Payment Date Prediction

Importing related Libraries

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold

import warnings
warnings.filterwarnings('ignore')
```

Store the dataset into the Dataframe

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

Check the shape of the dataframe

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

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

Check the Detail information of the dataframe

In [4]:
```
data.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
```

Display All the column names

In [5]:
```
data.columns
```

Out[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')
```

Describe the entire dataset

In [6]:
```
data.describe()
```

Out[6]:

	buisness_year	doc_id	document_create_date	document_create_date.1	due_in_date	posting_id	ar
count	50000.000000	5.000000e+04	5.000000e+04	5.000000e+04	5.000000e+04	50000.0	
mean	2019.305700	2.012238e+09	2.019351e+07	2.019354e+07	2.019368e+07	1.0	
std	0.460708	2.885235e+08	4.496041e+03	4.482134e+03	4.470614e+03	0.0	
min	2019.000000	1.928502e+09	2.018123e+07	2.018123e+07	2.018122e+07	1.0	
25%	2019.000000	1.929342e+09	2.019050e+07	2.019051e+07	2.019052e+07	1.0	
50%	2019.000000	1.929964e+09	2.019091e+07	2.019091e+07	2.019093e+07	1.0	
75%	2020.000000	1.930619e+09	2.020013e+07	2.020013e+07	2.020022e+07	1.0	
max	2020.000000	9.500000e+09	2.020052e+07	2.020052e+07	2.020071e+07	1.0	

Data Cleaning

- Show top 5 records from the dataset

In [7]:
```
data.head()
```

Out[7]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	docume
0	U001	0200769623	WAL-MAR corp	2020-02-11 00:00:00	2020.0	1.930438e+09	2020-01-26	
1	U001	0200980828	BEN E	2019-08-08 00:00:00	2019.0	1.929646e+09	2019-07-22	
2	U001	0200792734	MDV/ trust	2019-12-30 00:00:00	2019.0	1.929874e+09	2019-09-14	
3	CA02	0140105686	SYSC llc	NaN	2020.0	2.960623e+09	2020-03-30	
4	U001	0200769623	WAL-MAR foundation	2019-11-25 00:00:00	2019.0	1.930148e+09	2019-11-13	

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 = 0.12% null

In [8]:
```
percent = (data.isnull().sum() / len(data)) * 100
print(round(percent,3))
```

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

Display Invoice_id and Doc_Id

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

In [9]:
```
data.invoice_id
```

Out[9]:
```
0        1.930438e+09
1        1.929646e+09
2        1.929874e+09
3        2.960623e+09
4        1.930148e+09
            ...
49995    1.930797e+09
49996    1.929744e+09
49997    1.930537e+09
49998    1.930199e+09
49999    1.928576e+09
Name: invoice_id, Length: 50000, dtype: float64
```

In [10]:
```
data.doc_id
```

Out[10]:
```
0        1.930438e+09
1        1.929646e+09
2        1.929874e+09
3        2.960623e+09
4        1.930148e+09
            ...
49995    1.930797e+09
49996    1.929744e+09
49997    1.930537e+09
49998    1.930199e+09
49999    1.928576e+09
Name: doc_id, Length: 50000, dtype: float64
```

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

In [11]:

```
compare = np.where((data['baseline_create_date'] == data['document_create_date']) & (data[
for i in compare:
  print(i)
```

```
20200126.0
Same values
Same values
20200331.0
Same values
20190924.0
20191101.0
20200319.0
20190607.0
20190220.0
20200311.0
Same values
Same values
Same values
Same values
20200416.0
Same values
Same values
20191115.0
Same values
20200111.0
20190821.0
20200321.0
Same values
20200124.0
Same values
Same values
Same values
Same values
Same values
Same values
20190922.0
20190910.0
20190615.0
20190705.0
20200407.0
20190123.0
20200423.0
20190617.0
Same values
Same values
Same values
20200122.0
Same values
20190414.0
20200316.0
20190416.0
Same values
20200217.0
20190118.0
20190501.0
Same values
20200501.0
Same values
20200115.0
20200218.0
20190828.0
```

```
20190901.0
20190701.0
Same values
20200503.0
Same values
Same values
Same values
20190522.0
20190502.0
20190529.0
20190802.0
20190910.0
20200218.0
Same values
20191010.0
Same values
Same values
20190710.0
20200421.0
20190815.0
20200219.0
20191127.0
20190101.0
```

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

In [12]:
```
constant_cols =  [x for x in data.columns if data[x].nunique()==1]
print(constant_cols)
```

```
['posting_id']
```

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

In [13]:
```
data.isOpen.value_counts()
```

Out[13]:
```
0    40000
1    10000
Name: isOpen, dtype: int64
```

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

In [14]:
```
drop_cols = ['area_business','posting_id','invoice_id','document_create_date','isOpen','do
data.drop(drop_cols,axis=1,inplace=True)
data
```

Out[14]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due
0	U001	0200769623	WAL-MAR corp	2020-02-11 00:00:00	2020.0	1.930438e+09	2020-01-26	20

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due
1	U001	0200980828	BEN E	2019-08-08 00:00:00	2019.0	1.929646e+09	2019-07-22	20
2	U001	0200792734	MDV/ trust	2019-12-30 00:00:00	2019.0	1.929874e+09	2019-09-14	20
3	CA02	0140105686	SYSC llc	NaN	2020.0	2.960623e+09	2020-03-30	20
4	U001	0200769623	WAL-MAR foundation	2019-11-25 00:00:00	2019.0	1.930148e+09	2019-11-13	20
...	...	...	...	...	...	...	...	
49995	U001	0200561861	CO corporation	NaN	2020.0	1.930797e+09	2020-04-21	20
49996	U001	0200769623	WAL-MAR co	2019-09-03 00:00:00	2019.0	1.929744e+09	2019-08-15	20
49997	U001	0200772595	SAFEW associates	2020-03-05 00:00:00	2020.0	1.930537e+09	2020-02-19	20
49998	U001	0200726979	BJ'S llc	2019-12-12 00:00:00	2019.0	1.930199e+09	2019-11-27	20
49999	U001	0200020431	DEC corp	2019-01-15 00:00:00	2019.0	1.928576e+09	2019-01-05	20

50000 rows × 12 columns

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

In [15]:
```
data.columns
```

Out[15]:
```
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')
```

Show all the Dublicate rows from the dataframe

In [16]:
```
duplicate_rows = data.duplicated()
data[duplicate_rows]
```

Out[16]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due
1041	U001	0200769623	WAL-MAR in	2019-03-12 00:00:00	2019.0	1.928870e+09	2019-02-28	20
2400	U001	0200769623	WAL-MAR trust	2019-08-28 00:00:00	2019.0	1.929758e+09	2019-08-18	20
2584	U001	0200769623	WAL-MAR corporation	2019-12-16 00:00:00	2019.0	1.930217e+09	2019-12-04	20

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due
3755	U001	0200769623	WAL-MAR	2019-11-22 00:00:00	2019.0	1.930137e+09	2019-11-12	20
3873	CA02	0140104409	LOB associates	NaN	2020.0	2.960629e+09	2020-04-14	20
...	...	...	...	...	...	...	...	
49928	U001	0200915438	GROC trust	2019-08-15 00:00:00	2019.0	1.929646e+09	2019-07-25	20
49963	U001	0200759878	SA us	2019-01-29 00:00:00	2019.0	1.928614e+09	2019-01-13	20
49986	U001	0200772670	ASSOCIAT foundation	2019-06-12 00:00:00	2019.0	1.929403e+09	2019-05-29	20
49990	U001	0200765011	MAINES llc	2019-06-06 00:00:00	2019.0	1.929365e+09	2019-05-22	20
49991	U001	0200704045	RA trust	2019-10-25 00:00:00	2019.0	1.930001e+09	2019-10-10	20

1161 rows × 12 columns

Display the Number of Dublicate Rows

In [17]:
```
data.duplicated().sum()
```

Out[17]:
```
1161
```

Drop all the Dublicate Rows

In [18]:
```
data.drop_duplicates(inplace=True)
data
```

Out[18]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due
0	U001	0200769623	WAL-MAR corp	2020-02-11 00:00:00	2020.0	1.930438e+09	2020-01-26	20
1	U001	0200980828	BEN E	2019-08-08 00:00:00	2019.0	1.929646e+09	2019-07-22	20
2	U001	0200792734	MDV/ trust	2019-12-30 00:00:00	2019.0	1.929874e+09	2019-09-14	20
3	CA02	0140105686	SYSC llc	NaN	2020.0	2.960623e+09	2020-03-30	20
4	U001	0200769623	WAL-MAR foundation	2019-11-25 00:00:00	2019.0	1.930148e+09	2019-11-13	20
...	...	...	...	...	...	...	...	
49995	U001	0200561861	CO corporation	NaN	2020.0	1.930797e+09	2020-04-21	20

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due
49996	U001	0200769623	WAL-MAR co	2019-09-03 00:00:00	2019.0	1.929744e+09	2019-08-15	20
49997	U001	0200772595	SAFEW associates	2020-03-05 00:00:00	2020.0	1.930537e+09	2020-02-19	20
49998	U001	0200726979	BJ'S llc	2019-12-12 00:00:00	2019.0	1.930199e+09	2019-11-27	20
49999	U001	0200020431	DEC corp	2019-01-15 00:00:00	2019.0	1.928576e+09	2019-01-05	20

48839 rows × 12 columns

Now check for all dublicate rows now

- Note - It must be 0 by now

In [19]:
```
data.duplicated().sum()
```

Out[19]:
```
0
```

Check for the number of Rows and Columns in your dataset

In [20]:
```
data.shape
```

Out[20]:
```
(48839, 12)
```

Find out the total count of null values in each columns

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

Out[21]:
```
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
```

Data type Conversion

Please check the data type of each column of the dataframe

In [22]:
```
data.dtypes
```

Out[22]:
```
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
```

Check the datatype format of below columns

- clear_date
- posting_date
- due_in_date
- baseline_create_date

In [23]:
```
dtypes = []
dtypes.append(data['clear_date'].dtypes)
dtypes.append(data['posting_date'].dtypes)
dtypes.append(data['due_in_date'].dtypes)
dtypes.append(data['baseline_create_date'].dtypes)
names = ['clear_date','posting_date','due_in_date','baseline_create_date']
Data_types = pd.DataFrame({'Column_name': names,'Corresponding data type ': dtypes})
Data_types
```

Out[23]:

	Column_name	Corresponding data type
0	clear_date	object
1	posting_date	object
2	due_in_date	float64
3	baseline_create_date	float64

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**

In [24]:
```
data.clear_date=pd.to_datetime(data.clear_date)

data.posting_date=pd.to_datetime(data.posting_date)

data.due_in_date=pd.to_datetime(data.due_in_date,format='%Y%m%d')

data.baseline_create_date=pd.to_datetime(data.baseline_create_date,format='%Y%m%d')
```

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

columns

In [25]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48839 entries, 0 to 49999
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype         
---  ------                --------------  -----         
 0   business_code         48839 non-null  object        
 1   cust_number           48839 non-null  object        
 2   name_customer         48839 non-null  object        
 3   clear_date            39158 non-null  datetime64[ns]
 4   buisness_year         48839 non-null  float64       
 5   doc_id                48839 non-null  float64       
 6   posting_date          48839 non-null  datetime64[ns]
 7   due_in_date           48839 non-null  datetime64[ns]
 8   invoice_currency      48839 non-null  object        
 9   total_open_amount     48839 non-null  float64       
 10  baseline_create_date  48839 non-null  datetime64[ns]
 11  cust_payment_terms    48839 non-null  object        
dtypes: datetime64[ns](4), float64(3), object(5)
memory usage: 4.8+ MB
```

In [26]:
```
data
```

Out[26]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due
0	U001	0200769623	WAL-MAR corp	2020-02-11	2020.0	1.930438e+09	2020-01-26	20
1	U001	0200980828	BEN E	2019-08-08	2019.0	1.929646e+09	2019-07-22	20
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	20
3	CA02	0140105686	SYSC llc	NaT	2020.0	2.960623e+09	2020-03-30	20
4	U001	0200769623	WAL-MAR foundation	2019-11-25	2019.0	1.930148e+09	2019-11-13	20
...	...	...	...	...	...	...	...	
49995	U001	0200561861	CO corporation	NaT	2020.0	1.930797e+09	2020-04-21	20
49996	U001	0200769623	WAL-MAR co	2019-09-03	2019.0	1.929744e+09	2019-08-15	20
49997	U001	0200772595	SAFEW associates	2020-03-05	2020.0	1.930537e+09	2020-02-19	20
49998	U001	0200726979	BJ'S llc	2019-12-12	2019.0	1.930199e+09	2019-11-27	20
49999	U001	0200020431	DEC corp	2019-01-15	2019.0	1.928576e+09	2019-01-05	20

48839 rows × 12 columns

the invoice_currency column contains two different categories, USD and CAD

- Please do a count of each currency

In [27]:
```
data.invoice_currency.value_counts()
```

Out[27]:
```
USD    45011
CAD     3828
Name: invoice_currency, dtype: int64
```

display the "total_open_amount" column value

In [28]:
```
data.total_open_amount
```

Out[28]:
```
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
```

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

In [29]:
```
converted_usd = np.where((data['invoice_currency'] == 'USD'), data['total_open_amount']*0.
data['converted_usd'] = converted_usd
```

Display the new "converted_usd" column values

In [30]:
```
data.converted_usd
```

Out[30]:
```
0        37991.296
1        55759.620
2         1577.702
3         3299.700
4        23193.303
           ...
49995     2231.502
49996     4736.578
49997     4284.602
49998       44.436
49999     1253.210
Name: converted_usd, Length: 48839, dtype: float64
```

Display year wise total number of record

- Note - use "buisness_year" column for this

In [31]:
```
data.buisness_year.value_counts()
```

Out[31]:
```
2019.0    33975
2020.0    14864
Name: buisness_year, dtype: int64
```

Write the code to delete the following columns

- 'invoice_currency'
- 'total_open_amount',

In [32]:
```
data.drop('invoice_currency',axis=1,inplace=True)
```

In [33]:
```
data.drop('total_open_amount',axis=1,inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48839 entries, 0 to 49999
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype         
---  ------                --------------  -----         
 0   business_code         48839 non-null  object        
 1   cust_number           48839 non-null  object        
 2   name_customer         48839 non-null  object        
 3   clear_date            39158 non-null  datetime64[ns]
 4   buisness_year         48839 non-null  float64       
 5   doc_id                48839 non-null  float64       
 6   posting_date          48839 non-null  datetime64[ns]
 7   due_in_date           48839 non-null  datetime64[ns]
 8   baseline_create_date  48839 non-null  datetime64[ns]
 9   cust_payment_terms    48839 non-null  object        
 10  converted_usd         48839 non-null  float64       
dtypes: datetime64[ns](4), float64(3), object(4)
memory usage: 4.5+ MB
```

Write a code to check the number of columns in dataframe

In [34]:
```
data.shape[1]
```

Out[34]:
```
11
```

Splitting the Dataset

Look for all columns containing null value

- Note - Output expected is only one column

In [35]:
```
data[data.columns[data.isnull().any()]]
```

Out[35]:

	clear_date
0	2020-02-11
1	2019-08-08
2	2019-12-30
3	NaT
4	2019-11-25
...	...
49995	NaT
49996	2019-09-03

	clear_date
49998	2019-12-12
49999	2019-01-15

48839 rows × 1 columns

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

In [36]:
```
data.clear_date.isnull().sum()
```

Out[36]:
```
9681
```

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)

In [37]:
```
nulldata = data[data.clear_date.isnull()]
maindata = data[data.clear_date.notnull()]
```

Check the number of Rows and Columns for both the dataframes

In [38]:
```
nulldata.shape
```

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

In [39]:
```
maindata.shape
```

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

Display the 5 records from maindata and nulldata dataframes

In [40]:
```
nulldata.head()
```

Out[40]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in
3	CA02	0140105686	SYSC llc	NaT	2020.0	2.960623e+09	2020-03-30	2020-
7	U001	0200744019	TARG us	NaT	2020.0	1.930659e+09	2020-03-19	2020-
10	U001	0200418007	AM	NaT	2020.0	1.930611e+09	2020-03-11	2020-
14	U001	0200739534	OK systems	NaT	2020.0	1.930788e+09	2020-04-15	2020-
15	U001	0200353024	DECA corporation	NaT	2020.0	1.930817e+09	2020-04-23	2020-

In [41]:
```
maindata.head()
```

Out[41]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_
0	U001	0200769623	WAL-MAR corp	2020-02-11	2020.0	1.930438e+09	2020-01-26	2020-0
1	U001	0200980828	BEN E	2019-08-08	2019.0	1.929646e+09	2019-07-22	2019-0
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	2019-0
4	U001	0200769623	WAL-MAR foundation	2019-11-25	2019.0	1.930148e+09	2019-11-13	2019-1
5	CA02	0140106181	THE corporation	2019-12-04	2019.0	2.960581e+09	2019-09-20	2019-1

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

In [42]:
```
delay=maindata.clear_date-maindata.due_in_date
delay
```

Out[42]:
```
0        1 days
1       -3 days
2       92 days
4       -3 days
5       61 days
          ...
49994    0 days
49996    4 days
49997    0 days
49998    0 days
49999   -9 days
Length: 39158, dtype: timedelta64[ns]
```

In [43]:
```
maindata['delay']=delay
maindata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 39158 entries, 0 to 49999
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype         
---  ------                --------------  -----         
 0   business_code         39158 non-null  object        
 1   cust_number           39158 non-null  object        
 2   name_customer         39158 non-null  object        
 3   clear_date            39158 non-null  datetime64[ns]
 4   buisness_year         39158 non-null  float64       
 5   doc_id                39158 non-null  float64       
 6   posting_date          39158 non-null  datetime64[ns]
 7   due_in_date           39158 non-null  datetime64[ns]
 8   baseline_create_date  39158 non-null  datetime64[ns]
 9   cust_payment_terms    39158 non-null  object        
 10  converted_usd         39158 non-null  float64       
 11  delay                 39158 non-null  timedelta64[ns]
```

```
dtypes: datetime64[ns](4), float64(3), object(4), timedelta64[ns](1)
memory usage: 3.9+ MB
```

In [44]:
```
maindata.head()
```

Out[44]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_
0	U001	0200769623	WAL-MAR corp	2020-02-11	2020.0	1.930438e+09	2020-01-26	2020-0
1	U001	0200980828	BEN E	2019-08-08	2019.0	1.929646e+09	2019-07-22	2019-0
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	2019-0
4	U001	0200769623	WAL-MAR foundation	2019-11-25	2019.0	1.930148e+09	2019-11-13	2019-1
5	CA02	0140106181	THE corporation	2019-12-04	2019.0	2.960581e+09	2019-09-20	2019-1

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

In [45]:
```
avg_delay = maindata.groupby('name_customer')['delay'].mean(numeric_only=False)
avg_delay
```

Out[45]:
```
name_customer
11078 us                 17 days 00:00:00
17135 associates       -10 days +00:00:00
17135 llc               -3 days +00:00:00
236008 associates       -3 days +00:00:00
99 CE                     2 days 00:00:00
                              ...
YEN BROS corp             0 days 00:00:00
YEN BROS corporation    -1 days +12:00:00
YEN BROS llc            -2 days +00:00:00
ZARCO co                -1 days +00:00:00
ZIYAD  us                 6 days 00:00:00
Name: delay, Length: 3889, dtype: timedelta64[ns]
```

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

In [46]:
```
maindata['avg_delay'] = maindata['name_customer'].map(maindata.groupby('name_customer')['d
maindata.head()
```

Out[46]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_
0	U001	0200769623	WAL-MAR corp	2020-02-11	2020.0	1.930438e+09	2020-01-26	2020-0
1	U001	0200980828	BEN E	2019-08-08	2019.0	1.929646e+09	2019-07-22	2019-0

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_(
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	2019-0
4	U001	0200769623	WAL-MAR foundation	2019-11-25	2019.0	1.930148e+09	2019-11-13	2019-1
5	CA02	0140106181	THE corporation	2019-12-04	2019.0	2.960581e+09	2019-09-20	2019-1(

In [47]:
```
maindata
```

Out[47]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due
0	U001	0200769623	WAL-MAR corp	2020-02-11	2020.0	1.930438e+09	2020-01-26	20
1	U001	0200980828	BEN E	2019-08-08	2019.0	1.929646e+09	2019-07-22	20
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	20
4	U001	0200769623	WAL-MAR foundation	2019-11-25	2019.0	1.930148e+09	2019-11-13	20
5	CA02	0140106181	THE corporation	2019-12-04	2019.0	2.960581e+09	2019-09-20	20
...	...	...	...	...	...	...	...	
49994	U001	0200762301	C&S WH trust	2019-07-25	2019.0	1.929601e+09	2019-07-10	20
49996	U001	0200769623	WAL-MAR co	2019-09-03	2019.0	1.929744e+09	2019-08-15	20
49997	U001	0200772595	SAFEW associates	2020-03-05	2020.0	1.930537e+09	2020-02-19	20
49998	U001	0200726979	BJ'S llc	2019-12-12	2019.0	1.930199e+09	2019-11-27	20
49999	U001	0200020431	DEC corp	2019-01-15	2019.0	1.928576e+09	2019-01-05	20

39158 rows × 13 columns

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

In [48]:
```
maindata['avg_delay']=(maindata['avg_delay'].dt.total_seconds().astype(int))
```

Display the maindata dataframe

In [49]:
```
maindata
```

Out[49]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due
0	U001	0200769623	WAL-MAR corp	2020-02-11	2020.0	1.930438e+09	2020-01-26	20
1	U001	0200980828	BEN E	2019-08-08	2019.0	1.929646e+09	2019-07-22	20
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	20
4	U001	0200769623	WAL-MAR foundation	2019-11-25	2019.0	1.930148e+09	2019-11-13	20
5	CA02	0140106181	THE corporation	2019-12-04	2019.0	2.960581e+09	2019-09-20	20
...	...	...	...	...	...	...	...	
49994	U001	0200762301	C&S WH trust	2019-07-25	2019.0	1.929601e+09	2019-07-10	20
49996	U001	0200769623	WAL-MAR co	2019-09-03	2019.0	1.929744e+09	2019-08-15	20
49997	U001	0200772595	SAFEW associates	2020-03-05	2020.0	1.930537e+09	2020-02-19	20
49998	U001	0200726979	BJ'S llc	2019-12-12	2019.0	1.930199e+09	2019-11-27	20
49999	U001	0200020431	DEC corp	2019-01-15	2019.0	1.928576e+09	2019-01-05	20

39158 rows × 13 columns

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

In [50]:
```
maindata.drop(['delay','clear_date'],axis=1,inplace=True)
```

In [51]:
```
maindata
```

Out[51]:

	business_code	cust_number	name_customer	buisness_year	doc_id	posting_date	due_in_date	ba
0	U001	0200769623	WAL-MAR corp	2020.0	1.930438e+09	2020-01-26	2020-02-10	
1	U001	0200980828	BEN E	2019.0	1.929646e+09	2019-07-22	2019-08-11	
2	U001	0200792734	MDV/ trust	2019.0	1.929874e+09	2019-09-14	2019-09-29	
4	U001	0200769623	WAL-MAR foundation	2019.0	1.930148e+09	2019-11-13	2019-11-28	
5	CA02	0140106181	THE corporation	2019.0	2.960581e+09	2019-09-20	2019-10-04	
...	...	...	...	...	...	...	...	
49994	U001	0200762301	C&S WH trust	2019.0	1.929601e+09	2019-07-10	2019-07-25	
49996	U001	0200769623	WAL-MAR co	2019.0	1.929744e+09	2019-08-15	2019-08-30	
49997	U001	0200772595	SAFEW associates	2020.0	1.930537e+09	2020-02-19	2020-03-05	
49998	U001	0200726979	BJ'S llc	2019.0	1.930199e+09	2019-11-27	2019-12-12	
49999	U001	0200020431	DEC corp	2019.0	1.928576e+09	2019-01-05	2019-01-24	

39158 rows × 11 columns

Splitting of Train and the Test Data

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

In [52]:
```
X=maindata.drop(['avg_delay'],axis=1)
```

In [53]:
```
y=maindata['avg_delay']
```

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

In [54]:
```
X_train, X_loc_test, y_train, y_loc_test = train_test_split(X,y,test_size=0.4,random_state
```

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

In [55]:
```
X_train.shape, X_loc_test.shape, y_train.shape, y_loc_test.shape
```

Out[55]:
```
((23494, 10), (15664, 10), (23494,), (15664,))
```

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

In [56]:
```
X_val, X_test, y_val, y_test = train_test_split(X_loc_test,y_loc_test,test_size=0.5,random
```

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

In [57]:
```
X_val.shape, X_test.shape, y_val.shape, y_test.shape
```

Out[57]:
```
((7832, 10), (7832, 10), (7832,), (7832,))
```

Exploratory Data Analysis (EDA)

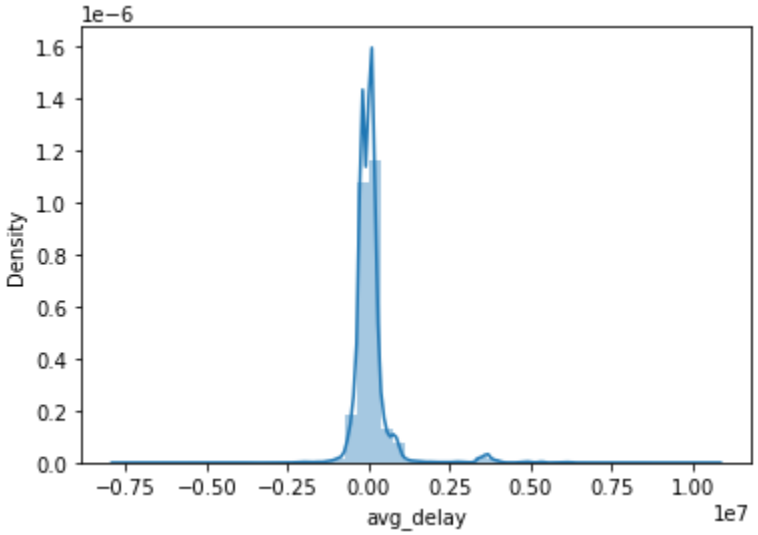
Distribution Plot of the target variable (use the dataframe which contains the target field)

e expected to make a distribution plot for the target variable

In [58]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [59]:
```
sns.distplot(y)
```

Out[59]:
```
<AxesSubplot:xlabel='avg_delay', ylabel='Density'>
```



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

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()

In [60]:
```
new_df = X_train.groupby(by=['name_customer'], as_index=False)['doc_id'].count()
new_df
```

Out[60]:

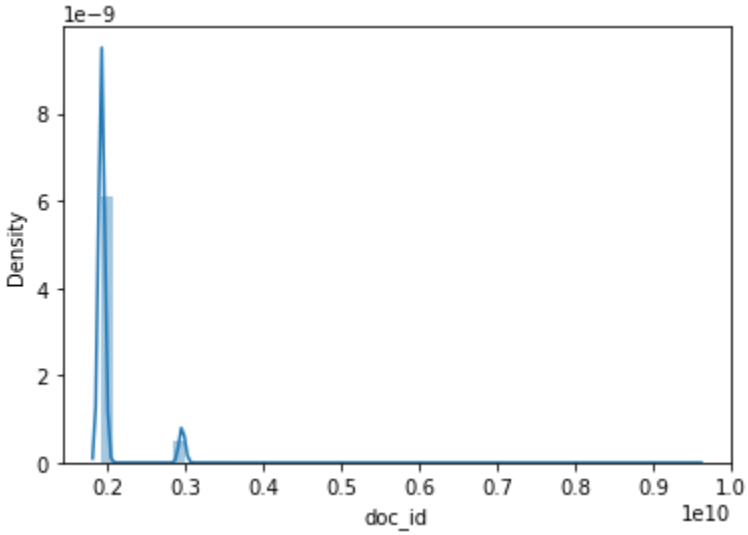
	name_customer	doc_id
0	11078 us	1
1	17135 associates	1
2	236008 associates	1
3	99 CE	2
4	99 CE associates	1
...	...	...
3083	YAEGER in	1
3084	YEN BROS	1
3085	YEN BROS corporation	1
3086	YEN BROS llc	1
3087	ZIYAD us	1

3088 rows × 2 columns

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

In [61]:
```
sns.distplot(X_train.doc_id)
```

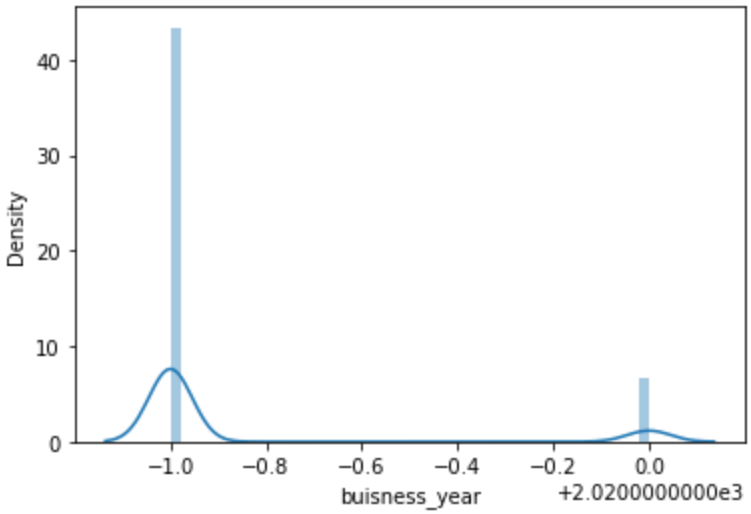
Out[61]:
```
<AxesSubplot:xlabel='doc_id', ylabel='Density'>
```



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

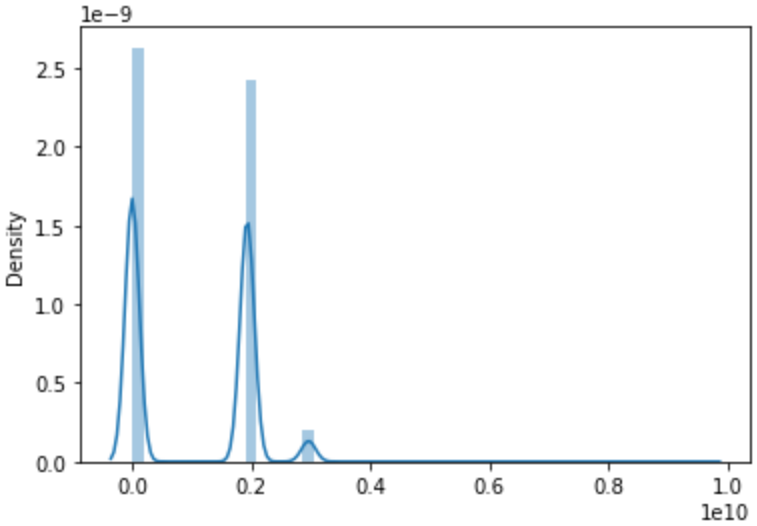
In [62]:
```
sns.distplot(X_train.buisness_year)
```

Out[62]:
```
<AxesSubplot:xlabel='buisness_year', ylabel='Density'>
```



In [63]:
```
sns.distplot([X_train.buisness_year, X_train.doc_id])
```

Out[63]:
```
<AxesSubplot:ylabel='Density'>
```



Feature Engineering

Display and describe the X_train dataframe

In [64]:
```
X_train
```

Out[64]:

	business_code	cust_number	name_customer	buisness_year	doc_id	posting_date	due_in_date	ba
0	U001	0200769623	WAL-MAR corp	2020.0	1.930438e+09	2020-01-26	2020-02-10	
1	U001	0200980828	BEN E	2019.0	1.929646e+09	2019-07-22	2019-08-11	
2	U001	0200792734	MDV/ trust	2019.0	1.929874e+09	2019-09-14	2019-09-29	
4	U001	0200769623	WAL-MAR foundation	2019.0	1.930148e+09	2019-11-13	2019-11-28	
5	CA02	0140106181	THE corporation	2019.0	2.960581e+09	2019-09-20	2019-10-04	
...	...	...	...	...	...	...	...	
29659	U001	0200772670	ASSOCIAT associates	2019.0	1.929725e+09	2019-08-08	2019-08-23	
29662	U001	0200794332	COST corporation	2020.0	1.930469e+09	2020-02-06	2020-02-21	
29663	U001	0200769623	WAL-MAR associates	2019.0	1.929143e+09	2019-04-14	2019-04-29	
29664	U001	0200696090	UNITE	2019.0	1.928950e+09	2019-03-18	2019-04-02	
29665	U001	200794332	COST in	2019.0	1.929087e+09	2019-04-08	2019-04-23	

23494 rows × 10 columns

In [65]:
```
X_train.describe()
```

Out[65]:

	buisness_year	doc_id	converted_usd
count	23494.000000	2.349400e+04	23494.000000
mean	2019.132842	2.012017e+09	23720.912705
std	0.339412	2.853757e+08	31796.580480
min	2019.000000	1.928502e+09	1.680000

	buisness_year	doc_id	converted_usd
25%	2019.000000	1.929181e+09	3449.332250
50%	2019.000000	1.929733e+09	12391.872500
75%	2019.000000	1.930209e+09	33171.463500
max	2020.000000	9.500000e+09	468015.352000

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

In [66]:
```
from sklearn.preprocessing import LabelEncoder
business_coder = LabelEncoder()
business_coder.fit(X_train['business_code'])
```

Out[66]:
```
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

In [67]:
```
X_train['business_code_enc'] = business_coder.fit_transform(X_train['business_code'])
```

In [68]:
```
X_val['business_code_enc'] = business_coder.transform(X_val['business_code'])
X_test['business_code_enc'] = business_coder.transform(X_test['business_code'])
```

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

In [69]:
```
X_train[['business_code','business_code_enc']]
```

Out[69]:

	business_code	business_code_enc
0	U001	1
1	U001	1
2	U001	1
4	U001	1
5	CA02	0
...	...	...
29659	U001	1
29662	U001	1
29663	U001	1
	U001	1

	business_code	business_code_enc
29665	U001	1

23494 rows × 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

In [70]:
```
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
```

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

In [71]:
```
X_train , X_val , X_test = custom(['business_code'])
```

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

In [72]:
```
X_train['cust_number'] = X_train['cust_number'].str.replace('CCCA',"1").str.replace('CCU',
X_test['cust_number'] = X_test['cust_number'].str.replace('CCCA',"1").str.replace('CCU',"2
X_val['cust_number'] = X_val['cust_number'].str.replace('CCCA',"1").str.replace('CCU',"2")
```

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.

In [73]:
```
#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)
```

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

In [74]:
```
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'])
```

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

In [75]:
```
X_train ,X_val, X_test = custom(['name_customer'])
```

Using Label Encoder for the "cust_payment_terms" column

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

In [76]:
```
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'])
```

In [77]:
```
X_train ,X_val, X_test = custom(['cust_payment_terms'])
```

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

- Note - You are expected yo use dtype

In [78]:
```
X_train.dtypes
```

Out[78]:
```
cust_number                    int32
buisness_year                float64
doc_id                       float64
posting_date          datetime64[ns]
due_in_date           datetime64[ns]
baseline_create_date  datetime64[ns]
converted_usd                float64
business_code_enc              int32
name_customer_enc              int32
```

```
cust_payment_terms_enc            int32
dtype: object
```

In [79]:
```
X_test.dtypes
```

Out[79]:
```
cust_number                       int32
buisness_year                   float64
doc_id                          float64
posting_date              datetime64[ns]
due_in_date               datetime64[ns]
baseline_create_date      datetime64[ns]
converted_usd                   float64
business_code_enc                 int32
name_customer_enc                 int32
cust_payment_terms_enc            int32
dtype: object
```

In [80]:
```
X_val.dtypes
```

Out[80]:
```
cust_number                       int32
buisness_year                   float64
doc_id                          float64
posting_date              datetime64[ns]
due_in_date               datetime64[ns]
baseline_create_date      datetime64[ns]
converted_usd                   float64
business_code_enc                 int32
name_customer_enc                 int32
cust_payment_terms_enc            int32
dtype: object
```

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

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

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

In [81]:
```
X_train['day_of_postingdate'] = X_train['posting_date'].dt.day
X_train['month_of_postingdate'] = X_train['posting_date'].dt.month
X_train['year_of_postingdate'] = X_train['posting_date'].dt.year

X_val['day_of_postingdate'] = X_val['posting_date'].dt.day
```

```
X_val['month_of_postingdate'] = X_val['posting_date'].dt.month
X_val['year_of_postingdate'] = X_val['posting_date'].dt.year


X_test['day_of_postingdate'] = X_test['posting_date'].dt.day
X_test['month_of_postingdate'] = X_test['posting_date'].dt.month
X_test['year_of_postingdate'] = X_test['posting_date'].dt.year
```

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

In [82]:
```
X_train ,X_val, X_test = custom(['posting_date'])
```

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

Extracting Day, Month, Year for 'baseline_create_date' column

In [83]:
```
X_train['day_of_createdate'] = X_train['baseline_create_date'].dt.day
X_train['month_of_createdate'] = X_train['baseline_create_date'].dt.month
X_train['year_of_createdate'] = X_train['baseline_create_date'].dt.year

X_val['day_of_createdate'] = X_val['baseline_create_date'].dt.day
X_val['month_of_createdate'] = X_val['baseline_create_date'].dt.month
X_val['year_of_createdate'] = X_val['baseline_create_date'].dt.year


X_test['day_of_createdate'] = X_test['baseline_create_date'].dt.day
X_test['month_of_createdate'] = X_test['baseline_create_date'].dt.month
X_test['year_of_createdate'] = X_test['baseline_create_date'].dt.year
```

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

In [84]:
```
X_train ,X_val, X_test = custom(['baseline_create_date'])
```

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

In [85]:
```
X_train['day_of_due'] = X_train['due_in_date'].dt.day
X_train['month_of_due'] = X_train['due_in_date'].dt.month
X_train['year_of_due'] = X_train['due_in_date'].dt.year

X_val['day_of_due'] = X_val['due_in_date'].dt.day
X_val['month_of_due'] = X_val['due_in_date'].dt.month
X_val['year_of_due'] = X_val['due_in_date'].dt.year


X_test['day_of_due'] = X_test['due_in_date'].dt.day
X_test['month_of_due'] = X_test['due_in_date'].dt.month
X_test['year_of_due'] = X_test['due_in_date'].dt.year
```

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

In [86]:
```
X_train ,X_val, X_test = custom(['due_in_date'])
```

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

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

In [87]:
```
X_train.dtypes
```

Out[87]:
```
cust_number                int32
buisness_year            float64
doc_id                   float64
converted_usd            float64
business_code_enc          int32
name_customer_enc          int32
cust_payment_terms_enc     int32
day_of_postingdate         int64
month_of_postingdate       int64
year_of_postingdate        int64
day_of_createdate          int64
month_of_createdate        int64
year_of_createdate         int64
day_of_due                 int64
month_of_due               int64
year_of_due                int64
dtype: object
```

In [88]:
```
X_test.dtypes
```

Out[88]:
```
cust_number                int32
buisness_year            float64
doc_id                   float64
converted_usd            float64
business_code_enc          int32
name_customer_enc          int32
cust_payment_terms_enc     int32
day_of_postingdate         int64
month_of_postingdate       int64
year_of_postingdate        int64
day_of_createdate          int64
month_of_createdate        int64
year_of_createdate         int64
day_of_due                 int64
month_of_due               int64
year_of_due                int64
dtype: object
```

In [89]:
```
X_val.dtypes
```

Out[89]:
```
cust_number                int32
buisness_year            float64
doc_id                   float64
converted_usd            float64
business_code_enc          int32
name_customer_enc          int32
cust_payment_terms_enc     int32
day_of_postingdate         int64
month_of_postingdate       int64
year_of_postingdate        int64
day_of_createdate          int64
month_of_createdate        int64
year_of_createdate         int64
day_of_due                 int64
month_of_due               int64
year_of_due                int64
dtype: object
```

Feature Selection

Filter Method

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

In [90]:
```
from sklearn.feature_selection import VarianceThreshold
constant_filter = VarianceThreshold(threshold=0)
constant_filter.fit(X_train)
len(X_train.columns[constant_filter.get_support()])
```

Out[90]:
```
16
```

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

In [91]:
```
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

In [92]:
```
x_train_T = X_train.T
print(x_train_T.duplicated().sum())
duplicated_columns = x_train_T[x_train_T.duplicated()].index.values
```

```
0
```

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

In [93]:
```
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

In [94]:
```
train=X_train.copy()
handling_correlation(train.copy(),0.85)
```

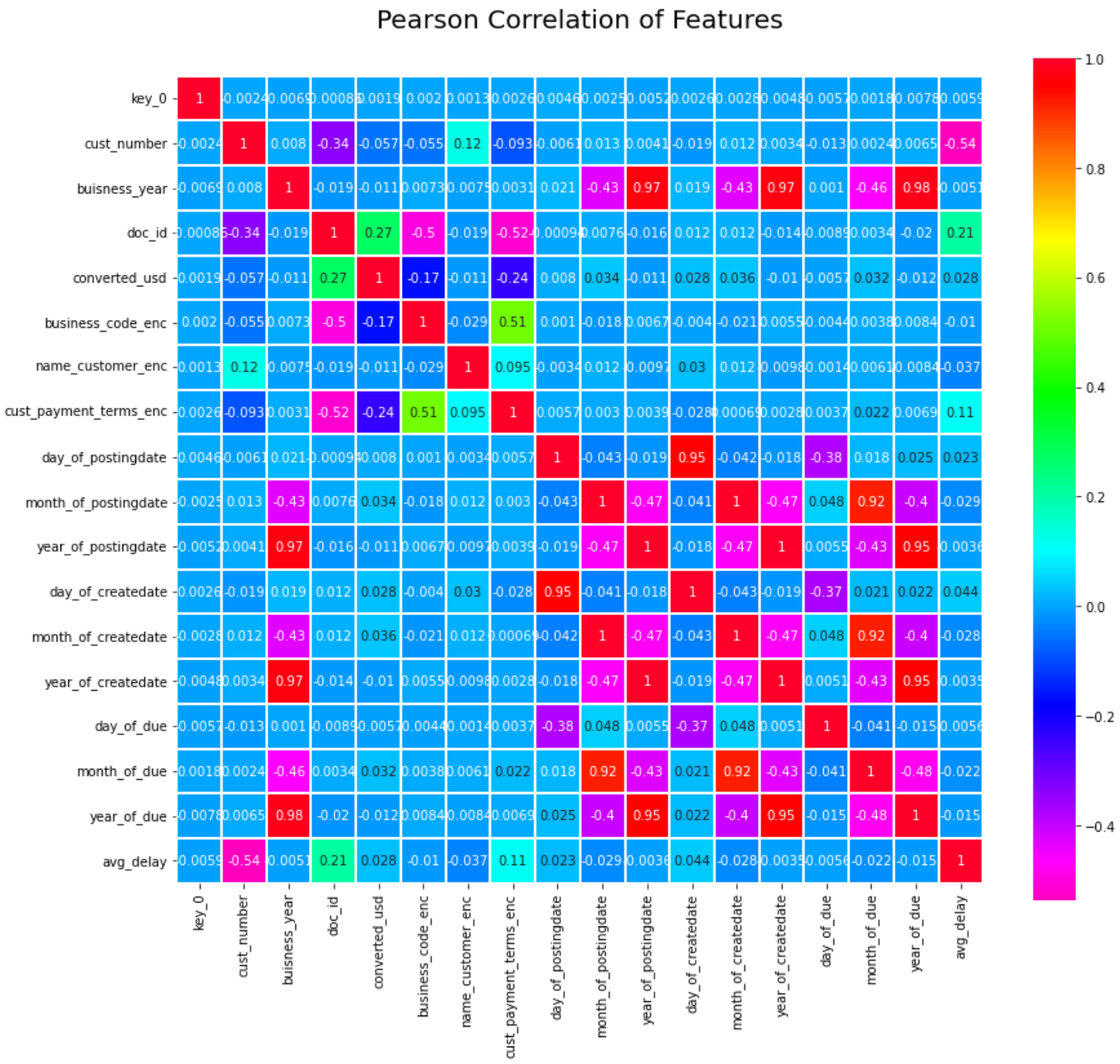
Out[94]:
```
['month_of_due',
 'year_of_due',
 'day_of_createdate',
 'month_of_createdate',
 'year_of_createdate',
 'year_of_postingdate']
```

Heatmap for X_train

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

In [95]:
```
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)
```

Out[95]: <AxesSubplot:title={'center':'Pearson Correlation of Features'}>



Calling variance threshold for threshold value = 0.8

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

In [96]:
```
from sklearn.feature_selection import VarianceThreshold
sel = VarianceThreshold(0.8)
sel.fit(X_train)
```

Out[96]:
```
VarianceThreshold(threshold=0.8)
```

In [97]:
```
sel.variances_
```

Out[97]:
```
array([1.79867713e+15, 1.15195317e-01, 8.14358365e+16, 1.01097950e+09,
       2.89199371e-01, 1.06851239e+06, 1.17330626e+02, 7.55002009e+01,
       1.22507253e+01, 1.15661120e-01, 7.71513423e+01, 1.22576245e+01,
       1.15788866e-01, 7.61732267e+01, 1.20393869e+01, 1.18619907e-01])
```

atures columns are

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

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

You need to make different blank list for different evaluation matrix

- MSE
- R2
- Algorithm

In [98]:
```
MSE_Score = []
R2_Score = []
Algorithm = []
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

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

In [99]:
```
from sklearn.linear_model import LinearRegression
Algorithm.append('LinearRegression')
regressor = LinearRegression()
regressor.fit(X_train, y_train)
predicted= regressor.predict(X_test)
```

Check for the

- Mean Square Error
- R Square Error

dicted dataset and store those data inside respective list for comparison

In [100…
```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [101…
```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

Out[101…
```
558328.6410950578
```

Display The Comparison Lists

In [102…
```
for i in Algorithm, MSE_Score, R2_Score:
    print(i,end=',')
```

```
['LinearRegression'],[301555728088.1202],[0.3184390806059786],
```

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

In [103…
```
from sklearn.svm import SVR
Algorithm.append('SVR')
regressor = SVR()
regressor.fit(X_train, y_train)
predicted= regressor.predict(X_test)
```

Check for the

- Mean Square Error
- R Square Error

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

In [104…
```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [105…
```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

Out[105…
```
698515.0460899331
```

Display The Comparison Lists

In [106…
```
for i in Algorithm, MSE_Score, R2_Score:
    print(i,end=',')
```

```
['LinearRegression', 'SVR'],[301555728088.1202, 444371876187.14124],[0.3184390806059786, -0.004346713647737532],
```

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

In [107…
```
from sklearn.tree import DecisionTreeRegressor
Algorithm.append('DecisionTreeRegressor')
regressor = DecisionTreeRegressor()
regressor.fit(X_train, y_train)
predicted= regressor.predict(X_test)
```

Check for the

- Mean Square Error
- R Square Error

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

In [108…
```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [109…
```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

Out[109…
```
439588.2071722649
```

Display The Comparison Lists

In [110…
```
for i in Algorithm, MSE_Score, R2_Score:
    print(i,end=',')
```

```
['LinearRegression', 'SVR', 'DecisionTreeRegressor'],[301555728088.1202, 444371876187.14124, 277733998172.1393],[0.3184390806059786, -0.004346713647737532, 0.3722797429804884],
```

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

In [111…
```
from sklearn.ensemble import RandomForestRegressor
Algorithm.append('RandomForestRegressor')
regressor = RandomForestRegressor()
regressor.fit(X_train, y_train)
predicted= regressor.predict(X_test)
```

Check for the

- Mean Square Error
- R Square Error

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

In [112…
```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [113…
```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

Out[113…
```
351488.8597574376
```

Display The Comparison Lists

In [114…
```
for i in Algorithm, MSE_Score, R2_Score:
    print(i,end=',')
```

```
['LinearRegression', 'SVR', 'DecisionTreeRegressor', 'RandomForestRegressor'],[30155572808
8.1202, 444371876187.14124, 277733998172.1393, 153291209090.09283],[0.3184390806059786, -
0.004346713647737532, 0.3722797429804884, 0.653539005659562],
```

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

In [115…
```
import xgboost as xgb
Algorithm.append('XGB Regressor')
regressor = xgb.XGBRegressor()
regressor.fit(X_train, y_train)
predicted = regressor.predict(X_test)
```

Check for the

- Mean Square Error
- R Square Error

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

In [116…
```
MSE_Score.append(mean_squared_error(y_test, predicted))
R2_Score.append(r2_score(y_test, predicted))
```

Check the same for the Validation set also

In [117…
```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
```

Out[117…
```
374951.5679731795
```

Display The Comparison Lists

In [118…
```
for i in Algorithm, MSE_Score, R2_Score:
    print(i,end=',')
```

```
['LinearRegression', 'SVR', 'DecisionTreeRegressor', 'RandomForestRegressor', 'XGB Regressor'],[301555728088.1202, 444371876187.14124, 277733998172.1393, 153291209090.09283, 164810594546.068],[0.3184390806059786, -0.004346713647737532, 0.3722797429804884, 0.653539005659562, 0.6275034765319765],
```

You need to make the comparison list into a comparison dataframe

In [119…
```
Comparison_list = list(zip(Algorithm, MSE_Score, R2_Score))
pd.DataFrame(Comparison_list, columns = ['Algorithm', 'MSE_Score', 'R2_Score'])
```

Out[119…

	Algorithm	MSE_Score	R2_Score
0	LinearRegression	3.015557e+11	0.318439
1	SVR	4.443719e+11	-0.004347
2	DecisionTreeRegressor	2.777340e+11	0.372280
3	RandomForestRegressor	1.532912e+11	0.653539
4	XGB Regressor	1.648106e+11	0.627503

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

In [120…
```
regressorfinal = xgb.XGBRegressor()
regressorfinal.fit(X_train, y_train)
predictedfinal = regressorfinal.predict(X_test)
predict_testfinal = regressorfinal.predict(X_val)
```

Calculate the Mean Square Error for test dataset

- Note - No need to change the code

In [121…
```
mean_squared_error(y_test,predictedfinal,squared=False)
```

Out[121…
```
405968.71128951304
```

Calculate the mean Square Error for validation dataset

In [122…
```
mean_squared_error(y_val,predict_testfinal,squared=False)
```

Out[122…
```
374951.5679731795
```

Calculate the R2 score for test

In [123…
```
r2_score(y_test,predictedfinal)
```

Out[123…
```
0.6275034765319765
```

Calculate the R2 score for Validation

In [124…
```
r2_score(y_val,predict_testfinal)
```

Out[124…
```
0.7094082728616182
```

Calculate the Accuracy for train Dataset

In [125…
```
regressorfinal.score(X_train, y_train) * 100
```

Out[125…
```
95.6043256423922
```

Calculate the accuracy for validation

In [126…
```
regressorfinal.score(X_val, y_val) * 100
```

Out[126…
```
70.94082728616182
```

Calculate the accuracy for test

In [127…
```
regressorfinal.score(X_test, y_test) * 100
```

Out[127…
```
62.75034765319765
```

Specify the reason behind choosing your machine learning model

- XGBoost is one of the most popular ML models due to its tendency to yield highly accurate results. The two main reasons why I chose this model are execution speed and model performance. XGBoost dominates structured or tabular datasets on classification and regression predictive modeling problems. Also, it provides least Mean Square Error (MSE) and highest R2 score as compared to other models which we tested.

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

Display the Nulldata

In [128…
```
nulldata
```

Out[128…

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due
3	CA02	0140105686	SYSC llc	NaT	2020.0	2.960623e+09	2020-03-30	20
7	U001	0200744019	TARG us	NaT	2020.0	1.930659e+09	2020-03-19	20
10	U001	0200418007	AM	NaT	2020.0	1.930611e+09	2020-03-11	20
14	U001	0200739534	OK systems	NaT	2020.0	1.930788e+09	2020-04-15	20
15	U001	0200353024	DECA corporation	NaT	2020.0	1.930817e+09	2020-04-23	20
...	...	...	...	...	...	...	...	
49975	U001	0200769623	WAL-MAR in	NaT	2020.0	1.930625e+09	2020-03-10	20
49980	U001	0200769623	WAL-MAR corporation	NaT	2020.0	1.930851e+09	2020-05-03	20
49982	U001	0200148860	DOLLA co	NaT	2020.0	1.930638e+09	2020-03-11	20
49992	U001	0200900909	SYSCO co	NaT	2020.0	1.930702e+09	2020-03-25	20
49995	U001	0200561861	CO corporation	NaT	2020.0	1.930797e+09	2020-04-21	20

9681 rows × 11 columns

Check for the number of rows and columns in the nulldata

In [129…
```
nulldata.shape
```

Out[129…
```
(9681, 11)
```

Check the Description and Information of the nulldata

In [130…
```
nulldata.describe()
```

Out[130…

	buisness_year	doc_id	converted_usd
count	9681.0	9.681000e+03	9681.000000
mean	2020.0	2.006165e+09	24670.052531
std	0.0	2.673629e+08	29965.999463
min	2020.0	1.930535e+09	0.504000
25%	2020.0	1.930658e+09	4159.743000
50%	2020.0	1.930731e+09	13647.935000
75%	2020.0	1.930818e+09	35090.699000
max	2020.0	2.960636e+09	457551.360000

In [131…
```
nulldata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9681 entries, 3 to 49995
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype         
---  ------                --------------  -----         
 0   business_code         9681 non-null   object        
 1   cust_number           9681 non-null   object        
 2   name_customer         9681 non-null   object        
 3   clear_date            0 non-null      datetime64[ns]
 4   buisness_year         9681 non-null   float64       
 5   doc_id                9681 non-null   float64       
 6   posting_date          9681 non-null   datetime64[ns]
 7   due_in_date           9681 non-null   datetime64[ns]
 8   baseline_create_date  9681 non-null   datetime64[ns]
 9   cust_payment_terms    9681 non-null   object        
 10  converted_usd         9681 non-null   float64       
dtypes: datetime64[ns](4), float64(3), object(4)
memory usage: 907.6+ KB
```

Storing the Nulldata into a different dataset

for BACKUP

In [132…
```
nulldata1 = nulldata.copy()
```

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

In [133…
```
from sklearn.preprocessing import LabelEncoder
business_codern = LabelEncoder()
business_codern.fit(nulldata['business_code'])
nulldata['business_code_enc'] = business_codern.transform(nulldata['business_code'])
```

…ed to manually replacing str values with numbers

- Note - No need to change the code

In [134…

```
nulldata['cust_number'] = nulldata['cust_number'].str.replace('CCCA',"1").str.replace('CCU
```

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

In [135…

```
nulldata['day_of_cleardate'] = nulldata['clear_date'].dt.day
nulldata['month_of_cleardate'] = nulldata['clear_date'].dt.month
nulldata['year_of_cleardate'] = nulldata['clear_date'].dt.year

nulldata['day_of_postingdate'] = nulldata['posting_date'].dt.day
nulldata['month_of_postingdate'] = nulldata['posting_date'].dt.month
nulldata['year_of_postingdate'] = nulldata['posting_date'].dt.year

nulldata['day_of_due'] = nulldata['due_in_date'].dt.day
nulldata['month_of_due'] = nulldata['due_in_date'].dt.month
nulldata['year_of_due'] = nulldata['due_in_date'].dt.year

nulldata['day_of_createdate'] = nulldata['baseline_create_date'].dt.day
nulldata['month_of_createdate'] = nulldata['baseline_create_date'].dt.month
nulldata['year_of_createdate'] = nulldata['baseline_create_date'].dt.year
```

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'
- ner' and store into 'name_customer_enc'

Note - No need to change the code

In [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'])
```

Check for the datatypes of all the columns of Nulldata

In [137…
```
nulldata.dtypes
```

Out[137…
```
business_code                  object
cust_number                     int32
name_customer                  object
clear_date             datetime64[ns]
buisness_year                 float64
doc_id                        float64
posting_date           datetime64[ns]
due_in_date            datetime64[ns]
baseline_create_date   datetime64[ns]
cust_payment_terms             object
converted_usd                 float64
business_code_enc               int32
day_of_cleardate              float64
month_of_cleardate            float64
year_of_cleardate             float64
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          int32
name_customer_enc               int32
dtype: object
```

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"

In [138…
```
nulldata.drop(['business_code',
'baseline_create_date',
'due_in_date',
'posting_date',
'name_customer',
```

```
'cust_payment_terms',
'day_of_cleardate',
'month_of_cleardate',
'year_of_cleardate'],axis=1, inplace = True)
```

Check the information of the "nulldata" dataframe

In [139…

```
nulldata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9681 entries, 3 to 49995
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   cust_number             9681 non-null   int32
 1   buisness_year           9681 non-null   float64
 2   doc_id                  9681 non-null   float64
 3   converted_usd           9681 non-null   float64
 4   business_code_enc       9681 non-null   int32
 5   day_of_postingdate      9681 non-null   int64
 6   month_of_postingdate    9681 non-null   int64
 7   year_of_postingdate     9681 non-null   int64
 8   day_of_due              9681 non-null   int64
 9   month_of_due            9681 non-null   int64
 10  year_of_due             9681 non-null   int64
 11  day_of_createdate       9681 non-null   int64
 12  month_of_createdate     9681 non-null   int64
 13  year_of_createdate      9681 non-null   int64
 14  cust_payment_terms_enc  9681 non-null   int32
 15  name_customer_enc       9681 non-null   int32
dtypes: float64(3), int32(4), int64(9)
memory usage: 1.1 MB
```

Compare "nulldata" with the "X_test" dataframe

- use info() method

In [140…

```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7832 entries, 39759 to 49999
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   cust_number             7832 non-null   int32
 1   buisness_year           7832 non-null   float64
 2   doc_id                  7832 non-null   float64
 3   converted_usd           7832 non-null   float64
 4   business_code_enc       7832 non-null   int32
 5   name_customer_enc       7832 non-null   int32
 6   cust_payment_terms_enc  7832 non-null   int32
 7   day_of_postingdate      7832 non-null   int64
 8   month_of_postingdate    7832 non-null   int64
 9   year_of_postingdate     7832 non-null   int64
 10  day_of_createdate       7832 non-null   int64
 11  month_of_createdate     7832 non-null   int64
 12  year_of_createdate      7832 non-null   int64
 13  day_of_due              7832 non-null   int64
 14  month_of_due            7832 non-null   int64
 15  year_of_due             7832 non-null   int64
```

```
dtypes: float64(3), int32(4), int64(9)
memory usage: 917.8 KB
```

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

In [141…
```
X_test.columns
```

Out[141…
```
Index(['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'],
      dtype='object')
```

In [142…
```
nulldata.columns
```

Out[142…
```
Index(['cust_number', 'buisness_year', 'doc_id', 'converted_usd',
       'business_code_enc', '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')
```

In [143…
```
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']]
```

Display the Final Dataset

In [144…
```
nulldata2
```

Out[144…

	cust_number	buisness_year	doc_id	converted_usd	business_code_enc	name_customer_enc	cust_
3	140105686	2020.0	2.960623e+09	3299.700	64	2712	
7	200744019	2020.0	1.930659e+09	7821.114	64	2795	
10	200418007	2020.0	1.930611e+09	2467.913	64	93	
14	200739534	2020.0	1.930788e+09	84773.955	64	2021	
15	200353024	2020.0	1.930817e+09	2608.242	64	722	
...	...	...	...	...	...	...	
49975	200769623	2020.0	1.930625e+09	9180.493	64	2987	
49980	200769623	2020.0	1.930851e+09	6229.797	64	2985	

	cust_number	buisness_year	doc_id	converted_usd	business_code_enc	name_customer_enc	cust_
49982	200148860	2020.0	1.930638e+09	3476.942	64	796	
49992	200900909	2020.0	1.930702e+09	1399.048	64	2759	
49995	200561861	2020.0	1.930797e+09	2231.502	64	547	

9681 rows × 16 columns

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

In [145…
```
final_result = regressorfinal.predict(nulldata2)
```

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

- Note - No need to change the code

In [146…
```
final_result = pd.Series(final_result,name = 'avg_delay')
```

Display the "avg_delay" column

In [147…
```
final_result
```

Out[147…
```
0       895962.125000
1      -148310.453125
2      -498700.687500
3      -194441.828125
4      -764120.375000
            ...
9676   -574882.062500
9677   -596077.187500
9678   -985206.062500
9679   -256389.390625
9680   -429530.562500
Name: avg_delay, Length: 9681, dtype: float32
```

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

In [148…
```
nulldata1.reset_index(drop=True,inplace=True)
Final = nulldata1.merge(final_result , on = nulldata.index )
```

Display the "Final" dataframe

In [149…
```
Final
```

Out[149…

	key_0	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_dat
0	3	CA02	0140105686	SYSC llc	NaT	2020.0	2.960623e+09	2020-03-3
1	7	U001	0200744019	TARG us	NaT	2020.0	1.930659e+09	2020-03-1
		U001	0200418007	AM	NaT	2020.0	1.930611e+09	2020-03-1

	key_0	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_dat
3	14	U001	0200739534	OK systems	NaT	2020.0	1.930788e+09	2020-04-1
4	15	U001	0200353024	DECA corporation	NaT	2020.0	1.930817e+09	2020-04-2
...	...	...	...	...	...	...	...	.
9676	49975	U001	0200769623	WAL-MAR in	NaT	2020.0	1.930625e+09	2020-03-1
9677	49980	U001	0200769623	WAL-MAR corporation	NaT	2020.0	1.930851e+09	2020-05-0
9678	49982	U001	0200148860	DOLLA co	NaT	2020.0	1.930638e+09	2020-03-1
9679	49992	U001	0200900909	SYSCO co	NaT	2020.0	1.930702e+09	2020-03-2
9680	49995	U001	0200561861	CO corporation	NaT	2020.0	1.930797e+09	2020-04-2

9681 rows × 13 columns

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

In [150…
```
Final.shape
```

Out[150…
```
(9681, 13)
```

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

In [151…
```
Final['clear_date'] = pd.to_datetime(Final['due_in_date']) + pd.to_timedelta(Final['avg_de
```

Display the "clear_date" column

In [152…
```
Final['clear_date']
```

Out[152…
```
0      2020-04-20 08:52:42.125000
1      2020-04-01 06:48:09.546875
2      2020-03-20 05:28:19.312500
3      2020-04-27 17:59:18.171875
4      2020-04-17 03:44:39.625000
                  ...
9676   2020-03-18 08:18:37.937500
9677   2020-05-11 02:25:22.812500
9678   2020-03-14 14:19:53.937500
9679   2020-04-06 00:46:50.609375
9680   2020-05-01 00:41:09.437500
Name: clear_date, Length: 9681, dtype: datetime64[ns]
```

Convert the average delay into number of days format

- Note - Formula = avg_delay//(24 * 3600)

e is given for this, no need to change

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

Display the "avg_delay" column

In [154…
```
Final['avg_delay']
```

Out[154…
```
0       10.0
1       -2.0
2       -6.0
3       -3.0
4       -9.0
        ...
9676    -7.0
9677    -7.0
9678   -12.0
9679    -3.0
9680    -5.0
Name: avg_delay, Length: 9681, dtype: float64
```

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

In [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)
```

Now you need to drop "key_0" and "avg_delay" columns from the "Final" Dataframe

In [156…
```
Final.drop(['key_0', 'avg_delay'],axis=1, inplace = True)
```

Display the count of each categoty of new "Aging Bucket" column

In [157…
```
Final['Aging Bucket'].value_counts()
```

Out[157…
```
0-15              1598
16-30               98
31-45               63
46-60                7
Greatar than 60      6
Name: Aging Bucket, dtype: int64
```

Display your final dataset with aging buckets

In [158…
```
Final
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date
0	CA02	0140105686	SYSC llc	2020-04-20 08:52:42.125000	2020.0	2.960623e+09	2020-03-30
1	U001	0200744019	TARG us	2020-04-01 06:48:09.546875	2020.0	1.930659e+09	2020-03-19
2	U001	0200418007	AM	2020-03-20 05:28:19.312500	2020.0	1.930611e+09	2020-03-11
3	U001	0200739534	OK systems	2020-04-27 17:59:18.171875	2020.0	1.930788e+09	2020-04-15
4	U001	0200353024	DECA corporation	2020-04-17 03:44:39.625000	2020.0	1.930817e+09	2020-04-23
...	...	...	...	...	...	...	...
9676	U001	0200769623	WAL-MAR in	2020-03-18 08:18:37.937500	2020.0	1.930625e+09	2020-03-10
9677	U001	0200769623	WAL-MAR corporation	2020-05-11 02:25:22.812500	2020.0	1.930851e+09	2020-05-03
9678	U001	0200148860	DOLLA co	2020-03-14 14:19:53.937500	2020.0	1.930638e+09	2020-03-11
9679	U001	0200900909	SYSCO co	2020-04-06 00:46:50.609375	2020.0	1.930702e+09	2020-03-25
9680	U001	0200561861	CO corporation	2020-05-01 00:41:09.437500	2020.0	1.930797e+09	2020-04-21

9681 rows × 12 columns

Store this dataframe into the .csv format

In [159…

```
Final.to_csv('HRC82581W_Aryan_Madaan.csv')
```

END OF THE PROJECT