Intruksi Tugas

Ubah data yang ada pada superstore.xlsx menjadi data yang memiliki permasalahan seperti:

- Inkonsistensi Data
- Invalid Data
- Data ganda (duplicate)
- Missing Value
- Outlier

Soal:

1. Buatkan Minimal 3 permasalahan pada data superstore.xlsx
2. Save file yg sudah dibuat kotor menjadi superstore-dirty-tugas1.xlsx
3. Perbaiki data dengan berbagai metode yang bisa dilakukan
4. Save file yang sudah diperbaiki menjadi superstore-clean-tugas1.xlsx
5. Buatkan file .pdf berisikan laporan sederhana terkait yang sudah dikerjakan

Format File: NamaGrup-TugasDW1-NamaLengkap.zip

```
# Load libraries
import random
import numpy as np
import pandas as pd

# Load dataset - original
data_ori = pd.read_excel("dataset/Superstores.xlsx")

# Overview Column and Missing Values
data_ori.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9994 non-null   int64
 1   Order ID       9994 non-null   object
 2   Order Date     9994 non-null   datetime64[ns]
 3   Ship Date      9994 non-null   datetime64[ns]
 4   Ship Mode      9994 non-null   object
 5   Customer ID    9994 non-null   object
 6   Customer Name  9994 non-null   object
 7   Segment        9994 non-null   object
 8   Country        9994 non-null   object
 9   City           9994 non-null   object
 10  State          9994 non-null   object
 11  Postal Code    9994 non-null   int64
 12  Region         9994 non-null   object
```

```
 13  Product ID     9994 non-null   object
 14  Category       9994 non-null   object
 15  Sub-Category   9994 non-null   object
 16  Product Name   9994 non-null   object
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
 21  Timestamps     9994 non-null   object
dtypes: datetime64[ns](2), float64(3), int64(3), object(14)
memory usage: 1.7+ MB
```

```
# Check Statistic Information
data_ori.describe()
```

```
            Row ID   Postal Code         Sales     Quantity
Discount  \
count  9994.000000   9994.000000   9994.000000  9994.000000
9994.000000
mean   4997.500000  55190.379428    229.858001     3.789574
0.156203
std    2885.163629  32063.693350    623.245101     2.225110
0.206452
min       1.000000   1040.000000      0.444000     1.000000
0.000000
25%    2499.250000  23223.000000     17.280000     2.000000
0.000000
50%    4997.500000  56430.500000     54.490000     3.000000
0.200000
75%    7495.750000  90008.000000    209.940000     5.000000
0.200000
max    9994.000000  99301.000000  22638.480000    14.000000
0.800000

            Profit
count  9994.000000
mean     28.656896
std     234.260108
min   -6599.978000
25%       1.728750
50%       8.666500
75%      29.364000
max    8399.976000
```

1. Membuat 3 Permasalahan pada Data Original

```
## Create copy dataframe to avoid re-assign variable
data_dirty = data_ori.copy()
```

```
# Overview Column and Missing Values
data_dirty.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9994 non-null   int64
 1   Order ID       9994 non-null   object
 2   Order Date     9994 non-null   datetime64[ns]
 3   Ship Date      9994 non-null   datetime64[ns]
 4   Ship Mode      9994 non-null   object
 5   Customer ID    9994 non-null   object
 6   Customer Name  9994 non-null   object
 7   Segment        9994 non-null   object
 8   Country        9994 non-null   object
 9   City           9994 non-null   object
 10  State          9994 non-null   object
 11  Postal Code    9994 non-null   int64
 12  Region         9994 non-null   object
 13  Product ID     9994 non-null   object
 14  Category       9994 non-null   object
 15  Sub-Category   9994 non-null   object
 16  Product Name   9994 non-null   object
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
 21  Timestamps     9994 non-null   object
dtypes: datetime64[ns](2), float64(3), int64(3), object(14)
memory usage: 1.7+ MB
```

Interpretation:

Based on information above, we could overview total rows in original dataset in 9.994 total rows, has several type of data such as int, object, datetime, and float, data original also spent 1,7 Mb memory size.

```
# 1. Membuat Missing Values pada Random Column dan Rows
## Define total row and column should be contain NaN
n_rows_missing = 50
n_cols_missing = 5

## Apply randomization for index selected based on total row and colum
above
random_idx_rows = np.random.choice(data_ori.index, n_rows_missing,
replace = False)
random_idx_cols = np.random.choice(data_ori.columns, n_cols_missing,
```

```
replace = False)

## Overview sample randomization result
print(f"[INFO] - 5 Sample Index Rows: {random_idx_rows[:5]}")
print(f"[INFO] - 5 Sample Index Cols: {random_idx_cols[:5]}")
```

```
[INFO] - 5 Sample Index Rows: [7400 2423 7147 4910 5015]
[INFO] - 5 Sample Index Cols: ['Profit' 'Sub-Category' 'Order ID'
'City' 'Ship Mode']
```

```
## Apply NaN on selected row and column
for row in random_idx_rows:
    data_dirty.loc[row, random_idx_cols] = np.nan

# Overview Column and Missing Values - After NaN applied
data_dirty.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9994 non-null   int64
 1   Order ID       9944 non-null   object
 2   Order Date     9994 non-null   datetime64[ns]
 3   Ship Date      9994 non-null   datetime64[ns]
 4   Ship Mode      9944 non-null   object
 5   Customer ID    9994 non-null   object
 6   Customer Name  9994 non-null   object
 7   Segment        9994 non-null   object
 8   Country        9994 non-null   object
 9   City           9944 non-null   object
 10  State          9994 non-null   object
 11  Postal Code    9994 non-null   int64
 12  Region         9994 non-null   object
 13  Product ID     9994 non-null   object
 14  Category       9994 non-null   object
 15  Sub-Category   9944 non-null   object
 16  Product Name   9994 non-null   object
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64
 19  Discount       9994 non-null   float64
 20  Profit         9944 non-null   float64
 21  Timestamps     9994 non-null   object
dtypes: datetime64[ns](2), float64(3), int64(3), object(14)
memory usage: 1.7+ MB
```

Interpretation:

Based on information of dataframe above, we could overview dataframe contains Missing Value in several column with total missing value approximately around 50 rows.

```
## Overview "Segment" column distribution - Before Randomization
data_dirty["Segment"].value_counts()
```

```
Consumer       5191
Corporate      3020
Home Office    1783
Name: Segment, dtype: int64
```

Interpretation:

Based on the provided information, it seems to be a distribution of customers among different segments. The data appears to represent the number of customers in each segment, with three distinct segments: Consumer, Corporate, and Home Office.

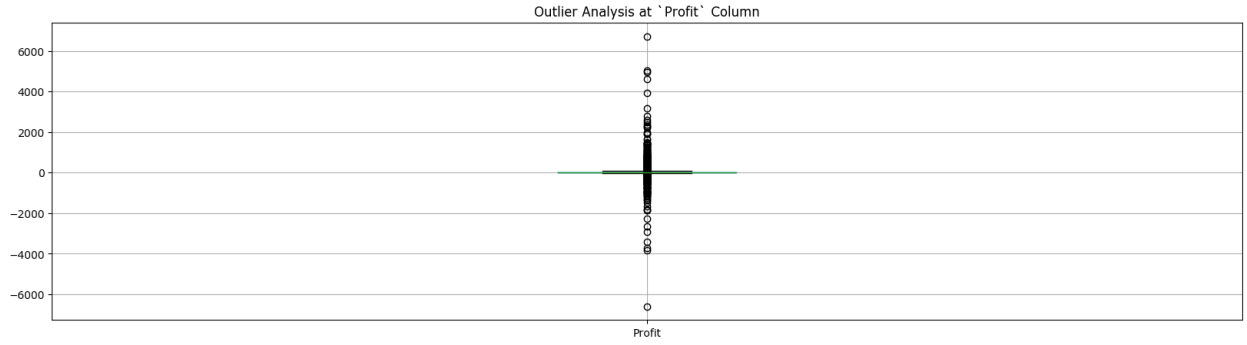
```
# 2. Mengubah format data pada kolom "Segment"
## Apply new "variable" on selected row in "Segment" column
for row in random_idx_rows:
    data_dirty.loc[row, "Segment"] = "Rich Customer"

## Overview "Segment" column distribution - After Randomization
data_dirty["Segment"].value_counts()
```

```
Consumer         5167
Corporate        3003
Home Office      1774
Rich Customer      50
Name: Segment, dtype: int64
```

Interpretation:

Based on the provided information, it seems to be a distribution of customers among different segments. The data appears to represent the number of customers in each segment, with three distinct segments: Consumer, Corporate, Home Office, and Rich Customer.

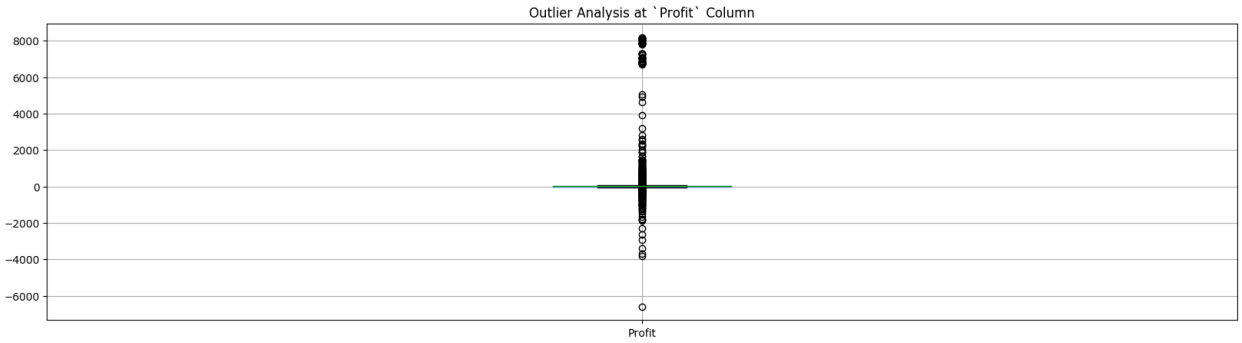
```
data_dirty.boxplot(column = ["Profit"], figsize = (20,
5)).set_title("Outlier Analysis at `Profit` Column");
```



Interpretation:

Based on figure above we could understand "Profit" column has large of rows indicated as outlier, with additional information minimum profit is **-6k** and maximum profit is approximate around **6k++**

```
## Apply randomization for index selected based on total rows
random_idx_rows = np.random.choice(data_ori.index, n_rows_missing,
replace = False)

## 3. Meningkatkan maximum value di kolom profit
## Apply new upper bound based on this formula
### (max_value + current_value)
for row in random_idx_rows:
    current_value = data_dirty.loc[row, 'Profit']
    data_dirty.loc[row, 'Profit'] = data_dirty['Profit'].max() +
current_value

data_dirty.boxplot(column = ["Profit"], figsize = (20,
5)).set_title("Outlier Analysis at `Profit` Column");
```



Interpretation:

Based on figure above we could understand "Profit" column has large of rows indicated as outlier, with additional information minimum profit is **-6k** and maximum profit is approximate around **8k++**

```
# Save to Excel - dirty
data_dirty.to_excel("dataset/superstore-dirty-tugas1.xlsx", index =
False)
```

2. Memperbaiki 3 Permasalahan pada Data Original

```
# Load libraries
import matplotlib.pyplot as plt

# Setup pandas display
pd.set_option("display.max_columns", None)

## Copy dirty dataframe
data_clean = data_dirty.copy()
```

```
## Masalah 1: Membuat Missing Values pada Random Column dan Rows
## Solusi 1: Imputasi dgn mean (pada tipe data numerikal) dan mode
(pada tipe data kategorikal)

## Overview which column contain NaN - Before Solve Problem
df_1 = pd.DataFrame(data_clean.isna().sum()).reset_index()
df_1.columns = ['Column', 'Count NaN']
df_1 = df_1[df_1['Count NaN'] != 0].reset_index(drop = True)
df_1['Type Data'] = df_1['Column'].apply(lambda col:
data_clean[col].dtype)
df_1
```

```
         Column  Count NaN Type Data
0      Order ID         50    object
1     Ship Mode         50    object
2          City         50    object
3  Sub-Category         50    object
4        Profit         50   float64
```

```
## Apply as mentioned
for column, column_type in zip(df_1['Column'], df_1['Type Data']):
    if column == "Order ID":
        continue
    elif column_type == "object":
        imputation = data_clean[column].mode().values[0]
    else:
        imputation = data_clean[column].mean()
    data_clean[column] = data_clean[column].fillna(imputation)

## Overview which column contain NaN - After Solve Problem
df_1 = pd.DataFrame(data_clean.isna().sum()).reset_index()
df_1.columns = ['Column', 'Count NaN']
df_1 = df_1[df_1['Count NaN'] != 0].reset_index(drop = True)
df_1['Type Data'] = df_1['Column'].apply(lambda col:
data_clean[col].dtype)
df_1
```

```
     Column  Count NaN Type Data
0  Order ID         50    object
```

```
# NOTE: Order ID should be unique, means we need to find duplicate
pattern based on selected column
## Strategy 1: Imputation with similar "Order Date", "Ship Date", and
"Customer ID"
indices_OrderId_NaN = data_clean[data_clean['Order ID'].isna()].index
for idx in indices_OrderId_NaN:
    try:
        value = data_clean[
            (data_clean['Order Date'] == data_clean.loc[idx, 'Order
Date']) & \
            (data_clean['Ship Date'] == data_clean.loc[idx, 'Ship
```

```
Date']) & \
            (data_clean['Customer ID'] == data_clean.loc[idx,
'Customer ID'])
        ].dropna().reset_index(drop = True)['Order ID'][0]
        data_clean.loc[idx, 'Order ID'] = value
    # NOTE: Strategy 1 not implemented due there is not similar row
    except (ValueError, KeyError):
        pass

## Overview which column contain NaN - After Implement Strategy 1
df_1 = pd.DataFrame(data_clean.isna().sum()).reset_index()
df_1.columns = ['Column', 'Count NaN']
df_1 = df_1[df_1['Count NaN'] != 0].reset_index(drop = True)
df_1['Type Data'] = df_1['Column'].apply(lambda col:
data_clean[col].dtype)
df_1
```

```
     Column  Count NaN Type Data
0  Order ID         13    object
```

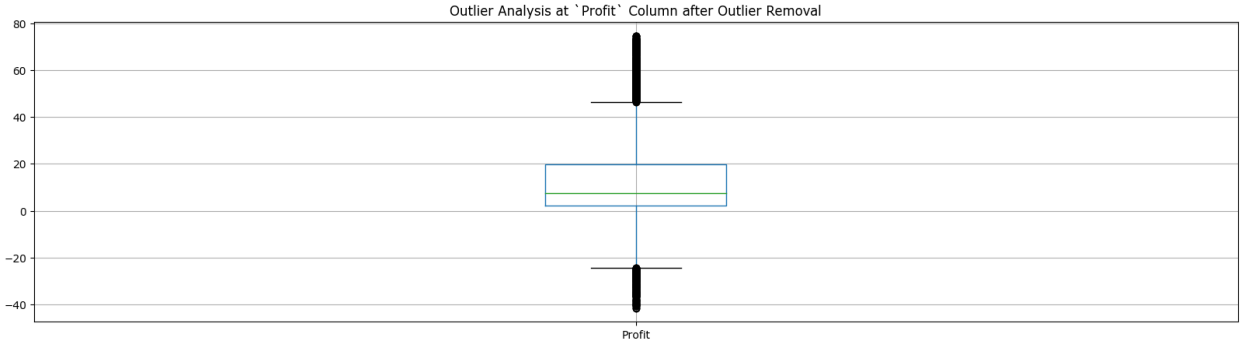
```
## Masalah 2: Mengubah format data pada kolom "Segment"
## Solusi 2: Normalisasi kategori dengan meninjau distribusi data
terdekat.

## Overview segment categories - before treating
pd.DataFrame(data_clean['Segment'].value_counts())
```

```
                Segment
Consumer           5167
Corporate          3003
Home Office        1774
Rich Customer        50
```

```
# NOTE: Usually customer segmentation has single category per
"Customer ID" or "Customer Name"
## Apply similar "Customer Name" with Original categories outside
"Rich Customer"
indices_anomaly_segment = data_clean[data_clean['Segment'] == "Rich
Customer"].index
for idx in indices_anomaly_segment:
    value = data_clean[
        (data_clean['Customer Name'] == data_clean.loc[idx, 'Customer
Name']) & \
        (data_clean['Segment'] != "Rich Customer")
    ]["Segment"].values[0]
    data_clean.loc[idx, "Segment"] = value

## Overview segment categories - after treating
pd.DataFrame(data_clean['Segment'].value_counts())
```

```
             Segment
Consumer        5191
Corporate       3020
Home Office     1783
```

```
## Masalah 3: Meningkatkan maximum value di kolom profit
## Solusi 3: Remove Outlier - decision was made according figure
boxplot above

# NOTE:
## Method Used: IQR
def remove_outliers_iqr(df, column_name, lower_bound=0.25,
upper_bound=0.75):
    """
    Remove outliers from a specific column using the Interquartile
Range (IQR) method.

    Parameters:
    - df: DataFrame
    - column_name: str
        Name of the column in which outliers should be removed.
    - lower_bound: float
        Lower bound of the IQR. Default is 0.25.
    - upper_bound: float
        Upper bound of the IQR. Default is 0.75.

    Returns:
    - DataFrame
        DataFrame with outliers removed from the specified column.
    """
    q1 = df[column_name].quantile(lower_bound)
    q3 = df[column_name].quantile(upper_bound)
    iqr = q3 - q1

    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr

    df_filtered = df[(df[column_name] >= lower_bound) &
(df[column_name] <= upper_bound)]

    return df_filtered

data_clean = remove_outliers_iqr(data_clean,
"Profit").reset_index(drop = True)

data_clean.boxplot(
    column = ["Profit"],
    figsize = (20, 5)
).set_title("Outlier Analysis at `Profit` Column after Outlier
Removal");
```




```
# Save to Excel - clean
data_clean.to_excel("dataset/superstore-clean-tugas1.xlsx", index =
False)
```