Cleaning up Data from Outliers

```
import sys
sys.path
#!pip install virtualenv
#!virtualenv /content/drive/MyDrive/virtual_env
```

```
['/content',
 '/env/python',
 '/usr/lib/python310.zip',
 '/usr/lib/python3.10',
 '/usr/lib/python3.10/lib-dynload',
 '',
 '/usr/local/lib/python3.10/dist-packages',
 '/usr/lib/python3/dist-packages',
 '/usr/local/lib/python3.10/dist-packages/IPython/extensions',
 '/usr/local/lib/python3.10/dist-packages/setuptools/_vendor',
 '/root/.ipython']
```

```
#!source /content/drive/MyDrive/virtual_env/bin/activate; pip install pyod
```

```
sys.path.append("/content/drive/MyDrive/virtual_env/lib/python3.10/site-packages")
```

```
# dislay all outputs in a cell (https://rb.gy/lbz7hn)
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Definition

- Outliers are observations that are significantly different from other data points.
- They are rare, distinct, or do not fit in some way.
- samples that are exceptionally far from the mainstream of the data.

Reason

> There can be many reasons for the presence of outliers in the data.
>
> - Sometimes the outliers may be genuine,
> - Sometimes because of data entry errors.

Detection/Identification

1. Univariate Outlier Detection

Outliers occur within a single variable.

> We will start the process of finding outliers by
>
> - Numerical/Quantitative methods
>   - describe
>   - IQR
>   - skewness
>   - z-score
>   - Modified Z-scores:
>   - Uses the `median` and `Median Absolute Deviation` (MAD) instead of `mean` and `standard deviation`
>   - more robust for skewed data.
> - Visualization methods
>   - Boxplot

    - Histogram
    - Scatterplot

- Mean is the only measure of central tendency that is affected by the outlier treatment which in turn impacts Standard deviation.

2. Multivariate Outlier Detection

1. Isolation Forest:
    - Isolates anomalies faster than normal data during random partitioning.
2. Local Outlier Factor (LOF):
    - Identifies outliers based on local density deviation from neighbors.
3. Clustering Techniques (K-means, Hierarchical):
    - Detect points far from established clusters or in small clusters.
4. Angle-based Outlier Detection (ABOD):
    - Analyzes angles between data points in high dimensions.

3. Machine Learning Based Approaches

1. Density-Based Anomaly Detection:
    - K-Nearest Neighbors (k-NN): Classifies based on nearest neighbors.
    - Local Outlier Factor (LOF): Scores data points based on neighbors' density compared to their own.
2. Clustering-Based Anomaly Detection:
    - K-means Algorithm: Common technique to group similar data points into clusters. Data points far from any cluster are flagged as anomalies.
3. Support Vector Machine-Based Anomaly Detection: One-Class SVM: Learns a boundary around normal data points, identifies anomalies as points falling outside the boundary.
4. Moving Average Using Discrete Linear Convolution: Smooths data to identify anomalies or deviations from the trend.

4. Gaussian Distribution

Assumes data follows a bell-shaped normal distribution curve. Fits a Gaussian distribution model to the data and identifies points with very low probability as anomalies.

5. Autoencoders (in Neural Networks)

Encode and reconstruct data points, trained to minimize reconstruction error and flag anomalies. Implemented with libraries like Keras and PyTorch.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df=sns.load_dataset('diamonds')


df.head()
df.shape
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

(53940, 10)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  category
 2   color    53940 non-null  category
 3   clarity  53940 non-null  category
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: category(3), float64(6), int64(1)
memory usage: 3.0 MB
```

```
# Remove all (categorical variables) columns between column index 1 to 3
dfnum=df.drop(df.iloc[:, 1:4], axis=1)  #delete multiple columns based on column numbers
# dfnum=df.drop(['cut','color','clarity'], axis = 1) #delete multiple columns based on columns names
dfnum.head(3)
```

	carat	depth	table	price	x	y	z
0	0.23	61.5	55.0	326	3.95	3.98	2.43
1	0.21	59.8	61.0	326	3.89	3.84	2.31
2	0.23	56.9	65.0	327	4.05	4.07	2.31

1 Numerical Methods

1.1 Describe

- `describe()` function provides a statistical summary of all the quantitative variables.

```
dfnum.describe()
```

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

```
dfnum.isna().sum()
```

	0
carat	0
depth	0
table	0
price	0
x	0
y	0
z	0

dtype: int64

```
dfnum['table']
```

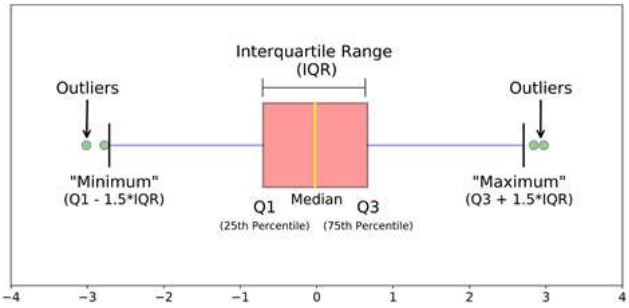
	table
0	55.0
1	61.0
2	65.0
3	58.0
4	58.0
...	...
53935	57.0
53936	55.0
53937	60.0
53938	58.0
53939	55.0

53940 rows × 1 columns

dtype: float64

1.2 Interquartile range (IQR)

- a measure of statistical dispersion
- the difference between the 75th and 25th percentiles IQR = Q3 − Q1.
- scores that lie 1.5 times of IQR above Q3 and below Q1 are outliers.



Steps

1. Sort the dataset in ascending order
2. calculate the 1st and 3rd quartiles(Q1, Q3)
3. compute IQR=Q3-Q1
4. compute lower bound = (Q1−1.5*IQR), upper bound = (Q3+1.5*IQR)

```
sample= [15, 101, 18, 7, 13, 16, 11, 21, 5, 15, 10, 9]
def detect_outliers_iqr(data):
    data = sorted(data)
```

```
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    # print(q1, q3)
    IQR = q3-q1
    lwr_bound = q1-(1.5*IQR)
    upr_bound = q3+(1.5*IQR)
    # print(lwr_bound, upr_bound)
    outliers = [i for i in data if (i<lwr_bound or i>upr_bound)]
    return outliers# Driver code
sample_outliers = detect_outliers_iqr(sample)
print("Outliers from IQR method: ", sample_outliers)
```

```
Outliers from IQR method:  [101]
```

```
Q1 = dfnum.quantile(0.25)
print(Q1)
```

```
carat      0.40
depth     61.00
table     56.00
price    950.00
x          4.71
y          4.72
z          2.91
Name: 0.25, dtype: float64
```

```
Q3 = dfnum.quantile(0.75)
print(Q3)
```

```
carat       1.04
depth      62.50
table      59.00
price    5324.25
x           6.54
y           6.54
z           4.04
Name: 0.75, dtype: float64
```

```
IQR = Q3 - Q1
print(IQR)
```

```
carat       0.64
depth       1.50
table       3.00
price    4374.25
x           1.83
y           1.82
z           1.13
dtype: float64
```

```
df1 = dfnum[~((dfnum.lt (Q1 - 1.5 * IQR)) |(dfnum.gt (Q3 + 1.5 * IQR))).any(axis=1)]

print(df1)
```

```
        carat  depth  table  price     x     y     z
0        0.23   61.5   55.0    326  3.95  3.98  2.43
1        0.21   59.8   61.0    326  3.89  3.84  2.31
3        0.29   62.4   58.0    334  4.20  4.23  2.63
4        0.31   63.3   58.0    335  4.34  4.35  2.75
5        0.24   62.8   57.0    336  3.94  3.96  2.48
...       ...    ...    ...    ...   ...   ...   ...
53935    0.72   60.8   57.0   2757  5.75  5.76  3.50
53936    0.72   63.1   55.0   2757  5.69  5.75  3.61
53937    0.70   62.8   60.0   2757  5.66  5.68  3.56
53938    0.86   61.0   58.0   2757  6.15  6.12  3.74
53939    0.75   62.2   55.0   2757  5.83  5.87  3.64

[47524 rows x 7 columns]
```

```
df2 = dfnum[~((dfnum.le(Q1 - 1.5 * IQR)) |(dfnum.ge (Q3 + 1.5 * IQR))).all(axis=1)]

print(df2)
```

```
        carat  depth  table  price     x     y     z
0        0.23   61.5   55.0    326  3.95  3.98  2.43
1        0.21   59.8   61.0    326  3.89  3.84  2.31
2        0.23   56.9   65.0    327  4.05  4.07  2.31
3        0.29   62.4   58.0    334  4.20  4.23  2.63
```

```
4       0.31   63.3   58.0    335  4.34  4.35  2.75
...      ...    ...    ...    ...   ...   ...   ...
53935   0.72   60.8   57.0   2757  5.75  5.76  3.50
53936   0.72   63.1   55.0   2757  5.69  5.75  3.61
53937   0.70   62.8   60.0   2757  5.66  5.68  3.56
53938   0.86   61.0   58.0   2757  6.15  6.12  3.74
53939   0.75   62.2   55.0   2757  5.83  5.87  3.64

[53940 rows x 7 columns]
```

1.3 Skewness

- The skewness value should be between -1 and +1, and any major deviation from this range indicates the presence of extreme values.
- Apply transformation if skewness is high.

```
if np.abs(skewness) > 1:
      print(f"Transforming column {col} due to high skewness.")
      dfnum[col] = np.log1p(dfnum[col] - dfnum[col].min() + 1)
```

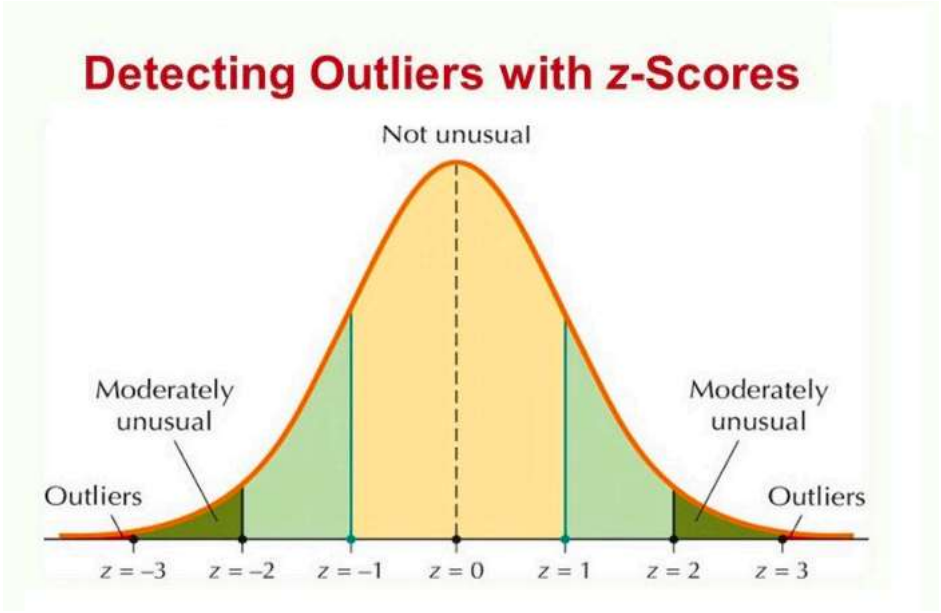
```
dfnum.skew()
```

	0
carat	1.116646
depth	-0.082294
table	0.796896
price	1.618395
x	0.378676
y	2.434167
z	1.522423

dtype: float64

1.4 Z-score

- Any data point whose Z-score falls out of 3rd standard deviation is an outlier.




```
def detect_outliers_zscore(data):
    thres = 3
    mean = np.mean(data)
    std = np.std(data)
```

```
    # print(mean, std)
    outliers = [i for i in data if np.abs((i - mean) / std) > thres] #list comprehension
    return outliers# Driver code
sample_outliers = detect_outliers_zscore(sample)
print("Outliers from Z-scores method: ", sample_outliers)
```

```
Outliers from Z-scores method:  [101]
```

1.5 Standard deviation

- A value that falls outside of 3 standard deviations is part of the distribution, but it is an unlikely or rare event at approximately 1 in 370 samples.
- 3 σ from the mean is a common cut-off in practice for identifying outliers in a Gaussian or Gaussian-like distribution.

```
def detect_outliers_stdv(data):
    data_mean, data_std = np.mean(data), np.std(data)
    # print(mean, std)
    # define outliers
    cut_off = data_std * 3
    lower, upper = data_mean - cut_off, data_mean + cut_off
    # identify outliers
    outliers = [i for i in data if i < lower or i > upper] #list comprehension
    return outliers# Driver code
sample_outliers = detect_outliers_stdv(sample)
print("Outliers from Standard dev method: ", sample_outliers)
```

```
Outliers from Standard dev method:  [101]
```

2 Visualization/Graphical Methods

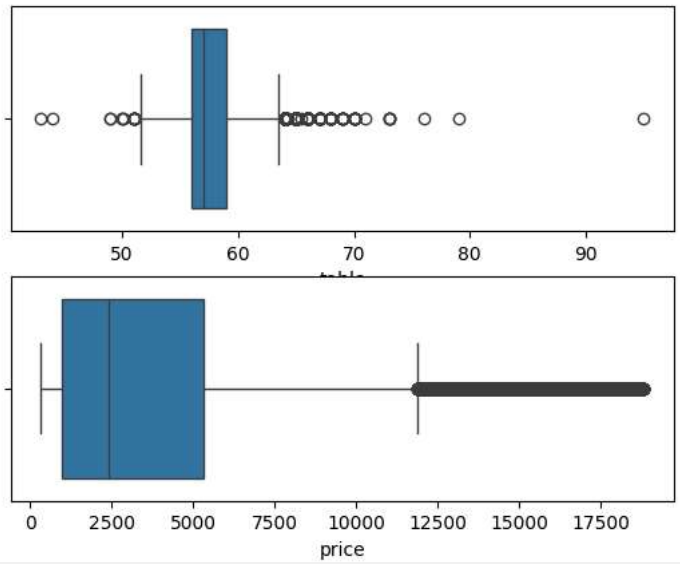
Some of the common plots used for outlier detection are

2.1 Box Plot

> - The box plot is a standardized way of displaying the distribution of data based on the five-number summary (minimum, first quartile (Q1), median, third quartile (Q3), and maximum).
> - It is often used to identify data distribution and detect outliers.

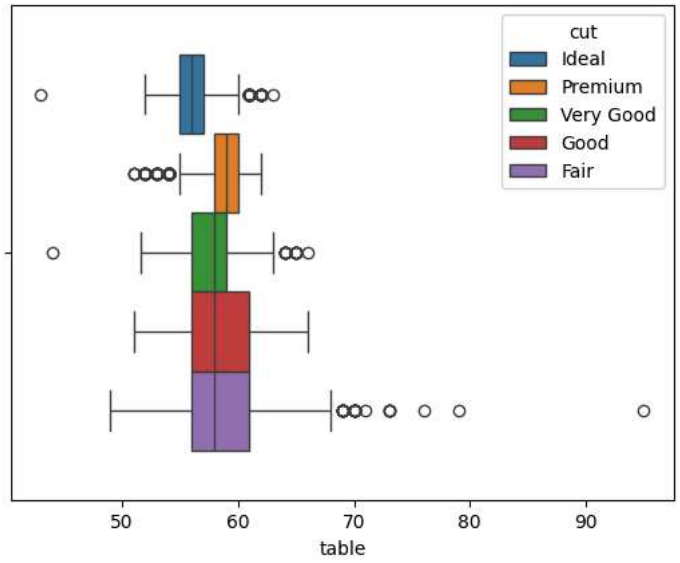
```
fig, axs = plt.subplots(nrows=2)
sns.boxplot(data=df, x='table', ax=axs[0])
sns.boxplot(data=df, x='price', ax=axs[1])
```

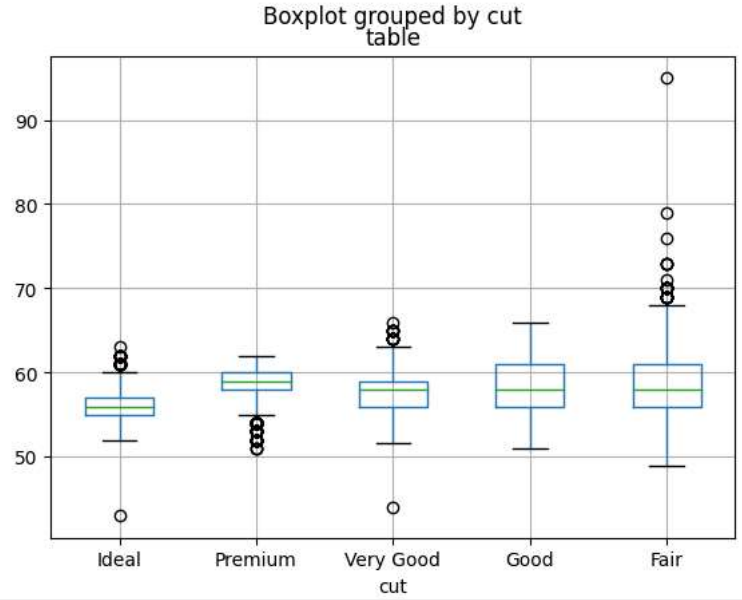
```
<Axes: xlabel='table'>
<Axes: xlabel='price'>
```



```
sns.boxplot(data=df,x='table', hue='cut')
df.boxplot(column='table', by='cut')
```

```
<Axes: xlabel='table'>
<Axes: title={'center': 'table'}, xlabel='cut'>
```



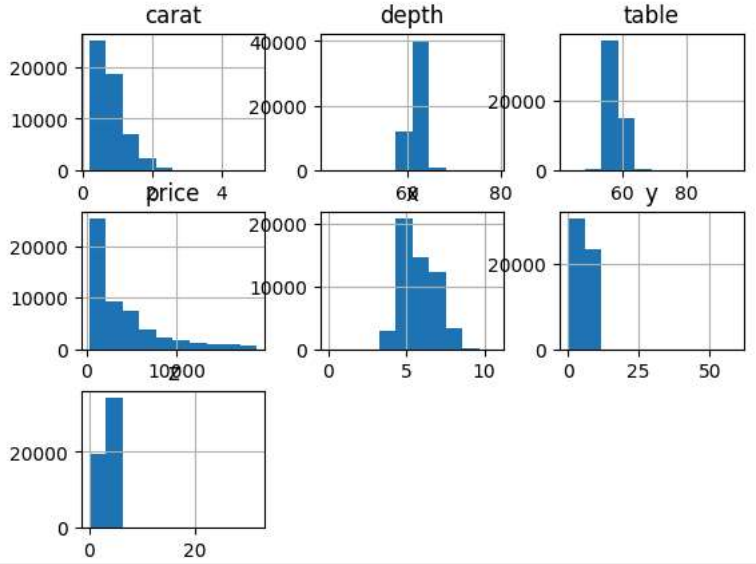


2.2 Histogram

> - A histogram is used to visualize the distribution of a numerical variable.
> - An outlier will appear outside the overall pattern of distribution.

```
dfnum.hist()
```

```
array([[<Axes: title={'center': 'carat'}>,
        <Axes: title={'center': 'depth'}>,
        <Axes: title={'center': 'table'}>],
       [<Axes: title={'center': 'price'}>, <Axes: title={'center': 'x'}>,
        <Axes: title={'center': 'y'}>],
       [<Axes: title={'center': 'z'}>, <Axes: >, <Axes: >]], dtype=object)
```

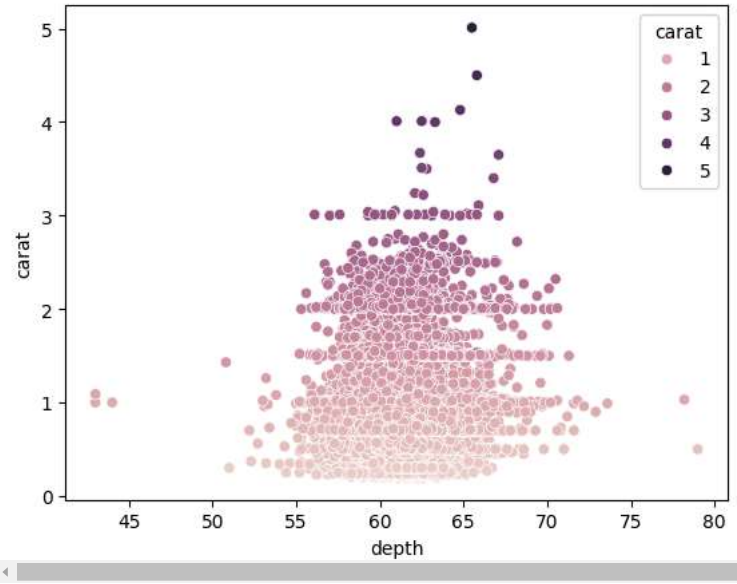


2.3 Scatterplot

- A scatterplot visualizes the relationship between two numerical variables.
- The data are displayed as a collection of points, and any points that fall outside the general clustering of the two variables may indicate outliers.

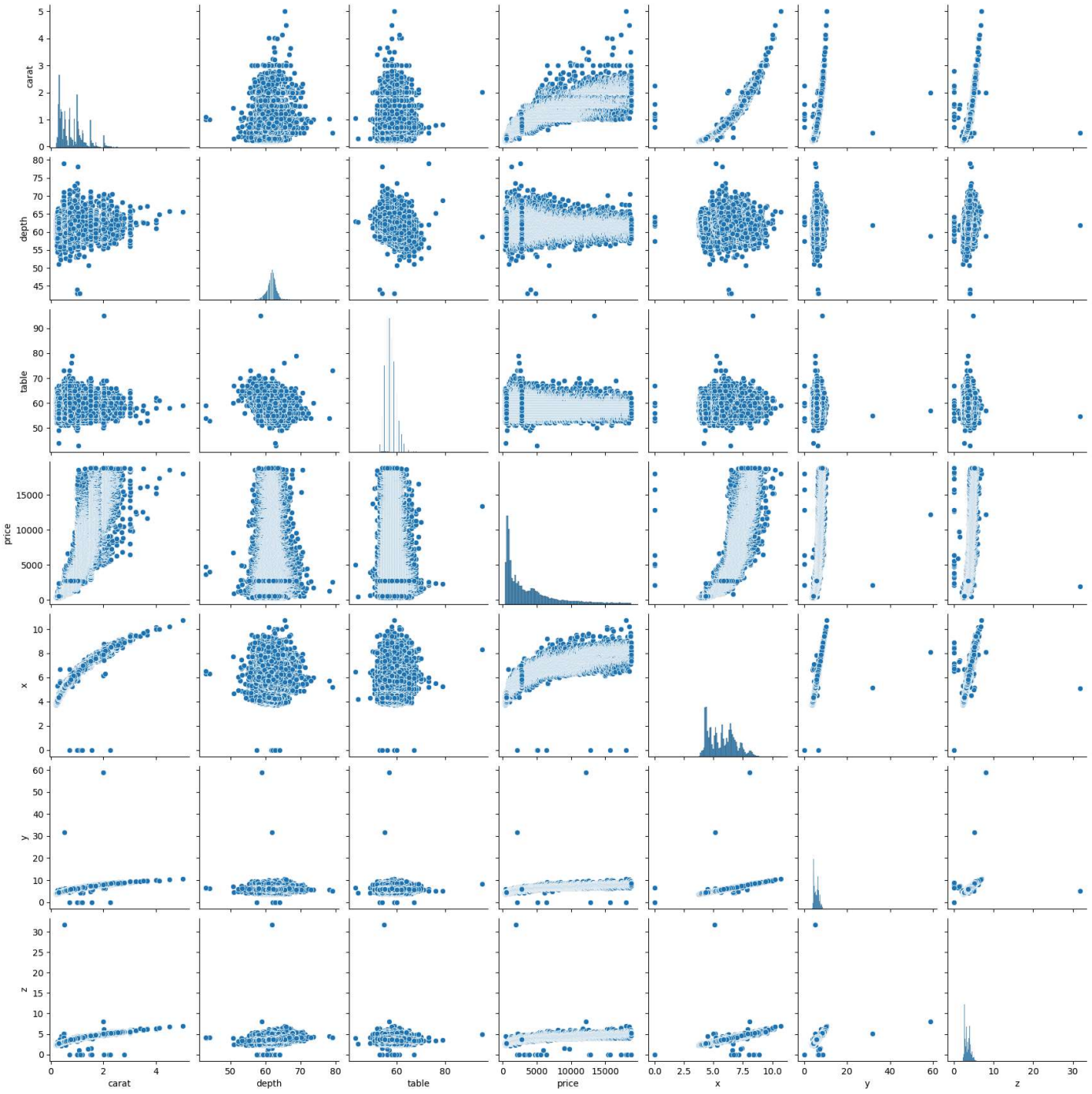
```
sns.scatterplot(data=df,x='depth',y='carat',hue='carat')
```

```
<Axes: xlabel='depth', ylabel='carat'>
```




```
sns.pairplot(dfnum)
```

<seaborn.axisgrid.PairGrid at 0x7d3f3ebd1390>



3 Advanced Algorithms

Outlier detector algorithms are of two types:

- Ones that consider each feature independently (such as HBOS, z-score, inter-quartile range, entropy-based tests and so on) works strictly with numeric data.
- Ones that consider all features at once (such as Isolation Forest IF, Local Outlier Factor LOF and KNN).
- popular `PyOD` (Python Outlier Detection) library

CountsOutlierDetector

- is an interpretable, intuitive, efficient outlier detector intended for tabular data.
- designed to provide clear explanations of the rows flagged as outliers and of their specific scores.
- is a multivariate histogram-based model
- standard detectors as `Isolation Forest` (IF), `Local Outlier Factor` (LOF), and `k Nearest Neighbors` (kNN), have algorithms that are straight-forwared to understand, but produce scores that may be difficult to assess.
- `CountsOutlierDetector` attempts to solve these issues by providing a highly transparent, non-stocastic outlier detection system.

- Working
  - finds `1D outliers` (outliers based on considering a single dimension at a time) by examing
    - *each column individually* and
    - *identifying all values that are unusual* with respect to their columns.
  - finds `2D outliers` by examining each pair of columns, *identifying the rows with pairs of unusual values* within each pair of columns.
    - For example, having fur may be common, as well as laying eggs, but the combination is rare.
    - with numeric columns, an age of 1 yr may be common and a height of 6' as well, but the combination rare, most likely flagging an error.
  - finds `nD outliers` : detector then considers sets of 3 columns, sets of 4 columns, and so on.
  - At each stage, the algorithm looks for instances that are unusual

Categorical vs Numeric Outlier Detectors

- most outlier detectors may be considered to be one of two types:
  - categorical
    - all columns are assumed to be categorical, and any numeric columns must be binned.
    - binning the values can loose some information, as the order of the bins is lost
    - can add greatly to model interpretability as it eliminates `data scaling` and `defining a distance metric` that removes any processing time required to calculate the distances between points
  - numeric-based detectors
    - all columns are assumed to be numeric, and any categorical columns must be numerically encoded using one-hot encoding or count encoding.
- `CountsOutlierDetector` is an example of the former, treating all features as categorical.
  - To determine if a value is rare, `COD` simply examines the count of its value;
  - to determine if a pair of values is rare, it simply examines the count of this pair of values, compared to other pairs of values in the two columns.
  - There may be some loss of fidelity in some cases, but overall, this allows for fast and interpretable results

```
CountsOutlierDetector(
            n_bins=7,
            bin_names=None,
            max_dimensions=3,
            threshold=0.05,
            check_marginal_probs=False,
            max_num_combinations=100_000,
            min_values_per_column=2,
            max_values_per_column=25,
```

```
            results_folder="",
            results_name="",
            run_parallel=False,
            verbose=False)
```

functions `predict(), get_most_flagged_rows(), plot_scores_distribution(),print_run_summary(),explain_row(row_index, max_plots=50),explain_features(features_arr)`

```
import os, sys
#from google.colab import drive
#drive.mount('/content/drive')
#nb_path = '/content/notebooks'
#os.symlink('/content/drive/My Drive/Colab Notebooks', nb_path)
sys.path.append('/content/drive/My Drive/Colab Notebooks')
#!pip install counts_outlier_detector


sys.path.insert(0, '..\\CountsOutlierDetector')
from counts_outlier_detector import CountsOutlierDetector


from sklearn.datasets import load_iris
iris = load_iris()
X, y = iris.data, iris.target
det = CountsOutlierDetector()
#dir(det) # shows all attributes of the package
results = det.fit_predict(X)
results
```

```
{'Scores': 0      0
      1      0
      2      0
      3      0
      4      0
      5      0
      6      0
      7      0
      8      0
      9      0
      10     0
      11     0
      12     0
      13     0
      14     0
      15     0
      16     0
      17     0
      18     0
      19     0
      20     0
      21     0
      22     0
      23     0
      24     0
      25     0
      26     0
      27     0
      28     0
      29     0
      30     0
      31     0
      32     0
      33     0
      34     0
      35     0
      36     0
      37     0
      38     0
      39     0
      40     0
      41     0
      42     0
      43     0
      44     0
      45     0
      46     0
      47     0
      48     0
      49     0
      50     0
```

```
51    0
52    0
53    0
54    0
55    0
56    0
57    0
```

PyOD

PyOD, established in 2017, has become a go-to Python library for detecting anomalous/outlying objects in multivariate data. This exciting yet challenging field is commonly referred to as `Outlier Detection` or `Anomaly Detection`.

- PyOD includes more than 50 detection algorithms, from classical LOF (SIGMOD 2000) to the cutting-edge ECOD and DIF (TKDE 2022 and 2023).
- PyOD offers a range of algorithms to suit your needs.
  - For time-series outlier detection, please use `TODS`.
  - For graph outlier detection, please use `PyGOD`.
- PyOD is featured for:
  - Unified, User-Friendly Interface across various algorithms.
  - Wide Range of Models, from classic techniques to the latest deep learning methods in `PyTorch`.
  - High Performance & Efficiency, leveraging `numba` and `joblib` for JIT compilation and parallel processing.
  - Fast Training & Prediction, achieved through the `SUOD` framework (accelerating large-scale unsupervised heterogeneous outlier detection).
- Installation

```
!pip install pyod
```

Implemented Algorithms

PyOD toolkit consists of three major functional groups:

(i) Individual Detection Algorithms :

Type	Abbr	Algorithm	Year	Class	
Probabilistic	ECOD	Unsupervised Outlier Detection Using Empirical Cumulative Distribution Functions	2022	pyod.models.ecod.ECOD	[AL
Probabilistic	COPOD	COPOD: Copula-Based Outlier Detection	2020	pyod.models.copod.COPOD	[AL
Probabilistic	ABOD	Angle-Based Outlier Detection	2008	pyod.models.abod.ABOD	[AK
Probabilistic	FastABOD	Fast Angle-Based Outlier Detection using approximation	2008	pyod.models.abod.ABOD	[AK
Probabilistic	MAD	Median Absolute Deviation (MAD)	1993	pyod.models.mad.MAD	[AI
Probabilistic	SOS	Stochastic Outlier Selection	2012	pyod.models.sos.SOS	[AJ
Probabilistic	QMCD	Quasi-Monte Carlo Discrepancy outlier detection	2001	pyod.models.qmcd.QMCD	[AF
Probabilistic	KDE	Outlier Detection with Kernel Density Functions	2007	pyod.models.kde.KDE	[AL
Probabilistic	Sampling	Rapid distance-based outlier detection via sampling	2013	pyod.models.sampling.Sampling	[AS
Probabilistic	GMM	Probabilistic Mixture Modeling for Outlier Analysis		pyod.models.gmm.GMM	[AA
Linear Model	PCA	Principal Component Analysis (the sum of weighted projected distances to the eigenvector hyperplanes)	2003	pyod.models.pca.PCA	[AS
Linear Model	KPCA	Kernel Principal Component Analysis	2007	pyod.models.kpca.KPCA	[AH
Linear Model	MCD	Minimum Covariance Determinant (use the mahalanobis distances as the outlier scores)	1999	pyod.models.mcd.MCD	[AH
Linear Model	CD	Use Cook's distance for outlier detection	1977	pyod.models.cd.CD	[AC
Linear Model	OCSVM	One-Class Support Vector Machines	2001	pyod.models.ocsvm.OCSVM	[AS
Linear Model	LMDD	Deviation-based Outlier Detection (LMDD)	1996	pyod.models.lmdd.LMDD	[AA
Proximity-Based	LOF	Local Outlier Factor	2000	pyod.models.lof.LOF	[AB
Proximity-Based	COF	Connectivity-Based Outlier Factor	2002	pyod.models.cof.COF	[AT
Proximity-Based	Incr. COF	Memory Efficient Connectivity-Based Outlier Factor (slower but reduce storage complexity)	2002	pyod.models.cof.COF	[AT
Proximity-Based	CBLOF	Clustering-Based Local Outlier Factor	2003	pyod.models.cblof.CBLOF	[AH
Proximity-Based	LOCI	LOCI: Fast outlier detection using the local correlation integral	2003	pyod.models.loci.LOCI	[AP
Proximity-Based	HBOS	Histogram-based Outlier Score	2012	pyod.models.hbos.HBOS	[AG
Proximity-Based	kNN	k Nearest Neighbors (use the distance to the kth nearest neighbor as the outlier score	2000	pyod.models.knn.KNN	[AA
Proximity-Based	AvgKNN	Average kNN (use the average distance to k nearest neighbors as the outlier score)	2002	pyod.models.knn.KNN	[AA
Proximity-Based	MedKNN	Median kNN (use the median distance to k nearest neighbors as the outlier score)	2002	pyod.models.knn.KNN	[AA

Type	Abbr	Algorithm	Year	Class	
Proximity-Based	SOD	Subspace Outlier Detection	2009	pyod.models.sod.SOD	[AKI
Proximity-Based	ROD	Rotation-based Outlier Detection	2020	pyod.models.rod.ROD	[AA
Outlier Ensembles	IForest	Isolation Forest	2008	pyod.models.iforest.IForest	[ALT
Outlier Ensembles	INNE	Isolation-based Anomaly Detection Using Nearest-Neighbor Ensembles	2018	pyod.models.inne.INNE	[AB
Outlier Ensembles	DIF	Deep Isolation Forest for Anomaly Detection	2023	pyod.models.dif.DIF	[]
Outlier Ensembles	FB	Feature Bagging	2005	pyod.models.feature_bagging.FeatureBagging	[ALI
Outlier Ensembles	LSCP	LSCP: Locally Selective Combination of Parallel Outlier Ensembles	2019	pyod.models.lscp.LSCP	[AZI
Outlier Ensembles	XGBOD	Extreme Boosting Based Outlier Detection (Supervised)	2018	pyod.models.xgbod.XGBOD	[AZI
Outlier Ensembles	LODA	Lightweight On-line Detector of Anomalies	2016	pyod.models.loda.LODA	[AP
Outlier Ensembles	SUOD	SUOD: Accelerating Large-scale Unsupervised Heterogeneous Outlier Detection (Acceleration)	2021	pyod.models.suod.SUOD	[AZI
Neural Networks	AutoEncoder	Fully connected AutoEncoder (use reconstruction error as the outlier score)	2015	pyod.models.auto_encoder.AutoEncoder	[AA
Neural Networks	VAE	Variational AutoEncoder (use reconstruction error as the outlier score)	2013	pyod.models.vae.VAE	[AK
Neural Networks	Beta-VAE	Variational AutoEncoder (all customized loss term by varying gamma and capacity)	2018	pyod.models.vae.VAE	[ABI
Neural Networks	SO_GAAL	Single-Objective Generative Adversarial Active Learning	2019	pyod.models.so_gaal.SO_GAAL	[ALI
Neural Networks	MO_GAAL	Multiple-Objective Generative Adversarial Active Learning	2019	pyod.models.mo_gaal.MO_GAAL	[ALI
Neural Networks	DeepSVDD	Deep One-Class Classification	2018	pyod.models.deep_svdd.DeepSVDD	[AR
Neural Networks	AnoGAN	Anomaly Detection with Generative Adversarial Networks	2017	pyod.models.anogan.AnoGAN	[AS
Neural Networks	ALAD	Adversarially learned anomaly detection	2018	pyod.models.alad.ALAD	[AZI
Neural Networks	DevNet	Deep Anomaly Detection with Deviation Networks	2019	pyod.models.devnet.DevNet	[AP
Neural Networks	AE1SVM	Autoencoder-based One-class Support Vector Machine	2019	pyod.models.ae1svm.AE1SVM	[AN
Graph-based	R-Graph	Outlier detection by R-graph	2017	pyod.models.rgraph.RGraph	[AY
Graph-based	LUNAR	LUNAR: Unifying Local Outlier Detection Methods via Graph Neural Networks	2022	pyod.models.lunar.LUNAR	[AG

(ii) Outlier Ensembles & Outlier Detector Combination Frameworks:

Type	Abbr	Algorithm	Year	Ref	
Outlier Ensembles		Feature Bagging	2005	pyod.models.feature_bagging.FeatureBagging	[ALK05]
Outlier Ensembles	LSCP	LSCP: Locally Selective Combination of Parallel Outlier Ensembles	2019	pyod.models.lscp.LSCP	[AZNHL1
Outlier Ensembles	XGBOD	Extreme Boosting Based Outlier Detection (Supervised)	2018	pyod.models.xgbod.XGBOD	[AZH18]
Outlier Ensembles	LODA	Lightweight On-line Detector of Anomalies	2016	pyod.models.loda.LODA	[APevny1
Outlier Ensembles	SUOD	SUOD: Accelerating Large-scale Unsupervised Heterogeneous Outlier Detection (Acceleration)	2021	pyod.models.suod.SUOD	[AZHC+2
Combination	Average	Simple combination by averaging the scores	2015	pyod.models.combination.average()	[AAS15]
Combination	Weighted Average	Simple combination by averaging the scores with detector weights	2015	pyod.models.combination.average()	[AAS15]
Combination	Maximization	Simple combination by taking the maximum scores	2015	pyod.models.combination.maximization()	[AAS15]
Combination	AOM	Average of Maximum	2015	pyod.models.combination.aom()	[AAS15]
Combination	MOA	Maximum of Average	2015	pyod.models.combination.moa()	[AAS15]
Combination	Median	Simple combination by taking the median of the scores	2015	pyod.models.combination.median()	[AAS15]
Combination	majority Vote	Simple combination by taking the majority vote of the labels (weights can be used)	2015	pyod.models.combination.majority_vote()	[AAS15]

(iii) Utility Functions:

Type	Name	Function
Data	pyod.utils.data.generate_data()	Synthesized data generation; normal data is generated by a multivariate Gaussian and outliers are generated by a uniform distribution
Data	pyod.utils.data.generate_data_clusters()	Synthesized data generation in clusters; more complex data patterns can be created with multiple clusters
Stat	pyod.utils.stat_models.wpearsonr()	Calculate the weighted Pearson correlation of two samples
Utility	pyod.utils.utility.get_label_n()	Turn raw outlier scores into binary labels by assign 1 to top n outlier scores
Utility	pyod.utils.utility.precision_n_scores()	calculate precision @ rank n

Outlier Detection Using PyOD

```
!pip install pyod
from pyod.utils.data import generate_data, get_outliers_inliers
from pyod.models.pca import PCA
from pyod.utils.data import evaluate_print
from pyod.utils.example import visualize
```

```
Collecting pyod
  Downloading pyod-2.0.2.tar.gz (165 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 165.8/165.8 kB 1.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from pyod) (1.4.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from pyod) (3.7.1)
Requirement already satisfied: numpy>=1.19 in /usr/local/lib/python3.10/dist-packages (from pyod) (1.26.4)
```

```
Requirement already satisfied: numba>=0.51 in /usr/local/lib/python3.10/dist-packages (from pyod) (0.60.0)
Requirement already satisfied: scipy>=1.5.1 in /usr/local/lib/python3.10/dist-packages (from pyod) (1.13.1)
Requirement already satisfied: scikit-learn>=0.22.0 in /usr/local/lib/python3.10/dist-packages (from pyod) (1.3.2)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba>=0.51->pyod) (0.43.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22.0->pyod) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyod) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyod) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyod) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyod) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyod) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyod) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyod) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->pyod) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib->pyod) (1.16.0
Building wheels for collected packages: pyod
  Building wheel for pyod (setup.py) ... done
  Created wheel for pyod: filename=pyod-2.0.2-py3-none-any.whl size=198469 sha256=e21f29f4fe27c2b6a3064c88c8c7b6fc9262c18134be9ab146ac24
  Stored in directory: /root/.cache/pip/wheels/77/c2/20/34d1f15b41b701ba69f42a32304825810d680754d509f91391
Successfully built pyod
Installing collected packages: pyod
Successfully installed pyod-2.0.2
```

```
# Start by creating a dataset using generate_data() from pyod
X_train, y_train = generate_data(train_only=True)
# Create dataframe from Pandas using the generated data
df_train = pd.DataFrame(X_train)
df_train['y'] = y_train

# Display first few rows
df_train.head()
```

	0	1	y
0	5.059895	5.061739	0.0
1	3.797757	6.284863	0.0
2	4.753695	4.887503	0.0
3	4.853710	3.510936	0.0
4	4.925893	6.155376	0.0

Treatment

1. Remove
2. Replace :
    - through various imputation methods, such as using the `mean`, `median`, `mode`, or `constant value`
    - techniques like `nearest neighbors`, `linear regression`, or `machine learning models` can be employed to predict values based on other features or variables in the dataset.
3. Transform
    - transforming outliers using mathematical functions or techniques alter the scale or shape of the data.
    - Transformations such as `logarithmic, square root, and inverse` can reduce skewness or asymmetry, making the data more normal or symmetrical.
    - `Standardization, normalization, and scaling` can change the range or magnitude of the data, facilitating comparability or consistency.

1 Remove/Trim/Delete

- completely remove outliers detected in the previous steps
    - IQR method
    - Z-score method
    - standard deviation method

```
# Trimming the outliers from the sample data
for i in sample_outliers:
```

```
a = np.delete(sample, np.where(np.atleast_1d(sample) == i))
print(a) # print(len(sample), len(a))
```

```
[15 18  7 13 16 11 21  5 15 10  9]
```

1.1 Using IQR Method

- Scores not in the range of $(Q_1 - 1.5\ IQR)$ and $(Q_3 + 1.5\ IQR)$ are outliers, and can be removed.
- The IQR method is based on quartiles and works well for `skewed distributions`.

```
dfnum_out = dfnum[~((dfnum.lt(Q1-1.5 * IQR))|(dfnum.gt (Q3 + 1.5 * IQR))).any(axis=1)]
dfnum_out.head(3)
dfnum_out.shape
```

	carat	depth	table	price	x	y	z
0	0.23	61.5	55.0	326	3.95	3.98	2.43
1	0.21	59.8	61.0	326	3.89	3.84	2.31
3	0.29	62.4	58.0	334	4.20	4.23	2.63

```
(47524, 7)
```

To get the list of outliers deleted above

```
# Calculate the outliers mask
outliers_mask = (dfnum.lt(Q1 - 1.5 * IQR)) | (dfnum.gt(Q3 + 1.5 * IQR))
outliers_mask.sample(10)  # True represents outlier
```

	carat	depth	table	price	x	y	z
53320	False	False	False	False	False	False	False
38787	False	False	False	False	False	False	False
26240	True	False	False	True	False	False	False
26328	True	False	False	True	False	False	False
52113	False	False	False	False	False	False	False
13637	False	False	False	False	False	False	False
8829	False	False	False	False	False	False	False
39194	False	False	False	False	False	False	False
29938	False	False	False	False	False	False	False
30398	False	False	False	False	False	False	False

```
# Extract the outliers (rows with outliers)
outliers = dfnum[outliers_mask]

# Display the outliers
print(outliers)
```

```
       carat  depth  table  price   x   y   z
0        NaN    NaN    NaN    NaN NaN NaN NaN
1        NaN    NaN    NaN    NaN NaN NaN NaN
2        NaN   56.9   65.0    NaN NaN NaN NaN
3        NaN    NaN    NaN    NaN NaN NaN NaN
4        NaN    NaN    NaN    NaN NaN NaN NaN
...      ...    ...    ...    ...  ..  ..  ..
53935    NaN    NaN    NaN    NaN NaN NaN NaN
53936    NaN    NaN    NaN    NaN NaN NaN NaN
53937    NaN    NaN    NaN    NaN NaN NaN NaN
53938    NaN    NaN    NaN    NaN NaN NaN NaN
53939    NaN    NaN    NaN    NaN NaN NaN NaN

[53940 rows x 7 columns]
```

```
# Sum the number of outliers for each column
outliers_count_columnwise =outliers_mask.sum(axis=0)
outliers_count_columnwise
```

	0
carat	1889
depth	2545
table	605
price	3540
x	32
y	29
z	49

dtype: int64

1.2 Using Z-Score Method

- The Z-score method standardizes the data and removes values that are too far from the mean (typically with a Z-score threshold like 3).
- The Z-score method is suitable for `normally distributed` datasets.

```
# Dictionary to hold the removed outliers column-wise
removed_outliers = {}

# Apply the detect_outliers_zscore function to each column and store outliers
for col in dfnum.columns:
    outliers = detect_outliers_zscore(dfnum[col])
    removed_outliers[col] = outliers

# Display the removed outliers column-wise
print("Removed outliers column-wise:")
for col, outliers in removed_outliers.items():
    print(f"{col}: {len(outliers)}")
```

```
Removed outliers column-wise:
carat: 439
depth: 685
table: 336
price: 1206
x: 43
y: 34
z: 55
```

1.3 Standard deviation method

```
# Dictionary to hold the removed outliers column-wise
removed_outliers = {}
total_outliers = 0
# Apply the detect_outliers_zscore function to each column and store outliers
for col in dfnum.columns:
    outliers = detect_outliers_stdv(dfnum[col])
    removed_outliers[col] = outliers
    total_outliers += len(outliers)
# Display the removed outliers column-wise
print("Removed outliers column-wise:")
for col, outliers in removed_outliers.items():
    print(f"{col}: {len(outliers)}")

print(f"\n Total number of outliers in the DataFrame: {total_outliers}")
```

```
Removed outliers column-wise:
carat: 439
depth: 685
table: 336
price: 1206
x: 43
y: 34
z: 55

 Total number of outliers in the DataFrame: 2798
```

```
# Calculate the outliers mask
outliers_mask = (dfnum.lt(Q1 - 1.5 * IQR)) | (dfnum.gt(Q3 + 1.5 * IQR))
outliers_mask.sample(10)  # True represents outlier
```

	carat	depth	table	price	x	y	z
15429	False	False	False	False	False	False	False
17804	False	False	False	False	False	False	False
29891	False	False	False	False	False	False	False
7111	False	False	False	False	False	False	False
4690	False	False	False	False	False	False	False
46223	False	False	False	False	False	False	False
43408	False	False	False	False	False	False	False
53324	False	False	False	False	False	False	False
13559	False	False	False	False	False	False	False
35267	False	False	False	False	False	False	False

1.4 Other method

- Identify rows to drop using a specific column as index
- `.index` property retrieves the index labels of rows in `z` column that are ≤ 2.7, or ≥ 4.5.

```
index = dfnum[(dfnum['z'] >= 4.5)|(dfnum['z'] <= 2.7)].index
dfnum.drop(index, inplace=True)
df['z'].describe()
df.z.skew()
```

	z
count	53940.000000
mean	3.538734
std	0.705699
min	0.000000
25%	2.910000
50%	3.530000
75%	4.040000
max	31.800000

dtype: float64

1.522422559068558

2 Replace

2.1 Quantile-based Flooring and Capping (Winsorization)

- do the flooring (e.g., the 10th percentile) for the lower values and
- capping (e.g., the 90th percentile) for the higher values.

```
quantiles_10=dfnum.quantile(.1)
quantiles_90=dfnum.quantile(.9)
# Combine the quantiles into a single DataFrame
quantiles_table = pd.DataFrame({
    '10 Percentile': quantiles_10,
    '90 Percentile': quantiles_90
})

quantiles_table
```

```
dfnum.skew()
```

	10 Percentile	90 Percentile
carat	0.35	1.18
depth	60.00	63.30
table	55.00	60.00
price	755.00	6787.00
x	4.52	6.77
y	4.52	6.76
z	2.79	4.18

	0
carat	0.354060
depth	-0.138462
table	0.800687
price	1.444523
x	0.046926
y	0.039000
z	0.013016

dtype: float64

The variable `price` has more skewness. To remove outliers by caping and then check the skewness reduced.

```
dfnum['price'] = np.where(dfnum['price'] <755, 755,dfnum['price'])
dfnum['price'] = np.where(dfnum['price'] >6787, 6787,dfnum['price'])
dfnum.price.skew()
```

```
0.5705300209243248
```

2.2 Median Imputation

- Replace the extreme values with median values.

```
print(df['table'].quantile(.5))
print(df['table'].quantile(.95))
```

```
57.0
61.0
```

```
df['table'] = np.where(df['table'] > 60, 57, df['table'])
df.describe()
```

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.005732	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	1.647403	3989.439738	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	58.000000	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	60.000000	18823.000000	10.740000	58.900000	31.800000

3 Imputation

Log Transformation

- Transformation of the skewed variables may also help correct the distribution of the variables.
- These could be logarithmic, square root, or square transformations.
- The most common is the logarithmic transformation

```
df["Log_table"] = df["table"].map(lambda i: np.log(i) if i > 0 else 0)
print(df['table'].skew())
print(df['Log_table'].skew())
```

```
-0.07442512975132845
-0.1527412975344145
```

```
!pip install nbconvert
!jupyter nbconvert --to html notebook.ipynb
```

References