Payment Date Prediction

Importing related Libraries

```
import pandas as pd
import numpy as np
import  matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Store the dataset into the Dataframe

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

Check the shape of the dataframe

```
df.shape
```

```
(50000, 19)
```

+ Code + Text

Check the Detail information of the dataframe

```
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
```

Display All the column names

```
df.columns
```

```
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

```
df.describe()
```

	buisness_year	doc_id	document_create_date	document_create_date.1	due_in_date	posting_id	area_business	total_open_amount	bas
count	50000.000000	5.000000e+04	5.000000e+04	5.000000e+04	5.000000e+04	50000.0	0.0	50000.000000	
mean	2019.305700	2.012238e+09	2.019351e+07	2.019354e+07	2.019368e+07	1.0	NaN	32337.021651	
std	0.460708	2.885235e+08	4.496041e+03	4.482134e+03	4.470614e+03	0.0	NaN	39205.975231	
min	2019.000000	1.928502e+09	2.018123e+07	2.018123e+07	2.018122e+07	1.0	NaN	0.720000	
25%	2019.000000	1.929342e+09	2.019050e+07	2.019051e+07	2.019052e+07	1.0	NaN	4928.312500	
50%	2019.000000	1.929964e+09	2.019091e+07	2.019091e+07	2.019093e+07	1.0	NaN	17609.010000	
75%	2020.000000	1.930619e+09	2.020013e+07	2.020013e+07	2.020022e+07	1.0	NaN	47133.635000	
max	2020.000000	9.500000e+09	2.020052e+07	2.020052e+07	2.020071e+07	1.0	NaN	668593.360000	

Data Cleaning

- Show top 5 records from the dataset

```
df.head()
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	document_create_date	document_create_date
0	U001	0200769623	WAL-MAR corp	2020-02-11 00:00:00	2020.0	1.930438e+09	2020-01-26	20200125	2020012
1	U001	0200980828	BEN E	2019-08-08 00:00:00	2019.0	1.929646e+09	2019-07-22	20190722	2019072
2	U001	0200792734	MDV/ trust	2019-12-30 00:00:00	2019.0	1.929874e+09	2019-09-14	20190914	2019091
3	CA02	0140105686	SYSC llc	NaN	2020.0	2.960623e+09	2020-03-30	20200330	2020033

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

```
df.isnull().mean()*100
```

```
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

```
df[["invoice_id","doc_id"]]
df[["invoice_id","doc_id"]].corr()
```

	invoice_id	doc_id
invoice_id	1.0	1.0
doc_id	1.0	1.0

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

```
df[["baseline_create_date","document_create_date","document_create_date.1"]]
df[["baseline_create_date","document_create_date","document_create_date.1"]].corr()
```

	baseline_create_date	document_create_date	document_create_date.1
baseline_create_date	1.000000	0.994078	0.999527
document_create_date	0.994078	1.000000	0.994547
document_create_date.1	0.999527	0.994547	1.000000

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

```
# nunique() if return 1 then it is constant
df.posting_id.value_counts()
df.posting_id.nunique()
```

```
1
```

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

```
# number of 0 =40000
# number of 1=10000
df.isOpen.value_counts()
```

```
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

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

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

```
df.shape
df.head()
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	invoice_currency	total_open_ar
0	U001	0200769623	WAL-MAR corp	2020-02-11 00:00:00	2020.0	1.930438e+09	2020-01-26	20200210.0	USD	542
1	U001	0200980828	BEN E	2019-08-08 00:00:00	2019.0	1.929646e+09	2019-07-22	20190811.0	USD	796
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	20190929.0	USD	23

Show all the Dublicate rows from the dataframe

```
# to check the duplicacy of the row true=duplicate , false = not duplictate
df.duplicated()
df.duplicated().sum()

# to print the duplicated row
df[df.duplicated()]
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	invoice_currency	total_ope
1041	U001	0200769623	WAL-MAR in	2019-03-12 00:00:00	2019.0	1.928870e+09	2019-02-28	20190315.0	USD	
2400	U001	0200769623	WAL-MAR trust	2019-08-28 00:00:00	2019.0	1.929758e+09	2019-08-18	20190902.0	USD	
2584	U001	0200769623	WAL-MAR corporation	2019-12-16 00:00:00	2019.0	1.930217e+09	2019-12-04	20191219.0	USD	

Display the Number of Dublicate Rows

```
df.duplicated().sum()
```

```
1161
```

2019-08-

Drop all the Dublicate Rows

```
df.drop_duplicates(keep='first',inplace=True)
```

Now check for all dublicate rows now

- Note - It must be 0 by now

```
df.duplicated().sum()
```

```
0
```

Check for the number of Rows and Columns in your dataset

```
num_rows=len(df)
num_columns=len(df.columns)
print("number of rows  : ", num_rows)
print("number of columns : ",num_columns)
```

```
number of rows  :  48839
number of columns :  12
```

Find out the total count of null values in each columns

```
# null value count of each column
df.isnull().sum()
```

```
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

```
df.dtypes
```

```
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

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

```
clear_date              object
posting_date            object
due_in_date             float64
baseline_create_date    float64
dtype: object
```

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**

```
# where the format is already like the given format if we do we will get the value error so we donot need to do that, where there is float or interger we have to give the format
df[["clear_date","posting_date","due_in_date","baseline_create_date"]]
df['clear_date']=pd.to_datetime(df['clear_date'])
df['posting_date']=pd.to_datetime(df['posting_date'])
df['due_in_date']=pd.to_datetime(df['due_in_date'],format="%Y%m%d")
df['baseline_create_date']=pd.to_datetime(df['baseline_create_date'],format="%Y%m%d")
```

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

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

```
clear_date            datetime64[ns]
posting_date          datetime64[ns]
due_in_date           datetime64[ns]
baseline_create_date    datetime64[ns]
dtype: object
```

the invoice_currency column contains two different categories, USD and CAD

- Please do a count of each currency

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

```
USD    45011
CAD     3828
Name: invoice_currency, dtype: int64
```

display the "total_open_amount" column value

```
df["total_open_amount"]
```

```
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

```
# will return the index array of the element where the condition is true
# np.where(df.invoice_currency=='CAD')

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

Display the new "converted_usd" column values

```
df.converted  usd
```

```
0        54273.28
1        79656.60
2         2253.86
3         2309.79
4        33133.29
           ...
49995     3187.86
49996     6766.54
49997     6120.86
49998       63.48
49999     1790.30
Name: converted_usd, Length: 48839, dtype: float64
```

Display year wise total number of record

- Note - use "buisness_year" column for this

```
df.buisness_year.value_counts()
```

```
2019.0    33975
2020.0    14864
Name: buisness_year, dtype: int64
```

Write the code to delete the following columns

- 'invoice_currency'
- 'total_open_amount',

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

Write a code to check the number of columns in dataframe

```
#number of columns
len(df.columns)
df.head()
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	cust_payn
0	U001	0200769623	WAL-MAR corp	2020-02-11	2020.0	1.930438e+09	2020-01-26	2020-02-10	2020-01-26	
1	U001	0200980828	BEN E	2019-08-08	2019.0	1.929646e+09	2019-07-22	2019-08-11	2019-07-22	
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	2019-09-29	2019-09-14	
3	CA02	0140105686	SYSC llc	NaT	2020.0	2.960623e+09	2020-03-30	2020-04-10	2020-03-31	

Splitting the Dataset

Look for all columns containing null value

- Note - Output expected is only one column

```
# the null_columns list contains only single value 'clear_date'
null_columns=[column for column in df.columns if df[column].isnull().any()]
print(null_columns)
```

```
['clear_date']
```

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

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

```
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 NO 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 Null data in that column (This is going to be our test dataset)

```
nulldata=df.loc[df.clear_date.isnull()]
maindata=df.loc[df.clear_date.notnull()]
```

Check the number of Rows and Columns for both the dataframes

```
# for maindata
print("rows : ",len(maindata))
print("columns : ",len(maindata.columns))
```

```
rows :  39158
columns :  11
```

```
# for nulldata
print("rows : ",len(nulldata))
print("columns : ",len(nulldata.columns))
```

```
rows :  9681
columns :  11
```

Display the 5 records from maindata and nulldata dataframes

```
maindata.head()
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	cust_payn
0	U001	0200769623	WAL-MAR corp	2020-02-11	2020.0	1.930438e+09	2020-01-26	2020-02-10	2020-01-26	
1	U001	0200980828	BEN E	2019-08-08	2019.0	1.929646e+09	2019-07-22	2019-08-11	2019-07-22	
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	2019-09-29	2019-09-14	
			WAL-MAR	2019-11-						

```
nulldata.head()
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	cust_pay
3	CA02	0140105686	SYSC llc	NaT	2020.0	2.960623e+09	2020-03-30	2020-04-10	2020-03-31	
7	U001	0200744019	TARG us	NaT	2020.0	1.930659e+09	2020-03-19	2020-04-03	2020-03-19	
10	U001	0200418007	AM	NaT	2020.0	1.930611e+09	2020-03-11	2020-03-26	2020-03-11	
14	U001	0200739534	OK systems	NaT	2020.0	1.930788e+09	2020-04-15	2020-04-30	2020-04-15	
15	U001	0200353024	DECA corporation	NaT	2020.0	1.930817e+09	2020-04-23	2020-04-26	2020-04-16	

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

```
maindata['Delay']=maindata['clear_date']-maindata['due_in_date']
maindata.Delay
maindata.head()
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	cust_payn
0	U001	0200769623	WAL-MAR corp	2020-02-11	2020.0	1.930438e+09	2020-01-26	2020-02-10	2020-01-26	
1	U001	0200980828	BEN E	2019-08-08	2019.0	1.929646e+09	2019-07-22	2019-08-11	2019-07-22	
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	2019-09-29	2019-09-14	
			WAL-MAR	2019-11-						

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

```
# will return a grouby object that contains information about groups
# print(maindata.groupby('name_customer'))

# maindata.name_customer.value_counts()
groupeddata=maindata.groupby(by='name_customer')['Delay'].mean(numeric_only=False)
display(groupeddata)
```

```
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
```

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

```
maindata["avg_delay"]=maindata.name_customer.map(groupeddata)
display(maindata)
# maindata.loc[maindata.name_customer=="WAL-MAR trust"]
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	cust_

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

2	U001	0200792734	MDV/ trust		2019.0	1.929874e+09	2019-09-14	2019-09-29	2019-09-14

```
maindata["avg_delay"]= maindata["avg_delay"].dt.total_seconds()
```

4	U001	0200769623			2019.0	1.930148e+09	2019-11-13	2019-11-28	2019-11-13

```
maindata.head()
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	cust_payn
0	U001	0200769623	WAL-MAR corp	2020-02-11	2020.0	1.930438e+09	2020-01-26	2020-02-10	2020-01-26	
1	U001	0200980828	BEN E	2019-08-08	2019.0	1.929646e+09	2019-07-22	2019-08-11	2019-07-22	
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	2019-09-29	2019-09-14	
			WAL-MAR	2019-11-						

Display the maindata dataframe

```
display(maindata)
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	cust_
0	U001	0200769623	WAL-MAR corp	2020-02-11	2020.0	1.930438e+09	2020-01-26	2020-02-10	2020-01-26	
1	U001	0200980828	BEN E	2019-08-08	2019.0	1.929646e+09	2019-07-22	2019-08-11	2019-07-22	
2	U001	0200792734	MDV/ trust	2019-12-30	2019.0	1.929874e+09	2019-09-14	2019-09-29	2019-09-14	
4	U001	0200769623	WAL-MAR foundation	2019-11-25	2019.0	1.930148e+09	2019-11-13	2019-11-28	2019-11-13	
5	CA02	0140106181	THE corporation	2019-12-04	2019.0	2.960581e+09	2019-09-20	2019-10-04	2019-09-24	

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

```
maindata.drop(["Delay","clear_date"],axis=1,inplace=True)
```

```
maindata.head()
```

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

```
from sklearn.model_selection import train_test_split
```

```
y=maindata["avg_delay"].to_frame()
display(y)
```

	avg_delay
0	-2.334702e+05

```
X=maindata.drop("avg_delay",axis=1)
display(X)
```

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

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

```
X_train,X_loc_test,y_train,y_loc_test=train_test_split(X,y,test_size=0.4)
```

```
X_train.shape
```

```
(23494, 10)
```

```
X_loc_test.shape
```

```
(15664, 10)
```

```
y_train.shape
```

```
(23494, 1)
```

```
y_loc_test.shape
```

```
(15664, 1)
```

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

```
# for X_train
print("number of rows : ",len(X_train))
print("number of columns : ",len(X_train.columns))
```

```
number of rows :  23494
number of columns :  10
```

```
# for X_loc_test
print("number of rows : ",len(X_loc_test))
print("number of columns : ",len(X_loc_test.columns))
```

```
number of rows :  15664
number of columns :  10
```

```
# for y_train
print("number of rows : ",len(y_train))
print("number of columns : ",len(y_train.columns))
```

```
number of rows :  23494
number of columns :  1
```

```
# for y_loc_test
print("number of rows : ",len(y_loc_test))
print("number of columns : ",len(y_loc_test.columns))
```

```
number of rows :  15664
number of columns :  1
```

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

```
X_val,X_test,y_val,y_test=train_test_split(X_loc_test,y_loc_test,test_size=0.5)
```

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

```
# for X_val
print("number of rows : ",len(X_val))
print("number of columns : ",len(X_val.columns))
```

```
number of rows :  7832
number of columns :  10
```

```
# for X_test
print("number of rows : ",len(X  test))
```

```
print("number of columns : ",len(X_test.columns))
```

```
number of rows :  7832
number of columns :  10
```

```
# for y_val
print("number of rows : ",len(y_val))
print("number of columns : ",len(y_val.columns))
```

```
number of rows :  7832
number of columns :  1
```

```
# for y_test
print("number of rows : ",len(y_test))
print("number of columns : ",len(y_test.columns))
```

```
number of rows :  7832
number of columns :  1
```

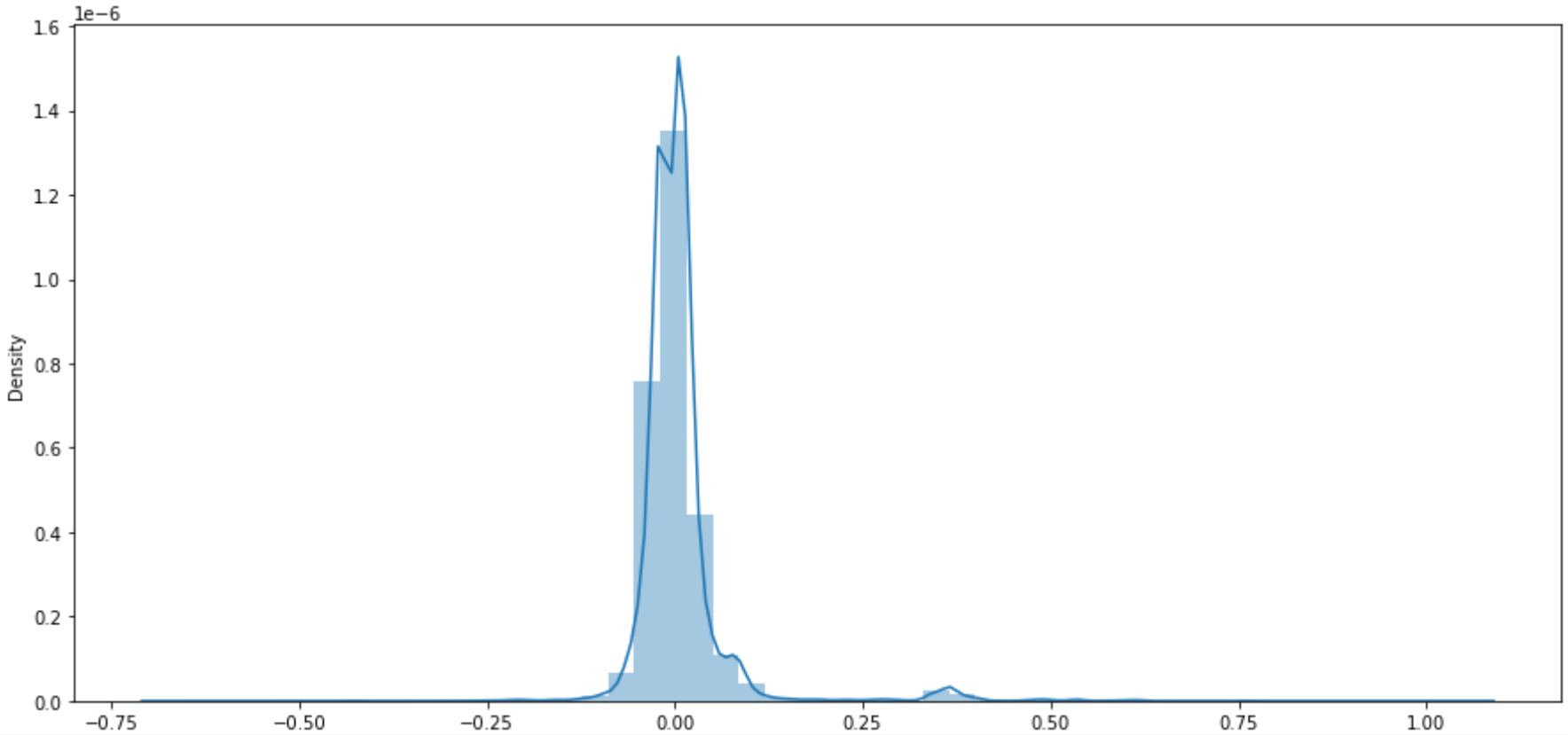
Exploratory Data Analysis (EDA)

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

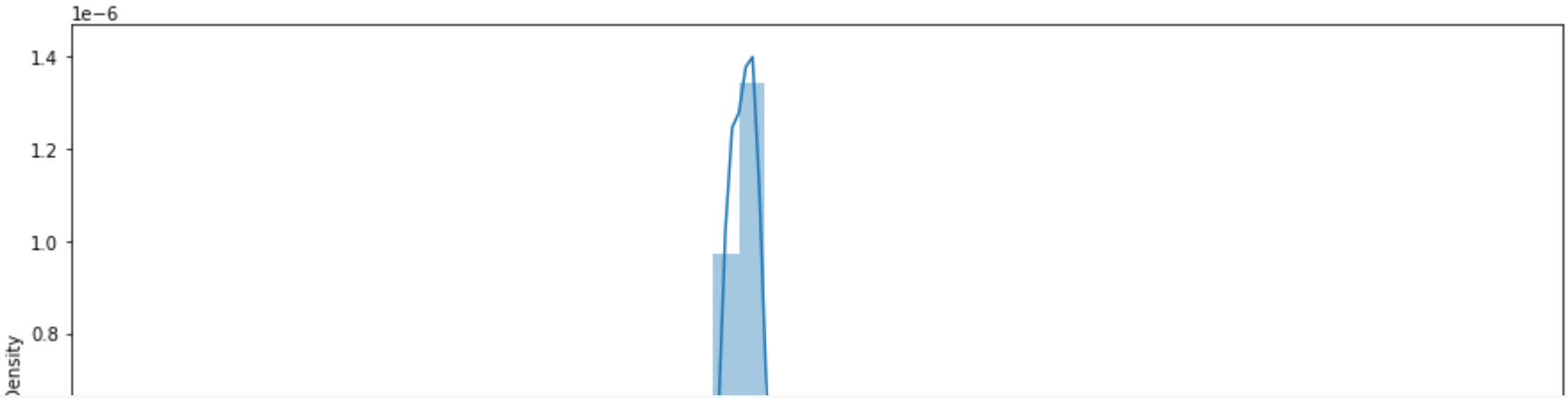
- Note - You are expected to make a distribution plot for the target variable

```
import plotly.figure_factory as ff
```

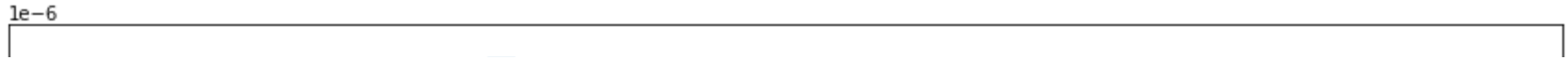
```
plt.subplots(figsize=(15,7))
displot_y_train=sns.distplot(y_train["avg_delay"])
```




```
plt.subplots(figsize=(15,7))
displot_y_val=sns.distplot(y_val["avg_delay"])
```




```
plt.subplots(figsize=(15,7))
displot_y_test=sns.distplot(y_test["avg_delay"])
```



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()

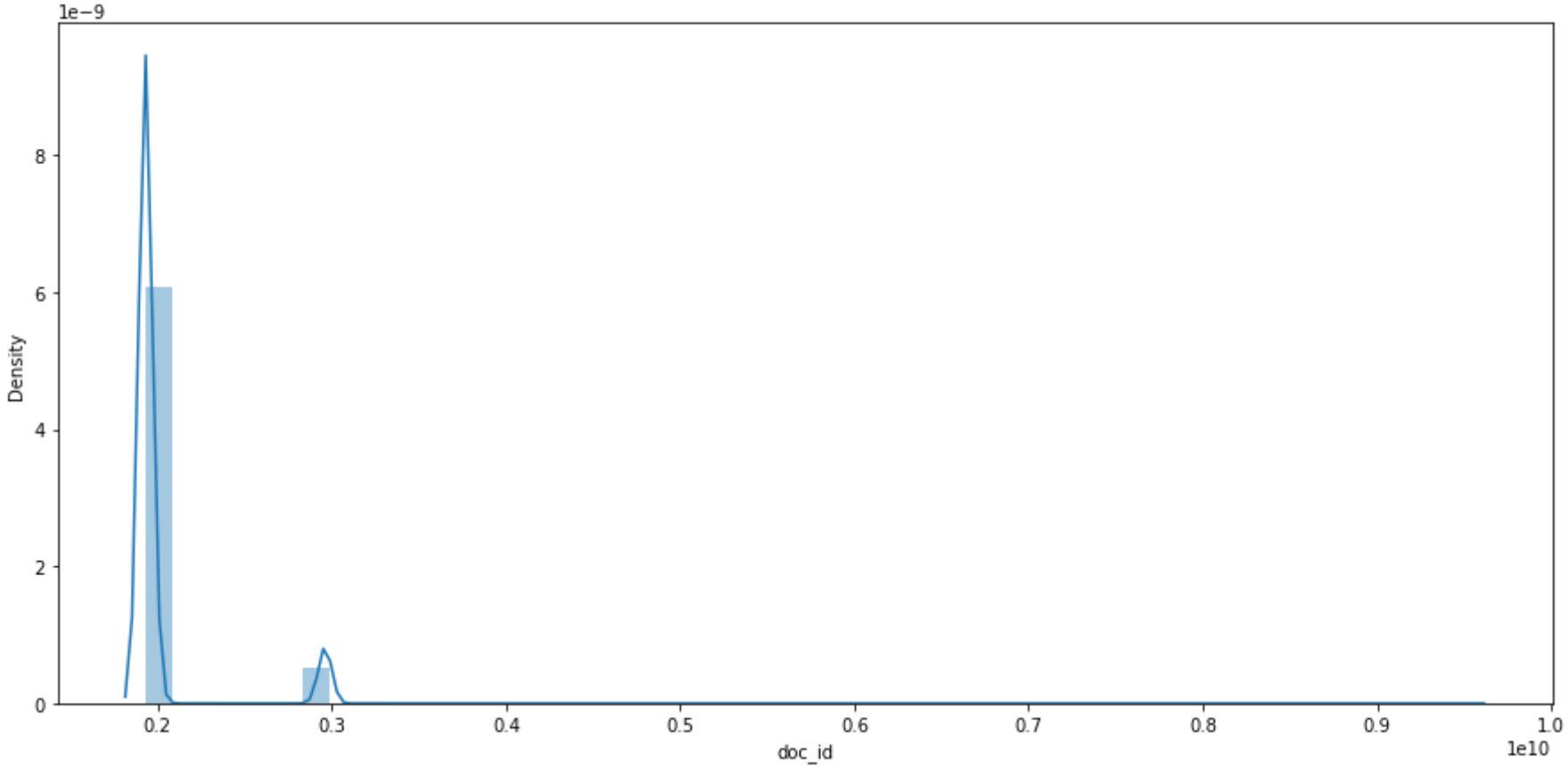
```
doc_id_grouped=X_train.groupby(by=['name_customer'],as_index=False)['doc_id'].count()
doc_id_grouped
```

	name_customer	doc_id
0	11078 us	1
1	17135 associates	1
2	236008 associates	1
3	99 CE	2
4	99 CE associates	2
...	...	...
3095	YEN BROS	1
3096	YEN BROS corp	1
3097	YEN BROS corporation	1
3098	ZARCO co	1
3099	ZIYAD us	1

3100 rows × 2 columns

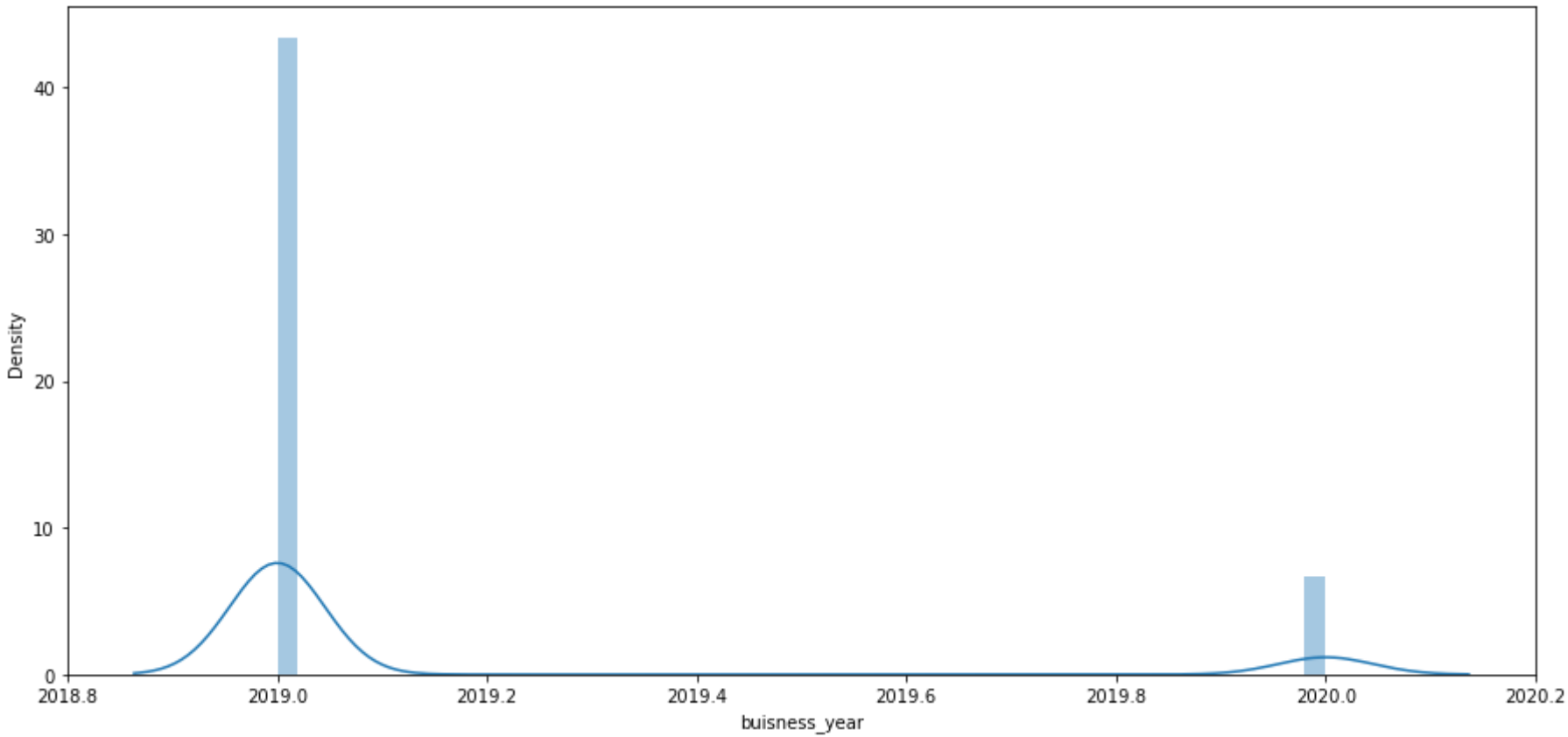
You can make another distribution plot of the "doc_id" column from x_train

```
plt.subplots(figsize=(15,7))
X_train_doc_id_plot=sns.distplot(X_train["doc_id"])
```



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

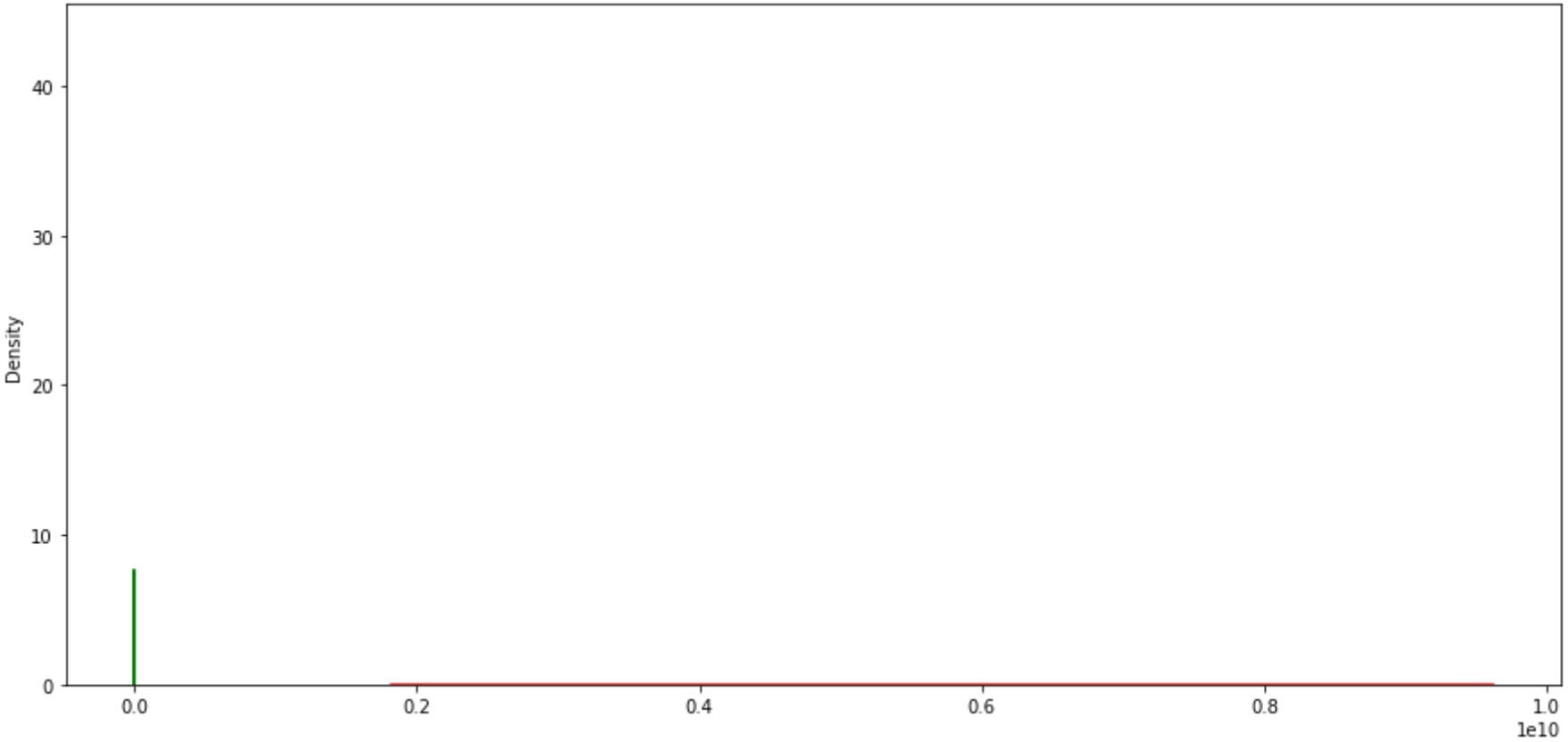
```
# X_train.head()
plt.subplots(figsize=(15,7))
business_year_xtrain=sns.distplot(X_train["buisness_year"])
```



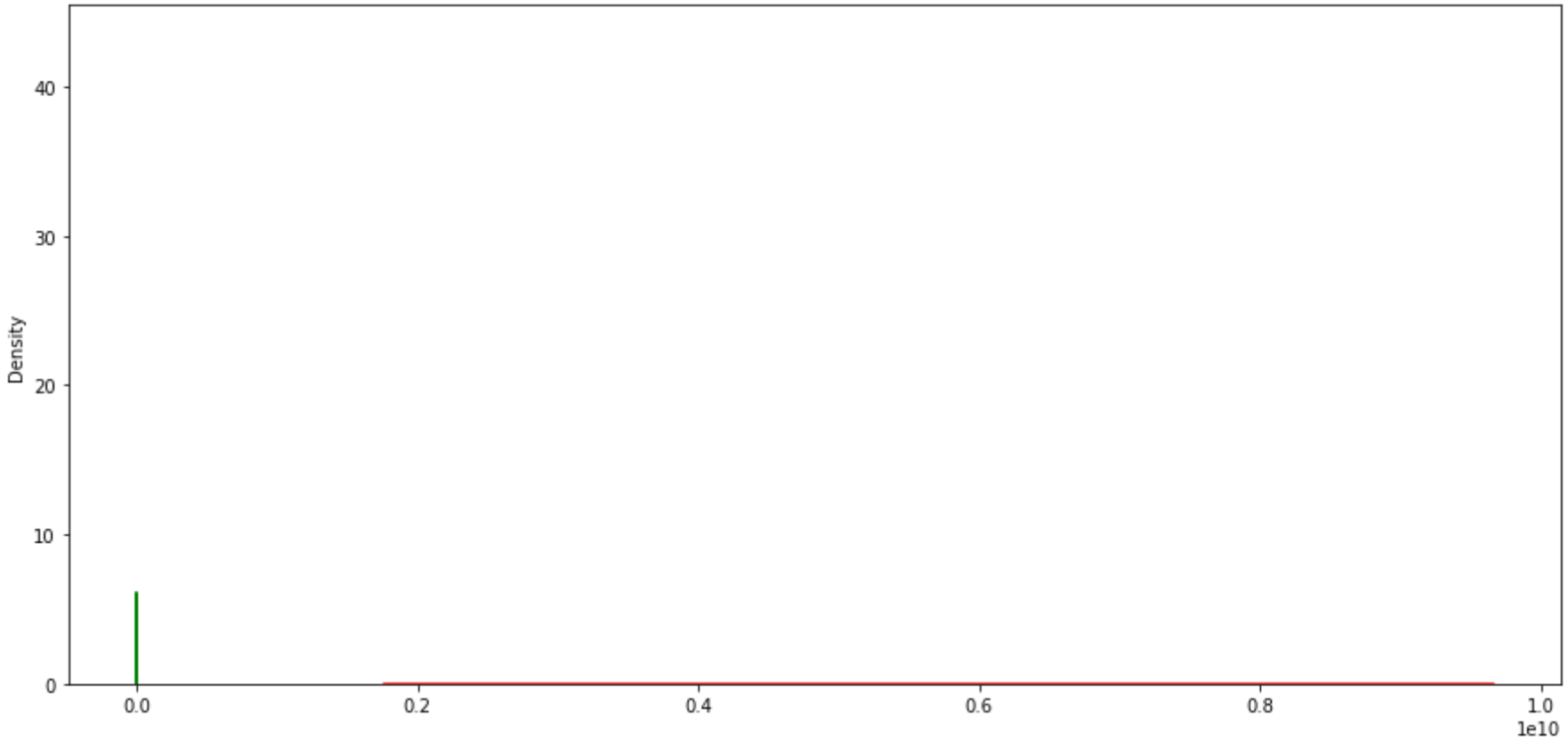

```
# X_val.head()
plt.subplots(figsize=(15,7))
business_year_xval=sns.distplot(X_val["buisness_year"])
```


```
# X_test.head()
plt.subplots(figsize=(15,7))
business_year_xtest=sns.distplot(X_test["buisness_year"])
```

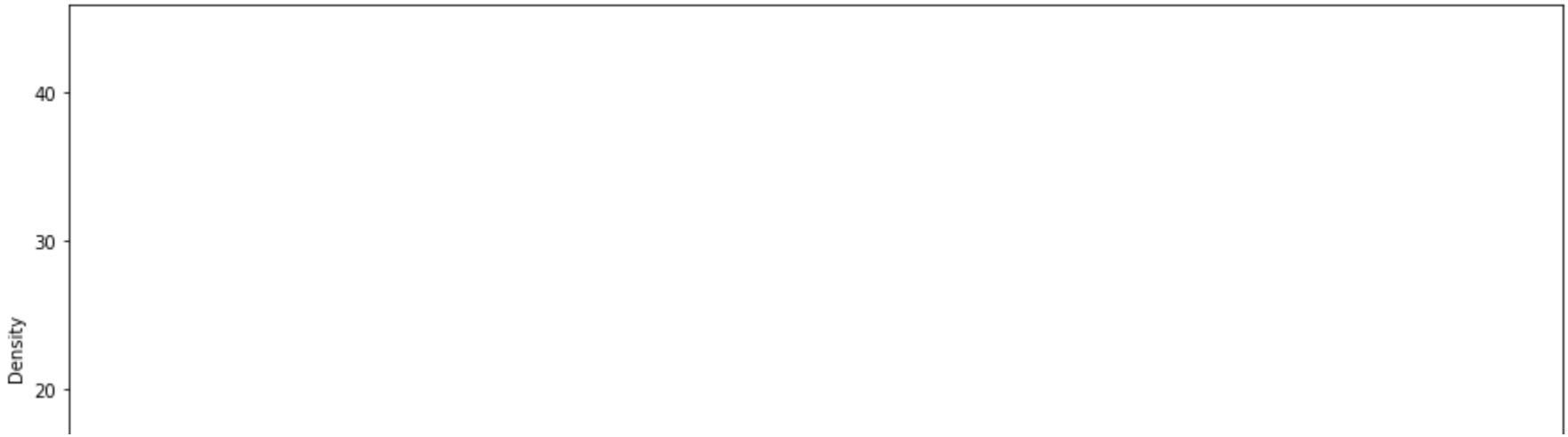

```
plt.subplots(figsize=(15,7))
business_year_xtrain=sns.distplot([X_train["buisness_year"]],color="green")
doc_id_xtrain=sns.distplot([X_train["doc_id"]],color="red")
```




```
plt.subplots(figsize=(15,7))
business_year_xval=sns.distplot([X_val["buisness_year"]],color="green")
doc_id_xval=sns.distplot([X_val["doc_id"]],color="red")
```




```
plt.subplots(figsize=(15,7))
business_year_xtest=sns.distplot([X_test["buisness_year"]],color="green")
doc_id_xtest=sns.distplot([X_test["doc_id"]],color="red")
```



Feature Engineering

Display and describe the X_train dataframe

```
display(X_train)
```

	business_code	cust_number	name_customer	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	cust_payment_ter
12476	U001	0200739006	AL	2019.0	1.930015e+09	2019-10-16	2019-10-31	2019-10-16	NA
25897	U013	0140103278	COS corporation	2020.0	1.991839e+09	2020-02-15	2020-04-15	2020-02-15	NA
49686	U001	0100043949	HOME R trust	2019.0	1.929747e+09	2019-08-21	2019-09-05	2019-08-21	NA
27284	U001	0200416837	DEC corp	2019.0	1.929962e+09	2019-10-04	2019-10-24	2019-10-01	NAI

```
X_train.describe()
```

	buisness_year	doc_id	converted_usd
count	23494.000000	2.349400e+04	23494.000000
mean	2019.133609	2.013074e+09	30658.801886
std	0.340239	2.869666e+08	36825.024553
min	2019.000000	1.928502e+09	0.790000
25%	2019.000000	1.929186e+09	4390.625000
50%	2019.000000	1.929739e+09	16664.115000
75%	2019.000000	1.930212e+09	44988.660000
max	2020.000000	9.500000e+09	632134.240000

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

```
from sklearn.preprocessing import LabelEncoder
```

```
business_coder = LabelEncoder()
business_coder.fit_transform(X_train["business_code"])
```

```
array([1, 5, 1, ..., 1, 1, 0])
```

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

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

```
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

```
X_train[["business_code","business_code_enc"]]
```

	business_code	business_code_enc
12476	U001	1
25897	U013	5
49686	U001	1
27284	U001	1
29658	U001	1

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

30121	U001	1

```
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

```
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

```
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.

```
#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

```
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'])
```

```
display(X_train)
```

	cust_number	name_customer	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	cust_payment_terms	converted_
12476	200739006	AL	2019.0	1.930015e+09	2019-10-16	2019-10-31	2019-10-16	NAA8	17850.(
25897	140103278	COS corporation	2020.0	1.991839e+09	2020-02-15	2020-04-15	2020-02-15	NAUZ	8366.:
49686	100043949	HOME R trust	2019.0	1.929747e+09	2019-08-21	2019-09-05	2019-08-21	NAA8	17157.:
27284	200416837	DEC corp	2019.0	1.929962e+09	2019-10-04	2019-10-24	2019-10-01	NAM4	3072.9
29658	100031704	DELHAIZE corporation	2019.0	1.929572e+09	2019-07-04	2019-07-19	2019-07-04	NAA8	99262.:
...	...	...	...	...	...	...	...	...	
2277	200794332	COST in	2019.0	1.929862e+09	2019-09-12	2019-09-27	2019-09-12	NAAX	14942.:
15422	200762301	C&S WH associates	2019.0	1.929068e+09	2019-04-01	2019-04-16	2019-04-01	NAC6	175012.:
30121	200701040	SOUTHE associates	2019.0	1.929996e+09	2019-10-15	2019-10-30	2019-10-15	NAA8	7428.:
25578	200771157	WEIS trust	2020.0	1.930402e+09	2020-01-16	2020-01-31	2020-01-16	NAA8	53236.4
40723	140106408	WAL M	2019.0	2.960527e+09	2019-02-01	2019-02-11	2019-02-01	CA10	42202.4

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

```
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

```
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'])
```

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

```
display(X_train)
```

	cust_number	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	converted_usd	business_code_enc	name_custome
12476	200739006	2019.0	1.930015e+09	2019-10-16	2019-10-31	2019-10-16	17850.080	1	
25897	140103278	2020.0	1.991839e+09	2020-02-15	2020-04-15	2020-02-15	8366.300	5	
49686	100043949	2019.0	1.929747e+09	2019-08-21	2019-09-05	2019-08-21	17157.250	1	

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

- Note - You are expected yo use dtype

```
X_train.dtypes
```

```
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
```

```
X_test.dtypes
```

```
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
```

```
X_val.dtypes
```

```
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

```
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

```
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

```
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

```
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

```
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

```
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

```
X_train.dtypes
```

```
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
```

```
X_val.dtypes
```

```
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
```

```
X_test.dtypes
```

```
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

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

```
16
```

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

```
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

```
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

```
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

```
train=X_train.copy()
handling_correlation(train.copy(),0.85)
```

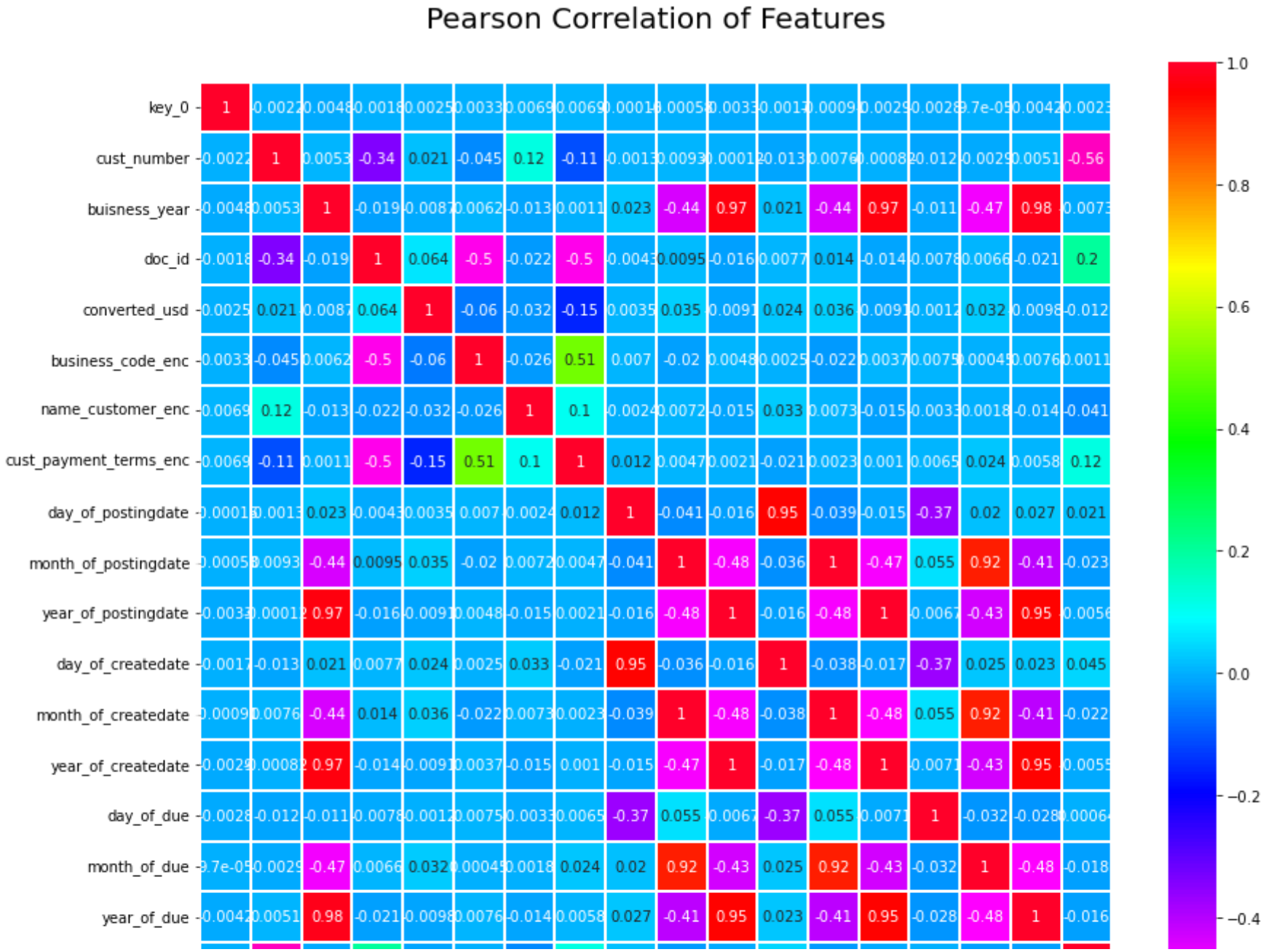
```
['year_of_postingdate',
 'month_of_due',
 'year_of_createdate',
 'day_of_createdate',
 'year_of_due',
 'month_of_createdate']
```

Heatmap for X_train

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

```
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)
```

<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

```
from sklearn.feature_selection import VarianceThreshold
sel = VarianceThreshold(0.8)
sel.fit(X_train)
```

```
VarianceThreshold(threshold=0.8)
```

```
sel.variances_
```

```
array([1.80332421e+15, 1.15757328e-01, 8.23463046e+16, 1.35602471e+09,
       2.84880474e-01, 1.07614740e+06, 1.32623109e+02, 7.57092784e+01,
       1.23277737e+01, 1.16214144e-01, 7.72557966e+01, 1.23335891e+01,
       1.16426729e-01, 7.62167269e+01, 1.21380579e+01, 1.19236899e-01])
```

```
display(X_train)
```

	cust_number	buisness_year	doc_id	converted_usd	business_code_enc	name_customer_enc	cust_payment_terms_enc	day_of_postingda
12476	200739006	2019.0	1.930015e+09	17850.080	1	53	20	
25897	140103278	2020.0	1.991839e+09	8366.300	5	610	51	

Important features 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

```
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

```
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

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

```
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

```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
r2_score(y_val,predict_test)
```

```
0.2950132568123759
```

Display The Comparison Lists

```
for i in Algorithm, MSE_Score, R2_Score:
  print(i,end=',')
```

```
['LinearRegression'],[283079398483.57697],[0.32133380599602734],
```

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

```
from sklearn.svm import SVR
Algorithm.append('Support Vector Regression')
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

```
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

```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
r2_score(y_val,predict_test)
```

```
-0.00634349454861427
```

Display The Comparison Lists

```
for i in Algorithm, MSE_Score, R2_Score:
  print(i,end=',')
```

```
['LinearRegression', 'Support Vector Regression'],[283079398483.57697, 420267116021.6393],[0.32133380599602734, -0.007565671056698919],
```

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

```
from sklearn.tree import DecisionTreeRegressor
Algorithm.append('DecisionTreeRegression')
reg = DecisionTreeRegressor()
reg.fit(X_train, y_train)
predicted = reg.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

```
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

```
predict_test= reg.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
r2_score(y_val,predict_test)
```

```
0.4424938202117279
```

Display The Comparison Lists

```
for i in Algorithm, MSE  Score, R2  Score:
```

```
print(i,end=',')
```

```
['LinearRegression', 'Support Vector Regression', 'DecisionTreeRegression'],[283079398483.57697, 420267116021.6393, 195396127112.5757],[0.32133380599602734, -0.00
```

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

```
from sklearn.ensemble import RandomForestRegressor
Algorithm.append('RandomForestRegression')
rf = RandomForestRegressor()
rf.fit(X_train,y_train)
predicted=rf.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

```
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

```
predict_test= rf.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
r2_score(y_val,predict_test)
```

```
0.6437566125842333
```

Display The Comparison Lists

```
for i in Algorithm, MSE_Score, R2_Score:
  print(i,end=',')
```

```
['LinearRegression', 'Support Vector Regression', 'DecisionTreeRegression', 'RandomForestRegression'],[283079398483.57697, 420267116021.6393, 195396127112.5757, 11
```

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

```
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

```
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

```
predict_test= regressor.predict(X_val)
mean_squared_error(y_val, predict_test, squared=False)
r2_score(y_val,predict_test)
```

```
0.7034630510343927
```

Display The Comparison Lists

```
for i in Algorithm, MSE_Score, R2_Score:
  print(i,end=',')
```

```
['LinearRegression', 'Support Vector Regression', 'DecisionTreeRegression', 'RandomForestRegression', 'XGB Regressor'],[283079398483.57697, 420267116021.6393, 19539
```

You need to make the comparison list into a comparison dataframe

```
Comparison = pd.DataFrame(list(zip(Algorithm, MSE_Score, R2_Score)), columns = ['Algorithm', 'MSE_Score', 'R2_Score'])
Comparison
```

	Algorithm	MSE_Score	R2_Score
0	LinearRegression	2.830794e+11	0.321334
1	Support Vector Regression	4.202671e+11	-0.007566
2	DecisionTreeRegression	1.953961e+11	0.531549
3	RandomForestRegression	1.128528e+11	0.729442
4	XGB Regressor	1.031899e+11	0.752608

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

```
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

```
mean_squared_error(y_test,predictedfinal,squared=False)
```

```
321231.89206060674
```

Calculate the mean Square Error for validation dataset

```
mean_squared_error(y_val,predict_testfinal,squared=False)
```

```
372223.4759621151
```

Calculate the R2 score for test

```
r2_score(y_test, predictedfinal)
```

```
0.7526082208946752
```

Calculate the R2 score for Validation

```
r2_score(y_val,predict_testfinal)
```

```
0.7034630510343927
```

Calculate the Accuracy for train Dataset

```
regressorfinal.score(X_train,y_train)*100
```

```
95.45501927693235
```

Calculate the accuracy for validation

```
regressorfinal.score(X_val,y_val)*100
```

```
70.34630510343926
```

Calculate the accuracy for test

```
regressorfinal.score(X_test,y_test)*100
```

```
75.26082208946751
```

Specify the reason behind choosing your machine learning model

The best model for regression is the one which have least Mean Squared Error and a large R2 Score .From all the above models the Extreme Gradient Boost Regression gave us the least Mean Squared Error and the considerably large R2 Score with respect to other model we have gone for. Hence we opt for the Extreme Gradient Boost Regression for better Performance of the model.

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

```
display(nulldata)
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	cust_
3	CA02	0140105686	SYSC llc	NaT	2020.0	2.960623e+09	2020-03-30	2020-04-10	2020-03-31	
7	U001	0200744019	TARG us	NaT	2020.0	1.930659e+09	2020-03-19	2020-04-03	2020-03-19	
10	U001	0200418007	AM	NaT	2020.0	1.930611e+09	2020-03-11	2020-03-26	2020-03-11	
14	U001	0200739534	OK systems	NaT	2020.0	1.930788e+09	2020-04-15	2020-04-30	2020-04-15	
15	U001	0200353024	DECA corporation	NaT	2020.0	1.930817e+09	2020-04-23	2020-04-26	2020-04-16	
...	...	...	...	...	...	...	...	...	...	
49975	U001	0200769623	WAL-MAR in	NaT	2020.0	1.930625e+09	2020-03-10	2020-03-25	2020-03-10	
49980	U001	0200769623	WAL-MAR corporation	NaT	2020.0	1.930851e+09	2020-05-03	2020-05-18	2020-05-03	
49982	U001	0200148860	DOLLA co	NaT	2020.0	1.930638e+09	2020-03-11	2020-03-26	2020-03-11	
49992	U001	0200900909	SYSCO co	NaT	2020.0	1.930702e+09	2020-03-25	2020-04-09	2020-03-25	
49995	U001	0200561861	CO corporation	NaT	2020.0	1.930797e+09	2020-04-21	2020-05-06	2020-04-21	

Check for the number of rows and columns in the nulldata

```
print("number of rows : ",len(nulldata))
print("number of columns : ",len(nulldata.columns))
```

```
number of rows :  9681
number of columns :  11
```

Check the Description and Information of the nulldata

```
nulldata.describe()
```

	buisness_year	doc_id	converted_usd
count	9681.0	9.681000e+03	9681.000000
mean	2020.0	2.006165e+09	32065.681125
std	0.0	2.673629e+08	35419.613688
min	2020.0	1.930535e+09	0.720000
25%	2020.0	1.930658e+09	5607.190000
50%	2020.0	1.930731e+09	19024.190000
75%	2020.0	1.930818e+09	47752.640000
max	2020.0	2.960636e+09	653644.800000

```
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

```
nulldata1=nulldata.copy(deep=True)
display(nulldata1)
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	cust_
3	CA02	0140105686	SYSC llc	NaT	2020.0	2.960623e+09	2020-03-30	2020-04-10	2020-03-31	
7	U001	0200744019	TARG us	NaT	2020.0	1.930659e+09	2020-03-19	2020-04-03	2020-03-19	
10	U001	0200418007	AM	NaT	2020.0	1.930611e+09	2020-03-11	2020-03-26	2020-03-11	
14	U001	0200739534	OK systems	NaT	2020.0	1.930788e+09	2020-04-15	2020-04-30	2020-04-15	
15	U001	0200353024	DECA corporation	NaT	2020.0	1.930817e+09	2020-04-23	2020-04-26	2020-04-16	
...	...	...	...	...	...	...	...	...	...	
49975	U001	0200769623	WAL-MAR in	NaT	2020.0	1.930625e+09	2020-03-10	2020-03-25	2020-03-10	
49980	U001	0200769623	WAL-MAR corporation	NaT	2020.0	1.930851e+09	2020-05-03	2020-05-18	2020-05-03	
49982	U001	0200148860	DOLLA co	NaT	2020.0	1.930638e+09	2020-03-11	2020-03-26	2020-03-11	
49992	U001	0200900909	SYSCO co	NaT	2020.0	1.930702e+09	2020-03-25	2020-04-09	2020-03-25	
49995	U001	0200561861	CO corporation	NaT	2020.0	1.930797e+09	2020-04-21	2020-05-06	2020-04-21	

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

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

Now you need to manually replacing str values with numbers

- Note - No need to change the code

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

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

```
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'
- 'name_customer' and store into 'name_customer_enc'

Note - No need to change the code

```
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

```
nulldata.dtypes
```

```
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"

```
nulldata.drop(["business_code","baseline_create_date","due_in_date","posting_date","name_customer","clear_date","cust_payment_terms","day_of_cleardate","month_of_cleardate"
```

Check the information of the "nulldata" dataframe

```
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

```
X_test.info()
# nulldata.compare(X_test) #ValueError: Can only compare identically-labeled DataFrame objects
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7832 entries, 27958 to 185
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

```
X_test.columns
```

```
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')
```

```
nulldata.columns
```

```
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')
```

```
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

```
display(nulldata2)
```

	cust_number	buisness_year	doc_id	converted_usd	business_code_enc	name_customer_enc	cust_payment_terms_enc	day_of_postingda
3	140105686	2020.0	2.960623e+09	2309.79	67	2707	5	
7	200744019	2020.0	1.930659e+09	11173.02	67	2792	20	
10	200418007	2020.0	1.930611e+09	3525.59	67	95	20	

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

```
final_result=regressorfinal.predict(nulldata2)
```

49975	200769623	2020.0	1.930625e+09	13114.99	67	2999	33	

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

- Note - No need to change the code

```
final_result = pd.Series(final_result,name='avg_delay')
```

9681 rows × 16 columns

Display the "avg_delay" column

```
final_result.to_frame()
```

	avg_delay
0	8.195107e+05
1	4.044711e+05
2	1.973854e+06
3	2.066468e+05
4	-1.732631e+05
...	...

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

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

9681 rows x 1 columns

Display the "Final" dataframe

```
display(Final)
```

	key_0	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date
0	3	CA02	0140105686	SYSC llc	NaT	2020.0	2.960623e+09	2020-03-30	2020-04-10	2020-03-31
1	7	U001	0200744019	TARG us	NaT	2020.0	1.930659e+09	2020-03-19	2020-04-03	2020-03-19
2	10	U001	0200418007	AM	NaT	2020.0	1.930611e+09	2020-03-11	2020-03-26	2020-03-11
3	14	U001	0200739534	OK systems	NaT	2020.0	1.930788e+09	2020-04-15	2020-04-30	2020-04-15
4	15	U001	0200353024	DECA corporation	NaT	2020.0	1.930817e+09	2020-04-23	2020-04-26	2020-04-16

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

```
print("Number of rows in Final Dataframe : ",len(Final))
print("Number of columns in Final Dataframe : ",len(Final.columns))
```

```
Number of rows in Final Dataframe :  9681
Number of columns in Final Dataframe :  13
```

9680	49995	U001	0200561861	CO corporation	NaT	2020.0	1.930797e+09	2020-04-21	2020-05-06	2020-04-21

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

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

Display the "clear_date" column

```
Final["clear_date"]
```

```
0      2020-04-19 11:38:30.687500
1      2020-04-07 16:21:11.125000
2      2020-04-17 20:17:33.625000
3      2020-05-02 09:24:06.843750
4      2020-04-23 23:52:16.921875
                  ...
9676   2020-03-31 11:40:53.875000
9677   2020-05-24 12:10:53.312500
9678   2020-03-22 08:03:40.218750
9679   2020-04-13 02:58:20.937500
9680   2020-05-09 04:13:32.812500
Name: clear_date, Length: 9681, dtype: datetime64[ns]
```

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

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

Display the "avg_delay" column

```
Final["avg_delay"]
```

```
0       9.0
1       4.0
2      22.0
3       2.0
4      -3.0
        ...
9676    6.0
```

```
9677    6.0
9678   -4.0
9679    4.0
9680    3.0
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

```
bins= [0,15,30,45,60,100]
labels =['0-15','16-30','31-45','46-60','Greater 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

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

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

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

```
0-15        8259
16-30        358
31-45         66
```

```
Greater than 60       3
46-60                 1
Name: Aging Bucket, dtype: int64
```

Display your final dataset with aging buckets

```
display(Final)
```

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	posting_date	due_in_date	baseline_create_date	c
0	CA02	0140105686	SYSC llc	2020-04-19 11:38:30.687500	2020.0	2.960623e+09	2020-03-30	2020-04-10	2020-03-31	
1	U001	0200744019	TARG us	2020-04-07 16:21:11.125000	2020.0	1.930659e+09	2020-03-19	2020-04-03	2020-03-19	
2	U001	0200418007	AM	2020-04-17 20:17:33.625000	2020.0	1.930611e+09	2020-03-11	2020-03-26	2020-03-11	
3	U001	0200739534	OK systems	2020-05-02 09:24:06.843750	2020.0	1.930788e+09	2020-04-15	2020-04-30	2020-04-15	
4	U001	0200353024	DECA corporation	2020-04-23 23:52:16.921875	2020.0	1.930817e+09	2020-04-23	2020-04-26	2020-04-16	
...	...	...	...	...	...	...	...	...	...	
9676	U001	0200769623	WAL-MAR in	2020-03-31 11:40:53.875000	2020.0	1.930625e+09	2020-03-10	2020-03-25	2020-03-10	
9677	U001	0200769623	WAL-MAR corporation	2020-05-24 12:10:53.312500	2020.0	1.930851e+09	2020-05-03	2020-05-18	2020-05-03	

Store this dataframe into the .csv format

```
import base64
from IPython.display import HTML
def create_download_link(df,title="Download CSV File",filename="PaymentPredicted.csv"):
    csv=df.to_csv()
    b64=base64.b64encode(csv.encode())
    payload=b64.decode()
    html='<a download="{filename}" href="data:text/csv;base64,{payload}" target="_blank">{title}</a>'
    html=html.format(payload=payload,title=title,filename=filename)
    return HTML(html)
create_download_link(Final)
```

Download CSV File

END OF THE PROJECT