Payment Date Prediction

Importing related Libraries

In [1]:

```
#importing genral libraries required
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_theme(style="darkgrid")
from fast_ml.feature_selection import get_constant_features
from sklearn.model_selection import train_test_split
```

Store the dataset into the Dataframe

In [2]:

```
#read dataset from csv file and displaying the dataset
data = pd.read_csv(r"C:\Users\KIIT\Desktop\Highradius Internship Training\Project\dataset.c
data
```

Out[2]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id
0	U001	0200769623	WAL-MAR corp	2020-02-11 00:00:00	2020.0	1.930438e+09
1	U001	0200980828	BEN E	2019-08-08 00:00:00	2019.0	1.929646e+09
2	U001	0200792734	MDV/ trust	2019-12-30 00:00:00	2019.0	1.929874e+09
3	CA02	0140105686	SYSC llc	NaN	2020.0	2.960623e+09
4	U001	0200769623	WAL-MAR foundation	2019-11-25 00:00:00	2019.0	1.930148e+09
...	...	...	...	...	...	...
49995	U001	0200561861	CO corporation	NaN	2020.0	1.930797e+09
49996	U001	0200769623	WAL-MAR co	2019-09-03 00:00:00	2019.0	1.929744e+09
49997	U001	0200772595	SAFEW associates	2020-03-05 00:00:00	2020.0	1.930537e+09
49998	U001	0200726979	BJ'S llc	2019-12-12 00:00:00	2019.0	1.930199e+09
49999	U001	0200020431	DEC corp	2019-01-15 00:00:00	2019.0	1.928576e+09

50000 rows × 19 columns

Check the shape of the dataframe

In [3]:

```
#showing number of rows and columns (number of rows, number of columns)
data.shape
```

Out[3]:

```
(50000, 19)
```

Check the Detail information of the dataframe

In [4]:

```
#details of dataset
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]:

```
#showing names of all columns
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', 'isOpe
n'],
      dtype='object')
```

Describe the entire dataset

In [6]:

```
#describing dataset
data.describe
```

Out[6]:

```
<bound method NDFrame.describe of       business_code cust_number       name
_customer           clear_date  \
0              U001  0200769623        WAL-MAR corp  2020-02-11 00:00:00
1              U001  0200980828              BEN E   2019-08-08 00:00:00
2              U001  0200792734          MDV/ trust  2019-12-30 00:00:00
3              CA02  0140105686            SYSC llc                  NaN
4              U001  0200769623  WAL-MAR foundation  2019-11-25 00:00:00
...             ...         ...                 ...                  ...
49995          U001  0200561861      CO corporation                  NaN
49996          U001  0200769623          WAL-MAR co  2019-09-03 00:00:00
49997          U001  0200772595    SAFEW associates  2020-03-05 00:00:00
49998          U001  0200726979           BJ'S  llc  2019-12-12 00:00:00
49999          U001  0200020431            DEC corp  2019-01-15 00:00:00

       buisness_year        doc_id posting_date  document_create_date  \
0             2020.0  1.930438e+09   2020-01-26              20200125
1             2019.0  1.929646e+09   2019-07-22              20190722
2             2019.0  1.929874e+09   2019-09-14              20190914
3             2020.0  2.960623e+09   2020-03-30              20200330
4             2019.0  1.930148e+09   2019-11-13              20191113
...              ...           ...          ...                   ...
49995         2020.0  1.930797e+09   2020-04-21              20200417
49996         2019.0  1.929744e+09   2019-08-15              20190814
49997         2020.0  1.930537e+09   2020-02-19              20200218
49998         2019.0  1.930199e+09   2019-11-27              20191126
49999         2019.0  1.928576e+09   2019-01-05              20190105

       document_create_date.1  due_in_date invoice_currency document type  \
0                    20200126   20200210.0              USD            RV
1                    20190722   20190811.0              USD            RV
2                    20190914   20190929.0              USD            RV
3                    20200330   20200410.0              CAD            RV
4                    20191113   20191128.0              USD            RV
...                       ...          ...              ...           ...
49995                20200421   20200506.0              USD            RV
49996                20190815   20190830.0              USD            RV
49997                20200219   20200305.0              USD            RV
49998                20191127   20191212.0              USD            RV
49999                20190105   20190124.0              USD            RV

       posting_id  area_business  total_open_amount  baseline_create_date  \
0             1.0            NaN           54273.28            20200126.0
1             1.0            NaN           79656.60            20190722.0
2             1.0            NaN            2253.86            20190914.0
3             1.0            NaN            3299.70            20200331.0
4             1.0            NaN           33133.29            20191113.0
...           ...            ...                ...                   ...
49995         1.0            NaN            3187.86            20200421.0
49996         1.0            NaN            6766.54            20190815.0
49997         1.0            NaN            6120.86            20200219.0
49998         1.0            NaN              63.48            20191127.0
49999         1.0            NaN            1790.30            20190101.0
```

```
      cust_payment_terms    invoice_id  isOpen
0                   NAH4  1.930438e+09       0
1                   NAD1  1.929646e+09       0
2                   NAA8  1.929874e+09       0
3                   CA10  2.960623e+09       1
4                   NAH4  1.930148e+09       0
...                  ...           ...     ...
49995               NAA8  1.930797e+09       1
49996               NAH4  1.929744e+09       0
49997               NAA8  1.930537e+09       0
49998               NAA8  1.930199e+09       0
49999               NAM4  1.928576e+09       0

[50000 rows x 19 columns]>
```

In [7]:

```
#showing basic statistical details of dataset
data.describe()
```

Out[7]:

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

Data Cleaning

- Show top 5 records from the dataset

In [8]:

```
#showing first 5 rows
data.head()
```

Out[8]:

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

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

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

In [9]:

```
#checking if there any null values present or not
total = data.isnull().sum().sort_values(ascending=False) #(total = number of null values pr
percent = (data.isnull().mean()*100).sort_values(ascending=False) #(percent = null values p
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data
```

Out[9]:

	Total	Percent
area_business	50000	100.000
clear_date	10000	20.000
invoice_id	6	0.012
business_code	0	0.000
invoice_currency	0	0.000
cust_payment_terms	0	0.000
baseline_create_date	0	0.000
total_open_amount	0	0.000
posting_id	0	0.000
document type	0	0.000
due_in_date	0	0.000
cust_number	0	0.000
document_create_date.1	0	0.000
document_create_date	0	0.000
posting_date	0	0.000
doc_id	0	0.000
buisness_year	0	0.000
name_customer	0	0.000
isOpen	0	0.000

Display Invoice_id and Doc_Id

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

In [10]:

```
#displaying invoice_id and doc_id columns
data[['invoice_id', 'doc_id']]
```

Out[10]:

	invoice_id	doc_id
0	1.930438e+09	1.930438e+09
1	1.929646e+09	1.929646e+09
2	1.929874e+09	1.929874e+09
3	2.960623e+09	2.960623e+09
4	1.930148e+09	1.930148e+09
...	...	...
49995	1.930797e+09	1.930797e+09
49996	1.929744e+09	1.929744e+09
49997	1.930537e+09	1.930537e+09
49998	1.930199e+09	1.930199e+09
49999	1.928576e+09	1.928576e+09

50000 rows × 2 columns

In [11]:

```
#checking same values present in both invoice_id and doc_id
num = (data['invoice_id'] == data['doc_id']).sum()
perc = (data['invoice_id'] == data['doc_id']).sum()*100/len(data.axes[0])
print(f"The number of same values between these two columns is {num} and percentage of simi
```

```
The number of same values between these two columns is 49994 and percentage
of similarity is 99.988%
```

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 [12]:

```
#checking same values present in both baseline_create_date and document_create_date
num = (data['baseline_create_date'] == data['document_create_date']).sum()
perc = (data['baseline_create_date'] == data['document_create_date']).sum()*100/len(data.ax
print(f"The number of same values between these two columns is {num} and percentage of simi
```

```
The number of same values between these two columns is 15963 and percentage
of similarity is 31.926%
```

In [13]:

```
#checking same values present in both baseline_create_date and document_create_date.1
num = (data['baseline_create_date'] == data['document_create_date.1']).sum()
perc = (data['baseline_create_date'] == data['document_create_date.1']).sum()*100/len(data.
print(f"The number of same values between these two columns is {num} and percentage of simi
```

```
The number of same values between these two columns is 44452 and percentage
of similarity is 88.904%
```

In [14]:

```
#checking same values present in both document_create_date and document_create_date.1
num = (data['document_create_date'] == data['document_create_date.1']).sum()
perc = (data['document_create_date'] == data['document_create_date.1']).sum()*100/len(data.
print(f"The number of same values between these two columns is {num} and percentage of simi
```

```
The number of same values between these two columns is 21232 and percentage
of similarity is 42.464%
```

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

In [15]:

```
#checking for number of unique elements present in this column
data['posting_id'].nunique()
```

Out[15]:

```
1
```

In [16]:

```
#counting of unique values
data['posting_id'].value_counts()
```

Out[16]:

```
1.0    50000
Name: posting_id, dtype: int64
```

In [17]:

```
#checking the columns with constant and quasi-constant variable
get_constant_features(data)
```

Out[17]:

	Desc	Var	Value	Perc
0	Constant	posting_id	1.0	100.000
1	Constant	area_business	NaN	100.000
2	Quasi Constant	document type	RV	99.988

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

In [18]:

```
#checking for number of unique elements present in this column
data['isOpen'].nunique()
```

Out[18]:

```
2
```

In [19]:

```
#counting of unique values
data['isOpen'].value_counts()
```

Out[19]:

```
0    40000
1    10000
Name: isOpen, dtype: int64
```

In [20]:

```
#checking if the target column's no. of null values and no. of 1's in 'isOpen' is equal or
data['clear_date'].isnull().sum() == (data['isOpen'] == 1).sum()
```

Out[20]:

```
True
```

In [21]:

```
#checking if the target column's no. of not null values and no. of 0's in 'isOpen' is equal
data['clear_date'].notnull().sum() == (data['isOpen'] == 0).sum()
```

Out[21]:

```
True
```

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 [22]:

```
data.drop(columns=['area_business', 'posting_id', 'invoice_id', 'document_create_date', 'is
```

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

In [23]:

```
data.columns
```

Out[23]:

```
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')
```

In [24]:

```
data.head()
```

Out[24]:

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

In [25]:

```
data.shape
```

Out[25]:

```
(50000, 12)
```

Show all the Duplicate rows from the dataframe

In [26]:

```
#checking for duplicated values
data[data.duplicated()]
#showing duplicated rows
#duplicate_data
```

Out[26]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id
1041	U001	0200769623	WAL-MAR in	2019-03-12 00:00:00	2019.0	1.928870e+09
2400	U001	0200769623	WAL-MAR trust	2019-08-28 00:00:00	2019.0	1.929758e+09
2584	U001	0200769623	WAL-MAR corporation	2019-12-16 00:00:00	2019.0	1.930217e+09
3755	U001	0200769623	WAL-MAR	2019-11-22 00:00:00	2019.0	1.930137e+09
3873	CA02	0140104409	LOB associates	NaN	2020.0	2.960629e+09
...	...	...	...	...	...	...
49928	U001	0200915438	GROC trust	2019-08-15 00:00:00	2019.0	1.929646e+09
49963	U001	0200759878	SA us	2019-01-29 00:00:00	2019.0	1.928614e+09
49986	U001	0200772670	ASSOCIAT foundation	2019-06-12 00:00:00	2019.0	1.929403e+09
49990	U001	0200765011	MAINES llc	2019-06-06 00:00:00	2019.0	1.929365e+09
49991	U001	0200704045	RA trust	2019-10-25 00:00:00	2019.0	1.930001e+09

1161 rows × 12 columns

Display the Number of Duplicate Rows

In [27]:

```
data.duplicated().sum()
```

Out[27]:

1161

Drop all the Duplicate Rows

In [28]:

```
data.drop_duplicates(inplace=True)
data.head()
```

Out[28]:

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

Now check for all duplicate rows now

- Note - It must be 0 by now

In [29]:

```
data.duplicated().sum()
```

Out[29]:

```
0
```

Check for the number of Rows and Columns in your dataset

In [30]:

```
data.shape
```

Out[30]:

```
(48839, 12)
```

Find out the total count of null values in each columns

In [31]:

```
data.isnull().sum().sort_values(ascending=False)
```

Out[31]:

```
clear_date             9681
business_code             0
cust_number               0
name_customer             0
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 [32]:

```
data.dtypes
```

Out[32]:

```
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 [33]:

```
data[['clear_date', 'posting_date', 'due_in_date', 'baseline_create_date']].dtypes
```

Out[33]:

```
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**

In [34]:

```
#converting object and float format into date format
data['clear_date'] = pd.to_datetime(data['clear_date'], format = '%Y%m%d', infer_datetime_f
data['posting_date'] = pd.to_datetime(data['posting_date'], format = '%Y%m%d', infer_dateti
data['due_in_date'] = pd.to_datetime(data['due_in_date'], format = '%Y%m%d', infer_datetime
data['baseline_create_date'] = pd.to_datetime(data['baseline_create_date'], format = '%Y%m%
```

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

In [35]:

```
data[['clear_date', 'posting_date', 'due_in_date', 'baseline_create_date']].dtypes
```

Out[35]:

```
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

In [36]:

```
data['invoice_currency'].value_counts()
```

Out[36]:

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

display the "total_open_amount" column value

In [37]:

```
data['total_open_amount']
```

Out[37]:

```
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 [38]:

```
data['converted_usd'] = data['total_open_amount']
#converting the 'total_open_amount' from CAD to USD, using 1 CAD = 0.7 USD
data.loc[data['invoice_currency'] == 'CAD', 'converted_usd'] = 0.7 * data['converted_usd']
#rounding the amount to two decimal places
data['converted_usd'] = round(data['converted_usd'],2)
```

Display the new "converted_usd" column values

In [39]:

```
data['converted_usd']
```

Out[39]:

```
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

In [40]:

```
data['buisness_year'].value_counts()
```

Out[40]:

```
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 [41]:

```
data.drop(columns=['invoice_currency', 'total_open_amount'], inplace=True)
```

Write a code to check the number of columns in dataframe

In [42]:

```
ncol = len(data.columns)
ncol
```

Out[42]:

```
11
```

Splitting the Dataset

Look for all columns containing null value

- Note - Output expected is only one column

In [43]:

```
data.isnull().sum() > 0
```

Out[43]:

```
business_code           False
cust_number             False
name_customer           False
clear_date               True
buisness_year           False
doc_id                  False
posting_date            False
due_in_date             False
baseline_create_date    False
cust_payment_terms      False
converted_usd           False
dtype: bool
```

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

In [44]:

```
data['clear_date'].isnull().sum()
```

Out[44]:

```
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)

In [45]:

```
#dividing into train and test dataset on the basis of 'clear_date'
train = data[data['clear_date'].notnull()].copy()
test = data[data['clear_date'].isnull()].copy()
```

Check the number of Rows and Columns for both the dataframes

In [46]:

```
train.shape
```

Out[46]:

```
(39158, 11)
```

In [47]:

```
test.shape
```

Out[47]:

```
(9681, 11)
```

Display the 5 records from maindata and nulldata dataframes

In [48]:

```
train.head()
```

Out[48]:

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

In [49]:

```
test.head()
```

Out[49]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id	po
3	CA02	0140105686	SYSC llc	NaT	2020.0	2.960623e+09	:
7	U001	0200744019	TARG us	NaT	2020.0	1.930659e+09	:
10	U001	0200418007	AM	NaT	2020.0	1.930611e+09	:
14	U001	0200739534	OK systems	NaT	2020.0	1.930788e+09	:
15	U001	0200353024	DECA corporation	NaT	2020.0	1.930817e+09	:

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 [50]:

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

Out[50]:

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

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 respect 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 [51]:

```
train['name_customer'].nunique()
```

Out[51]:

```
3889
```

In [52]:

```
avgdelay = train.groupby('name_customer')['Delay'].mean(numeric_only=False)
avgdelay
```

Out[52]:

```
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 [53]:

```
train['avg_delay'] = train['name_customer'].map(avgdelay)
train.head()
```

Out[53]:

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

In [54]:

```
train.loc[train['name_customer'] == '11078 us']
```

Out[54]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id
5177	CA02	0100054234	11078 us	2019-05-02	2019.0	2.960539e+09

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 [55]:

```
train['avg_delay'].dtypes
```

Out[55]:

```
dtype('<m8[ns]')
```

In [56]:

```
train['avg_delay'] = train['avg_delay'].dt.total_seconds()
```

Display the maindata dataframe

In [57]:

```
train
```

Out[57]:

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

39158 rows × 13 columns

In [58]:

```
train.loc[train['name_customer'] == '11078 us']
```

Out[58]:

	business_code	cust_number	name_customer	clear_date	buisness_year	doc_id
5177	CA02	0100054234	11078 us	2019-05-02	2019.0	2.960539e+09

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 [59]:

```
train.drop(columns = ['Delay', 'clear_date'], inplace=True)
train.head()
```

Out[59]:

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

In [60]:

```
train.shape
```

Out[60]:

```
(39158, 11)
```

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 [61]:

```
Y = train[['avg_delay']].copy()
Y.shape
```

Out[61]:

```
(39158, 1)
```

In [62]:

```
X = train.copy()
X.drop(columns = 'avg_delay', inplace=True)
X.shape
```

Out[62]:

```
(39158, 10)
```

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 [63]:

```
X_train, X_loc_test, y_train, y_loc_test = train_test_split(X, Y, test_size=.40, shuffle=Fa
```

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

In [64]:

```
X_train.shape
```

Out[64]:

```
(23494, 10)
```

In [65]:

```
X_loc_test.shape
```

Out[65]:

```
(15664, 10)
```

In [66]:

```
y_train.shape
```

Out[66]:

```
(23494, 1)
```

In [67]:

```
y_loc_test.shape
```

Out[67]:

```
(15664, 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

In [68]:

```
X_val, X_test, y_val, y_test = train_test_split(X_loc_test, y_loc_test, test_size=.50, shuf
```

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

In [69]:

```
X_val.shape
```

Out[69]:

```
(7832, 10)
```

In [70]:

```
X_test.shape
```

Out[70]:

```
(7832, 10)
```

In [71]:

```
y_val.shape
```

Out[71]:

```
(7832, 1)
```

In [72]:

```
y_test.shape
```

Out[72]:

```
(7832, 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

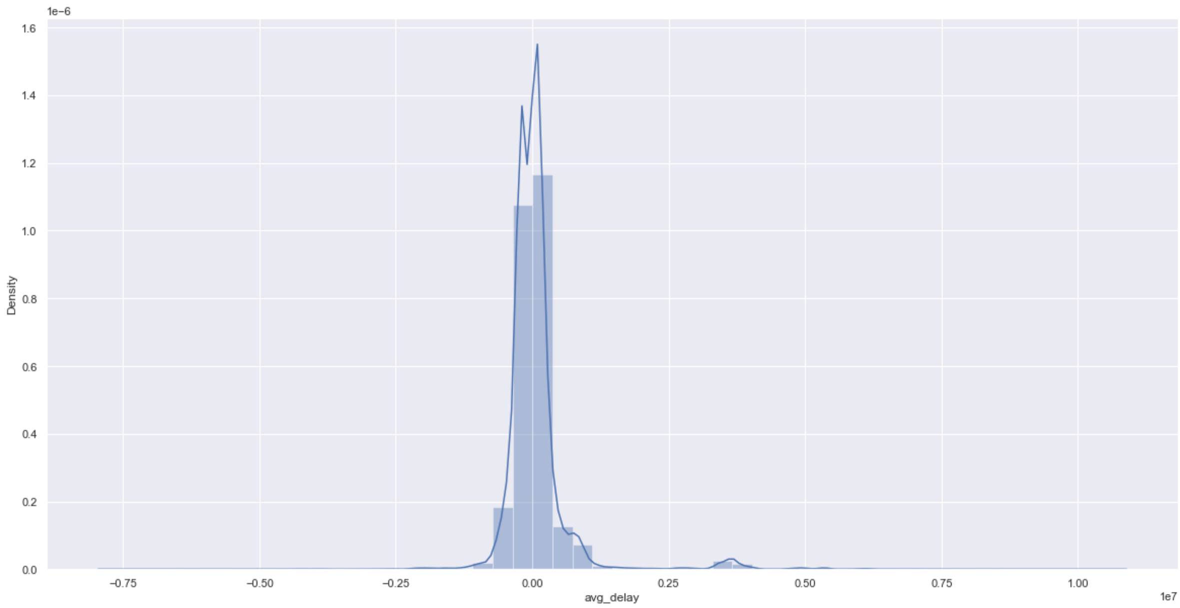
In [73]:

```
plt.subplots(figsize=(20,10))
sns.distplot(y_train['avg_delay'])
```

```
C:\Users\KIIT\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a fut
ure version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

Out[73]:

```
<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 [74]:

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

Out[74]:

	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

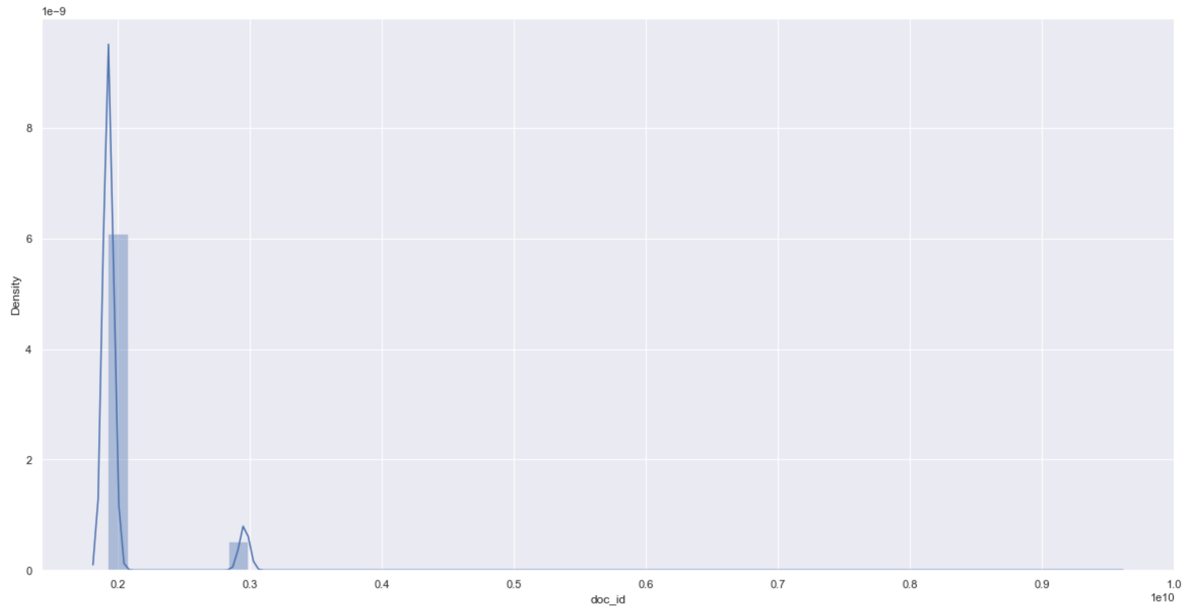
3088 rows × 2 columns

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

In [75]:

```
plt.subplots(figsize=(20,10))
res = sns.distplot(X_train['doc_id'])
plt.show()
```

```
C:\Users\KIIT\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a fut
ure version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

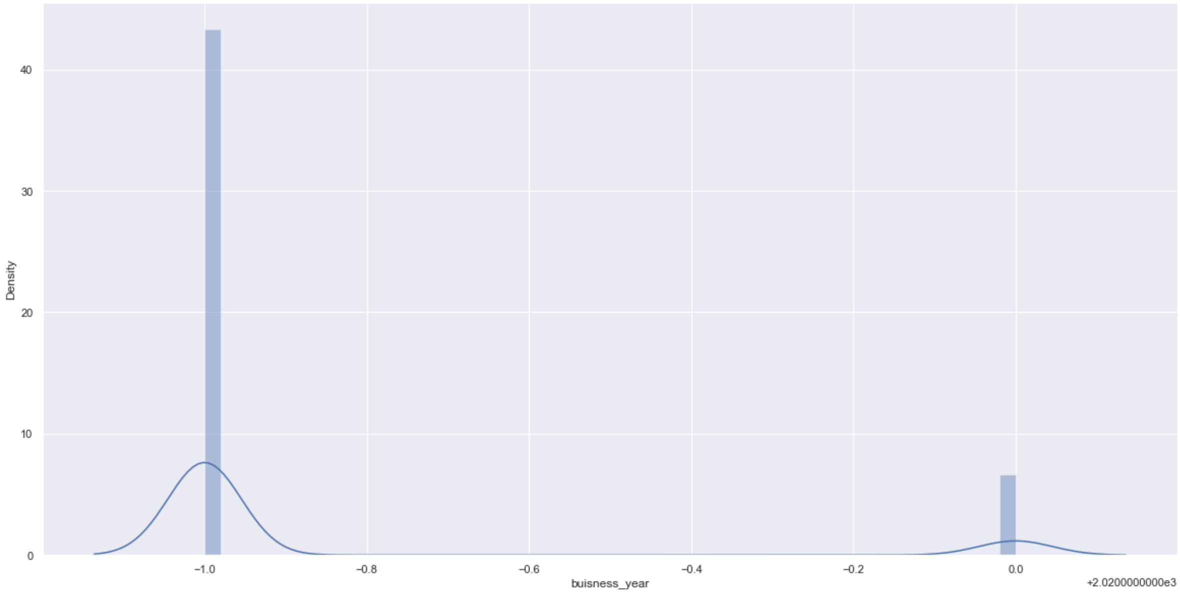


Create a Distribution plot only for business_year and a seperate distribution plot of "business_year" column along with the doc_id" column

In [76]:

```
plt.subplots(figsize=(20,10))
res = sns.distplot(X_train['buisness_year'])
plt.show()
```

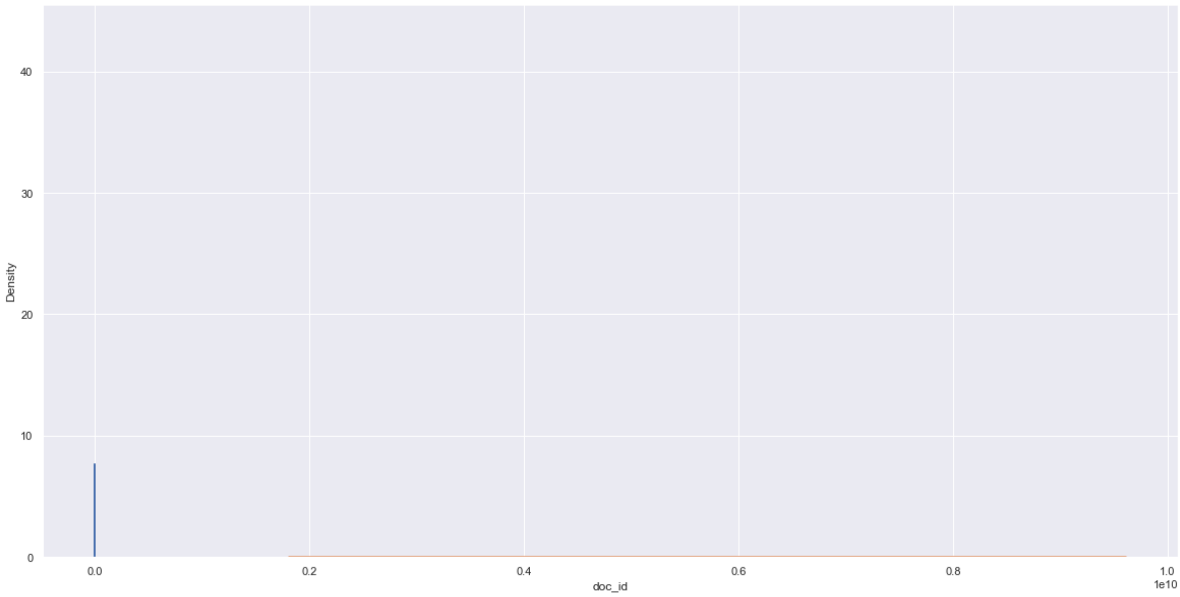
```
C:\Users\KIIT\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a fut
ure version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



In [77]:

```
plt.subplots(figsize=(20,10))
for c in ['buisness_year', 'doc_id']:
    sns.distplot(X_train[c])
```

```
C:\Users\KIIT\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a fut
ure version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\KIIT\anaconda3\lib\site-packages\seaborn\distributions.py:2619: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a fut
ure version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



Feature Engineering

Display and describe the X_train dataframe

In [ ]:

```

```

In [ ]:

```

```

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 [78]:

```
from sklearn.preprocessing import LabelEncoder
business_coder = LabelEncoder()
business_coder.___(X_train[__________])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_1880/3313907676.py in <module>
      1 from sklearn.preprocessing import LabelEncoder
      2 business_coder = LabelEncoder()
----> 3 business_coder.___(X_train[__________])

AttributeError: 'LabelEncoder' object has no attribute '___'
```

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 [ ]:

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

In [ ]:

```
X_val['business_code_enc'] = business_coder.__________(X_val['business_code'])
X_test['business_code_enc'] = business_coder.____________(X_test['business_code'])
```

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

In [ ]:

```

```

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 [ ]:

```
def ________(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 [ ]:

```
________ , ______ , _______ = ______(['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 [ ]:

```
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 [ ]:

```
#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 [ ]:

```
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 [ ]:

```
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 [ ]:

```
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 [ ]:

```
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 [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

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 [ ]:

```
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 [ ]:

```
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 [ ]:

```

```

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

In [ ]:

```

```

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 [ ]:

```

```

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

In [ ]:

```

```

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 [ ]:

```

```

Feature Selection

Filter Method

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

In [ ]:

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

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

In [ ]:

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

- 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 [ ]:

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

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 [ ]:

```
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 [ ]:

```
train=X_train.copy()
____________________(train.copy(),____)
```

Heatmap for X_train

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

In [ ]:

```
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)
```

Calling variance threshold for threshold value = 0.8

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

In [ ]:

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

In [ ]:

```
sel.variances_
```

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

In [ ]:

```
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 [ ]:

```
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

In [ ]:

```
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 [ ]:

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

Display The Comparison Lists

In [ ]:

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

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 [ ]:

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 [ ]:

Check the same for the Validation set also

In [ ]:

Display The Comparison Lists

In [ ]:

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 [ ]:

```

```

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 [ ]:

```

```

Check the same for the Validation set also

In [ ]:

```

```

Display The Comparison Lists

In [ ]:

```

```

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 [ ]:

```

```

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 [ ]:

```

```

Check the same for the Validation set also

In [ ]:

```

```

Display The Comparison Lists

In [ ]:

```

```

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 [ ]:

```
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 [ ]:

```

```

Check the same for the Validation set also

In [ ]:

```

```

Display The Comparison Lists

In [ ]:

```

```

You need to make the comparison list into a comparison dataframe

In [ ]:

```

```

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 [ ]:

```
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 [ ]:

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

Calculate the mean Square Error for validation dataset

In [ ]:

```

```

Calculate the R2 score for test

In [ ]:

```

```

Calculate the R2 score for Validation

In [ ]:

```

```

Calculate the Accuracy for train Dataset

In [ ]:

```

```

Calculate the accuracy for validation

In [ ]:

```

```

Calculate the accuracy for test

In [ ]:

```

```

Specify the reason behind choosing your machine learning model

- Note : Provide your answer as a text here

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 [ ]:

```

```

Check for the number of rows and columns in the nulldata

In [ ]:

```

```

Check the Description and Information of the nulldata

In [ ]:

```

```

Storing the Nulldata into a different dataset

for BACKUP

In [ ]:

```

```

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 [ ]:

```
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

In [ ]:

```
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 [ ]:

```

```

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

In [ ]:

```
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 [ ]:

```

```

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 [ ]:

```

```

Check the information of the "nulldata" dataframe

In [ ]:

```

```

Compare "nulldata" with the "X_test" dataframe

- use info() method

In [ ]:

```

```

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 [ ]:

```
X_test.columns
```

In [ ]:

```
nulldata.columns
```

In [ ]:

```
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 [ ]:

```

```

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

In [ ]:

```

```

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

- Note - No need to change the code

In [ ]:

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

Display the "avg_delay" column

In [ ]:

```

```

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

In [ ]:

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

Display the "Final" dataframe

In [ ]:

```

```

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

In [ ]:

```

```

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 [ ]:

```
Final['clear_date'] = pd.to_datetime(Final['due_in_date']) + pd.to_timedelta(Final['avg_del
```

Display the "clear_date" column

In [ ]:

```

```

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

In [ ]:

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

Display the "avg_delay" column

In [ ]:

```

```

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 [ ]:

```
bins= ____________________
labels =___________________________
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 [ ]:

```

```

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

In [ ]:

```

```

Display your final dataset with aging buckets

In [ ]:

```

```

Store this dataframe into the .csv format

In [ ]:

```

```

END OF THE PROJECT