Preprocessing

```
!pip install dataprep
```

```
import numpy as np
import pandas as pd
from datetime import datetime, timedelta
from sklearn.preprocessing import OrdinalEncoder
# import dataprep.eda.create_report as report
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv("Copper_Set_Result.csv")
df
```

```
# verify the number of unique values in each features

for i in list(df.columns):
    print(f"{i}:{df[i].nunique()}")
```

```
id:84139
item_date:109
quantity tons:84140
customer:1004
country:17
status:9
item type:7
application:30
thickness:496
width:1179
material_ref:11928
product_ref:31
delivery date:25
selling_price:5152
```

```
# verify datatypes of all features
df.dtypes
```

```
id                object
item_date          int64
quantity tons    float64
customer           int64
country            int64
status            object
item type         object
application        int64
thickness        float64
width            float64
material_ref      object
product_ref        int64
delivery date    float64
selling_price    float64
dtype: object
```

```
# convert the data type from object to numeric

df['quantity tons'] = pd.to_numeric(df['quantity tons'], errors='coerce')
df['item_date_1'] = pd.to_datetime(df['item_date'], format='%Y%m%d', errors='coerce').dt.date
df['delivery date_1'] = pd.to_datetime(df['delivery date'], format='%Y%m%d', errors='coerce').dt.date
df.head(3)
```

	id	item_date	quantity tons	customer	country	status	item type	application	thickness	width	
0	EC06F063-9DF0-440C-8764-0B0C05A4F6AE	20210401	54.151139	30156308	28	Won	W	10	2.00	1500.0	
1	4E5F4B3D-DDDF-499D-AFDE-A3227EC49425	20210401	768.024839	30202938	25	Won	W	41	0.80	1210.0	0000000000000000000000000(
2	E140FF1B-2407-4C02-A0DD-780A093B1158	20210401	386.127949	30153963	30	Won	WI	28	0.38	952.0	

```
# check any null values in data
df.isnull().sum()
```

```
id                   1
item_date            0
quantity tons        0
customer             0
country              0
status               0
item type            0
application          0
thickness            0
width                0
material_ref     36095
product_ref          0
delivery date        1
selling_price        1
item_date_1          1
delivery date_1      2
dtype: int64
```

```
# Some rubbish values are present in ‘Material_ref’ which starts with ‘00000’ value which should be converted into null

df['material_ref'] = df['material_ref'].apply(lambda x: np.nan if str(x).startswith('00000') else x)
df.head(3)
```

	id	item_date	quantity tons	customer	country	status	item type	application	thickness	width	material_ref	product_ref
0	EC06F063-9DF0-440C-8764-0B0C05A4F6AE	20210401	54.151139	30156308	28	Won	W	10	2.00	1500.0	DEQ1 S460MC	1670798778
1	4E5F4B3D-DDDF-499D-AFDE-A3227EC49425	20210401	768.024839	30202938	25	Won	W	41	0.80	1210.0	NaN	1668701718
2	E140FF1B-2407-4C02-A0DD-780A093B1158	20210401	386.127949	30153963	30	Won	WI	28	0.38	952.0	S0380700	628377

```
# check null values for all features
df.isnull().sum()
```

```
id                   1
item_date            0
quantity tons        0
customer             0
country              0
status               0
item type            0
application          0
thickness            0
width                0
material_ref     46350
product_ref          0
delivery date        1
selling_price        1
item_date_1          1
delivery date_1      2
dtype: int64
```

```
# material ref have more than 55% are null values and id have all are unique values. so we have drop both columns.

df.drop(columns=['id','material_ref'], inplace=True)
df
```

	item_date	quantity tons	customer	country	status	item type	application	thickness	width	product_ref	delivery date	selling_price
0	20210401	54.151139	30156308	28	Won	W	10	2.00	1500.0	1670798778	20210701.0	854.00
1	20210401	768.024839	30202938	25	Won	W	41	0.80	1210.0	1668701718	20210401.0	1047.00
2	20210401	386.127949	30153963	30	Won	WI	28	0.38	952.0	628377	20210101.0	644.33
3	20210401	202.411065	30349574	32	Won	S	59	2.30	1317.0	1668701718	20210101.0	768.00
4	20210401	785.526262	30211560	28	Won	W	10	4.00	2000.0	640665	20210301.0	577.00
...	...	...	...	...	...	...	...	...	...	...	...	...
84135	20201207	5.511658	30205658	32	Won	W	10	1.20	1180.0	611993	20210401.0	916.00
84136	20201207	4.424904	30205658	32	Won	W	10	0.50	1000.0	611993	20210401.0	1008.00
84137	20201207	9.326179	30205658	32	Won	W	10	0.70	1000.0	611993	20210401.0	976.00
84138	20201207	28.795410	30201589	84	Won	S	15	8.00	1470.0	640405	20210101.0	1025.00
84139	20201207	0.707309	30205658	32	Won	W	10	1.20	1180.0	6	NaN	NaN

84140 rows × 14 columns

```
df.describe().T
```

	count	mean	std	min	25%	
item_date	84140.0	2.020880e+07	3.402485e+03	1.995000e+07	2.021011e+07	2.021021
quantity tons	84140.0	9.741604e+01	3.980261e+02	1.867763e-03	9.933240e+00	2.994542
customer	84140.0	3.023196e+07	1.262725e+05	1.245800e+04	3.016598e+07	3.020519
country	84140.0	4.478980e+01	2.435805e+01	2.500000e+01	2.600000e+01	3.000000
application	84140.0	2.576099e+01	1.741726e+01	2.000000e+00	1.000000e+01	1.500000
thickness	84140.0	2.561124e+00	9.137542e+00	1.800000e-01	7.000000e-01	1.500000
width	84140.0	1.298919e+03	2.555994e+02	1.000000e+00	1.180000e+03	1.250000
product_ref	84140.0	4.888029e+08	7.279335e+08	6.000000e+00	6.119930e+05	6.406650
delivery date	84139.0	2.021058e+07	3.482334e+04	2.019040e+07	2.021040e+07	2.021040

```
# quantity and selling price values are not below 0. so we convert to null for below 0 values.

df['quantity tons'] = df['quantity tons'].apply(lambda x: np.nan if x<=0 else x)
df['selling_price'] = df['selling_price'].apply(lambda x: np.nan if x<=0 else x)
df.describe().T
```

	count	mean	std	min	25%	
item_date	84140.0	2.020880e+07	3.402485e+03	1.995000e+07	2.021011e+07	2.021021e
quantity tons	84140.0	9.741604e+01	3.980261e+02	1.867763e-03	9.933240e+00	2.994542e
customer	84140.0	3.023196e+07	1.262725e+05	1.245800e+04	3.016598e+07	3.020519e
country	84140.0	4.478980e+01	2.435805e+01	2.500000e+01	2.600000e+01	3.000000e
application	84140.0	2.576099e+01	1.741726e+01	2.000000e+00	1.000000e+01	1.500000e
thickness	84140.0	2.561124e+00	9.137542e+00	1.800000e-01	7.000000e-01	1.500000e
width	84140.0	1.298919e+03	2.555994e+02	1.000000e+00	1.180000e+03	1.250000e
product_ref	84140.0	4.888029e+08	7.279335e+08	6.000000e+00	6.119930e+05	6.406650e
delivery date	84139.0	2.021058e+07	3.482334e+04	2.019040e+07	2.021040e+07	2.021040e

```
# check null values for all features
df.isnull().sum()
```

```
item_date          0
quantity tons      0
customer           0
country            0
status             0
item type          0
application        0
thickness          0
width              0
product_ref        0
delivery date      1
selling_price      7
item_date_1        1
delivery date_1    2
dtype: int64
```

```
# Handling null values using median and mode
# median - middle value in dataset (asc/desc), mode - value that appears most frequently in dataset

# object datatype using mode
df['item_date'].fillna(df['item_date'].mode().iloc[0], inplace=True)
df['item_date_1'].fillna(df['item_date_1'].mode().iloc[0], inplace=True)
df['status'].fillna(df['status'].mode().iloc[0], inplace=True)
df['delivery date'].fillna(df['delivery date'].mode().iloc[0], inplace=True)
df['delivery date_1'].fillna(df['delivery date_1'].mode().iloc[0], inplace=True)

#numerical datatype using median
df['quantity tons'].fillna(df['quantity tons'].median(), inplace=True)
df['customer'].fillna(df['customer'].median(), inplace=True)
df['country'].fillna(df['country'].median(), inplace=True)
df['application'].fillna(df['application'].median(), inplace=True)
df['thickness'].fillna(df['thickness'].median(), inplace=True)
df['selling_price'].fillna(df['selling_price'].median(), inplace=True)
```

```
df.isnull().sum()
```

```
item_date          0
quantity tons      0
customer           0
country            0
status             0
item type          0
application        0
thickness          0
width              0
product_ref        0
delivery date      0
selling_price      0
item_date_1        0
delivery date_1    0
dtype: int64
```

```
df['status'].unique()
```

```
array(['Won', 'Draft', 'To be approved', 'Lost', 'Not lost for AM',
       'Wonderful', 'Revised', 'Offered', 'Offerable'], dtype=object)
```

```
df['item type'].unique()
```

```
array(['W', 'WI', 'S', 'Others', 'PL', 'IPL', 'SLAWR'], dtype=object)
```

```
# convert categorical data into numerical data - using map and ordinal encoder methods

df['status'] = df['status'].map({'Lost':0, 'Won':1, 'Draft':2, 'To be approved':3, 'Not lost for AM':4,
                                 'Wonderful':5, 'Revised':6, 'Offered':7, 'Offerable':8})
df['item type'] = OrdinalEncoder().fit_transform(df[['item type']])
df
```

	item_date	quantity tons	customer	country	status	item type	application	thickness	width	product_ref	delivery date	selling_price
0	20210401	54.151139	30156308	28	1	5.0	10	2.00	1500.0	1670798778	20210701.0	854.00
1	20210401	768.024839	30202938	25	1	5.0	41	0.80	1210.0	1668701718	20210401.0	1047.00
2	20210401	386.127949	30153963	30	1	6.0	28	0.38	952.0	628377	20210101.0	644.33
3	20210401	202.411065	30349574	32	1	3.0	59	2.30	1317.0	1668701718	20210101.0	768.00
4	20210401	785.526262	30211560	28	1	5.0	10	4.00	2000.0	640665	20210301.0	577.00
...	...	...	...	...	...	...	...	...	...	...	...	...
84135	20201207	5.511658	30205658	32	1	5.0	10	1.20	1180.0	611993	20210401.0	916.00
84136	20201207	4.424904	30205658	32	1	5.0	10	0.50	1000.0	611993	20210401.0	1008.00
84137	20201207	9.326179	30205658	32	1	5.0	10	0.70	1000.0	611993	20210401.0	976.00
84138	20201207	28.795410	30201589	84	1	3.0	15	8.00	1470.0	640405	20210101.0	1025.00
84139	20201207	0.707309	30205658	32	1	5.0	10	1.20	1180.0	6	20210401.0	927.00

84140 rows × 14 columns

```
# array(['W', 'WI', 'S', 'Others', 'PL', 'IPL', 'SLAWR'], dtype=object)
df['item type'].unique()
```

```
array([5., 6., 3., 1., 2., 0., 4.])
```

```
# final verification of null values after encoding
df.isnull().sum()
```

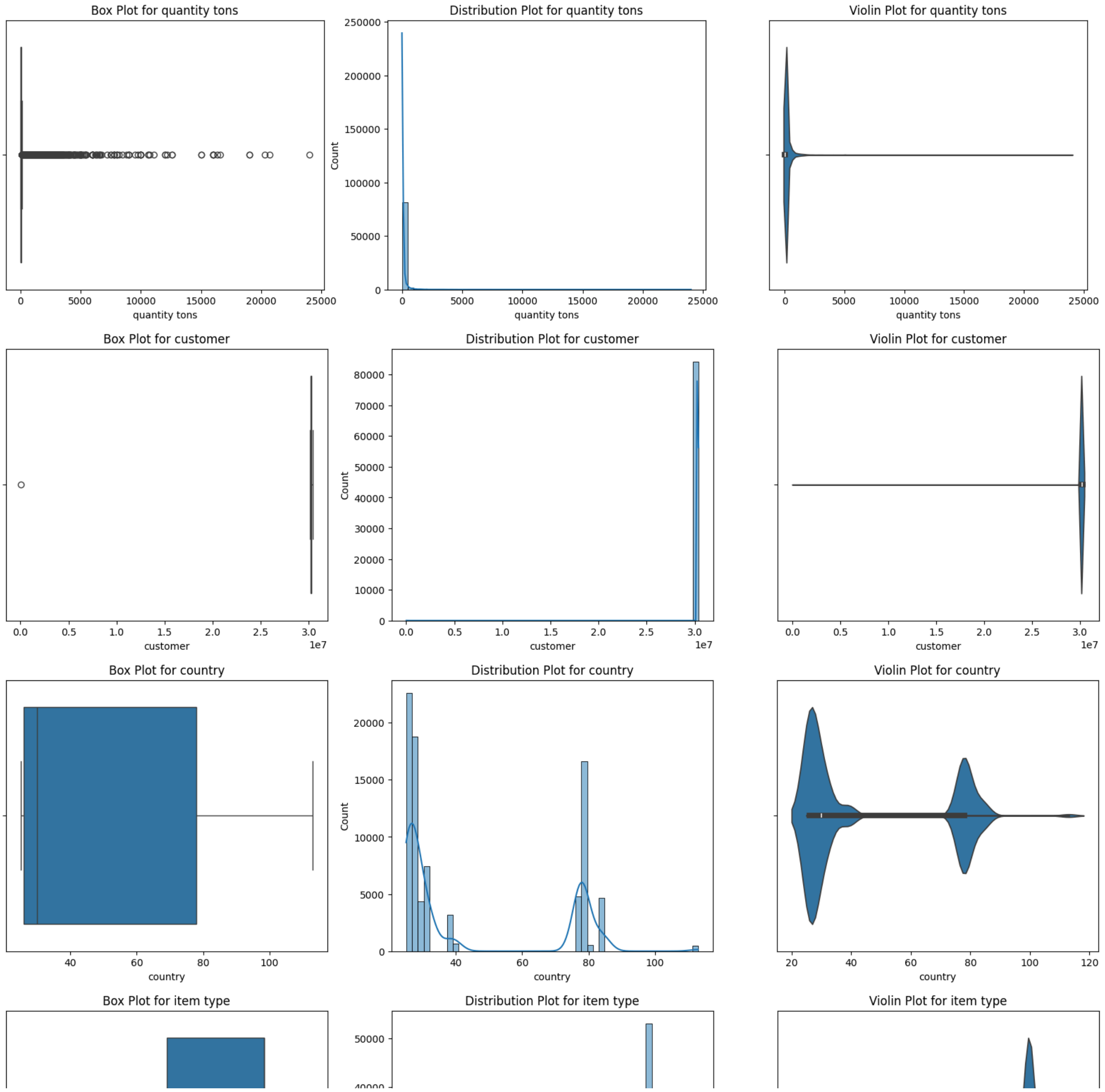
```
item_date          0
quantity tons      0
customer           0
country            0
status             0
item type          0
application        0
thickness          0
width              0
product_ref        0
delivery date      0
selling_price      0
item_date_1        0
delivery date_1    0
dtype: int64
```

```
df.describe().T
```

	count	mean	std	min	25%	
item_date	84140.0	2.020880e+07	3.402485e+03	1.995000e+07	2.021011e+07	2.021021e
quantity tons	84140.0	9.741604e+01	3.980261e+02	1.867763e-03	9.933240e+00	2.994542e
customer	84140.0	3.023196e+07	1.262725e+05	1.245800e+04	3.016598e+07	3.020519e
country	84140.0	4.478980e+01	2.435805e+01	2.500000e+01	2.600000e+01	3.000000e
status	84140.0	1.315023e+00	1.264524e+00	0.000000e+00	1.000000e+00	1.000000e
item type	84140.0	4.225493e+00	1.058621e+00	0.000000e+00	3.000000e+00	5.000000e
application	84140.0	2.576099e+01	1.741726e+01	2.000000e+00	1.000000e+01	1.500000e
thickness	84140.0	2.561124e+00	9.137542e+00	1.800000e-01	7.000000e-01	1.500000e
width	84140.0	1.298919e+03	2.555994e+02	1.000000e+00	1.180000e+03	1.250000e
product_ref	84140.0	4.888029e+08	7.279335e+08	6.000000e+00	6.119930e+05	6.406650e
delivery date	84140.0	2.021058e+07	3.482313e+04	2.019040e+07	2.021040e+07	2.021040e

Skewness Handling - Feature Scaling (Log Transformation)

```
# find outliers - box plot & skewed data - hist plot and violin plot

def plot(df, column):
    plt.figure(figsize=(20,5))
    plt.subplot(1,3,1)
    sns.boxplot(data=df, x=column)
    plt.title(f'Box Plot for {column}')

    plt.subplot(1,3,2)
    sns.histplot(data=df, x=column, kde=True, bins=50)
    plt.title(f'Distribution Plot for {column}')

    plt.subplot(1,3,3)
    sns.violinplot(data=df, x=column)
    plt.title(f'Violin Plot for {column}')
    plt.show()


for i in ['quantity tons', 'customer', 'country', 'item type', 'application', 'thickness', 'width', 'selling_price']:
    plot(df, i)
```

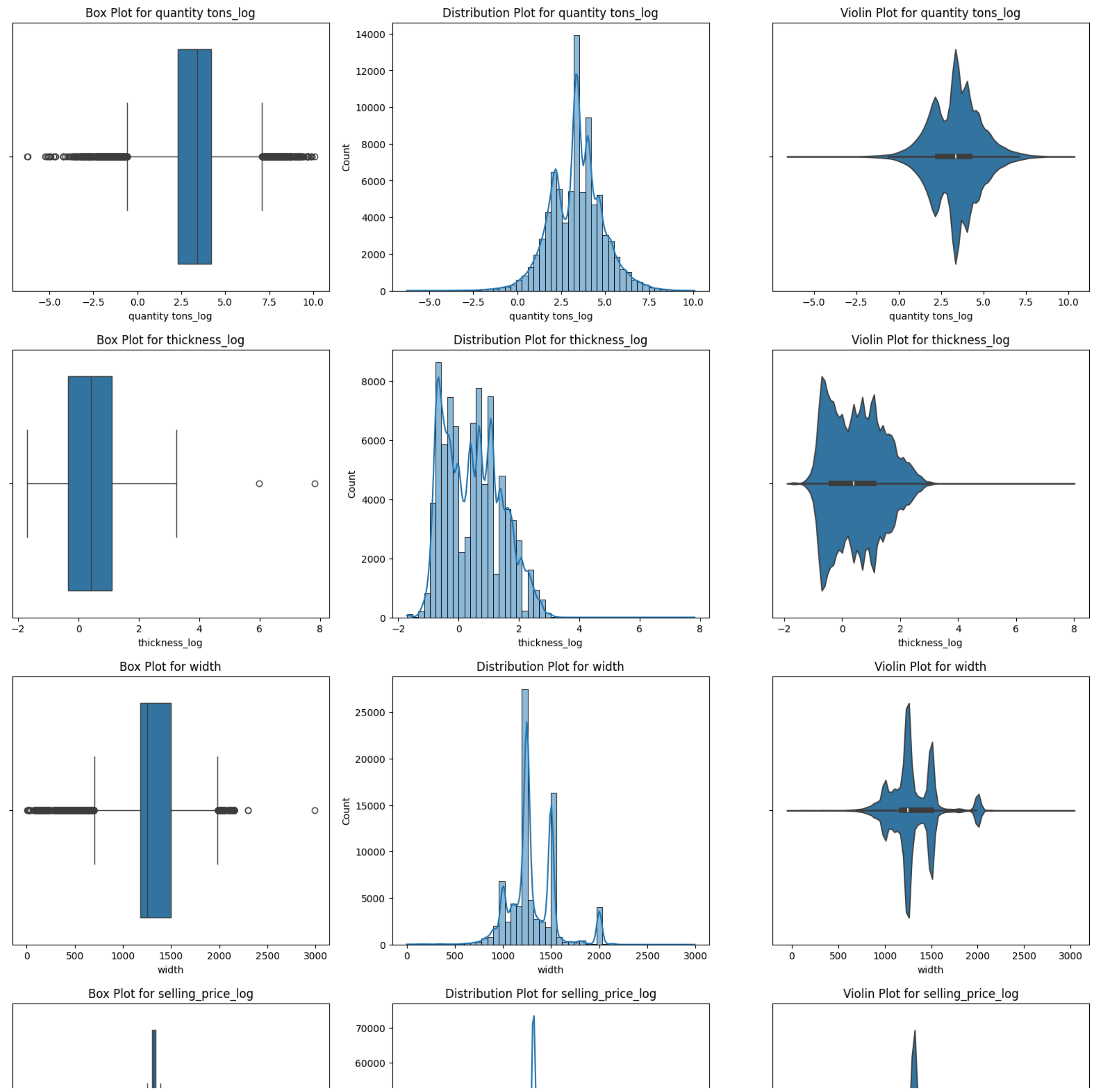



```
# quantity tons, thickness and selling price data are skewd. so using the log transformation method to handle the skewness data

df1 = df.copy()
df1['quantity tons_log'] = np.log(df1['quantity tons'])
df1['thickness_log'] = np.log(df1['thickness'])
df1['selling_price_log'] = np.log(df1['selling_price'])
df1
```

	item_date	quantity tons	customer	country	status	item type	application	thickness	width	product_ref	delivery date	selling_price
0	20210401	54.151139	30156308	28	1	5.0	10	2.00	1500.0	1670798778	20210701.0	854.00
1	20210401	768.024839	30202938	25	1	5.0	41	0.80	1210.0	1668701718	20210401.0	1047.00
2	20210401	386.127949	30153963	30	1	6.0	28	0.38	952.0	628377	20210101.0	644.33
3	20210401	202.411065	30349574	32	1	3.0	59	2.30	1317.0	1668701718	20210101.0	768.00
4	20210401	785.526262	30211560	28	1	5.0	10	4.00	2000.0	640665	20210301.0	577.00
...	...	...	...	...	...	...	...	...	...	...	...	...
84135	20201207	5.511658	30205658	32	1	5.0	10	1.20	1180.0	611993	20210401.0	916.00
84136	20201207	4.424904	30205658	32	1	5.0	10	0.50	1000.0	611993	20210401.0	1008.00
84137	20201207	9.326179	30205658	32	1	5.0	10	0.70	1000.0	611993	20210401.0	976.00
84138	20201207	28.795410	30201589	84	1	3.0	15	8.00	1470.0	640405	20210101.0	1025.00
84139	20201207	0.707309	30205658	32	1	5.0	10	1.20	1180.0	6	20210401.0	927.00

84140 rows × 17 columns

```
# after log transformation the data are normally distributed and reduced the skewness. [hist plot and violin plot]
for i in ['quantity tons_log', 'thickness_log', 'width', 'selling_price_log']:
    plot(df1, i)
```



Outliers Handling - Interquartile Range (IQR) method

```
df2 = df1.copy()
df2
```

	item_date	quantity tons	customer	country	status	item type	application	thickness	width	product_ref	delivery date	selling_price
0	20210401	54.151139	30156308	28	1	5.0	10	2.00	1500.0	1670798778	20210701.0	854.00
1	20210401	768.024839	30202938	25	1	5.0	41	0.80	1210.0	1668701718	20210401.0	1047.00
2	20210401	386.127949	30153963	30	1	6.0	28	0.38	952.0	628377	20210101.0	644.33
3	20210401	202.411065	30349574	32	1	3.0	59	2.30	1317.0	1668701718	20210101.0	768.00
4	20210401	785.526262	30211560	28	1	5.0	10	4.00	2000.0	640665	20210301.0	577.00
...	...	...	...	...	...	...	...	...	...	...	...	...
84135	20201207	5.511658	30205658	32	1	5.0	10	1.20	1180.0	611993	20210401.0	916.00
84136	20201207	4.424904	30205658	32	1	5.0	10	0.50	1000.0	611993	20210401.0	1008.00
84137	20201207	9.326179	30205658	32	1	5.0	10	0.70	1000.0	611993	20210401.0	976.00
84138	20201207	28.795410	30201589	84	1	3.0	15	8.00	1470.0	640405	20210101.0	1025.00
84139	20201207	0.707309	30205658	32	1	5.0	10	1.20	1180.0	6	20210401.0	927.00

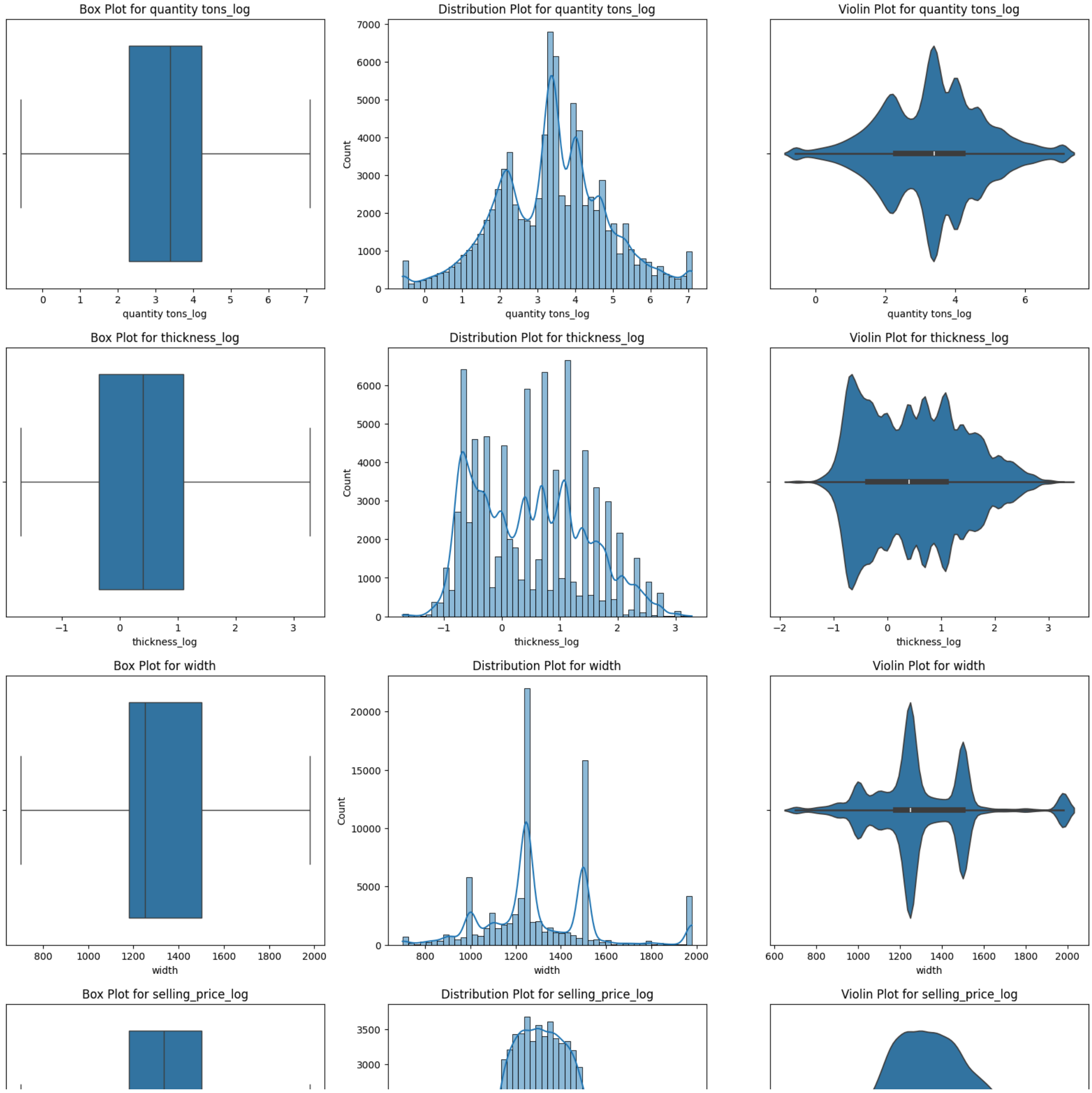
84140 rows × 17 columns

```
# Using IQR and clip() methods to handle the outliers and add a new column of dataframe

def outlier(df, column):
    iqr = df[column].quantile(0.75) - df[column].quantile(0.25)
    upper_threshold = df[column].quantile(0.75) + (1.5*iqr)
    lower_threshold = df[column].quantile(0.25) - (1.5*iqr)
    df[column] = df[column].clip(lower_threshold, upper_threshold)


# (Ex: lower threshold = 5 and upper threshold = 20)
# above upper threshold values (>20) are converted to upper threshold value (20) in features
# below lower threshold values (<5)  are converted to lower threshold value (5)  in features

outlier(df2, 'quantity tons_log')
outlier(df2, 'thickness_log')
outlier(df2, 'selling_price_log')
outlier(df2, 'width')
df2
```

	item_date	quantity tons	customer	country	status	item type	application	thickness	width	product_ref	delivery date	selling_price
0	20210401	54.151139	30156308	28	1	5.0	10	2.00	1500.0	1670798778	20210701.0	854.00
1	20210401	768.024839	30202938	25	1	5.0	41	0.80	1210.0	1668701718	20210401.0	1047.00
2	20210401	386.127949	30153963	30	1	6.0	28	0.38	952.0	628377	20210101.0	644.33
3	20210401	202.411065	30349574	32	1	3.0	59	2.30	1317.0	1668701718	20210101.0	768.00
4	20210401	785.526262	30211560	28	1	5.0	10	4.00	1980.0	640665	20210301.0	577.00
...	...	...	...	...	...	...	...	...	...	...	...	...
84135	20201207	5.511658	30205658	32	1	5.0	10	1.20	1180.0	611993	20210401.0	916.00
84136	20201207	4.424904	30205658	32	1	5.0	10	0.50	1000.0	611993	20210401.0	1008.00
84137	20201207	9.326179	30205658	32	1	5.0	10	0.70	1000.0	611993	20210401.0	976.00
84138	20201207	28.795410	30201589	84	1	3.0	15	8.00	1470.0	640405	20210101.0	1025.00
84139	20201207	0.707309	30205658	32	1	5.0	10	1.20	1180.0	6	20210401.0	927.00

84140 rows × 17 columns

```
# transform the outliers to within range using IQR and clip() methods - box plot

for i in ['quantity tons_log', 'thickness_log', 'width', 'selling_price_log']:
    plot(df2, i)
```




```
df2.describe().T
```

	count	mean	std	min	25%	
item_date	84140.0	2.020880e+07	3.402485e+03	1.995000e+07	2.021011e+07	2.02
quantity tons	84140.0	9.741604e+01	3.980261e+02	1.867763e-03	9.933240e+00	2.994
customer	84140.0	3.023196e+07	1.262725e+05	1.245800e+04	3.016598e+07	3.02(
country	84140.0	4.478980e+01	2.435805e+01	2.500000e+01	2.600000e+01	3.00(
status	84140.0	1.315023e+00	1.264524e+00	0.000000e+00	1.000000e+00	1.00(
item type	84140.0	4.225493e+00	1.058621e+00	0.000000e+00	3.000000e+00	5.00(
application	84140.0	2.576099e+01	1.741726e+01	2.000000e+00	1.000000e+01	1.50(
thickness	84140.0	2.561124e+00	9.137542e+00	1.800000e-01	7.000000e-01	1.50(
width	84140.0	1.299520e+03	2.458421e+02	7.000000e+02	1.180000e+03	1.25(
product_ref	84140.0	4.888029e+08	7.279335e+08	6.000000e+00	6.119930e+05	6.40(
delivery date	84140.0	2.021058e+07	3.482313e+04	2.019040e+07	2.021040e+07	2.02
selling_price	84140.0	3.308676e+03	4.875413e+05	1.000000e-01	8.300000e+02	9.27(
quantity tons_log	84140.0	3.365730e+00	1.442399e+00	-5.868095e-01	2.295887e+00	3.39
thickness_log	84140.0	4.783848e-01	9.324948e-01	-1.714798e+00	-3.566749e-	4.05

```
# after add the new column of 'quantity tons_log', 'thickness_log', 'selling_price_log', drop the existing columns
df3 = df2.drop(columns=['quantity tons', 'thickness', 'selling_price'])
df3
```

	item_date	customer	country	status	item type	application	width	product_ref	delivery date	item_date_1	delivery date_1	quantity tons_log	thi
0	20210401	30156308	28	1	5.0	10	1500.0	1670798778	20210701.0	2021-04-01	2021-07-01	3.991779	
1	20210401	30202938	25	1	5.0	41	1210.0	1668701718	20210401.0	2021-04-01	2021-04-01	6.643822	
2	20210401	30153963	30	1	6.0	28	952.0	628377	20210101.0	2021-04-01	2021-01-01	5.956169	
3	20210401	30349574	32	1	3.0	59	1317.0	1668701718	20210101.0	2021-04-01	2021-01-01	5.310301	
4	20210401	30211560	28	1	5.0	10	1980.0	640665	20210301.0	2021-04-01	2021-03-01	6.666354	
...	...	...	...	...	...	...	...	...	...	...	...	...	
84135	20201207	30205658	32	1	5.0	10	1180.0	611993	20210401.0	2020-12-07	2021-04-01	1.706866	
84136	20201207	30205658	32	1	5.0	10	1000.0	611993	20210401.0	2020-12-07	2021-04-01	1.487249	
84137	20201207	30205658	32	1	5.0	10	1000.0	611993	20210401.0	2020-12-07	2021-04-01	2.232825	
84138	20201207	30201589	84	1	3.0	15	1470.0	640405	20210101.0	2020-12-07	2021-01-01	3.360216	
84139	20201207	30205658	32	1	5.0	10	1180.0	6	20210401.0	2020-12-07	2021-04-01	-0.346288	

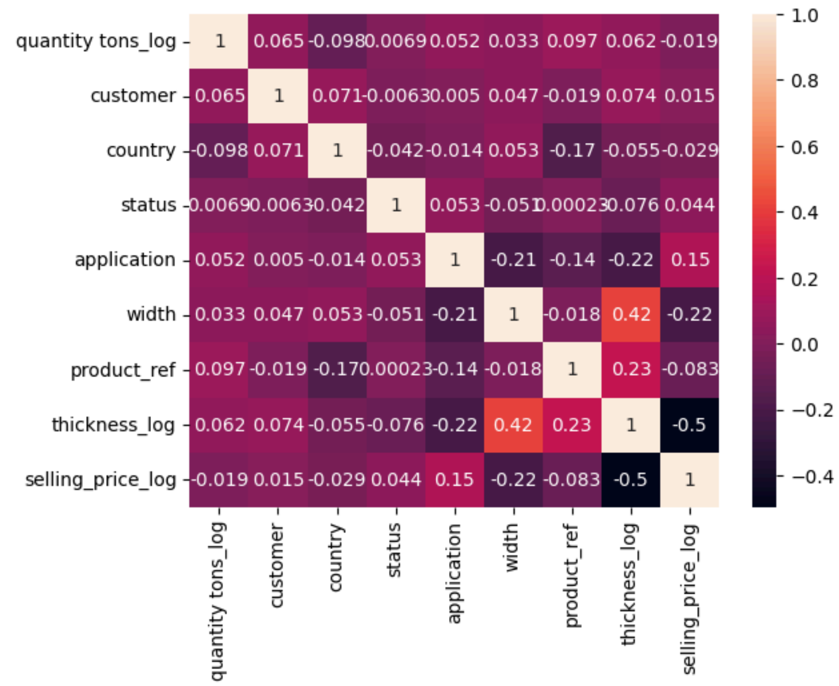
84140 rows × 14 columns

```
# check the data types
df3.dtypes
```

```
item_date              int64
customer               int64
country                int64
status                 int64
item type            float64
application            int64
width                float64
product_ref            int64
delivery date        float64
item_date_1           object
delivery date_1       object
quantity tons_log    float64
thickness_log        float64
selling_price_log    float64
dtype: object
```

```
# Need to verify any columns are highly correlated using Heatmap. If any columns correalaion value >= 0.7 (absolute value), drop the col

col = ['quantity tons_log','customer','country','status','application','width','product_ref','thickness_log','selling_price_log']
df_heatmap = df3[col].corr()
sns.heatmap(df_heatmap, annot=True)
```

```
<Axes: >
```




```
# The highest value is (0.4 or -0.42) only, So there is no columns are highly correlated and no need to drop any columns.
```

Wrong Delivery Date Handling

```
df4 = df3.copy()
df4
```

	item_date	customer	country	status	item type	application	width	product_ref	delivery date	item_date_1	delivery date_1	quantity tons_log	thi
0	20210401	30156308	28	1	5.0	10	1500.0	1670798778	20210701.0	2021-04-01	2021-07-01	3.991779	
1	20210401	30202938	25	1	5.0	41	1210.0	1668701718	20210401.0	2021-04-01	2021-04-01	6.643822	
2	20210401	30153963	30	1	6.0	28	952.0	628377	20210101.0	2021-04-01	2021-01-01	5.956169	
3	20210401	30349574	32	1	3.0	59	1317.0	1668701718	20210101.0	2021-04-01	2021-01-01	5.310301	
4	20210401	30211560	28	1	5.0	10	1980.0	640665	20210301.0	2021-04-01	2021-03-01	6.666354	
...	...	...	...	...	...	...	...	...	...	...	...	...	
84135	20201207	30205658	32	1	5.0	10	1180.0	611993	20210401.0	2020-12-07	2021-04-01	1.706866	
84136	20201207	30205658	32	1	5.0	10	1000.0	611993	20210401.0	2020-12-07	2021-04-01	1.487249	
84137	20201207	30205658	32	1	5.0	10	1000.0	611993	20210401.0	2020-12-07	2021-04-01	2.232825	
84138	20201207	30201589	84	1	3.0	15	1470.0	640405	20210101.0	2020-12-07	2021-01-01	3.360216	
84139	20201207	30205658	32	1	5.0	10	1180.0	6	20210401.0	2020-12-07	2021-04-01	-0.346288	

84140 rows × 14 columns

```
# The 'delivery date' is previous date of 'item date'. so this is impossible. delivery date is always greater.
```

```
# find the difference between item and delivery date and add the new column of dataframe

df4['Date_difference'] = (df4['delivery date_1'] - df4['item_date_1']).dt.days
df4.head()
```

	item_date	customer	country	status	item type	application	width	product_ref	delivery date	item_date_1	delivery date_1	quantity tons_log	thicknes
0	20210401	30156308	28	1	5.0	10	1500.0	1670798778	20210701.0	2021-04-01	2021-07-01	3.991779	0.6
1	20210401	30202938	25	1	5.0	41	1210.0	1668701718	20210401.0	2021-04-01	2021-04-01	6.643822	-0.2
2	20210401	30153963	30	1	6.0	28	952.0	628377	20210101.0	2021-04-01	2021-01-01	5.956169	-0.9
3	20210401	30349574	32	1	3.0	59	1317.0	1668701718	20210101.0	2021-04-01	2021-01-01	5.310301	0.8
4	20210401	30211560	28	1	5.0	10	1980.0	640665	20210301.0	2021-04-01	2021-03-01	6.666354	1.3

```
# convert the data type using pandas
df4['item_date_1'] = pd.to_datetime(df4['item_date_1'])

# split the day, month, and year from 'item_date_1' column and add dataframe (This data also help us to prediction)
df4['item_date_day'] = df4['item_date_1'].dt.day
df4['item_date_month'] = df4['item_date_1'].dt.month
df4['item_date_year'] = df4['item_date_1'].dt.year
df4
```

	item_date	customer	country	status	item type	application	width	product_ref	delivery date	item_date_1	delivery date_1	quantity tons_log	thi
0	20210401	30156308	28	1	5.0	10	1500.0	1670798778	20210701.0	2021-04-01	2021-07-01	3.991779	
1	20210401	30202938	25	1	5.0	41	1210.0	1668701718	20210401.0	2021-04-01	2021-04-01	6.643822	
2	20210401	30153963	30	1	6.0	28	952.0	628377	20210101.0	2021-04-01	2021-01-01	5.956169	
3	20210401	30349574	32	1	3.0	59	1317.0	1668701718	20210101.0	2021-04-01	2021-01-01	5.310301	
4	20210401	30211560	28	1	5.0	10	1980.0	640665	20210301.0	2021-04-01	2021-03-01	6.666354	
...	...	...	...	...	...	...	...	...	...	...	...	...	
84135	20201207	30205658	32	1	5.0	10	1180.0	611993	20210401.0	2020-12-07	2021-04-01	1.706866	
84136	20201207	30205658	32	1	5.0	10	1000.0	611993	20210401.0	2020-12-07	2021-04-01	1.487249	
84137	20201207	30205658	32	1	5.0	10	1000.0	611993	20210401.0	2020-12-07	2021-04-01	2.232825	
84138	20201207	30201589	84	1	3.0	15	1470.0	640405	20210101.0	2020-12-07	2021-01-01	3.360216	
84139	20201207	30205658	32	1	5.0	10	1180.0	6	20210401.0	2020-12-07	2021-04-01	-0.346288	

84140 rows × 18 columns

```
# split the non-negative value of 'Date_difference' column in separate dataframe
df_f1 = df4[df4['Date_difference']>=0]

# after split, the index values are unordered. so need to reset the index to ascending order from 0
df_f1 = df_f1.reset_index(drop=True)
df_f1
```

	item_date	customer	country	status	item type	application	width	product_ref	delivery date	item_date_1	delivery date_1	quantity tons_log	thi
0	20210401	30156308	28	1	5.0	10	1500.0	1670798778	20210701.0	2021-04-01	2021-07-01	3.991779	
1	20210401	30202938	25	1	5.0	41	1210.0	1668701718	20210401.0	2021-04-01	2021-04-01	6.643822	
2	20210401	30202938	25	1	5.0	41	1265.0	1668701718	20210401.0	2021-04-01	2021-04-01	5.419608	
3	20210401	30209509	30	2	5.0	41	1125.0	611993	20210701.0	2021-04-01	2021-07-01	1.259203	
4	20210401	30341428	38	1	3.0	10	1275.0	1668701376	20210701.0	2021-04-01	2021-07-01	4.235147	
...	...	...	...	...	...	...	...	...	...	...	...	...	
81185	20201207	30205658	32	1	5.0	10	1180.0	611993	20210401.0	2020-12-07	2021-04-01	1.706866	
81186	20201207	30205658	32	1	5.0	10	1000.0	611993	20210401.0	2020-12-07	2021-04-01	1.487249	
81187	20201207	30205658	32	1	5.0	10	1000.0	611993	20210401.0	2020-12-07	2021-04-01	2.232825	
81188	20201207	30201589	84	1	3.0	15	1470.0	640405	20210101.0	2020-12-07	2021-01-01	3.360216	
81189	20201207	30205658	32	1	5.0	10	1180.0	6	20210401.0	2020-12-07	2021-04-01	-0.346288	

81190 rows × 18 columns

```
# split the negative value of 'Date_difference' column in another dataframe
df_f2 = df4[df4['Date_difference']<0]

# after split, the index values are unordered. so need to reset the index to ascending order from 0
df_f2 = df_f2.reset_index(drop=True)
df_f2
```

	item_date	customer	country	status	item type	application	width	product_ref	delivery date	item_date_1	delivery date_1	quantity tons_log	thic
0	20210401	30153963	30	1	6.0	28	952.0	628377	20210101.0	2021-04-01	2021-01-01	5.956169	
1	20210401	30349574	32	1	3.0	59	1317.0	1668701718	20210101.0	2021-04-01	2021-01-01	5.310301	
2	20210401	30211560	28	1	5.0	10	1980.0	640665	20210301.0	2021-04-01	2021-03-01	6.666354	
3	20210401	30342192	32	1	5.0	41	1220.0	611993	20210101.0	2021-04-01	2021-01-01	4.730808	
4	20210401	30342192	32	1	5.0	41	1220.0	611993	20210101.0	2021-04-01	2021-01-01	4.736160	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2945	20201207	30394817	78	1	2.0	10	1171.0	628377	20201201.0	2020-12-07	2020-12-01	4.618578	
2946	20201207	30394817	78	1	2.0	10	1510.0	628377	20201201.0	2020-12-07	2020-12-01	0.185946	
2947	20201207	30394817	78	1	2.0	10	920.0	628377	20201201.0	2020-12-07	2020-12-01	1.773963	
2948	20201207	30394817	78	1	2.0	10	1306.0	628377	20201201.0	2020-12-07	2020-12-01	4.549376	
2949	20201207	30394817	78	1	2.0	10	1150.0	628377	20201201.0	2020-12-07	2020-12-01	1.440539	

2950 rows × 18 columns

```
# These 16108 values 'delivery date' are lesser than 'item date'.
# First we need to train the ML model using correct 'delivery date' data (df_f1) and predict the 'Date_difference'(df_f2) using ML model
```

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
```

```
df_f1.columns
```

```
Index(['item_date', 'customer', 'country', 'status', 'item type',
       'application', 'width', 'product_ref', 'delivery date', 'item_date_1',
       'delivery date_1', 'quantity tons_log', 'thickness_log',
       'selling_price_log', 'Date_difference', 'item_date_day',
       'item_date_month', 'item_date_year'],
      dtype='object')
```

```
# find best algorithm for prediction based on R2, mean absolute error, mean squared error and root mean squared error values

def machine_learning_delivery_date(df, algorithm):

    x = df.drop(columns=['item_date_1','delivery date_1','Date_difference'], axis=1)
    y = df['Date_difference']
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)

    model = algorithm().fit(x_train, y_train)
    y_pred = model.predict(x_test)

    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)

    metrics = {'Algorithm': str(algorithm).split("'")[1].split(".")[-1],
               'R2': r2,
               'Mean Absolute Error': mae,
               'Mean Squared Error': mse,
               'Root Mean Squared Error': rmse}

    return metrics
```

```
print(machine_learning_delivery_date(df_f1, DecisionTreeRegressor))
print(machine_learning_delivery_date(df_f1, ExtraTreesRegressor))
print(machine_learning_delivery_date(df_f1, RandomForestRegressor))
print(machine_learning_delivery_date(df_f1, AdaBoostRegressor))
print(machine_learning_delivery_date(df_f1, GradientBoostingRegressor))
print(machine_learning_delivery_date(df_f1, XGBRegressor))
```

```
{'Algorithm': 'DecisionTreeRegressor', 'R2': 0.9969758658623625, 'Mean Absolute Error': 0.017551422588988792, 'Mean Squared Error':
{'Algorithm': 'ExtraTreesRegressor', 'R2': 0.9999311532190497, 'Mean Absolute Error': 0.030073900726690514, 'Mean Squared Error': 0
{'Algorithm': 'RandomForestRegressor', 'R2': 0.9999752659148513, 'Mean Absolute Error': 0.00340867101859834, 'Mean Squared Error': 6
{'Algorithm': 'AdaBoostRegressor', 'R2': 0.9366327229766369, 'Mean Absolute Error': 7.969662690054617, 'Mean Squared Error': 99.2266
{'Algorithm': 'GradientBoostingRegressor', 'R2': 0.9966288119171449, 'Mean Absolute Error': 0.9796698434829255, 'Mean Squared Error
{'Algorithm': 'XGBRegressor', 'R2': 0.9999931270167448, 'Mean Absolute Error': 0.0550553833621068, 'Mean Squared Error': 0.01096272
```

```
# Random Forest algorithm is low bias and reduce overfitting compared to others.
```

```
# train the model by using Random Forest Regression algorithm to predict 'Date difference'
# 'item_date_1','delivery date_1' - this columns are non-numerical and cannot passed, so skip the columns in model training and predicti

def ml_date_difference():

    # train the model by using correct delivery date (df_f1) dataframe
    x = df_f1.drop(columns=['item_date_1','delivery date_1','Date_difference'], axis=1)
    y = df_f1['Date_difference']
    x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)

    model = RandomForestRegressor().fit(x_train, y_train)

    # predict the 'Date_difference' of df_f2 columns using model
    y_pred_list = []

    for index, row in df_f2.iterrows():
        input_data = row.drop(['item_date_1','delivery date_1','Date_difference'])
        y_pred = model.predict([input_data])
        y_pred_list.append(y_pred[0])

    return y_pred_list


# Machine learning model predict the date difference of (df_f2) datafame
date_difference = ml_date_difference()


print(date_difference)
```

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1
```

```
# convert float values into integer using list comprehension method
date_difference1 = [int(round(i,0)) for i in date_difference]
print(date_difference1)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,
```

```
# add 'Date_difference' column in the dataframe
df_f2['Date_difference'] = pd.DataFrame(date_difference1)
df_f2
```

	item_date	customer	country	status	item type	application	width	product_ref	delivery date	item_date_1	delivery date_1	quantity tons_log	thicl
0	20210401	30153963	30	1	6.0	28	952.0	628377	20210101.0	2021-04-01	2021-01-01	5.956169	
1	20210401	30349574	32	1	3.0	59	1317.0	1668701718	20210101.0	2021-04-01	2021-01-01	5.310301	
2	20210401	30211560	28	1	5.0	10	1980.0	640665	20210301.0	2021-04-01	2021-03-01	6.666354	
3	20210401	30342192	32	1	5.0	41	1220.0	611993	20210101.0	2021-04-01	2021-01-01	4.730808	
4	20210401	30342192	32	1	5.0	41	1220.0	611993	20210101.0	2021-04-01	2021-01-01	4.736160	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2945	20201207	30394817	78	1	2.0	10	1171.0	628377	20201201.0	2020-12-07	2020-12-01	4.618578	
2946	20201207	30394817	78	1	2.0	10	1510.0	628377	20201201.0	2020-12-07	2020-12-01	0.185946	
2947	20201207	30394817	78	1	2.0	10	920.0	628377	20201201.0	2020-12-07	2020-12-01	1.773963	
2948	20201207	30394817	78	1	2.0	10	1306.0	628377	20201201.0	2020-12-07	2020-12-01	4.549376	
2949	20201207	30394817	78	1	2.0	10	1150.0	628377	20201201.0	2020-12-07	2020-12-01	1.440539	

2950 rows × 18 columns

```
# calculate delivery date (item_date + Date_difference = delivery_date)

def find_delivery_date(item_date, date_difference):

    result_date = item_date + timedelta(days=date_difference)

    delivery_date = result_date.strftime("%Y-%m-%d")

    return delivery_date


# find out the delivery date and add to dataframe

df_f2['item_date_1'] = pd.to_datetime(df_f2['item_date_1'])
df_f2['delivery date_1'] = df_f2.apply(lambda x: find_delivery_date(x['item_date_1'], x['Date_difference']), axis=1)
df_f2
```

	item_date	customer	country	status	item type	application	width	product_ref	delivery date	item_date_1	delivery date_1	quantity tons_log	thicl
0	20210401	30153963	30	1	6.0	28	952.0	628377	20210101.0	2021-04-01	2021-04-01	5.956169	
1	20210401	30349574	32	1	3.0	59	1317.0	1668701718	20210101.0	2021-04-01	2021-04-01	5.310301	
2	20210401	30211560	28	1	5.0	10	1980.0	640665	20210301.0	2021-04-01	2021-04-01	6.666354	
3	20210401	30342192	32	1	5.0	41	1220.0	611993	20210101.0	2021-04-01	2021-04-01	4.730808	
4	20210401	30342192	32	1	5.0	41	1220.0	611993	20210101.0	2021-04-01	2021-04-01	4.736160	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2945	20201207	30394817	78	1	2.0	10	1171.0	628377	20201201.0	2020-12-07	2021-01-01	4.618578	
2946	20201207	30394817	78	1	2.0	10	1510.0	628377	20201201.0	2020-12-07	2021-01-01	0.185946	
2947	20201207	30394817	78	1	2.0	10	920.0	628377	20201201.0	2020-12-07	2021-01-01	1.773963	
2948	20201207	30394817	78	1	2.0	10	1306.0	628377	20201201.0	2020-12-07	2021-01-01	4.549376	
2949	20201207	30394817	78	1	2.0	10	1150.0	628377	20201201.0	2020-12-07	2021-01-01	1.440539	

2950 rows × 18 columns

```
# Finally concatinate the both dataframe into single dataframe
df_final = pd.concat([df_f1,df_f2], axis=0, ignore_index=True)
df_final
```

	item_date	customer	country	status	item type	application	width	product_ref	delivery date	item_date_1	delivery date_1	quantity tons_log	thic
0	20210401	30156308	28	1	5.0	10	1500.0	1670798778	20210701.0	2021-04-01	2021-07-01	3.991779	
1	20210401	30202938	25	1	5.0	41	1210.0	1668701718	20210401.0	2021-04-01	2021-04-01	6.643822	
2	20210401	30202938	25	1	5.0	41	1265.0	1668701718	20210401.0	2021-04-01	2021-04-01	5.419608	
3	20210401	30209509	30	2	5.0	41	1125.0	611993	20210701.0	2021-04-01	2021-07-01	1.259203	
4	20210401	30341428	38	1	3.0	10	1275.0	1668701376	20210701.0	2021-04-01	2021-07-01	4.235147	
...	...	...	...	...	...	...	...	...	...	...	...	...	
84135	20201207	30394817	78	1	2.0	10	1171.0	628377	20201201.0	2020-12-07	2021-01-01	4.618578	
84136	20201207	30394817	78	1	2.0	10	1510.0	628377	20201201.0	2020-12-07	2021-01-01	0.185946	
84137	20201207	30394817	78	1	2.0	10	920.0	628377	20201201.0	2020-12-07	2021-01-01	1.773963	
84138	20201207	30394817	78	1	2.0	10	1306.0	628377	20201201.0	2020-12-07	2021-01-01	4.549376	
84139	20201207	30394817	78	1	2.0	10	1150.0	628377	20201201.0	2020-12-07	2021-01-01	1.440539	

84140 rows × 18 columns

```
# split the day, month, and year from 'delivery_date_1' column and add dataframe (This data also help us to prediction)

df_final['delivery date_1'] = pd.to_datetime(df_final['delivery date_1'])

df_final['delivery_date_day'] = df_final['delivery date_1'].dt.day
df_final['delivery_date_month'] = df_final['delivery date_1'].dt.month
df_final['delivery_date_year'] = df_final['delivery date_1'].dt.year

# finally drop the item_date, delivery_date and date_difference columns
df_final.drop(columns=['item_date','delivery date','item_date_1','delivery date_1','Date_difference'], inplace=True)
df_final
```

	customer	country	status	item type	application	width	product_ref	quantity tons_log	thickness_log	selling_price_log	item_date_day
0	30156308	28	1	5.0	10	1500.0	1670798778	3.991779	0.693147	6.749931	1
1	30202938	25	1	5.0	41	1210.0	1668701718	6.643822	-0.223144	6.953684	1
2	30202938	25	1	5.0	41	1265.0	1668701718	5.419608	0.405465	6.890609	1
3	30209509	30	2	5.0	41	1125.0	611993	1.259203	-0.967584	6.377342	1
4	30341428	38	1	3.0	10	1275.0	1668701376	4.235147	-0.510826	7.217443	1
...	...	...	...	...	...	...	...	...	...	...	...
84135	30394817	78	1	2.0	10	1171.0	628377	4.618578	0.405465	6.486161	7
84136	30394817	78	1	2.0	10	1510.0	628377	0.185946	-0.693147	6.513230	7
84137	30394817	78	1	2.0	10	920.0	628377	1.773963	-0.223144	6.601230	7
84138	30394817	78	1	2.0	10	1306.0	628377	4.549376	0.405465	6.562444	7
84139	30394817	78	1	2.0	10	1150.0	628377	1.440539	-0.693147	6.561031	7

84140 rows × 16 columns

Classification Method - Predict Status

```
from imblearn.combine import SMOTETomek
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
import matplotlib.pyplot as plt
import pickle
```

```
df_final.head()
```

	customer	country	status	item type	application	width	product_ref	quantity tons_log	thickness_log	selling_price_log	item_date_day	item
0	30156308	28	1	5.0	10	1500.0	1670798778	3.991779	0.693147	6.749931	1	
1	30202938	25	1	5.0	41	1210.0	1668701718	6.643822	-0.223144	6.953684	1	
2	30202938	25	1	5.0	41	1265.0	1668701718	5.419608	0.405465	6.890609	1	
3	30209509	30	2	5.0	41	1125.0	611993	1.259203	-0.967584	6.377342	1	
4	30341428	38	1	3.0	10	1275.0	1668701376	4.235147	-0.510826	7.217443	1	

```
# check data types
df_final.dtypes
```

```
customer                int64
country                 int64
status                  int64
item type             float64
application             int64
width                 float64
product_ref             int64
quantity tons_log     float64
thickness_log         float64
selling_price_log     float64
item_date_day           int64
item_date_month         int64
item_date_year          int64
delivery_date_day       int64
delivery_date_month     int64
delivery_date_year      int64
dtype: object
```

```
df_c = df_final.copy()

# filter the status column values only 1 & 0 rows in a new dataframe ['Won':1 & 'Lost':0]
df_c = df_c[(df_c.status == 1) | (df_c.status == 0)]
df_c
```

	customer	country	status	item type	application	width	product_ref	quantity tons_log	thickness_log	selling_price_log	item_date_day
0	30156308	28	1	5.0	10	1500.0	1670798778	3.991779	0.693147	6.749931	1
1	30202938	25	1	5.0	41	1210.0	1668701718	6.643822	-0.223144	6.953684	1
2	30202938	25	1	5.0	41	1265.0	1668701718	5.419608	0.405465	6.890609	1
4	30341428	38	1	3.0	10	1275.0	1668701376	4.235147	-0.510826	7.217443	1
5	30202938	25	1	5.0	41	1165.0	1668701718	6.446714	0.405465	6.890609	1
...	...	...	...	...	...	...	...	...	...	...	...
84135	30394817	78	1	2.0	10	1171.0	628377	4.618578	0.405465	6.486161	7
84136	30394817	78	1	2.0	10	1510.0	628377	0.185946	-0.693147	6.513230	7
84137	30394817	78	1	2.0	10	920.0	628377	1.773963	-0.223144	6.601230	7
84138	30394817	78	1	2.0	10	1306.0	628377	4.549376	0.405465	6.562444	7
84139	30394817	78	1	2.0	10	1150.0	628377	1.440539	-0.693147	6.561031	7

70014 rows × 16 columns

```
# check no of rows (records) of each 1 and 0 in dataframe
df_c['status'].value_counts()
```

```
1    56900
0    13114
Name: status, dtype: int64
```

```
# in status feature, the 'Won' and 'Lost' value difference is very high. So we need to oversampling to reduce the difference

x = df_c.drop('status', axis=1)
y = df_c['status']

x_new, y_new = SMOTETomek().fit_resample(x,y)
```

```
x.shape, y.shape, x_new.shape, y_new.shape
```

```
((70014, 15), (70014,), (112662, 15), (112662,))
```

```
# check the accuracy of training and testing using metrics
# algorithm.__name__  - it return the algorithm name

def machine_learning_classification(x_new,y_new, algorithm):

    x_train, x_test, y_train, y_test = train_test_split(x_new, y_new, test_size=0.2, random_state=42)
    model = algorithm().fit(x_train, y_train)

    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)

    accuracy_train = metrics.accuracy_score(y_train, y_pred_train)
    accuracy_test = metrics.accuracy_score(y_test, y_pred_test)

    # algo = str(algorithm).split("'")[1].split(".")[-1]
    accuracy_metrics = {'algorithm'    : algorithm.__name__,
                        'accuracy_train': accuracy_train,
                        'accuracy_test' : accuracy_test}

    return accuracy_metrics
```

```
print(machine_learning_classification(x_new, y_new, DecisionTreeClassifier))
print(machine_learning_classification(x_new, y_new, ExtraTreesClassifier))
print(machine_learning_classification(x_new, y_new, RandomForestClassifier))
print(machine_learning_classification(x_new, y_new, AdaBoostClassifier))
print(machine_learning_classification(x_new, y_new, GradientBoostingClassifier))
print(machine_learning_classification(x_new, y_new, XGBClassifier))
```

```
{'algorithm': 'DecisionTreeClassifier', 'accuracy_train': 1.0, 'accuracy_test': 0.968845692983624}
{'algorithm': 'ExtraTreesClassifier', 'accuracy_train': 1.0, 'accuracy_test': 0.9869080903563662}
{'algorithm': 'RandomForestClassifier', 'accuracy_train': 0.9999778095840407, 'accuracy_test': 0.9846891226201571}
{'algorithm': 'AdaBoostClassifier', 'accuracy_train': 0.8086409479745698, 'accuracy_test': 0.8070385656592554}
{'algorithm': 'GradientBoostingClassifier', 'accuracy_train': 0.8567497697744344, 'accuracy_test': 0.8529268184440598}
{'algorithm': 'XGBClassifier', 'accuracy_train': 0.9736488810482753, 'accuracy_test': 0.9642302400923091}
```

```
# before oversampling result
#{'algorithm': 'DecisionTreeClassifier', 'accuracy_train': 1.0, 'accuracy_test': 0.968845692983624}
#{'algorithm': 'ExtraTreesClassifier', 'accuracy_train': 1.0, 'accuracy_test': 0.9869080903563662}
#{'algorithm': 'RandomForestClassifier', 'accuracy_train': 0.9999778095840407, 'accuracy_test': 0.9846891226201571}
#{'algorithm': 'AdaBoostClassifier', 'accuracy_train': 0.8086409479745698, 'accuracy_test': 0.8070385656592554}
#{'algorithm': 'GradientBoostingClassifier', 'accuracy_train': 0.8567497697744344, 'accuracy_test': 0.8529268184440598}
#{'algorithm': 'XGBClassifier', 'accuracy_train': 0.9736488810482753, 'accuracy_test': 0.9642302400923091}
```

```
# we got good accuracy after oversampling
# ExtraTreesClassifier and RandomForestClassifier both have good testing accuracy, but in training accuracy is overfitting.
# RandomForestClassifier is good interpretability, so i select the algorithm
```

```
# GridsearchCV is a cross validation function.

# Hyper parameter tuning - we give parameter values manually in the algorithm to reduce the overfitting issue and get better accuracy.

# so using gridserachcv method - to pass the mulitiple values in each parameters and it try to evaluate all the combination of values ar
# finally return the best accuracy parameter values based on the score.

# example: {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2}

# note: This process can take long time (avg: 1 hour 15 mins). Please wait be patient.


# refer parameter values: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
```

```
x_train, x_test, y_train, y_test = train_test_split(x_new,y_new,test_size=0.2,random_state=42)

param_grid = {'max_depth'        : [2, 5, 10, 20],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf' : [1, 2, 4],
              'max_features'     : ['sqrt', 'log2']}

grid_search = GridSearchCV(estimator=RandomForestClassifier(), param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(x_train, y_train)
```

```
# evaluate all the parameter combinations and return the best parameters based on score
grid_search.best_params_
```

```
grid_search.best_score_
```

```
# passing the parameters in the random forest algorithm and check the accuracy for training and testing

x_train, x_test, y_train, y_test = train_test_split(x_new,y_new,test_size=0.2,random_state=42)

model = RandomForestClassifier(max_depth=20, max_features='sqrt', min_samples_leaf=1, min_samples_split=2).fit(x_train, y_train)
y_pred_train = model.predict(x_train)
y_pred_test = model.predict(x_test)
accuracy_train = metrics.accuracy_score(y_train, y_pred_train)
accuracy_test = metrics.accuracy_score(y_test, y_pred_test)
accuracy_train, accuracy_test
```

```
(0.9935980649957283, 0.9786979097323925)
```

```
# now the training accuracy overfitting reduced. so now model will predict effectively for unseen data
```

```
# predict the status and check the accuracy using metrics

x_train, x_test, y_train, y_test = train_test_split(x_new,y_new,test_size=0.2,random_state=42)

model = RandomForestClassifier(max_depth=20, max_features='sqrt', min_samples_leaf=1, min_samples_split=2).fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_true=y_test, y_pred=y_pred))
print(classification_report(y_true=y_test, y_pred=y_pred))
```

```
[[11026    91]
 [  399 11017]]
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     11117
           1       0.99      0.97      0.98     11416

    accuracy                           0.98     22533
   macro avg       0.98      0.98      0.98     22533
weighted avg       0.98      0.98      0.98     22533
```

```
# Receiver Operating Characteristic (ROC) Curve and Area Under the Curve (AUC)

FP,TP,threshold = roc_curve(y_true=y_test, y_score=y_pred)
auc_curve = auc(x=FP, y=TP)
print(auc_curve)
```

```
0.9784316961800015
```

```
plt.plot(FP, TP, label=f"ROC Curve (area={round(auc_curve, 2)}) ")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.10])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```


1.0