Dealing with Missing Values in Dataset

Mousa Tayseer Jafar

Deakin University - Australia

2022-08-07

Overview

Problem Definition

Defn

What is a Missing Value?

- Missing data is defined as the values or data that is not stored (or not present) for some variables in the given dataset.
- Missing value can bias the results of the machine learning models and reduce the accuracy of the model.
- Below is a sample of the missing data from the Tabular Playground Series - June 2022 dataset.

In [5]:
```
df=pd.read_csv('data.csv')
df.head(15)
```

Out[5]:

	row_id	F_1_0	F_1_1	F_1_2	F_1_3	F_1_4	F_1_5	F_1_6	F_1_7	F_1_8	...	F_4_5	F_4_6	F_4_7	F_4_8	F_4_9
0	0	-0.354591	-0.464038	2.304115	0.734486	1.696395	0.136285	-0.518344	0.502640	-1.852504	...	3.744152	0.794438	0.265185	-0.561809	0.196480
1	1	1.380940	-0.499626	-0.418548	1.911725	-0.826130	-1.715371	-0.577091	-1.041486	0.596067	...	-2.895826	-0.738275	2.361818	-0.060753	0.727249
2	2	0.256023	-1.059874	NaN	0.345678	1.513814	1.243864	-0.509648	-0.800481	-0.115945	...	2.252834	0.472496	2.491386	0.353381	-0.260682
3	3	-0.728420	-2.432399	-2.453602	-0.020509	0.333397	0.086049	-1.787601	0.667011	0.761564	...	2.004600	-4.664806	-0.847211	-0.264249	0.664334
4	4	0.590212	-0.066127	0.468009	-1.096038	0.119399	-1.809710	0.466358	-0.053196	-0.580320	...	0.976937	2.558883	3.377724	0.846891	0.696032
5	5	0.088609	NaN	0.270489	0.251012	0.280627	-1.705229	0.320000	-0.263370	2.066337	...	-0.359761	1.740050	1.927704	-0.082221	-0.548425
6	6	0.533210	1.662285	0.601138	-0.662212	0.033856	-0.056208	-1.781594	0.326768	1.583396	...	-2.149355	4.226621	-1.136903	0.171289	0.703419
7	7	-1.737332	-1.009690	0.294162	1.002915	0.700551	-1.071100	2.044173	-0.545110	-0.950346	...	3.150719	1.134666	3.063516	0.046446	0.790186
8	8	-0.334935	-0.801940	-0.754364	-0.369206	0.781484	1.024585	-0.394121	-0.588350	0.092130	...	1.075616	2.535868	-0.617469	0.107925	-1.063796
9	9	0.144170	-0.580150	-0.896780	-0.550178	1.231237	-0.474318	-1.230894	0.474766	0.367473	...	-1.630001	1.543487	0.823975	0.036584	-1.029416
10	10	-0.357771	1.619492	0.241737	-0.931236	-0.439494	0.544001	-0.276768	0.299616	0.031999	...	-1.950920	2.514465	0.133249	-0.287813	0.114724
11	11	0.696625	0.033878	-0.552245	-0.508829	1.326049	1.225479	0.846370	0.576243	-0.598348	...	-0.446187	2.841221	7.281518	-0.093614	-0.008082
12	12	-1.086734	0.186637	-0.226690	-1.586084	-0.855886	-0.234057	1.269481	0.816826	-0.242076	...	-2.315479	1.522740	-1.669253	0.705616	0.633991
13	13	0.322598	1.323594	1.655550	0.790347	0.457020	-0.536445	0.777910	-0.298853	-2.251969	...	1.135202	1.809612	-0.424931	0.224817	-0.845311
14	14	0.745005	0.566602	0.017445	1.927276	0.285437	0.906739	0.462676	0.782369	-1.389328	...	-3.328979	-5.713877	-1.104388	0.517917	-0.221236

15 rows × 81 columns

Figure 1: Missing vlaues

Definition

How is Missing Value Represented In The Dataset? In Pandas, usually, missing values are represented by **NaN**.

In [5]:
```
df=pd.read_csv('data.csv')
df.head(15)
```

Out[5]:

	row_id	F_1_0	F_1_1	F_1_2	F_1_3	F_1_4	F_1_5	F_1_6	F_1_7	F_1_8	...	F_4_5	F_4_6	F_4_7	F_4_8	F_4_9
0	0	-0.354591	-0.464038	2.304115	0.734486	1.696395	0.136285	-0.518344	0.502640	-1.852504	...	3.744152	0.794438	0.265185	-0.561809	0.196480
1	1	1.380940	-0.499626	-0.418548	1.911725	-0.826130	-1.715371	-0.577091	-1.041486	0.596067	...	-2.895826	-0.738275	2.361818	-0.060753	0.727249
2	2	0.256023	-1.059874	NaN	0.345678	1.513814	1.243864	-0.509648	-0.800481	-0.115945	...	2.252834	0.472496	2.491386	0.353381	-0.260682
3	3	-0.728420	-2.432399	-2.453602	-0.020509	0.333397	0.086049	-1.787601	0.667011	0.761564	...	2.004600	-4.664806	-0.847211	-0.264249	0.664334
4	4	0.590212	-0.066127	0.468009	-1.096038	0.119399	-1.809710	0.466358	-0.053196	-0.580320	...	0.976937	2.558883	3.377724	0.846891	0.696032
5	5	0.088609	NaN	0.270489	0.251012	0.280627	-1.705229	0.320000	-0.263370	2.066337	...	-0.359761	1.740050	1.927704	-0.082221	-0.548425
6	6	0.533210	1.662285	0.601138	-0.662212	0.033856	-0.056208	-1.781594	0.326768	1.583396	...	-2.149355	4.226621	-1.136903	0.171289	0.703419
7	7	-1.737332	-1.009690	0.294162	1.002915	0.700551	-1.071100	2.044173	-0.545110	-0.950346	...	3.150719	1.134666	3.063516	0.046446	0.790186
8	8	-0.334935	-0.801940	-0.754364	-0.369206	0.781484	1.024585	-0.394121	-0.588350	0.092130	...	1.075616	2.535868	-0.617469	0.107925	-1.063796
9	9	0.144170	-0.580150	-0.896780	-0.550178	1.231237	-0.474318	-1.230894	0.474766	0.367473	...	-1.630001	1.543487	0.823975	0.036584	-1.029416
10	10	-0.357771	1.619492	0.241737	-0.931236	-0.439494	0.544001	-0.276768	0.299616	0.031999	...	-1.950920	2.514465	0.133249	-0.287813	0.114724
11	11	0.696625	0.033878	-0.552245	-0.508829	1.326049	1.225479	0.846370	0.576243	-0.598348	...	-0.446187	2.841221	7.281518	-0.093614	-0.008082
12	12	-1.086734	0.186637	-0.226690	-1.586084	-0.855886	-0.234057	1.269481	0.816826	-0.242076	...	-2.315479	1.522740	-1.669253	0.705616	0.633991
13	13	0.322598	1.323594	1.655550	0.790347	0.457020	-0.536445	0.777910	-0.298853	-2.251969	...	1.135202	1.809612	-0.424931	0.224817	-0.845311
14	14	0.745005	0.566602	0.017445	1.927276	0.285437	0.906739	0.462676	0.782369	-1.389328	...	-3.328979	-5.713877	-1.104388	0.517917	-0.221236

15 rows × 81 columns

Figure 2: in each Row

In [7]:
```
df.isnull()
```

Out[7]:

	row_id	F_1_0	F_1_1	F_1_2	F_1_3	F_1_4	F_1_5	F_1_6	F_1_7	F_1_8	...	F_4_5	F_4_6	F_4_7	F_4_8	F_4_9	F_4_10	F_4_11	F_4_12	F_4_13	F_4_1
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fals
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fals
2	False	False	False	True	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fals
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fals
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fals
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
999995	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fals
999996	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fals
999997	False	False	False	False	False	False	False	False	False	False	...	False	False	False	True	False	False	False	False	False	Fals
999998	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fals
999999	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	Fals

1000000 rows × 81 columns

Figure 3: in all dataset

Why Do We Need To Care About Handling Missing Data?

- The real-world data often has a lot of missing values.
- The cause of missing values can be data corruption or failure to record data.
- The handling of missing data is very important during the preprocessing of the dataset as many machine learning algorithms do not support missing values.
- Three problems associated with missing values:
  - Loss of efficiency,
  - Complications in handling and analyzing the data,
  - Bias resulting from differences between missing and complete data.

Figure 4: Types Of Missing Values

Data Processing

Loading Data

In [1]:
```
import pandas as pd
```

In [2]:
```
df=pd.read_csv('data.csv')
df.head()
```

Out[2]:

	row_id	F_1_0	F_1_1	F_1_2	F_1_3	F_1_4	F_1_5	F_1_6	F_1_7	F_1_8	...	F_4_5	F_4_6	F_4_7	F_4_8
0	0	-0.354591	-0.464038	2.304115	0.734486	1.696395	0.136285	-0.518344	0.502640	-1.852504	...	3.744152	0.794438	0.265185	-0.561809
1	1	1.380940	-0.499626	-0.418548	1.911725	-0.826130	-1.715371	-0.577091	-1.041486	0.596067	...	-2.895826	-0.738275	2.361818	-0.060753
2	2	0.256023	-1.059874	NaN	0.345678	1.513814	1.243864	-0.509648	-0.800481	-0.115945	...	2.252834	0.472496	2.491386	0.353381
3	3	-0.728420	-2.432399	-2.453602	-0.020509	0.333397	0.086049	-1.787601	0.667011	0.761564	...	2.004600	-4.664806	-0.847211	-0.264249
4	4	0.590212	-0.066127	0.468009	-1.096038	0.119399	-1.809710	0.466358	-0.053196	-0.580320	...	0.976937	2.558883	3.377724	0.846891

5 rows × 81 columns

In [3]:
```
df.shape
```

Out[3]:
```
(1000000, 81)
```

Figure 5: Load dataset

- Load the Tabular Playground Series - June 2022 dataset
- import the dataset from csv files

Check for missing values in the dataset


```
In [8]: df.isnull().sum()

Out[8]: row id        0
        F_1_0     18397
        F_1_1     18216
        F_1_2     18008
        F_1_3     18250
                  ...
        F_4_10    18225
        F_4_11    18119
        F_4_12    18306
        F_4_13    17995
        F_4_14    18267
        Length: 81, dtype: int64
```

Figure 6: missing values in Each row

```
In [9]: df.isnull().sum().sum()

Out[9]: 1000000
```

Figure 7: missing values in the dataset

Visualization Missing Values

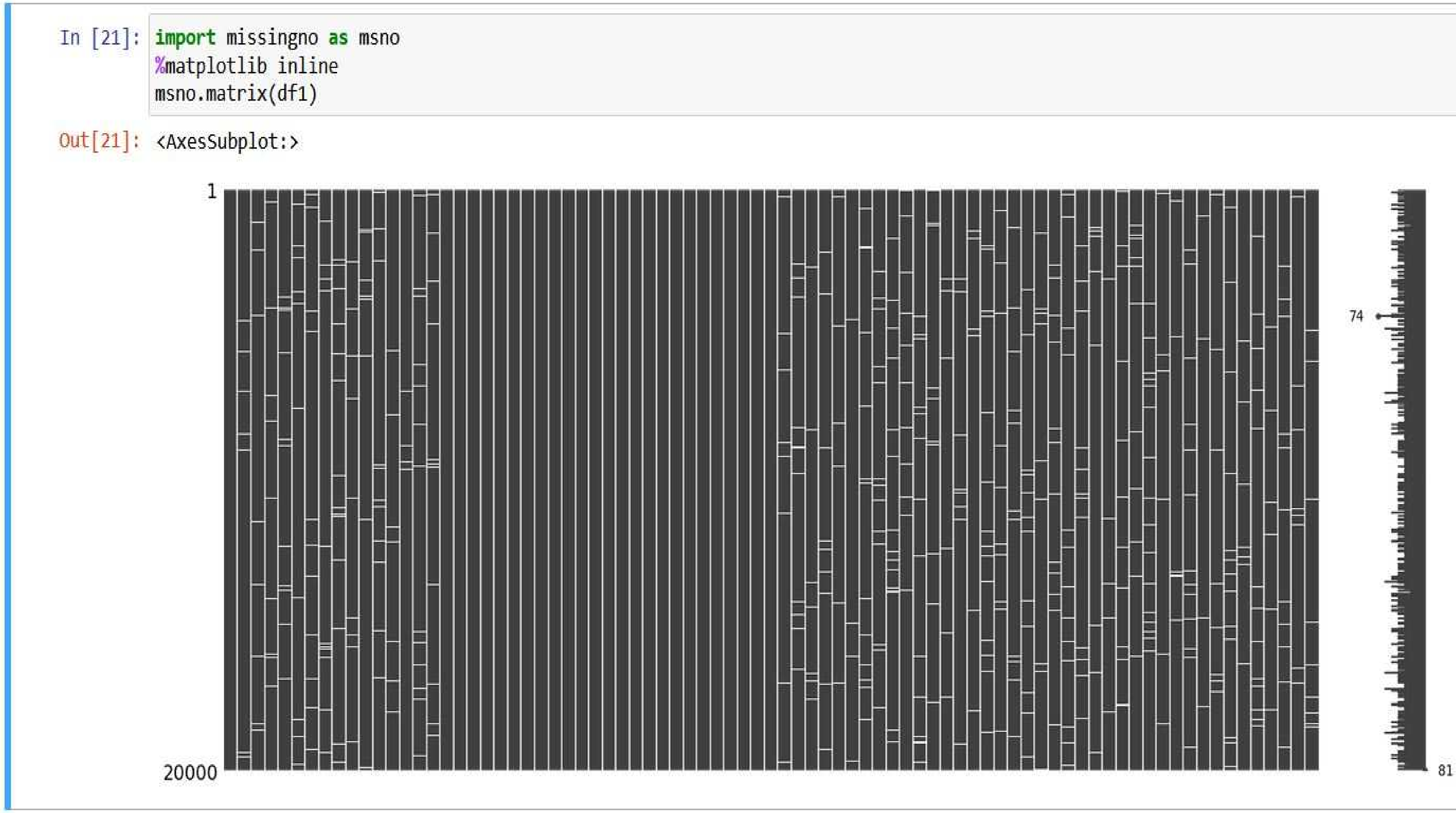


Figure 8: Missing Values

Visualization Missing Values


```
In [46]: df3.isnull().sum().sum()
         import missingno as msno
         %matplotlib inline
         msno.matrix(df3)

Out[46]: <AxesSubplot:>
```

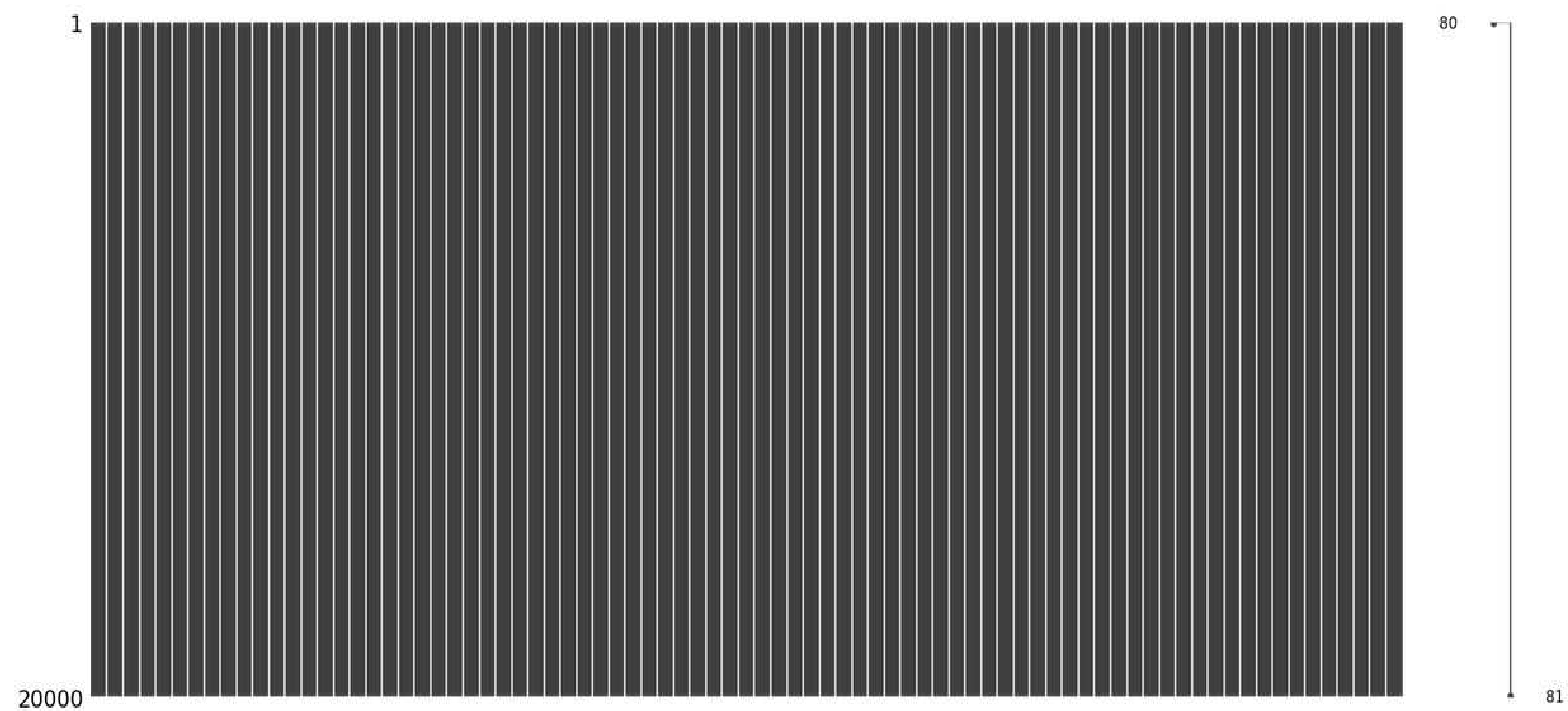


Figure 9: After Removing NAN vlaues

Handle Missing Values

How To Handle Missing Values?

Analyze each column with missing values carefully to understand the reasons behind the missing values, as it is crucial to find out the strategy for handling the missing values.

There are many ways of handling missing values:

- **Deleting Missing values**
- **Replacing With Arbitrary Value (Zero)**
- **Replacing with Previous Value – Forward Fill**
- **Replacing with Next Value – Backward Fill**
- **Interpolation Method**

Selecting Data

- The dataset (Tabular Playground Series - June 2022) has huge data 1000000 Rows , and 81 columns.

```
In [6]: df.shape

Out[6]: (1000000, 81)
```

Figure 10: Tabular Playground Series - June 2022 dataset

- We will select 5% of dataset.

```
In [7]: df1 = df.sample(frac =.05)
        df1.shape

Out[7]: (50000, 81)
```

Figure 11: Selecting 5 % percent of data

Number of Missing Values in Selecting Data


```
In [8]: df1.isnull().sum()

Out[8]: row_id      0
        F_1_0     881
        F_1_1     873
        F_1_2     928
        F_1_3     932
                 ...
        F_4_10    892
        F_4_11    839
        F_4_12    959
        F_4_13    873
        F_4_14    895
        Length: 81, dtype: int64
```

Figure 12: The Number of missing values for each row

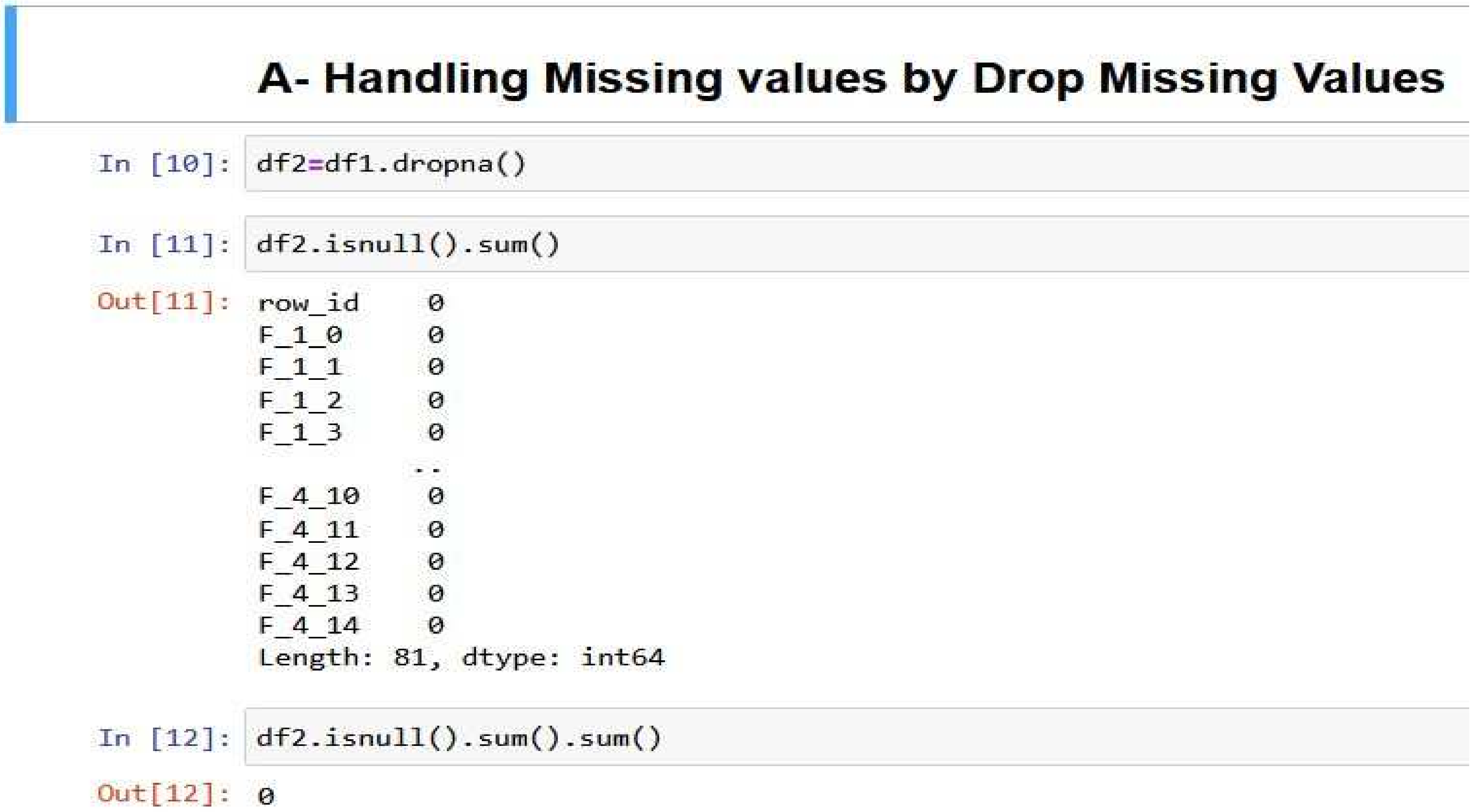
```
In [9]: df1.isnull().sum().sum()

Out[9]: 50064
```

Figure 13: The total of missing values in Selecting Data

A-Deleting the Missing values

Missing values can be handled by deleting rows or columns that have null values.



A- Handling Missing values by Drop Missing Values

```
In [10]: df2=df1.dropna()

In [11]: df2.isnull().sum()

Out[11]: row_id    0
         F_1_0     0
         F_1_1     0
         F_1_2     0
         F_1_3     0
                  ..
         F_4_10    0
         F_4_11    0
         F_4_12    0
         F_4_13    0
         F_4_14    0
         Length: 81, dtype: int64

In [12]: df2.isnull().sum().sum()

Out[12]: 0
```

Figure 14: Drop Missing values

B-Replacing With Arbitrary Value (Zero)

We will replace the missing values with some arbitrary value using the following code.

B- Handling Missing values by Replacing with Zero

```
In [16]: df11=df1.fillna(0)

In [17]: df2.isnull().sum()

Out[17]: row_id    0
         F_1_0     0
         F_1_1     0
         F_1_2     0
         F_1_3     0
                  ..
         F_4_10    0
         F_4_11    0
         F_4_12    0
         F_4_13    0
         F_4_14    0
         Length: 81, dtype: int64

In [18]: df11.isnull().sum().sum()

Out[18]: 0
```

Figure 15: Replace missing values with ‘0’.

C-Replacing with Previous Value – Forward Fill

The missing value is imputed using the previous value and imputing the values with the previous value is more appropriate.

C-Handling Missing values by Replacing with Previous Value – Forward Fill

```
In [22]: df4=df1.fillna(method='pad')

In [23]: df4.isnull().sum()

Out[23]: row_id    0
         F_1_0     0
         F_1_1     0
         F_1_2     0
         F_1_3     0
                  ..
         F_4_10    0
         F_4_11    0
         F_4_12    0
         F_4_13    0
         F_4_14    0
         Length: 81, dtype: int64

In [24]: df4.isnull().sum().sum()

Out[24]: 0
```

Figure 16: Replacing with previous value – Forward fill.

D-Replacing with Next Value – Backward Fill

In backward fill, the missing value is imputed using the next value.

D-Handling Missing values by Replacing with Next Value – Backward Fill

```
In [27]: df5=df1.fillna(method='bfill')

In [28]: df5.isnull().sum()

Out[28]: row_id    0
         F_1_0     0
         F_1_1     0
         F_1_2     0
         F_1_3     0
                  ..
         F_4_10    0
         F_4_11    0
         F_4_12    0
         F_4_13    0
         F_4_14    0
         Length: 81, dtype: int64

In [29]: df5.isnull().sum().sum()

Out[29]: 0
```

Figure 17: We are replacing the missing values with next value.

E-Interpolation Method

Missing values can also be imputed using interpolation. Pandas interpolate method can be used to replace the missing values with different interpolation methods like 'polynomial', 'linear', 'quadratic'. Default method is 'linear'.

E-Handling Missing values by Interpolate Method

```
In [19]: df3=df1.interpolate(method='linear')
```

```
In [20]: df3.isnull().sum()
```

```
Out[20]: row_id    0
         F_1_0     0
         F_1_1     0
         F_1_2     0
         F_1_3     0
                  ..
         F_4_10    0
         F_4_11    0
         F_4_12    0
         F_4_13    0
         F_4_14    0
         Length: 81, dtype: int64
```

```
In [21]: df3.isnull().sum().sum()
```

```
Out[21]: 0
```

Figure 18: We are replacing the missing values with linear method.

Build The Models

1- Linear Regression Model

Linear Regression for Interpolate Method

```
In [66]: from sklearn.linear_model import LinearRegression
         Lin=LinearRegression()
         Lin.fit(X_train,y_train)

Out[66]: LinearRegression()

In [67]: print('The accuracy of LinearRegression: ',Lin.score(X_train,y_train))

         The accuracy of LinearRegression:  0.6587766661021908
```

Figure 19: Interpolate Method

Linear Regression for Replacing with Previous Value – Forward Fill

```
In [96]: from sklearn.linear_model import LinearRegression
         Lin=LinearRegression()
         Lin.fit(X_train,y_train)

Out[96]: LinearRegression()

In [97]: print('The accuracy of LinearRegression: ',Lin.score(X_train,y_train))

         The accuracy of LinearRegression:  0.6336116694335745
```

Figure 20: Replacing with Previous Value – Forward Fill

2- Decision Tree Model

Decision Tree for Interpolate Method

```
In [62]: from sklearn.tree import DecisionTreeClassifier
         tree=DecisionTreeClassifier(criterion='entropy',random_state=0)
         tree.fit(X_train,y_train)
Out[62]: DecisionTreeClassifier(criterion='entropy', random_state=0)

In [63]: print('The accuracy of Decision Tree: ',tree.score(X_train,y_train))
         The accuracy of Decision Tree:  1.0
```

Figure 21: Interpolate Method

Decision Tree for Replacing with Previous Value – Forward Fill

```
In [30]: from sklearn.tree import DecisionTreeClassifier
         tree=DecisionTreeClassifier(criterion='entropy',random_state=0)
         tree.fit(X_train,y_train)
Out[30]: DecisionTreeClassifier(criterion='entropy', random_state=0)

In [31]: print('The accuracy of Decision Tree: ',tree.score(X_train,y_train))
         The accuracy of Decision Tree:  1.0
```

Figure 22: Replacing with Previous Value – Forward Fill

3- Random Forest Model

Random Forest for Interpolate Method

```
In [64]: from sklearn.ensemble import RandomForestClassifier
         forst=RandomForestClassifier(n_estimators=10,criterion='entropy',random_state=0)
         forst.fit(X_train,y_train)

Out[64]: RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)

In [65]: print('The accuracy of Random Forest : ',forst.score(X_train,y_train))

         The accuracy of Random Forest :  0.9994375
```

Figure 23: Interpolate Method

Random Forest for Replacing with Previous Value – Forward Fill

```
In [34]: from sklearn.ensemble import RandomForestClassifier
         forst=RandomForestClassifier(n_estimators=10,criterion='entropy',random_state=0)
         forst.fit(X_train,y_train)

Out[34]: RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)

In [35]: print('The accuracy of Random Forest : ',forst.score(X_train,y_train))

         The accuracy of Random Forest :  0.9993125
```

Figure 24: Replacing with Previous Value – Forward Fill

Conclusion

Evaluating models

Missing data is a common problem in statistical analysis.
Random Forest, Decision Tree, and Linear Regression, were used as Machine Learning Methods to measure the Accuracy after handling the missing values.

Table 1: Accuracy

Type of handling missing Values	#Decision Tree	#Random Forest	#Linear Regression
Drop Missing Values	1.0	0.9994858	0.9047421
Missing Values Zero	1.0	0.9993125	0.7113801
Interpolate Method	1.0	0.9994375	0.6587766
Forward Fill	1.0	0.9993125	0.6336116
Backward Fill	1.0	0.9993125	0.6207112

The presence of missing values in a dataset can affect the performance of a classifier constructed using that dataset as a training sample.
Several methods have been proposed to treat missing data.

The Results

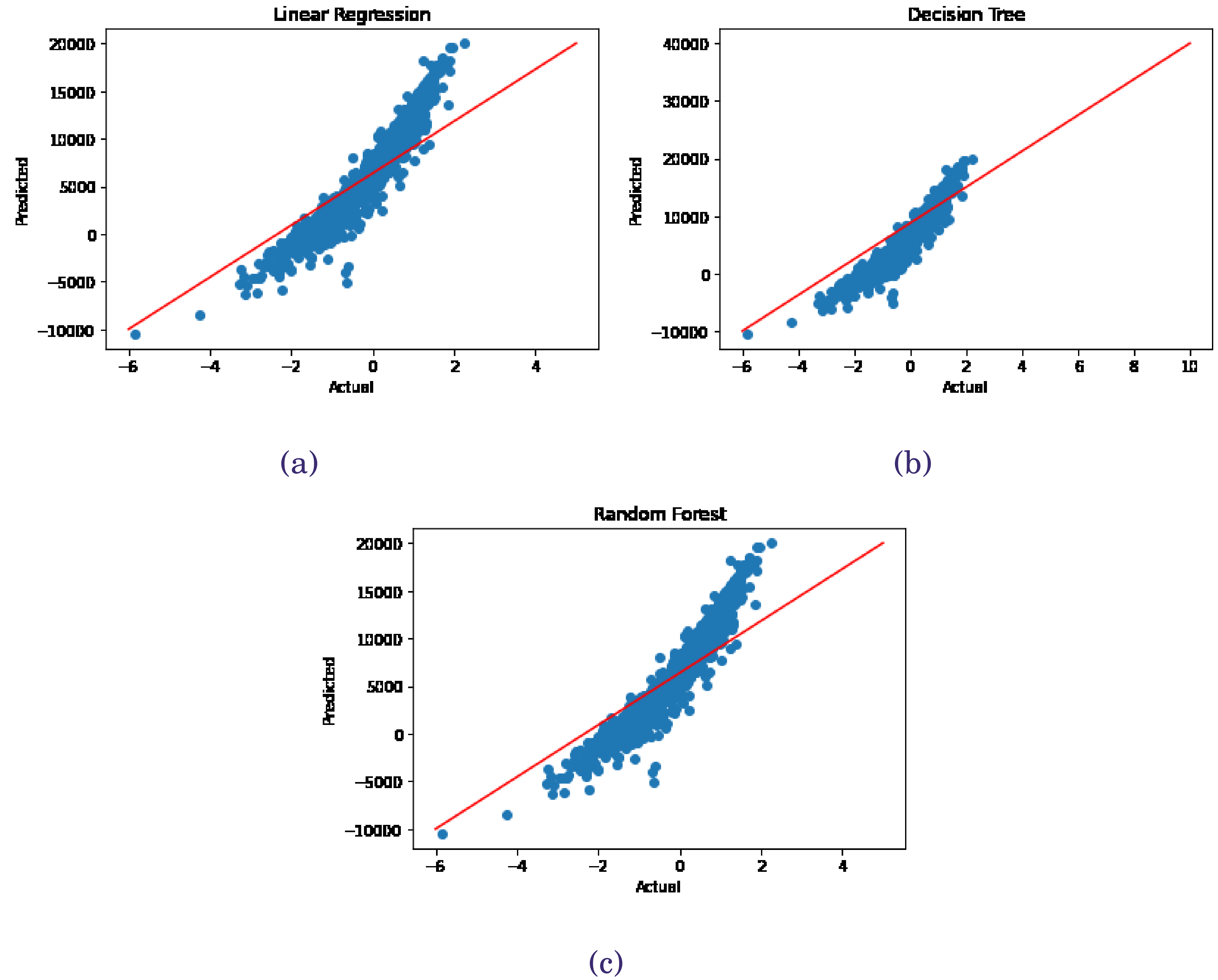


Figure 25: (Predicted for LR,DT,and RF)

Questions?

Thank you- Stay Safe

Contact Information

Mousa Tayseer Jafar

Deakin University, Australia

MUOSAJAFAR@GMAIL.COM

TEAM FOR UNIVERSAL LEARNING AND INTELLIGENT PROCESSING