In [205]:

```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
from sqlalchemy import create_engine
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from statsmodels.tsa.holtwinters import SimpleExpSmoothing # SES
from statsmodels.tsa.holtwinters import Holt # Holts Exponential Smoothing
from statsmodels.tsa.holtwinters import ExponentialSmoothing # Holt Winter's Exponential Smoothing

from sklearn.pipeline import Pipeline
import pickle, joblib
```

In [2]:

```
user = 'root' # user name
pw = '12345' # password
db = 'TMT_Steel_CL_db' # database
# creating engine to connect database
engine = create_engine(f"mysql+pymysql://{user}:{pw}@localhost/{db}")

df = pd.read_csv(r"C:\Users\MsK_PC\360digiTMG\Data_Science\DS_TMT_Steel_Project\Client_data\Dataset.csv")

# dumping data into database
df.to_sql('steel_data', con = engine, if_exists = 'replace', chunksize = 1000, index = False)
```

Out[2]:

33045

In [3]:

```
# loading data from database
sql = 'select * from steel_data'

steel = pd.read_sql_query(sql, con = engine )
steel
```

Out[3]:

	Date	FY	Products	dia	dia group	grade	type	length	Voucher Type	Quantity	Rate	Value
0	04-03-2017	FY 18	25MM TATA TISCON FE500D (S)	25 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	25.49	40000.0	1019600.0
1	04-03-2017	FY 18	08MM TATA TISCON FE500D (S)	08 MM	08 MM	500D	FULL LENGTH	12 METER	Sales	9.09	43200.0	392688.0
2	04-03-2017	FY 18	10MM TATA TISCON FE500D (S)	10 MM	10 MM	500D	FULL LENGTH	12 METER	Sales	3.94	41700.0	164298.0
3	04-03-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.93	41200.0	79516.0
4	04-03-2017	FY 18	16MM TATA TISCON FE500D (S)	16 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.03	41200.0	42436.0
...	...	...	...	...	...	...	...	...	...	...	...	...
33040	3/25/2023	FY 23	20MM TATA TISCON FE500D (T)	20 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	4.08	60000.0	244800.0
33041	3/30/2023	FY 23	08MM TATA TISCON FE550D (T)	08 MM	08 MM	550D	FULL LENGTH	12 METER	Credit Note	-1.07	61000.0	-65270.0
33042	3/30/2023	FY 23	10MM TATA TISCON FE550D (T)	10 MM	10 MM	550D	FULL LENGTH	12 METER	Credit Note	-1.50	60000.0	-90000.0
33043	3/30/2023	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Credit Note	-20.04	59000.0	-1182360.0
33044	3/30/2023	FY 23	20MM TATA TISCON FE500D (T)	20 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Credit Note	-4.60	59000.0	-271400.0

33045 rows × 12 columns

Copy of Original data

""" import pandas as pd import numpy as np import pickle from sqlalchemy import create_engine

user = 'root' # user name pw = '12345' # password db = 'TMT_Steel_CL_db' # database

creating engine to connect database

engine = create_engine(f"mysql+pymysql://{user}:{pw}@localhost/{db}")

df = pd.read_csv(r"C:\Users\MsK_PC\360digiTMG\Data_Science\DS_TMT_Steel_Project\Client_data\Dataset.csv")

dumping data into database

df.to_sql('steel_data1', con = engine, if_exists = 'replace', chunksize = 1000, index = False)"""

In [12]:

```
#steel['dia'].value_counts()
steel['length'].value_counts()
```

Out[12]:

```
length
12 METER         30846
CUSTOMISED        1290
0 METER            615
7 - 10 METER       177
4 - 7 METER         92
10 - 12 METER       25
Name: count, dtype: int64
```

In [13]:

```
steel.dtypes
```

Out[13]:

```
Date             object
FY               object
Products         object
dia              object
dia group        object
grade            object
type             object
length           object
Voucher Type     object
Quantity        float64
Rate            float64
Value           float64
dtype: object
```

In [14]:

```
steel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33045 entries, 0 to 33044
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Date          33045 non-null  object
 1   FY            33045 non-null  object
 2   Products      33045 non-null  object
 3   dia           33045 non-null  object
 4   dia group     33045 non-null  object
 5   grade         33045 non-null  object
 6   type          33045 non-null  object
 7   length        33045 non-null  object
 8   Voucher Type  33045 non-null  object
 9   Quantity      33045 non-null  float64
 10  Rate          33045 non-null  float64
 11  Value         33045 non-null  float64
dtypes: float64(3), object(9)
memory usage: 3.0+ MB
```

In [15]:

```
steel.describe()
```

Out[15]:

	Quantity	Rate	Value
count	33045.000000	33045.000000	3.304500e+04
mean	5.921329	48518.210554	2.839427e+05
std	6.668724	9640.507727	3.312660e+05
min	-32.340000	19590.000000	-2.126355e+06
25%	1.990000	41750.000000	8.814000e+04
50%	3.900000	45700.000000	1.801470e+05
75%	7.010000	56000.000000	3.255000e+05
max	41.680000	83000.000000	2.611980e+06

In [16]:

```
steel.Quantity.mean()
steel.Rate.mean()
steel.Value.mean()
```

Out[16]:

```
283942.715259192
```

In [17]:

```
steel.Quantity.median()
steel.Rate.median()
steel.Value.median()
```

Out[17]:

```
180147.0
```

In [18]:

```
steel.Date.mode()
steel.FY.mode()
steel.Products.mode()
steel.dia.mode()
steel['dia group'].mode()
steel.grade.mode()
steel.type.mode()
steel.length.mode()
steel['Voucher Type'].mode()
steel.Rate.mode()
steel.Value.mode()
```

Out[18]:

```
0    126000.0
Name: Value, dtype: float64
```

In [19]:

```
# Calculate variances for numerical columns using numpy
numerical_data = steel.drop(columns=['Date','FY','Products','dia','dia group','grade','type','length','Voucher Type'])  # Re
variances = np.var(numerical_data, axis=0)

variances
```

Out[19]:

```
Quantity    4.447053e+01
Rate        9.293658e+07
Value       1.097339e+11
dtype: float64
```

In [20]:

```
# checking for skewness
numerical_data.skew()
```

Out[20]:

```
Quantity    2.202827
Rate        0.704654
Value       2.580335
dtype: float64
```

In [21]:

```
# checking for kurtosis
numerical_data.kurt()
```

Out[21]:

```
Quantity    5.002973
Rate       -0.080464
Value       8.178931
dtype: float64
```

In [22]:

```
# Data Preprocessing
# Checking for duplicacies
d= steel.duplicated()
d
```

Out[22]:

```
0        False
1        False
2        False
3        False
4        False
         ...
33040    False
33041    False
33042    False
33043    False
33044    False
Length: 33045, dtype: bool
```

In [23]:

```
sum(d)
```

Out[23]:

```
66
```

In [24]:

```
#Removing Duplicates
d_new = steel.drop_duplicates() # Returns DataFrame with duplicate rows removed.
```

In [25]:

```
d_new
```

Out[25]:

	Date	FY	Products	dia	dia group	grade	type	length	Voucher Type	Quantity	Rate	Value
0	04-03-2017	FY 18	25MM TATA TISCON FE500D (S)	25 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	25.49	40000.0	1019600.0
1	04-03-2017	FY 18	08MM TATA TISCON FE500D (S)	08 MM	08 MM	500D	FULL LENGTH	12 METER	Sales	9.09	43200.0	392688.0
2	04-03-2017	FY 18	10MM TATA TISCON FE500D (S)	10 MM	10 MM	500D	FULL LENGTH	12 METER	Sales	3.94	41700.0	164298.0
3	04-03-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.93	41200.0	79516.0
4	04-03-2017	FY 18	16MM TATA TISCON FE500D (S)	16 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.03	41200.0	42436.0
...	...	...	...	...	...	...	...	...	...	...	...	...
33040	3/25/2023	FY 23	20MM TATA TISCON FE500D (T)	20 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	4.08	60000.0	244800.0
33041	3/30/2023	FY 23	08MM TATA TISCON FE550D (T)	08 MM	08 MM	550D	FULL LENGTH	12 METER	Credit Note	-1.07	61000.0	-65270.0
33042	3/30/2023	FY 23	10MM TATA TISCON FE550D (T)	10 MM	10 MM	550D	FULL LENGTH	12 METER	Credit Note	-1.50	60000.0	-90000.0
33043	3/30/2023	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Credit Note	-20.04	59000.0	-1182360.0
33044	3/30/2023	FY 23	20MM TATA TISCON FE500D (T)	20 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Credit Note	-4.60	59000.0	-271400.0

32979 rows × 12 columns

In [26]:

```
# Segregating data based on their data types
numeric_features = d_new.select_dtypes(include = ['float64']).columns
numeric_features
```

Out[26]:

```
Index(['Quantity', 'Rate', 'Value'], dtype='object')
```

In [27]:

```
categorical_features = d_new.select_dtypes(include = ['object']).columns
categorical_features
```

Out[27]:

```
Index(['Date', 'FY', 'Products', 'dia', 'dia group', 'grade', 'type', 'length',
       'Voucher Type'],
      dtype='object')
```

In [28]:

```
#checking for missing values
d_new.isna().sum()
```

Out[28]:

```
Date            0
FY              0
Products        0
dia             0
dia group       0
grade           0
type            0
length          0
Voucher Type    0
Quantity        0
Rate            0
Value           0
dtype: int64
```

In [29]:

```
# Imputation techniques to handle missing data
# Mode imputation for (categorical) data
cat_pipeline = Pipeline(steps=[('impute1', SimpleImputer(strategy = 'most_frequent'))])

# Mean imputation for Continuous (Float) data
num_pipeline = Pipeline(steps=[('impute2', SimpleImputer(strategy = 'mean'))])
```

In [30]:

```
# 1st Imputation Transformer
preprocessor = ColumnTransformer([
        ('mode', cat_pipeline, categorical_features),
        ('mean', num_pipeline, numeric_features)])

print(preprocessor)

# Fit the data to train imputation pipeline model
impute_data = preprocessor.fit(d_new)

# Save the pipeline
joblib.dump(impute_data, 'impute')
```

```
ColumnTransformer(transformers=[('mode',
                                 Pipeline(steps=[('impute1',
                                                  SimpleImputer(strategy='most_frequent'))]),
                                 Index(['Date', 'FY', 'Products', 'dia', 'dia group', 'grade', 'type', 'lengt
h',
       'Voucher Type'],
      dtype='object')),
                                ('mean',
                                 Pipeline(steps=[('impute2', SimpleImputer())]),
                                 Index(['Quantity', 'Rate', 'Value'], dtype='object'))])
```

Out[30]:

```
['impute']
```

In [ ]:

```
# Transform the original data
#X1 = pd.DataFrame(impute_data.transform(d_new), columns = d_new.columns).convert_dtypes()

#X1.isna().sum()
```

In [31]:

```
d_new
```

Out[31]:

	Date	FY	Products	dia	dia group	grade	type	length	Voucher Type	Quantity	Rate	Value
0	04-03-2017	FY 18	25MM TATA TISCON FE500D (S)	25 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	25.49	40000.0	1019600.0
1	04-03-2017	FY 18	08MM TATA TISCON FE500D (S)	08 MM	08 MM	500D	FULL LENGTH	12 METER	Sales	9.09	43200.0	392688.0
2	04-03-2017	FY 18	10MM TATA TISCON FE500D (S)	10 MM	10 MM	500D	FULL LENGTH	12 METER	Sales	3.94	41700.0	164298.0
3	04-03-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.93	41200.0	79516.0
4	04-03-2017	FY 18	16MM TATA TISCON FE500D (S)	16 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.03	41200.0	42436.0
...	...	...	...	...	...	...	...	...	...	...	...	...
33040	3/25/2023	FY 23	20MM TATA TISCON FE500D (T)	20 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	4.08	60000.0	244800.0
33041	3/30/2023	FY 23	08MM TATA TISCON FE550D (T)	08 MM	08 MM	550D	FULL LENGTH	12 METER	Credit Note	-1.07	61000.0	-65270.0
33042	3/30/2023	FY 23	10MM TATA TISCON FE550D (T)	10 MM	10 MM	550D	FULL LENGTH	12 METER	Credit Note	-1.50	60000.0	-90000.0
33043	3/30/2023	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Credit Note	-20.04	59000.0	-1182360.0
33044	3/30/2023	FY 23	20MM TATA TISCON FE500D (T)	20 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Credit Note	-4.60	59000.0	-271400.0

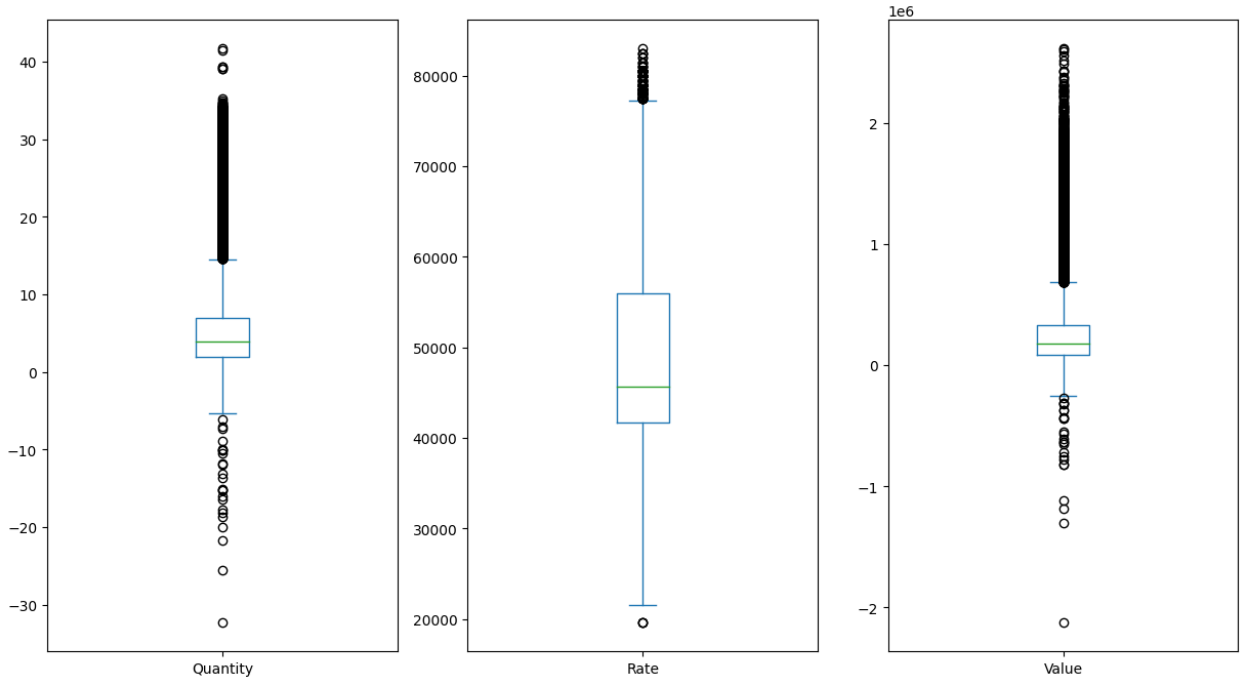
32979 rows × 12 columns

In [32]:

```
# Multiple boxplots in a single visualization.
# Columns with larger scales affect other columns.
# Below code ensures each column gets its own y-axis.

# pandas plot() function with parameters kind = 'box' and subplots = True

d_new.plot(kind = 'box', subplots = True, sharey = False, figsize = (15, 8))

'''sharey True or 'all': x-axis or y-axis will be shared among all subplots.
False or 'none': each subplot x- or y-axis will be independent.'''
```

Out[32]:

```
"sharey True or 'all': x-axis or y-axis will be shared among all subplots.\nFalse or 'none': each subplot x-
or y-axis will be independent."
```



In [33]:

```
# Removing outliers using winsorization technique
from feature_engine.outliers import Winsorizer
```

In [34]:

```
d_new.columns
```

Out[34]:

```
Index(['Date', 'FY', 'Products', 'dia', 'dia group', 'grade', 'type', 'length',
       'Voucher Type', 'Quantity', 'Rate', 'Value'],
      dtype='object')
```

In [35]:

```
winsor = Winsorizer(capping_method = 'iqr', # choose  IQR rule boundaries or gaussian for mean and std
                          tail = 'both', # cap left, right or both tails
                          fold = 1.5,
                          variables = ['Quantity', 'Rate', 'Value'])
outlier_pipeline = Pipeline(steps = [('winsor', winsor)])
outlier_pipeline


preprocessor1 = ColumnTransformer(transformers = [('wins',
                                                   outlier_pipeline,
                                                   numeric_features)],
                                  remainder = 'drop') # Drop all other columns
print(preprocessor1)


# Fit the data
winz_data = preprocessor1.fit(d_new)

# Save the pipeline
joblib.dump(winz_data, 'winzor')

steel_new  = pd.DataFrame(winz_data.transform(d_new), columns= numeric_features)
steel_new .info()

#steel_new = winsor.fit_transform(d_new[['Quantity', 'Rate', 'Value']])
```

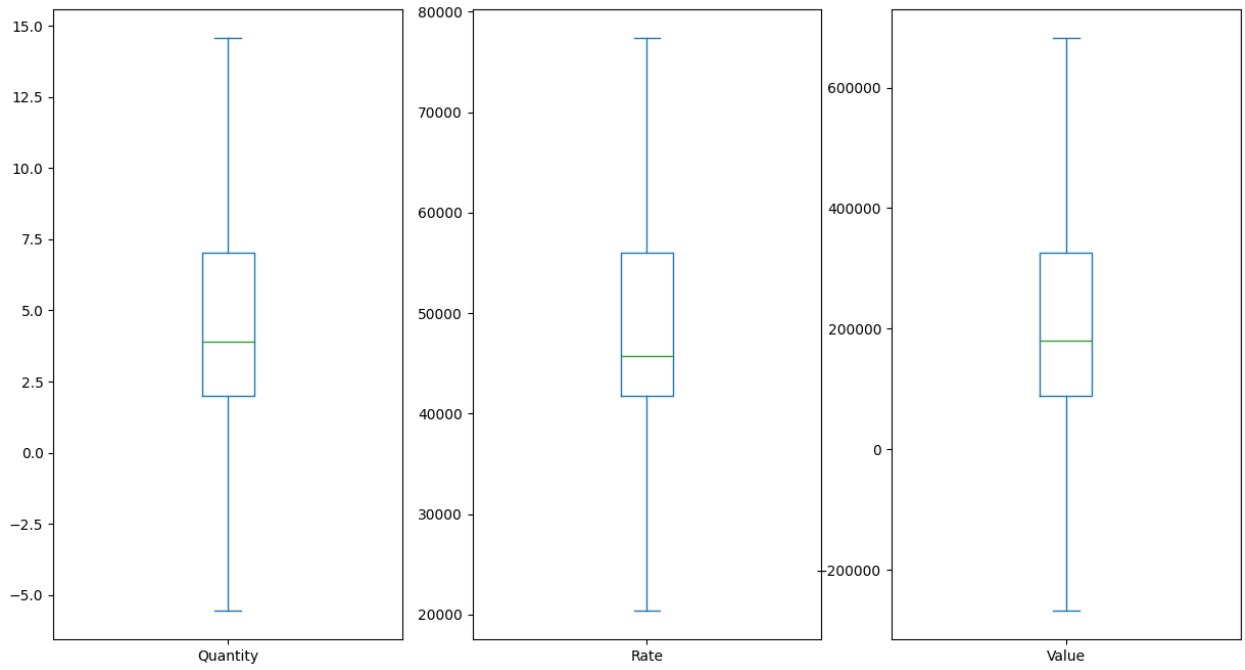
```
ColumnTransformer(transformers=[('wins',
                                 Pipeline(steps=[('winsor',
                                                  Winsorizer(capping_method='iqr',
                                                             fold=1.5,
                                                             tail='both',
                                                             variables=['Quantity',
                                                                        'Rate',
                                                                        'Value']))]),
                                 Index(['Quantity', 'Rate', 'Value'], dtype='object'))])
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32979 entries, 0 to 32978
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Quantity  32979 non-null  float64
 1   Rate      32979 non-null  float64
 2   Value     32979 non-null  float64
dtypes: float64(3)
memory usage: 773.1 KB
```

In [36]:

```
steel_new.plot(kind = 'box', subplots = True, sharey = False, figsize = (15, 8))

'''sharey True or 'all': x- or y-axis will be shared among all subplots.
False or 'none': each subplot x- or y-axis will be independent.'''
```

Out[36]:

```
"sharey True or 'all': x- or y-axis will be shared among all subplots.\nFalse or 'none': each subplot x- or y
-axis will be independent."
```



In [37]:

```
numerical_data2 = d_new.drop(columns=['Date','FY','Products','dia','dia group','grade','type','length','Voucher Type'])
```

Checking the skewness, kurtosis and variance value after outlier treatment

In [38]:

```
steel_new.skew()
```

Out[38]:

```
Quantity    1.064677
Rate        0.686953
Value       1.023098
dtype: float64
```

In [39]:

```
numerical_data2.skew()
```

Out[39]:

```
Quantity    2.200781
Rate        0.703662
Value       2.578341
dtype: float64
```

In [40]:

```
steel_new.kurt()
```

Out[40]:

```
Quantity    0.121707
Rate       -0.155223
Value       0.023423
dtype: float64
```

In [41]:

```
numerical_data2.kurt()
```

Out[41]:

```
Quantity    4.992102
Rate       -0.083138
Value       8.164775
dtype: float64
```

In [42]:

```
steel_new.std()
```

Out[42]:

```
Quantity         4.258855
Rate          9617.103014
Value       200229.179747
dtype: float64
```

In [43]:

```
numerical_data2.std()
```

Out[43]:

```
Quantity         6.671836
Rate          9643.022919
Value       331456.854884
dtype: float64
```

In [44]:

```
categorical_data = d_new[categorical_features]
categorical_data
#categorical_data.isna().sum()
```

Out[44]:

	Date	FY	Products	dia	dia group	grade	type	length	Voucher Type
0	04-03-2017	FY 18	25MM TATA TISCON FE500D (S)	25 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales
1	04-03-2017	FY 18	08MM TATA TISCON FE500D (S)	08 MM	08 MM	500D	FULL LENGTH	12 METER	Sales
2	04-03-2017	FY 18	10MM TATA TISCON FE500D (S)	10 MM	10 MM	500D	FULL LENGTH	12 METER	Sales
3	04-03-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales
4	04-03-2017	FY 18	16MM TATA TISCON FE500D (S)	16 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales
...	...	...	...	...	...	...	...	...	...
33040	3/25/2023	FY 23	20MM TATA TISCON FE500D (T)	20 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST
33041	3/30/2023	FY 23	08MM TATA TISCON FE550D (T)	08 MM	08 MM	550D	FULL LENGTH	12 METER	Credit Note
33042	3/30/2023	FY 23	10MM TATA TISCON FE550D (T)	10 MM	10 MM	550D	FULL LENGTH	12 METER	Credit Note
33043	3/30/2023	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Credit Note
33044	3/30/2023	FY 23	20MM TATA TISCON FE500D (T)	20 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Credit Note

32979 rows × 9 columns

In [45]:

```
# Reset the indices to align them correctly
categorical_data = categorical_data.reset_index(drop=True)
steel_new = steel_new.reset_index(drop=True)

# Concatenate the categorical and numerical data(after outlier treatment)
tmt_steel_data = pd.concat([categorical_data, steel_new], axis=1)

# Check the number of rows in the concatenated dataframe
print(len(tmt_steel_data))  # This should match the number of rows after resetting indices
```

```
32979
```

In [46]:

```
tmt_steel_data
```

Out[46]:

	Date	FY	Products	dia	dia group	grade	type	length	Voucher Type	Quantity	Rate	Value
0	04-03-2017	FY 18	25MM TATA TISCON FE500D (S)	25 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	14.55875	40000.0	681954.5
1	04-03-2017	FY 18	08MM TATA TISCON FE500D (S)	08 MM	08 MM	500D	FULL LENGTH	12 METER	Sales	9.09000	43200.0	392688.0
2	04-03-2017	FY 18	10MM TATA TISCON FE500D (S)	10 MM	10 MM	500D	FULL LENGTH	12 METER	Sales	3.94000	41700.0	164298.0
3	04-03-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.93000	41200.0	79516.0
4	04-03-2017	FY 18	16MM TATA TISCON FE500D (S)	16 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.03000	41200.0	42436.0
...	...	...	...	...	...	...	...	...	...	...	...	...
32974	3/25/2023	FY 23	20MM TATA TISCON FE500D (T)	20 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	4.08000	60000.0	244800.0
32975	3/30/2023	FY 23	08MM TATA TISCON FE550D (T)	08 MM	08 MM	550D	FULL LENGTH	12 METER	Credit Note	-1.07000	61000.0	-65270.0
32976	3/30/2023	FY 23	10MM TATA TISCON FE550D (T)	10 MM	10 MM	550D	FULL LENGTH	12 METER	Credit Note	-1.50000	60000.0	-90000.0
32977	3/30/2023	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Credit Note	-5.55125	59000.0	-267997.5
32978	3/30/2023	FY 23	20MM TATA TISCON FE500D (T)	20 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Credit Note	-4.60000	59000.0	-267997.5

32979 rows × 12 columns

In [ ]:

```
# checking for duplicates after outlier treatment
#d1=tmt_steel_data.duplicated()
#sum(d1)
```

In [ ]:

```
#Removing Duplicates
#d_new1 = tmt_steel_data.drop_duplicates() # Returns DataFrame with duplicate rows removed.
```

In [ ]:

```
#d_new1.to_csv('Steel_Dataset.csv', index=False)
```

In [47]:

```
#Performing Label encoding on categorical data
from sklearn.preprocessing import LabelEncoder
```

In [51]:

```
# Creating instance of labelencoder
#labelencoder = LabelEncoder()

#X= d_new1.iloc[:, 1:9]
```

In [ ]:

```
#X
```

In [52]:

```
#X['FY'] = labelencoder.fit_transform(X['FY'])
#X['Products'] = labelencoder.fit_transform(X['Products'])
#X['dia'] = labelencoder.fit_transform(d_new1['dia'])
#X['dia group'] = labelencoder.fit_transform(X['dia group'])
#X['grade'] = labelencoder.fit_transform(X['grade'])
#X['type'] = labelencoder.fit_transform(X['type'])
#X['length'] = labelencoder.fit_transform(X['length'])
#X['Voucher Type'] = labelencoder.fit_transform(X['Voucher Type'])
```

In [ ]:

```
# Reset the indices to align them correctly and concate
#X = X.reset_index(drop=True)
#umerical_data = d_new1[numeric_features]
#numerical_data = numerical_data.reset_index(drop=True)


#Label_enc_data = pd.concat([X, numerical_data], axis=1)
```

In [ ]:

```
#Label_enc_data
```

In [ ]:

```
# Address the scaling issue
#scale_pipeline = Pipeline(steps=[('scale', StandardScaler())])
# scale_pipeline = Pipeline(steps=[('scale', MinMaxScaler())])

#preprocessor2 = ColumnTransformer(transformers = [('num',
#                                                   scale_pipeline, numeric_features)],
#                                  remainder = 'drop')

#print(preprocessor2)

#scale = preprocessor2.fit(Label_enc_data)

#joblib.dump(scale, 'scale')

#scaled_data = pd.DataFrame(scale.transform(Label_enc_data), columns = numeric_features)
#scaled_data.columns
#scaled_data.info()
########################
```

In [ ]:

```
#scaled_data
```

In [ ]:

```
#scaled_data = scaled_data.reset_index(drop=True)


#clean_data = pd.concat([X,scaled_data], axis=1)
```

In [ ]:

```
#clean_data
```

In [53]:

```
# Calculate the correlation coefficient
correlation = tmt_steel_data["Quantity"].corr(tmt_steel_data["Value"])
```

In [54]:

```
correlation
```

Out[54]:

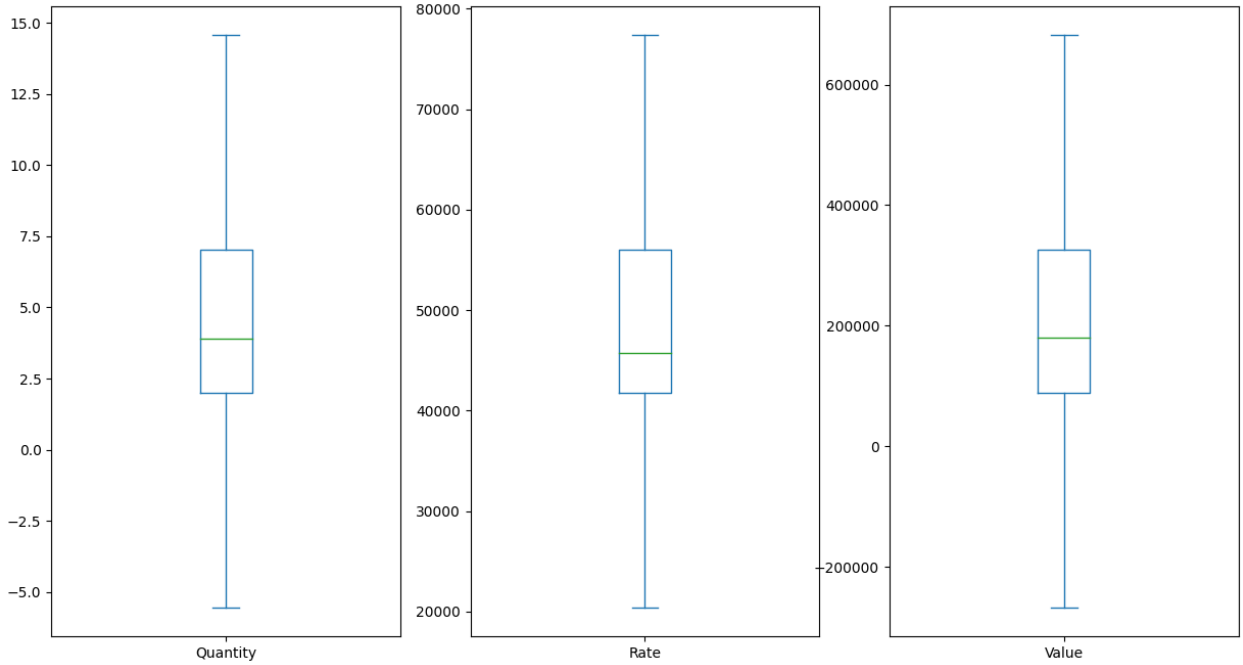
```
0.9762349764610823
```

In [55]:

```
# Multiple boxplots in a single visualization.
# Columns with larger scales affect other columns.
# Below code ensures each column gets its own y-axis.

# pandas plot() function with parameters kind = 'box' and subplots = True

tmt_steel_data.plot(kind = 'box', subplots = True, sharey = False, figsize = (15, 8))

'''sharey True or 'all': x-axis or y-axis will be shared among all subplots.
False or 'none': each subplot x- or y-axis will be independent.'''
```

Out[55]:

```
"sharey True or 'all': x-axis or y-axis will be shared among all subplots.\nFalse or 'none': each subplot x-
or y-axis will be independent."
```



In [111]:

```
# Filtering data for "12mm" diameter
df = d_new[d_new["dia"] == "12 MM"]
```

In [112]:

```
df
```

Out[112]:

	Date	FY	Products	dia	dia group	grade	type	length	Voucher Type	Quantity	Rate	Value
3	04-03-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.93	41200.00	79516.00
7	04-04-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	15.24	37976.19	578757.14
10	04-06-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	14.62	41500.00	606730.00
25	04-06-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	3.03	41000.00	124230.00
33	04-07-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	5.15	40800.00	210120.00
...	...	...	...	...	...	...	...	...	...	...	...	...
33024	03-07-2023	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	9.98	61950.00	618261.00
33027	03-10-2023	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	15.16	63200.00	958112.00
33031	3/23/2023	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Sales A/c GST	2.09	60400.00	126236.00
33038	3/25/2023	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	8.66	60000.00	519600.00
33043	3/30/2023	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Credit Note	-20.04	59000.00	-1182360.00

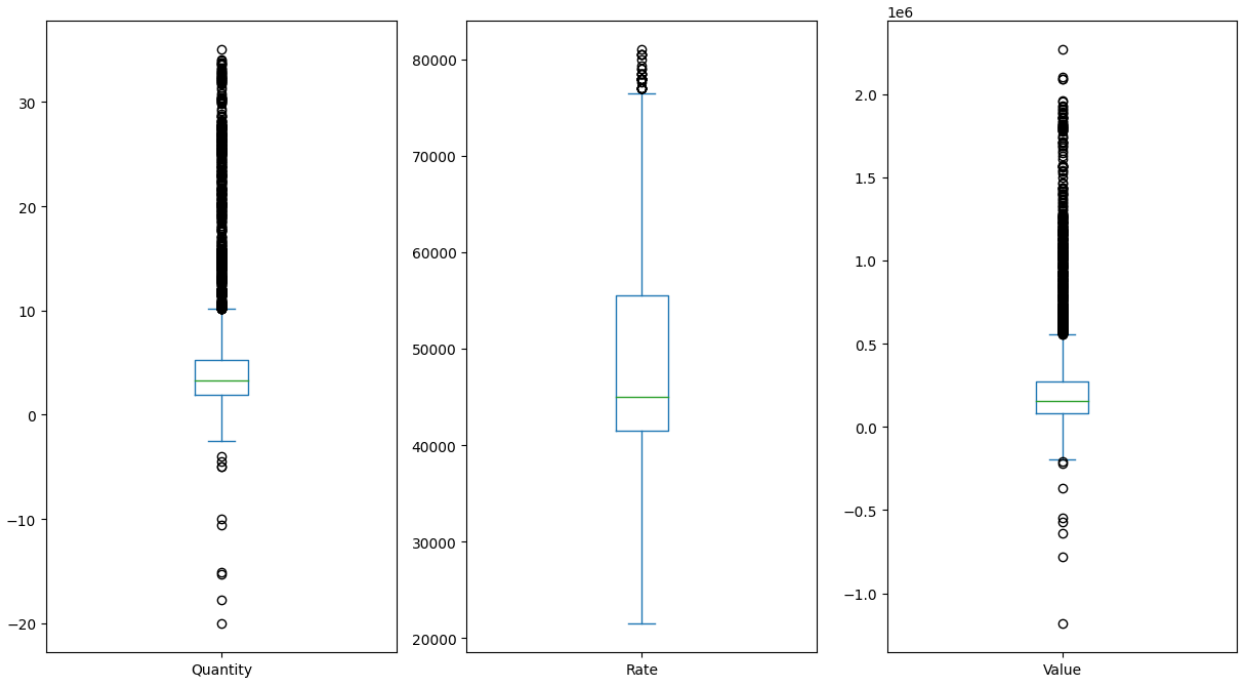
6557 rows × 12 columns

In [113]:

```
df.plot(kind = 'box', subplots = True, sharey = False, figsize = (15, 8))
```

Out[113]:

```
Quantity       AxesSubplot(0.125,0.11;0.227941x0.77)
Rate        AxesSubplot(0.398529,0.11;0.227941x0.77)
Value       AxesSubplot(0.672059,0.11;0.227941x0.77)
dtype: object
```



In [68]:

```
winsor = Winsorizer(capping_method = 'iqr', # choose  IQR rule boundaries or gaussian for mean and std
                          tail = 'both', # cap left, right or both tails
                          fold = 1.5,
                          variables = ['Quantity', 'Rate', 'Value'])
outlier_pipeline = Pipeline(steps = [('winsor', winsor)])
outlier_pipeline


preprocessor1 = ColumnTransformer(transformers = [('wins',
                                                   outlier_pipeline,
                                                   numeric_features)],
                                  remainder = 'drop') # Drop all other columns
print(preprocessor1)


# Fit the data
winz_data = preprocessor1.fit(filtered_data)

# Save the pipeline
joblib.dump(winz_data, 'winzor')

filtered_data  = pd.DataFrame(winz_data.transform(filtered_data), columns= numeric_features)
filtered_data .info()

#steel_new = winsor.fit_transform(d_new[['Quantity', 'Rate', 'Value']])
```

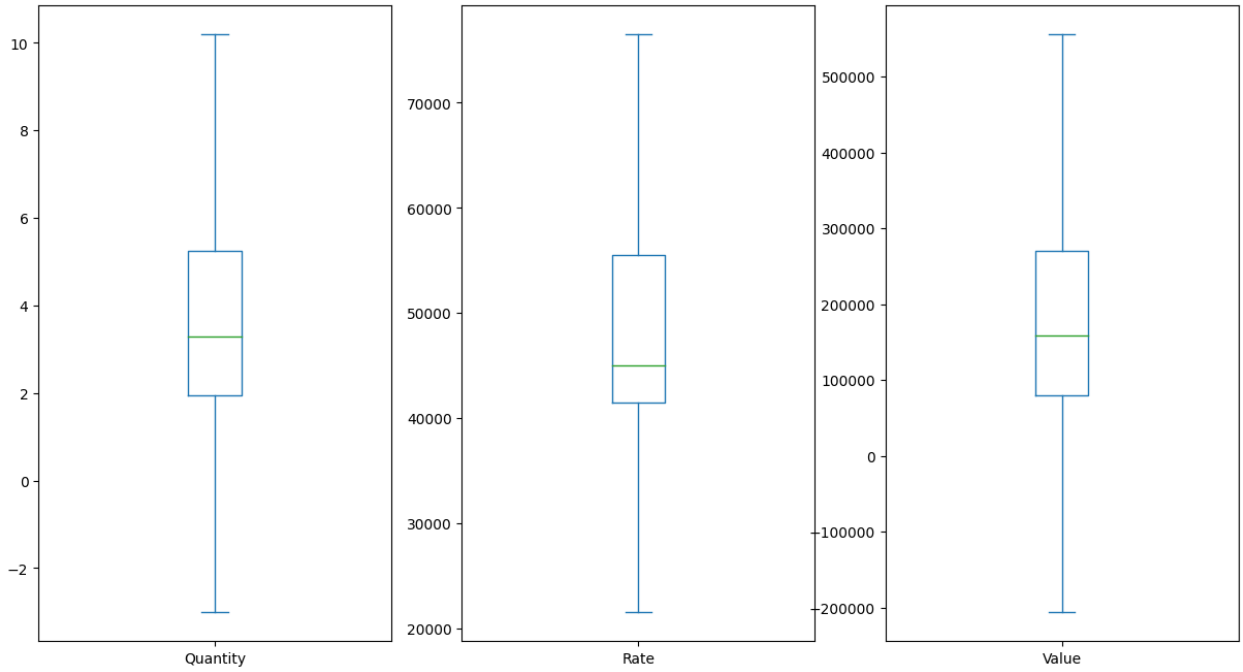
```
ColumnTransformer(transformers=[('wins',
                                 Pipeline(steps=[('winsor',
                                                  Winsorizer(capping_method='iqr',
                                                             fold=1.5,
                                                             tail='both',
                                                             variables=['Quantity',
                                                                        'Rate',
                                                                        'Value']))]),
                                 Index(['Quantity', 'Rate', 'Value'], dtype='object'))])
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6557 entries, 0 to 6556
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Quantity  6557 non-null   float64
 1   Rate      6557 non-null   float64
 2   Value     6557 non-null   float64
dtypes: float64(3)
memory usage: 153.8 KB
```

In [69]:

```
filtered_data.plot(kind = 'box', subplots = True, sharey = False, figsize = (15, 8))
```

Out[69]:

```
Quantity       AxesSubplot(0.125,0.11;0.227941x0.77)
Rate        AxesSubplot(0.398529,0.11;0.227941x0.77)
Value       AxesSubplot(0.672059,0.11;0.227941x0.77)
dtype: object
```



In [70]:

```
filtered_data
```

Out[70]:

	Quantity	Rate	Value
0	1.93	41200.00	79516.0
1	10.19	37976.19	555690.0
2	10.19	41500.00	555690.0
3	3.03	41000.00	124230.0
4	5.15	40800.00	210120.0
...	...	...	...
6552	9.98	61950.00	555690.0
6553	10.19	63200.00	555690.0
6554	2.09	60400.00	126236.0
6555	8.66	60000.00	519600.0
6556	-3.01	59000.00	-206150.0

6557 rows × 3 columns

In [75]:

```
categorical_data1 = df[categorical_features]
categorical_data1
```

Out[75]:

	Date	FY	Products	dia	dia group	grade	type	length	Voucher Type
3	04-03-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales
7	04-04-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales
10	04-06-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales
25	04-06-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales
33	04-07-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales
...	...	...	...	...	...	...	...	...	...
33024	03-07-2023	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST
33027	03-10-2023	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST
33031	3/23/2023	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Sales A/c GST
33038	3/25/2023	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST
33043	3/30/2023	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Credit Note

6557 rows × 9 columns

In [76]:

```
# Reset the indices to align them correctly
categorical_data1 = categorical_data1.reset_index(drop=True)
filtered_data = filtered_data.reset_index(drop=True)

# Concatenate the categorical and numerical data(after outlier treatment)
tmt_12mm_data = pd.concat([categorical_data1, filtered_data], axis=1)

# Check the number of rows in the concatenated dataframe
print(len(tmt_12mm_data))  # This should match the number of rows after resetting indices
```

```
6557
```

In [77]:

```
tmt_12mm_data
```

Out[77]:

	Date	FY	Products	dia	dia group	grade	type	length	Voucher Type	Quantity	Rate	Value
0	04-03-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.93	41200.00	79516.0
1	04-04-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	10.19	37976.19	555690.0
2	04-06-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	10.19	41500.00	555690.0
3	04-06-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	3.03	41000.00	124230.0
4	04-07-2017	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	5.15	40800.00	210120.0
...	...	...	...	...	...	...	...	...	...	...	...	...
6552	03-07-2023	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	9.98	61950.00	555690.0
6553	03-10-2023	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	10.19	63200.00	555690.0
6554	3/23/2023	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Sales A/c GST	2.09	60400.00	126236.0
6555	3/25/2023	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	8.66	60000.00	519600.0
6556	3/30/2023	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Credit Note	-3.01	59000.00	-206150.0

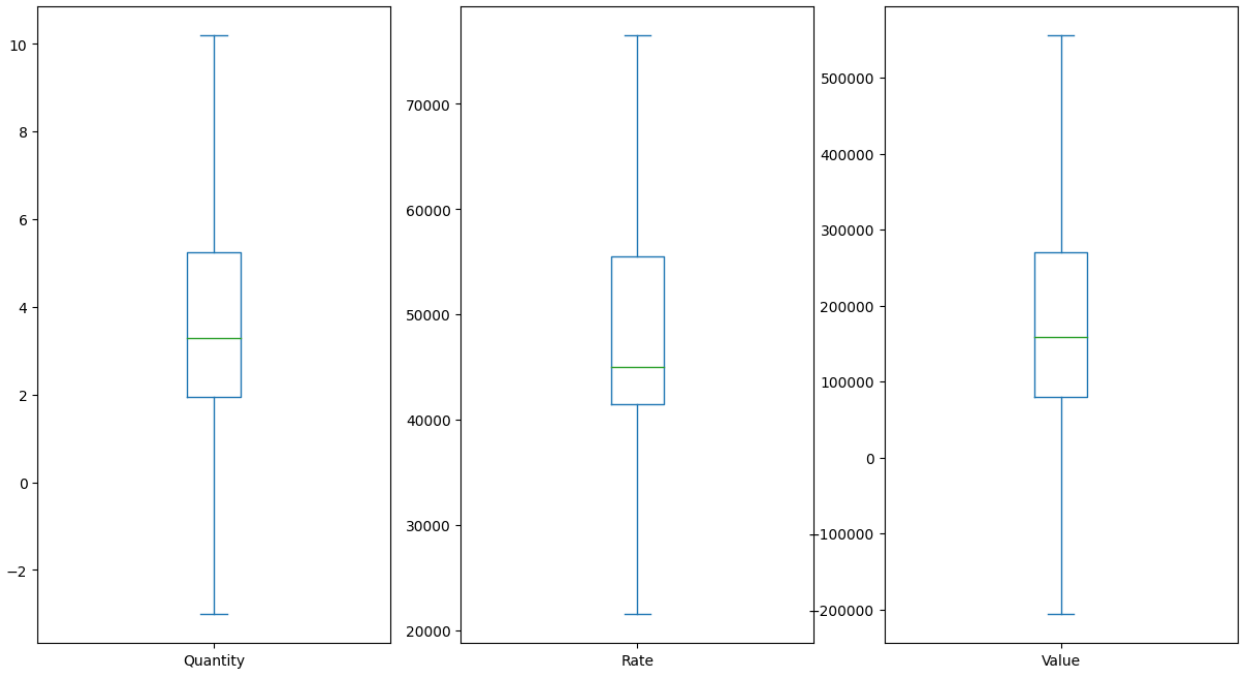
6557 rows × 12 columns

In [78]:

```
tmt_12mm_data.plot(kind = 'box', subplots = True, sharey = False, figsize = (15, 8))
```

Out[78]:

```
Quantity       AxesSubplot(0.125,0.11;0.227941x0.77)
Rate        AxesSubplot(0.398529,0.11;0.227941x0.77)
Value       AxesSubplot(0.672059,0.11;0.227941x0.77)
dtype: object
```



In [80]:

```
# Converting the data type of date column from object to date.
tmt_12mm_data['Date'] = pd.to_datetime(tmt_12mm_data['Date'], format= 'mixed')


# Extract the month from the 'Date' column
tmt_12mm_data['Month'] = tmt_12mm_data['Date'].dt.month
```

In [81]:

```
tmt_12mm_data
```

Out[81]:

	Date	FY	Products	dia	dia group	grade	type	length	Voucher Type	Quantity	Rate	Value	Month
0	2017-04-03	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.93	41200.00	79516.0	4
1	2017-04-04	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	10.19	37976.19	555690.0	4
2	2017-04-06	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	10.19	41500.00	555690.0	4
3	2017-04-06	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	3.03	41000.00	124230.0	4
4	2017-04-07	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	5.15	40800.00	210120.0	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6552	2023-03-07	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	9.98	61950.00	555690.0	3
6553	2023-03-10	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	10.19	63200.00	555690.0	3
6554	2023-03-23	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Sales A/c GST	2.09	60400.00	126236.0	3
6555	2023-03-25	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	8.66	60000.00	519600.0	3
6556	2023-03-30	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Credit Note	-3.01	59000.00	-206150.0	3

6557 rows × 13 columns

In [90]:

```
# Calculate months elapsed since the first observation
tmt_12mm_data['t'] = ((tmt_12mm_data['Date'] - tmt_12mm_data['Date'].iloc[0]) / pd.Timedelta(days=30)).astype(int)  # Assumi
tmt_12mm_data["t_square"] =tmt_12mm_data["t"] * tmt_12mm_data["t"]
 # Quadratic trend or polynomial with '2' degrees trend is captured
tmt_12mm_data["log_Quantity"] = np.log(tmt_12mm_data["Quantity"]) # Exponential trend is captured
```

```
C:\Users\MsK_PC\anaconda3\envs\DS\lib\site-packages\pandas\core\arraylike.py:396: RuntimeWarning: invalid val
ue encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

In [146]:

```
tmt_12mm_data
```

Out[146]:

	Date	FY	Products	dia	dia group	grade	type	length	Voucher Type	Quantity	Rate	Value	Month	t	t_square	log_Quai
0	2017-04-03	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	1.93	41200.00	79516.0	4	0	0	0.657
1	2017-04-04	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	10.19	37976.19	555690.0	4	0	0	2.321
2	2017-04-06	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	10.19	41500.00	555690.0	4	0	0	2.321
3	2017-04-06	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	3.03	41000.00	124230.0	4	0	0	1.108
4	2017-04-07	FY 18	12MM TATA TISCON FE500D (S)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales	5.15	40800.00	210120.0	4	0	0	1.638
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
6552	2023-03-07	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	9.98	61950.00	555690.0	3	72	5184	2.300
6553	2023-03-10	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	10.19	63200.00	555690.0	3	72	5184	2.321
6554	2023-03-23	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Sales A/c GST	2.09	60400.00	126236.0	3	72	5184	0.737
6555	2023-03-25	FY 23	12MM TATA TISCON FE500D (T)	12 MM	12 MM - 32 MM	500D	FULL LENGTH	12 METER	Sales A/c GST	8.66	60000.00	519600.0	3	72	5184	2.158
6556	2023-03-30	FY 23	12MM TATA TISCON FE550D (T)	12 MM	12 MM - 32 MM	550D	FULL LENGTH	12 METER	Credit Note	-3.01	59000.00	-206150.0	3	72	5184	

6557 rows × 16 columns

In [147]:

```
df1= tmt_12mm_data[['Date','dia','Quantity','Rate','Value','Month','t','t_square','log_Quantity']]
```

In [148]:

```
df1
```

Out[148]:

	Date	dia	Quantity	Rate	Value	Month	t	t_square	log_Quantity
0	2017-04-03	12 MM	1.93	41200.00	79516.0	4	0	0	0.657520
1	2017-04-04	12 MM	10.19	37976.19	555690.0	4	0	0	2.321407
2	2017-04-06	12 MM	10.19	41500.00	555690.0	4	0	0	2.321407
3	2017-04-06	12 MM	3.03	41000.00	124230.0	4	0	0	1.108563
4	2017-04-07	12 MM	5.15	40800.00	210120.0	4	0	0	1.638997
...	...	...	...	...	...	...	...	...	...
6552	2023-03-07	12 MM	9.98	61950.00	555690.0	3	72	5184	2.300583
6553	2023-03-10	12 MM	10.19	63200.00	555690.0	3	72	5184	2.321407
6554	2023-03-23	12 MM	2.09	60400.00	126236.0	3	72	5184	0.737164
6555	2023-03-25	12 MM	8.66	60000.00	519600.0	3	72	5184	2.158715
6556	2023-03-30	12 MM	-3.01	59000.00	-206150.0	3	72	5184	NaN

6557 rows × 9 columns

In [149]:

```
import calendar
# Map numeric month values to month names using the calendar module
df1['Month'] = df1['Month'].apply(lambda x: calendar.month_name[x])
```

```
C:\Users\MsK_PC\AppData\Local\Temp\ipykernel_13168\2684757153.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#r
eturning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returnin
g-a-view-versus-a-copy)
  df1['Month'] = df1['Month'].apply(lambda x: calendar.month_name[x])
```

In [150]:

```
df1
```

Out[150]:

	Date	dia	Quantity	Rate	Value	Month	t	t_square	log_Quantity
0	2017-04-03	12 MM	1.93	41200.00	79516.0	April	0	0	0.657520
1	2017-04-04	12 MM	10.19	37976.19	555690.0	April	0	0	2.321407
2	2017-04-06	12 MM	10.19	41500.00	555690.0	April	0	0	2.321407
3	2017-04-06	12 MM	3.03	41000.00	124230.0	April	0	0	1.108563
4	2017-04-07	12 MM	5.15	40800.00	210120.0	April	0	0	1.638997
...	...	...	...	...	...	...	...	...	...
6552	2023-03-07	12 MM	9.98	61950.00	555690.0	March	72	5184	2.300583
6553	2023-03-10	12 MM	10.19	63200.00	555690.0	March	72	5184	2.321407
6554	2023-03-23	12 MM	2.09	60400.00	126236.0	March	72	5184	0.737164
6555	2023-03-25	12 MM	8.66	60000.00	519600.0	March	72	5184	2.158715
6556	2023-03-30	12 MM	-3.01	59000.00	-206150.0	March	72	5184	NaN

6557 rows × 9 columns

In [151]:

```
month_dummies = pd.DataFrame(pd.get_dummies(df1['Month']))
d1 = pd.concat([df1, month_dummies], axis = 1)
d1 = d1.drop(columns = "Month")
```

In [152]:

```
d1
```

Out[152]:

	Date	dia	Quantity	Rate	Value	t	t_square	log_Quantity	April	August	December	February	January	July	June
0	2017-04-03	12 MM	1.93	41200.00	79516.0	0	0	0.657520	True	False	False	False	False	False	False
1	2017-04-04	12 MM	10.19	37976.19	555690.0	0	0	2.321407	True	False	False	False	False	False	False
2	2017-04-06	12 MM	10.19	41500.00	555690.0	0	0	2.321407	True	False	False	False	False	False	False
3	2017-04-06	12 MM	3.03	41000.00	124230.0	0	0	1.108563	True	False	False	False	False	False	False
4	2017-04-07	12 MM	5.15	40800.00	210120.0	0	0	1.638997	True	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6552	2023-03-07	12 MM	9.98	61950.00	555690.0	72	5184	2.300583	False	False	False	False	False	False	False
6553	2023-03-10	12 MM	10.19	63200.00	555690.0	72	5184	2.321407	False	False	False	False	False	False	False
6554	2023-03-23	12 MM	2.09	60400.00	126236.0	72	5184	0.737164	False	False	False	False	False	False	False
6555	2023-03-25	12 MM	8.66	60000.00	519600.0	72	5184	2.158715	False	False	False	False	False	False	False
6556	2023-03-30	12 MM	-3.01	59000.00	-206150.0	72	5184	NaN	False	False	False	False	False	False	False

6557 rows × 20 columns

In [153]:

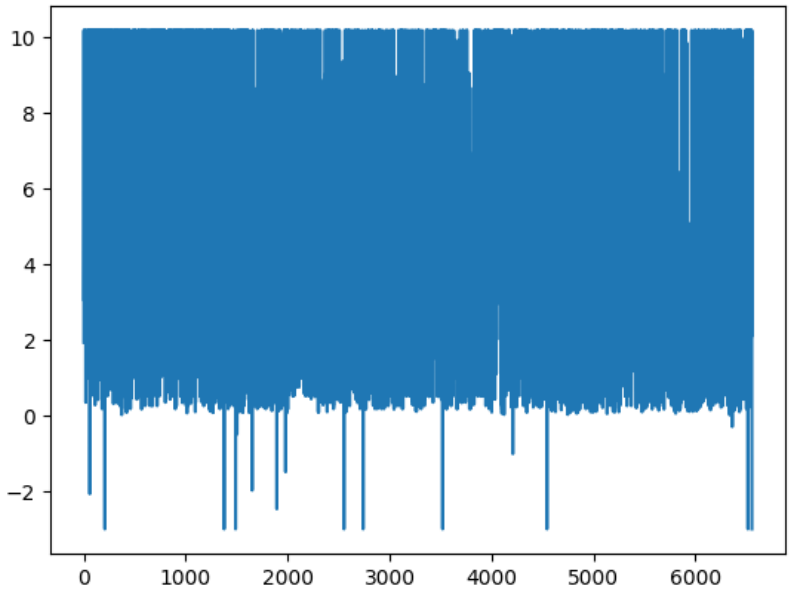
```
# Data Partition
Train = d1.head(5246)
Test = d1.tail(1311)
```

In [154]:

```
# Visualization - Time plot
d1.Quantity.plot()
```

Out[154]:

```
<AxesSubplot: >
```



In [155]:

```
# to change the index value in pandas data frame
# Test.set_index(np.arange(1, 13))

######################## Linear ###########################
import statsmodels.formula.api as smf

linear_model = smf.ols('Quantity ~ t', data = Train).fit()
pred_linear =  pd.Series(linear_model.predict(pd.DataFrame(Test['t'])))
rmse_linear = np.sqrt(np.mean((np.array(Test['Quantity']) - np.array(pred_linear))**2))
rmse_linear
```

Out[155]:

```
3.0050578735585494
```

In [156]:

```
################## Additive Seasonality Quadratic Trend ############################

add_sea_Quad = smf.ols('Quantity ~ t + t_square + January + February + March + April + May + June + July + August + Septembe
pred_add_sea_quad = pd.Series(add_sea_Quad.predict(Test[['January', 'February', 'March', 'April', 'May', 'June', 'July', 'Au
rmse_add_sea_quad = np.sqrt(np.mean((np.array(Test['Quantity']) - np.array(pred_add_sea_quad))**2))
rmse_add_sea_quad
```

Out[156]:

```
3.0352208038324733
```

In [157]:

```
# Handling missing values.
from feature_engine.imputation import RandomSampleImputer

random_imputer = RandomSampleImputer(['log_Quantity'])
df1["log_Quantity"] = pd.DataFrame(random_imputer.fit_transform(df1[["log_Quantity"]]))

df1.isna().sum()
```

```
C:\Users\MsK_PC\AppData\Local\Temp\ipykernel_13168\2683403812.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#r
eturning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returnin
g-a-view-versus-a-copy)
  df1["log_Quantity"] = pd.DataFrame(random_imputer.fit_transform(df1[["log_Quantity"]]))
```

Out[157]:

```
Date           0
dia            0
Quantity       0
Rate           0
Value          0
Month          0
t              0
t_square       0
log_Quantity   0
dtype: int64
```

In [158]:

```
df1
```

Out[158]:

	Date	dia	Quantity	Rate	Value	Month	t	t_square	log_Quantity
0	2017-04-03	12 MM	1.93	41200.00	79516.0	April	0	0	0.657520
1	2017-04-04	12 MM	10.19	37976.19	555690.0	April	0	0	2.321407
2	2017-04-06	12 MM	10.19	41500.00	555690.0	April	0	0	2.321407
3	2017-04-06	12 MM	3.03	41000.00	124230.0	April	0	0	1.108563
4	2017-04-07	12 MM	5.15	40800.00	210120.0	April	0	0	1.638997
...	...	...	...	...	...	...	...	...	...
6552	2023-03-07	12 MM	9.98	61950.00	555690.0	March	72	5184	2.300583
6553	2023-03-10	12 MM	10.19	63200.00	555690.0	March	72	5184	2.321407
6554	2023-03-23	12 MM	2.09	60400.00	126236.0	March	72	5184	0.737164
6555	2023-03-25	12 MM	8.66	60000.00	519600.0	March	72	5184	2.158715
6556	2023-03-30	12 MM	-3.01	59000.00	-206150.0	March	72	5184	1.085189

6557 rows × 9 columns

In [159]:

```
###################### Exponential ##############################

Exp = smf.ols('log_Quantity ~ t', data = Train).fit()
pred_Exp = pd.Series(Exp.predict(pd.DataFrame(Test['t'])))
rmse_Exp = np.sqrt(np.mean((np.array(Test['Quantity']) - np.array(np.exp(pred_Exp)))**2))
rmse_Exp
```

Out[159]:

```
3.3162814472333517
```

In [160]:

```
##################### Quadratic ###############################

Quad = smf.ols('Quantity ~ t + t_square', data = Train).fit()
pred_Quad = pd.Series(Quad.predict(Test[["t", "t_square"]]))
rmse_Quad = np.sqrt(np.mean((np.array(Test['Quantity']) - np.array(pred_Quad))**2))
rmse_Quad
```

Out[160]:

```
3.007922357840731
```

In [162]:

```
asonality ########################

anuary + February + March + April + May + June + July + August + September + October + November + December', data = Train).f
a.predict(Test[['January','February','March','April','May','June','July','August','September','October','November','December
(np.array(Test['Quantity']) - np.array(pred_add_sea))**2))
```

Out[162]:

```
3.024651775140944
```

In [163]:

```
e Seasonality ##################

~ January + February + March + April + May + June + July + August + September + October + November + December', data = Train
a.predict(Test[['January','February','March','April','May','June','July','August','September','October','November','December
(np.array(Test['Quantity']) - np.array(np.exp(pred_Mult_sea)))**2))
```

Out[163]:

```
3.140507207666266
```

In [165]:

```
################## Additive Seasonality Quadratic Trend ############################

add_sea_Quad = smf.ols('Quantity ~ t + t_square + January + February + March + April + May + June + July + August + Septembe
pred_add_sea_quad = pd.Series(add_sea_Quad.predict(Test[['January','February','March','April','May','June','July','August','
rmse_add_sea_quad = np.sqrt(np.mean((np.array(Test['Quantity']) - np.array(pred_add_sea_quad))**2))
rmse_add_sea_quad
```

Out[165]:

```
3.035220803832364
```

In [166]:

```
################## Multiplicative Seasonality Linear Trend  ###########

Mul_sea_linear = smf.ols('Quantity ~ t + January + February + March + April + May + June + July + August + September + Octob
pred_Mult_sea_linear = pd.Series(Mul_sea_linear.predict(Test[['January','February','March','April','May','June','July','Augu
rmse_Mult_sea_linear = np.sqrt(np.mean((np.array(Test['Quantity']) - np.array(np.exp(pred_Mult_sea_linear)))**2))
rmse_Mult_sea_linear
```

Out[166]:

```
44.08469037318112
```

In [168]:

```
################## Testing ########################################

Models = {"MODEL":pd.Series(["rmse_linear", "rmse_Exp", "rmse_Quad", "rmse_add_sea", "rmse_Mult_sea", "rmse_add_sea_quad", "
table_rmse = pd.DataFrame(Models)
table_rmse
```

Out[168]:

	MODEL	RMSE_Values
0	rmse_linear	3.005058
1	rmse_Exp	3.316281
2	rmse_Quad	3.007922
3	rmse_add_sea	3.024652
4	rmse_Mult_sea	3.140507
5	rmse_add_sea_quad	3.035221
6	rmse_Mult_sea_linear	44.084690

In [177]:

```
# 'rmse_linear' has the least RMSE value among the models prepared so far. Use these features and build forecasting model us
model_full = smf.ols('Quantity ~ t', data = Train).fit()

predict_data1 = pd.read_csv(r"C:\Users\MsK_PC\360digiTMG\Data_Science\DS_TMT_Steel_Project\Client_data\Predict_data.csv")

pred_new  = pd.Series(model_full.predict(predict_data1))
pred_new
```

Out[177]:

```
0     3.567823
1     3.555061
2     3.542299
3     3.529536
4     3.516774
5     3.504012
6     3.491250
7     3.478488
8     3.465725
9     3.452963
10    3.440201
11    3.427439
12    3.414677
dtype: float64
```

In [178]:

```
predict_data1["forecasted_Quantity"] = pd.Series(pred_new)
```

In [179]:

```
predict_data1
```

Out[179]:

	Date	t	t_square	dia	April	May	June	July	August	September	October	November	December	January	February	March	t
0	01-04-23	73	5329	12MM	1	0	0	0	0	0	0	0	0	0	0	0	
1	01-05-23	74	5476	12MM	0	1	0	0	0	0	0	0	0	0	0	0	
2	31-05-23	75	5625	12MM	0	1	0	0	0	0	0	0	0	0	0	0	
3	30-06-23	76	5776	12MM	0	0	1	0	0	0	0	0	0	0	0	0	
4	30-07-23	77	5929	12MM	0	0	0	1	0	0	0	0	0	0	0	0	
5	29-08-23	78	6084	12MM	0	0	0	0	1	0	0	0	0	0	0	0	
6	28-09-23	79	6241	12MM	0	0	0	0	0	1	0	0	0	0	0	0	
7	28-10-23	80	6400	12MM	0	0	0	0	0	0	1	0	0	0	0	0	
8	27-11-23	81	6561	12MM	0	0	0	0	0	0	0	1	0	0	0	0	
9	27-12-23	82	6724	12MM	0	0	0	0	0	0	0	0	1	0	0	0	
10	26-01-24	83	6889	12MM	0	0	0	0	0	0	0	0	0	1	0	0	
11	25-02-24	84	7056	12MM	0	0	0	0	0	0	0	0	0	0	1	0	
12	26-03-24	85	7225	12MM	0	0	0	0	0	0	0	0	0	0	0	1	

In [227]:

```
# The models and results have save and load method, so you don't need to use the pickle module directly.
# to save model
model_full.save("model.pickle")

import os
os.getcwd()

# to load model
from statsmodels.regression.linear_model import OLSResults
model = OLSResults.load("model.pickle")
```

In [181]:

```
# RESIDUALS MIGHT HAVE ADDITIONAL INFORMATION!

# Autoregression Model (AR)
# Calculating Residuals from best model applied on full data
# AV - FV
full_res = d1.Quantity - model.predict(d1)

full_res
```

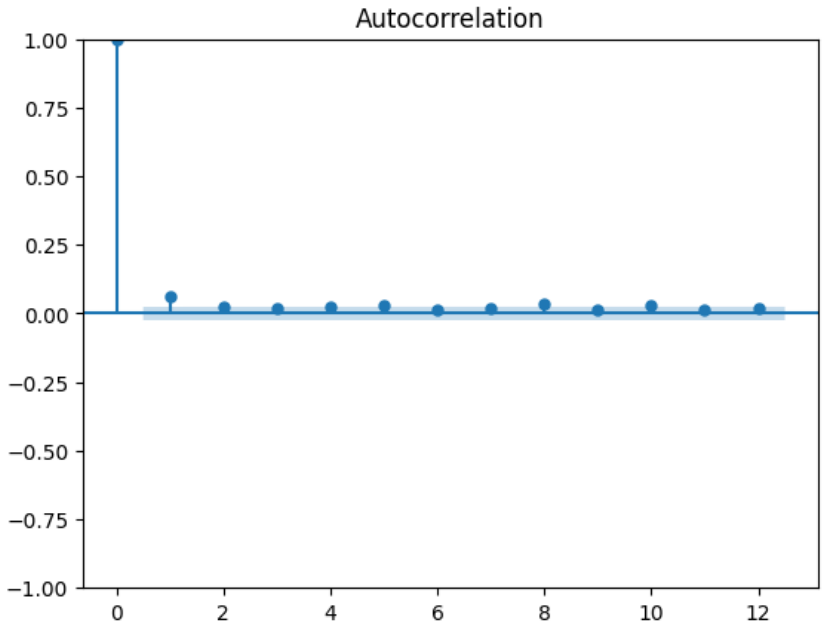
Out[181]:

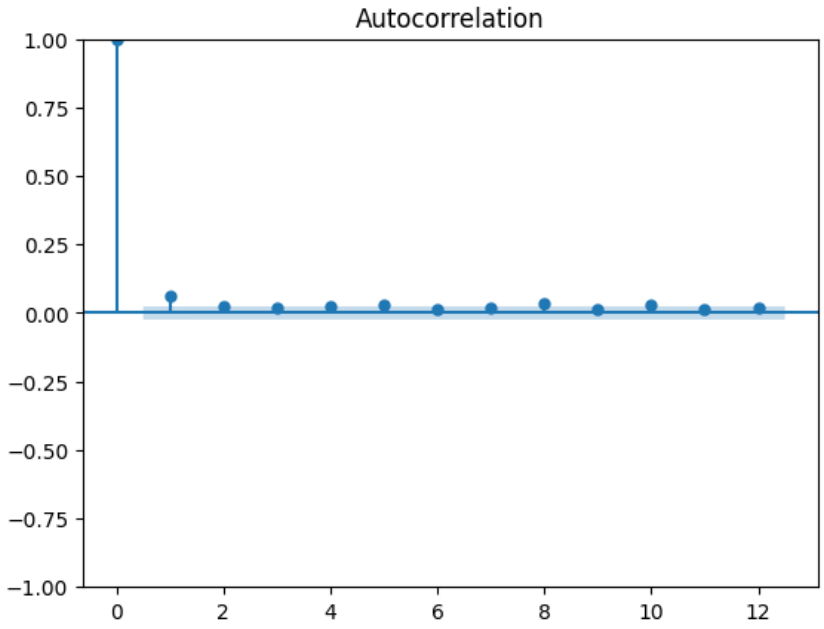
```
0      -2.569463
1       5.690537
2       5.690537
3      -1.469463
4       0.650537
          ...
6552    6.399415
6553    6.609415
6554   -1.490585
6555    5.079415
6556   -6.590585
Length: 6557, dtype: float64
```

In [182]:

```
# ACF plot on residuals
import statsmodels.graphics.tsaplots as tsa_plots
tsa_plots.plot_acf(full_res, lags = 12)
# ACF is an (complete) auto-correlation function gives values
# of auto-correlation of any time series with its lagged values.
```

Out[182]:

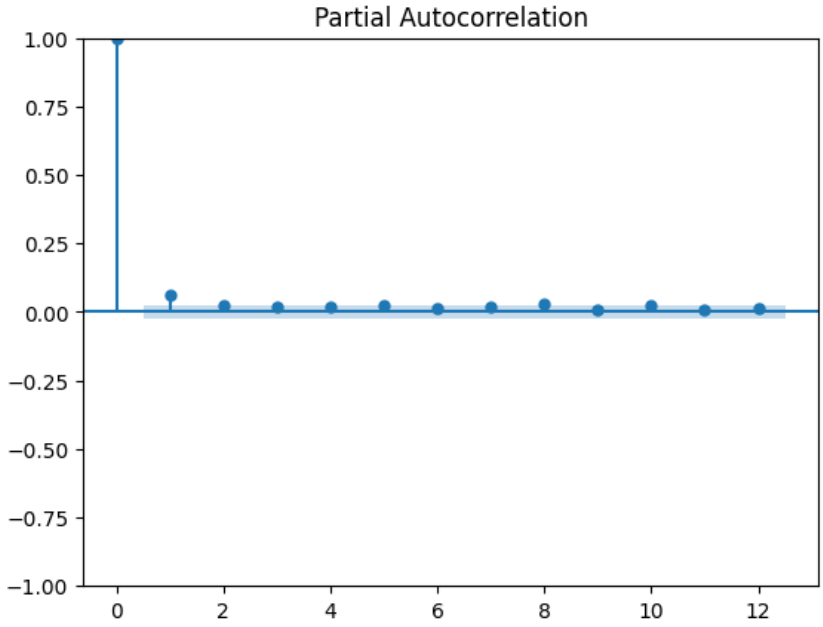


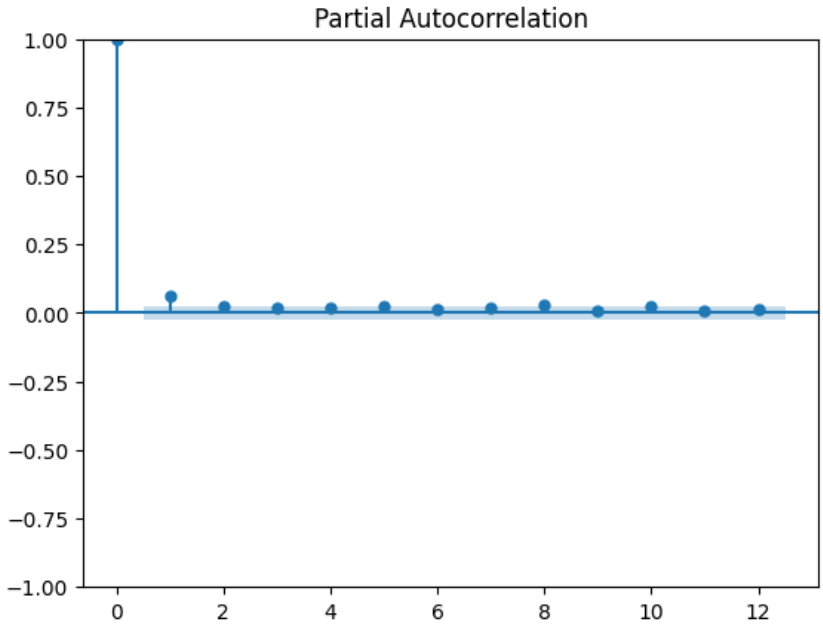


In [183]:

```
# PACF is a partial auto-correlation function.
# It finds correlations of Y with lags of the residuals of the time series
tsa_plots.plot_pacf(full_res, lags = 12)
```

Out[183]:





In [187]:

```
# Alternative approach for ACF plot is explained in next 2 lines
# from pandas.plotting import autocorrelation_plot
# autocorrelation_ppyplot.show()

# AR Autoregressive model
from statsmodels.tsa.ar_model import AutoReg
model_ar = AutoReg(full_res, lags = [1])
model_fit = model_ar.fit()

print('Coefficients: %s' % model_fit.params)
```

```
Coefficients: const    0.036166
y.L1     0.060479
dtype: float64
```

In [188]:

```
pred_res = model_fit.predict(start = len(full_res), end = len(full_res) + len(predict_data1) - 1, dynamic = False)
pred_res.reset_index(drop = True, inplace = True)

# The Final Predictions using ASQT and AR(1) Model
final_pred = pred_new + pred_res
final_pred
```

Out[188]:

```
0     3.205398
1     3.569307
2     3.579326
3     3.567941
4     3.555262
5     3.542505
6     3.529743
7     3.516981
8     3.504219
9     3.491457
10    3.478695
11    3.465932
12    3.453170
dtype: float64
```

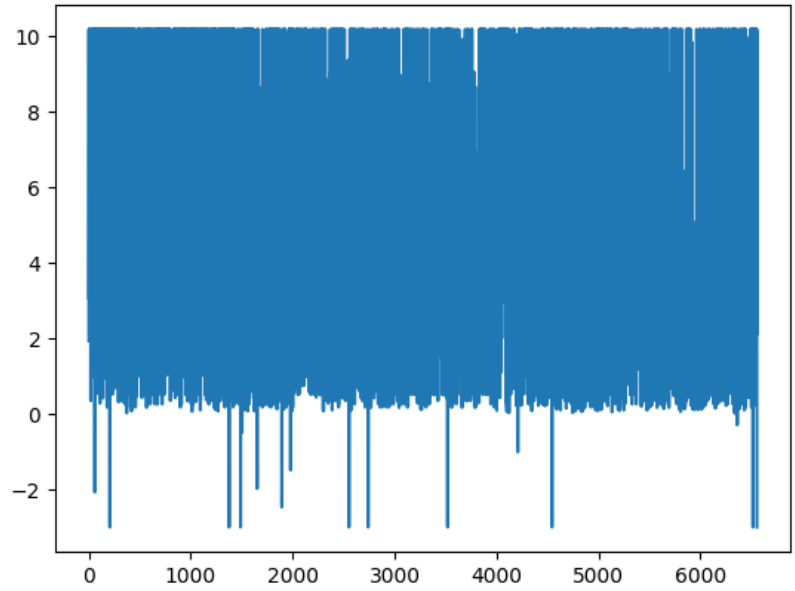
Data-driven approach

In [214]:

```
d1.Quantity.plot()
```

Out[214]:

```
<AxesSubplot: >
```



In [209]:

```
# Creating a function to calculate the MAPE value for test data
def MAPE(pred, actual):
    temp = np.abs((pred - actual)/actual)*100
    return np.mean(temp)

# Moving Average for the time series
mv_pred = d1["Quantity"].rolling(2).mean()
mv_pred.tail(1311)
MAPE(mv_pred.tail(1311), Test.Quantity)
```

Out[209]:

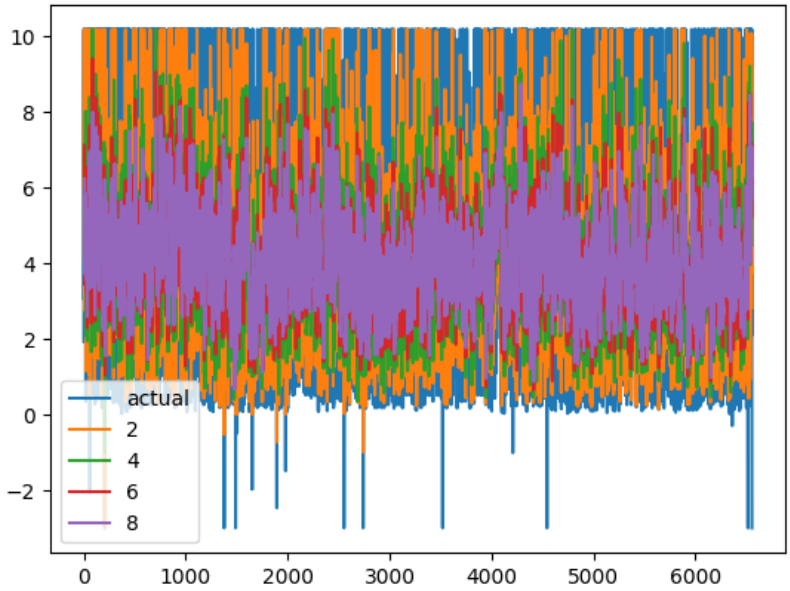
```
133.73035866678583
```

In [210]:

```
# Plot with Moving Averages
d1.Quantity.plot(label = "actual")
for i in range(2, 9, 2):
    d1["Quantity"].rolling(i).mean().plot(label = str(i))
plt.legend(loc = 3)
```

Out[210]:

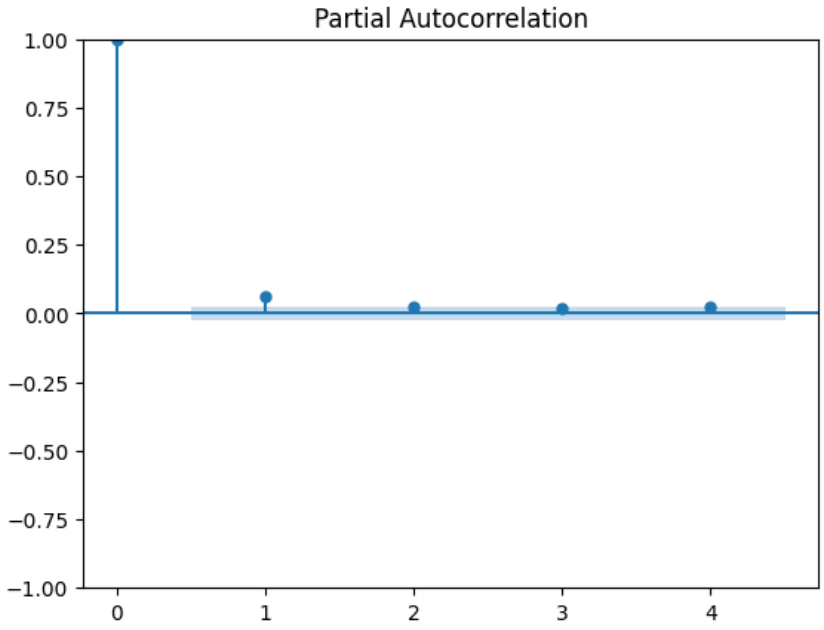
```
<matplotlib.legend.Legend at 0x25dc9a4f9d0>
```

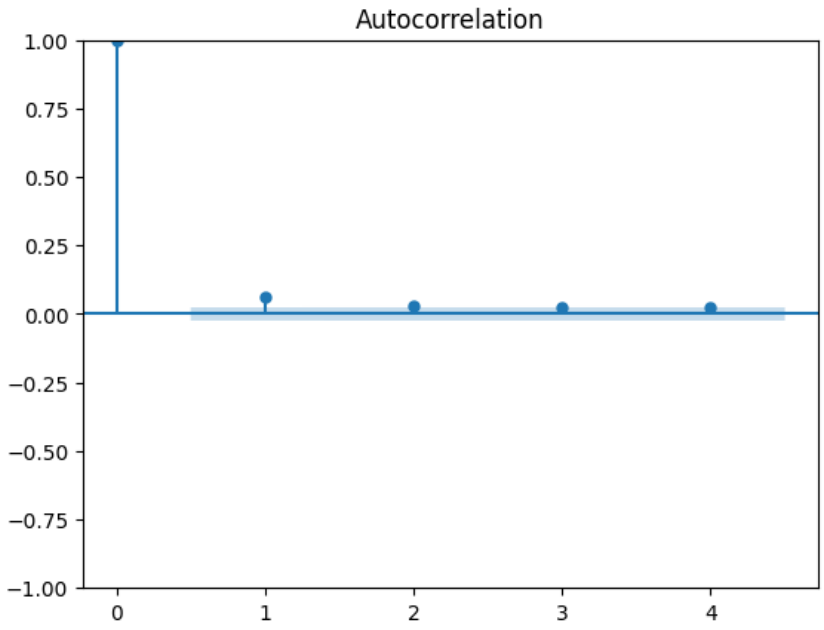


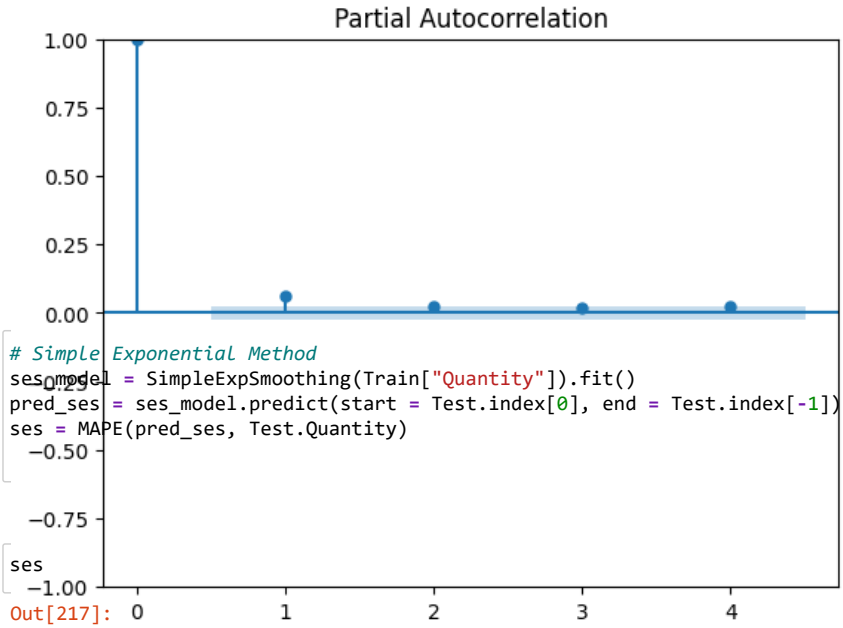
In [213]:

```
# ACF and PACF plot on Original data sets
import statsmodels.graphics.tsaplots as tsa_plots
tsa_plots.plot_acf(d1.Quantity, lags = 4)
tsa_plots.plot_pacf(d1.Quantity, lags = 4)
# ACF is an (complete) auto-correlation function gives values
# of auto-correlation of any time series with its lagged values.

# PACF is a partial auto-correlation function.
# It finds correlations of present with lags of the residuals of the time series
```

Out[213]:








```
# Simple Exponential Method
ses_model = SimpleExpSmoothing(Train["Quantity"]).fit()
pred_ses = ses_model.predict(start = Test.index[0], end = Test.index[-1])
ses = MAPE(pred_ses, Test.Quantity)
```

```
ses
```

Out[217]:

```
246.36357532364934
```

In [221]:

```
# Holt method
hw_model = Holt(Train["Quantity"]).fit()
pred_hw = hw_model.predict(start = Test.index[0], end = Test.index[-1])
hw = MAPE(pred_hw, Test.Quantity)
hw
```

Out[221]:

```
23321.422920668403
```

In [222]:

```
# Holts winter exponential smoothing with additive seasonality and additive trend
hwe_model_add_add = ExponentialSmoothing(Train["Quantity"], seasonal = "add", trend = "add", seasonal_periods = 4).fit()
pred_hwe_add_add = hwe_model_add_add.predict(start = Test.index[0], end = Test.index[-1])
hwe = MAPE(pred_hwe_add_add, Test.Quantity)
hwe
```

Out[222]:

```
235.66170926654524
```

In [224]:

```
# comparing all mape's
di = pd.Series({'Simple Exponential Method':ses, 'Holt method ':hw, 'hw_additive seasonality and additive trend':hwe})
mape = pd.DataFrame(di, columns=['mape'])
mape
```

Out[224]:

	mape
Simple Exponential Method	246.363575
Holt method	23321.422921
hw_additive seasonality and additive trend	235.661709

In [226]:

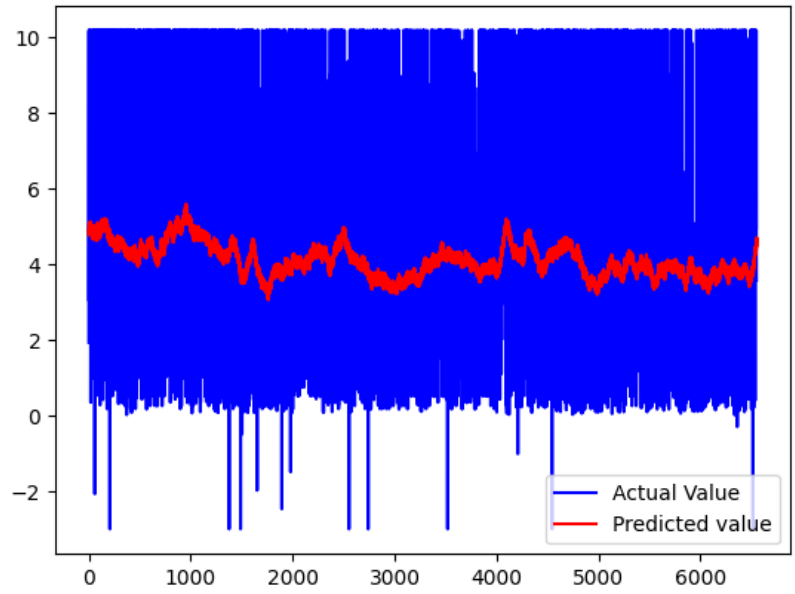
```
# Final Model on 100% Data
hwe_model_add_add = ExponentialSmoothing(d1["Quantity"], seasonal = "add", trend = "add", seasonal_periods = 4).fit()

# The models and results instances all have a save and load method, so you don't need to use the pickle module directly.
# to save model
hwe_model_add_add.save("model2.pickle")
```

In [228]:

```
import os
os.getcwd()

# to load model
from statsmodels.regression.linear_model import OLSResults
model_2 = OLSResults.load("model2.pickle")

############
```

In [231]:

```
# Load the new data which includes the entry for future 4 values
new_data = pd.read_csv(r"C:\Users\MsK_PC\360digiTMG\Data_Science\DS_TMT_Steel_Project\Client_data\Data_driven.csv")

newdata_pred = model_2.predict(start = new_data.index[0], end = new_data.index[-1])
newdata_pred

fig, ax = plt.subplots()
ax.plot(new_data.Quantity, '-b', label = 'Actual Value')
ax.plot(newdata_pred, '-r', label = 'Predicted value')
ax.legend();
plt.show()

##########
```



In [ ]:

In [ ]: