Revised Project

February 22, 2022

1 Payment Date Prediction

1.0.1 Importing related Libraries

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import VarianceThreshold
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
```

```
/opt/conda/lib/python3.9/site-packages/xgboost/compat.py:36: FutureWarning:
pandas.Int64Index is deprecated and will be removed from pandas in a future
version. Use pandas.Index with the appropriate dtype instead.
  from pandas import MultiIndex, Int64Index
```

1.0.2 Store the dataset into the Dataframe

[2]:
```
df = pd.read_csv(r"./h2h_assignment_dataset.csv")
```

1.0.3 Check the shape of the dataframe

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

[3]:
```
(50000, 19)
```

1.0.4 Check the Detail information of the dataframe

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
```

```
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   business_code           50000 non-null  object
 1   cust_number             50000 non-null  object
 2   name_customer           50000 non-null  object
 3   clear_date              40000 non-null  object
 4   buisness_year           50000 non-null  float64
 5   doc_id                  50000 non-null  float64
 6   posting_date            50000 non-null  object
 7   document_create_date    50000 non-null  int64
 8   document_create_date.1  50000 non-null  int64
 9   due_in_date             50000 non-null  float64
 10  invoice_currency        50000 non-null  object
 11  document type           50000 non-null  object
 12  posting_id              50000 non-null  float64
 13  area_business           0 non-null      float64
 14  total_open_amount       50000 non-null  float64
 15  baseline_create_date    50000 non-null  float64
 16  cust_payment_terms      50000 non-null  object
 17  invoice_id              49994 non-null  float64
 18  isOpen                  50000 non-null  int64
dtypes: float64(8), int64(3), object(8)
memory usage: 7.2+ MB
```

1.0.5 Display All the column names

[5]:
```
df.columns
```

[5]:
```
Index(['business_code', 'cust_number', 'name_customer', 'clear_date',
       'buisness_year', 'doc_id', 'posting_date', 'document_create_date',
       'document_create_date.1', 'due_in_date', 'invoice_currency',
       'document type', 'posting_id', 'area_business', 'total_open_amount',
       'baseline_create_date', 'cust_payment_terms', 'invoice_id', 'isOpen'],
      dtype='object')
```

1.0.6 Describe the entire dataset

[6]:
```
df.describe()
```

[6]:
```
       buisness_year        doc_id  document_create_date  \
count   50000.000000  5.000000e+04          5.000000e+04
mean     2019.305700  2.012238e+09          2.019351e+07
std         0.460708  2.885235e+08          4.496041e+03
min      2019.000000  1.928502e+09          2.018123e+07
25%      2019.000000  1.929342e+09          2.019050e+07
50%      2019.000000  1.929964e+09          2.019091e+07
```

```
75%       2020.000000  1.930619e+09          2.020013e+07
max       2020.000000  9.500000e+09          2.020052e+07

       document_create_date.1   due_in_date  posting_id  area_business  \
count            5.000000e+04  5.000000e+04     50000.0            0.0
mean             2.019354e+07  2.019368e+07         1.0            NaN
std              4.482134e+03  4.470614e+03         0.0            NaN
min              2.018123e+07  2.018122e+07         1.0            NaN
25%              2.019051e+07  2.019052e+07         1.0            NaN
50%              2.019091e+07  2.019093e+07         1.0            NaN
75%              2.020013e+07  2.020022e+07         1.0            NaN
max              2.020052e+07  2.020071e+07         1.0            NaN

       total_open_amount  baseline_create_date    invoice_id        isOpen
count       50000.000000          5.000000e+04  4.999400e+04  50000.000000
mean        32337.021651          2.019354e+07  2.011340e+09      0.200000
std         39205.975231          4.482701e+03  2.766335e+08      0.400004
min             0.720000          2.018121e+07  1.928502e+09      0.000000
25%          4928.312500          2.019050e+07  1.929342e+09      0.000000
50%         17609.010000          2.019091e+07  1.929964e+09      0.000000
75%         47133.635000          2.020013e+07  1.930619e+09      0.000000
max        668593.360000          2.020052e+07  2.960636e+09      1.000000
```

2 Data Cleaning

- Show top 5 records from the dataset

[7]:
```
df.head(5)
```

[7]:
```
  business_code cust_number       name_customer           clear_date  \
0          U001  0200769623        WAL-MAR corp  2020-02-11 00:00:00
1          U001  0200980828              BEN E   2019-08-08 00:00:00
2          U001  0200792734          MDV/ trust  2019-12-30 00:00:00
3          CA02  0140105686            SYSC llc                  NaN
4          U001  0200769623  WAL-MAR foundation  2019-11-25 00:00:00

   buisness_year        doc_id posting_date  document_create_date  \
0         2020.0  1.930438e+09   2020-01-26              20200125
1         2019.0  1.929646e+09   2019-07-22              20190722
2         2019.0  1.929874e+09   2019-09-14              20190914
3         2020.0  2.960623e+09   2020-03-30              20200330
4         2019.0  1.930148e+09   2019-11-13              20191113

   document_create_date.1  due_in_date invoice_currency document type  \
0                20200126   20200210.0              USD            RV
1                20190722   20190811.0              USD            RV
2                20190914   20190929.0              USD            RV
```

```
3              20200330   20200410.0            CAD            RV
4              20191113   20191128.0            USD            RV

   posting_id  area_business  total_open_amount  baseline_create_date  \
0         1.0            NaN           54273.28            20200126.0
1         1.0            NaN           79656.60            20190722.0
2         1.0            NaN            2253.86            20190914.0
3         1.0            NaN            3299.70            20200331.0
4         1.0            NaN           33133.29            20191113.0

  cust_payment_terms    invoice_id  isOpen
0               NAH4  1.930438e+09       0
1               NAD1  1.929646e+09       0
2               NAA8  1.929874e+09       0
3               CA10  2.960623e+09       1
4               NAH4  1.930148e+09       0
```

2.0.1 Display the Null values percentage against every columns (compare to the total number of records)

- Output expected : area_business - 100% null, clear_data = 20% null, invoice_id = 0.012% null

[8]:
```
df.isnull().mean().mul(100).sort_values(ascending=False)
```

[8]:
```
area_business             100.000
clear_date                 20.000
invoice_id                  0.012
business_code               0.000
invoice_currency            0.000
cust_payment_terms          0.000
baseline_create_date        0.000
total_open_amount           0.000
posting_id                  0.000
document type               0.000
due_in_date                 0.000
cust_number                 0.000
document_create_date.1      0.000
document_create_date        0.000
posting_date                0.000
doc_id                      0.000
buisness_year               0.000
name_customer               0.000
isOpen                      0.000
dtype: float64
```

2.0.2 Display Invoice_id and Doc_Id

- Note - Many of the would have same invoice_id and doc_id

[9]:
```
df.loc[:, ["doc_id", "invoice_id"]]
```

[9]:
```
             doc_id    invoice_id
0      1.930438e+09  1.930438e+09
1      1.929646e+09  1.929646e+09
2      1.929874e+09  1.929874e+09
3      2.960623e+09  2.960623e+09
4      1.930148e+09  1.930148e+09
…               …             …
49995  1.930797e+09  1.930797e+09
49996  1.929744e+09  1.929744e+09
49997  1.930537e+09  1.930537e+09
49998  1.930199e+09  1.930199e+09
49999  1.928576e+09  1.928576e+09

[50000 rows x 2 columns]
```

Write a code to check - 'baseline_create_date',"document_create_date",'document_create_date.1' - these columns are almost same.

- Please note, if they are same, we need to drop them later

[10]:
```
df.groupby(
    ["baseline_create_date", "document_create_date", "document_create_date.1"]
).size()
```

[10]:
```
baseline_create_date  document_create_date  document_create_date.1
20181214.0            20190108              20190108                   1
                      20190201              20190201                   1
20181230.0            20181226              20181230                   1
                      20181228              20181230                   1
                      20181229              20181230                  44
                                                                      ..
20200515.0            20200515              20200515                   1
20200517.0            20200513              20200517                   1
20200518.0            20200516              20200518                   1
20200519.0            20200519              20200519                   1
20200522.0            20200522              20200522                   1
Length: 5852, dtype: int64
```

Please check, Column 'posting_id' is constant columns or not

[11]:
```
df.columns[df.nunique() <= 1]  # Constant columns
```

[11]:
```
Index(['posting_id', 'area_business'], dtype='object')
```

Please check 'isOpen' is a constant column and relevant column for this project or not

[12]:
```
df.columns[df.nunique() <= 1]  # Constant columns
```

[12]:
```
Index(['posting_id', 'area_business'], dtype='object')
```

2.0.3 Write the code to drop all the following columns from the dataframe

- 'area_business'
- "posting_id"
- "invoice_id"
- "document_create_date"
- "isOpen"
- 'document type'
- 'document_create_date.1

[13]:
```
df.drop(
    columns=[
        "area_business",
        "posting_id",
        "invoice_id",
        "document_create_date",
        "isOpen",
        "document type",
        "document_create_date.1",
    ],
    axis=1,
    inplace=True,
)
```

2.0.4 Please check from the dataframe whether all the columns are removed or not

[14]:
```
df.columns
```

[14]:
```
Index(['business_code', 'cust_number', 'name_customer', 'clear_date',
       'buisness_year', 'doc_id', 'posting_date', 'due_in_date',
       'invoice_currency', 'total_open_amount', 'baseline_create_date',
       'cust_payment_terms'],
      dtype='object')
```

2.0.5 Show all the Dublicate rows from the dataframe

[15]:
```
df[df.duplicated()]
```

[15]:
```
      business_code cust_number       name_customer          clear_date  \
1041           U001  0200769623          WAL-MAR in 2019-03-12 00:00:00
2400           U001  0200769623       WAL-MAR trust 2019-08-28 00:00:00
2584           U001  0200769623 WAL-MAR corporation 2019-12-16 00:00:00
```

```
3755          U001  0200769623          WAL-MAR  2019-11-22 00:00:00
3873          CA02  0140104409     LOB associates                NaN
…             …     …                         …                    …
49928         U001  0200915438         GROC trust  2019-08-15 00:00:00
49963         U001  0200759878              SA us  2019-01-29 00:00:00
49986         U001  0200772670 ASSOCIAT foundation  2019-06-12 00:00:00
49990         U001  0200765011          MAINES llc  2019-06-06 00:00:00
49991         U001  0200704045           RA trust  2019-10-25 00:00:00

       buisness_year        doc_id posting_date  due_in_date invoice_currency  \
1041          2019.0  1.928870e+09   2019-02-28   20190315.0              USD
2400          2019.0  1.929758e+09   2019-08-18   20190902.0              USD
2584          2019.0  1.930217e+09   2019-12-04   20191219.0              USD
3755          2019.0  1.930137e+09   2019-11-12   20191127.0              USD
3873          2020.0  2.960629e+09   2020-04-14   20200425.0              CAD
…                …             …          …           …             …
49928         2019.0  1.929646e+09   2019-07-25   20190809.0              USD
49963         2019.0  1.928614e+09   2019-01-13   20190128.0              USD
49986         2019.0  1.929403e+09   2019-05-29   20190613.0              USD
49990         2019.0  1.929365e+09   2019-05-22   20190606.0              USD
49991         2019.0  1.930001e+09   2019-10-10   20191025.0              USD

       total_open_amount  baseline_create_date cust_payment_terms
1041            19557.41            20190228.0              NAH4
2400             5600.41            20190818.0              NAH4
2584            35352.17            20191204.0              NAH4
3755             2982.64            20191112.0              NAH4
3873            82975.82            20200415.0              CA10
…                    …                   …                 …
49928            6969.00            20190725.0              NAA8
49963           10968.24            20190113.0              NAH4
49986          155837.53            20190529.0              NAU5
49990            4008.05            20190522.0              NAA8
49991           73002.24            20191010.0              NAA8

[1161 rows x 12 columns]
```

2.0.6 Display the Number of Dublicate Rows

[16]:
```
df.duplicated().shape[0]
```

[16]:
```
50000
```

2.0.7 Drop all the Dublicate Rows

[17]:
```python
df.drop_duplicates(inplace=True)
```

Now check for all dublicate rows now

- Note - It must be 0 by now

[18]:
```python
df[df.duplicated()]  # It is 0 indeed, all duplicate rows have been dropped
```

[18]:
```
Empty DataFrame
Columns: [business_code, cust_number, name_customer, clear_date, buisness_year,
doc_id, posting_date, due_in_date, invoice_currency, total_open_amount,
baseline_create_date, cust_payment_terms]
Index: []
```

2.0.8 Check for the number of Rows and Columns in your dataset

[19]:
```python
df.shape[0]
```

[19]:
```
48839
```

2.0.9 Find out the total count of null values in each columns

[20]:
```python
df.isnull().sum()
```

[20]:
```
business_code              0
cust_number                0
name_customer              0
clear_date              9681
buisness_year              0
doc_id                     0
posting_date               0
due_in_date                0
invoice_currency           0
total_open_amount          0
baseline_create_date       0
cust_payment_terms         0
dtype: int64
```

3 Data type Conversion

3.0.1 Please check the data type of each column of the dataframe

[21]:
```python
df.dtypes
```

```
[21]: business_code            object
      cust_number              object
      name_customer            object
      clear_date               object
      buisness_year           float64
      doc_id                  float64
      posting_date             object
      due_in_date             float64
      invoice_currency         object
      total_open_amount       float64
      baseline_create_date    float64
      cust_payment_terms       object
      dtype: object
```

3.0.2 Check the datatype format of below columns

- clear_date
- posting_date
- due_in_date
- baseline_create_date

```
[22]: df.loc[
          :, ["clear_date", "posting_date", "due_in_date", "baseline_create_date"]
      ].dtypes
```

```
[22]: clear_date               object
      posting_date             object
      due_in_date             float64
      baseline_create_date    float64
      dtype: object
```

3.0.3 converting date columns into date time formats

- clear_date
- posting_date
- due_in_date
- baseline_create_date
- **Note - You have to convert all these above columns into "%Y%m%d" format**

```
[23]: dt_lis = ["baseline_create_date", "due_in_date", "clear_date", "posting_date"]
      df[dt_lis[0]], df[dt_lis[1]] = df[dt_lis[0]].astype("int"), df[
          dt_lis[1]
      ].astype("int")
```

[24]:
```
df[dt_lis[0]], df[dt_lis[1]] = (
    pd.to_datetime(df[dt_lis[0]], format="%Y%m%d").dt.date,
    pd.to_datetime(df[dt_lis[1]], format="%Y%m%d").dt.date,
)
```

[25]:
```
df[dt_lis[2]], df[dt_lis[3]] = (
    pd.to_datetime(df[dt_lis[2]], format="%Y-%m-%d").dt.date,
    pd.to_datetime(df[dt_lis[3]], format="%Y-%m-%d").dt.date,
)
```

[26]:
```
df.loc[:, dt_lis]
```

[26]:
```
      baseline_create_date due_in_date  clear_date posting_date
0               2020-01-26  2020-02-10  2020-02-11   2020-01-26
1               2019-07-22  2019-08-11  2019-08-08   2019-07-22
2               2019-09-14  2019-09-29  2019-12-30   2019-09-14
3               2020-03-31  2020-04-10         NaT   2020-03-30
4               2019-11-13  2019-11-28  2019-11-25   2019-11-13
…                      …           …          …           …
49995           2020-04-21  2020-05-06         NaT   2020-04-21
49996           2019-08-15  2019-08-30  2019-09-03   2019-08-15
49997           2020-02-19  2020-03-05  2020-03-05   2020-02-19
49998           2019-11-27  2019-12-12  2019-12-12   2019-11-27
49999           2019-01-01  2019-01-24  2019-01-15   2019-01-05

[48839 rows x 4 columns]
```

3.0.4 Please check the datatype of all the columns after conversion of the above 4 columns

[27]:
```
df.dtypes
```

[27]:
```
business_code            object
cust_number              object
name_customer            object
clear_date               object
buisness_year           float64
doc_id                  float64
posting_date             object
due_in_date              object
invoice_currency         object
total_open_amount       float64
baseline_create_date     object
cust_payment_terms       object
dtype: object
```

the invoice_currency column contains two different categories, USD and CAD

- Please do a count of each currency

[28]:
```
df["invoice_currency"].value_counts()
```

[28]:
```
USD    45011
CAD     3828
Name: invoice_currency, dtype: int64
```

display the "total_open_amount" column value

[29]:
```
df["total_open_amount"]
```

[29]:
```
0        54273.28
1        79656.60
2         2253.86
3         3299.70
4        33133.29
           …
49995     3187.86
49996     6766.54
49997     6120.86
49998       63.48
49999     1790.30
Name: total_open_amount, Length: 48839, dtype: float64
```

3.0.5 Convert all CAD into USD currency of "total_open_amount" column

- 1 CAD = 0.7 USD
- Create a new column i.e "converted_usd" and store USD and convered CAD to USD

[30]:
```
df["converted_usd"] = np.where(
    (df["invoice_currency"] == "CAD"),
    df["total_open_amount"] * 0.7,
    df["total_open_amount"],
)
```

3.0.6 Display the new "converted_usd" column values

[31]:
```
df.loc[
    0:9,
    [
        "invoice_currency",
        "total_open_amount",
        "converted_usd",
    ],
]
```

[31]:

	invoice_currency	total_open_amount	converted_usd
0	USD	54273.28	54273.280
1	USD	79656.60	79656.600
2	USD	2253.86	2253.860
3	CAD	3299.70	2309.790
4	USD	33133.29	33133.290
5	CAD	22225.84	15558.088
6	USD	7358.49	7358.490
7	USD	11173.02	11173.020
8	USD	15995.04	15995.040
9	USD	28.63	28.630

3.0.7 Display year wise total number of record

- Note - use "buisness_year" column for this

[32]:
```
df["buisness_year"].value_counts()
```

[32]:
```
2019.0    33975
2020.0    14864
Name: buisness_year, dtype: int64
```

3.0.8 Write the code to delete the following columns

- 'invoice_currency'
- 'total_open_amount',

[33]:
```
df.drop(columns=["invoice_currency", "total_open_amount"], axis=1, inplace=True)
```

3.0.9 Write a code to check the number of columns in dataframe

[34]:
```
df.shape[1]
```

[34]: 11

4 Splitting the Dataset

4.0.1 Look for all columns containing null value

- Note - Output expected is only one column

[35]:
```
null_columns = df.columns[df.isna().any()].tolist()
null_columns
```

[35]: ['clear_date']

Find out the number of null values from the column that you got from the above code

[36]:
```
df[null_columns].isna().sum()   # 9681 rows of null
```

[36]:
```
clear_date    9681
dtype: int64
```

4.0.2 On basis of the above column we are spliting data into dataset

- First dataframe (refer that as maindata) only containing the rows, that have NULL data in that column (This is going to be our train dataset)
- Second dataframe (refer that as nulldata) that contains the columns, that have Not Null data in that column (This is going to be our test dataset)

[37]:
```
filter = df[null_columns[0]].isna()
maindata = df[filter]
nulldata = df[~filter]
```

4.0.3 Check the number of Rows and Columns for both the dataframes

[38]:
```
maindata.shape
```

[38]:
```
(9681, 11)
```

[39]:
```
nulldata.shape
```

[39]:
```
(39158, 11)
```

4.0.4 Display the 5 records from maindata and nulldata dataframes

[40]:
```
maindata.head(5)
```

[40]:
```
   business_code cust_number     name_customer clear_date  buisness_year  \
3           CA02  0140105686          SYSC llc        NaT         2020.0
7           U001  0200744019           TARG us        NaT         2020.0
10          U001  0200418007                AM        NaT         2020.0
14          U001  0200739534        OK systems        NaT         2020.0
15          U001  0200353024  DECA corporation        NaT         2020.0

          doc_id posting_date due_in_date baseline_create_date  \
3   2.960623e+09   2020-03-30  2020-04-10           2020-03-31
7   1.930659e+09   2020-03-19  2020-04-03           2020-03-19
10  1.930611e+09   2020-03-11  2020-03-26           2020-03-11
14  1.930788e+09   2020-04-15  2020-04-30           2020-04-15
15  1.930817e+09   2020-04-23  2020-04-26           2020-04-16

   cust_payment_terms  converted_usd
3                CA10        2309.79
7                NAA8       11173.02
```

```
10               NAA8        3525.59
14               NAA8      121105.65
15               NAM2        3726.06
```

[41]:
```python
nulldata.head(5)
```

[41]:
```
  business_code  cust_number       name_customer  clear_date  buisness_year  \
0          U001   0200769623        WAL-MAR corp  2020-02-11         2020.0
1          U001   0200980828              BEN E   2019-08-08         2019.0
2          U001   0200792734          MDV/ trust  2019-12-30         2019.0
4          U001   0200769623  WAL-MAR foundation  2019-11-25         2019.0
5          CA02   0140106181     THE  corporation  2019-12-04         2019.0

         doc_id posting_date due_in_date baseline_create_date  \
0  1.930438e+09   2020-01-26  2020-02-10           2020-01-26
1  1.929646e+09   2019-07-22  2019-08-11           2019-07-22
2  1.929874e+09   2019-09-14  2019-09-29           2019-09-14
4  1.930148e+09   2019-11-13  2019-11-28           2019-11-13
5  2.960581e+09   2019-09-20  2019-10-04           2019-09-24

  cust_payment_terms  converted_usd
0               NAH4      54273.280
1               NAD1      79656.600
2               NAA8       2253.860
4               NAH4      33133.290
5               CA10      15558.088
```

4.1 Considering the maindata

Generate a new column "Delay" from the existing columns

- Note - You are expected to create a new column 'Delay' from two existing columns, "clear_date" and "due_in_date"
- Formula - Delay = clear_date - due_in_date

[42]:
```python
df["Delay"] = df["clear_date"] - df["due_in_date"]

df
```

[42]:
```
      business_code cust_number       name_customer  clear_date  \
0              U001  0200769623        WAL-MAR corp  2020-02-11
1              U001  0200980828              BEN E   2019-08-08
2              U001  0200792734          MDV/ trust  2019-12-30
3              CA02  0140105686            SYSC llc         NaT
4              U001  0200769623  WAL-MAR foundation  2019-11-25
…                …          …                   …           …
49995          U001  0200561861      CO corporation         NaT
49996          U001  0200769623          WAL-MAR co  2019-09-03
```

```
49997          U001  0200772595   SAFEW associates  2020-03-05
49998          U001  0200726979          BJ'S  llc  2019-12-12
49999          U001  0200020431           DEC corp  2019-01-15

       buisness_year        doc_id posting_date due_in_date  \
0             2020.0  1.930438e+09   2020-01-26  2020-02-10
1             2019.0  1.929646e+09   2019-07-22  2019-08-11
2             2019.0  1.929874e+09   2019-09-14  2019-09-29
3             2020.0  2.960623e+09   2020-03-30  2020-04-10
4             2019.0  1.930148e+09   2019-11-13  2019-11-28
…                …           …          …          …
49995         2020.0  1.930797e+09   2020-04-21  2020-05-06
49996         2019.0  1.929744e+09   2019-08-15  2019-08-30
49997         2020.0  1.930537e+09   2020-02-19  2020-03-05
49998         2019.0  1.930199e+09   2019-11-27  2019-12-12
49999         2019.0  1.928576e+09   2019-01-05  2019-01-24

      baseline_create_date cust_payment_terms  converted_usd   Delay
0               2020-01-26               NAH4       54273.28  1 days
1               2019-07-22               NAD1       79656.60 -3 days
2               2019-09-14               NAA8        2253.86 92 days
3               2020-03-31               CA10        2309.79     NaT
4               2019-11-13               NAH4       33133.29 -3 days
…                      …                 …             …      …
49995           2020-04-21               NAA8        3187.86     NaT
49996           2019-08-15               NAH4        6766.54  4 days
49997           2020-02-19               NAA8        6120.86  0 days
49998           2019-11-27               NAA8          63.48  0 days
49999           2019-01-01               NAM4        1790.30 -9 days

[48839 rows x 12 columns]
```

4.1.1 Generate a new column "avgdelay" from the existing columns

- Note - You are expected to make a new column "avgdelay" by grouping "name_customer" column with reapect to mean of the "Delay" column.
- This new column "avg_delay" is meant to store "customer_name" wise delay
- groupby('name_customer')['Delay'].mean(numeric_only=False)
- Display the new "avg_delay" column

[43]:
```
df["avgdelay"] = df.groupby("name_customer")["Delay"].transform("mean")

df
```

[43]:
```
      business_code cust_number       name_customer  clear_date  \
0              U001  0200769623        WAL-MAR corp  2020-02-11
1              U001  0200980828              BEN E   2019-08-08
```

```
2               U001  0200792734         MDV/ trust  2019-12-30
3               CA02  0140105686          SYSC llc         NaT
4               U001  0200769623  WAL-MAR foundation  2019-11-25
…                  …           …                  …           …
49995           U001  0200561861     CO corporation          NaT
49996           U001  0200769623         WAL-MAR co  2019-09-03
49997           U001  0200772595   SAFEW associates  2020-03-05
49998           U001  0200726979          BJ'S  llc  2019-12-12
49999           U001  0200020431           DEC corp  2019-01-15

       buisness_year        doc_id posting_date due_in_date  \
0             2020.0  1.930438e+09   2020-01-26  2020-02-10
1             2019.0  1.929646e+09   2019-07-22  2019-08-11
2             2019.0  1.929874e+09   2019-09-14  2019-09-29
3             2020.0  2.960623e+09   2020-03-30  2020-04-10
4             2019.0  1.930148e+09   2019-11-13  2019-11-28
…                  …             …            …           …
49995         2020.0  1.930797e+09   2020-04-21  2020-05-06
49996         2019.0  1.929744e+09   2019-08-15  2019-08-30
49997         2020.0  1.930537e+09   2020-02-19  2020-03-05
49998         2019.0  1.930199e+09   2019-11-27  2019-12-12
49999         2019.0  1.928576e+09   2019-01-05  2019-01-24

      baseline_create_date cust_payment_terms  converted_usd   Delay  \
0               2020-01-26               NAH4       54273.28  1 days
1               2019-07-22               NAD1       79656.60 -3 days
2               2019-09-14               NAA8        2253.86 92 days
3               2020-03-31               CA10        2309.79     NaT
4               2019-11-13               NAH4       33133.29 -3 days
…                        …                  …              …       …
49995           2020-04-21               NAA8        3187.86     NaT
49996           2019-08-15               NAH4        6766.54  4 days
49997           2020-02-19               NAA8        6120.86  0 days
49998           2019-11-27               NAA8          63.48  0 days
49999           2019-01-01               NAM4        1790.30 -9 days

                         avgdelay
0     -3 days +07:08:49.779837776
1                19 days 00:00:00
2       8 days 02:10:54.545454545
3       2 days 19:03:31.764705882
4     -3 days +19:33:27.692307693
…                               …
49995 -1 days +17:08:34.285714286
49996 -3 days +12:40:08.540925267
49997   1 days 01:08:34.285714285
49998   1 days 13:36:42.985074626
```

```
49999 -4 days +02:20:52.173913044

[48839 rows x 13 columns]
```

You need to add the "avg_delay" column with the maindata, mapped with "name_customer" column

- Note - You need to use map function to map the avgdelay with respect to "name_customer" column

[44]:
```
df
```

[44]:
```
      business_code cust_number       name_customer  clear_date  \
0              U001  0200769623        WAL-MAR corp  2020-02-11
1              U001  0200980828              BEN E   2019-08-08
2              U001  0200792734          MDV/ trust  2019-12-30
3              CA02  0140105686            SYSC llc         NaT
4              U001  0200769623  WAL-MAR foundation  2019-11-25
…               …          …                  …          …
49995          U001  0200561861      CO corporation         NaT
49996          U001  0200769623          WAL-MAR co  2019-09-03
49997          U001  0200772595    SAFEW associates  2020-03-05
49998          U001  0200726979           BJ'S  llc  2019-12-12
49999          U001  0200020431            DEC corp  2019-01-15

       buisness_year        doc_id posting_date due_in_date  \
0             2020.0  1.930438e+09   2020-01-26  2020-02-10
1             2019.0  1.929646e+09   2019-07-22  2019-08-11
2             2019.0  1.929874e+09   2019-09-14  2019-09-29
3             2020.0  2.960623e+09   2020-03-30  2020-04-10
4             2019.0  1.930148e+09   2019-11-13  2019-11-28
…                …            …          …          …
49995         2020.0  1.930797e+09   2020-04-21  2020-05-06
49996         2019.0  1.929744e+09   2019-08-15  2019-08-30
49997         2020.0  1.930537e+09   2020-02-19  2020-03-05
49998         2019.0  1.930199e+09   2019-11-27  2019-12-12
49999         2019.0  1.928576e+09   2019-01-05  2019-01-24

      baseline_create_date cust_payment_terms  converted_usd   Delay  \
0               2020-01-26               NAH4       54273.28  1 days
1               2019-07-22               NAD1       79656.60 -3 days
2               2019-09-14               NAA8        2253.86 92 days
3               2020-03-31               CA10        2309.79     NaT
4               2019-11-13               NAH4       33133.29 -3 days
…                    …                 …             …     …
49995           2020-04-21               NAA8        3187.86     NaT
49996           2019-08-15               NAH4        6766.54  4 days
49997           2020-02-19               NAA8        6120.86  0 days
```

```
49998           2019-11-27                NAA8          63.48  0 days
49999           2019-01-01                NAM4        1790.30 -9 days

                         avgdelay
0     -3 days +07:08:49.779837776
1                19 days 00:00:00
2       8 days 02:10:54.545454545
3       2 days 19:03:31.764705882
4     -3 days +19:33:27.692307693
…                             …
49995 -1 days +17:08:34.285714286
49996 -3 days +12:40:08.540925267
49997   1 days 01:08:34.285714285
49998   1 days 13:36:42.985074626
49999 -4 days +02:20:52.173913044

[48839 rows x 13 columns]
```

4.1.2 Observe that the "avg_delay" column is in days format. You need to change the format into seconds

- Days_format : 17 days 00:00:00
- Format in seconds : 1641600.0

[45]:
```
df["avgdelay"] = df["avgdelay"].dt.total_seconds()
```

4.1.3 Display the maindata dataframe

[46]:
```
df
```

[46]:
```
      business_code cust_number       name_customer  clear_date  \
0              U001  0200769623        WAL-MAR corp  2020-02-11
1              U001  0200980828              BEN E   2019-08-08
2              U001  0200792734         MDV/ trust  2019-12-30
3              CA02  0140105686            SYSC llc         NaT
4              U001  0200769623  WAL-MAR foundation  2019-11-25
…               …          …                  …          …
49995          U001  0200561861      CO corporation         NaT
49996          U001  0200769623          WAL-MAR co  2019-09-03
49997          U001  0200772595    SAFEW associates  2020-03-05
49998          U001  0200726979           BJ'S  llc  2019-12-12
49999          U001  0200020431            DEC corp  2019-01-15

       buisness_year        doc_id posting_date due_in_date  \
0             2020.0  1.930438e+09   2020-01-26  2020-02-10
1             2019.0  1.929646e+09   2019-07-22  2019-08-11
2             2019.0  1.929874e+09   2019-09-14  2019-09-29
```

```
3              2020.0  2.960623e+09   2020-03-30  2020-04-10
4              2019.0  1.930148e+09   2019-11-13  2019-11-28
…                …          …           …           …
49995          2020.0  1.930797e+09   2020-04-21  2020-05-06
49996          2019.0  1.929744e+09   2019-08-15  2019-08-30
49997          2020.0  1.930537e+09   2020-02-19  2020-03-05
49998          2019.0  1.930199e+09   2019-11-27  2019-12-12
49999          2019.0  1.928576e+09   2019-01-05  2019-01-24

      baseline_create_date cust_payment_terms  converted_usd   Delay  \
0               2020-01-26               NAH4       54273.28  1 days
1               2019-07-22               NAD1       79656.60 -3 days
2               2019-09-14               NAA8        2253.86 92 days
3               2020-03-31               CA10        2309.79     NaT
4               2019-11-13               NAH4       33133.29 -3 days
…                    …                  …              …      …
49995           2020-04-21               NAA8        3187.86     NaT
49996           2019-08-15               NAH4        6766.54  4 days
49997           2020-02-19               NAA8        6120.86  0 days
49998           2019-11-27               NAA8          63.48  0 days
49999           2019-01-01               NAM4        1790.30 -9 days

           avgdelay
0     -2.334702e+05
1      1.641600e+06
2      6.990545e+05
3      2.414118e+05
4     -1.887923e+05
…               …
49995 -2.468571e+04
49996 -2.135915e+05
49997  9.051429e+04
49998  1.354030e+05
49999 -3.371478e+05

[48839 rows x 13 columns]
```

4.1.4 Since you have created the "avg_delay" column from "Delay" and "clear_date" column, there is no need of these two columns anymore

- You are expected to drop "Delay" and "clear_date" columns from maindata dataframe

[47]:
```
df.drop(columns=["Delay", "clear_date"], inplace=True)

df
```

[47]:
```
      business_code cust_number       name_customer  buisness_year  \
0              U001  0200769623        WAL-MAR corp         2020.0
1              U001  0200980828              BEN E          2019.0
2              U001  0200792734          MDV/ trust         2019.0
3              CA02  0140105686            SYSC llc         2020.0
4              U001  0200769623  WAL-MAR foundation         2019.0
…               …          …                   …           …
49995          U001  0200561861      CO corporation         2020.0
49996          U001  0200769623          WAL-MAR co         2019.0
49997          U001  0200772595    SAFEW associates         2020.0
49998          U001  0200726979           BJ'S  llc         2019.0
49999          U001  0200020431            DEC corp         2019.0

             doc_id posting_date due_in_date baseline_create_date  \
0      1.930438e+09   2020-01-26  2020-02-10           2020-01-26
1      1.929646e+09   2019-07-22  2019-08-11           2019-07-22
2      1.929874e+09   2019-09-14  2019-09-29           2019-09-14
3      2.960623e+09   2020-03-30  2020-04-10           2020-03-31
4      1.930148e+09   2019-11-13  2019-11-28           2019-11-13
…               …          …           …                   …
49995  1.930797e+09   2020-04-21  2020-05-06           2020-04-21
49996  1.929744e+09   2019-08-15  2019-08-30           2019-08-15
49997  1.930537e+09   2020-02-19  2020-03-05           2020-02-19
49998  1.930199e+09   2019-11-27  2019-12-12           2019-11-27
49999  1.928576e+09   2019-01-05  2019-01-24           2019-01-01

      cust_payment_terms  converted_usd      avgdelay
0                   NAH4       54273.28 -2.334702e+05
1                   NAD1       79656.60  1.641600e+06
2                   NAA8        2253.86  6.990545e+05
3                   CA10        2309.79  2.414118e+05
4                   NAH4       33133.29 -1.887923e+05
…                    …            …            …
49995               NAA8        3187.86 -2.468571e+04
49996               NAH4        6766.54 -2.135915e+05
49997               NAA8        6120.86  9.051429e+04
49998               NAA8          63.48  1.354030e+05
49999               NAM4        1790.30 -3.371478e+05

[48839 rows x 11 columns]
```

5 Splitting of Train and the Test Data

5.0.1 You need to split the "maindata" columns into X and y dataframe

- Note - y should have the target column i.e. "avg_delay" and the other column should be in X

- X is going to hold the source fields and y will be going to hold the target fields

[48]:
```
df = df.dropna()
```

[49]:
```
X_train, X_loc_test, y_train, y_local_test = train_test_split(
    df, df["avgdelay"], train_size=0.60
)
```

[50]:
```
X_train
```

[50]:
```
      business_code cust_number       name_customer  buisness_year  \
9096           U001  0200705742     DOT corporation         2020.0
37667          U001  0200744019     TARG associates         2019.0
17974          U001  0200148860            DOLLA co         2020.0
26144          U001  0200799367            MCL corp         2019.0
17743          U001  0200769623  WAL-MAR corporation        2019.0
…               …         …                   …              …
25802          U001  0200756072           REINHA in         2019.0
38417          U001  0200803720           DEC trust         2019.0
13442          U001   200769623  WAL-MAR corporation        2019.0
36678          U001  0200732755           KROGER us         2019.0
10282          U001   200729290           KROGER in         2020.0

             doc_id posting_date due_in_date baseline_create_date  \
9096   1.930607e+09   2020-03-05  2020-03-25           2020-03-05
37667  1.929621e+09   2019-07-18  2019-08-02           2019-07-18
17974  1.930594e+09   2020-03-03  2020-03-18           2020-03-03
26144  1.929702e+09   2019-08-05  2019-10-19           2019-08-05
17743  1.928613e+09   2019-01-13  2019-01-28           2019-01-13
…               …          …         …                  …
25802  1.929983e+09   2019-10-10  2019-10-25           2019-10-10
38417  1.930228e+09   2019-12-06  2019-12-11           2019-12-01
13442  1.929578e+09   2019-07-06  2019-07-21           2019-07-06
36678  1.928742e+09   2019-02-07  2019-02-22           2019-02-07
10282  1.930747e+09   2020-04-05  2020-04-20           2020-04-05

      cust_payment_terms  converted_usd       avgdelay
9096                NAD1       21966.55 -625959.183673
37667               NAA8        5683.13  192270.422535
17974               NAA8      103910.32 -660342.857143
26144               NAWN        8772.02 -101828.571429
17743               NAH4       19565.42 -218946.589595
…                    …             …              …
25802               NAA8         474.10  486000.000000
38417               NAM2        5326.14 -312289.156627
13442               NAH4       52426.59 -218946.589595
36678               NAA8       11029.37   57600.000000
```

```
10282                NAA8       28343.81   50269.090909

[29103 rows x 11 columns]
```

You are expected to split both the dataframes into train and test format in 60:40 ratio

- Note - The expected output should be in “X_train”, “X_loc_test”, “y_train”, “y_loc_test” format

5.0.2 Please check for the number of rows and columns of all the new dataframes (all 4)

[51]:
```
X_train.shape, X_loc_test.shape, y_train.shape, y_local_test.shape
```

[51]:
```
((29103, 11), (19403, 11), (29103,), (19403,))
```

5.0.3 Now you are expected to split the “X_loc_test” and “y_loc_test” dataset into “Test” and “Validation” (as the names given below) dataframe with 50:50 format

- Note - The expected output should be in “X_val”, “X_test”, “y_val”, “y_test” format

[52]:
```
X_val, X_test, y_val, y_test = train_test_split(
    X_loc_test, y_local_test, train_size=0.5
)
```

[53]:
```
X_train["avgdelay"] = X_train["avgdelay"].fillna(0)
X_test["avgdelay"] = X_test["avgdelay"].fillna(0)
X_val["avgdelay"] = X_val["avgdelay"].fillna(0)
```

5.0.4 Please check for the number of rows and columns of all the 4 dataframes

[54]:
```
X_val.shape, X_test.shape, y_val.shape, y_test.shape
```

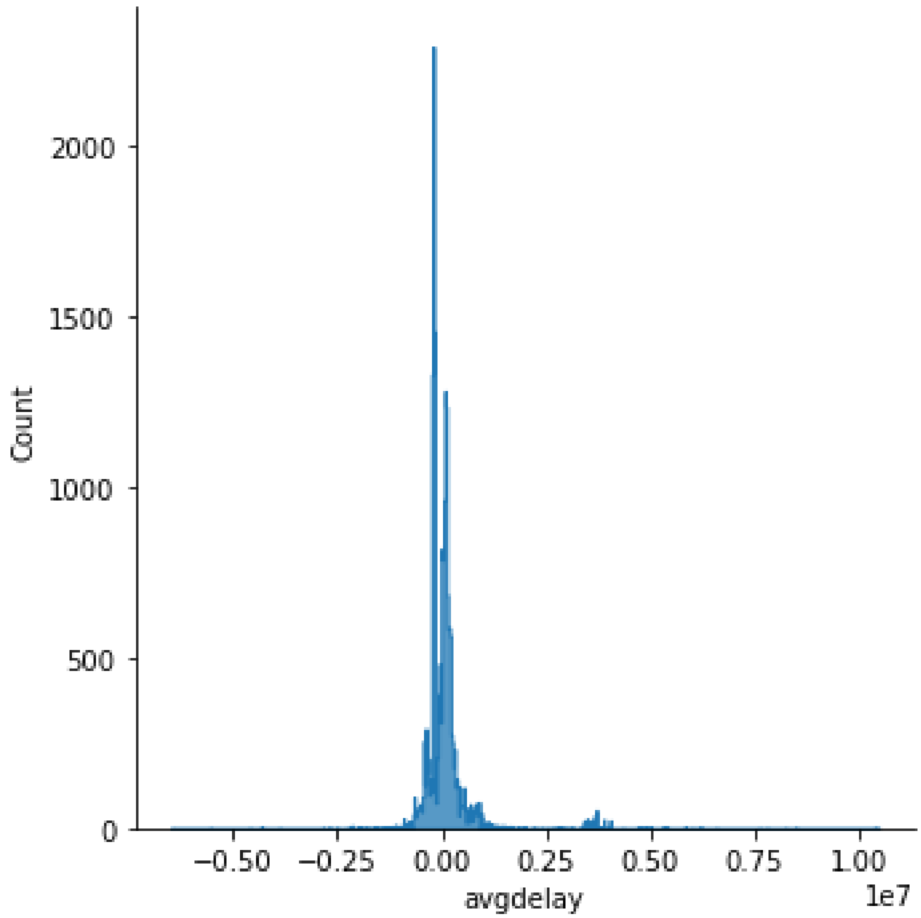
[54]:
```
((9701, 11), (9702, 11), (9701,), (9702,))
```

6 Exploratory Data Analysis (EDA)

6.0.1 Distribution Plot of the target variable (use the dataframe which contains the target field)

- Note - You are expected to make a distribution plot for the target variable

[55]:
```
sns.displot(y_local_test, element="step")
plt.show()
```



6.0.2 You are expected to group the X_train dataset on 'name_customer' column with 'doc_id' in the x_train set

6.0.3 Need to store the outcome into a new dataframe

- Note code given for groupby statement- X_train.groupby(by=['name_customer'], as_index=False)['doc_id'].count()

[56]:
```
x_train = X_train.groupby(by=["name_customer"], as_index=False)[
    "doc_id"
].count()

x_train
```

[56]:

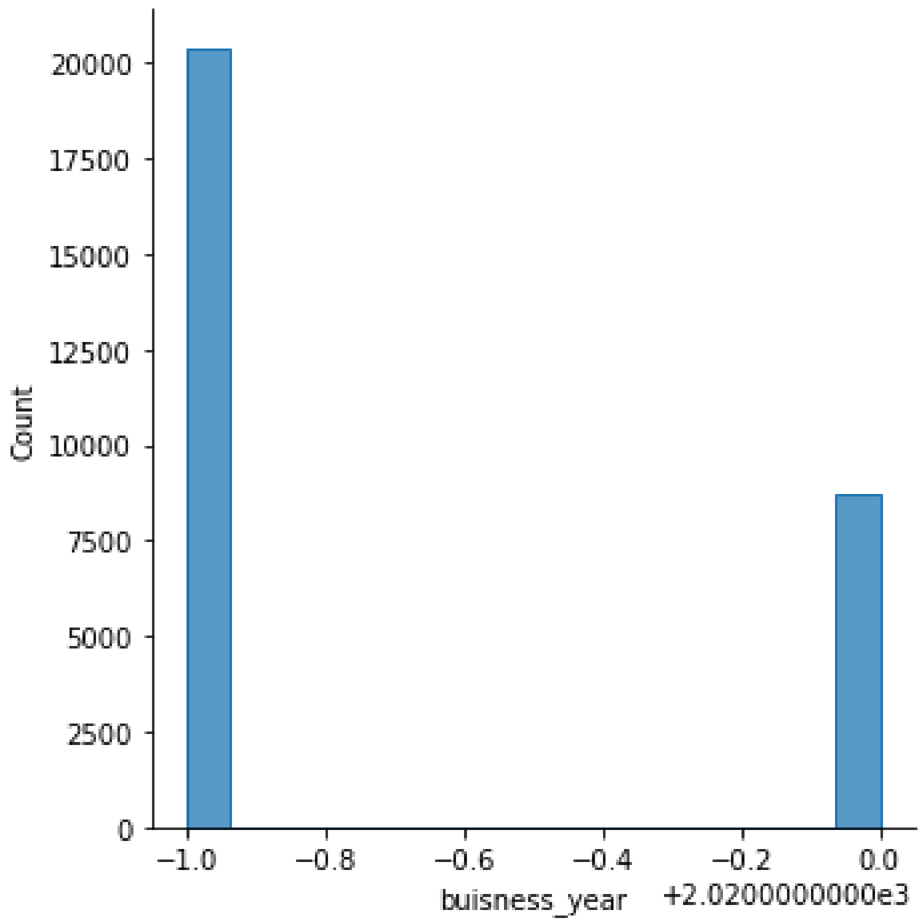
	name_customer	doc_id
0	11078 us	1
1	17135 associates	1
2	17135 llc	1
3	236008 associates	1

```
4                    99 CE        2
…                     …     …
3154              YEN BROS        1
3155          YEN BROS corp       1
3156  YEN BROS corporation        2
3157               ZARCO co       1
3158              ZIYAD  us       1

[3159 rows x 2 columns]
```

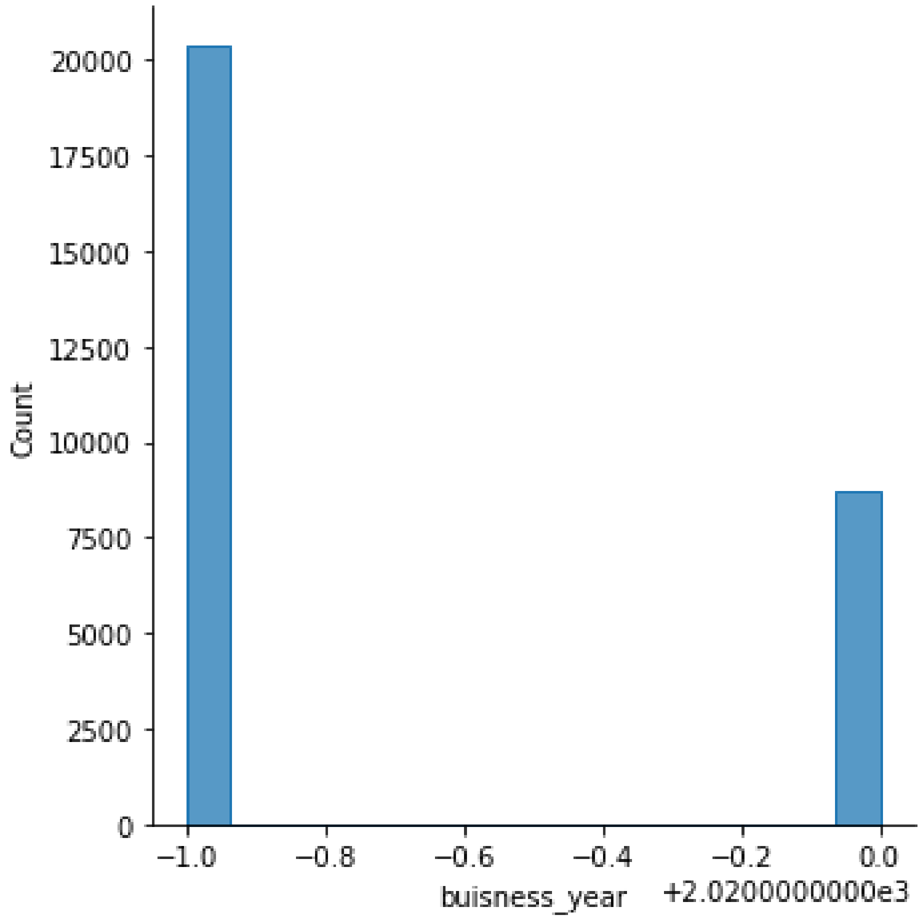
6.0.4 You can make another distribution plot of the "doc_id" column from x_train

[57]:
```
sns.displot(X_train["buisness_year"], element="step")
plt.show()
```

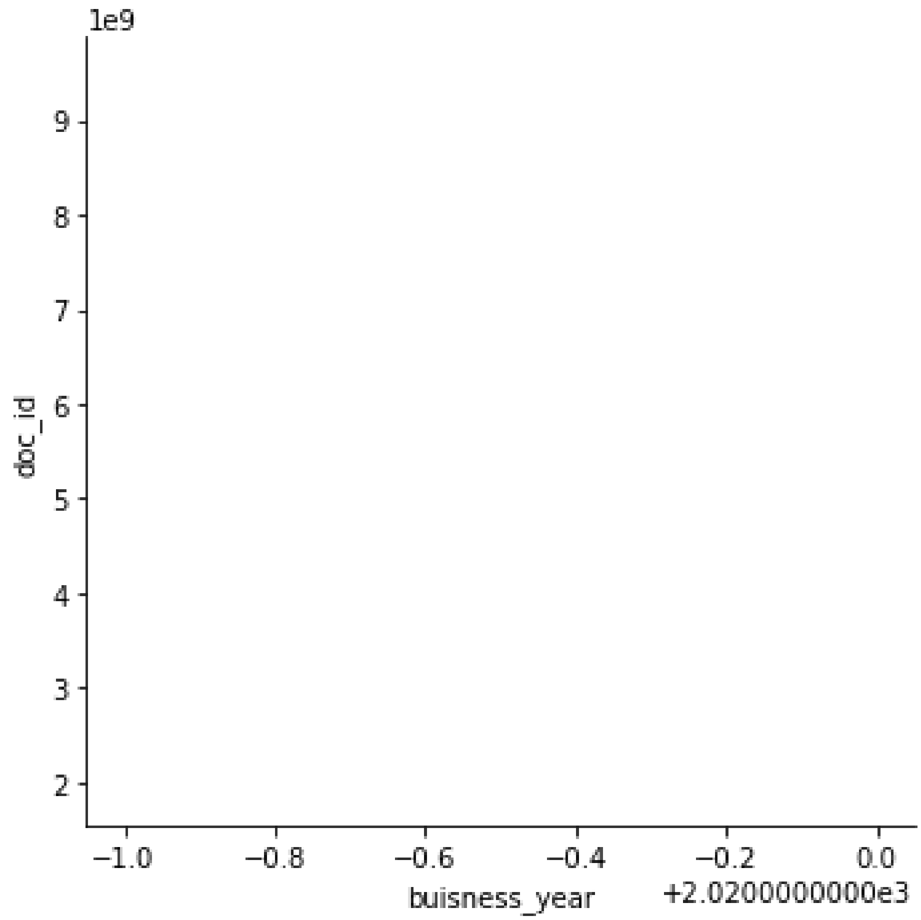


Create a Distribution plot only for business_year and a seperate distribution plot of "business_year" column along with the doc_id" column

[58]:
```
sns.displot(X_train["buisness_year"], element="step")
plt.show()
```



[59]:
```
sns.displot(x=X_train["buisness_year"], y=X_train["doc_id"])
plt.show()
```



7 Feature Engineering

7.0.1 Display and describe the X_train dataframe

[60]:
```
X_train
```

[60]:
```
      business_code cust_number        name_customer  buisness_year  \
9096           U001  0200705742      DOT corporation         2020.0
37667          U001  0200744019      TARG associates         2019.0
17974          U001  0200148860             DOLLA co         2020.0
26144          U001  0200799367             MCL corp         2019.0
17743          U001  0200769623  WAL-MAR corporation         2019.0
…                …           …                    …              …
25802          U001  0200756072            REINHA in         2019.0
38417          U001  0200803720            DEC trust         2019.0
13442          U001   200769623  WAL-MAR corporation         2019.0
36678          U001  0200732755            KROGER us         2019.0
```

```
10282          U001   200729290           KROGER in         2020.0

             doc_id posting_date due_in_date baseline_create_date  \
9096   1.930607e+09   2020-03-05  2020-03-25           2020-03-05
37667  1.929621e+09   2019-07-18  2019-08-02           2019-07-18
17974  1.930594e+09   2020-03-03  2020-03-18           2020-03-03
26144  1.929702e+09   2019-08-05  2019-10-19           2019-08-05
17743  1.928613e+09   2019-01-13  2019-01-28           2019-01-13
…               …          …          …                   …
25802  1.929983e+09   2019-10-10  2019-10-25           2019-10-10
38417  1.930228e+09   2019-12-06  2019-12-11           2019-12-01
13442  1.929578e+09   2019-07-06  2019-07-21           2019-07-06
36678  1.928742e+09   2019-02-07  2019-02-22           2019-02-07
10282  1.930747e+09   2020-04-05  2020-04-20           2020-04-05

      cust_payment_terms  converted_usd       avgdelay
9096                NAD1       21966.55 -625959.183673
37667               NAA8        5683.13  192270.422535
17974               NAA8      103910.32 -660342.857143
26144               NAWN        8772.02 -101828.571429
17743               NAH4       19565.42 -218946.589595
…                    …            …            …
25802               NAA8         474.10  486000.000000
38417               NAM2        5326.14 -312289.156627
13442               NAH4       52426.59 -218946.589595
36678               NAA8       11029.37   57600.000000
10282               NAA8       28343.81   50269.090909

[29103 rows x 11 columns]
```

[61]:
```python
X_train.describe()
```

[61]:

	buisness_year	doc_id	converted_usd	avgdelay
count	29103.000000	2.910300e+04	29103.000000	2.910300e+04
mean	2019.300072	2.014118e+09	31177.944213	6.166169e+04
std	0.458297	2.957865e+08	36656.610312	6.471071e+05
min	2019.000000	1.928502e+09	0.720000	-7.689600e+06
25%	2019.000000	1.929347e+09	4728.600000	-2.053220e+05
50%	2019.000000	1.929957e+09	17280.610000	1.080000e+04
75%	2020.000000	1.930610e+09	46141.095000	1.354030e+05
max	2020.000000	9.500000e+09	668593.360000	1.062720e+07

The "business_code" column inside X__train, is a categorical column, so you need to perform Labelencoder on that particular column

- Note - call the Label Encoder from sklearn library and use the fit() function on "business_code" column

- Note - Please fill in the blanks (two) to complete this code

[62]:
```
business_coder = LabelEncoder()
business_coder.fit(X_train["business_code"])
```

[62]:
```
LabelEncoder()
```

You are expected to store the value into a new column i.e. "business_code_enc"

- Note - For Training set you are expected to use fit_trainsform()
- Note - For Test set you are expected to use the trainsform()
- Note - For Validation set you are expected to use the trainsform()
- Partial code is provided, please fill in the blanks

[63]:
```
X_train["business_code_enc"] = business_coder.fit_transform(
    X_train["business_code"]
)
```

[64]:
```
X_val["business_code_enc"] = business_coder.transform(X_val["business_code"])
X_test["business_code_enc"] = business_coder.transform(X_test["business_code"])
```

7.0.2 Display "business_code" and "business_code_enc" together from X_train dataframe

[65]:
```
X_train.loc[:, ["business_code", "business_code_enc"]]
```

[65]:

	business_code	business_code_enc
9096	U001	1
37667	U001	1
17974	U001	1
26144	U001	1
17743	U001	1
…	…	…
25802	U001	1
38417	U001	1
13442	U001	1
36678	U001	1
10282	U001	1

```
[29103 rows x 2 columns]
```

Create a function called "custom" for dropping the columns 'business_code' from train, test and validation dataframe

- Note - Fill in the blank to complete the code

[66]:
```
def custom(col, traindf=X_train, valdf=X_val, testdf=X_test):
    traindf.drop(col, axis=1, inplace=True)
    valdf.drop(col, axis=1, inplace=True)
    testdf.drop(col, axis=1, inplace=True)

    return traindf, valdf, testdf
```

7.0.3 Call the function by passing the column name which needed to be dropped from train, test and validation dataframes. Return updated dataframes to be stored in X_train ,X_val, X_test

- Note = Fill in the blank to complete the code

[67]:
```
X_train, X_val, X_test = custom(["business_code"])
```

7.0.4 Manually replacing str values with numbers, Here we are trying manually replace the customer numbers with some specific values like, 'CCCA' as 1, 'CCU' as 2 and so on. Also we are converting the datatype "cust_number" field to int type.

- We are doing it for all the three dataframes as shown below. This is fully completed code. No need to modify anything here

[68]:
```
X_train["cust_number"] = (
    X_train["cust_number"]
    .str.replace("CCCA", "1")
    .str.replace("CCU", "2")
    .str.replace("CC", "3")
    .astype(int)
)
X_test["cust_number"] = (
    X_test["cust_number"]
    .str.replace("CCCA", "1")
    .str.replace("CCU", "2")
    .str.replace("CC", "3")
    .astype(int)
)
X_val["cust_number"] = (
    X_val["cust_number"]
    .str.replace("CCCA", "1")
    .str.replace("CCU", "2")
    .str.replace("CC", "3")
    .astype(int)
)
```

It differs from LabelEncoder by handling new classes and providing a value for it [Unknown]. Unknown will be added in fit and transform will take care of new item.

It gives unknown class id.

This will fit the encoder for all the unique values and introduce unknown value

- Note - Keep this code as it is, we will be using this later on.

[69]:
```
# For encoding unseen labels
class EncoderExt(object):
    def __init__(self):
        self.label_encoder = LabelEncoder()

    def fit(self, data_list):
        self.label_encoder = self.label_encoder.fit(
            list(data_list) + ["Unknown"]
        )
        self.classes_ = self.label_encoder.classes_
        return self

    def transform(self, data_list):
        new_data_list = list(data_list)
        for unique_item in np.unique(data_list):
            if unique_item not in self.label_encoder.classes_:
                new_data_list = [
                    "Unknown" if x == unique_item else x for x in new_data_list
                ]
        return self.label_encoder.transform(new_data_list)
```

7.0.5 Use the user define Label Encoder function called "EncoderExt" for the "name_customer" column

- Note - Keep the code as it is, no need to change

[70]:
```
label_encoder = EncoderExt()
label_encoder.fit(X_train["name_customer"])
X_train["name_customer_enc"] = label_encoder.transform(X_train["name_customer"])
X_val["name_customer_enc"] = label_encoder.transform(X_val["name_customer"])
X_test["name_customer_enc"] = label_encoder.transform(X_test["name_customer"])
```

7.0.6 As we have created the a new column "name_customer_enc", so now drop "name_customer" column from all three dataframes

- Note - Keep the code as it is, no need to change

[71]:
```
X_train, X_val, X_test = custom(["name_customer"])
```

7.0.7 Using Label Encoder for the "cust_payment_terms" column

- Note - Keep the code as it is, no need to change

[72]:
```
label_encoder1 = EncoderExt()
label_encoder1.fit(X_train["cust_payment_terms"])
X_train["cust_payment_terms_enc"] = label_encoder1.transform(
    X_train["cust_payment_terms"]
)
X_val["cust_payment_terms_enc"] = label_encoder1.transform(
    X_val["cust_payment_terms"]
)
X_test["cust_payment_terms_enc"] = label_encoder1.transform(
    X_test["cust_payment_terms"]
)
```

[73]:
```
X_train, X_val, X_test = custom(["cust_payment_terms"])
```

7.1 Check the datatype of all the columns of Train, Test and Validation dataframes realted to X

- Note - You are expected yo use dtype

[74]:
```
X_train.dtypes
```

[74]:
```
cust_number                  int64
buisness_year              float64
doc_id                     float64
posting_date                object
due_in_date                 object
baseline_create_date        object
converted_usd              float64
avgdelay                   float64
business_code_enc            int64
name_customer_enc            int64
cust_payment_terms_enc       int64
dtype: object
```

[75]:
```
X_val.dtypes
```

[75]:
```
cust_number                  int64
buisness_year              float64
doc_id                     float64
posting_date                object
due_in_date                 object
baseline_create_date        object
converted_usd              float64
avgdelay                   float64
business_code_enc            int64
name_customer_enc            int64
cust_payment_terms_enc       int64
```

```
dtype: object
```

[76]:
```
X_test.dtypes
```

[76]:
```
cust_number                   int64
buisness_year               float64
doc_id                      float64
posting_date                 object
due_in_date                  object
baseline_create_date         object
converted_usd               float64
avgdelay                    float64
business_code_enc             int64
name_customer_enc             int64
cust_payment_terms_enc        int64
dtype: object
```

7.1.1 From the above output you can notice their are multiple date columns with datetime format

7.1.2 In order to pass it into our, we need to convert it into float format

7.1.3 You need to extract day, month and year from the "posting_date" column

1. Extract days from "posting_date" column and store it into a new column "day_of_postingdate" for train, test and validation dataset
2. Extract months from "posting_date" column and store it into a new column "month_of_postingdate" for train, test and validation dataset
3. Extract year from "posting_date" column and store it into a new column "year_of_postingdate" for train, test and validation dataset

- Note - You are supposed yo use
- dt.day
- dt.month
- dt.year

[77]:
```
X_train["day_of_postingdate"] = pd.to_datetime(
    X_train["posting_date"], format="%Y-%m-%d"
).dt.day
X_train["month_of_postingdate"] = pd.to_datetime(
    X_train["posting_date"], format="%Y-%m-%d"
).dt.month
X_train["year_of_postingdate"] = pd.to_datetime(
    X_train["posting_date"], format="%Y-%m-%d"
).dt.year

X_val["day_of_postingdate"] = pd.to_datetime(
```

```
    X_val["posting_date"], format="%Y-%m-%d"
).dt.day
X_val["month_of_postingdate"] = pd.to_datetime(
    X_val["posting_date"], format="%Y-%m-%d"
).dt.month
X_val["year_of_postingdate"] = pd.to_datetime(
    X_val["posting_date"], format="%Y-%m-%d"
).dt.year

X_test["day_of_postingdate"] = pd.to_datetime(
    X_test["posting_date"], format="%Y-%m-%d"
).dt.day
X_test["month_of_postingdate"] = pd.to_datetime(
    X_test["posting_date"], format="%Y-%m-%d"
).dt.month
X_test["year_of_postingdate"] = pd.to_datetime(
    X_test["posting_date"], format="%Y-%m-%d"
).dt.year
```

7.1.4 pass the "posting_date" column into the Custom function for train, test and validation dataset

[78]:
```
X_train, X_val, X_test = custom(["posting_date"])
```

7.1.5 You need to extract day, month and year from the "baseline_create_date" column

1. Extract days from "baseline_create_date" column and store it into a new column "day_of_createdate" for train, test and validation dataset
2. Extract months from "baseline_create_date" column and store it into a new column "month_of_createdate" for train, test and validation dataset
3. Extract year from "baseline_create_date" column and store it into a new column "year_of_createdate" for train, test and validation dataset

- Note - You are supposed yo use
- dt.day
- dt.month
- dt.year
- Note - Do as it is been shown in the previous two code boxes

7.1.6 Extracting Day, Month, Year for 'baseline_create_date' column

[79]:
```
X_train["day_of_baselinecreatedate"] = pd.to_datetime(
    X_train["baseline_create_date"], format="%Y-%m-%d"
).dt.day
```

```
X_train["month_of_baselinecreatedate"] = pd.to_datetime(
    X_train["baseline_create_date"], format="%Y-%m-%d"
).dt.month
X_train["year_of_baselinecreatedate"] = pd.to_datetime(
    X_train["baseline_create_date"], format="%Y-%m-%d"
).dt.year

X_val["day_of_baselinecreatedate"] = pd.to_datetime(
    X_val["baseline_create_date"], format="%Y-%m-%d"
).dt.day
X_val["month_of_baselinecreatedate"] = pd.to_datetime(
    X_val["baseline_create_date"], format="%Y-%m-%d"
).dt.month
X_val["year_of_baselinecreatedate"] = pd.to_datetime(
    X_val["baseline_create_date"], format="%Y-%m-%d"
).dt.year

X_test["day_of_baselinecreatedate"] = pd.to_datetime(
    X_test["baseline_create_date"], format="%Y-%m-%d"
).dt.day
X_test["month_of_baselinecreatedate"] = pd.to_datetime(
    X_test["baseline_create_date"], format="%Y-%m-%d"
).dt.month
X_test["year_of_baselinecreatedate"] = pd.to_datetime(
    X_test["baseline_create_date"], format="%Y-%m-%d"
).dt.year
```

7.1.7 pass the "baseline_create_date" column into the Custom function for train, test and validation dataset

[80]:
```
X_train, X_val, X_test = custom(["baseline_create_date"])
```

7.1.8 You need to extract day, month and year from the "due_in_date" column

1. Extract days from "due_in_date" column and store it into a new column "day_of_due" for train, test and validation dataset
2. Extract months from "due_in_date" column and store it into a new column "month_of_due" for train, test and validation dataset
3. Extract year from "due_in_date" column and store it into a new column "year_of_due" for train, test and validation dataset

- Note - You are supposed yo use
- dt.day
- dt.month
- dt.year
- Note - Do as it is been shown in the previous code

[81]:

```
X_train["day_of_dueindate"] = pd.to_datetime(
    X_train["due_in_date"], format="%Y-%m-%d"
).dt.day
X_train["month_of_dueindate"] = pd.to_datetime(
    X_train["due_in_date"], format="%Y-%m-%d"
).dt.month
X_train["year_of_dueindate"] = pd.to_datetime(
    X_train["due_in_date"], format="%Y-%m-%d"
).dt.year

X_val["day_of_dueindate"] = pd.to_datetime(
    X_val["due_in_date"], format="%Y-%m-%d"
).dt.day
X_val["month_of_dueindate"] = pd.to_datetime(
    X_val["due_in_date"], format="%Y-%m-%d"
).dt.month
X_val["year_of_dueindate"] = pd.to_datetime(
    X_val["due_in_date"], format="%Y-%m-%d"
).dt.year

X_test["day_of_dueindate"] = pd.to_datetime(
    X_test["due_in_date"], format="%Y-%m-%d"
).dt.day
X_test["month_of_dueindate"] = pd.to_datetime(
    X_test["due_in_date"], format="%Y-%m-%d"
).dt.month
X_test["year_of_dueindate"] = pd.to_datetime(
    X_test["due_in_date"], format="%Y-%m-%d"
).dt.year
```

pass the "due_in_date" column into the Custom function for train, test and validation dataset

[82]:

```
X_train, X_val, X_test = custom(["due_in_date"])
```

7.1.9 Check for the datatypes for train, test and validation set again

- Note - all the data type should be in either int64 or float64 format

[83]:

```
pd.DataFrame([X_train.dtypes, X_val.dtypes, X_test.dtypes])
```

[83]:

```
  cust_number buisness_year   doc_id converted_usd avgdelay business_code_enc  \
0       int64       float64  float64       float64  float64             int64
1       int64       float64  float64       float64  float64             int64
2       int64       float64  float64       float64  float64             int64

  name_customer_enc cust_payment_terms_enc day_of_postingdate  \
0             int64                  int64              int64
1             int64                  int64              int64
```

```
2              int64                    int64                int64

  month_of_postingdate year_of_postingdate day_of_baselinecreatedate  \
0                int64               int64                     int64
1                int64               int64                     int64
2                int64               int64                     int64

  month_of_baselinecreatedate year_of_baselinecreatedate day_of_dueindate  \
0                       int64                      int64            int64
1                       int64                      int64            int64
2                       int64                      int64            int64

  month_of_dueindate year_of_dueindate
0              int64             int64
1              int64             int64
2              int64             int64
```

8 Feature Selection

8.0.1 Filter Method

- Calling the VarianceThreshold Function
- Note - Keep the code as it is, no need to change

[84]:
```
constant_filter = VarianceThreshold(threshold=0)
constant_filter.fit(X_train)
len(X_train.columns[constant_filter.get_support()])
```

[84]:
```
17
```

- Note - Keep the code as it is, no need to change

[85]:
```
constant_columns = [
    column
    for column in X_train.columns
    if column not in X_train.columns[constant_filter.get_support()]
]
print(len(constant_columns))
```

```
0
```

- transpose the feature matrice
- print the number of duplicated features
- select the duplicated features columns names
- Note - Keep the code as it is, no need to change

[86]:
```
x_train_T = X_train.T
print(x_train_T.duplicated().sum())
duplicated_columns = x_train_T[x_train_T.duplicated()].index.values
```

```
0
```

8.0.2 Filtering depending upon correlation matrix value

- We have created a function called handling correlation which is going to return fields based on the correlation matrix value with a threshold of 0.8
- Note - Keep the code as it is, no need to change

[87]:
```
def handling_correlation(X_train, threshold=0.8):
    corr_features = set()
    corr_matrix = X_train.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold:
                colname = corr_matrix.columns[i]
                corr_features.add(colname)
    return list(corr_features)
```

- Note : Here we are trying to find out the relevant fields, from X_train
- Please fill in the blanks to call handling_correlation() function with a threshold value of 0.85

[88]:
```
train = X_train.copy()
handling_correlation(train.copy(), 0.85)
```

[88]:
```
['year_of_baselinecreatedate',
 'day_of_baselinecreatedate',
 'month_of_dueindate',
 'year_of_postingdate',
 'month_of_baselinecreatedate',
 'year_of_dueindate']
```

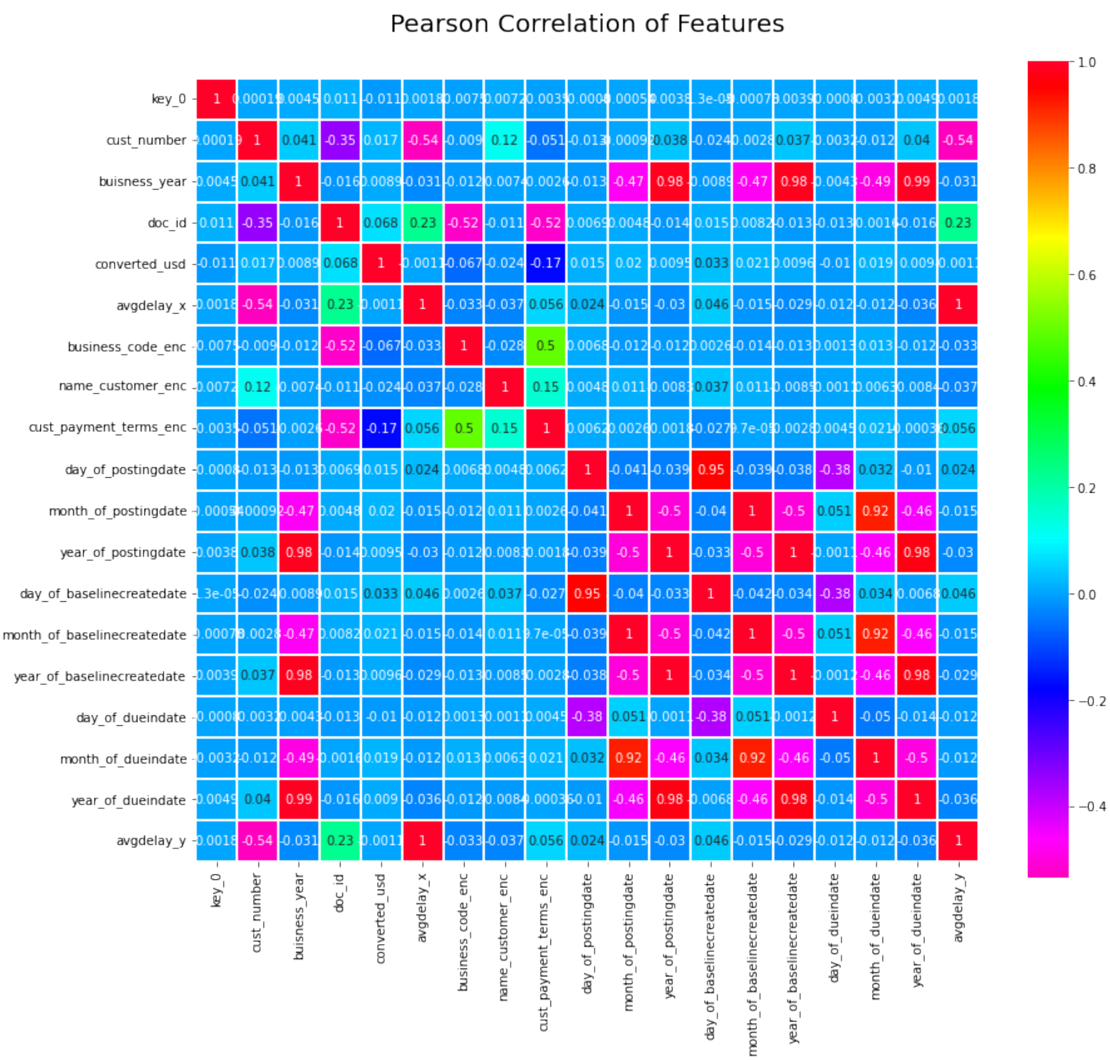
8.0.3 Heatmap for X_train

- Note - Keep the code as it is, no need to change

[89]:
```
colormap = plt.cm.RdBu
plt.figure(figsize=(14, 12))
plt.title("Pearson Correlation of Features", y=1.05, size=20)
sns.heatmap(
    X_train.merge(y_train, on=X_train.index).corr(),
    linewidths=0.1,
    vmax=1.0,
    square=True,
    cmap="gist_rainbow_r",
```

```
    linecolor="white",
    annot=True,
)

plt.show()
```



Calling variance threshold for threshold value = 0.8

- Note - Fill in the blanks to call the appropriate method

[90]:
```
sel = VarianceThreshold(0.8)
sel.fit(X_train)
```

[90]: `VarianceThreshold(threshold=0.8)`

[91]:
```
sel.variances_
```

[91]:
```
array([1.65606317e+15, 2.10028858e-01, 8.74866240e+16, 1.34366091e+09,
       4.18733271e+11, 2.53048105e-01, 1.13304023e+06, 1.38965448e+02,
       7.77025990e+01, 1.08006704e+01, 2.13044270e-01, 7.92272847e+01,
       1.08068290e+01, 2.13075088e-01, 7.49036886e+01, 1.05795343e+01,
       2.11431483e-01])
```

8.0.4 Important features columns are

- 'year_of_createdate'
- 'year_of_due'
- 'day_of_createdate'
- 'year_of_postingdate'
- 'month_of_due'
- 'month_of_createdate'

9 Modelling

Now you need to compare with different machine learning models, and needs to find out the best predicted model

- Linear Regression
- Decision Tree Regression
- Random Forest Regression
- Support Vector Regression
- Extreme Gradient Boost Regression

9.0.1 You need to make different blank list for different evaluation matrix

- MSE
- R2
- Algorithm

[92]:
```
MSE_Score = []
R2_Score = []
Algorithm = []
```

9.0.2 You need to start with the baseline model Linear Regression

- Step 1 : Call the Linear Regression from sklearn library
- Step 2 : make an object of Linear Regression
- Step 3 : fit the X_train and y_train dataframe into the object
- Step 4 : Predict the output by passing the X_test Dataset into predict function
- Note - Append the Algorithm name into the algorithm list for tracking purpose

[93]:
```
X_train.isna().any()
```

[93]:
```
cust_number                    False
buisness_year                  False
doc_id                         False
converted_usd                  False
avgdelay                       False
business_code_enc              False
name_customer_enc              False
cust_payment_terms_enc         False
day_of_postingdate             False
month_of_postingdate           False
year_of_postingdate            False
day_of_baselinecreatedate      False
month_of_baselinecreatedate    False
year_of_baselinecreatedate     False
day_of_dueindate               False
month_of_dueindate             False
year_of_dueindate              False
dtype: bool
```

9.0.3 Fix NaN by dropping and resizing the X_train

[94]:
```
# X_train = X_train.dropna()
# y_train = y_train.dropna()
# y_test = y_test.dropna()
```

[95]:
```
Algorithm.append("LinearRegression")
regressor = LinearRegression()
regressor.fit(X_train, y_train)
predicted = regressor.predict(X_test)
```

[96]:
```
predicted
```

[96]:
```
array([  33230.76923077,   -7200.        ,   53169.23076923, …,
       -205321.95734002,  108000.        ,  -55408.69565217])
```

9.0.4 Check for the

- Mean Square Error
- R Square Error

for y_test and predicted dataset and store those data inside respective list for comparison

[97]:
```
from sklearn.metrics import mean_squared_error, r2_score

MSE_Score.append(mean_squared_error(y_test, predicted))
```

```
R2_Score.append(r2_score(y_test, predicted))
```

9.0.5 Check the same for the Validation set also

[98]:
```
predict_test = regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

[98]: 7.295755323904816e-09

9.0.6 Display The Comparison Lists

[99]:
```
for i in Algorithm, MSE_Score, R2_Score:
    print(i, end=",")
```

```
['LinearRegression'],[5.4634169023269406e-17],[1.0],
```

9.0.7 You need to start with the baseline model Support Vector Regression

- Step 1 : Call the Support Vector Regressor from sklearn library
- Step 2 : make an object of SVR
- Step 3 : fit the X_train and y_train dataframe into the object
- Step 4 : Predict the output by passing the X_test Dataset into predict function
- Note - Append the Algorithm name into the algorithm list for tracking purpose

[100]:
```
Algorithm.append("SVR")
regressor = SVR()
regressor.fit(X_train, y_train)
predicted = regressor.predict(X_test)
```

9.0.8 Check for the

- Mean Square Error
- R Square Error

for "y_test" and "predicted" dataset and store those data inside respective list for comparison

[101]:
```
from sklearn.metrics import mean_squared_error, r2_score

MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

9.0.9 Check the same for the Validation set also

[102]:
```
predict_test = regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

[102]:
```
633598.5440599344
```

9.0.10 Display The Comparison Lists

[103]:
```
for i in Algorithm, MSE_Score, R2_Score:
    print(i, end=", ")
```

```
['LinearRegression', 'SVR'], [5.4634169023269406e-17, 373365685563.2006], [1.0,
-0.0037956319855831921],
```

9.0.11 Your next model would be Decision Tree Regression

- Step 1 : Call the Decision Tree Regressor from sklearn library
- Step 2 : make an object of Decision Tree
- Step 3 : fit the X_train and y_train dataframe into the object
- Step 4 : Predict the output by passing the X_test Dataset into predict function
- Note - Append the Algorithm name into the algorithm list for tracking purpose

[104]:
```
Algorithm.append("DecisionTreeRegressor")
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
predicted = regressor.predict(X_test)
```

9.0.12 Check for the

- Mean Square Error
- R Square Error

for y_test and predicted dataset and store those data inside respective list for comparison

[105]:
```
from sklearn.metrics import mean_squared_error, r2_score

MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

9.0.13 Check the same for the Validation set also

[106]:
```
predict_test = regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

[106]:
```
33624.3228818838
```

9.0.14 Display The Comparison Lists

[107]:
```
for i in Algorithm, MSE_Score, R2_Score:
    print(i, end=", ")
```

```
['LinearRegression', 'SVR', 'DecisionTreeRegressor'], [5.4634169023269406e-17,
373365685563.2006, 111540670.85945106], [1.0, -0.003795631985583192,
0.999700122312985],
```

9.0.15 Your next model would be Random Forest Regression

- Step 1 : Call the Random Forest Regressor from sklearn library
- Step 2 : make an object of Random Forest
- Step 3 : fit the X_train and y_train dataframe into the object
- Step 4 : Predict the output by passing the X_test Dataset into predict function
- Note - Append the Algorithm name into the algorithm list for tracking purpose

[108]:
```
Algorithm.append("RandomForestRegressor")
regressor = RandomForestRegressor()
regressor.fit(X_train, y_train)
predicted = regressor.predict(X_test)
```

9.0.16 Check for the

- Mean Square Error
- R Square Error

for y_test and predicted dataset and store those data inside respective list for comparison

[109]:
```
from sklearn.metrics import mean_squared_error, r2_score

MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

9.0.17 Check the same for the Validation set also

[110]:
```
predict_test = regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

[110]: 17781.86603922322

9.0.18 Display The Comparison Lists

[111]:
```
for i in Algorithm, MSE_Score, R2_Score:
    print(i, end=", ")
```

```
['LinearRegression', 'SVR', 'DecisionTreeRegressor', 'RandomForestRegressor'],
[5.4634169023269406e-17, 373365685563.2006, 111540670.85945106,
55724795.84029109], [1.0, -0.003795631985583192, 0.999700122312985,
0.999850183589921],
```

9.0.19 The last but not the least model would be XGBoost or Extreme Gradient Boost Regression

- Step 1 : Call the XGBoost Regressor from xgb library
- Step 2 : make an object of Xgboost
- Step 3 : fit the X_train and y_train dataframe into the object
- Step 4 : Predict the output by passing the X_test Dataset into predict function
- Note - Append the Algorithm name into the algorithm list for tracking purpose### Extreme Gradient Boost Regression
- Note - No need to change the code

[112]:
```python
Algorithm.append("XGB Regressor")
regressor = xgb.XGBRegressor()
regressor.fit(X_train, y_train)
predicted = regressor.predict(X_test)
```

```
/opt/conda/lib/python3.9/site-packages/xgboost/data.py:262: FutureWarning:
pandas.Int64Index is deprecated and will be removed from pandas in a future
version. Use pandas.Index with the appropriate dtype instead.
  elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
```

9.0.20 Check for the

- Mean Square Error
- R Square Error

for y_test and predicted dataset and store those data inside respective list for comparison

[113]:
```python
from sklearn.metrics import mean_squared_error, r2_score

MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

9.0.21 Check the same for the Validation set also

[114]:
```python
predict_test = regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

```
/opt/conda/lib/python3.9/site-packages/xgboost/data.py:262: FutureWarning:
pandas.Int64Index is deprecated and will be removed from pandas in a future
version. Use pandas.Index with the appropriate dtype instead.
  elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
```

[114]: 11982.937234311521

9.0.22 Display The Comparison Lists

[115]:
```python
for i in Algorithm, MSE_Score, R2_Score:
    print(i, end=", ")
```

```
['LinearRegression', 'SVR', 'DecisionTreeRegressor', 'RandomForestRegressor',
'XGB Regressor'], [5.4634169023269406e-17, 373365685563.2006,
111540670.85945106, 55724795.84029109, 122522221.34443663], [1.0,
-0.003795631985583192, 0.999700122312985, 0.999850183589921, 0.999670598356083],
```

9.1 You need to make the comparison list into a comparison dataframe

[116]:
```python
pd.DataFrame([Algorithm, MSE_Score, R2_Score])
```

[116]:
```
                  0                     1                      2  \
0  LinearRegression                   SVR  DecisionTreeRegressor
1               0.0   373365685563.200623       111540670.859451
2               1.0             -0.003796                 0.9997

                       3                 4
0  RandomForestRegressor     XGB Regressor
1        55724795.840291  122522221.344437
2                0.99985          0.999671
```

9.2 Now from the Comparison table, you need to choose the best fit model

- Step 1 - Fit X_train and y_train inside the model
- Step 2 - Predict the X_test dataset
- Step 3 - Predict the X_val dataset
- Note - No need to change the code

[117]:
```python
regressorfinal = xgb.XGBRegressor()
regressorfinal.fit(X_train, y_train)
predictedfinal = regressorfinal.predict(X_test)
predict_testfinal = regressorfinal.predict(X_val)
```

```
/opt/conda/lib/python3.9/site-packages/xgboost/data.py:262: FutureWarning:
pandas.Int64Index is deprecated and will be removed from pandas in a future
version. Use pandas.Index with the appropriate dtype instead.
  elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
```

9.2.1 Calculate the Mean Square Error for test dataset

- Note - No need to change the code

[118]:
```
mean_squared_error(y_test, predictedfinal, squared=False)
```

[118]: 11068.975623084398

9.2.2 Calculate the mean Square Error for validation dataset

[119]:
```
mean_squared_error(y_val, predict_testfinal, squared=False)
```

[119]: 11982.937234311521

9.2.3 Calculate the R2 score for test

[120]:
```
r2_score(y_test, predictedfinal)
```

[120]: 0.999670598356083

9.2.4 Calculate the R2 score for Validation

[121]:
```
r2_score(y_val, predict_testfinal)
```

[121]: 0.9996403900262987

9.2.5 Calculate the Accuracy for train Dataset

[122]:
```
from sklearn.ensemble import RandomForestRegressor

# Choosing RandomForestRegressor because it performs good with floating point␣
 ↪numbers.

clf = RandomForestRegressor()
trained_model = clf.fit(X_train, y_train)
trained_model.fit(X_train, y_train)
predictions = trained_model.predict(X_test)
```

[123]:
```
clf.score(X_train, y_train)
```

[123]: 0.9998779592694278

9.2.6 Calculate the accuracy for validation

[124]:
```
clf.score(X_val, y_val)
```

[124]: 0.9987990708072391

9.2.7 Calculate the accuracy for test

[125]:
```
clf.score(X_test, y_test)
```

[125]:
```
0.9998401816872527
```

9.3 Specify the reason behind choosing your machine learning model

- Note : Choosing RandomForestRegressor because it **performs good with floating point numbers.**

9.4 Now you need to pass the Nulldata dataframe into this machine learning model

In order to pass this Nulldata dataframe into the ML model, we need to perform the following

- Step 1 : Label Encoding
- Step 2 : Day, Month and Year extraction
- Step 3 : Change all the column data type into int64 or float64
- Step 4 : Need to drop the useless columns

9.4.1 Display the Nulldata

[126]:
```
nulldata
```

[126]:
```
      business_code cust_number        name_customer  clear_date  \
0              U001  0200769623         WAL-MAR corp  2020-02-11
1              U001  0200980828               BEN E   2019-08-08
2              U001  0200792734           MDV/ trust  2019-12-30
4              U001  0200769623  WAL-MAR foundation  2019-11-25
5              CA02  0140106181     THE  corporation  2019-12-04
…                 …          …                   …          …
49994          U001  0200762301         C&S WH trust  2019-07-25
49996          U001  0200769623           WAL-MAR co  2019-09-03
49997          U001  0200772595     SAFEW associates  2020-03-05
49998          U001  0200726979            BJ'S  llc  2019-12-12
49999          U001  0200020431             DEC corp  2019-01-15

       buisness_year        doc_id posting_date due_in_date  \
0             2020.0  1.930438e+09   2020-01-26  2020-02-10
1             2019.0  1.929646e+09   2019-07-22  2019-08-11
2             2019.0  1.929874e+09   2019-09-14  2019-09-29
4             2019.0  1.930148e+09   2019-11-13  2019-11-28
5             2019.0  2.960581e+09   2019-09-20  2019-10-04
…                  …             …          …          …
49994         2019.0  1.929601e+09   2019-07-10  2019-07-25
49996         2019.0  1.929744e+09   2019-08-15  2019-08-30
```

```
49997          2020.0  1.930537e+09   2020-02-19  2020-03-05
49998          2019.0  1.930199e+09   2019-11-27  2019-12-12
49999          2019.0  1.928576e+09   2019-01-05  2019-01-24

      baseline_create_date cust_payment_terms  converted_usd
0               2020-01-26               NAH4      54273.280
1               2019-07-22               NAD1      79656.600
2               2019-09-14               NAA8       2253.860
4               2019-11-13               NAH4      33133.290
5               2019-09-24               CA10      15558.088
…                        …                  …              …
49994           2019-07-10               NAC6      84780.400
49996           2019-08-15               NAH4       6766.540
49997           2020-02-19               NAA8       6120.860
49998           2019-11-27               NAA8         63.480
49999           2019-01-01               NAM4       1790.300

[39158 rows x 11 columns]
```

9.4.2 Check for the number of rows and columns in the nulldata

[127]:
```
pd.DataFrame(nulldata.shape, index=["Rows", "Columns"])
```

[127]:
```
             0
Rows     39158
Columns     11
```

9.4.3 Check the Description and Information of the nulldata

[128]:
```
nulldata.describe()
```

[128]:
```
       buisness_year        doc_id  converted_usd
count   39158.000000  3.915800e+04   39158.000000
mean     2019.132361  2.013764e+09   30735.355408
std         0.338887  2.938359e+08   36530.556929
min      2019.000000  1.928502e+09       0.790000
25%      2019.000000  1.929181e+09    4527.342500
50%      2019.000000  1.929734e+09   16894.392000
75%      2019.000000  1.930209e+09   45462.315000
max      2020.000000  9.500000e+09  668593.360000
```

9.4.4 Storing the Nulldata into a different dataset

10 for BACKUP

[129]:
```
nulldata_copy = nulldata.copy()
```

10.0.1 Call the Label Encoder for Nulldata

- Note - you are expected to fit "business_code" as it is a categorical variable
- Note - No need to change the code

[130]:
```
from sklearn.preprocessing import LabelEncoder

business_codern = LabelEncoder()
business_codern.fit(nulldata["business_code"])
nulldata["business_code_enc"] = business_codern.transform(
    nulldata["business_code"]
)
```

```
/tmp/ipykernel_9134/1523850286.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#returning-a-view-versus-a-copy
  nulldata["business_code_enc"] = business_codern.transform(
```

10.0.2 Now you need to manually replacing str values with numbers

- Note - No need to change the code

[131]:
```
nulldata["cust_number"] = (
    nulldata["cust_number"]
    .str.replace("CCCA", "1")
    .str.replace("CCU", "2")
    .str.replace("CC", "3")
    .astype(int)
)
```

```
/tmp/ipykernel_9134/1817234853.py:1: 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#returning-a-view-versus-a-copy
  nulldata["cust_number"] = (
```

10.1 You need to extract day, month and year from the "clear_date", "posting_date", "due_in_date", "baseline_create_date" columns

1. Extract day from "clear_date" column and store it into 'day_of_cleardate'

2. Extract month from "clear_date" column and store it into 'month_of_cleardate'

3. Extract year from "clear_date" column and store it into 'year_of_cleardate'

4. Extract day from "posting_date" column and store it into 'day_of_postingdate'

5. Extract month from "posting_date" column and store it into 'month_of_postingdate'

6. Extract year from "posting_date" column and store it into 'year_of_postingdate'

7. Extract day from "due_in_date" column and store it into 'day_of_due'

8. Extract month from "due_in_date" column and store it into 'month_of_due'

9. Extract year from "due_in_date" column and store it into 'year_of_due'

10. Extract day from "baseline_create_date" column and store it into 'day_of_createdate'

11. Extract month from "baseline_create_date" column and store it into 'month_of_createdate'

12. Extract year from "baseline_create_date" column and store it into 'year_of_createdate'

- Note - You are supposed To use -
- dt.day
- dt.month
- dt.year

[132]:
```
nulldata["day_of_cleardate"] = pd.to_datetime(
    nulldata["clear_date"], format="%Y-%m-%d"
).dt.day

nulldata["month_of_cleardate"] = pd.to_datetime(
    nulldata["clear_date"], format="%Y-%m-%d"
).dt.month

nulldata["year_of_cleardate"] = pd.to_datetime(
    nulldata["clear_date"], format="%Y-%m-%d"
```

```
).dt.year
```

```
/tmp/ipykernel_9134/3068545488.py:1: 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#returning-a-view-versus-a-copy
  nulldata["day_of_cleardate"] = pd.to_datetime(
/tmp/ipykernel_9134/3068545488.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#returning-a-view-versus-a-copy
  nulldata["month_of_cleardate"] = pd.to_datetime(
/tmp/ipykernel_9134/3068545488.py:9: 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#returning-a-view-versus-a-copy
  nulldata["year_of_cleardate"] = pd.to_datetime(
```

[133]:

```
nulldata["day_of_postingdate"] = pd.to_datetime(
    nulldata["posting_date"], format="%Y-%m-%d"
).dt.day

nulldata["month_of_postingdate"] = pd.to_datetime(
    nulldata["posting_date"], format="%Y-%m-%d"
).dt.month

nulldata["year_of_postingdate"] = pd.to_datetime(
    nulldata["posting_date"], format="%Y-%m-%d"
).dt.year
```

```
/tmp/ipykernel_9134/3489165157.py:1: 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#returning-a-view-versus-a-copy
  nulldata["day_of_postingdate"] = pd.to_datetime(
/tmp/ipykernel_9134/3489165157.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#returning-a-view-versus-a-copy
  nulldata["month_of_postingdate"] = pd.to_datetime(
/tmp/ipykernel_9134/3489165157.py:9: 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#returning-a-view-versus-a-copy
  nulldata["year_of_postingdate"] = pd.to_datetime(
```

[134]:
```
nulldata["day_of_due"] = pd.to_datetime(
    nulldata["due_in_date"], format="%Y-%m-%d"
).dt.day

nulldata["month_of_due"] = pd.to_datetime(
    nulldata["due_in_date"], format="%Y-%m-%d"
).dt.month

nulldata["year_of_due"] = pd.to_datetime(
    nulldata["due_in_date"], format="%Y-%m-%d"
).dt.year
```

```
/tmp/ipykernel_9134/3265168148.py:1: 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#returning-a-view-versus-a-copy
  nulldata["day_of_due"] = pd.to_datetime(
/tmp/ipykernel_9134/3265168148.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#returning-a-view-versus-a-copy
  nulldata["month_of_due"] = pd.to_datetime(
/tmp/ipykernel_9134/3265168148.py:9: 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#returning-a-view-versus-a-copy
  nulldata["year_of_due"] = pd.to_datetime(
```

[135]:
```
nulldata["day_of_createdate"] = pd.to_datetime(
    nulldata["baseline_create_date"], format="%Y-%m-%d"
).dt.day
```

```
nulldata["month_of_createdate"] = pd.to_datetime(
    nulldata["baseline_create_date"], format="%Y-%m-%d"
).dt.month

nulldata["year_of_createdate"] = pd.to_datetime(
    nulldata["baseline_create_date"], format="%Y-%m-%d"
).dt.year
```

```
/tmp/ipykernel_9134/2175278356.py:1: 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#returning-a-view-versus-a-copy
  nulldata["day_of_createdate"] = pd.to_datetime(
/tmp/ipykernel_9134/2175278356.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#returning-a-view-versus-a-copy
  nulldata["month_of_createdate"] = pd.to_datetime(
/tmp/ipykernel_9134/2175278356.py:9: 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#returning-a-view-versus-a-copy
  nulldata["year_of_createdate"] = pd.to_datetime(
```

10.1.1 Use Label Encoder1 of all the following columns -

- 'cust_payment_terms' and store into 'cust_payment_terms_enc'
- 'business_code' and store into 'business_code_enc'
- 'name_customer' and store into 'name_customer_enc'

Note - No need to change the code

[136]:
```
nulldata["cust_payment_terms_enc"] = label_encoder1.transform(
    nulldata["cust_payment_terms"]
)
nulldata["business_code_enc"] = label_encoder1.transform(
    nulldata["business_code"]
)
nulldata["name_customer_enc"] = label_encoder.transform(
    nulldata["name_customer"]
)
```

```
/tmp/ipykernel_9134/2238043230.py:1: 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#returning-a-view-versus-a-copy
  nulldata["cust_payment_terms_enc"] = label_encoder1.transform(
/tmp/ipykernel_9134/2238043230.py:4: 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#returning-a-view-versus-a-copy
  nulldata["business_code_enc"] = label_encoder1.transform(
/tmp/ipykernel_9134/2238043230.py:7: 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#returning-a-view-versus-a-copy
  nulldata["name_customer_enc"] = label_encoder.transform(
```

10.1.2 Check for the datatypes of all the columns of Nulldata

[137]:
```
nulldata.dtypes
```

[137]:
```
business_code             object
cust_number                int64
name_customer             object
clear_date                object
buisness_year            float64
doc_id                   float64
posting_date              object
due_in_date               object
baseline_create_date      object
cust_payment_terms        object
converted_usd            float64
business_code_enc          int64
day_of_cleardate           int64
month_of_cleardate         int64
year_of_cleardate          int64
day_of_postingdate         int64
month_of_postingdate       int64
year_of_postingdate        int64
day_of_due                 int64
month_of_due               int64
year_of_due                int64
```

```
day_of_createdate            int64
month_of_createdate          int64
year_of_createdate           int64
cust_payment_terms_enc       int64
name_customer_enc            int64
dtype: object
```

10.1.3 Now you need to drop all the unnecessary columns -

- 'business_code'
- "baseline_create_date"
- "due_in_date"
- "posting_date"
- "name_customer"
- "clear_date"
- "cust_payment_terms"
- 'day_of_cleardate'
- "month_of_cleardate"
- "year_of_cleardate"

[138]:
```
nulldata.drop(
    columns=[
        "business_code",
        "baseline_create_date",
        "due_in_date",
        "posting_date",
        "name_customer",
        "clear_date",
        "cust_payment_terms",
        "day_of_cleardate",
        "year_of_cleardate",
    ],
    inplace=True,
)
```

```
/tmp/ipykernel_9134/669707772.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  nulldata.drop(
```

10.1.4 Check the information of the "nulldata" dataframe

[139]:
```
nulldata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 39158 entries, 0 to 49999
```

```
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   cust_number             39158 non-null  int64
 1   buisness_year           39158 non-null  float64
 2   doc_id                  39158 non-null  float64
 3   converted_usd           39158 non-null  float64
 4   business_code_enc       39158 non-null  int64
 5   month_of_cleardate      39158 non-null  int64
 6   day_of_postingdate      39158 non-null  int64
 7   month_of_postingdate    39158 non-null  int64
 8   year_of_postingdate     39158 non-null  int64
 9   day_of_due              39158 non-null  int64
 10  month_of_due            39158 non-null  int64
 11  year_of_due             39158 non-null  int64
 12  day_of_createdate       39158 non-null  int64
 13  month_of_createdate     39158 non-null  int64
 14  year_of_createdate      39158 non-null  int64
 15  cust_payment_terms_enc  39158 non-null  int64
 16  name_customer_enc       39158 non-null  int64
dtypes: float64(3), int64(14)
memory usage: 5.4 MB
```

10.1.5 Compare "nulldata" with the "X_test" dataframe

- use info() method

[140]:
```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9702 entries, 9643 to 16601
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   cust_number                  9702 non-null   int64
 1   buisness_year                9702 non-null   float64
 2   doc_id                       9702 non-null   float64
 3   converted_usd                9702 non-null   float64
 4   avgdelay                     9702 non-null   float64
 5   business_code_enc            9702 non-null   int64
 6   name_customer_enc            9702 non-null   int64
 7   cust_payment_terms_enc       9702 non-null   int64
 8   day_of_postingdate           9702 non-null   int64
 9   month_of_postingdate         9702 non-null   int64
 10  year_of_postingdate          9702 non-null   int64
 11  day_of_baselinecreatedate    9702 non-null   int64
 12  month_of_baselinecreatedate  9702 non-null   int64
 13  year_of_baselinecreatedate   9702 non-null   int64
```

```
 14   day_of_dueindate            9702 non-null   int64
 15   month_of_dueindate          9702 non-null   int64
 16   year_of_dueindate           9702 non-null   int64
dtypes: float64(4), int64(13)
memory usage: 1.3 MB
```

10.1.6 You must have noticed that there is a mismatch in the column sequence while compairing the dataframes

- Note - In order to fed into the machine learning model, you need to edit the sequence of "nulldata", similar to the "X_test" dataframe
- Display all the columns of the X_test dataframe
- Display all the columns of the Nulldata dataframe
- Store the Nulldata with new sequence into a new dataframe
- Note - The code is given below, no need to change

[141]:
```python
X_test.columns
```

[141]:
```
Index(['cust_number', 'buisness_year', 'doc_id', 'converted_usd', 'avgdelay',
       'business_code_enc', 'name_customer_enc', 'cust_payment_terms_enc',
       'day_of_postingdate', 'month_of_postingdate', 'year_of_postingdate',
       'day_of_baselinecreatedate', 'month_of_baselinecreatedate',
       'year_of_baselinecreatedate', 'day_of_dueindate', 'month_of_dueindate',
       'year_of_dueindate'],
      dtype='object')
```

[142]:
```python
nulldata.columns
```

[142]:
```
Index(['cust_number', 'buisness_year', 'doc_id', 'converted_usd',
       'business_code_enc', 'month_of_cleardate', 'day_of_postingdate',
       'month_of_postingdate', 'year_of_postingdate', 'day_of_due',
       'month_of_due', 'year_of_due', 'day_of_createdate',
       'month_of_createdate', 'year_of_createdate', 'cust_payment_terms_enc',
       'name_customer_enc'],
      dtype='object')
```

[143]:
```python
nulldata2 = nulldata[
    [
        "cust_number",
        "buisness_year",
        "doc_id",
        "converted_usd",
        "business_code_enc",
        "name_customer_enc",
        "cust_payment_terms_enc",
        "day_of_postingdate",
```

```
        "month_of_postingdate",
        "year_of_postingdate",
        "day_of_createdate",
        "month_of_createdate",
        "year_of_createdate",
        "day_of_due",
        "month_of_due",
        "year_of_due",
    ]
]
```

10.1.7 Display the Final Dataset

[144]:
```
nulldata
```

[144]:
```
       cust_number  buisness_year        doc_id  converted_usd  \
0        200769623         2020.0  1.930438e+09      54273.280
1        200980828         2019.0  1.929646e+09      79656.600
2        200792734         2019.0  1.929874e+09       2253.860
4        200769623         2019.0  1.930148e+09      33133.290
5        140106181         2019.0  2.960581e+09      15558.088
…                …              …             …              …
49994    200762301         2019.0  1.929601e+09      84780.400
49996    200769623         2019.0  1.929744e+09       6766.540
49997    200772595         2020.0  1.930537e+09       6120.860
49998    200726979         2019.0  1.930199e+09         63.480
49999    200020431         2019.0  1.928576e+09       1790.300

       business_code_enc  month_of_cleardate  day_of_postingdate  \
0                     68                   2                  26
1                     68                   8                  22
2                     68                  12                  14
4                     68                  11                  13
5                     68                  12                  20
…                      …                   …                   …
49994                 68                   7                  10
49996                 68                   9                  15
49997                 68                   3                  19
49998                 68                  12                  27
49999                 68                   1                   5

       month_of_postingdate  year_of_postingdate  day_of_due  month_of_due  \
0                         1                 2020          10             2
1                         7                 2019          11             8
2                         9                 2019          29             9
4                        11                 2019          28            11
5                         9                 2019           4            10
```

```
…                         …                  …          …            …
49994                     7               2019         25            7
49996                     8               2019         30            8
49997                     2               2020          5            3
49998                    11               2019         12           12
49999                     1               2019         24            1

       year_of_due  day_of_createdate  month_of_createdate  \
0             2020                 26                    1
1             2019                 22                    7
2             2019                 14                    9
4             2019                 13                   11
5             2019                 24                    9
…              …                 …                    …
49994         2019                 10                    7
49996         2019                 15                    8
49997         2020                 19                    2
49998         2019                 27                   11
49999         2019                  1                    1

       year_of_createdate  cust_payment_terms_enc  name_customer_enc
0                    2020                      37               3058
1                    2019                      31                298
2                    2019                      22               1909
4                    2019                      37               3060
5                    2019                       6               2869
…                     …                      …                  …
49994                2019                      27                457
49996                2019                      37               3057
49997                2020                      22               2452
49998                2019                      22                334
49999                2019                      41                729

[39158 rows x 17 columns]
```

10.1.8 Now you can pass this dataset into you final model and store it into "final_result"

[145]:
```
final_result = regressor.predict(nulldata)
```

```
/opt/conda/lib/python3.9/site-packages/xgboost/data.py:262: FutureWarning:
pandas.Int64Index is deprecated and will be removed from pandas in a future
version. Use pandas.Index with the appropriate dtype instead.
  elif isinstance(data.columns, (pd.Int64Index, pd.RangeIndex)):
```

10.1.9 you need to make the final_result as dataframe, with a column name "avg_delay"

- Note - No need to change the code

[146]:
```
final_result = pd.Series(final_result, name="avg_delay")
```

10.1.10 Display the "avg_delay" column

[147]:
```
final_result
```

[147]:
```
0        4895.571777
1         313.243774
2        -287.062469
3         202.362686
4        1075.488159
            …
39153      51.104931
39154     179.338745
39155    3624.087158
39156    1705.571411
39157    2588.733887
Name: avg_delay, Length: 39158, dtype: float32
```

10.1.11 Now you need to merge this final_result dataframe with the BACKUP of "nulldata" Dataframe which we have created in earlier steps

[148]:
```
nulldata_copy.reset_index(drop=True, inplace=True)
Final = nulldata_copy.merge(final_result, on=nulldata.index)
```

10.1.12 Display the "Final" dataframe

[149]:
```
Final
```

[149]:
```
       key_0 business_code cust_number       name_customer  clear_date  \
0          0          U001  0200769623        WAL-MAR corp  2020-02-11
1          1          U001  0200980828              BEN E   2019-08-08
2          2          U001  0200792734         MDV/ trust  2019-12-30
3          4          U001  0200769623  WAL-MAR foundation  2019-11-25
4          5          CA02  0140106181    THE  corporation  2019-12-04
…          …             …          …                  …           …
39153  49994          U001  0200762301        C&S WH trust  2019-07-25
39154  49996          U001  0200769623          WAL-MAR co  2019-09-03
39155  49997          U001  0200772595    SAFEW associates  2020-03-05
39156  49998          U001  0200726979           BJ'S  llc  2019-12-12
39157  49999          U001  0200020431            DEC corp  2019-01-15
```

```
       buisness_year        doc_id posting_date due_in_date  \
0             2020.0  1.930438e+09   2020-01-26  2020-02-10
1             2019.0  1.929646e+09   2019-07-22  2019-08-11
2             2019.0  1.929874e+09   2019-09-14  2019-09-29
3             2019.0  1.930148e+09   2019-11-13  2019-11-28
4             2019.0  2.960581e+09   2019-09-20  2019-10-04
…                …             …          …          …
39153         2019.0  1.929601e+09   2019-07-10  2019-07-25
39154         2019.0  1.929744e+09   2019-08-15  2019-08-30
39155         2020.0  1.930537e+09   2020-02-19  2020-03-05
39156         2019.0  1.930199e+09   2019-11-27  2019-12-12
39157         2019.0  1.928576e+09   2019-01-05  2019-01-24

      baseline_create_date cust_payment_terms  converted_usd    avg_delay
0               2020-01-26               NAH4      54273.280  4895.571777
1               2019-07-22               NAD1      79656.600   313.243774
2               2019-09-14               NAA8       2253.860  -287.062469
3               2019-11-13               NAH4      33133.290   202.362686
4               2019-09-24               CA10      15558.088  1075.488159
…                    …                 …             …           …
39153           2019-07-10               NAC6      84780.400    51.104931
39154           2019-08-15               NAH4       6766.540   179.338745
39155           2020-02-19               NAA8       6120.860  3624.087158
39156           2019-11-27               NAA8         63.480  1705.571411
39157           2019-01-01               NAM4       1790.300  2588.733887

[39158 rows x 13 columns]
```

10.1.13 Check for the Number of Rows and Columns in your "Final" dataframe

[150]:
```
Final.shape
```

[150]: (39158, 13)

10.1.14 Now, you need to do convert the below fields back into date and time format

- Convert "due_in_date" into datetime format
- Convert "avg_delay" into datetime format
- Create a new column "clear_date" and store the sum of "due_in_date" and "avg_delay"
- display the new "clear_date" column
- Note - Code is given below, no need to change

[151]:
```
Final["clear_date"] = pd.to_datetime(Final["due_in_date"]) + pd.to_timedelta(
    Final["avg_delay"], unit="s"
)
```

10.1.15 Display the "clear_date" column

```
[152]: Final["clear_date"]
```

```
[152]: 0        2020-02-10 01:21:35.571777344
       1        2019-08-11 00:05:13.243774414
       2        2019-09-28 23:55:12.937530518
       3        2019-11-28 00:03:22.362686157
       4        2019-10-04 00:17:55.488159180
                              …
       39153    2019-07-25 00:00:51.104930878
       39154    2019-08-30 00:02:59.338745117
       39155    2020-03-05 01:00:24.087158203
       39156    2019-12-12 00:28:25.571411133
       39157    2019-01-24 00:43:08.733886719
       Name: clear_date, Length: 39158, dtype: datetime64[ns]
```

10.1.16 Convert the average delay into number of days format

- Note - Formula = avg_delay//(24 * 3600)
- Note - full code is given for this, no need to change

```
[153]: Final["avg_delay"] = Final.apply(
           lambda row: row.avg_delay // (24 * 3600), axis=1
       )
```

10.1.17 Display the "avg_delay" column

```
[154]: Final["avg_delay"]
```

```
[154]: 0         0.0
       1         0.0
       2        -1.0
       3         0.0
       4         0.0
                …
       39153     0.0
       39154     0.0
       39155     0.0
       39156     0.0
       39157     0.0
       Name: avg_delay, Length: 39158, dtype: float64
```

10.1.18 Now you need to convert average delay column into bucket

- Need to perform binning
- create a list of bins i.e. bins= [0,15,30,45,60,100]

- create a list of labels i.e. labels = [‘0-15’,‘16-30’,‘31-45’,‘46-60’,‘Greatar than 60’]
- perform binning by using cut() function from “Final” dataframe
- Please fill up the first two rows of the code

[155]:
```
bins = [0, 15, 30, 45, 60, 100]
labels = ["0-15", "16-30", "31-45", "46-60", "Greatar than 60"]
Final["Aging Bucket"] = pd.cut(
    Final["avg_delay"], bins=bins, labels=labels, right=False
)
```

10.1.19 Now you need to drop “key_0” and “avg_delay” columns from the “Final” Dataframe

[156]:
```
Final.drop(columns=["key_0", "avg_delay"], axis=1, inplace=True)
```

10.1.20 Display the count of each categoty of new “Aging Bucket” column

[157]:
```
Final["Aging Bucket"].value_counts()
```

[157]:
```
0-15               29548
16-30                  0
31-45                  0
46-60                  0
Greatar than 60        0
Name: Aging Bucket, dtype: int64
```

10.1.21 Display your final dataset with aging buckets

[158]:
```
Final["Aging Bucket"]
```

[158]:
```
0        0-15
1        0-15
2         NaN
3        0-15
4        0-15
         …
39153    0-15
39154    0-15
39155    0-15
39156    0-15
39157    0-15
Name: Aging Bucket, Length: 39158, dtype: category
Categories (5, object): ['0-15' < '16-30' < '31-45' < '46-60' < 'Greatar than
60']
```

10.1.22 Store this dataframe into the .csv format

[159]:
```
Final.to_csv("Predicted Dates.csv")
```

11 END OF THE PROJECT