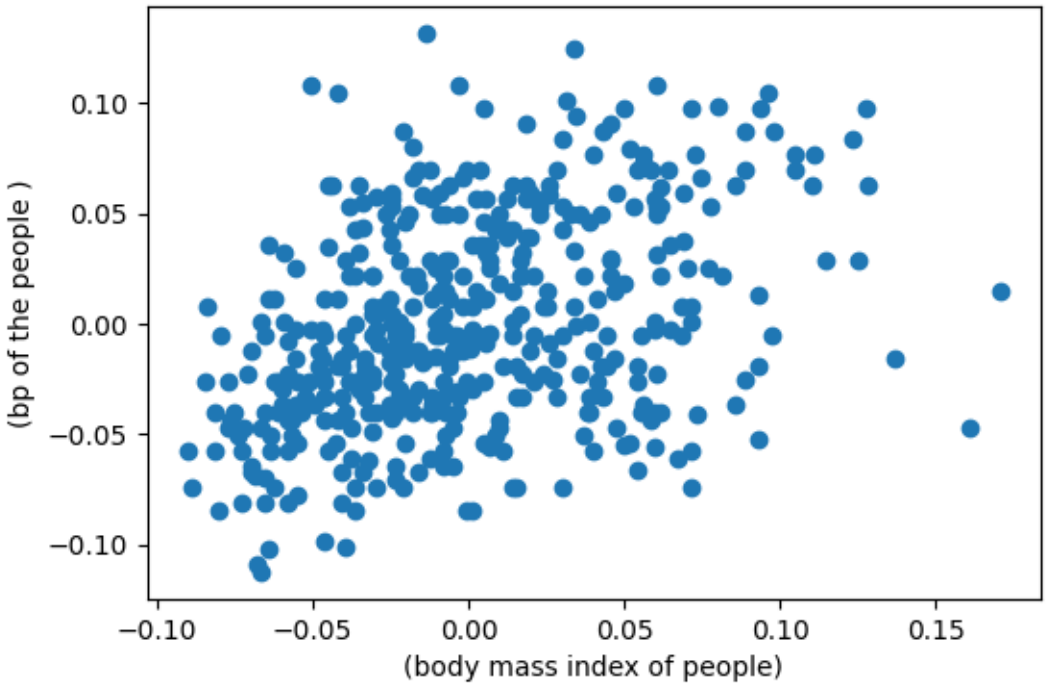


1.3 Visualisation et suppression des outlier à l'aide d'un nuage de points

Cette méthode est utilisée lorsque vous disposez de données numériques appariées, lorsque votre variable dépendante a plusieurs valeurs pour chaque valeur de la variable indépendante, ou lorsque vous essayez de déterminer la relation entre deux variables. En utilisant le nuage de points, il est également possible de détecter des outlier dans le jeu de données.

Pour tracer un nuage de points, deux variables ayant une relation entre elles sont nécessaires. Dans cet exemple, les variables « Proportion des hectares d'entreprises non commerciales par ville » et « Taux d'imposition sur la propriété pleine valeur pour 10 000 $ » sont utilisées. Leurs noms de colonnes respectifs sont **« INDUS »** et **« TAX »**.

[7]:
```
fig, ax = plt.subplots(figsize=(6, 4))
ax.scatter(df_diabetics['bmi'], df_diabetics['bp'])
ax.set_xlabel('(body mass index of people)')
ax.set_ylabel('(bp of the people )')
plt.show()
```



En examinant le graphique, on peut résumer que la majorité des points de données se trouvent dans le coin inférieur gauche du graphique, mais il y a quelques points qui se situent exactement à l'opposé, c'est-à-dire dans le coin supérieur droit. Ces points dans le coin supérieur droit peuvent être considérés comme des outlier.

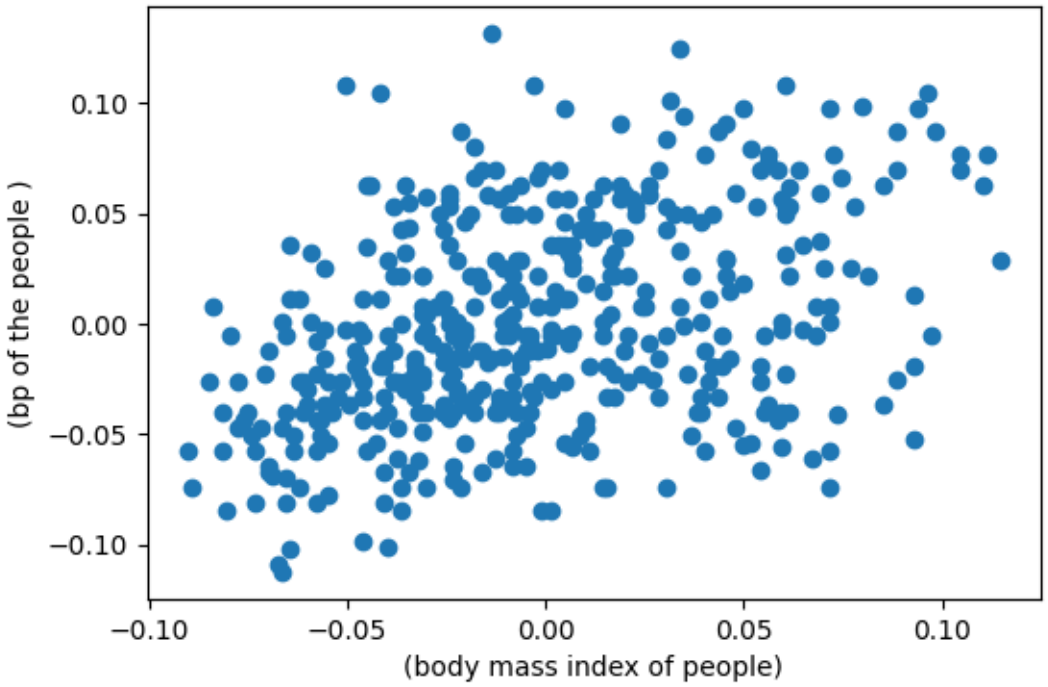
En utilisant une approximation, on peut dire que tous les points de données pour lesquels $\mathbf{x} > \mathbf{20}$ et $\mathbf{y} > \mathbf{600}$ sont des outlier. Le code suivant permet de trouver la position exacte de tous ces points qui satisfont ces conditions.

1.3.1 Suppression des outlier dans les colonnes BMI et BP combinées

Ici, la fonction `np.where()` de NumPy est utilisée pour trouver les positions (indices) où la condition `(df_diabetics['bmi'] > 0.12) & (df_diabetics['bp'] < 0.8)` est vraie dans le DataFrame **df_diabetics**. Cette condition identifie les outliers où **'bmi'** est supérieur à 0.12 et **'bp'** est inférieur à 0.8. Le résultat donne les indices des lignes et des colonnes des positions des outlier dans le DataFrame.

[8]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

outlier_indices = np.where((df_diabetics['bmi'] > 0.12) & (df_diabetics['bp'] <␣
↪0.8))

no_outliers = df_diabetics.drop(outlier_indices[0])
```

```
# Scatter plot without outliers
fig, ax_no_outliers = plt.subplots(figsize=(6, 4))
ax_no_outliers.scatter(no_outliers['bmi'], no_outliers['bp'])
ax_no_outliers.set_xlabel('(body mass index of people)')
ax_no_outliers.set_ylabel('(bp of the people )')
plt.show()
```



2 Z-score

Le Z-score, également appelé score standard, est une valeur qui permet de comprendre à quelle distance un point de données se trouve de la moyenne. En définissant une valeur seuil, on peut utiliser les Z-scores des points de données pour identifier les outlier.

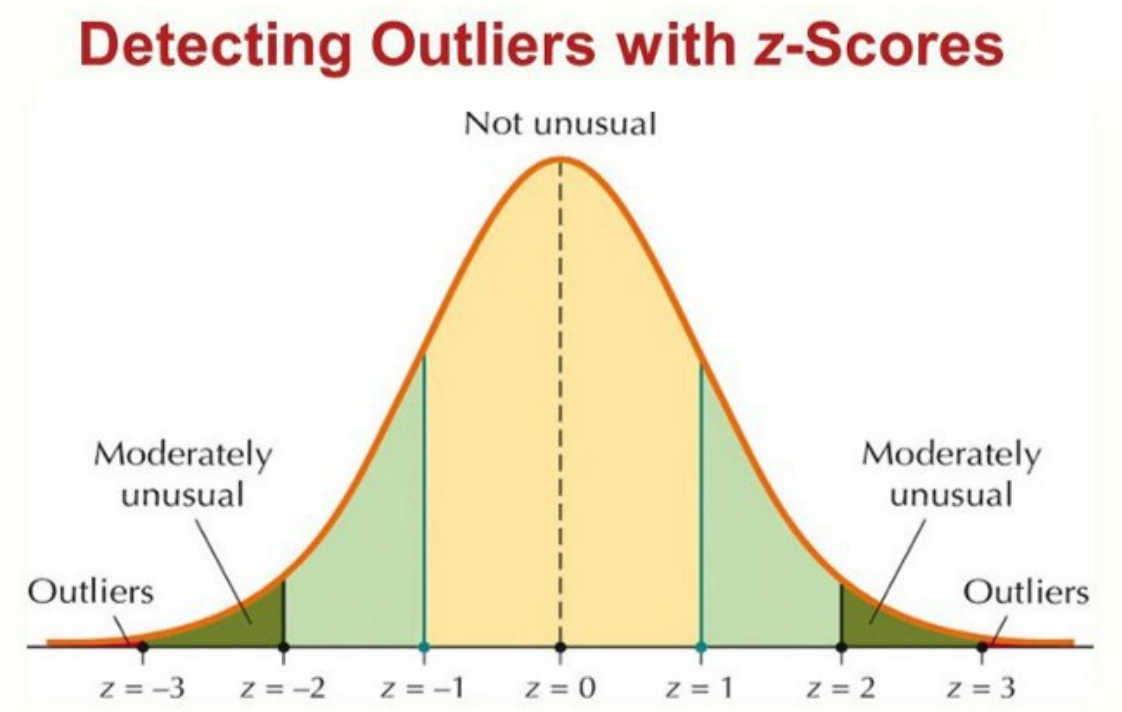
Zscore = (valeur_donnée - moyenne) / écart_type

Dans cet exemple, nous calculons les Z-scores pour la colonne **'age'** du DataFrame **df_diabetics** en utilisant la fonction `zscore` du module **stats** de SciPy. Le tableau résultant **z** contient les valeurs absolues des Z-scores pour chaque point de données de la colonne **'age'**, indiquant de combien d'écarts-types chaque valeur s'écarte de la moyenne.

[9]:
```
from scipy import stats
import numpy as np
z = np.abs(stats.zscore(df_diabetics['age']))
print(z)
```

```
0      0.800500
1      0.039567
2      1.793307
3      1.872441
4      0.113172
         …
437    0.876870
438    0.115937
439    0.876870
440    0.956004
441    0.956004
Name: age, Length: 442, dtype: float64
```

Maintenant, pour définir une valeur seuil pour les outlier, on choisit généralement **3.0**. En effet, 99,7 % des points de données se situent entre +/- 3 écarts-types (selon l'approche de la distribution gaussienne).



2.0.1 Suppression des outlier avec le Z-Score

Supprimons les lignes où la valeur du Z-Score est supérieure à 2.

Dans cet exemple, une valeur seuil de **2** est définie, puis la fonction `np.where()` de NumPy est utilisée pour identifier les positions (indices) dans le tableau des Z-scores **z** où le Z-Score absolu est supérieur au seuil spécifié (**2**). Les positions des outlier dans la colonne **'age'** sont affichées en fonction du critère du Z-Score.

[10]:
```
import numpy as np
```

```
threshold_z = 2

outlier_indices = np.where(z > threshold_z)[0]
no_outliers = df_diabetics.drop(outlier_indices)
print("Original DataFrame Shape:", df_diabetics.shape)
print("DataFrame Shape after Removing Outliers:", no_outliers.shape)
```

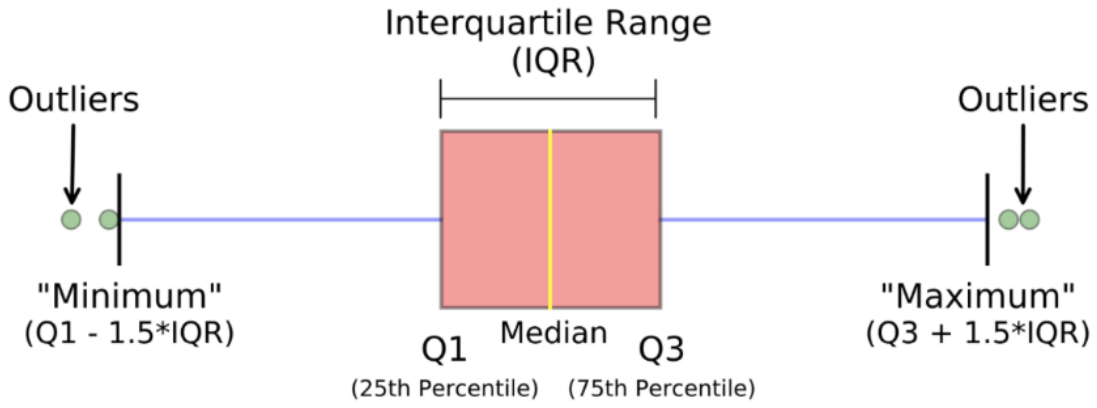
```
Original DataFrame Shape: (442, 10)
DataFrame Shape after Removing Outliers: (426, 10)
```

3 IQR (Inter-Quartile Range)

L'approche de l'IQR (Inter-Quartile Range) pour identifier les outlier est l'une des plus couramment utilisées et les plus fiables dans le domaine de la recherche.

$IQR = Quartile3 - Quartile1$

Dans cet exemple, nous calculons l'intervalle interquartile (IQR) pour la colonne **'bmi'** du DataFrame **df_diabetics**. La méthode commence par calculer le premier quartile (**Q1**) et le troisième quartile (**Q3**) en utilisant la méthode du point médian, puis détermine l'IQR comme la différence entre **Q3** et **Q1**. Cela fournit une mesure de la dispersion des 50 % des données centrales dans la colonne **'bmi'**.



[11]:
```
# np.percentile usage:
# np.percentile(array, n) returns the nth percentile of the array

# IQR
Q1 = np.percentile(df_diabetics['bmi'], 25, method='midpoint')
Q3 = np.percentile(df_diabetics['bmi'], 75, method='midpoint')
IQR = Q3 - Q1
print(IQR)
```

```
0.06520763046978838
```

Pour définir les valeurs limites des outlier, on établit des bornes supérieures et inférieures en fonction de la plage normale du jeu de données. Ces bornes sont définies comme suit (on considère une valeur

de **1.5*IQR**) :

- **upper = Q3 + 1.5*IQR**
- **lower = Q1 – 1.5*IQR**

Dans la formule ci-dessus, selon les statistiques, une augmentation de 0.5 de l'IQR est prise en compte (**new_IQR = IQR + 0.5*IQR**) **pour inclure toutes les données se situant dans** 2.7 écarts-types** de la distribution gaussienne.

[12]:
```
# Above Upper bound
upper = Q3+1.5*IQR
upper_array = np.array(df_diabetics['bmi'] >= upper)
print("Upper Bound:", upper)
print(upper_array.sum())

# Below Lower bound
lower = Q1-1.5*IQR
lower_array = np.array(df_diabetics['bmi'] <= lower)
print("Lower Bound:", lower)
print(lower_array.sum())
```

```
Upper Bound: 0.12879000811776306
3
Lower Bound: -0.13204051376139045
0
```

3.0.1 Suppression des outliers dans un jeu de données en utilisant l'IQR

Dans cet exemple, nous utilisons la méthode de l'intervalle interquartile (IQR) pour détecter et supprimer les outlier dans la colonne **'bmi'** du jeu de données sur le diabète.

La méthode calcule les limites supérieure et inférieure basées sur l'IQR, identifie les indices des outlier à l'aide de tableaux booléens, puis supprime les lignes correspondantes du DataFrame. Le résultat est un nouveau DataFrame avec les outlier exclues. Les dimensions du DataFrame avant et après la suppression des outlier sont affichées pour comparaison.

[13]:
```
# Importing
import sklearn
from sklearn.datasets import load_diabetes
import pandas as pd

# Load the dataset
diabetes = load_diabetes()

# Create the dataframe
column_name = diabetes.feature_names
df_diabetes = pd.DataFrame(diabetes.data)
df_diabetes .columns = column_name
df_diabetes .head()
print("Old Shape: ", df_diabetes.shape)
```

```
''' Detection '''
# IQR
# Calculate the upper and lower limits
Q1 = df_diabetes['bmi'].quantile(0.25)
Q3 = df_diabetes['bmi'].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR

# Create arrays of Boolean values indicating the outlier rows
upper_array = np.where(df_diabetes['bmi'] >= upper)[0]
lower_array = np.where(df_diabetes['bmi'] <= lower)[0]

# Removing the outliers
df_diabetes.drop(index=upper_array, inplace=True)
df_diabetes.drop(index=lower_array, inplace=True)

# Print the new shape of the DataFrame
print("New Shape: ", df_diabetes.shape)
```

```
Old Shape:  (442, 10)
New Shape:  (439, 10)
```

[ ]:

3.1 DBSCAN - Density-Based Spatial Clustering of Applications with Noise

DBSCAN is a powerful density-based data clustering algorithm. Clustering is an unsupervised learning technique where we try to group the data points based on specific characteristics. DBSCAN was proposed by Martin Ester et al. in 1996. It works on the assumption that clusters are dense regions in space separated by regions of lower density.

To cluster data points DBSCAN algorithm separates the high-density regions of the data from the low-density areas. It uses distance and a minimum number of points per cluster to classify a point as an outlier. This approch is similar to the K-mean clustering.

PARAMETERS:

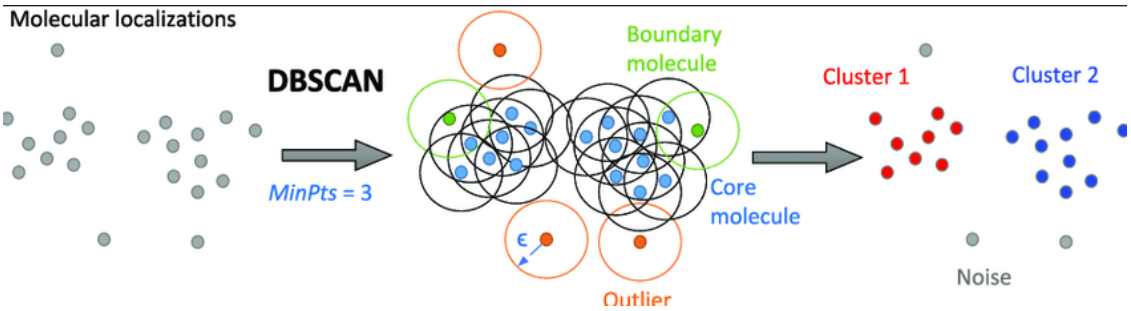
Epsilon is the radius of the circle to be created around each data point to check the density.

minPoints is the minimum number of data points required inside that circle for that data point to be classified as a Core point. This includes the point itself.

In higher dimensions the circle becomes hypersphere, epsilon becomes the radius of that hypersphere, and minPoints is the minimum number of data points required inside that hypersphere.

DBSCAN algorithm doesn't make assumptions about how data are distributed.

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html



Let's first run DBSCAN without any parameter optimization and see the results.

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

# scale data first
X = StandardScaler().fit_transform(df6.values)

db = DBSCAN(eps=3.0, min_samples=10).fit(X)
labels = db.labels_
```

[ ]:
```
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print('The number of clusters in dataset is:', n_clusters_)
```

```
The number of clusters in dataset is: 39
```

The number of clusters does not include outliers/noise in the dataset.

Labels are the labels of the clusters. If the label is -1, then the observation is an outlier/noise.

[ ]:
```
pd.Series(labels).value_counts()
```

```
 0     197235
 1      32473
-1      20243
 9      16078
 2      12290
 5       2047
 7       1416
 10       828
 3        333
 24       287
 12       214
 16       213
 13       206
 17       181
 11       170
 18        81
 4         80
```

```
 30         48
 37         36
 20         32
 22         32
 23         25
 31         24
 14         21
 29         20
 15         19
 25         18
 28         17
 26         16
 33         16
 21         15
 32         14
 34         12
 35         12
 38         10
 6          10
 8          10
 36         10
 19          8
 27          7
dtype: int64
```

4 Conclusion

En conclusion, des outils de visualisation comme les boîtes à moustaches (**box plots**) et les nuages de points (**scatter plots**) permettent d'identifier les outlier, tandis que des méthodes mathématiques telles que les Z-scores et l'Intervalle Interquartile (**IQR**) offrent des approches robustes pour leur détection et leur gestion.