Table of Contents

Feature engineering

Transactional Data

Data loading

In [273…

```
# IMPORT THE NECESSARY LIBRARY FOR ANALYSIS

import sys
# scipy # for statistics
import scipy
from sklearn.cluster import KMeans
# numpy for array, matrix and vector calculations
import numpy as np
# matplotlib for graphs
import pandas as pd
# scikit-learn for machine learning
import sklearn

# Load  specialised libraries
from pandas.plotting import scatter_matrix
import matplotlib.pyplot
import tkinter
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
plt.style.use('ggplot') # ggplot style
%matplotlib inline
import seaborn as sns #lightly better visuals than matplot
# model selection
from sklearn import model_selection
# kpi: evaulating the performance of the model
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```python
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.metrics import fbeta_score
# the stars of the show: the models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier()
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.ensemble import ExtraTreesClassifier # Extra Trees
from warnings import simplefilter
#ignore warnings
simplefilter(action= 'ignore',category =FutureWarning)
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import RobustScaler
import joblib  # to store the final model
from sklearn.model_selection import validation_curve,StratifiedKFold,GridSearchCV # for tu
from sklearn.feature_selection import SelectKBest,chi2 # k best and chi
from sklearn.feature_selection import RFE # Recursive feature elimination
from sklearn.svm import LinearSVC #linear svc for L1 feature selection
from functools import reduce # to merge data frame
from statsmodels.stats.outliers_influence import variance_inflation_factor # mulicollinea
from datetime import datetime, date
from xverse.transformer import WOE
from xverse.transformer import MonotonicBinning
from xverse.ensemble import VotingSelector
#from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
from sklearn.neural_network import MLPClassifier
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
from sklearn.model_selection import train_test_split # To randomly split the dataset
from datetime import timedelta # To define potential churn date
# for market basket analysis
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import squarify
```

In [274…

```python
# connecting our database to get data necessary for analysis
import pyodbc #importing python database
```

In [275…

```python
# connect to SQL

# establish an open connection to SQL
conn = pyodbc.connect('Driver={SQL Server};'
'Server=DESKTOP-1VJ4H95\MSSQLSERVER01;'
'Database=AdventureWorksDW2017;'
'Trusted_Connection=yes;')
```

In [276…

```python
# pull the data tranasational data

# pull the necessary fields for analysis from SQL (AdventureWorksDW2017 Database)
# plug your SQL query inside the """ """
```

```
transaction_df = pd.read_sql_query("""
SELECT
 FIS.[CustomerKey] AS Customer_id
 ,FIS.OrderDate AS Order_date
 ,FIS.[SalesOrderNumber] AS Sales_order_number
 ,FIS.SalesOrderLineNumber AS Sales_order_line_number
 ,DP.EnglishProductName AS Product
 ,FIS.OrderQuantity AS Quantity
 ,FIS.SalesAmount AS Revenue
 ,FIS.TotalProductCost AS Cost
FROM
[dbo].[FactInternetSales] AS FIS
LEFT JOIN [dbo].[DimProduct] AS DP
ON FIS.ProductKey = DP.ProductKey
""", conn)

conn.close() # please close it after
```

exploratory data analysis

In [277…

```
#let have a look at the data
transaction_df.head()
```

Out[277…

	Customer_id	Order_date	Sales_order_number	Sales_order_line_number	Product	Quantity	Revenue	Cost
0	21768	2010-12-29	SO43697	1	Road-150 Red, 62	1	3578.27	2171.29
1	28389	2010-12-29	SO43698	1	Mountain-100 Silver, 44	1	3399.99	1912.15
2	25863	2010-12-29	SO43699	1	Mountain-100 Silver, 44	1	3399.99	1912.15
3	14501	2010-12-29	SO43700	1	Road-650 Black, 62	1	699.10	413.15
4	11003	2010-12-29	SO43701	1	Mountain-100 Silver, 44	1	3399.99	1912.15

In [278…

```
# create a new column profit
transaction_df['Profit'] = transaction_df['Revenue']-transaction_df['Cost']
transaction_df.head()
```

Out[278…

	Customer_id	Order_date	Sales_order_number	Sales_order_line_number	Product	Quantity	Revenue	Cost	
0	21768	2010-12-29	SO43697	1	Road-150 Red, 62	1	3578.27	2171.29	1
1	28389	2010-12-29	SO43698	1	Mountain-100 Silver, 44	1	3399.99	1912.15	1
2	25863	2010-12-29	SO43699	1	Mountain-100 Silver, 44	1	3399.99	1912.15	1
3	14501	2010-12-29	SO43700	1	Road-650 Black, 62	1	699.10	413.15	

	Customer_id	Order_date	Sales_order_number	Sales_order_line_number	Product	Quantity	Revenue	Cost
4	11003	2010-12-29	SO43701	1	Mountain-100 Silver, 44	1	3399.99	1912.15	1

In [279…

```
# convert quantity to float to be use later on for building ml model
transaction_df['Quantity']=transaction_df['Quantity'].astype(float)
#check to see if it had been implemented
transaction_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60398 entries, 0 to 60397
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Customer_id              60398 non-null  int64
 1   Order_date               60398 non-null  datetime64[ns]
 2   Sales_order_number       60398 non-null  object
 3   Sales_order_line_number  60398 non-null  int64
 4   Product                  60398 non-null  object
 5   Quantity                 60398 non-null  float64
 6   Revenue                  60398 non-null  float64
 7   Cost                     60398 non-null  float64
 8   Profit                   60398 non-null  float64
dtypes: datetime64[ns](1), float64(4), int64(2), object(2)
memory usage: 4.1+ MB
```

In [280…

```
# check for duplicate values
print('Number of duplicates is:',transaction_df.duplicated().sum())
```

```
Number of duplicates is: 0
```

Data transformation

In [281…

```
# let sum up the data per transaction
trans_df= transaction_df.groupby(['Customer_id','Sales_order_number','Order_date']).agg({'


trans_df.head()
```

Out[281…

	Customer_id	Sales_order_number	Order_date	Quantity	Revenue	Profit
0	11000	SO43793	2011-01-19	1.00	3399.99	1487.84
1	11000	SO51522	2013-01-18	2.00	2341.97	1068.13
2	11000	SO57418	2013-05-03	5.00	2507.03	957.72
3	11001	SO43767	2011-01-15	1.00	3374.99	1476.90
4	11001	SO51493	2013-01-16	6.00	2419.93	1091.99

In [282…

```
#let get days elapsed period between transaction for every customer
trans_df['Days_elapsed'] =trans_df.groupby('Customer_id')['Order_date'].diff()
trans_df['Days_elapsed']= trans_df['Days_elapsed']/np.timedelta64(1,'D')
trans_df.head()
```

Out[282…

	Customer_id	Sales_order_number	Order_date	Quantity	Revenue	Profit	Days_elapsed
0	11000	SO43793	2011-01-19	1.00	3399.99	1487.84	NaN

	Customer_id	Sales_order_number	Order_date	Quantity	Revenue	Profit	Days_elapsed
1	11000	SO51522	2013-01-18	2.00	2341.97	1068.13	730.00
2	11000	SO57418	2013-05-03	5.00	2507.03	957.72	105.00
3	11001	SO43767	2011-01-15	1.00	3374.99	1476.90	NaN
4	11001	SO51493	2013-01-16	6.00	2419.93	1091.99	732.00

In [283…

```
# after grouping check to see if the siZe has treduced from 60000 because we only have 18

trans_df.shape
```

Out[283…

```
(27659, 7)
```

In [284…

```
# agregate data per customer
aggs=['sum','mean','median','min','max']
trans_per_customer = trans_df.groupby(['Customer_id','Sales_order_number','Order_date']).a
                                                              'Profit':aggs,'Days_elapsed':agg

trans_per_customer.head()
```

Out[284…

	Customer_id	Sales_order_number	Order_date	Quantity					Revenue		...	
				sum	mean	median	min	max	sum	mean	...	sum
0	11000	SO43793	2011-01-19	1.00	1.00	1.00	1.00	1.00	3399.99	3399.99	...	1487.84
1	11000	SO51522	2013-01-18	2.00	2.00	2.00	2.00	2.00	2341.97	2341.97	...	1068.13
2	11000	SO57418	2013-05-03	5.00	5.00	5.00	5.00	5.00	2507.03	2507.03	...	957.72
3	11001	SO43767	2011-01-15	1.00	1.00	1.00	1.00	1.00	3374.99	3374.99	...	1476.90
4	11001	SO51493	2013-01-16	6.00	6.00	6.00	6.00	6.00	2419.93	2419.93	...	1091.99

5 rows × 23 columns

In [285…

```
trans_per_customer.shape
```

Out[285…

```
(27659, 23)
```

In [286…

```
#### here we can see from above that the customer 11000 puchased items at three different
```

In [287…

```
# agregate data per customer,remove order date and 'Sales_order_number' from agregate
aggs=['sum','mean','median','min','max']
trans_per_customer = trans_df.groupby('Customer_id').agg({'Quantity':aggs,'Revenue':aggs,
                                                              'Profit':aggs,'Days_elapsed':agg

trans_per_customer.head()
```

Out[287…

	Customer_id	Quantity					Revenue				...			
		sum	mean	median	min	max	sum	mean	median	min	...	sum	mean	median
0	11000	8.00	2.67	2.00	1.00	5.00	8248.99	2749.66	2507.03	2341.97	...	3513.69	1171.23	1068.13
1	11001	11.00	3.67	4.00	1.00	6.00	6383.88	2127.96	2419.93	588.96	...	2795.88	931.96	1091.99
2	11002	4.00	1.33	1.00	1.00	2.00	8114.04	2704.68	2419.06	2294.99	...	3454.88	1151.63	1043.01

	Customer_id	Quantity					Revenue				...			
		sum	mean	median	min	max	sum	mean	median	min	...	sum	mean	median
3	11003	9.00	3.00	4.00	1.00	4.00	8139.29	2713.10	2420.34	2318.96	...	3467.13	1155.71	1054.45
4	11004	6.00	2.00	2.00	1.00	3.00	8196.01	2732.00	2419.06	2376.96	...	3501.91	1167.30	1090.03

5 rows × 21 columns

In [288…

```
# we now have 18448 customers,single view per customer
trans_per_customer.shape
```

Out[288…

```
(18484, 21)
```

In [289…

```
#drop the customer_id
trans=trans_per_customer.drop('Customer_id',axis =1)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py:4153: PerformanceWarnin
g: dropping on a non-lexsorted multi-index without a level parameter may impact performanc
e.
  obj = obj._drop_axis(labels, axis, level=level, errors=errors)
```

In [290…

```
#now let us join to flatten the column

trans.columns =["_".join(trans) for trans in trans.columns.ravel()]
#check if it had been implemented
trans.head()
```

Out[290…

	Quantity_sum	Quantity_mean	Quantity_median	Quantity_min	Quantity_max	Revenue_sum	Revenue_mean	Reve
0	8.00	2.67	2.00	1.00	5.00	8248.99	2749.66	
1	11.00	3.67	4.00	1.00	6.00	6383.88	2127.96	
2	4.00	1.33	1.00	1.00	2.00	8114.04	2704.68	
3	9.00	3.00	4.00	1.00	4.00	8139.29	2713.10	
4	6.00	2.00	2.00	1.00	3.00	8196.01	2732.00	

In [291…

```
#add customer_id back to the tables
trans.insert(0,'Customer_id',trans_per_customer['Customer_id'])
trans.head()
```

Out[291…

	Customer_id	Quantity_sum	Quantity_mean	Quantity_median	Quantity_min	Quantity_max	Revenue_sum	Revenu
0	11000	8.00	2.67	2.00	1.00	5.00	8248.99	
1	11001	11.00	3.67	4.00	1.00	6.00	6383.88	
2	11002	4.00	1.33	1.00	1.00	2.00	8114.04	
3	11003	9.00	3.00	4.00	1.00	4.00	8139.29	
4	11004	6.00	2.00	2.00	1.00	3.00	8196.01	

5 rows × 21 columns

Behavioral Data

RFM Analysis

In [292…

```
# calculate RFM metrics
#filter the necessary column
rfm =trans_df[['Customer_id','Order_date','Revenue']]
rfm.head()
```

Out[292…

	Customer_id	Order_date	Revenue
0	11000	2011-01-19	3399.99
1	11000	2013-01-18	2341.97
2	11000	2013-05-03	2507.03
3	11001	2011-01-15	3374.99
4	11001	2013-01-16	2419.93

In [293…

```
# 1) sort the data by CustomerID and Order_date
rfm = rfm.sort_values(["Customer_id","Order_date"])
rfm.head()
```

Out[293…

	Customer_id	Order_date	Revenue
0	11000	2011-01-19	3399.99
1	11000	2013-01-18	2341.97
2	11000	2013-05-03	2507.03
3	11001	2011-01-15	3374.99
4	11001	2013-01-16	2419.93

In [294…

```
# group by customer id
rfm= rfm.groupby("Customer_id").agg({ "Order_date": "max","Revenue":["count","sum"]})
rfm.head()
```

Out[294…

	Order_date	Revenue	
	max	count	sum
Customer_id			
11000	2013-05-03	3	8248.99
11001	2013-12-10	3	6383.88
11002	2013-02-23	3	8114.04
11003	2013-05-10	3	8139.29
11004	2013-05-01	3	8196.01

In [295…

```
#Tidy Up
#we need to merge the column headers
# flatten column headers
```

```
rfm.columns = ["_".join(rfm) for rfm in rfm.columns.ravel()]
rfm.head()
```

Out[295...

	Order_date_max	Revenue_count	Revenue_sum
Customer_id			
11000	2013-05-03	3	8248.99
11001	2013-12-10	3	6383.88
11002	2013-02-23	3	8114.04
11003	2013-05-10	3	8139.29
11004	2013-05-01	3	8196.01

In [296...

```
### we need to make some changes as the recency parameter (Order_date_max) is shown a date
```

In [297...

```
# find out last transaction in our data
lastTransaction = max(rfm.Order_date_max)
lastTransaction
```

Out[297...

```
Timestamp('2014-01-28 00:00:00')
```

In [298...

```
# now lets create a column for the duration since last purchase and remove days stamp fron
rfm["DaysElapsed"] = (lastTransaction - rfm["Order_date_max"])/np.timedelta64(1,'D')
rfm.head()
```

Out[298...

	Order_date_max	Revenue_count	Revenue_sum	DaysElapsed
Customer_id				
11000	2013-05-03	3	8248.99	270.00
11001	2013-12-10	3	6383.88	49.00
11002	2013-02-23	3	8114.04	339.00
11003	2013-05-10	3	8139.29	263.00
11004	2013-05-01	3	8196.01	272.00

In [299...

```
# 1) drop the Order_date_max column
rfm.drop("Order_date_max", axis = 1, inplace = True) # axis =1 is columns
# have a look
rfm.head()
```

Out[299...

	Revenue_count	Revenue_sum	DaysElapsed
Customer_id			
11000	3	8248.99	270.00
11001	3	6383.88	49.00
11002	3	8114.04	339.00
11003	3	8139.29	263.00
11004	3	8196.01	272.00

In [300…

```
# 2) rename the columns: Recency (R), Frequency (F) and Monetary Value (M)
rfm.rename(columns={"Revenue_count": "Frequency",
                    "Revenue_sum": "Monetary",
                    "DaysElapsed": "Recency"}, inplace = True)
rfm.head()
```

Out[300…

	Frequency	Monetary	Recency
Customer_id			
11000	3	8248.99	270.00
11001	3	6383.88	49.00
11002	3	8114.04	339.00
11003	3	8139.29	263.00
11004	3	8196.01	272.00

In [301…

```
#now let us rearranged the order of our data frame to be in the form of R F M
rfm = rfm[['Recency','Frequency','Monetary']]
rfm.head()
```

Out[301…

	Recency	Frequency	Monetary
Customer_id			
11000	270.00	3	8248.99
11001	49.00	3	6383.88
11002	339.00	3	8114.04
11003	263.00	3	8139.29
11004	272.00	3	8196.01

In [302…

```
#### lets put these customers into bins. We will categorize each customers into quartiles
```

In [303…

```
# Add score to rmf tables
quantiles = rfm.quantile(q=[0.25,0.5,0.75])
quantiles = quantiles.to_dict()
quantiles
```

Out[303…

```
{'Recency': {0.25: 86.0, 0.5: 168.0, 0.75: 263.0},
 'Frequency': {0.25: 1.0, 0.5: 1.0, 0.75: 2.0},
 'Monetary': {0.25: 49.97, 0.5: 270.26500000000004, 0.75: 2511.275}}
```

In [304…

```
#### What this means is- lets take Recency: for 0.25 quantile we have 86. So this says tha
```

In [305…

```
# now let us create score on a scale of  1 to 4 with 1 being the best and 4 the worst for
# because we are looking for the least numbers of days since a customer has done business
def Rscore(x,p,d):
    '''
    create scores for Recency ,values in the first percntile are the best scores(1)
    '''
    if x <= d[p][0.25]:
        return 1
```

```
    elif x <= d[p][0.5]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4
```

In [306…

```
# We will do exactly opposite for Frequency and monetary value as we want those values to
def FMscore(x,p,d):

    '''
    values in the first percentile are the worst scores(4)
    '''
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.50]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1
```

In [307…

```
# add segement
rfm['R'] = rfm['Recency'].apply(Rscore,args = ('Recency', quantiles))
rfm['F'] = rfm['Frequency'].apply(FMscore,args = ('Frequency',quantiles))
rfm['M'] = rfm['Monetary'].apply(FMscore,args = ('Monetary', quantiles))
rfm.head()
```

Out[307…

	Recency	Frequency	Monetary	R	F	M
Customer_id						
11000	270.00	3	8248.99	4	1	1
11001	49.00	3	6383.88	1	1	1
11002	339.00	3	8114.04	4	1	1
11003	263.00	3	8139.29	3	1	1
11004	272.00	3	8196.01	4	1	1

In [308…

```
#Add and combined RFM segement
rfm['RFM_segment']= rfm.R.map(str)+ rfm.F.map(str) + rfm.M.map(str)
rfm['RFM_score'] = rfm[['R','F','M']].sum(axis = 1)
rfm.head()
```

Out[308…

	Recency	Frequency	Monetary	R	F	M	RFM_segment	RFM_score
Customer_id								
11000	270.00	3	8248.99	4	1	1	411	6
11001	49.00	3	6383.88	1	1	1	111	3
11002	339.00	3	8114.04	4	1	1	411	6
11003	263.00	3	8139.29	3	1	1	311	5
11004	272.00	3	8196.01	4	1	1	411	6

In [309…

```python
# now let us assign label from total score
Score_labels = ['Gold','Silver','Bronze','Green']
Score_groups = pd.qcut(rfm.RFM_score,q=4,labels=Score_labels)
rfm['RFM_status'] = Score_groups.values
rfm.head()
```

Out[309…

	Recency	Frequency	Monetary	R	F	M	RFM_segment	RFM_score	RFM_status
Customer_id									
11000	270.00	3	8248.99	4	1	1	411	6	Gold
11001	49.00	3	6383.88	1	1	1	111	3	Gold
11002	339.00	3	8114.04	4	1	1	411	6	Gold
11003	263.00	3	8139.29	3	1	1	311	5	Gold
11004	272.00	3	8196.01	4	1	1	411	6	Gold

In [310…

```python
# get insight into label(quantile)
quantiles = rfm['RFM_score'].quantile(q=[0.25,0.5,0.75])
quantiles = quantiles.to_dict()
quantiles
```

Out[310…

```
{0.25: 6.0, 0.5: 9.0, 0.75: 10.0}
```

In [311…

```python
#let retrieve the % of our most valuable customer,those ones in 111 segment
print('The share of 111 segment is {:.2f}%'.format(len(rfm[rfm['RFM_segment']=='111'])/ler
```

```
The share of 111 segment is 1.77%
```

In [312…

```python
#peek at the valuable customer
valuable_customer=rfm[rfm['RFM_segment']=='111']
valuable_customer.head()
```

Out[312…

	Recency	Frequency	Monetary	R	F	M	RFM_segment	RFM_score	RFM_status
Customer_id									
11001	49.00	3	6383.88	1	1	1	111	3	Gold
11029	78.00	3	6565.29	1	1	1	111	3	Gold
11030	80.00	3	6471.32	1	1	1	111	3	Gold
11031	76.00	3	6478.60	1	1	1	111	3	Gold
11032	81.00	3	6525.56	1	1	1	111	3	Gold

In [313…

```python
# now let get the number of customers represented by this percentage (1.77%)
print('The most valuable customer are just',len(rfm[rfm['RFM_segment']=='111']),'in number
```

```
The most valuable customer are just 328 in numbers
```

In [314…

```python
# now remove R F M columns
cols=['R','F','M']
rfm.drop(cols,axis =1, inplace = True)
# put customer_id back to the frame
```

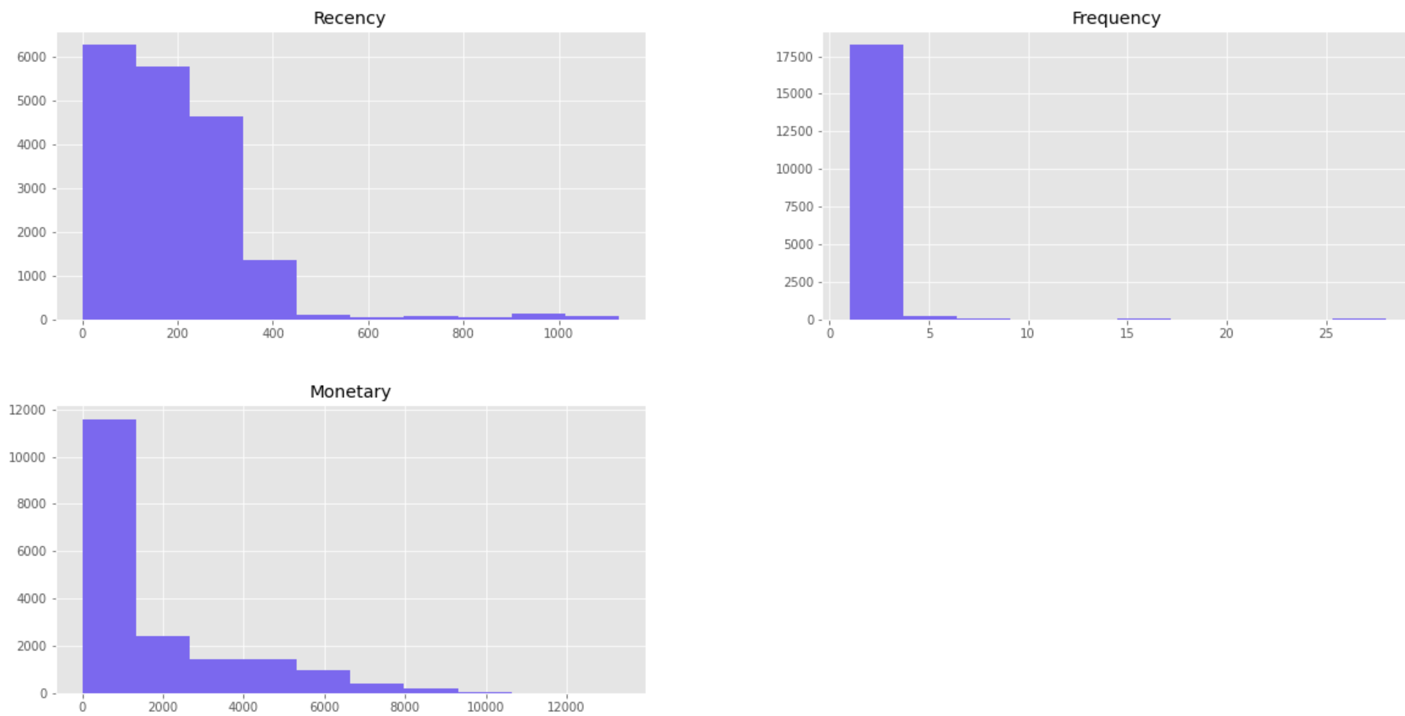
```python
rfm =rfm.reset_index()
rfm.head()
```

Out[314…

	Customer_id	Recency	Frequency	Monetary	RFM_segment	RFM_score	RFM_status
0	11000	270.00	3	8248.99	411	6	Gold
1	11001	49.00	3	6383.88	111	3	Gold
2	11002	339.00	3	8114.04	411	6	Gold
3	11003	263.00	3	8139.29	311	5	Gold
4	11004	272.00	3	8196.01	411	6	Gold

In [315…

```python
# plot the graph and see the distribution of the data
rfm.iloc[:,1:4].hist(figsize=(20,10),color = 'mediumslateblue')
pass
```



In [316…

```python
#let get the statistical summary of the RFM
rfm.iloc[:,1:4].describe().transpose()
```

Out[316…

	count	mean	std	min	25%	50%	75%	max
Recency	18484.00	189.33	146.29	0.00	86.00	168.00	263.00	1126.00
Frequency	18484.00	1.50	1.10	1.00	1.00	1.00	2.00	28.00
Monetary	18484.00	1588.33	2124.23	2.29	49.97	270.27	2511.28	13295.38

In [317…

```python
#get the skewness of the data
rfm.iloc[:,1:4].skew()
```

Out[317…

```
Recency       2.46
Frequency    12.58
Monetary      1.41
dtype: float64
```

In [318…

```
#### If the skewness is between -0.5 and 0.5, the data are fairly symmetrical.
#### If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data are moderately
#### If the skewness is less than -1 or greater than 1, the data are highly skewed.

#### # As  we can see from the above,our data are highly skewed
```

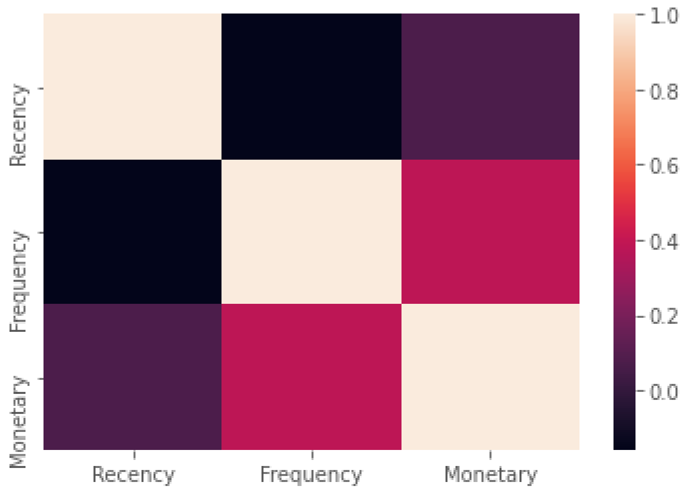
K means clusterring

In [319…

```
sns.heatmap(rfm.iloc[:,1:4].corr())
```

Out[319…

```
<AxesSubplot:>
```

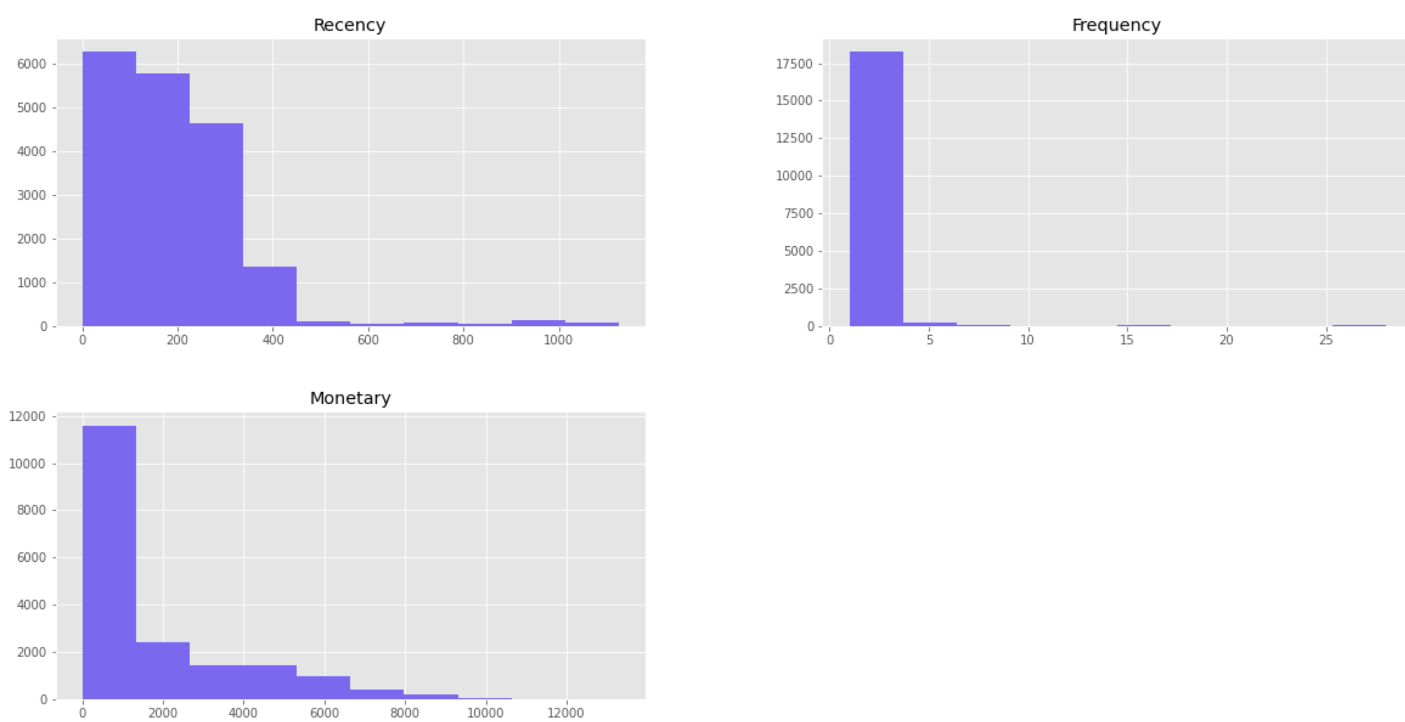


Data Normalizing.

The range of variables shows large variation. K-Means is distance based, so adjusting range common range is required to avoid building biased model.

In [320…

```
# Plot data
rfm.iloc[:,1:4].hist(figsize=(20,10), color = 'mediumslateblue')
pass
```



In [321…

```
rfm.iloc[:,1:4].describe()
```

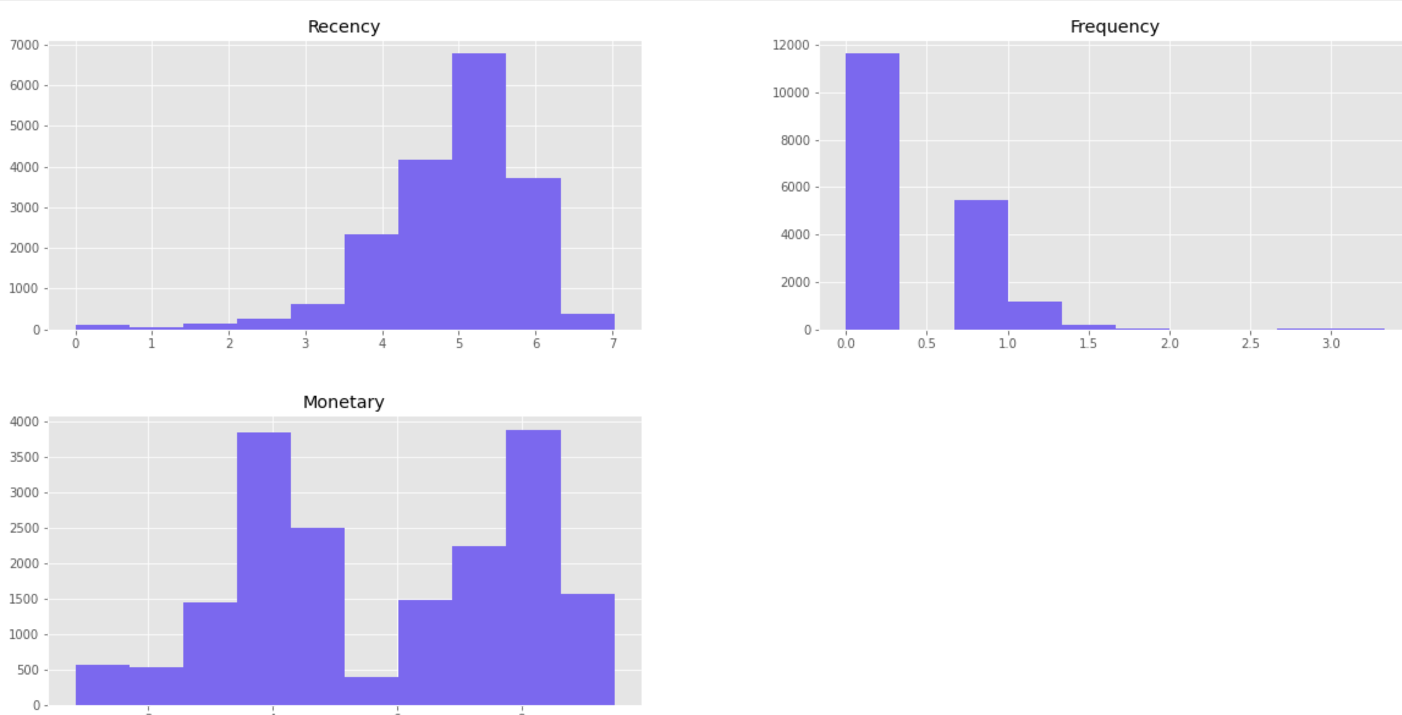
Out[321...

	Recency	Frequency	Monetary
count	18484.00	18484.00	18484.00
mean	189.33	1.50	1588.33
std	146.29	1.10	2124.23
min	0.00	1.00	2.29
25%	86.00	1.00	49.97
50%	168.00	1.00	270.27
75%	263.00	2.00	2511.28
max	1126.00	28.00	13295.38

In [ ]:

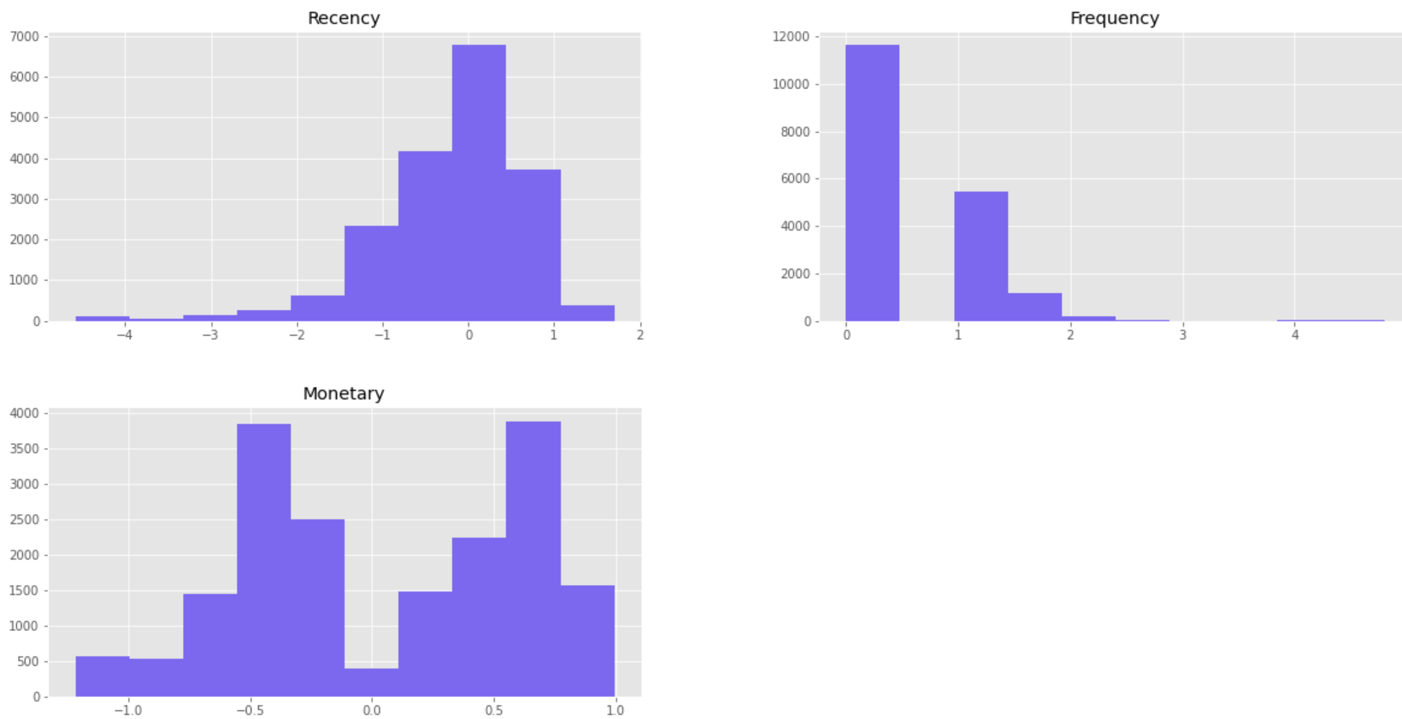
In [322...

```python
# We need to normalise and scale data for the K-means model
# The values lower than or equal to zero go negative infinite when they are in log scale
# The function below converts those values into 1
def neg_to_zero(x):
    if x <= 0:
        return 1
    else:
        return x

# Apply the function to Recency column
rfm['Recency'] = [neg_to_zero(x) for x in rfm.Recency]
# Unskew the data
rfm_log = rfm[['Recency', 'Frequency', 'Monetary']].apply(np.log, axis = 1).round(3)

# PLot logged data
rfm_log.hist(figsize=(20,10), color = 'mediumslateblue')
pass
```



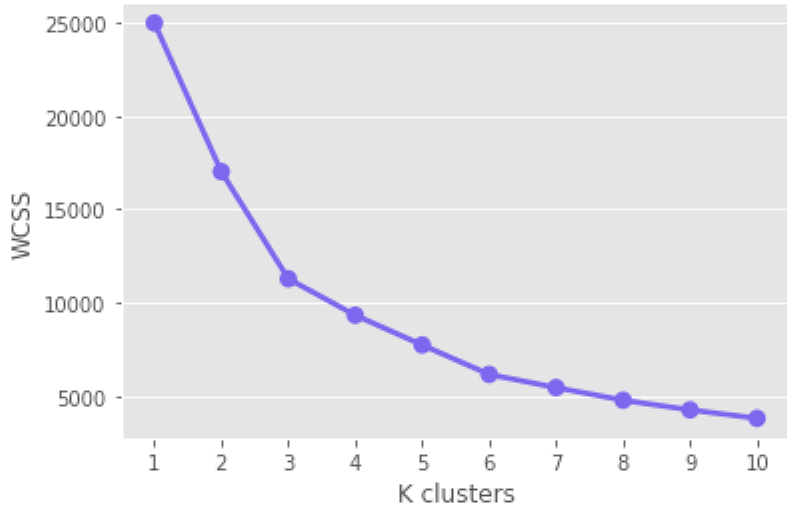
In [323…

```
# Scale data
scaler = RobustScaler()
rfm_scaled = scaler.fit_transform(rfm_log)
# Transform into a dataframe
rfm_scaled = pd.DataFrame(rfm_scaled, index = rfm.index, columns = rfm_log.columns)
# plot the graph and see the distribution of the data
rfm_scaled.hist(figsize=(20,10),color = 'mediumslateblue')
pass
```



In [324…

```
# Choose k number of clusters using the Elbow method
def elbow_plot (features):
    wcss = {}
    for k in range(1, 11):
        kmeans = KMeans(n_clusters= k, init= 'k-means++', max_iter= 300)
        kmeans.fit(features)
        wcss[k] = kmeans.inertia_

    # Plot the WCSS values
    sns.pointplot(x = list(wcss.keys()), y = list(wcss.values()), color = 'mediumslateblue
    plt.xlabel('K clusters')
    plt.ylabel('WCSS')
    plt.show()

elbow_plot(rfm_scaled)
```



In [325…

```
# We choose k = 4 where the change in Within Cluster Sum of Squares (WCSS) levels off
seed = 53
kmeans = KMeans(n_clusters= 4, init= 'k-means++', max_iter= 300, random_state = seed)
kmeans.fit(rfm_scaled)
# Assign the clusters to rfm dataframe
rfm['RFM_cluster'] = kmeans.labels_
rfm.head()
```

Out[325…

	Customer_id	Recency	Frequency	Monetary	RFM_segment	RFM_score	RFM_status	RFM_cluster
0	11000	270.00	3	8248.99	411	6	Gold	2
1	11001	49.00	3	6383.88	111	3	Gold	2
2	11002	339.00	3	8114.04	411	6	Gold	2
3	11003	263.00	3	8139.29	311	5	Gold	2
4	11004	272.00	3	8196.01	411	6	Gold	2

In [326…

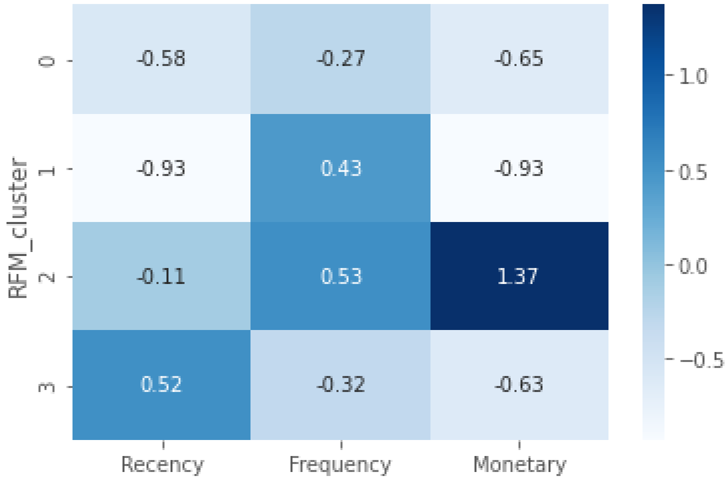
```
kmeans.labels_
```

Out[326…

```
array([2, 2, 2, ..., 3, 3, 3])
```

In [327…

```
# Visualise clusters with heatmap
# Calculate the mean value in total
total_avg = rfm.iloc[:, 1:4].mean()
total_avg

# Calculate the proportional gap with total mean
cluster_avg_K = rfm.groupby('RFM_cluster').mean().iloc[:, 1:4]
prop_rfm_K = cluster_avg_K/total_avg - 1

# Plot heatmap
sns.heatmap(prop_rfm_K, cmap= 'Blues', fmt= '.2f', annot = True)
plt.plot()
pass
```



In [328…

```
# Cluster 0 needs attention as they are almost lost! These are the biggest spenders with a
# Cluster 2 also needs attention as they are lost customers. They are irregular buyers wit
# Cluster 3 comprises of loyal customers: high frequency and very low recency but low mone
# Cluster 1 has the lowest market share and average frequency. They have also purchased re
```

In [329…

```
#### clearly those in cluster 2 are the best performing customer.
```

In [330…

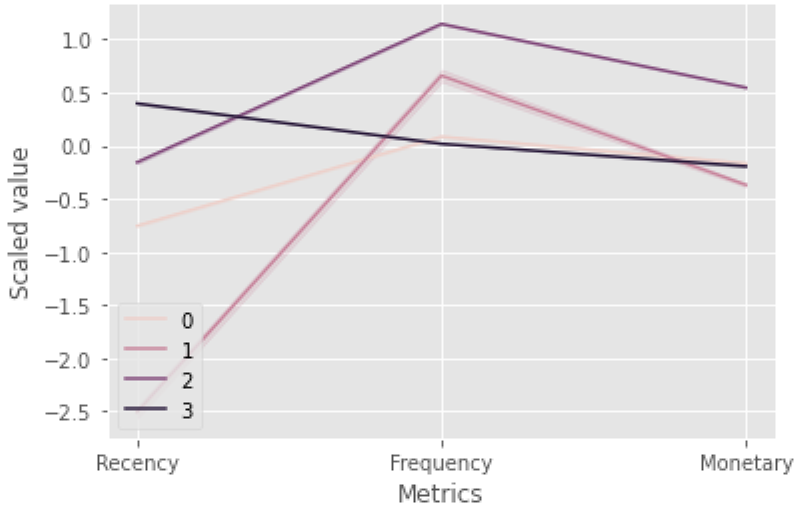
```
# Alternative way to visualise 3D data: Snake plot
# Assign cluster column
rfm_scaled['RFM_cluster'] = kmeans.labels_
rfm_scaled

# Melt the dataframe
rfm_melted = pd.melt(frame= rfm_scaled,
                     id_vars= ['RFM_cluster'],
                     var_name = 'Metrics',
                     value_name = 'Scaled value')
rfm_melted.head()
```

Out[330…

	RFM_cluster	Metrics	Scaled value
0	2	Recency	0.42
1	2	Recency	-1.10
2	2	Recency	0.63
3	2	Recency	0.40
4	2	Recency	0.43

In [331…

```
# PLot snake plot with K-Means
sns.lineplot(x = 'Metrics', y = 'Scaled value', hue = 'RFM_cluster', data = rfm_melted)
plt.legend(loc = 'lower left')
pass
```



In [332…

```
# To have a single view join rfm with taransactional data
single_view = pd.merge(trans,rfm, on =['Customer_id'])
single_view.head()
```

Out[332…

	Customer_id	Quantity_sum	Quantity_mean	Quantity_median	Quantity_min	Quantity_max	Revenue_sum	Revenu
0	11000	8.00	2.67	2.00	1.00	5.00	8248.99	
1	11001	11.00	3.67	4.00	1.00	6.00	6383.88	
2	11002	4.00	1.33	1.00	1.00	2.00	8114.04	
3	11003	9.00	3.00	4.00	1.00	4.00	8139.29	
4	11004	6.00	2.00	2.00	1.00	3.00	8196.01	

5 rows × 28 columns

In [333…

```
# check to see if total number of customer is maitained and that there is no duplicate
single_view.shape
```

Out[333…

```
(18484, 28)
```

Tenure and Churning

In [334…

```
tenure = trans_df[["Customer_id","Order_date"]]
tenure= tenure.groupby("Customer_id").agg({"Order_date":["min","max"]})
tenure.columns = ["_".join(tenure) for tenure in tenure.columns.ravel()]
tenure.head()
```

Out[334…

	Order_date_min	Order_date_max
Customer_id		
11000	2011-01-19	2013-05-03
11001	2011-01-15	2013-12-10
11002	2011-01-07	2013-02-23
11003	2010-12-29	2013-05-10
11004	2011-01-23	2013-05-01

In [335...

```
# get tenure in months
tenure['Tenure_months'] = tenure['Order_date_max'] - tenure['Order_date_min']
tenure['Tenure_months'] = tenure['Tenure_months'] /np.timedelta64(1,'M')
tenure.head()
```

Out[335...

	Order_date_min	Order_date_max	Tenure_months
Customer_id			
11000	2011-01-19	2013-05-03	27.43
11001	2011-01-15	2013-12-10	34.83
11002	2011-01-07	2013-02-23	25.56
11003	2010-12-29	2013-05-10	28.35
11004	2011-01-23	2013-05-01	27.24

In [336...

```
# now let us add recency to the tenure
tenure['Recency'] =  (lastTransaction - tenure["Order_date_max"])/np.timedelta64(1,'D')
tenure.head()
```

Out[336...

	Order_date_min	Order_date_max	Tenure_months	Recency
Customer_id				
11000	2011-01-19	2013-05-03	27.43	270.00
11001	2011-01-15	2013-12-10	34.83	49.00
11002	2011-01-07	2013-02-23	25.56	339.00
11003	2010-12-29	2013-05-10	28.35	263.00
11004	2011-01-23	2013-05-01	27.24	272.00

In [337...

```
### Churn
#### Customer whose maximum oder date(last transaction date) greater or equal to 8 month i
```

In [338...

```
churn_days =240
def churn(x):
    '''
    Determine churn or not if the recency is greater or equal to 8 months(240 days),
    the customer has churn
    '''
    if x >= churn_days:
        return 1
    else:
        return 0
# apply the function on the recency column to get churn column
tenure['Churn'] = [churn(x) for x in tenure.Recency]
```

In [339...

```
# remove some columns that are not necessary
cols =['Order_date_min','Order_date_max','Recency']
tenure.drop(cols, axis =1,inplace =True)
tenure.head()
```

Out[339...

	Tenure_months	Churn
Customer_id		
11000	27.43	1
11001	34.83	0
11002	25.56	1
11003	28.35	1
11004	27.24	1

In [340…

```
#now join the tenure backe to the transactional dataframe and RFM data frame to have a si
single_view = pd.merge(single_view,tenure, on =['Customer_id'])
single_view.head()
```

Out[340…

	Customer_id	Quantity_sum	Quantity_mean	Quantity_median	Quantity_min	Quantity_max	Revenue_sum	Revenu
0	11000	8.00	2.67	2.00	1.00	5.00	8248.99	
1	11001	11.00	3.67	4.00	1.00	6.00	6383.88	
2	11002	4.00	1.33	1.00	1.00	2.00	8114.04	
3	11003	9.00	3.00	4.00	1.00	4.00	8139.29	
4	11004	6.00	2.00	2.00	1.00	3.00	8196.01	

5 rows × 30 columns

In [341…

```
# check to see if total number of customers is maitained and that there is no duplicate
single_view.shape
```

Out[341…

```
(18484, 30)
```

Deomographic Analysis

Data loading

In [342…

```
# connect to SQL

# establish an open connection to SQL
conn = pyodbc.connect('Driver={SQL Server};'
'Server=DESKTOP-1VJ4H95\MSSQLSERVER01;'
'Database=AdventureWorksDW2017;'
'Trusted_Connection=yes;')
```

In [343…

```
# pull the DMOGRAPHIC data

# pull the necessary fields for analysis from SQL (AdventureWorksDW2017 Database)
# plug your SQL query inside the """ """
demographics = pd.read_sql_query("""
--- DEMOGRAPHIC_DATA
SELECT
DC.[CustomerKey] AS Customer_id
,DC.BirthDate AS Birth_date
,DC.MaritalStatus AS Marital_status
```

```
,DC.Gender
,DC.YearlyIncome AS Yearly_income
,DC.NumberChildrenAtHome AS Number_children_at_home
,DC.EnglishEducation AS Education
,DC.EnglishOccupation AS Ocupation

,DC.CommuteDistance AS Commute_distance
,DC.TotalChildren AS Total_children
,DC.HouseOwnerFlag AS House_ownership
,DC.NumberCarsOwned AS Car_ownership

FROM [dbo].[DimCustomer] AS DC

""", conn)

conn.close() # please close it after
```

Data exploration

In [344…

```
demographics.head()
```

Out[344…

	Customer_id	Birth_date	Marital_status	Gender	Yearly_income	Number_children_at_home	Education	Ocupation
0	11000	1971-10-06	M	M	90000.00	0	Bachelors	Professiona
1	11001	1976-05-10	S	M	60000.00	3	Bachelors	Professiona
2	11002	1971-02-09	M	M	60000.00	3	Bachelors	Professiona
3	11003	1973-08-14	S	F	70000.00	0	Bachelors	Professiona
4	11004	1979-08-05	S	F	80000.00	5	Bachelors	Professiona

In [345…

```
# check for duplicate values
print('Number of duplicates is:',demographics.duplicated().sum())
```

```
Number of duplicates is: 0
```

In [346…

```
#get info about the demographics
demographics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18484 entries, 0 to 18483
Data columns (total 12 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Customer_id              18484 non-null  int64  
 1   Birth_date               18484 non-null  object 
 2   Marital_status           18484 non-null  object 
 3   Gender                   18484 non-null  object 
 4   Yearly_income            18484 non-null  float64
 5   Number_children_at_home  18484 non-null  int64  
 6   Education                18484 non-null  object 
 7   Ocupation                18484 non-null  object 
 8   Commute_distance         18484 non-null  object 
 9   Total_children           18484 non-null  int64  
 10  House_ownership          18484 non-null  object 
```

```
 11  Car_ownership          18484 non-null  int64
dtypes: float64(1), int64(4), object(7)
memory usage: 1.7+ MB
```

In [347…

```python
# This function converts the birthday to age
def age(born):
    born = datetime.strptime(born, "%Y-%m-%d").date()
    today = date.today()
    return today.year - born.year - ((today.month,
                                      today.day) < (born.month,
                                                    born.day))

demographics['Age'] = demographics['Birth_date'].apply(age)
demographics.head()
```

Out[347…

	Customer_id	Birth_date	Marital_status	Gender	Yearly_income	Number_children_at_home	Education	Ocupation
0	11000	1971-10-06	M	M	90000.00	0	Bachelors	Professiona
1	11001	1976-05-10	S	M	60000.00	3	Bachelors	Professiona
2	11002	1971-02-09	M	M	60000.00	3	Bachelors	Professiona
3	11003	1973-08-14	S	F	70000.00	0	Bachelors	Professiona
4	11004	1979-08-05	S	F	80000.00	5	Bachelors	Professiona

In [348…

```python
#get min age to split into bins
demographics.Age.min()
```

Out[348… 35

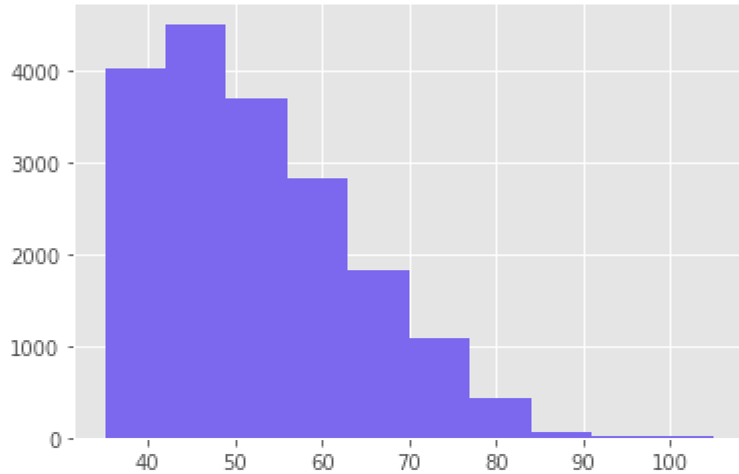
In [349…

```python
#get the max age for the bins
demographics['Age'].max()
```

Out[349… 105

In [350…

```python
# Create a new field for age group
# Show distribution to determine bins
demographics['Age'].hist(color = 'mediumslateblue')
pass
```



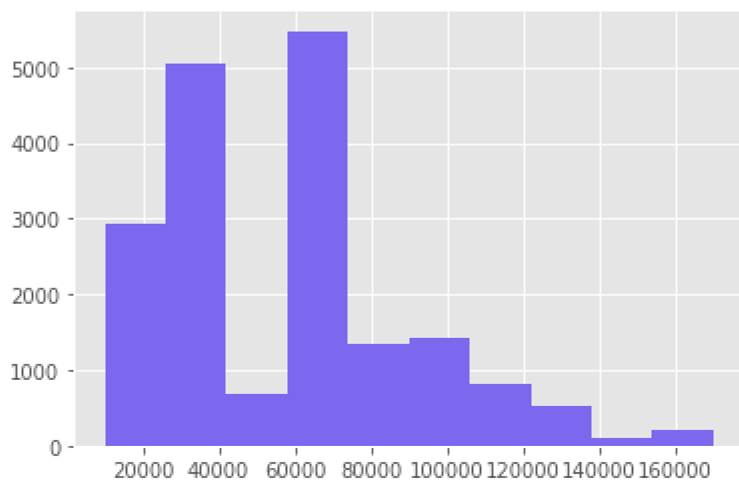
Data transformarion

In [351…

```
# Create bands
bins = [0, 30, 40, 60, np.inf]
names = ['20s', '30s','40-50s', '60s or older']
demographics['Age_group'] = pd.cut(demographics['Age'], bins, labels = names)
```

In [352…

```
# Create a new field for income group
# Show distribution to determine bins
demographics['Yearly_income'].hist(color = 'mediumslateblue')
pass
```



In [353…

```
# Create bands
bins = [0, 12000, 50000, 145000, np.inf]
names = ['Low','Lower-middle', 'Upper-middle', 'High']
demographics['Income_group']= pd.cut(demographics['Yearly_income'], bins, labels = names)
demographics.head()
```

Out[353…

	Customer_id	Birth_date	Marital_status	Gender	Yearly_income	Number_children_at_home	Education	Ocupation
0	11000	1971-10-06	M	M	90000.00	0	Bachelors	Professiona
1	11001	1976-05-10	S	M	60000.00	3	Bachelors	Professiona
2	11002	1971-02-09	M	M	60000.00	3	Bachelors	Professiona

	Customer_id	Birth_date	Marital_status	Gender	Yearly_income	Number_children_at_home	Education	Ocupation
3	11003	1973-08-14	S	F	70000.00	0	Bachelors	Professiona
4	11004	1979-08-05	S	F	80000.00	5	Bachelors	Professiona

In [354…

```
#To have a single view join rfm with taransactional data
single_view = pd.merge(single_view,demographics, on =['Customer_id'])
single_view.head()
```

Out[354…

	Customer_id	Quantity_sum	Quantity_mean	Quantity_median	Quantity_min	Quantity_max	Revenue_sum	Revenu
0	11000	8.00	2.67	2.00	1.00	5.00	8248.99	
1	11001	11.00	3.67	4.00	1.00	6.00	6383.88	
2	11002	4.00	1.33	1.00	1.00	2.00	8114.04	
3	11003	9.00	3.00	4.00	1.00	4.00	8139.29	
4	11004	6.00	2.00	2.00	1.00	3.00	8196.01	

5 rows × 44 columns

Attitudinal data

Data loading

In [355…

```
# connect to SQL

# establish an open connection to SQL
conn = pyodbc.connect('Driver={SQL Server};'
'Server=DESKTOP-1VJ4H95\MSSQLSERVER01;'
'Database=AdventureWorksDW2017;'
'Trusted_Connection=yes;')
```

In [356…

```
# pull the DMOGRAPHIC data

# pull the necessary fields for analysis from SQL (AdventureWorksDW2017 Database)
# plug your SQL query inside the """ """
sales_driver = pd.read_sql_query("""
--ATTIUDINAL DATA (Sales Reason)
SELECT
FIS.CustomerKey AS Customer_id
,FIS.SalesOrderNumber As Sales_order_number
,fis.SalesOrderLineNumber AS Sales_order_line_number

,DSR.SalesReasonReasonType AS Sales_reason_type
,DSR.SalesReasonName AS Sales_reason
FROM[dbo].[FactInternetSales] FIS

LEFT JOIN [dbo].[FactInternetSalesReason]FISR

ON FIS.SalesOrderNumber = FISR.SalesOrderNumber

LEFT JOIN [dbo].[DimSalesReason] DSR

ON FISR.SalesReasonKey = DSR.SalesReasonKey
```

```
""", conn)

conn.close() # please close it after
```

Data exploration

In [357…

```
sales_driver.head()
```

Out[357…

	Customer_id	Sales_order_number	Sales_order_line_number	Sales_reason_type	Sales_reason
0	21768	SO43697	1	Other	Manufacturer
1	21768	SO43697	1	Other	Quality
2	27645	SO43702	1	Other	Manufacturer
3	27645	SO43702	1	Other	Quality
4	16624	SO43703	1	Other	Manufacturer

In [358…

```
# check for duplicate values
print('Number of duplicates is:',sales_driver.duplicated().sum())
```

```
Number of duplicates is: 123642
```

In [359…

```
sales_driver.drop_duplicates(subset=None ,keep = 'first',inplace = True)
sales_driver.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 70944 entries, 0 to 194585
Data columns (total 5 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Customer_id              70944 non-null  int64 
 1   Sales_order_number       70944 non-null  object
 2   Sales_order_line_number  70944 non-null  int64 
 3   Sales_reason_type        64515 non-null  object
 4   Sales_reason             64515 non-null  object
dtypes: int64(2), object(3)
memory usage: 3.2+ MB
```

In [360…

```
# how many null values are in our data
sales_driver.isnull().sum()
```

Out[360…

```
Customer_id                   0
Sales_order_number            0
Sales_order_line_number       0
Sales_reason_type          6429
Sales_reason               6429
dtype: int64
```

In [361…

```
# replacing the missing values
sales_driver =sales_driver.fillna(sales_driver.mode().iloc[0])
#Check if it had been implemented
sales_driver.isnull().sum()
```

Out[361…

```
Customer_id                0
Sales_order_number         0
Sales_order_line_number    0
```

```
Sales_reason_type         0
Sales_reason              0
dtype: int64
```

Data transformation

In [362…

```
# summarise data per customer
# apply one hot encoding for Sales_reason and Sales_order_line_number
cols = ['Sales_reason_type','Sales_reason']
#sales_driver=pd.get_dummies(sales_driver)
sales_driver=pd.get_dummies(sales_driver, columns = [v for v in cols], drop_first = False)
sales_driver.head()
```

Out[362…

	Customer_id	Sales_order_number	Sales_order_line_number	Sales_reason_type_Marketing	Sales_reason_type_Other
0	21768	SO43697	1	0	1
1	21768	SO43697	1	0	1
2	27645	SO43702	1	0	1
3	27645	SO43702	1	0	1
4	16624	SO43703	1	0	1

In [363…

```
sales_driver.columns
```

Out[363…

```
Index(['Customer_id', 'Sales_order_number', 'Sales_order_line_number',
       'Sales_reason_type_Marketing', 'Sales_reason_type_Other',
       'Sales_reason_type_Promotion', 'Sales_reason_Manufacturer',
       'Sales_reason_On Promotion', 'Sales_reason_Other', 'Sales_reason_Price',
       'Sales_reason_Quality', 'Sales_reason_Review',
       'Sales_reason_Television  Advertisement'],
      dtype='object')
```

In [364…

```
# rename columns that is not proper
sales_driver = sales_driver.rename(columns={'Sales_reason_On Promotion':'Sales_reason_On_P

sales_driver.head()
```

Out[364…

	Customer_id	Sales_order_number	Sales_order_line_number	Sales_reason_type_Marketing	Sales_reason_type_Other
0	21768	SO43697	1	0	1
1	21768	SO43697	1	0	1
2	27645	SO43702	1	0	1
3	27645	SO43702	1	0	1
4	16624	SO43703	1	0	1

In [365…

```
# Agreggate to find maxinum value
sales_driver = sales_driver.groupby('Customer_id').agg({'Sales_reason_type_Marketing': 'ma
        'Sales_reason_type_Other': 'max',
       'Sales_reason_type_Promotion':'max',
       'Sales_reason_Manufacturer':'max',
       'Sales_reason_On_Promotion': 'max',
       'Sales_reason_Other': 'max',
       'Sales_reason_Price': 'max',
```

```
        'Sales_reason_Quality': 'max',
        'Sales_reason_Review' : 'max',
        'Sales_reason_Television  Advertisement': 'max'}).reset_index()
sales_driver.head()
```

Out[365…

	Customer_id	Sales_reason_type_Marketing	Sales_reason_type_Other	Sales_reason_type_Promotion	Sales_reason_Ma
0	11000	0	1	1	
1	11001	0	1	0	
2	11002	0	1	1	
3	11003	0	1	0	
4	11004	0	1	0	

In [366…

```
sales_driver.columns
```

Out[366…

```
Index(['Customer_id', 'Sales_reason_type_Marketing', 'Sales_reason_type_Other',
       'Sales_reason_type_Promotion', 'Sales_reason_Manufacturer',
       'Sales_reason_On_Promotion', 'Sales_reason_Other', 'Sales_reason_Price',
       'Sales_reason_Quality', 'Sales_reason_Review',
       'Sales_reason_Television  Advertisement'],
      dtype='object')
```

In [367…

```
# rename columns that is not proper
sales_driver = sales_driver.rename(columns={'Sales_reason_Television  Advertisement':'Sale

sales_driver.head()
```

Out[367…

	Customer_id	Sales_reason_type_Marketing	Sales_reason_type_Other	Sales_reason_type_Promotion	Sales_reason_Ma
0	11000	0	1	1	
1	11001	0	1	0	
2	11002	0	1	1	
3	11003	0	1	0	
4	11004	0	1	0	

In [368…

```
#check to see if the number of customers remains
sales_driver.shape
```

Out[368…

```
(18484, 11)
```

In [369…

```
# now merge the sales_driver table with demographic and taransactional data

single_view = pd.merge(single_view,sales_driver, on =['Customer_id'])
single_view.head().transpose()
```

Out[369…

	0	1	2	3	4
Customer_id	11000	11001	11002	11003	11004
Quantity_sum	8.00	11.00	4.00	9.00	6.00
Quantity_mean	2.67	3.67	1.33	3.00	2.00

	0	1	2	3	4
Quantity_median	2.00	4.00	1.00	4.00	2.00
Quantity_min	1.00	1.00	1.00	1.00	1.00
Quantity_max	5.00	6.00	2.00	4.00	3.00
Revenue_sum	8248.99	6383.88	8114.04	8139.29	8196.01
Revenue_mean	2749.66	2127.96	2704.68	2713.10	2732.00
Revenue_median	2507.03	2419.93	2419.06	2420.34	2419.06
Revenue_min	2341.97	588.96	2294.99	2318.96	2376.96
Revenue_max	3399.99	3374.99	3399.99	3399.99	3399.99
Profit_sum	3513.69	2795.88	3454.88	3467.13	3501.91
Profit_mean	1171.23	931.96	1151.63	1155.71	1167.30
Profit_median	1068.13	1091.99	1043.01	1054.45	1090.03
Profit_min	957.72	227.00	924.04	924.84	924.04
Profit_max	1487.84	1476.90	1487.84	1487.84	1487.84
Days_elapsed_sum	835.00	1060.00	778.00	863.00	829.00
Days_elapsed_mean	417.50	530.00	389.00	431.50	414.50
Days_elapsed_median	417.50	530.00	389.00	431.50	414.50
Days_elapsed_min	105.00	328.00	54.00	125.00	99.00
Days_elapsed_max	730.00	732.00	724.00	738.00	730.00
Recency	270.00	49.00	339.00	263.00	272.00
Frequency	3	3	3	3	3
Monetary	8248.99	6383.88	8114.04	8139.29	8196.01
RFM_segment	411	111	411	311	411
RFM_score	6	3	6	5	6
RFM_status	Gold	Gold	Gold	Gold	Gold
RFM_cluster	2	2	2	2	2
Tenure_months	27.43	34.83	25.56	28.35	27.24
Churn	1	0	1	1	1
Birth_date	1971-10-06	1976-05-10	1971-02-09	1973-08-14	1979-08-05
Marital_status	M	S	M	S	S
Gender	M	M	M	F	F
Yearly_income	90000.00	60000.00	60000.00	70000.00	80000.00
Number_children_at_home	0	3	3	0	5
Education	Bachelors	Bachelors	Bachelors	Bachelors	Bachelors
Ocupation	Professional	Professional	Professional	Professional	Professional
Commute_distance	1-2 Miles	0-1 Miles	2-5 Miles	5-10 Miles	1-2 Miles
Total_children	2	3	3	0	5

	0	1	2	3	4
House_ownership	1	0	1	0	1
Car_ownership	0	1	1	1	4
Age	49	45	50	48	42
Age_group	40-50s	40-50s	40-50s	40-50s	40-50s
Income_group	Upper-middle	Upper-middle	Upper-middle	Upper-middle	Upper-middle
Sales_reason_type_Marketing	0	0	0	0	0
Sales_reason_type_Other	1	1	1	1	1
Sales_reason_type_Promotion	1	0	1	0	0
Sales_reason_Manufacturer	0	0	0	0	0
Sales_reason_On_Promotion	1	0	1	0	0
Sales_reason_Other	0	0	0	0	0
Sales_reason_Price	1	1	1	1	1
Sales_reason_Quality	0	0	0	0	0
Sales_reason_Review	0	0	0	0	0
Sales_reason_Television_Advertisement	0	0	0	0	0

In [370…

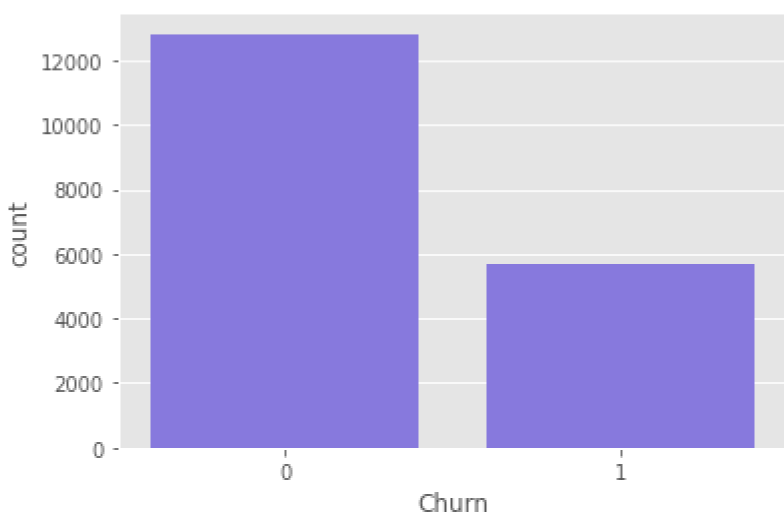
```
# keep a copy of the data
single_view_copy = single_view.copy()
#single_view_copy.head()
```

Eploratory Data Analysis

Hold out sample

In [371…

```
# PLot countplot for target variable
sns.countplot(data = single_view, x = 'Churn', color = 'mediumslateblue')
pass
```



In [372…

```
# get the distribution of churn
single_view['Churn'].value_counts()
```

Out[372...

```
0    12817
1     5667
Name: Churn, dtype: int64
```

In [373...

```
# Hold out sample for data scoring; 50% of non churners
# we would use 50 % of non churner to score our model
non_churners = single_view[single_view['Churn'] == 0]
non_churners.head()
```

Out[373...

	Customer_id	Quantity_sum	Quantity_mean	Quantity_median	Quantity_min	Quantity_max	Revenue_sum	Rever
1	11001	11.00	3.67	4.00	1.00	6.00	6383.88	
12	11012	5.00	2.50	2.50	2.00	3.00	81.26	
13	11013	5.00	2.50	2.50	2.00	3.00	113.96	
17	11017	4.00	1.33	1.00	1.00	2.00	6434.31	
18	11018	7.00	2.33	2.00	1.00	4.00	6533.28	

5 rows × 54 columns

In [374...

```
# now create the score data to represent 50% of non churner
score_pct = 0.5
model_data, score_data = train_test_split(non_churners,
                                          test_size = score_pct,
                                          random_state = seed)


len(model_data)
```

Out[374...

```
6408
```

In [375...

```
#Concat non churner model data with churner data
# our model data = model_data(50% of non_churner after splitting) + churner
churners = single_view[single_view['Churn'] == 1]
model_data = pd.concat([model_data, churners], ignore_index=True)
model_data.shape
```

Out[375...

```
(12075, 54)
```

In [376...

```
#churners.head()
# get a copy of these data to be used for MBA analysis later
mba_data = model_data.copy()
```

In [377...

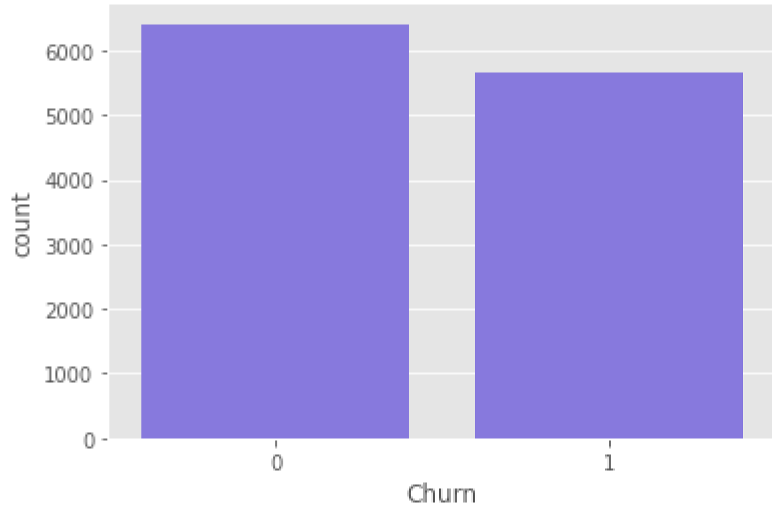
```
# check to see if we have a balance data_set
# what percentage does churner represent in the data set
100*len(churners)/len(model_data)
# almost equal to non churner.This is a balce data set there would be no need to resample
```

Out[377...

```
46.93167701863354
```

In [378...

```
# vizualize the distribution
sns.countplot(data = model_data, x = 'Churn', color = 'mediumslateblue')
pass
```



In [379…

```
model_data.head().transpose()
```

Out[379…

	0	1	2	3	4
Customer_id	14242	12378	14382	18129	24967
Quantity_sum	4.00	2.00	5.00	3.00	1.00
Quantity_mean	2.00	1.00	2.50	3.00	1.00
Quantity_median	2.00	1.00	2.50	3.00	1.00
Quantity_min	1.00	1.00	1.00	3.00	1.00
Quantity_max	3.00	1.00	4.00	3.00	1.00
Revenue_sum	3351.40	4366.41	3373.91	96.46	4.99
Revenue_mean	1675.70	2183.20	1686.95	96.46	4.99
Revenue_median	1675.70	2183.20	1686.95	96.46	4.99
Revenue_min	1000.44	2071.42	1000.44	96.46	4.99
Revenue_max	2350.96	2294.99	2373.47	96.46	4.99
Profit_sum	1464.99	1996.57	1486.93	40.59	3.12
Profit_mean	732.49	998.29	743.46	40.59	3.12
Profit_median	732.49	998.29	743.46	40.59	3.12
Profit_min	394.79	953.56	394.79	40.59	3.12
Profit_max	1070.20	1043.01	1092.14	40.59	3.12
Days_elapsed_sum	356.00	352.00	283.00	0.00	0.00
Days_elapsed_mean	356.00	352.00	283.00	NaN	NaN
Days_elapsed_median	356.00	352.00	283.00	NaN	NaN
Days_elapsed_min	356.00	352.00	283.00	NaN	NaN
Days_elapsed_max	356.00	352.00	283.00	NaN	NaN
Recency	239.00	187.00	230.00	237.00	29.00
Frequency	2	2	2	1	1
Monetary	3351.40	4366.41	3373.91	96.46	4.99

	0	1	2	3	4
RFM_segment	321	321	321	343	144
RFM_score	6	6	6	10	9
RFM_status	Gold	Gold	Gold	Bronze	Silver
RFM_cluster	2	2	2	3	0
Tenure_months	11.70	11.56	9.30	0.00	0.00
Churn	0	0	0	0	0
Birth_date	1947-11-30	1985-02-16	1955-11-03	1962-12-04	1953-08-17
Marital_status	M	S	S	M	S
Gender	M	F	M	F	F
Yearly_income	70000.00	30000.00	70000.00	100000.00	40000.00
Number_children_at_home	0	0	0	3	1
Education	Bachelors	Partial College	Bachelors	Partial College	High School
Ocupation	Management	Clerical	Management	Professional	Professional
Commute_distance	1-2 Miles	2-5 Miles	10+ Miles	5-10 Miles	10+ Miles
Total_children	5	0	5	2	2
House_ownership	1	1	1	1	1
Car_ownership	2	1	2	4	2
Age	73	36	65	58	68
Age_group	60s or older	30s	60s or older	40-50s	60s or older
Income_group	Upper-middle	Lower-middle	Upper-middle	Upper-middle	Lower-middle
Sales_reason_type_Marketing	0	0	0	0	0
Sales_reason_type_Other	1	1	1	1	1
Sales_reason_type_Promotion	0	1	0	0	0
Sales_reason_Manufacturer	0	0	0	0	0
Sales_reason_On_Promotion	0	1	0	0	0
Sales_reason_Other	0	0	0	0	0
Sales_reason_Price	1	0	1	1	1
Sales_reason_Quality	0	0	0	0	0
Sales_reason_Review	0	1	0	0	0
Sales_reason_Television_Advertisement	0	0	0	0	0

Missing and duplicate data

In [380…

```
# Count the number of missing data per variable
# Below we look at missing data using the pandas isnull() function
# the trick is: boolean value True = 1 so the sum
# gives the number of records

# Dataset.isnull().sum() is actually a dataframe and we are filtering out the zeros
```

```
# within the square brackets []

# filter on missing data variables
missings = model_data.isnull().sum()[model_data.isnull().sum()!=0]
print('Missing value per variable [5] : \n', missings)
```

```
Missing value per variable [5] :
 Days_elapsed_mean      7972
Days_elapsed_median    7972
Days_elapsed_min       7972
Days_elapsed_max       7972
dtype: int64
```

In [381…

```
#### There would be no need to fill the missing value, more than 2/3 of the customers have
```

In [382…

```
# Below we look at duplicated line in the datasets.
# if any duplicates are present:
# drop them with the following code

print('Duplicates Number : \n', model_data.duplicated().sum())
model_data.drop_duplicates(subset=None, keep='first',inplace= True)
```

```
Duplicates Number :
 0
```

Categorical data

In [383…

```
# Categorical Variables Distribution
#We take a look at the number of instances (rows) that belong to each category.
#We do this for all the categorical variables
```

In [384…

```
# Get columns
cols = model_data.columns
cols
```

Out[384…

```
Index(['Customer_id', 'Quantity_sum', 'Quantity_mean', 'Quantity_median',
       'Quantity_min', 'Quantity_max', 'Revenue_sum', 'Revenue_mean',
       'Revenue_median', 'Revenue_min', 'Revenue_max', 'Profit_sum',
       'Profit_mean', 'Profit_median', 'Profit_min', 'Profit_max',
       'Days_elapsed_sum', 'Days_elapsed_mean', 'Days_elapsed_median',
       'Days_elapsed_min', 'Days_elapsed_max', 'Recency', 'Frequency',
       'Monetary', 'RFM_segment', 'RFM_score', 'RFM_status', 'RFM_cluster',
       'Tenure_months', 'Churn', 'Birth_date', 'Marital_status', 'Gender',
       'Yearly_income', 'Number_children_at_home', 'Education', 'Ocupation',
       'Commute_distance', 'Total_children', 'House_ownership',
       'Car_ownership', 'Age', 'Age_group', 'Income_group',
       'Sales_reason_type_Marketing', 'Sales_reason_type_Other',
       'Sales_reason_type_Promotion', 'Sales_reason_Manufacturer',
       'Sales_reason_On_Promotion', 'Sales_reason_Other', 'Sales_reason_Price',
       'Sales_reason_Quality', 'Sales_reason_Review',
       'Sales_reason_Television_Advertisement'],
      dtype='object')
```

In [385…

```
# delete some data that are not necessary for analysis
# customer id is randomly generated number and revenue_sum is the same as Monetary
#'Days_elapsed_sum', 'Days_elapsed_mean', 'Days_elapsed_median','Days_elapsed_min', 'Days_
# has alot of missing values about 7972
del_var = ['Customer_id', 'Quantity_min', 'Revenue_sum', 'Birth_date',
           'Days_elapsed_sum', 'Days_elapsed_mean', 'Days_elapsed_median',
           'Days_elapsed_min', 'Days_elapsed_max']
```

```
model_data.drop(del_var, axis = 1, inplace= True)
model_data.head()
```

Out[385…

	Quantity_sum	Quantity_mean	Quantity_median	Quantity_max	Revenue_mean	Revenue_median	Revenue_min	R
0	4.00	2.00	2.00	3.00	1675.70	1675.70	1000.44	
1	2.00	1.00	1.00	1.00	2183.20	2183.20	2071.42	
2	5.00	2.50	2.50	4.00	1686.95	1686.95	1000.44	
3	3.00	3.00	3.00	3.00	96.46	96.46	96.46	
4	1.00	1.00	1.00	1.00	4.99	4.99	4.99	

5 rows × 45 columns

In [ ]:

In [386…

```
# distribution / frequency per category

#categorical variables
cat_var =['RFM_segment', 'RFM_score', 'RFM_status', 'RFM_cluster', 'Marital_status', 'Genc
       'Total_children', 'Number_children_at_home', 'Education', 'Ocupation', 'House_owner
       'Car_ownership', 'Commute_distance', 'Age_group', 'Income_group', 'Sales_reason_typ
       'Sales_reason_type_Other', 'Sales_reason_type_Promotion', 'Sales_reason_Manufacture
       'Sales_reason_On_Promotion', 'Sales_reason_Other', 'Sales_reason_Price', 'Sales_rea
       'Sales_reason_Review', 'Sales_reason_Television_Advertisement']

for var in cat_var:
    print(model_data.groupby(var).size())
```

```
RFM_segment
111      164
112       35
113       97
114        2
121      328
122       99
123      258
124       40
141        2
142      465
143      348
144      516
211      154
212       12
213       51
214        1
221      432
222      109
223      159
224       37
241        7
242      460
243      348
244      522
311      139
312        7
313       25
321      571
322       97
```

```
323     154
324      29
341       6
342     575
343     484
344     728
411      43
413       8
421     810
422     137
423      88
424      17
441     281
442    1118
443     870
444    1242
dtype: int64
RFM_score
3      164
4      517
5      779
6     1043
7     1604
8     1150
9     1837
10    2141
11    1598
12    1242
dtype: int64
RFM_status
Gold      2503
Silver    4591
Bronze    2141
Green     2840
dtype: int64
RFM_cluster
0    2136
1     458
2    3600
3    5881
dtype: int64
Marital_status
M    6504
S    5571
dtype: int64
Gender
F    5978
M    6097
dtype: int64
Total_children
0    3341
1    2416
2    2467
3    1434
4    1519
5     898
dtype: int64
Number_children_at_home
0    7258
1    1633
2    1103
3     802
4     676
5     603
dtype: int64
Education
```

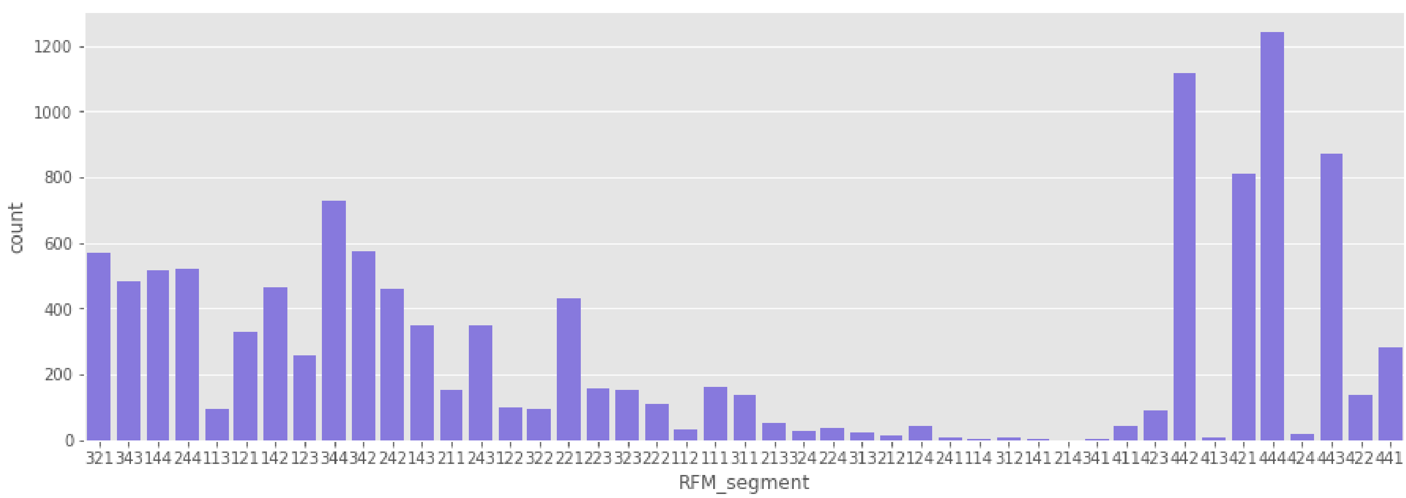
```
Bachelors              3445
Graduate Degree        2089
High School            2162
Partial College        3314
Partial High School    1065
dtype: int64
Ocupation
Clerical          1917
Management        1976
Manual            1591
Professional      3576
Skilled Manual    3015
dtype: int64
House_ownership
0    3959
1    8116
dtype: int64
Car_ownership
0    2761
1    3174
2    4283
3    1036
4     821
dtype: int64
Commute_distance
0-1 Miles     4108
1-2 Miles     2160
10+ Miles     1601
2-5 Miles     2152
5-10 Miles    2054
dtype: int64
Age_group
20s                0
30s             2187
40-50s          7165
60s or older    2723
dtype: int64
Income_group
Low              767
Lower-middle    4915
Upper-middle    6203
High             190
dtype: int64
Sales_reason_type_Marketing
0    11686
1      389
dtype: int64
Sales_reason_type_Other
0      427
1    11648
dtype: int64
Sales_reason_type_Promotion
0    10096
1     1979
dtype: int64
Sales_reason_Manufacturer
0    10844
1     1231
dtype: int64
Sales_reason_On_Promotion
0    10096
1     1979
dtype: int64
Sales_reason_Other
0    11240
1      835
```

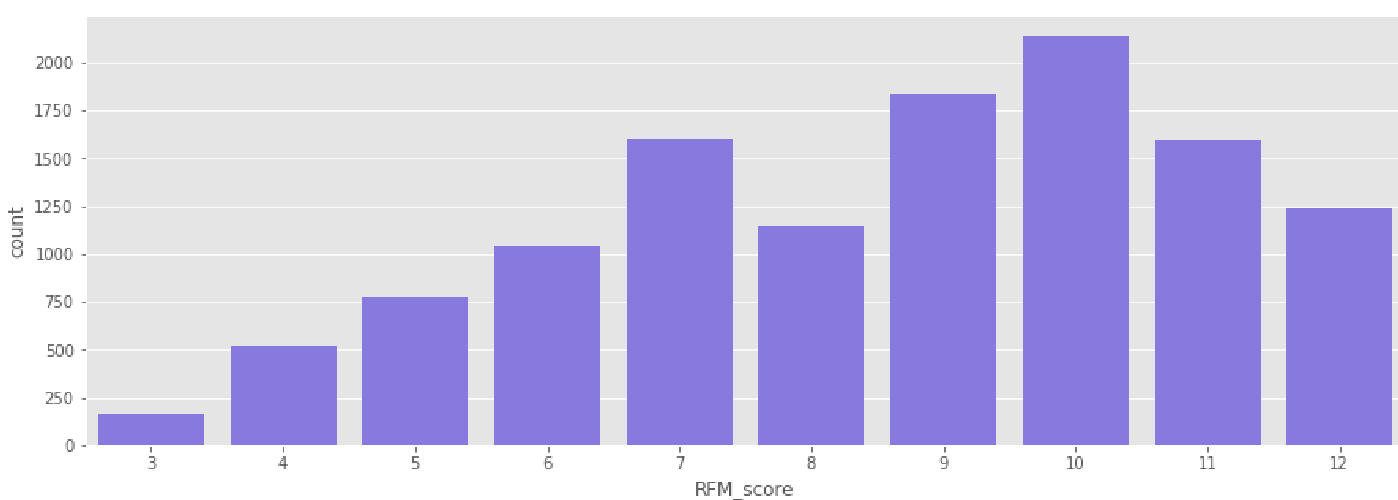
```
dtype: int64
Sales_reason_Price
0     1262
1    10813
dtype: int64
Sales_reason_Quality
0    10938
1     1137
dtype: int64
Sales_reason_Review
0    11307
1      768
dtype: int64
Sales_reason_Television_Advertisement
0    11686
1      389
dtype: int64
```

In [387…

```python
# PLot distribution/frequency per category
cat_df = model_data[cat_var]
plt.rcParams['figure.figsize'] = (15, 5) # Chart sizes

for i, col in enumerate(cat_df.columns):
   plt.figure(i)
   sns.countplot(x=col, data=cat_df, color = 'mediumslateblue')
```

```
<ipython-input-387-f8c192283c57>:6: RuntimeWarning: More than 20 figures have been opened.
Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained unt
il explicitly closed and may consume too much memory. (To control this warning, see the rc
Param `figure.max_open_warning`).
  plt.figure(i)
```

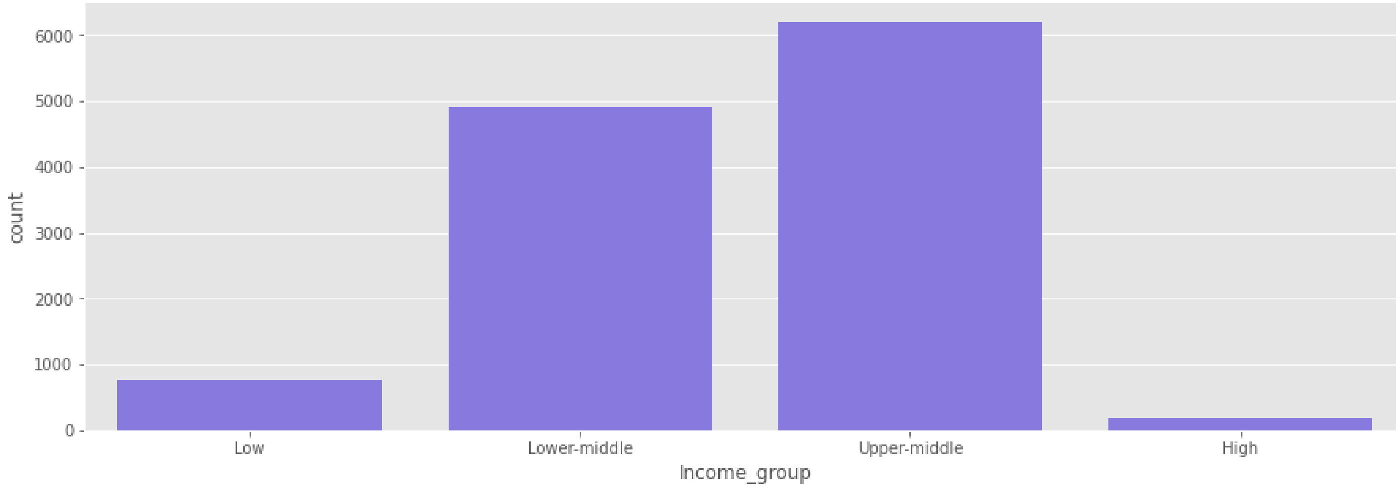


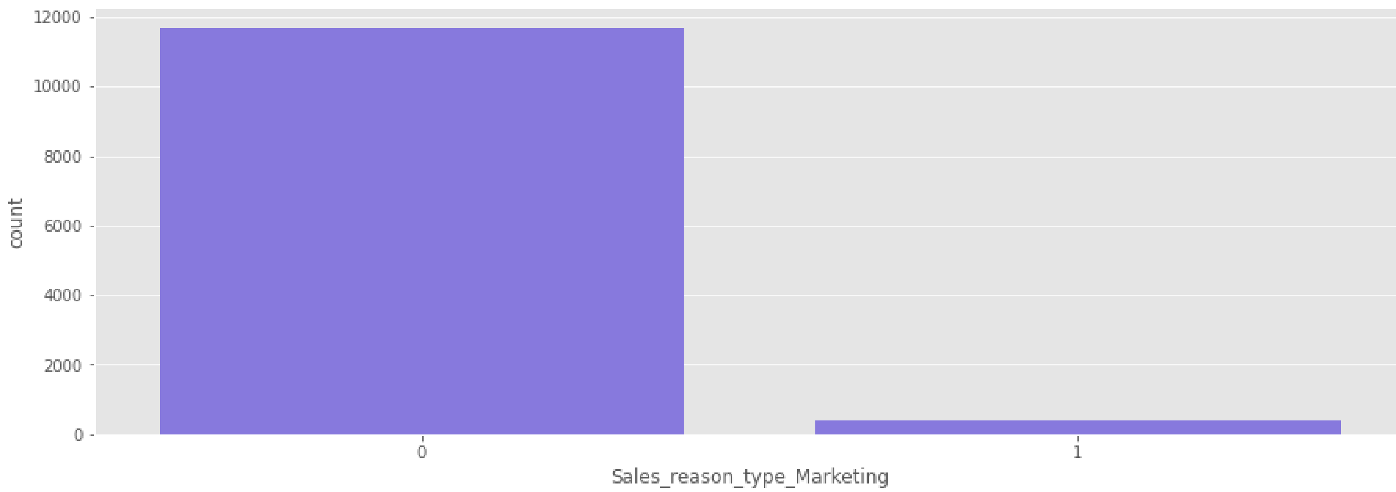


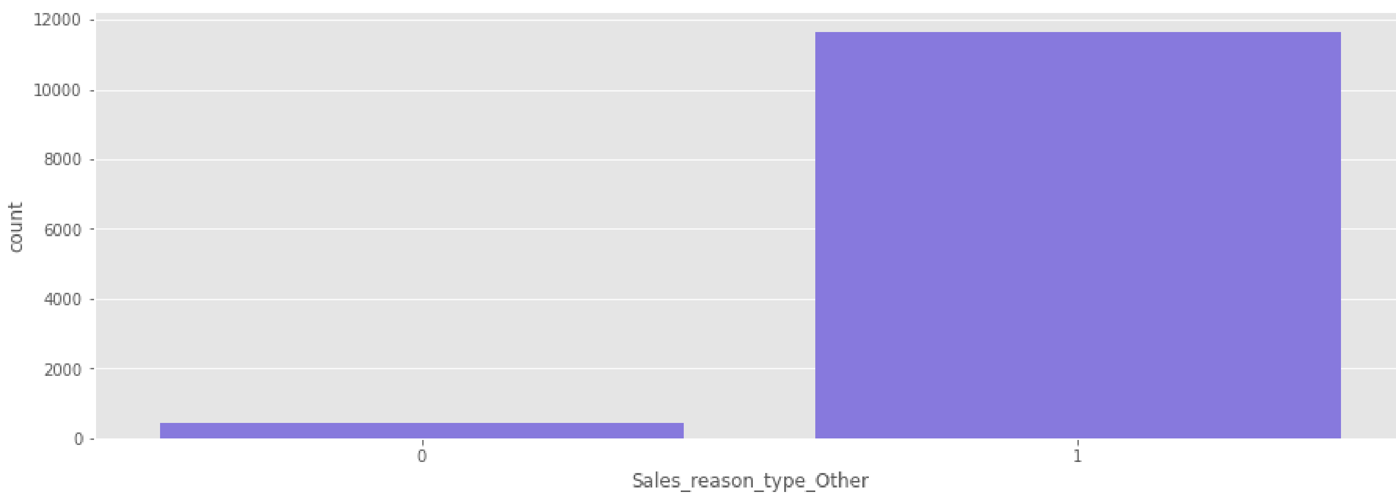

count
RFM_status
Gold
Silver
Bronze
Green
RFM_cluster
0
1
2
3
Marital_status
M
S
Gender
M
F
0
1000
2000
3000
4000
5000
6000

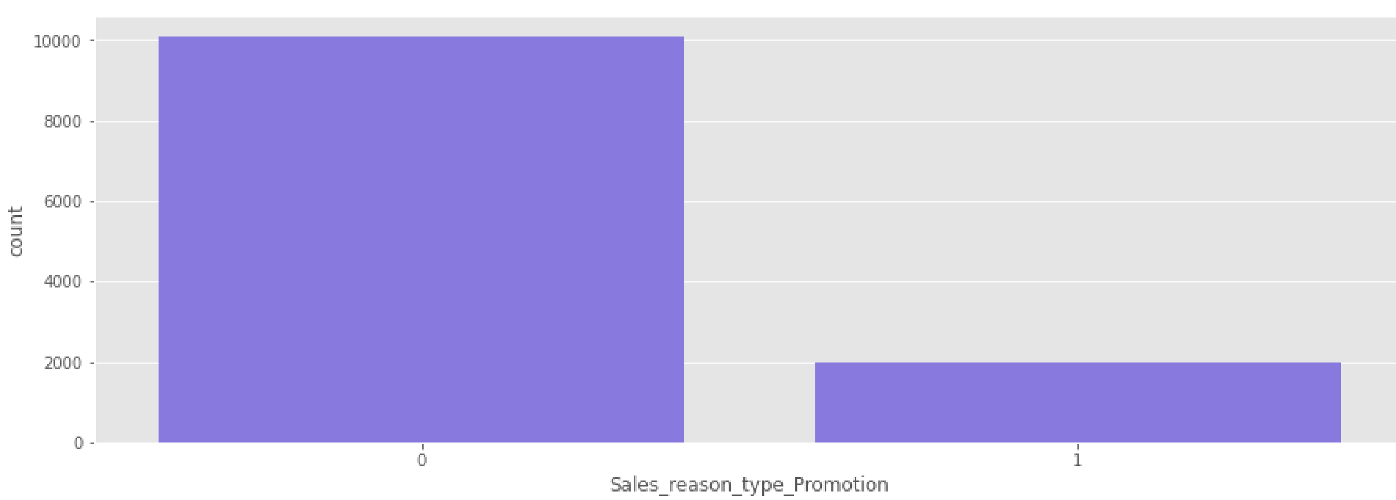
count
3500
3000
2500
2000
1500
1000
500
0
0
1
2
3
4
5
Total_children
count
7000
6000
5000
4000
3000
2000
1000
0
0
1
2
3
4
5
Number_children_at_home
count
3500
3000
2500
2000
1500
1000
500
0
Bachelors
Partial College
High School
Graduate Degree
Partial High School
Education
count
3500
3000
2500
2000
1500
1000
500
0
Management
Clerical
Professional
Manual
Skilled Manual
Ocupation

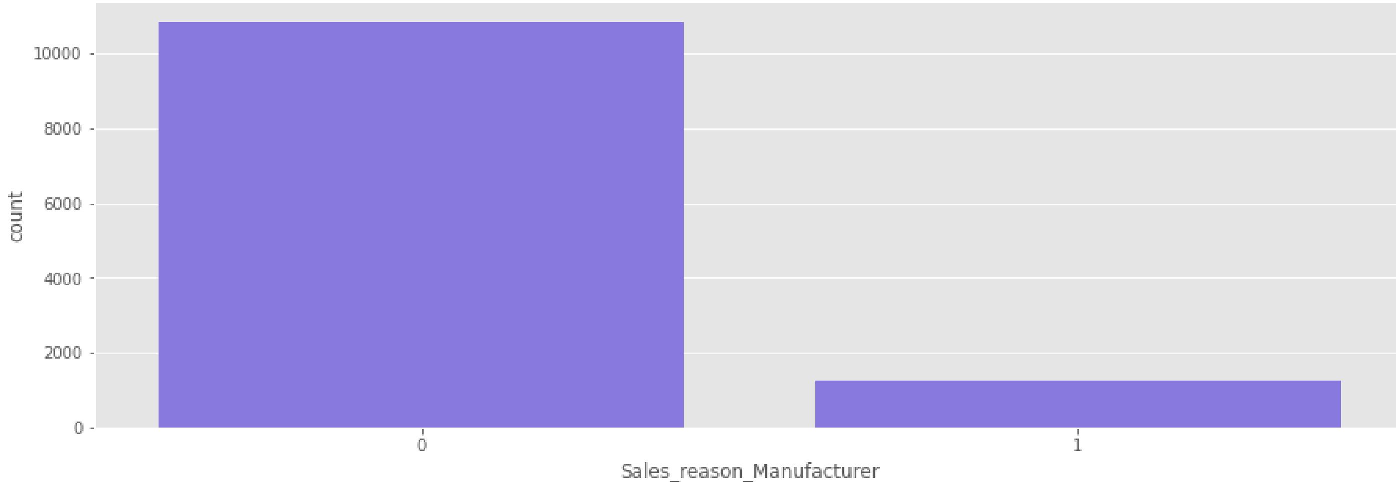

count
House_ownership
1
0
0
1000
2000
3000
4000
5000
6000
7000
8000
count
Car_ownership
0
1
2
3
4
500
1500
2500
3500
count
Commute_distance
1-2 Miles
2-5 Miles
10+ Miles
5-10 Miles
0-1 Miles
count
Age_group
20s
30s
40-50s
60s or older

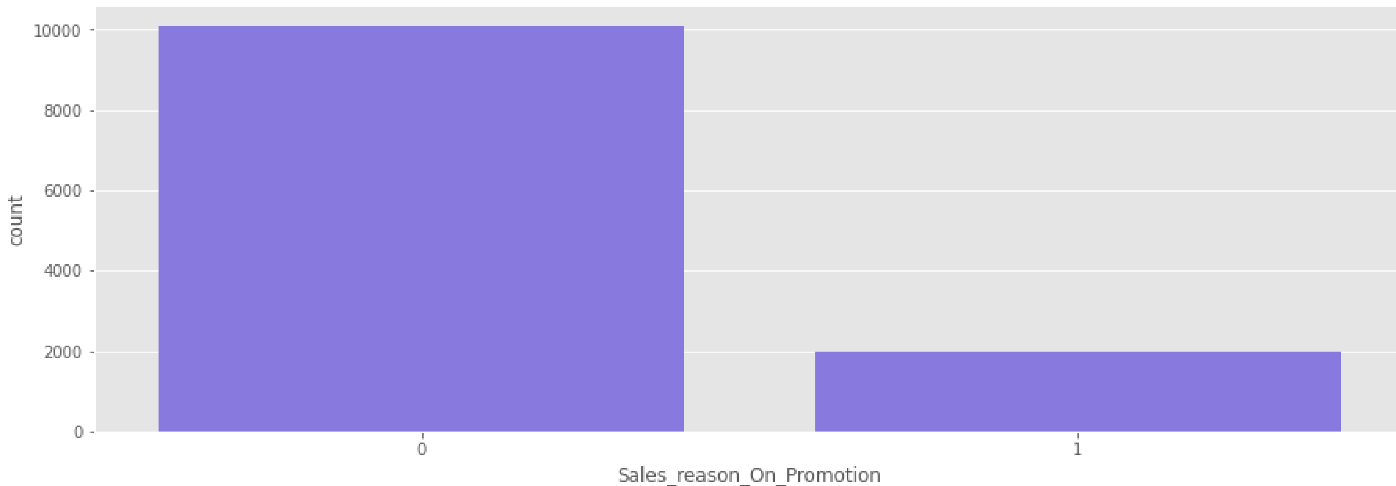

count
Low
Lower-middle
Upper-middle
High
Income_group

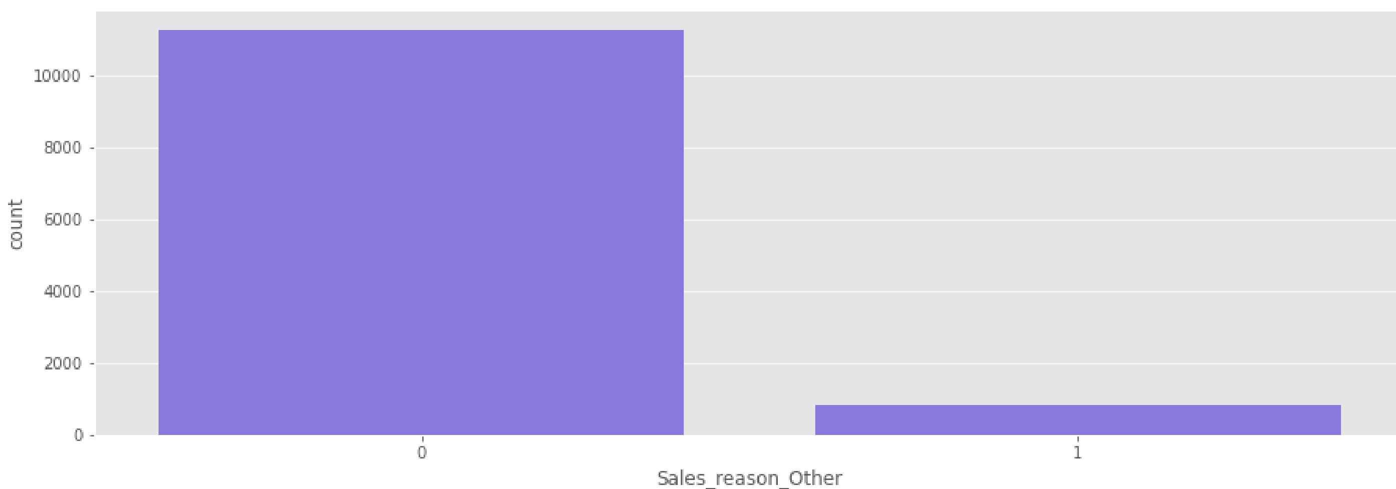

count
0
1
Sales_reason_type_Marketing

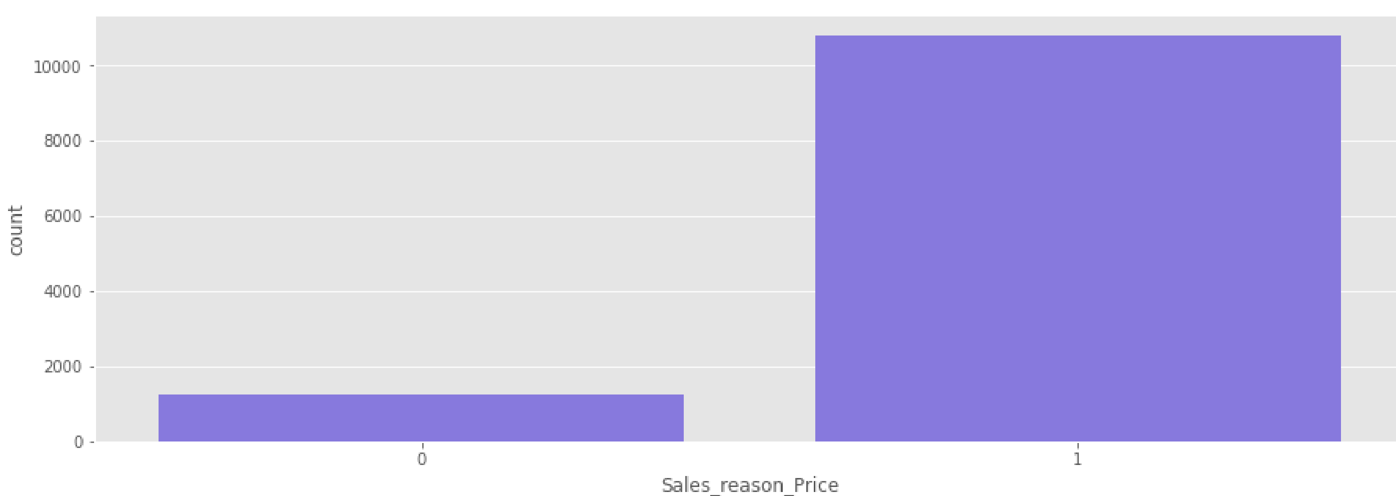

count
0
1
Sales_reason_type_Other

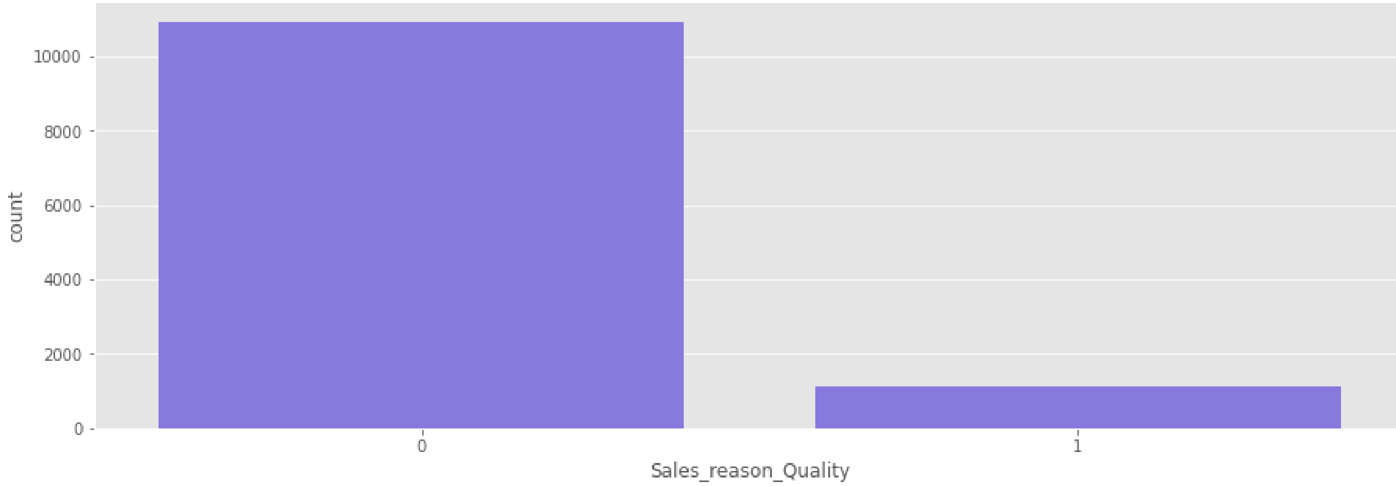

count
0
1
Sales_reason_type_Promotion

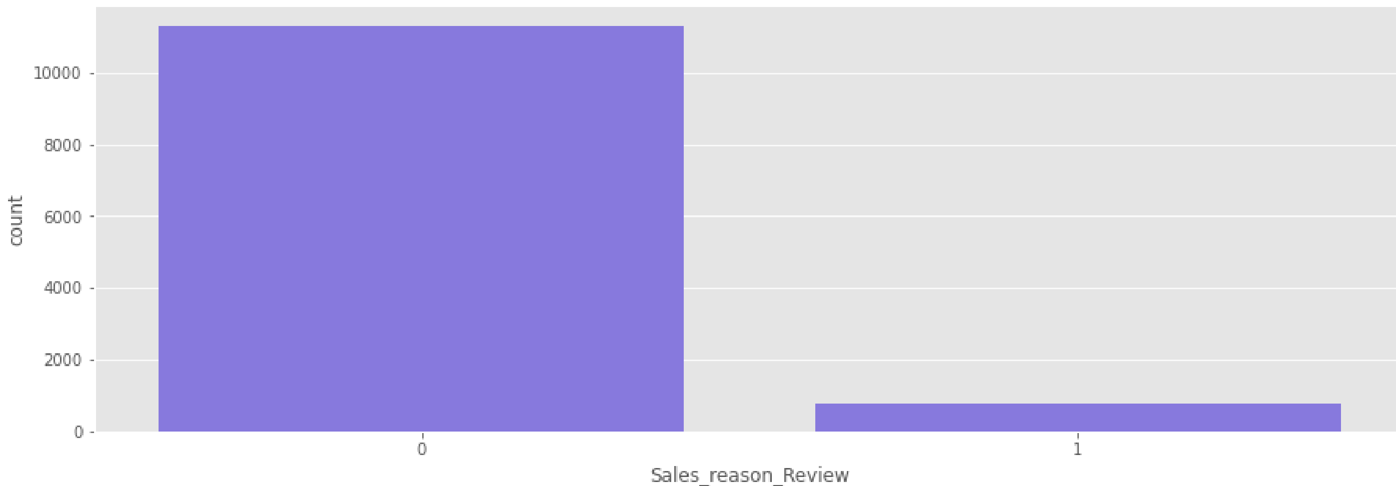

count
10000
8000
6000
4000
2000
0
0
1
Sales_reason_Manufacturer

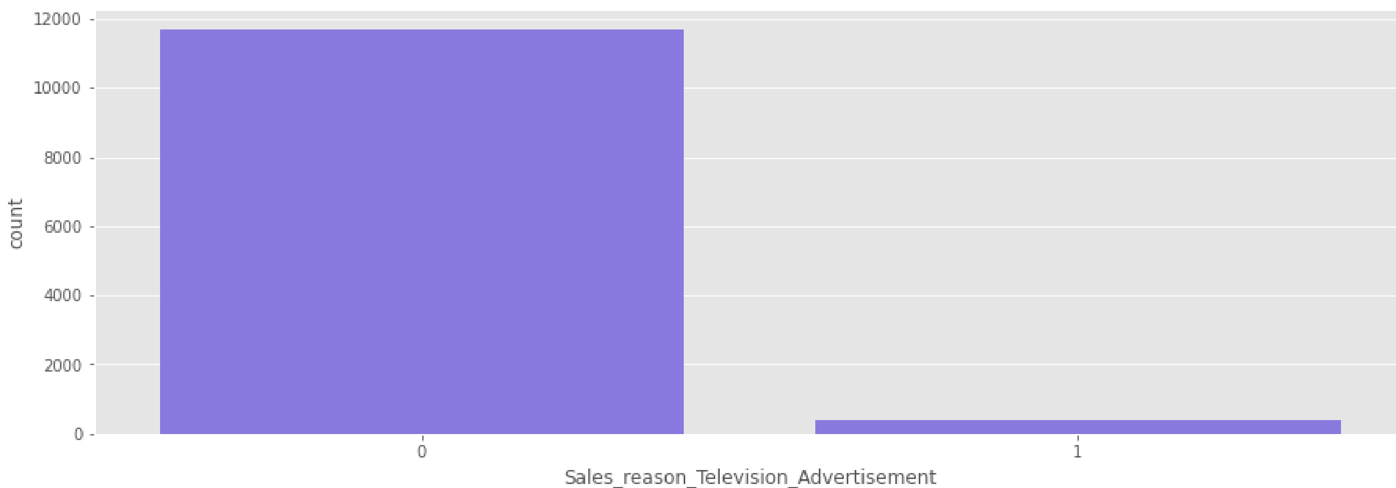

count
10000
8000
6000
4000
2000
0
0
1
Sales_reason_On_Promotion


count
10000
8000
6000
4000
2000
0
0
1
Sales_reason_Other


count
10000
8000
6000
4000
2000
0
0
1
Sales_reason_Price







In [388…

```
### Transforming Categorical Variable in Boolean 0/1 variables
```

In [389…

```
# Most machine learning models only accept numerical variables
# Some of our variables contains string or letter
# for example Gender = M or F
# Further more a more restricted number of model require boolean variables
# Below we transform all variables to boolean by default

# Encode categorical data (besides drivers which are already binary)
dummies = ['RFM_segment', 'RFM_score', 'RFM_status', 'RFM_cluster', 'Marital_status', 'Ger
           'Total_children', 'Number_children_at_home', 'Education', 'Ocupation', 'House_c
           'Car_ownership', 'Commute_distance', 'Age_group', 'Income_group']

encoded_data = pd.get_dummies(model_data, columns = [v for v in dummies], drop_first = Fal
```

In [390…

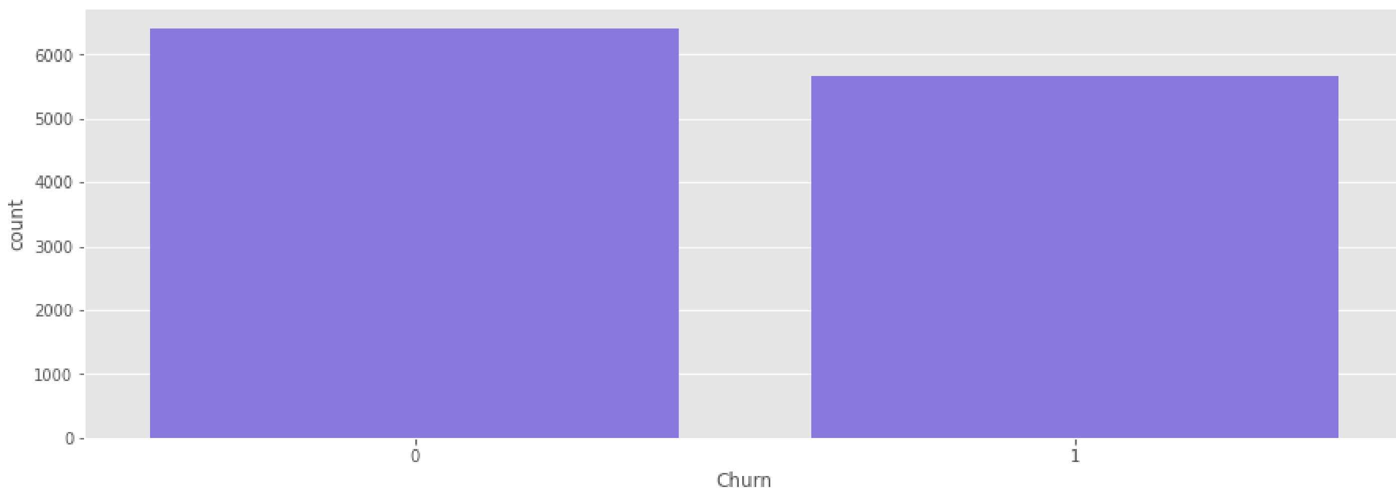
```
# Get frequency of target variable
encoded_data['Churn'].value_counts()
```

Out[390…

```
0    6408
1    5667
Name: Churn, dtype: int64
```

In [391…

```
sns.countplot(x = encoded_data['Churn'], color = 'mediumslateblue')
pass
```



Continuous data

In [392…

```
# Filter continuous variables
cont = [v for v in cols if v not in cat_var and v not in del_var and v != 'Churn']
```

In [393…

```
# Get quick stats
encoded_data[cont].describe().transpose()
```

Out[393…

	count	mean	std	min	25%	50%	75%	max
Quantity_sum	12075.00	3.12	2.33	1.00	2.00	3.00	4.00	68.00
Quantity_mean	12075.00	2.22	0.94	1.00	1.50	2.00	3.00	8.00
Quantity_median	12075.00	2.22	0.95	1.00	1.50	2.00	3.00	8.00
Quantity_max	12075.00	2.52	1.10	1.00	2.00	2.00	3.00	8.00
Revenue_mean	12075.00	931.55	1065.61	2.29	39.98	178.98	1878.30	3578.27
Revenue_median	12075.00	930.78	1065.02	2.29	39.98	178.98	1958.77	3578.27
Revenue_min	12075.00	789.20	954.85	2.29	37.93	178.98	1249.84	3578.27
Revenue_max	12075.00	1075.14	1251.28	2.29	42.28	187.98	2319.99	3578.27
Profit_sum	12075.00	635.91	847.24	1.43	25.03	168.59	1058.52	5254.60
Profit_mean	12075.00	379.69	430.16	1.43	24.40	107.68	755.23	1487.84
Profit_median	12075.00	379.68	430.55	1.43	24.36	107.68	771.52	1487.84
Profit_min	12075.00	320.14	385.01	1.43	21.90	104.73	481.60	1487.84
Profit_max	12075.00	439.42	506.25	1.43	25.03	117.68	924.56	1487.84
Recency	12075.00	225.99	163.10	1.00	107.00	226.00	305.00	1126.00

	count	mean	std	min	25%	50%	75%	max
Frequency	12075.00	1.43	0.94	1.00	1.00	1.00	2.00	28.00
Monetary	12075.00	1552.88	2064.03	2.29	48.97	293.40	2477.78	13269.27
Tenure_months	12075.00	4.63	8.10	0.00	0.00	0.00	6.93	35.78
Yearly_income	12075.00	56841.41	32094.53	10000.00	30000.00	60000.00	70000.00	170000.00
Age	12075.00	51.77	11.56	35.00	42.00	50.00	59.00	105.00

In [394...

```
# Data Visualisation

#We are going to look at two types of plots:

#Univariate plots to better understand each attribute

#Multivariate plots to better understand the relationships between attributes.
```

In [395...

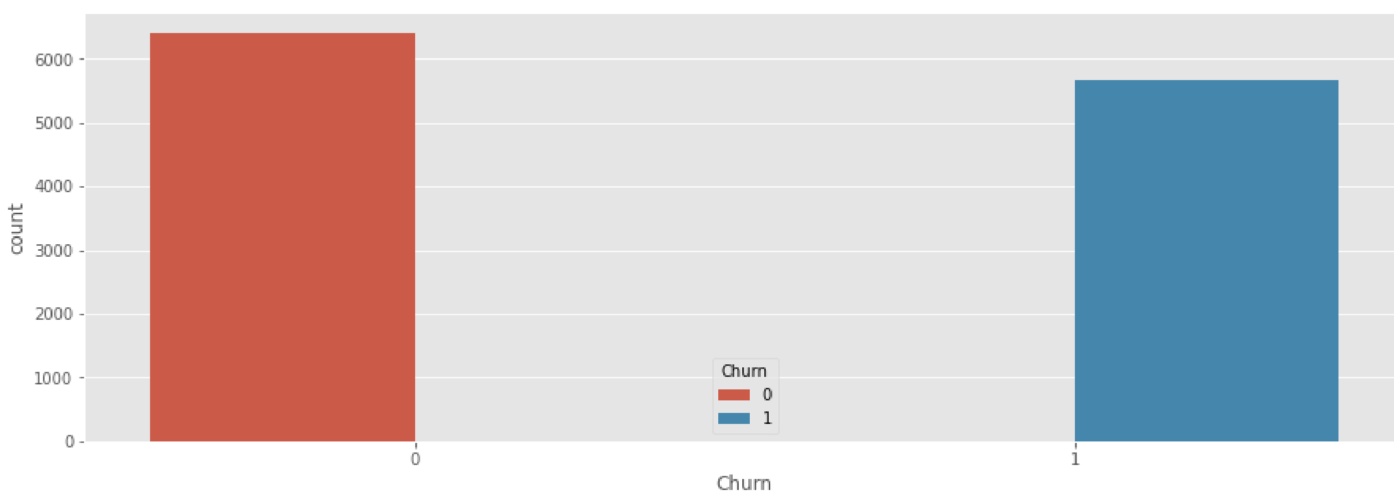
```
### 1 Univariate Plots
#We plot each individual variable.
#This gives us a much clearer idea of the distribution of the input attributes:

#Given that the input variables are numeric, we can create box and whisker plots of each.
#box and whisker plots
```

In [396...

```
## Univariate Analysis of churn with some categorical data
```

In [397...

```
# let plot count sample for the target variable
#sns. countplot(data= single_view,x= 'Churn', Color='mediumslateblue')
# let us see how they churn
# Show the plot
sns.countplot(x ='Churn', data =encoded_data, hue = 'Churn')

# Show the plot
plt.show()
```

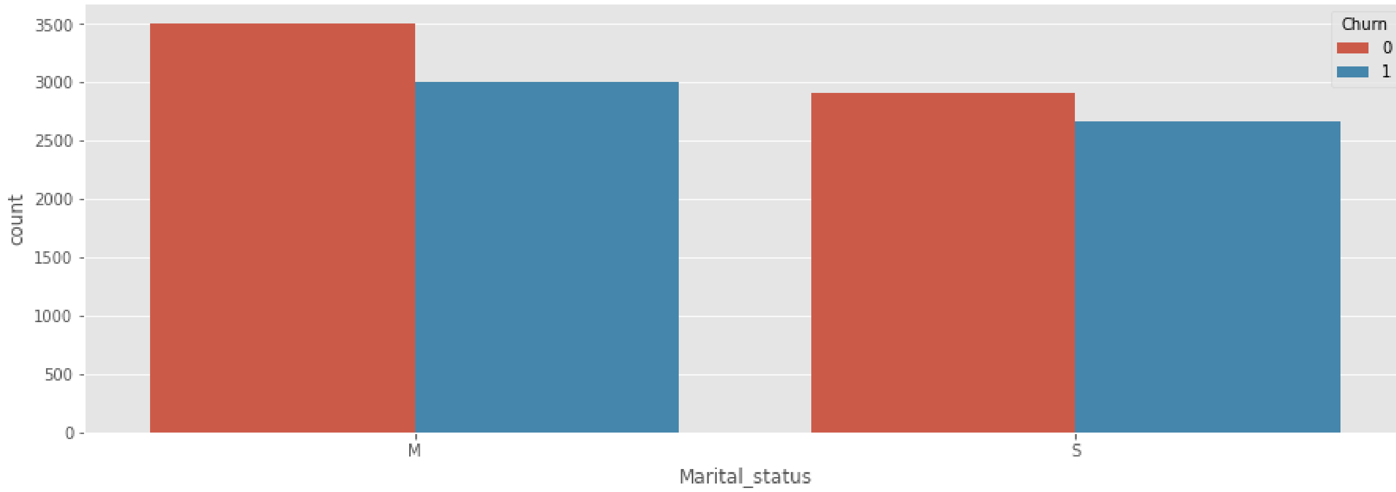


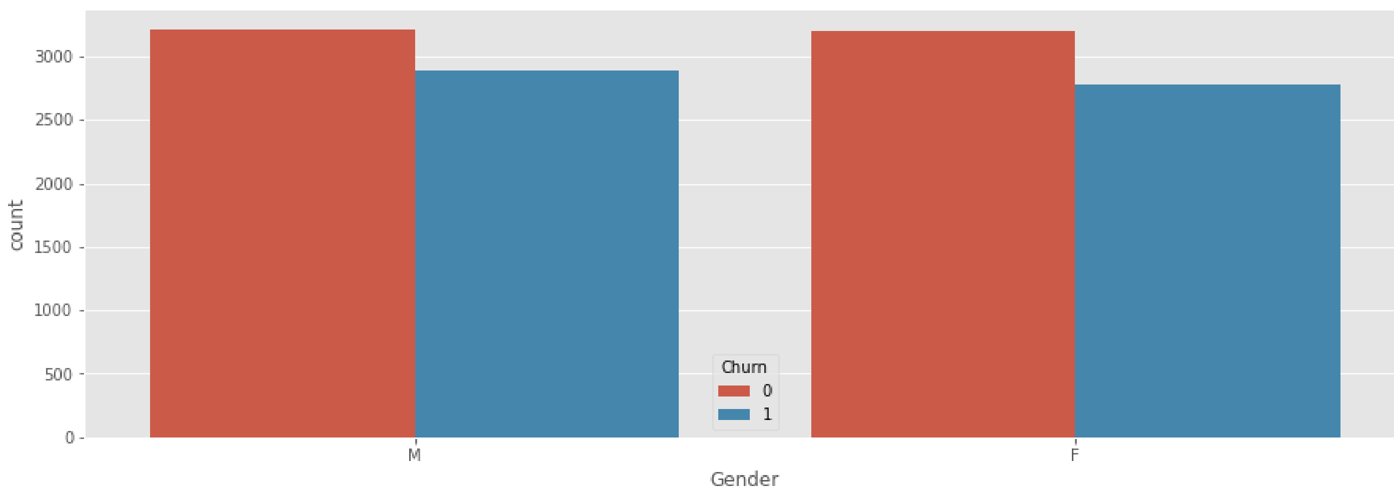
In [398...

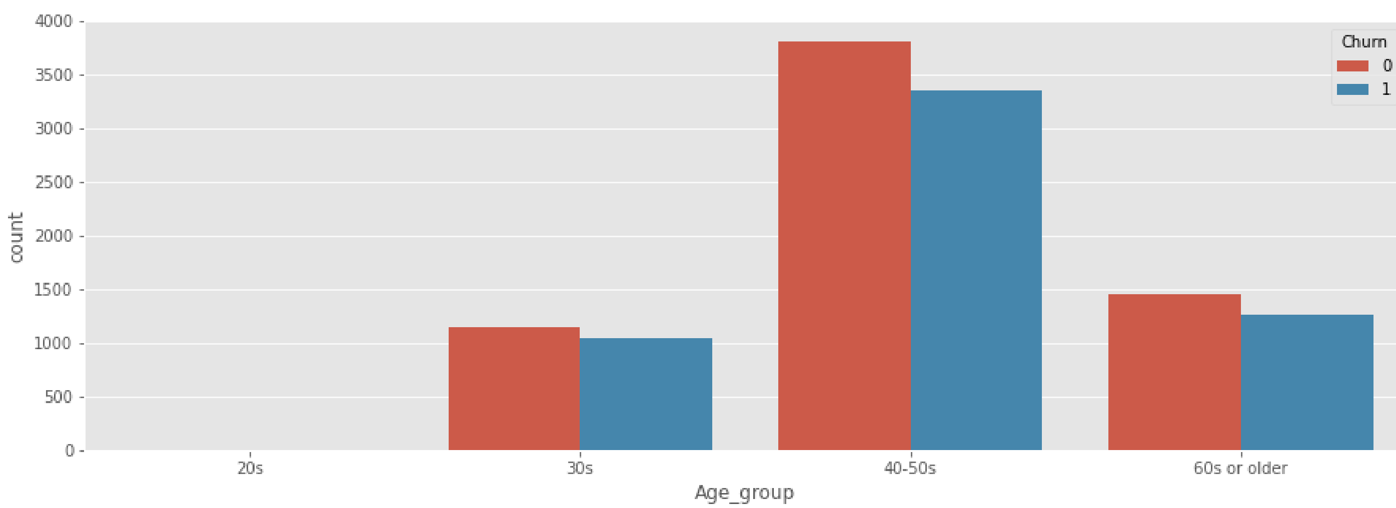
```
## Univariate Plots
#We plot each individual variable. This gives us a much clearer idea of the distribution 

#Given that the input variables are numeric, we can create box and whisker plots of each.
```

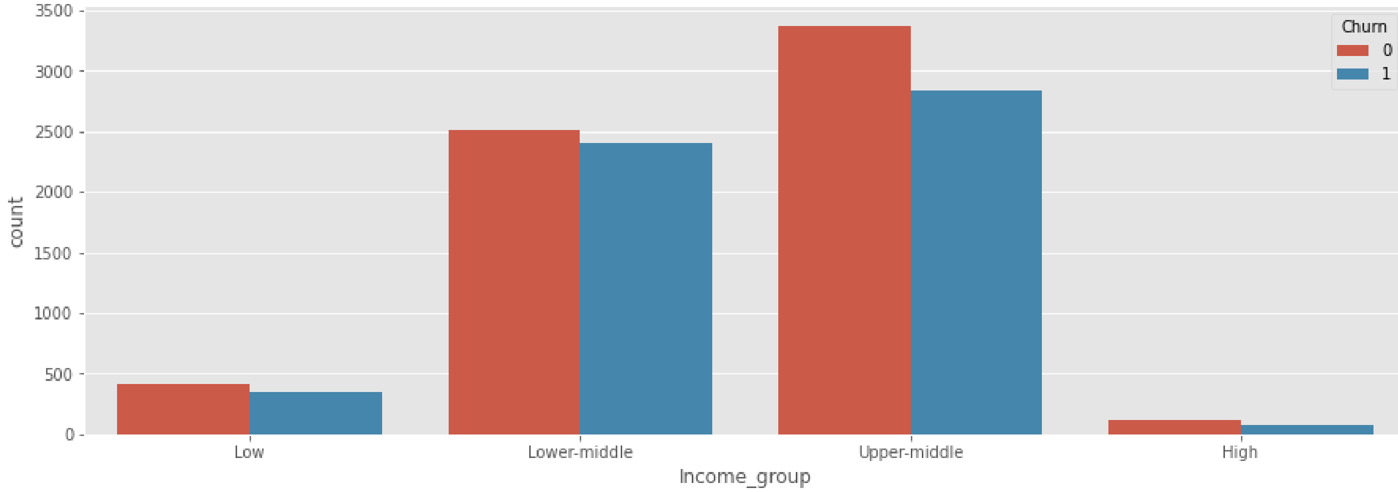
In [399…

```
#get cat varaible including churn to plot count plot
small_data = model_data[['Churn','Marital_status','Gender','Age_group', 'Income_group','Ec
                          'House_ownership']]
for i,predictor in enumerate(small_data.drop(columns=['Churn'])):
    plt.figure(i)
    sns.countplot(data = small_data,x=predictor,hue = 'Churn')
```

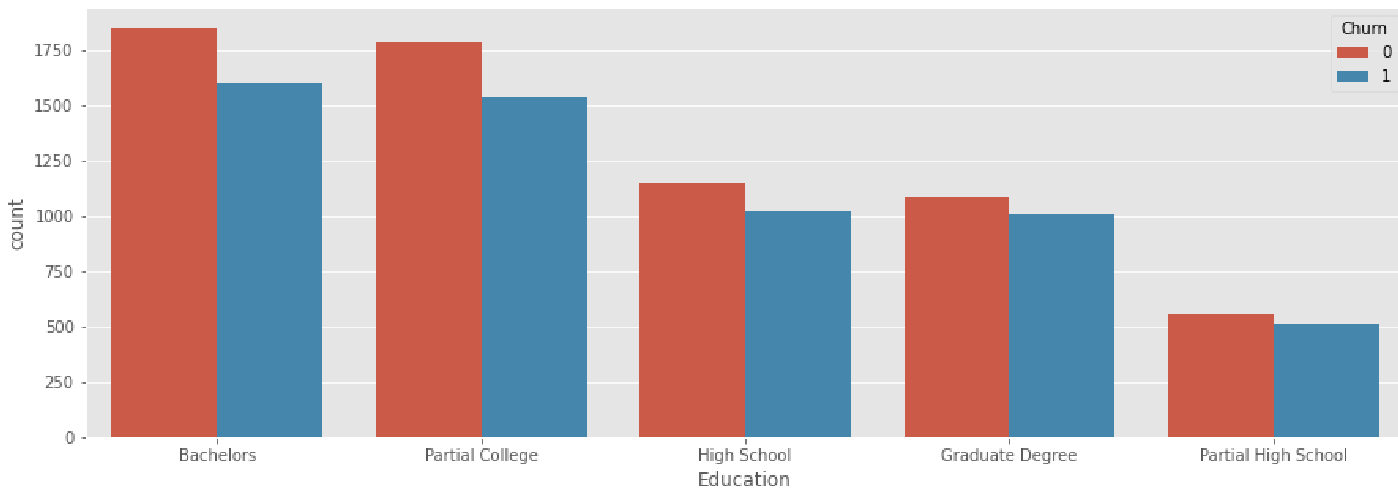




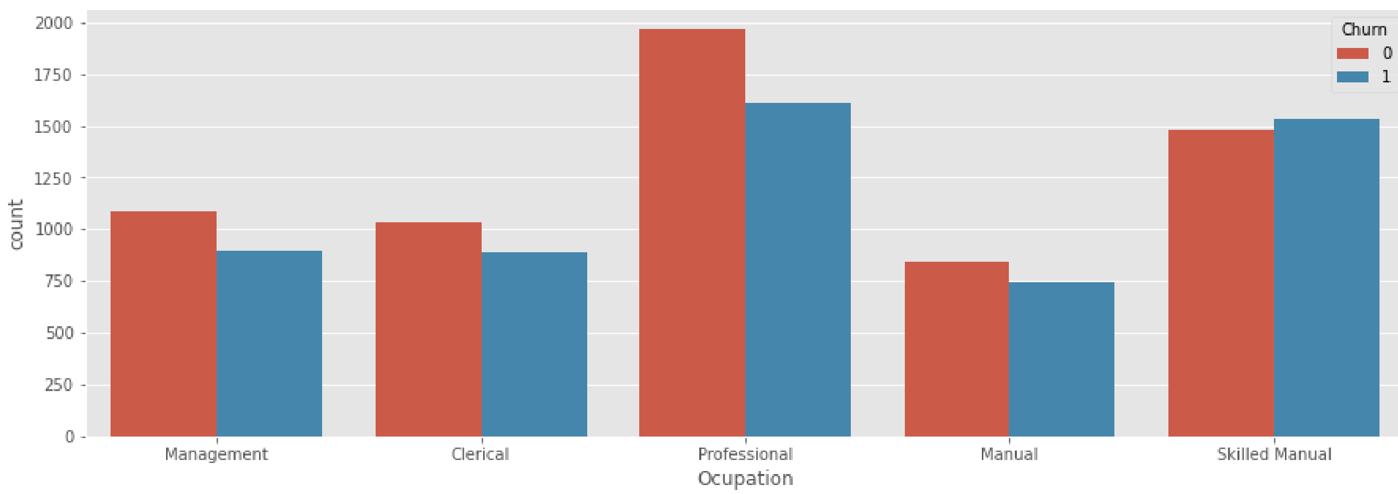




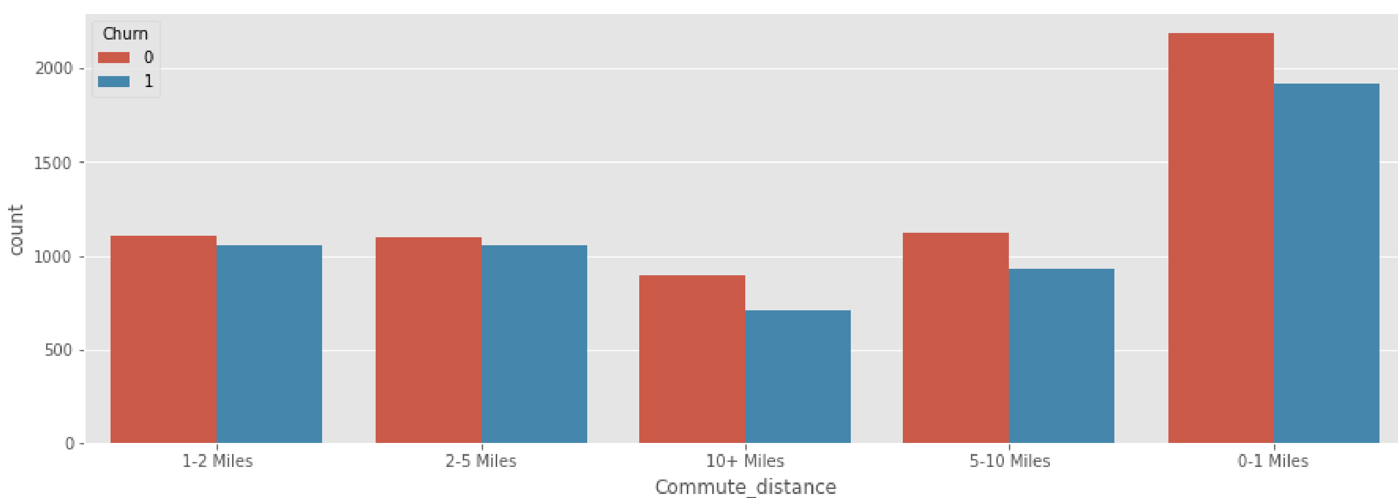
Churn
0
1
count
Low
Lower-middle
Upper-middle
High
Income_group



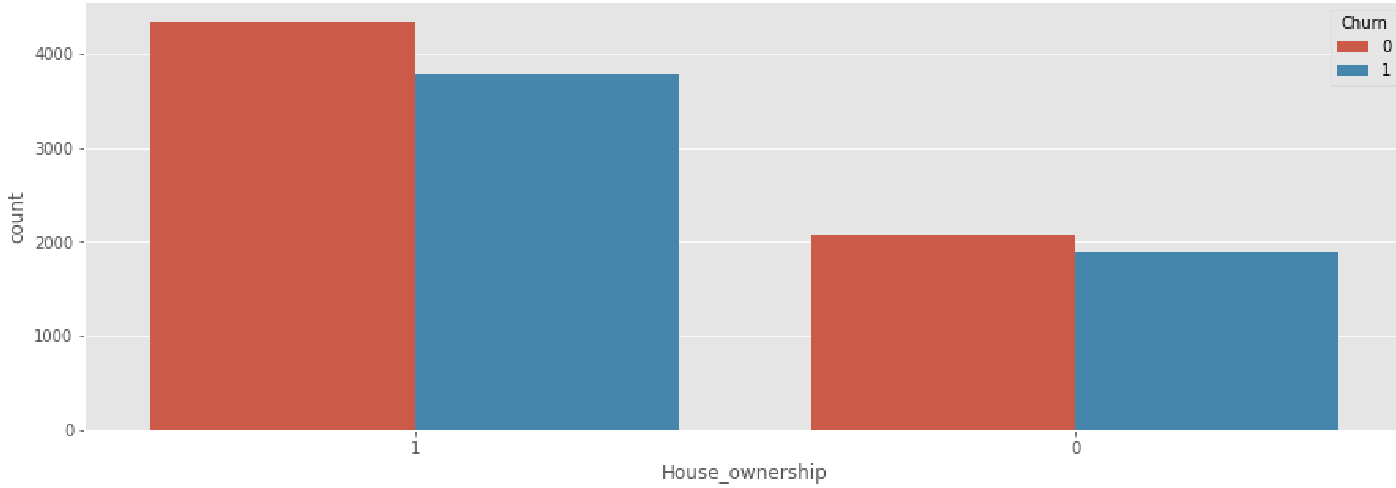
Churn
0
1
count
Bachelors
Partial College
High School
Graduate Degree
Partial High School
Education



Churn
0
1
count
Management
Clerical
Professional
Manual
Skilled Manual
Ocupation

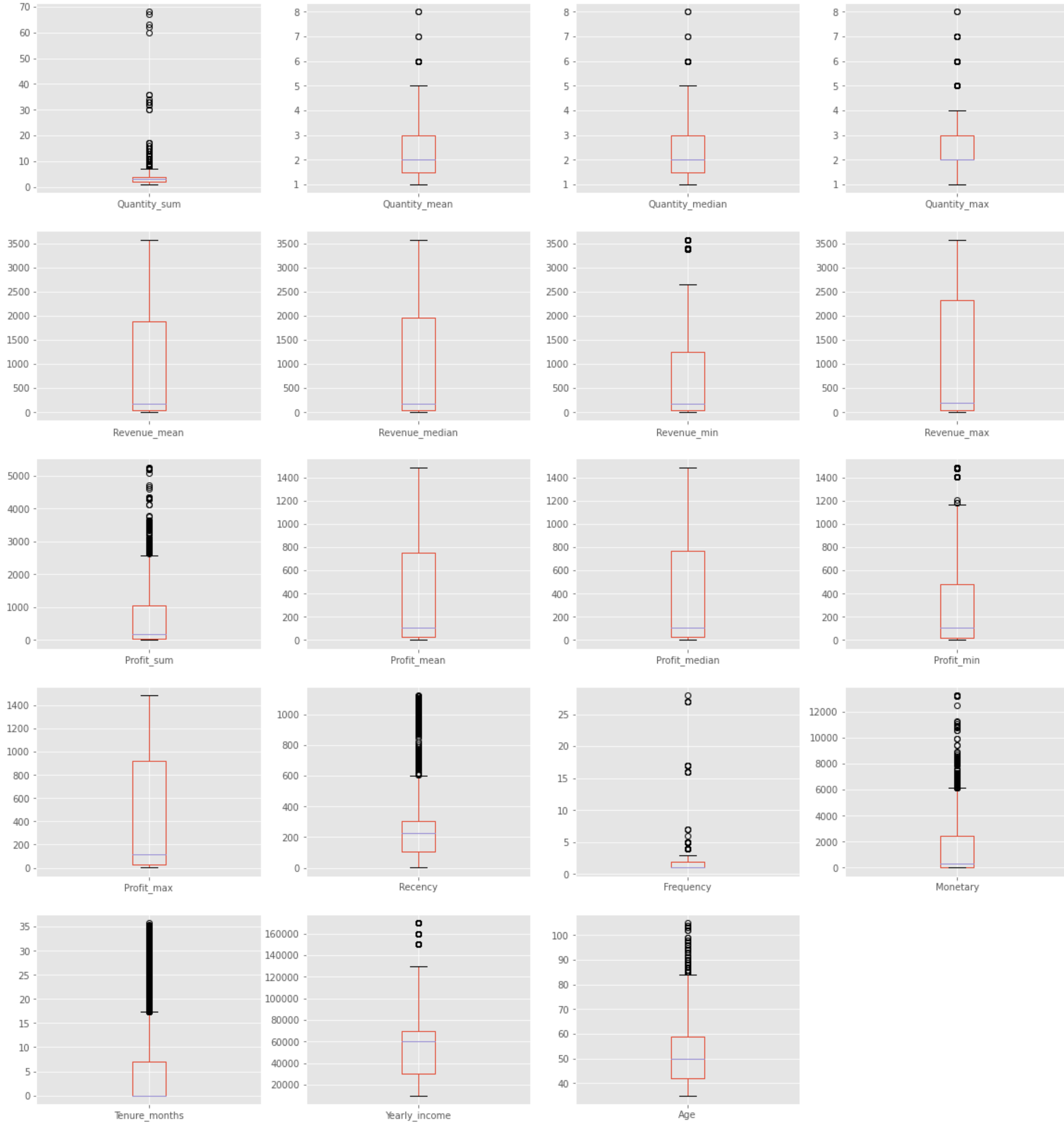


Churn
0
1
count
1-2 Miles
2-5 Miles
10+ Miles
5-10 Miles
0-1 Miles
Commute_distance



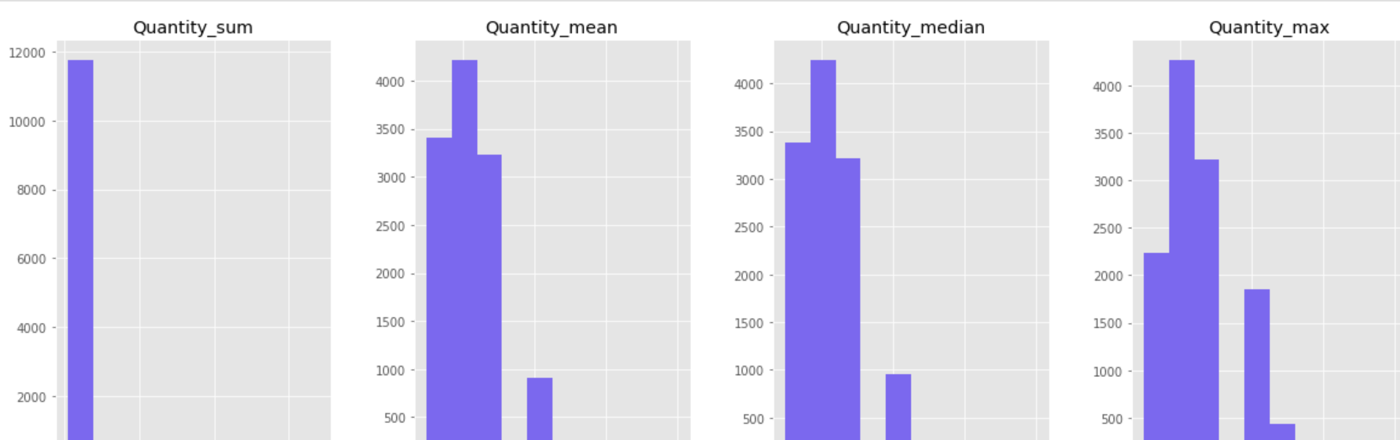
In [400…

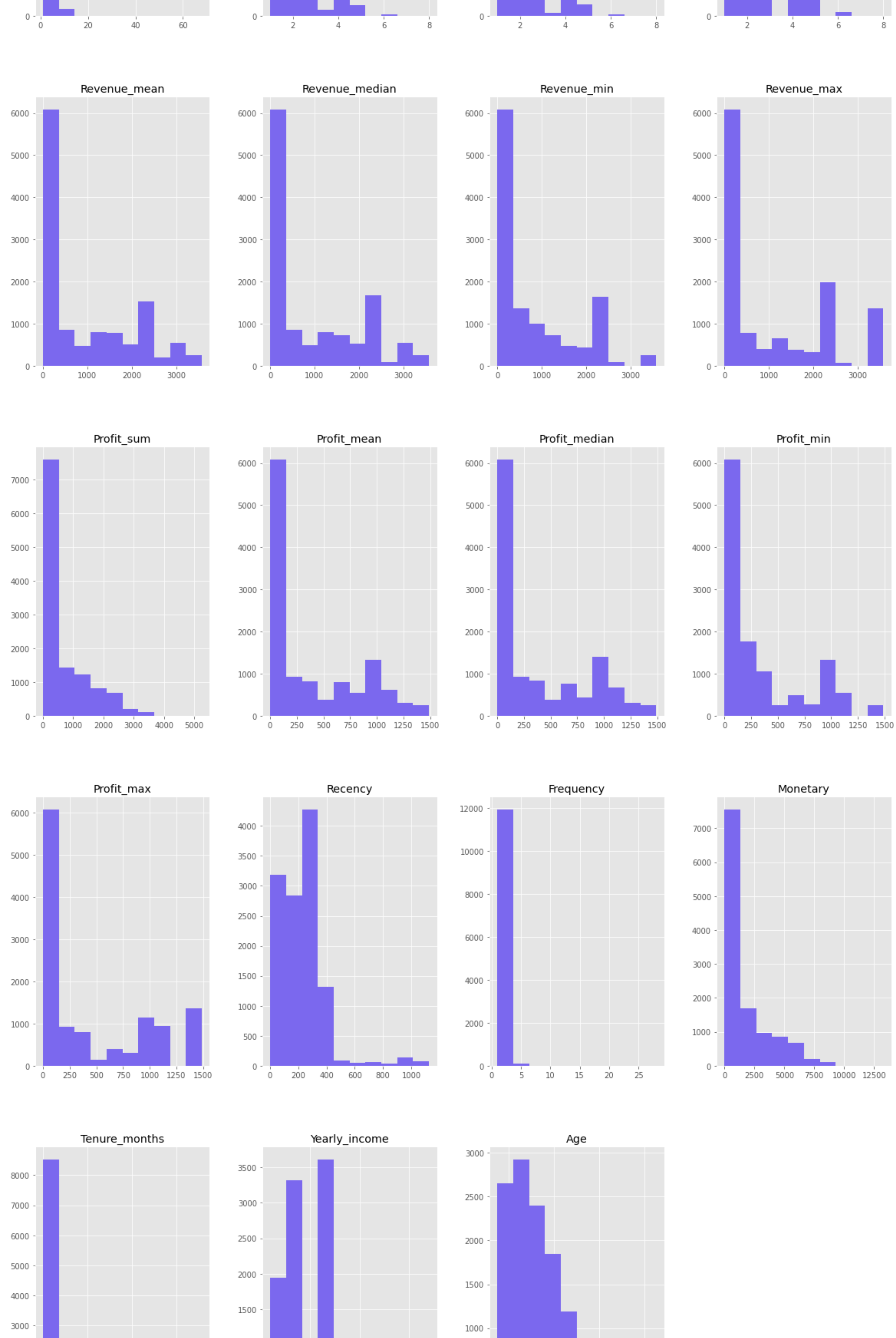
```
# drawing box plots, one graph with 43 subplots, !! do not share axes !!


# Plot boxplots
encoded_data[cont].plot(kind='box', subplots=True, figsize=(20,40),
                        layout=(9,4), sharex=False, sharey=False)
plt.show()
```



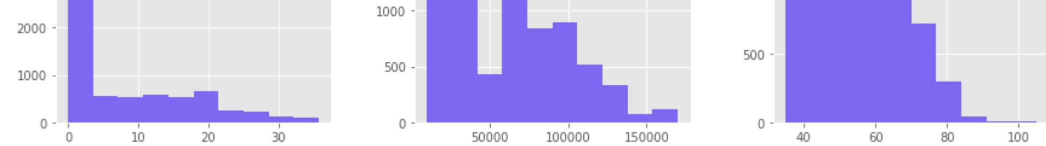
In [401…

```
# PLot histograms
encoded_data[cont].hist(figsize=(20,40), color = 'mediumslateblue')
pass
```





Revenue_mean
Revenue_median
Revenue_min
Revenue_max
Profit_sum
Profit_mean
Profit_median
Profit_min
Profit_max
Recency
Frequency
Monetary
Tenure_months
Yearly_income
Age



Feature selection

Information value

In [402…

```
# check on the data again
encoded_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12075 entries, 0 to 12074
Columns: 139 entries, Quantity_sum to Income_group_High
dtypes: float64(17), int64(3), uint8(119)
memory usage: 3.6 MB
```

In [403…

```
encoded_data.head()
```

Out[403…

	Quantity_sum	Quantity_mean	Quantity_median	Quantity_max	Revenue_mean	Revenue_median	Revenue_min	R
0	4.00	2.00	2.00	3.00	1675.70	1675.70	1000.44	
1	2.00	1.00	1.00	1.00	2183.20	2183.20	2071.42	
2	5.00	2.50	2.50	4.00	1686.95	1686.95	1000.44	
3	3.00	3.00	3.00	3.00	96.46	96.46	96.46	
4	1.00	1.00	1.00	1.00	4.99	4.99	4.99	

5 rows × 139 columns

In [404…

```
list(encoded_data.columns)
```

Out[404…

```
['Quantity_sum',
 'Quantity_mean',
 'Quantity_median',
 'Quantity_max',
 'Revenue_mean',
 'Revenue_median',
 'Revenue_min',
 'Revenue_max',
 'Profit_sum',
 'Profit_mean',
 'Profit_median',
 'Profit_min',
 'Profit_max',
 'Recency',
 'Frequency',
 'Monetary',
 'Tenure_months',
 'Churn',
 'Yearly_income',
 'Age',
 'Sales_reason_type_Marketing',
 'Sales_reason_type_Other',
 'Sales_reason_type_Promotion',
```

```
'Sales_reason_Manufacturer',
'Sales_reason_On_Promotion',
'Sales_reason_Other',
'Sales_reason_Price',
'Sales_reason_Quality',
'Sales_reason_Review',
'Sales_reason_Television_Advertisement',
'RFM_segment_111',
'RFM_segment_112',
'RFM_segment_113',
'RFM_segment_114',
'RFM_segment_121',
'RFM_segment_122',
'RFM_segment_123',
'RFM_segment_124',
'RFM_segment_141',
'RFM_segment_142',
'RFM_segment_143',
'RFM_segment_144',
'RFM_segment_211',
'RFM_segment_212',
'RFM_segment_213',
'RFM_segment_214',
'RFM_segment_221',
'RFM_segment_222',
'RFM_segment_223',
'RFM_segment_224',
'RFM_segment_241',
'RFM_segment_242',
'RFM_segment_243',
'RFM_segment_244',
'RFM_segment_311',
'RFM_segment_312',
'RFM_segment_313',
'RFM_segment_321',
'RFM_segment_322',
'RFM_segment_323',
'RFM_segment_324',
'RFM_segment_341',
'RFM_segment_342',
'RFM_segment_343',
'RFM_segment_344',
'RFM_segment_411',
'RFM_segment_413',
'RFM_segment_421',
'RFM_segment_422',
'RFM_segment_423',
'RFM_segment_424',
'RFM_segment_441',
'RFM_segment_442',
'RFM_segment_443',
'RFM_segment_444',
'RFM_score_3',
'RFM_score_4',
'RFM_score_5',
'RFM_score_6',
'RFM_score_7',
'RFM_score_8',
'RFM_score_9',
'RFM_score_10',
'RFM_score_11',
'RFM_score_12',
'RFM_status_Gold',
'RFM_status_Silver',
'RFM_status_Bronze',
'RFM_status_Green',
```

```
 'RFM_cluster_0',
 'RFM_cluster_1',
 'RFM_cluster_2',
 'RFM_cluster_3',
 'Marital_status_M',
 'Marital_status_S',
 'Gender_F',
 'Gender_M',
 'Total_children_0',
 'Total_children_1',
 'Total_children_2',
 'Total_children_3',
 'Total_children_4',
 'Total_children_5',
 'Number_children_at_home_0',
 'Number_children_at_home_1',
 'Number_children_at_home_2',
 'Number_children_at_home_3',
 'Number_children_at_home_4',
 'Number_children_at_home_5',
 'Education_Bachelors',
 'Education_Graduate Degree',
 'Education_High School',
 'Education_Partial College',
 'Education_Partial High School',
 'Ocupation_Clerical',
 'Ocupation_Management',
 'Ocupation_Manual',
 'Ocupation_Professional',
 'Ocupation_Skilled Manual',
 'House_ownership_0',
 'House_ownership_1',
 'Car_ownership_0',
 'Car_ownership_1',
 'Car_ownership_2',
 'Car_ownership_3',
 'Car_ownership_4',
 'Commute_distance_0-1 Miles',
 'Commute_distance_1-2 Miles',
 'Commute_distance_10+ Miles',
 'Commute_distance_2-5 Miles',
 'Commute_distance_5-10 Miles',
 'Age_group_20s',
 'Age_group_30s',
 'Age_group_40-50s',
 'Age_group_60s or older',
 'Income_group_Low',
 'Income_group_Lower-middle',
 'Income_group_Upper-middle',
 'Income_group_High']
```

In [405…

```
X = encoded_data.drop('Churn',axis = 1)
y = encoded_data.Churn
```

In [406…

```
# Initialize Weight of Evidence
info_value = WOE()
# Fit data
info_value.fit(X, y)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\arraylike.py:358: RuntimeWarning: d
ivide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[406…

```
WOE(mono_custom_binning={'Age': array([ 35.,  45.,  56., 105.]),
```

```
                    'Age_group_20s': array([0.]),
                    'Age_group_30s': array([0., 1.]),
                    'Age_group_40-50s': array([0., 1.]),
                    'Age_group_60s or older': array([0., 1.]),
                    'Car_ownership_0': array([0., 1.]),
                    'Car_ownership_1': array([0., 1.]),
                    'Car_ownership_2': array([0., 1.]),
                    'Car_ownership_3': array([0., 1.]),
                    'Car_ownership_4': array([0., 1....
          'Income_group_High': {0: 0.004786432965288621,
                                1: -0.30455668224836374},
          'Income_group_Low': {0: 0.002118191567904476,
                               1: -0.03126334724868251},
          'Income_group_Lower-middle': {0: -0.05410961609985217,
                                        1: 0.07852604072695553},
          'Income_group_Upper-middle': {0: 0.05201299299191906,
                                        1: -0.04938982662178889},
          'Marital_status_M': {0: 0.03847177558946039,
                               1: -0.03302428814110548}, ...})
```

In [407…

```
# Weight of evidence transformation dataset
info_value.woe_df.head(10)
```

Out[407…

	Variable_Name	Category	Count	Event	Non_Event	Event_Rate	Non_Event_Rate	Event_Distribution	Non_Event_I
0	Age	(34.999, 45.0]	4295	2044	2251	0.48	0.52	0.36	
1	Age	(45.0, 56.0]	3969	1878	2091	0.47	0.53	0.33	
2	Age	(56.0, 105.0]	3811	1745	2066	0.46	0.54	0.31	
3	Age_group_20s	0	12075	5667	6408	0.47	0.53	1.00	
4	Age_group_30s	0	9888	4619	5269	0.47	0.53	0.82	
5	Age_group_30s	1	2187	1048	1139	0.48	0.52	0.18	
6	Age_group_40-50s	0	4910	2314	2596	0.47	0.53	0.41	
7	Age_group_40-50s	1	7165	3353	3812	0.47	0.53	0.59	
8	Age_group_60s or older	0	9352	4401	4951	0.47	0.53	0.78	
9	Age_group_60s or older	1	2723	1266	1457	0.46	0.54	0.22	

In [408…

```
# Information value dataset; IV = (Event% - Non-event%) * WOE
iv_df = info_value.iv_df
iv_df
```

Out[408…

	Variable_Name	Information_Value
55	RFM_cluster_3	1.22
113	RFM_status_Green	0.80
112	RFM_status_Gold	0.70
61	RFM_score_5	0.32

	Variable_Name	Information_Value
64	RFM_score_8	0.18
...	...	...
86	RFM_segment_241	0.00
74	RFM_segment_141	0.00
69	RFM_segment_114	0.00
81	RFM_segment_214	0.00
1	Age_group_20s	0.00

138 rows × 2 columns

In [409…

```
# Rename Variable_Name to index (to be used later in voting section)
iv_df.rename(columns={"Variable_Name":"index"}, inplace=True)
iv_df
```

Out[409…

	index	Information_Value
55	RFM_cluster_3	1.22
113	RFM_status_Green	0.80
112	RFM_status_Gold	0.70
61	RFM_score_5	0.32
64	RFM_score_8	0.18
...	...	...
86	RFM_segment_241	0.00
74	RFM_segment_141	0.00
69	RFM_segment_114	0.00
81	RFM_segment_214	0.00
1	Age_group_20s	0.00

138 rows × 2 columns

Random Forest

In [410…

```
# Initialize Random Forest
rf = RandomForestClassifier(random_state = seed)
# Fit data
rf.fit(X,y)
# Produce predictions
preds = rf.predict(X)
# Calculate accuracy
accuracy = accuracy_score(preds,y)
print(accuracy)
```

```
1.0
```

In [411…

```
# Create a dataframe with variable importance scores
rf_df = pd.DataFrame(rf.feature_importances_, columns = ["RF"], index = X.columns)
```

```
rf_df = rf_df.reset_index()
rf_df.sort_values(['RF'], ascending=0)
```

Out[411…

	index	RF
13	Recency	0.41
91	RFM_cluster_3	0.06
88	RFM_cluster_0	0.05
84	RFM_status_Gold	0.03
71	RFM_segment_442	0.03
...	...	...
37	RFM_segment_141	0.00
130	Age_group_20s	0.00
44	RFM_segment_214	0.00
32	RFM_segment_114	0.00
30	RFM_segment_112	0.00

138 rows × 2 columns

Recursive feature elimination

In [412…

```
log_reg = LogisticRegression(random_state = seed)
rfe = RFE(log_reg, 20)
rfe.fit(X, y)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[412…

```
RFE(estimator=LogisticRegression(random_state=53), n_features_to_select=20)
```

In [413…

```
rfe_df = pd.DataFrame(rfe.support_, columns = ["RFE"], index = X.columns)
rfe_df = rfe_df.reset_index()
rfe_df[rfe_df['RFE'] == True]
```

Out[413…

	index	RFE
50	RFM_segment_242	True
51	RFM_segment_243	True
52	RFM_segment_244	True
58	RFM_segment_323	True
61	RFM_segment_342	True
63	RFM_segment_344	True
66	RFM_segment_421	True
71	RFM_segment_442	True
72	RFM_segment_443	True
73	RFM_segment_444	True
76	RFM_score_5	True
77	RFM_score_6	True
79	RFM_score_8	True
80	RFM_score_9	True
83	RFM_score_12	True
84	RFM_status_Gold	True
87	RFM_status_Green	True
88	RFM_cluster_0	True
90	RFM_cluster_2	True
91	RFM_cluster_3	True

Extra Trees

In [414…

```
etc = ExtraTreesClassifier(random_state = seed)
etc.fit(X, y)
etc_df = pd.DataFrame(etc.feature_importances_, columns = ["Extra_trees"], index = X.colur
etc_df = etc_df.reset_index()
etc_df.sort_values(['Extra_trees'], ascending=0)
```

Out[414…

	index	Extra_trees
13	Recency	0.14
91	RFM_cluster_3	0.08
88	RFM_cluster_0	0.05
84	RFM_status_Gold	0.04
87	RFM_status_Green	0.04
...	...	...

	index	Extra_trees
30	RFM_segment_112	0.00
37	RFM_segment_141	0.00
44	RFM_segment_214	0.00
130	Age_group_20s	0.00
32	RFM_segment_114	0.00

138 rows × 2 columns

Chi Square

In [415…

```
kbest = SelectKBest(score_func=chi2, k=5)
chi_sq = kbest.fit(X, y)
pd.options.display.float_format = '{:.2f}'.format
chi_sq_df = pd.DataFrame(chi_sq.scores_, columns = ["Chi_square"], index = X.columns)
chi_sq_df = chi_sq_df.reset_index()
chi_sq_df.sort_values('Chi_square', ascending=0)
```

Out[415…

	index	Chi_square
13	Recency	691911.00
17	Yearly_income	348666.54
6	Revenue_min	115063.21
5	Revenue_median	79755.85
4	Revenue_mean	76573.83
...	...	...
96	Total_children_0	0.03
120	Car_ownership_0	0.03
110	Education_High School	0.02
115	Ocupation_Manual	0.01
130	Age_group_20s	NaN

138 rows × 2 columns

L1

In [416…

```
lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, y)
l1 = SelectFromModel(lsvc,prefit=True)
l1_df = pd.DataFrame(l1.get_support(), columns = ["L1"], index = X.columns)
l1_df = l1_df.reset_index()
l1_df[l1_df['L1'] == True]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: L
iblinear failed to converge, increase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
```

Out[416…

	index	L1
1	Quantity_mean	True

	index	L1
4	Revenue_mean	True
5	Revenue_median	True
6	Revenue_min	True
7	Revenue_max	True
8	Profit_sum	True
9	Profit_mean	True
10	Profit_median	True
11	Profit_min	True
12	Profit_max	True
13	Recency	True
14	Frequency	True
15	Monetary	True
16	Tenure_months	True
18	Age	True
96	Total_children_0	True
120	Car_ownership_0	True
131	Age_group_30s	True
132	Age_group_40-50s	True
135	Income_group_Lower-middle	True

Feature voting

In [417…

```
# Combine altogether
dfs = [iv_df, rf_df, rfe_df, etc_df, chi_sq_df, l1_df]
summary = reduce(lambda left,right: pd.merge(left,right,on='index'), dfs)
summary.head()
```

Out[417…

	index	Information_Value	RF	RFE	Extra_trees	Chi_square	L1
0	RFM_cluster_3	1.22	0.06	True	0.08	1691.33	False
1	RFM_status_Green	0.80	0.03	True	0.04	1582.95	False
2	RFM_status_Gold	0.70	0.03	True	0.04	1295.59	False
3	RFM_score_5	0.32	0.01	True	0.01	549.78	False
4	RFM_score_8	0.18	0.01	True	0.01	407.69	False

In [418…

```
# Calculate scores
# Filter columns with non-binary values
columns = ['Information_Value', 'RF', 'Extra_trees', 'Chi_square']

score_table = pd.DataFrame({},[])
score_table['index'] = summary['index']

# Assign 1 if the score is in the top 5, else 0
```

```
for i in columns:
    score_table[i] = summary['index'].isin(list(summary.nlargest(5,i)['index'])).astype(in

# Convert True to 1 and False to 0
score_table['RFE'] = summary['RFE'].astype(int)
score_table['L1'] = summary['L1'].astype(int)
```

In [419…

```
score_table['Final_score'] = score_table.sum(axis=1)
score_table.sort_values('Final_score',ascending=0)
```

Out[419…

	index	Information_Value	RF	Extra_trees	Chi_square	RFE	L1	Final_score
0	RFM_cluster_3	1	1	1	0	1	0	4
2	RFM_status_Gold	1	1	1	0	1	0	4
30	Recency	0	1	1	1	0	1	4
7	RFM_cluster_0	0	1	1	0	1	0	3
1	RFM_status_Green	1	0	1	0	1	0	3
...	...	...	...	...	...	...	...	...
64	RFM_segment_143	0	0	0	0	0	0	0
63	Number_children_at_home_5	0	0	0	0	0	0	0
61	Total_children_1	0	0	0	0	0	0	0
60	Car_ownership_1	0	0	0	0	0	0	0
137	Age_group_20s	0	0	0	0	0	0	0

138 rows × 8 columns

Multicollinearity check

In [420…

```
# Filter variables with score >= 2
select_var = X[list(score_table[score_table['Final_score'] >= 2]['index'])]
```

In [421…

```
def calculate_vif(features):
    vif = pd.DataFrame()
    vif["Features"] = features.columns
    vif["VIF"] = [variance_inflation_factor(features.values, i) for i in range(features.sh
    return(vif)

vif = calculate_vif(select_var)
vif
```

Out[421…

	Features	VIF
0	RFM_cluster_3	5.39
1	RFM_status_Green	2.89
2	RFM_status_Gold	2.25
3	RFM_score_5	1.47
4	RFM_score_8	1.20
5	RFM_cluster_0	1.23

	Features	VIF
6	RFM_segment_442	1.63
7	Revenue_min	27.09
8	Recency	6.01
9	Revenue_median	656.36
10	Revenue_mean	712.59

In [422…

```
# Narrow down the features until their VIF is equal to or lower than 5
while vif['VIF'][vif['VIF'] > 5].any():
    remove = vif.sort_values('VIF',ascending=0)['Features'][:1]
    select_var.drop(remove,axis=1,inplace=True)
    vif = calculate_vif(select_var)

vif
```

Out[422…

	Features	VIF
0	RFM_cluster_3	3.70
1	RFM_status_Green	2.76
2	RFM_status_Gold	1.80
3	RFM_score_5	1.45
4	RFM_score_8	1.19
5	RFM_cluster_0	1.21
6	RFM_segment_442	1.62
7	Revenue_min	1.91

In [423…

```
final_features = vif['Features']
```

In [424…

```
# Create the final dataframe with all selected features and label
final_var = list(vif['Features']) + ['Churn']
final_df = encoded_data[final_var]
final_df.head()
```

Out[424…

	RFM_cluster_3	RFM_status_Green	RFM_status_Gold	RFM_score_5	RFM_score_8	RFM_cluster_0	RFM_segment_442
0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0
3	1	0	0	0	0	0	0
4	0	0	0	0	0	1	0

In [425…

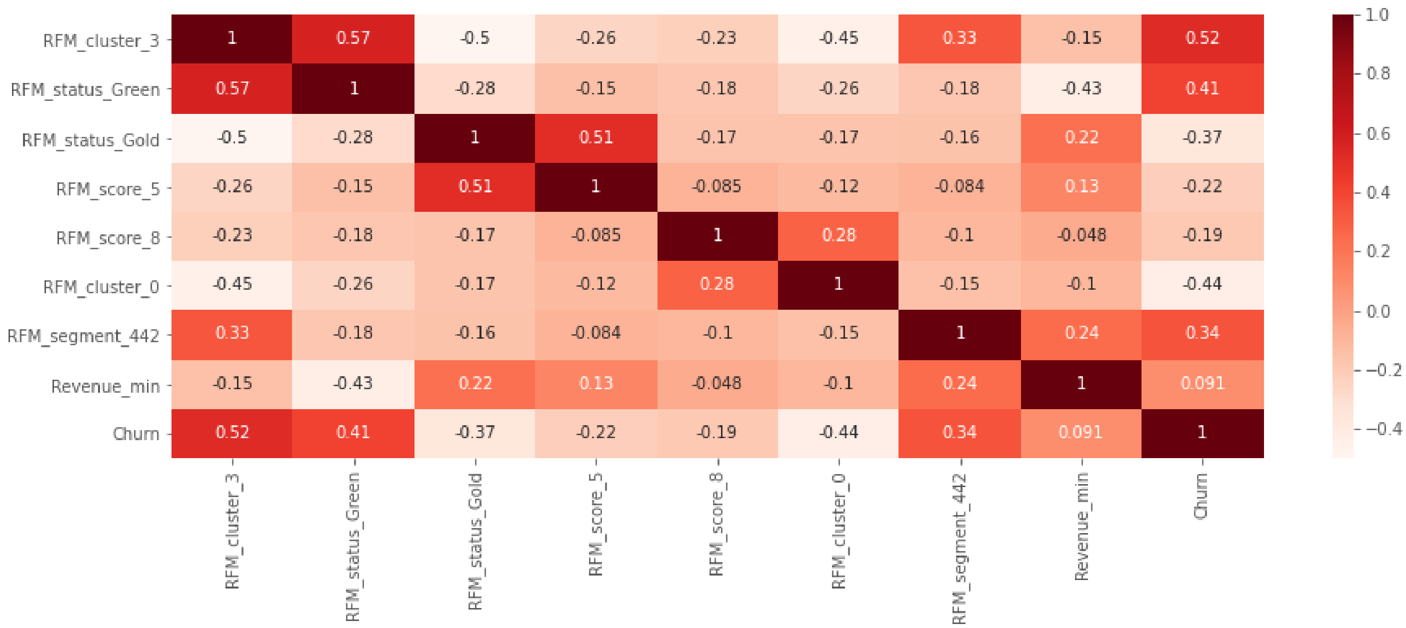
```
# Final check for correlation between variables
corr = final_df.corr()
corr
```

Out[425…

	RFM_cluster_3	RFM_status_Green	RFM_status_Gold	RFM_score_5	RFM_score_8	RFM_cluster_0	R
RFM_cluster_3	1.00	0.57	-0.50	-0.26	-0.23	-0.45	
RFM_status_Green	0.57	1.00	-0.28	-0.15	-0.18	-0.26	
RFM_status_Gold	-0.50	-0.28	1.00	0.51	-0.17	-0.17	
RFM_score_5	-0.26	-0.15	0.51	1.00	-0.09	-0.12	
RFM_score_8	-0.23	-0.18	-0.17	-0.09	1.00	0.28	
RFM_cluster_0	-0.45	-0.26	-0.17	-0.12	0.28	1.00	
RFM_segment_442	0.33	-0.18	-0.16	-0.08	-0.10	-0.15	
Revenue_min	-0.15	-0.43	0.22	0.13	-0.05	-0.10	
Churn	0.52	0.41	-0.37	-0.22	-0.19	-0.44	

In [426…

```
# Plot heatmap
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, annot=True, cmap=plt
pass
```



Model selection

In [427…

```
# Split dataset into train/test
X = final_df.loc[:, final_df.columns != 'Churn']
y = final_df['Churn']
test_pct = 0.3
# No need to add stratify parameter - the dataset is balanced
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size = test_pct, random_stat

print(len(X_train), len(y_train))
```

```
8452 8452
```

Decision Tree

In [428…

```
# Fit the training set in the Decsion Tree
tree = DecisionTreeClassifier(random_state = seed)
tree.fit(X_train, y_train)
```

```
# Predict train set
train_pred = tree.predict(X_train)
# Predict test set
test_pred= tree.predict(X_test)

# Calculate scores
train_acc = 100*tree.score(X_train, y_train)
test_acc = 100*tree.score(X_test, y_test)
train_f1 = 100*fbeta_score(y_train, train_pred, beta = 1)
test_f1 = 100*fbeta_score(y_test, test_pred, beta = 1)

# Gather the results in a dataframe
# Create a list with the metric names
metrics = ['train_accuracy', 'test_accuracy', 'train_f1_score', 'test_f1_score']
# Create an array with the score values
scores = np.array([train_acc, test_acc, train_f1, test_f1])
# Create the dataframe
pd.options.display.float_format = '{:.2f}'.format
DT_df = pd.DataFrame(scores, columns = ['Decision_tree'], index = metrics).reset_index()
DT_df
```

Out[428...

	index	Decision_tree
0	train_accuracy	90.71
1	test_accuracy	87.47
2	train_f1_score	90.15
3	test_f1_score	86.44

Random Forest

In [429...

```
# Train the model with the data
rf = RandomForestClassifier (random_state = seed)
rf.fit(X_train, y_train)

# Predict train and test sets
train_pred = rf.predict(X_train)
test_pred = rf.predict(X_test)

train_acc = 100*rf.score(X_train, y_train)
test_acc = 100*rf.score(X_test, y_test)
train_f1 = 100*fbeta_score(y_train, train_pred, beta = 1)
test_f1 = 100*fbeta_score(y_test, test_pred, beta = 1)

# Create the dataframe
scores = np.array([train_acc, test_acc, train_f1, test_f1])
RF_df = pd.DataFrame(scores, columns = ['Random_forest'], index = metrics).reset_index()
RF_df
```

Out[429...

	index	Random_forest
0	train_accuracy	90.71
1	test_accuracy	86.97
2	train_f1_score	90.29
3	test_f1_score	86.11

Extra Trees

In [430…

```
etc = ExtraTreesClassifier(random_state = seed)
etc.fit(X_train, y_train)

train_pred = etc.predict(X_train)
test_pred = etc.predict(X_test)

train_acc = 100*etc.score(X_train, y_train)
test_acc = 100*etc.score(X_test, y_test)
train_f1 = 100*fbeta_score(y_train, train_pred, beta = 1)
test_f1 = 100*fbeta_score(y_test, test_pred, beta = 1)

scores = np.array([train_acc, test_acc, train_f1, test_f1])
ETC_df = pd.DataFrame(scores, columns = ['Extra_trees'], index = metrics).reset_index()
ETC_df
```

Out[430…

	index	Extra_trees
0	train_accuracy	90.71
1	test_accuracy	87.41
2	train_f1_score	90.15
3	test_f1_score	86.40

Gradient Boosting

In [431…

```
gb = GradientBoostingClassifier(random_state = seed)
gb.fit(X_train, y_train)

train_pred = gb.predict(X_train)
test_pred = gb.predict(X_test)

train_acc = 100*gb.score(X_train, y_train)
test_acc = 100*gb.score(X_test, y_test)
train_f1 = 100*fbeta_score(y_train, train_pred, beta = 1)
test_f1 = 100*fbeta_score(y_test, test_pred, beta = 1)

scores = np.array([train_acc, test_acc, train_f1, test_f1])
GB_df = pd.DataFrame(scores, columns = ['Gradient_boosting'], index = metrics).reset_index
GB_df
```

Out[431…

	index	Gradient_boosting
0	train_accuracy	88.35
1	test_accuracy	88.82
2	train_f1_score	87.42
3	test_f1_score	87.64

Logistic Regression

In [432…

```
# Scale data
scaler = MinMaxScaler(feature_range=(-1, 1))
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
scaler.fit(X_test)
X_test_scaled = scaler.transform(X_test)
```

In [433…

```
logreg = LogisticRegression(random_state = seed).fit(X_train_scaled, y_train)

train_pred = logreg.predict(X_train_scaled)
test_pred = logreg.predict(X_test_scaled)

train_acc = 100*logreg.score(X_train_scaled, y_train)
test_acc = 100*logreg.score(X_test_scaled, y_test)
train_f1 = 100*fbeta_score(y_train, train_pred, beta = 1)
test_f1 = 100*fbeta_score(y_test, test_pred, beta = 1)

scores = np.array([train_acc, test_acc, train_f1, test_f1])
LR_df = pd.DataFrame(scores, columns = ['Logistic_regression'], index = metrics).reset_inc
LR_df
```

Out[433…

	index	Logistic_regression
0	train_accuracy	84.44
1	test_accuracy	84.24
2	train_f1_score	83.84
3	test_f1_score	83.29

Linear Discriminant Analysis

In [434…

```
lda = LinearDiscriminantAnalysis()
# Use scaled data
lda.fit(X_train_scaled, y_train)

train_pred = lda.predict(X_train_scaled)
test_pred = lda.predict(X_test_scaled)

train_acc = 100*lda.score(X_train_scaled, y_train)
test_acc = 100*lda.score(X_test_scaled, y_test)
train_f1 = 100*fbeta_score(y_train, train_pred, beta = 1)
test_f1 = 100*fbeta_score(y_test, test_pred, beta = 1)

scores = np.array([train_acc, test_acc, train_f1, test_f1])
LDA_df = pd.DataFrame(scores, columns = ['LDA'], index = metrics).reset_index()
LDA_df
```

Out[434…

	index	LDA
0	train_accuracy	84.55
1	test_accuracy	84.57
2	train_f1_score	83.73
3	test_f1_score	83.38

K Nearest Neighbors

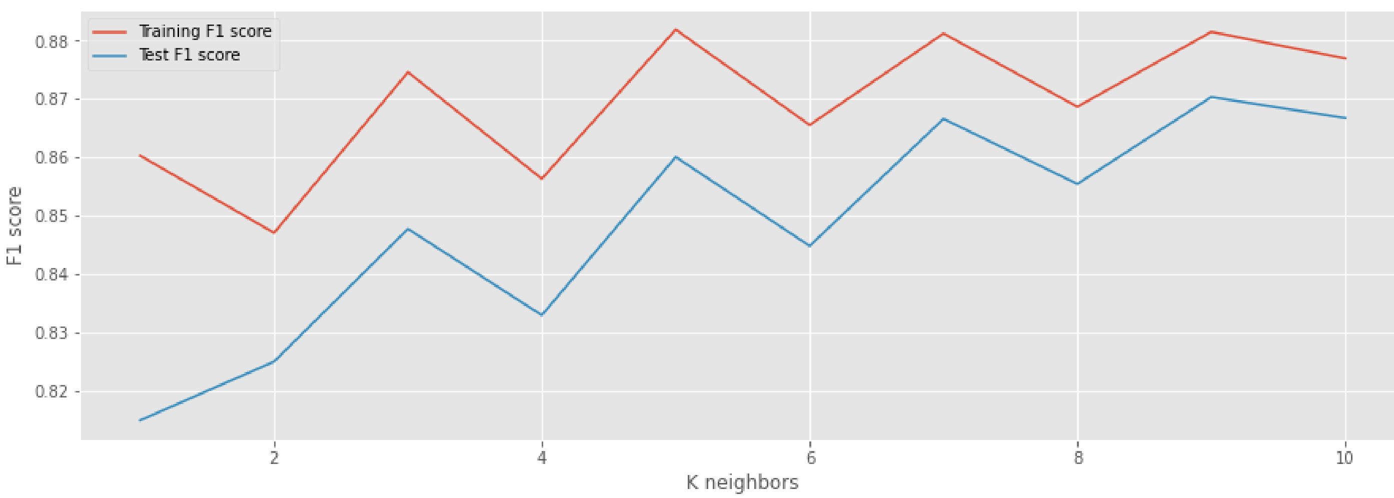
In [435…

```
# Select k
training_F1 = []
test_F1 = []

# Try n_neighbours from 1 to 10
neighbors_settings = range(1, 11)
```

```python
    # Run the model for different n_neighbours
    for k in neighbors_settings:
        # Build the model
        knn = KNeighborsClassifier(n_neighbors = k)
        # Use scaled data
        knn.fit(X_train_scaled, y_train)

        # Predict train set
        train_pred = knn.predict(X_train_scaled)
        # Predict test set
        test_pred = knn.predict(X_test_scaled)

        # Record training set F1 score
        training_F1.append(fbeta_score(y_train, train_pred, beta = 1))
        # Record test set F1 score - More important than training accuracy
        test_F1.append(fbeta_score(y_test, test_pred, beta = 1))
```

In [436…

```python
# Plot training and test F1 score
plt.plot(neighbors_settings, training_F1, label = "Training F1 score")
plt.plot(neighbors_settings, test_F1, label = "Test F1 score")
plt.ylabel("F1 score")
plt.xlabel("K neighbors")
plt.legend()
pass
```



In [437…

```python
# I chose k = 6 where fluctuation flattens and F1 score on sets doesn't vary significantly
knn = KNeighborsClassifier(n_neighbors = 6)
knn.fit(X_train_scaled, y_train)

train_pred = knn.predict(X_train_scaled)
test_pred = knn.predict(X_test_scaled)

train_acc = 100*knn.score(X_train_scaled, y_train)
test_acc = 100*knn.score(X_test_scaled, y_test)
train_f1 = 100*fbeta_score(y_train, train_pred, beta = 1)
test_f1 = 100*fbeta_score(y_test, test_pred, beta = 1)

scores = np.array([train_acc, test_acc, train_f1, test_f1])
KNN_df = pd.DataFrame(scores, columns = ['KNN'], index = metrics).reset_index()
KNN_df
```

Out[437…

	index	KNN
0	train_accuracy	87.85
1	test_accuracy	86.28

	index	KNN
2	train_f1_score	86.56
3	test_f1_score	84.48

Neural Network

In [438…

```
mlp = MLPClassifier(random_state = seed)
mlp.fit(X_train, y_train)

train_pred = mlp.predict(X_train)
test_pred = mlp.predict(X_test)

train_acc = 100*mlp.score(X_train, y_train)
test_acc = 100*mlp.score(X_test, y_test)
train_f1 = 100*fbeta_score(y_train, train_pred, beta = 1)
test_f1 = 100*fbeta_score(y_test, test_pred, beta = 1)

scores = np.array([train_acc, test_acc, train_f1, test_f1])
NN_df = pd.DataFrame(scores, columns = ['Neural_network'], index = metrics).reset_index()
NN_df
```

Out[438…

	index	Neural_network
0	train_accuracy	82.77
1	test_accuracy	82.56
2	train_f1_score	83.38
3	test_f1_score	82.80

Naive Bayes

In [439…

```
gnb = GaussianNB()
gnb.fit(X_train, y_train)

train_pred = gnb.predict(X_train)
test_pred = gnb.predict(X_test)

train_acc = 100*gnb.score(X_train, y_train)
test_acc = 100*gnb.score(X_test, y_test)
train_f1 = 100*fbeta_score(y_train, train_pred, beta = 1)
test_f1 = 100*fbeta_score(y_test, test_pred, beta = 1)

scores = np.array([train_acc, test_acc, train_f1, test_f1])
NB_df = pd.DataFrame(scores, columns = ['Naive_bayes'], index = metrics).reset_index()
NB_df
```

Out[439…

	index	Naive_bayes
0	train_accuracy	81.34
1	test_accuracy	80.49
2	train_f1_score	82.29
3	test_f1_score	81.22

In [440…

```
### Support Vector Machine
```

In [441…

```
svc = SVC(random_state = seed)
# Use scaled data
svc.fit(X_train_scaled, y_train)

train_pred = svc.predict(X_train_scaled)
test_pred = svc.predict(X_test_scaled)

train_acc = 100*svc.score(X_train_scaled, y_train)
test_acc = 100*svc.score(X_test_scaled, y_test)
train_f1 = 100*fbeta_score(y_train, train_pred, beta = 1)
test_f1 = 100*fbeta_score(y_test, test_pred, beta = 1)

scores = np.array([train_acc, test_acc, train_f1, test_f1])
SVM_df = pd.DataFrame(scores, columns = ['SVM'], index = metrics).reset_index()
SVM_df
```

Out[441…

	index	SVM
0	train_accuracy	88.23
1	test_accuracy	88.57
2	train_f1_score	87.16
3	test_f1_score	87.22

champion model

In [442…

```
# Create a dataframe with scores
# Combine altogether
dfs = [DT_df, RF_df, ETC_df, GB_df, LR_df, LDA_df, KNN_df, NN_df, NB_df, SVM_df]
summary = reduce(lambda left,right: pd.merge(left, right, on = 'index'), dfs)
summary
```

Out[442…

	index	Decision_tree	Random_forest	Extra_trees	Gradient_boosting	Logistic_regression	LDA	KNN	Neu
0	train_accuracy	90.71	90.71	90.71	88.35	84.44	84.55	87.85	
1	test_accuracy	87.47	86.97	87.41	88.82	84.24	84.57	86.28	
2	train_f1_score	90.15	90.29	90.15	87.42	83.84	83.73	86.56	
3	test_f1_score	86.44	86.11	86.40	87.64	83.29	83.38	84.48	

In [443…

```
# Filter F1 scores
f1_summary = summary.iloc[3:,:]
f1_summary = f1_summary.transpose()
f1_summary.columns = f1_summary.iloc[0]
f1_summary.drop(f1_summary.index[0], inplace = True)
f1_summary = f1_summary.sort_values(["test_f1_score"],ascending=0)
f1_summary
```

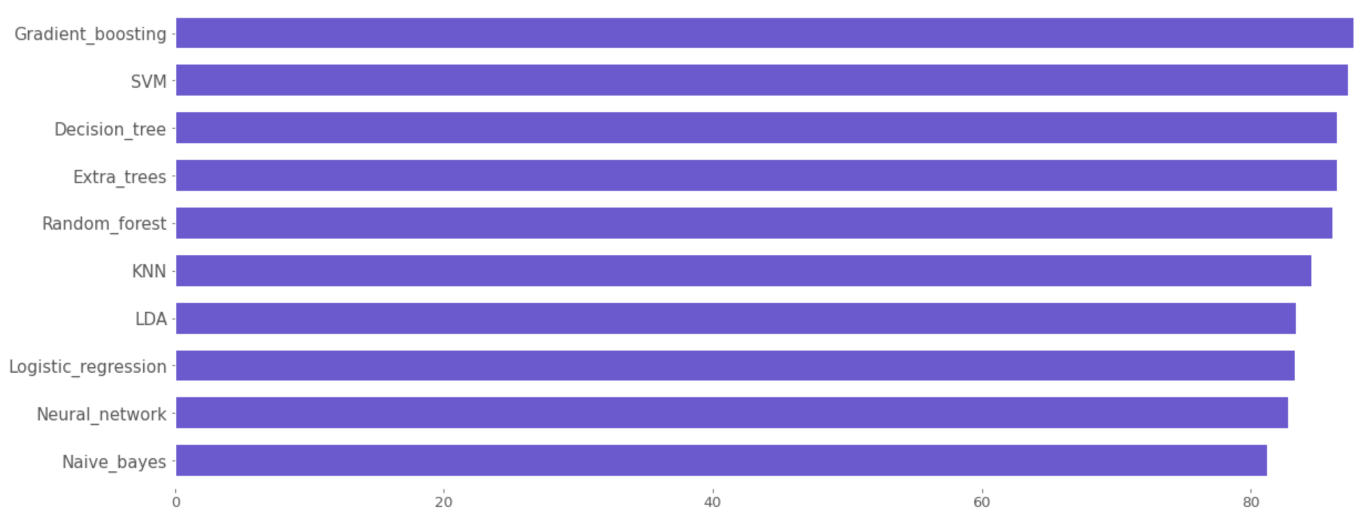
Out[443…

index	test_f1_score
Gradient_boosting	87.64
SVM	87.22
Decision_tree	86.44

index	test_f1_score
Extra_trees	86.40
Random_forest	86.11
KNN	84.48
LDA	83.38
Logistic_regression	83.29
Neural_network	82.80
Naive_bayes	81.22

In [444…

```
# PLot F1 scores
chart = f1_summary.plot(y ='test_f1_score',
                        kind = 'barh', color ='slateblue',
                        legend = False, width = 0.65, figsize=(20,8))
chart.set_facecolor('white')
chart.set_ylabel('')
chart.invert_yaxis()
plt.yticks(fontsize=15)
plt.xticks(fontsize=13)
pass
```



In [445…

```
# Gradient Boosting is the champion model!
```

Hyperparameter tuning

Validation curves

In [446…

```
#### Ideally, we would want both the validation curve and the training curve to look as si
#### If both scores are low, the model is likely to be underfitting. This means either the
#### If the training curve reaches a high score relatively quickly and the validation curv
#### We would want the value of the parameter where the training and validation curves are
```

In [447…

```
# Plot validation curve for various parameters to estimate their optimal values
# This will help us narrow down the potential values and therefore reduce running time of
# Check out current parametrers
print(gb.get_params())
```

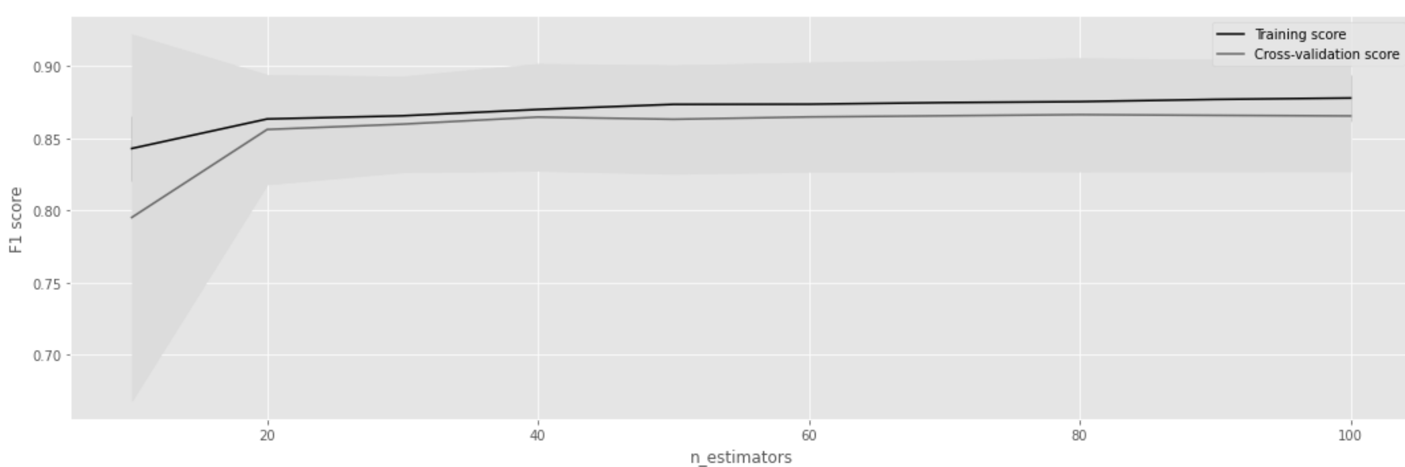
```
{'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0.1, 'los
s': 'deviance', 'max_depth': 3, 'max_features': None, 'max_leaf_nodes': None, 'min_impurit
y_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split':
2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_iter_no_change': None, 'random
_state': 53, 'subsample': 1.0, 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': 0, 'w
arm_start': False}
```

In [448…

```
# n_estimators parameter
# Create a list with the range of values for n_estimators
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 100, num = 10)]
# Obtain train and test scores from validation curve
# Use all data for cross validation - no need to split into train and test sets
train_scores, test_scores = validation_curve(gb, X, y,
                                             param_name = "n_estimators",
                                             param_range = n_estimators,
                                             cv = 3, scoring= "f1",
                                             n_jobs= -1)
```

In [449…

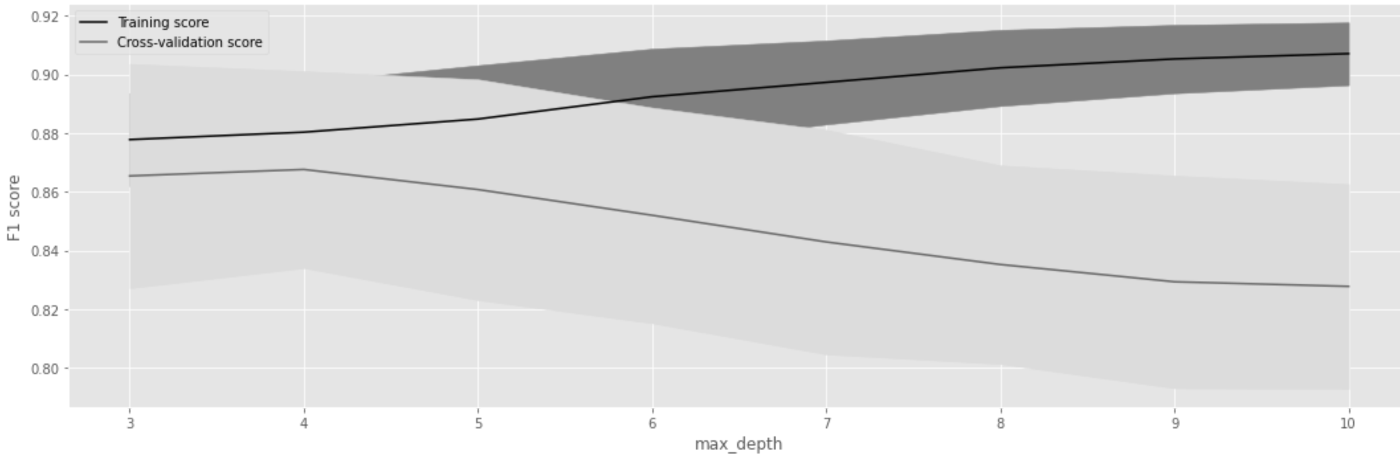
```
def validation_curve_plot(parameter, values, train_scores, test_scores):
    '''Calculate mean and standard deviation for training set scores'''
    train_mean = np.mean(train_scores, axis = 1)
    train_std = np.std(train_scores, axis = 1)

    '''Calculate mean and standard deviation for test set scores'''
    test_mean = np.mean(test_scores, axis = 1)
    test_std = np.std(test_scores, axis = 1)

    '''Plot mean accuracy scores for training and test sets'''
    plt.plot(values, train_mean, label="Training score", color="black")
    plt.plot(values, test_mean, label="Cross-validation score", color="dimgrey")

    '''Plot accurancy bands for training and test sets'''
    plt.fill_between(values, train_mean - train_std, train_mean + train_std, color="gray")
    plt.fill_between(values, test_mean - test_std, test_mean + test_std, color="gainsboro'

    '''Create plot'''
    plt.xlabel(parameter)
    plt.ylabel("F1 score")
    plt.tight_layout()
    plt.legend(loc="best")

# Call function to plot validation curve
validation_curve_plot("n_estimators", n_estimators, train_scores, test_scores)
```



In [450…

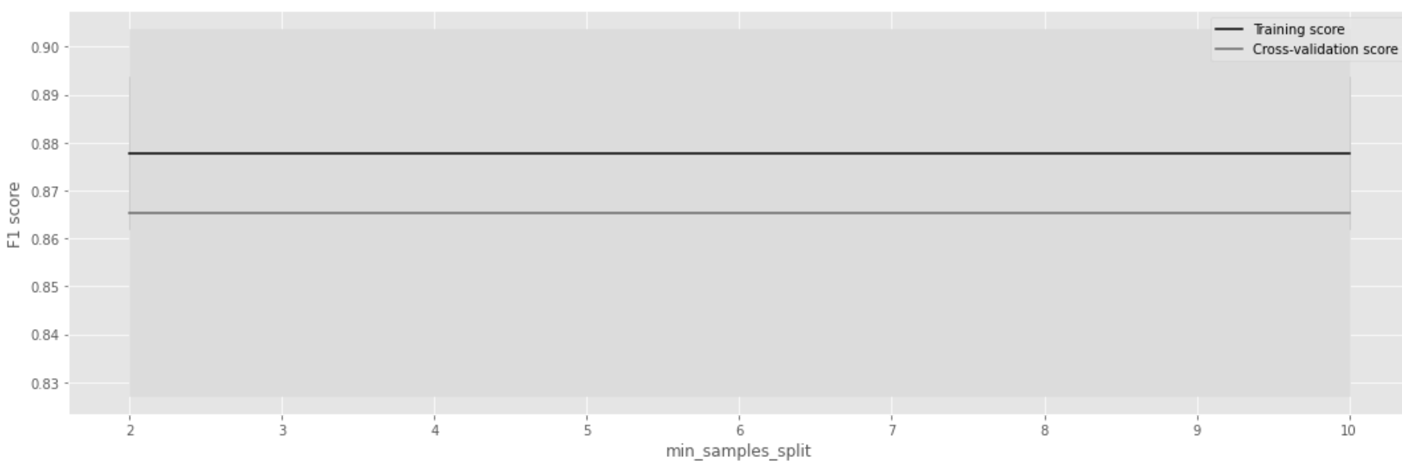
```
max_depth = [int(x) for x in np.linspace(start = 3, stop = 10, num = 10)]

train_scores, test_scores = validation_curve(gb, X, y,
```

```
                                                 param_name = "max_depth",
                                                 param_range = max_depth,
                                                 cv = 3, scoring= "f1",
                                                 n_jobs= -1)

validation_curve_plot("max_depth", max_depth, train_scores, test_scores)
```



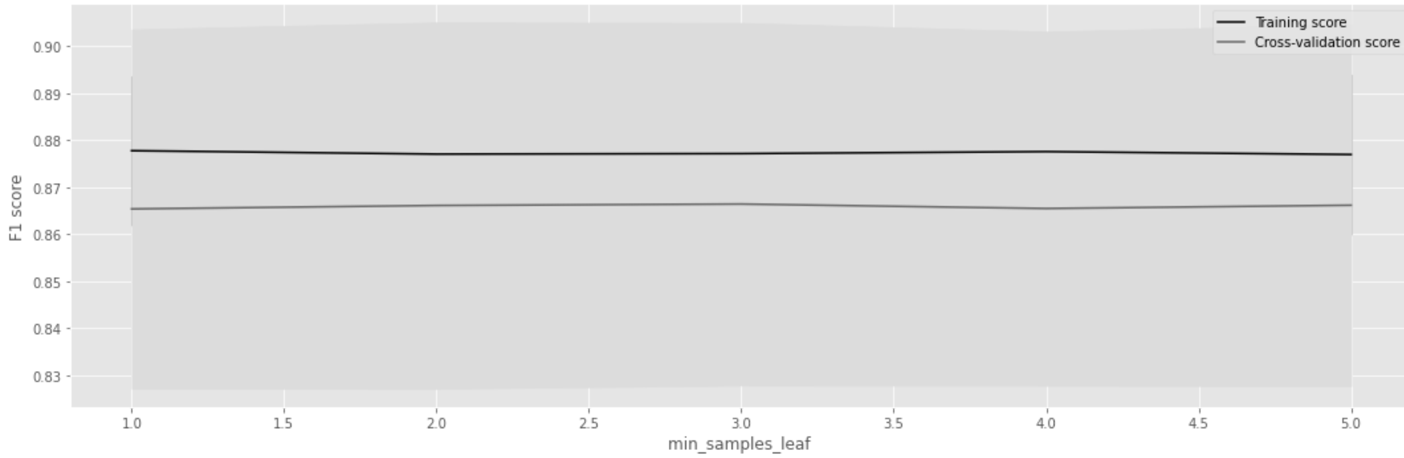
In [451…

```
min_samples_split = [int(x) for x in np.linspace(start = 2, stop = 10, num = 10)]

train_scores, test_scores = validation_curve(gb, X, y,
                                                 param_name = "min_samples_split",
                                                 param_range = min_samples_split,
                                                 cv = 3, scoring= "f1",
                                                 n_jobs= -1)

validation_curve_plot("min_samples_split", min_samples_split, train_scores, test_scores)
```



In [452…

```
min_samples_leaf = [int(x) for x in np.linspace(start = 1, stop = 5, num = 10)]

train_scores, test_scores = validation_curve(gb, X, y,
                                                 param_name = "min_samples_leaf",
                                                 param_range = min_samples_leaf,
                                                 cv = 3, scoring= "f1",
                                                 n_jobs= -1)

validation_curve_plot("min_samples_leaf ", min_samples_leaf, train_scores, test_scores)
```



Grid search cross validation

In [453…

```
# Create lists with potential optimal values for each parameter
n_estimators = [20, 40, 100]
max_features = ['auto', 'sqrt']
max_depth = [3, 4]
min_samples_split = [2, 10]
min_samples_leaf = [1, 5]
random_state = [seed]

# Create a dictionary with the range of parameter values
grid = {'n_estimators': n_estimators,
        'max_features': max_features,
        'max_depth': max_depth,
        'min_samples_split': min_samples_split,
        'min_samples_leaf': min_samples_leaf,
        'random_state': random_state}

# Define cross-validation method
cv_method = StratifiedKFold(n_splits = 3, random_state = seed, shuffle = True)

# Intialize Grid Search model
gs = GridSearchCV(estimator = gb, param_grid = grid, cv = cv_method,
                  scoring = 'f1', verbose = 2, n_jobs = -1)

# Train model with data
gs.fit(X, y)

# Print optimal parameter values after tuning
print(gs.best_params_)
```

```
Fitting 3 folds for each of 48 candidates, totalling 144 fits
{'max_depth': 4, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 10,
'n_estimators': 100, 'random_state': 53}
```

In [454…

```
# print how our model looks after hyper-parameter tuning
print(gs.best_estimator_)
```

```
GradientBoostingClassifier(max_depth=4, max_features='sqrt',
                           min_samples_split=10, random_state=53)
```

In [455…

```
## Model evaluation
```

In [456…

```
### Classification report
```

In [457…
```
# Print the score of the best estimator
best_score = gs.best_score_ * 100
print("Best mean test score: {:.2f}%".format(best_score))
```

```
Best mean test score: 87.53%
```

In [458…
```
# Calculate improvement of Grid search model on baseline
baseline = f1_summary.loc['Gradient_boosting','test_f1_score']
print('Improvement on baseline model of {:0.2f}%'.format(100 * (best_score - baseline) / b
```

```
Improvement on baseline model of -0.13%
```

In [459…
```
# Make predictions
gs_pred = gs.best_estimator_.predict(X)
# Print classification report
print(classification_report(y, gs_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.91      0.89      6408
           1       0.89      0.86      0.88      5667

    accuracy                           0.89     12075
   macro avg       0.89      0.88      0.89     12075
weighted avg       0.89      0.89      0.89     12075
```

In [460…
```
# very high accuracy and f1 score
```

Confusion matrix

In [461…
```
confusion_df= pd.crosstab(y,pd.Series(gs_pred),rownames=['Actual'],colnames=['Pred'])
confusion_df
```

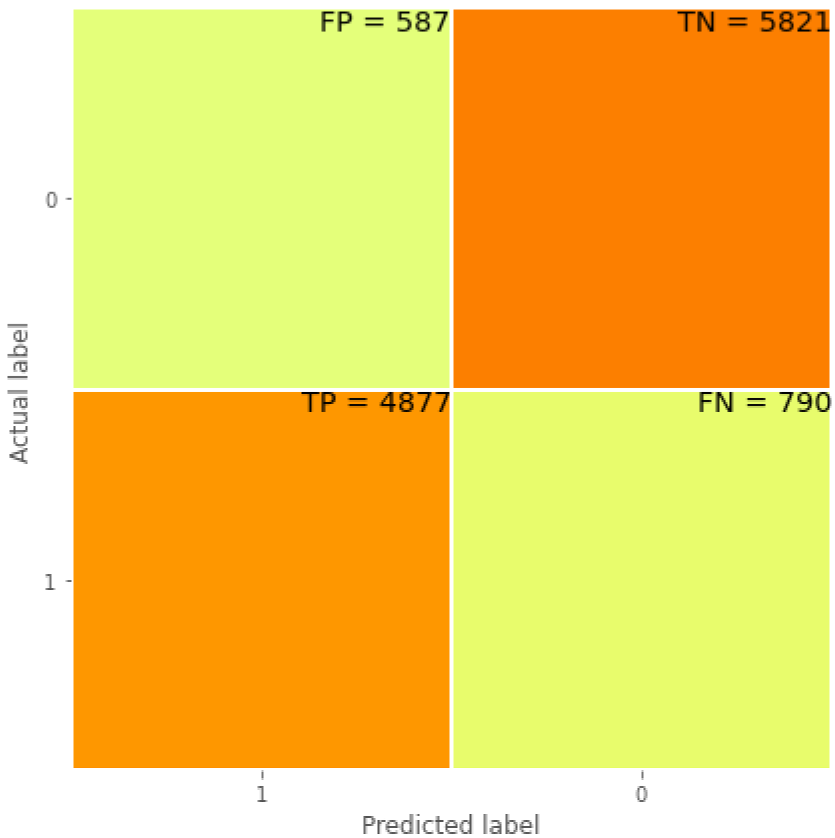
Out[461…

Pred	0	1
Actual		
0	5821	587
1	790	4877

In [462…
```
# Define elements of confusion matrix for later use in A/B testing
tp = confusion_df.loc[1,1]
tn = confusion_df.loc[0,0]
fp = confusion_df.loc[0,1]
fn = confusion_df.loc[1,0]
```

In [463…
```
# Plot a heatmap of the confusion matrix
con_mat = confusion_matrix(y, gs_pred)
con_mat

colormap = plt.cm.Wistia
fig, ax = plt.subplots(figsize=(7,7))
ax = sns.heatmap(con_mat,cmap=colormap,linewidths=0.1,linecolor='white',annot = False, cba

ax.set_ylim(2, 0)
ax.set_xlim(2, 0)
ax.set_aspect("equal")
```

```
plt.yticks(rotation=0)

# Set y and x label
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Insert text in each cell of matrix
s = [['TN','FP'],
     ['FN', 'TP']]
for i in range(2):
    for j in range(2):
        plt.text(j,i, str(s[i][j])+" = "+str(con_mat[i][j]), ha="right", va ="top", size='
```



In [464…

```
con_mat = confusion_matrix(y, gs_pred)
```

In [465…

```
# Save model
final_model = gs.best_estimator_
filename = 'final_churning_model'
joblib.dump(final_model, filename)
```

Out[465…

```
['final_churning_model']
```

A/B testing

Maximum profit

In [466…

```
# Calculate average revenue per customer for a year
# Find latest date of transaction data
max(transaction_df['Order_date'])
```

Out[466…

```
Timestamp('2014-01-28 00:00:00')
```

In [467…

```
# Calculate average revenue per customer for a year
# Find least date of transaction data
min(transaction_df['Order_date'])
```

Out[467…

```
Timestamp('2010-12-29 00:00:00')
```

In [468…

```
# Summarise transaction sales by Order Date
daily_sales = transaction_df.groupby('Order_date').agg({'Revenue':'sum'})
daily_sales.head()
```

Out[468…

	Revenue
Order_date	
2010-12-29	14477.34
2010-12-30	13931.52
2010-12-31	15012.18
2011-01-01	7156.54
2011-01-02	15012.18

In [469…

```
# Summarise data by monthly sales
monthly_sales = daily_sales.resample('MS').sum()
monthly_sales.head()
```

Out[469…

	Revenue
Order_date	
2010-12-01	43421.04
2011-01-01	469823.91
2011-02-01	466334.90
2011-03-01	485198.66
2011-04-01	502073.85

In [470…

```
# get yearly sales
yearly_sales = daily_sales.resample('Y').sum()
yearly_sales.head()
```

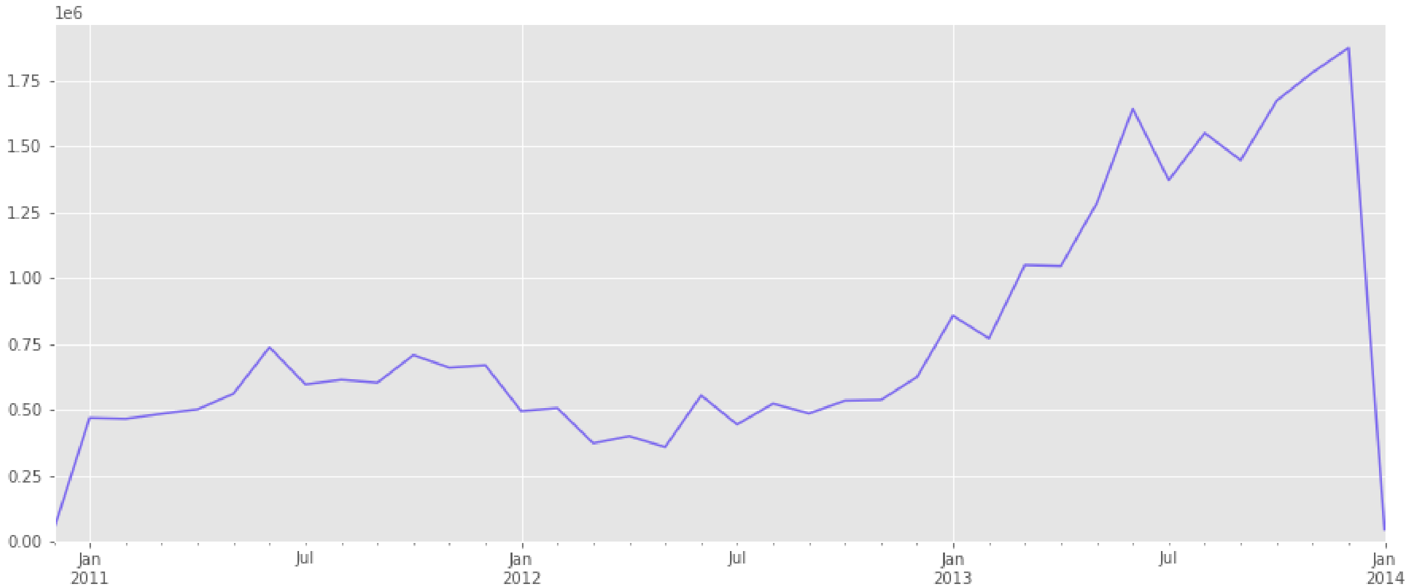
Out[470…

	Revenue
Order_date	
2010-12-31	43421.04
2011-12-31	7075525.93
2012-12-31	5842485.20
2013-12-31	16351550.34
2014-12-31	45694.72

In [471…

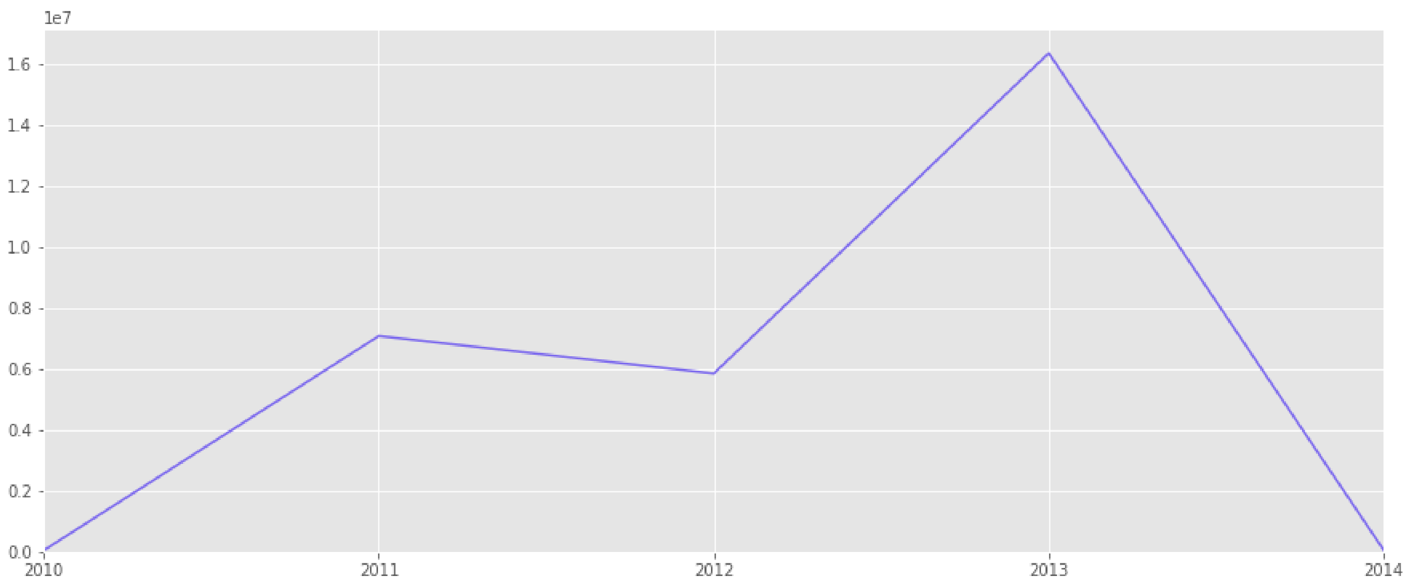
```
# Plot transactional data on a monthly basis
ax = monthly_sales['Revenue'].plot(figsize=(15, 6), color = 'mediumslateblue')
```

```
ax.xaxis.set_label_text("")
ax.set_ylim(ymin=0)
pass
```



In [472…

```
# Plot transactional data on a monthly basis
ax = yearly_sales['Revenue'].plot(figsize=(15, 6), color = 'mediumslateblue')
ax.xaxis.set_label_text("")
ax.set_ylim(ymin=0)
pass
```



In [473…

```
# we have peak revenue in 2013
```

In [474…

```
# Calculate revenue per customer for each of the 3 full years
transaction_df = transaction_df.set_index('Order_date')
df1 = transaction_df['2011-01-01' : '2011-12-31']
df2 = transaction_df['2012-01-01' : '2012-12-31']
df3 = transaction_df['2013-01-01' : '2013-12-31']
df_list = [df1, df2, df3]

rev_list = []
for df in df_list:
    rev = sum(df.Revenue)
    customer_num = len(df.Customer_id.unique())
```

```
    customer_rev = rev / customer_num
    rev_list.append(customer_rev)

# Calculate average annual revenue per customer
annual_customer_rev = sum(rev_list)/len(rev_list)
annual_customer_rev
```

Out[474…

```
1975.3446095266863
```

In [475…

```
# Let's assume there is a cost per customer
# E.g. a voucher to entice customers to stay this could be considered mitigation cost
mitigation_cost = 350
# Alternatively we could set cost to be 5% of the average annual customer revenue
# mitigation_cost = annual_customer_rev * 0.05
```

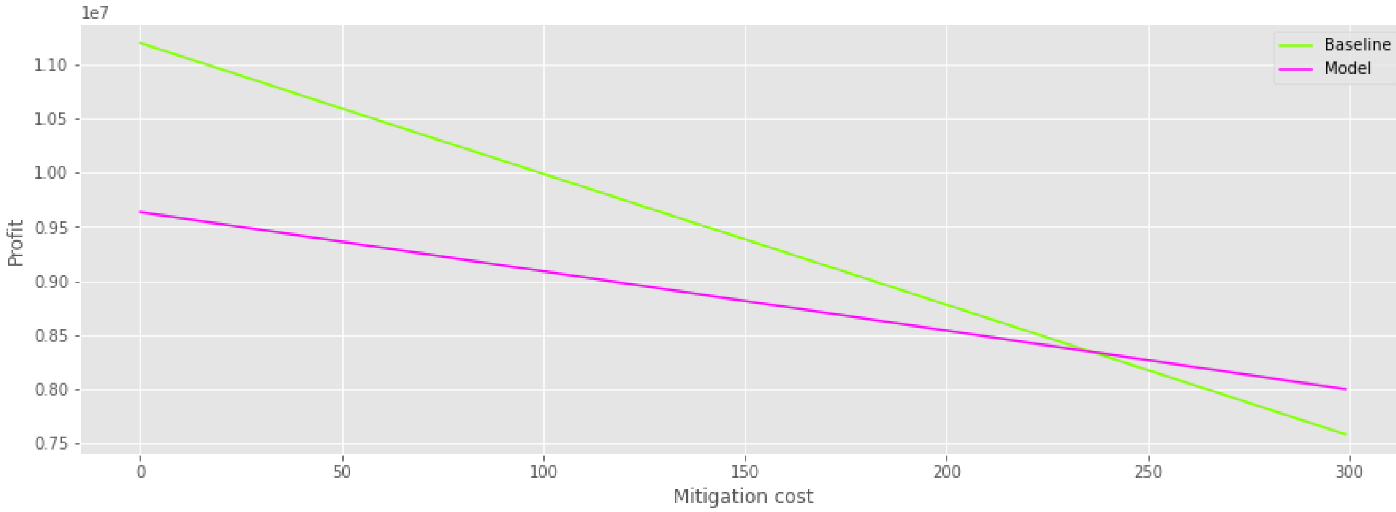
In [476…

```
# Calculate profit for baseline and model scenarios
#If the mitigation cost is applied on every customer
baseline_spend = mitigation_cost * (tp + tn + fp + fn)
baseline_rev = annual_customer_rev * (tp + fn)
baseline_profit = baseline_rev - baseline_spend

#If the mitigation cost is applied on what the model indicates
model_spend = mitigation_cost * (tp + fp)
model_rev = annual_customer_rev * tp
model_profit = model_rev - model_spend

print('If we targeted all customers with an effective mitigation strategy, the profit woul
print('If we targeted only the predicted churners with an effective mitigation strategy, t
print('This is an improvement of {:0.2f}%'.format((model_profit-baseline_profit)/baseline_
```

```
If we targeted all customers with an effective mitigation strategy, the profit would be 69
68027
If we targeted only the predicted churners with an effective mitigation strategy, the prof
it would be 7721355
This is an improvement of 10.81%
```

In [477…

```
# If we are unsure about mitigation cost, we can determine the best strategy by trying va
baseline_profit = []
model_profit =[]

for i in range(300):
    baseline_profit.append(baseline_rev - (tp+fp+tn+fn) * i)
    model_profit.append(model_rev - (tp+fp) * i)


fig = plt.figure()
ax = plt.axes()

x = range(300)
y1 = baseline_profit
y2 = model_profit
ax.plot(x,y1,color='chartreuse', label ='Baseline')
ax.plot(x,y2,color='fuchsia', label ='Model')
ax.legend()
plt.xlabel('Mitigation cost')
plt.ylabel('Profit')
pass
```



In [478…

```
y1 = np.array(y1)
y2 = np.array(y2)
idx = np.argwhere(np.diff(np.sign(y1 - y2))).flatten()
int(idx)
```

Out[478…

```
236
```

In [479…

```
print('Our model is superior to the baseline scenario when the mitigation cost exceeds',ir
```

```
Our model is superior to the baseline scenario when the mitigation cost exceeds 236 dollar
s per customer
```

Maximum return

In [480…

```
# Goal: Maximum return on investment; our metric in this case is precision
# Calculate baseline and model return
baseline_return = (baseline_rev - baseline_spend) / baseline_spend *100
model_return = (model_rev - model_spend) / model_spend *100
```

In [481…

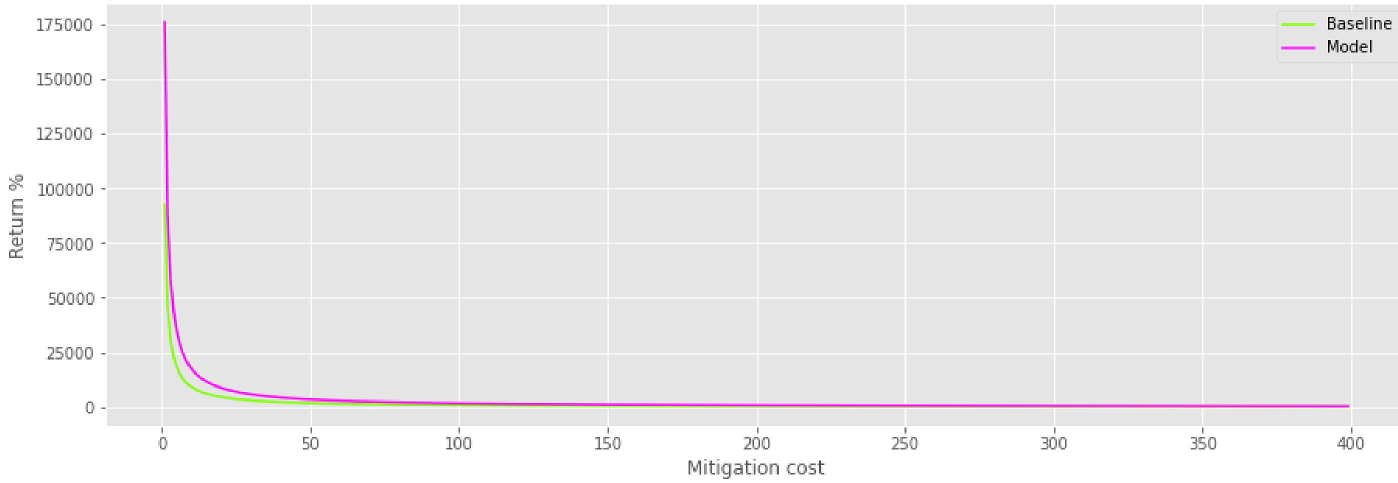
```
print('If we targeted all customers with an effective mitigation strategy, the return on i
print('If we targeted only the predicted churners with an effective mitigation strategy, t
```

```
If we targeted all customers with an effective mitigation strategy, the return on investme
nt would be 164.87%
If we targeted only the predicted churners with an effective mitigation strategy, the retu
rn on investment would be 403.75%
```

In [482…

```
# If we are unsure about mitigation cost, we can determine the best strategy by trying var
baseline_return = []
model_return =[]

for i in range(1,400):
    baseline_return.append((baseline_rev - (tp+fp+tn+fn) * i)/ ((tp+fp+tn+fn) * i) * 100)
    model_return.append((model_rev - (tp+fp) * i)/ ((tp+fp) * i) * 100)

fig = plt.figure()
ax = plt.axes()

x = range(1,400)
y1 = baseline_return
y2 = model_return
ax.plot(x,y1,color='chartreuse', label ='Baseline')
ax.plot(x,y2,color='fuchsia', label ='Model')
```

```
ax.legend()
plt.xlabel('Mitigation cost')
plt.ylabel('Return %')
pass
```



In [483…

```
print('Our model would give better return on investment compared to the baseline scenario
```

```
Our model would give better return on investment compared to the baseline scenario regardl
ess of the mitigation cost
```

Data scoring

It is now time for us to apply the model earlier built that identifies the characteristics of customers who churned on new data(score data) in order to get the list of customers with prospensity to churn in our new data.

In [484…

```
score_data.head()
```

Out[484…

	Customer_id	Quantity_sum	Quantity_mean	Quantity_median	Quantity_min	Quantity_max	Revenue_sum	R
4071	15071	2.00	1.00	1.00	1.00	1.00	2746.55	
14598	25598	4.00	4.00	4.00	4.00	4.00	1151.76	
11148	22148	1.00	1.00	1.00	1.00	1.00	69.99	
10998	21998	2.00	2.00	2.00	2.00	2.00	56.98	
8239	19239	2.00	2.00	2.00	2.00	2.00	78.98	

5 rows × 54 columns

In [485…

```
score_data.shape
```

Out[485…

```
(6409, 54)
```

In [486…

```
score_data.head().transpose()
```

Out[486…

	4071	14598	11148	10998	8239
Customer_id	15071	25598	22148	21998	19239
Quantity_sum	2.00	4.00	1.00	2.00	2.00

	4071	14598	11148	10998	8239
Quantity_mean	1.00	4.00	1.00	2.00	2.00
Quantity_median	1.00	4.00	1.00	2.00	2.00
Quantity_min	1.00	4.00	1.00	2.00	2.00
Quantity_max	1.00	4.00	1.00	2.00	2.00
Revenue_sum	2746.55	1151.76	69.99	56.98	78.98
Revenue_mean	1373.28	1151.76	69.99	56.98	78.98
Revenue_median	1373.28	1151.76	69.99	56.98	78.98
Revenue_min	564.99	1151.76	69.99	56.98	78.98
Revenue_max	2181.56	1151.76	69.99	56.98	78.98
Profit_sum	1117.65	426.99	43.81	35.67	45.88
Profit_mean	558.83	426.99	43.81	35.67	45.88
Profit_median	558.83	426.99	43.81	35.67	45.88
Profit_min	256.77	426.99	43.81	35.67	45.88
Profit_max	860.88	426.99	43.81	35.67	45.88
Days_elapsed_sum	467.00	0.00	0.00	0.00	0.00
Days_elapsed_mean	467.00	NaN	NaN	NaN	NaN
Days_elapsed_median	467.00	NaN	NaN	NaN	NaN
Days_elapsed_min	467.00	NaN	NaN	NaN	NaN
Days_elapsed_max	467.00	NaN	NaN	NaN	NaN
Recency	131.00	55.00	220.00	198.00	39.00
Frequency	2	1	1	1	1
Monetary	2746.55	1151.76	69.99	56.98	78.98
RFM_segment	221	142	343	343	143
RFM_score	5	7	10	10	8
RFM_status	Gold	Silver	Bronze	Bronze	Silver
RFM_cluster	2	0	3	3	0
Tenure_months	15.34	0.00	0.00	0.00	0.00
Churn	0	0	0	0	0
Birth_date	1981-12-13	1976-01-14	1952-08-24	1964-05-07	1979-04-17
Marital_status	S	M	M	M	M
Gender	F	F	F	M	M
Yearly_income	30000.00	40000.00	10000.00	90000.00	40000.00
Number_children_at_home	0	0	1	0	0
Education	Bachelors	Graduate Degree	Partial High School	Bachelors	Graduate Degree
Ocupation	Clerical	Clerical	Clerical	Professional	Skilled Manual
Commute_distance	0-1 Miles	0-1 Miles	5-10 Miles	5-10 Miles	1-2 Miles

	4071	14598	11148	10998	8239
Total_children	0	0	2	2	1
House_ownership	1	1	1	1	1
Car_ownership	0	0	2	1	0
Age	39	45	68	57	42
Age_group	30s	40-50s	60s or older	40-50s	40-50s
Income_group	Lower-middle	Lower-middle	Low	Upper-middle	Lower-middle
Sales_reason_type_Marketing	0	0	0	0	0
Sales_reason_type_Other	1	1	1	1	1
Sales_reason_type_Promotion	1	0	0	0	0
Sales_reason_Manufacturer	0	0	0	0	0
Sales_reason_On_Promotion	1	0	0	0	0
Sales_reason_Other	0	0	0	1	0
Sales_reason_Price	1	1	0	1	1
Sales_reason_Quality	0	0	0	0	0
Sales_reason_Review	0	0	1	0	0
Sales_reason_Television_Advertisement	0	0	0	0	0

In [487…

```
score_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6409 entries, 4071 to 13935
Data columns (total 54 columns):
 #   Column                                 Non-Null Count  Dtype  
---  ------                                 --------------  -----  
 0   Customer_id                            6409 non-null   int64  
 1   Quantity_sum                           6409 non-null   float64
 2   Quantity_mean                          6409 non-null   float64
 3   Quantity_median                        6409 non-null   float64
 4   Quantity_min                           6409 non-null   float64
 5   Quantity_max                           6409 non-null   float64
 6   Revenue_sum                            6409 non-null   float64
 7   Revenue_mean                           6409 non-null   float64
 8   Revenue_median                         6409 non-null   float64
 9   Revenue_min                            6409 non-null   float64
 10  Revenue_max                            6409 non-null   float64
 11  Profit_sum                             6409 non-null   float64
 12  Profit_mean                            6409 non-null   float64
 13  Profit_median                          6409 non-null   float64
 14  Profit_min                             6409 non-null   float64
 15  Profit_max                             6409 non-null   float64
 16  Days_elapsed_sum                       6409 non-null   float64
 17  Days_elapsed_mean                      2762 non-null   float64
 18  Days_elapsed_median                    2762 non-null   float64
 19  Days_elapsed_min                       2762 non-null   float64
 20  Days_elapsed_max                       2762 non-null   float64
 21  Recency                                6409 non-null   float64
 22  Frequency                              6409 non-null   int64  
 23  Monetary                               6409 non-null   float64
 24  RFM_segment                            6409 non-null   object 
```

```
 25   RFM_score                          6409 non-null   int64
 26   RFM_status                         6409 non-null   category
 27   RFM_cluster                        6409 non-null   int32
 28   Tenure_months                      6409 non-null   float64
 29   Churn                              6409 non-null   int64
 30   Birth_date                         6409 non-null   object
 31   Marital_status                     6409 non-null   object
 32   Gender                             6409 non-null   object
 33   Yearly_income                      6409 non-null   float64
 34   Number_children_at_home            6409 non-null   int64
 35   Education                          6409 non-null   object
 36   Ocupation                          6409 non-null   object
 37   Commute_distance                   6409 non-null   object
 38   Total_children                     6409 non-null   int64
 39   House_ownership                    6409 non-null   object
 40   Car_ownership                      6409 non-null   int64
 41   Age                                6409 non-null   int64
 42   Age_group                          6409 non-null   category
 43   Income_group                       6409 non-null   category
 44   Sales_reason_type_Marketing        6409 non-null   uint8
 45   Sales_reason_type_Other            6409 non-null   uint8
 46   Sales_reason_type_Promotion        6409 non-null   uint8
 47   Sales_reason_Manufacturer          6409 non-null   uint8
 48   Sales_reason_On_Promotion          6409 non-null   uint8
 49   Sales_reason_Other                 6409 non-null   uint8
 50   Sales_reason_Price                 6409 non-null   uint8
 51   Sales_reason_Quality               6409 non-null   uint8
 52   Sales_reason_Review                6409 non-null   uint8
 53   Sales_reason_Television_Advertisement  6409 non-null   uint8
dtypes: category(3), float64(24), int32(1), int64(8), object(8), uint8(10)
memory usage: 2.1+ MB
```

In [488…

```python
# Double check for missing values and duplicates
score_data.isnull().sum()[score_data.isnull().sum()!=0]
```

Out[488…

```
Days_elapsed_mean      3647
Days_elapsed_median    3647
Days_elapsed_min       3647
Days_elapsed_max       3647
dtype: int64
```

In [489…

```python
# Check for potential duplicate rows
print('Number of duplicates:', score_data.duplicated().sum())
```

```
Number of duplicates: 0
```

In [490…

```python
# Delete fields that are of no use for our modelling
# Exclude Customer_id from deleted list so we can idetify churning customers later on
del_var.remove('Customer_id')
del_var = del_var + ['Churn']# churn also needs to be deleted because that is what we want
score_data.drop(del_var, axis = 1, inplace= True)
```

In [491…

```python
# Get distribution/frequency per category
for var in cat_var:
    print(score_data.groupby(var).size())
```

```
RFM_segment
111    164
112     43
113    102
114      3
121    349
```

```
122     85
123    265
124     35
141      1
142    443
143    329
144    501
211    178
212     10
213     45
214      2
221    486
222     96
223    173
224     26
241      5
242    417
243    377
244    512
311    107
312      2
313     17
321    387
322     70
323    107
324     10
341      7
342    341
343    295
344    419
dtype: int64
RFM_score
3      164
4      570
5      790
6      799
7      745
8      886
9     1229
10     807
11     419
dtype: int64
RFM_status
Gold      2323
Silver    2860
Bronze     807
Green      419
dtype: int64
RFM_cluster
0    2053
1     399
2    2371
3    1586
dtype: int64
Marital_status
M    3507
S    2902
dtype: int64
Gender
F    3155
M    3254
dtype: int64
Total_children
0    1824
1    1203
2    1312
```

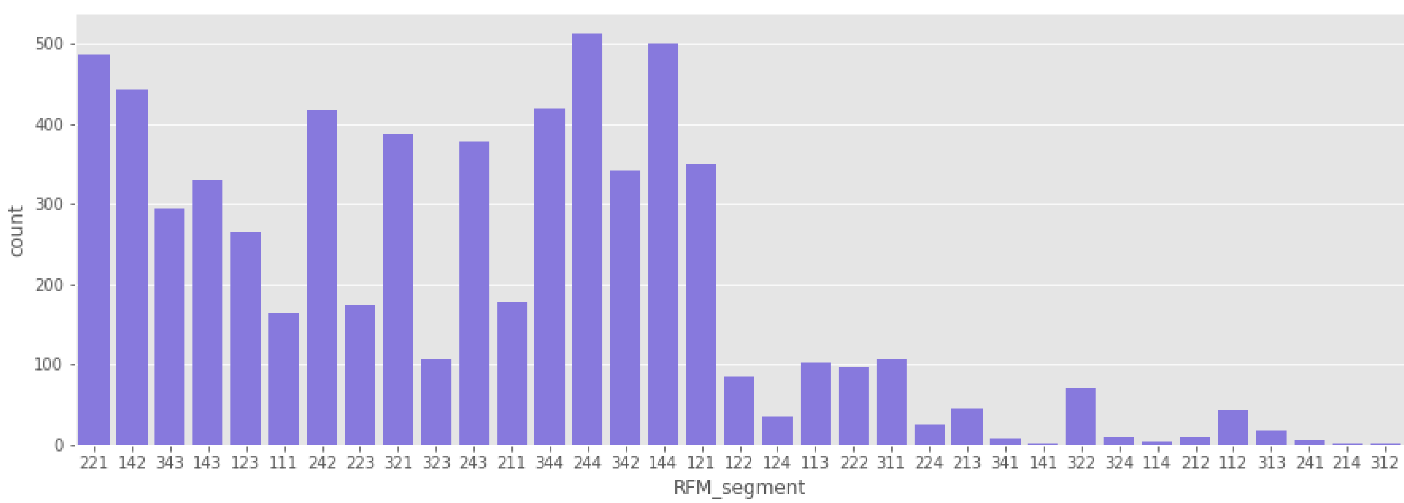
```
3     760
4     784
5     526
dtype: int64
Number_children_at_home
0    3858
1     827
2     545
3     402
4     413
5     364
dtype: int64
Education
Bachelors              1911
Graduate Degree        1100
High School            1132
Partial College        1750
Partial High School     516
dtype: int64
Ocupation
Clerical          1011
Management        1099
Manual             793
Professional      1944
Skilled Manual    1562
dtype: int64
House_ownership
0    2023
1    4386
dtype: int64
Car_ownership
0    1477
1    1709
2    2174
3     609
4     440
dtype: int64
Commute_distance
0-1 Miles     2202
1-2 Miles     1072
10+ Miles      893
2-5 Miles     1082
5-10 Miles    1160
dtype: int64
Age_group
20s               0
30s            1129
40-50s         3843
60s or older   1437
dtype: int64
Income_group
Low              388
Lower-middle    2556
Upper-middle    3346
High             119
dtype: int64
Sales_reason_type_Marketing
0    6141
1     268
dtype: int64
Sales_reason_type_Other
0     173
1    6236
dtype: int64
Sales_reason_type_Promotion
0    5267
```

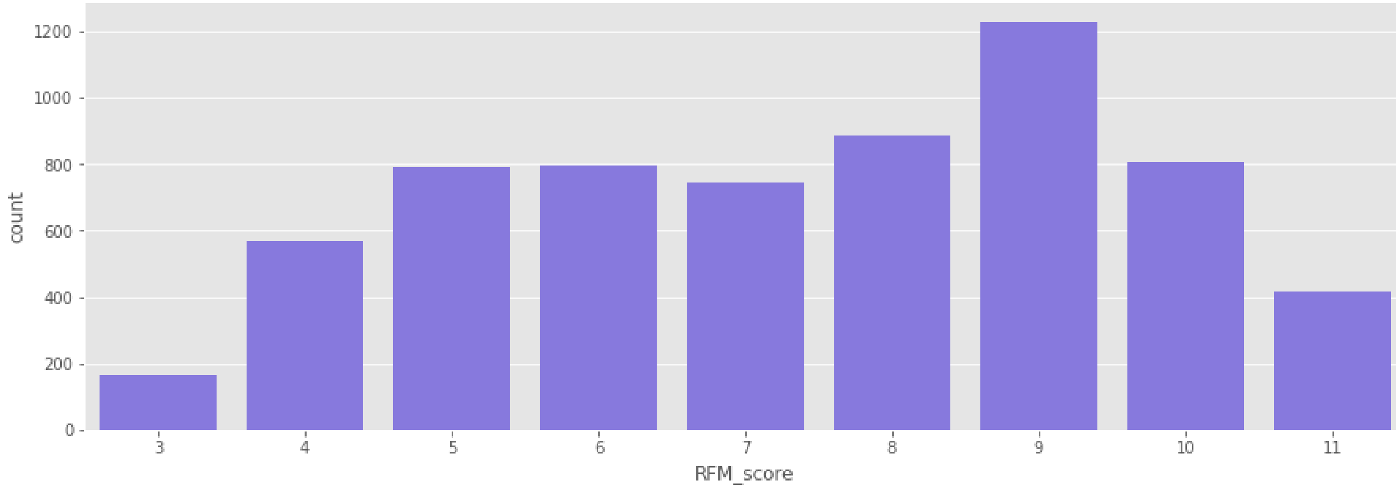
```
1    1142
dtype: int64
Sales_reason_Manufacturer
0    5918
1     491
dtype: int64
Sales_reason_On_Promotion
0    5267
1    1142
dtype: int64
Sales_reason_Other
0    5907
1     502
dtype: int64
Sales_reason_Price
0     449
1    5960
dtype: int64
Sales_reason_Quality
0    5995
1     414
dtype: int64
Sales_reason_Review
0    5959
1     450
dtype: int64
Sales_reason_Television_Advertisement
0    6141
1     268
dtype: int64
```

In [492…

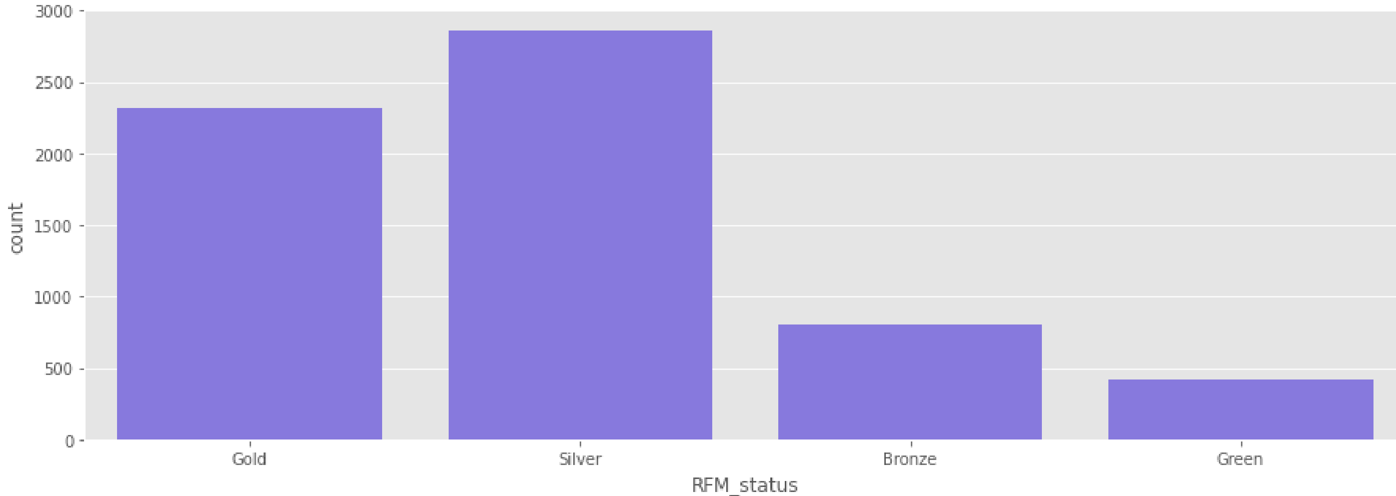
```python
# PLot distribution/frequency per category
cat_df = score_data[cat_var]
plt.rcParams['figure.figsize'] = (15, 5) # Chart sizes

for i, col in enumerate(cat_df.columns):
   plt.figure(i)
   sns.countplot(x=col, data=cat_df, color = 'mediumslateblue')
```

```
<ipython-input-492-1349f22694a4>:6: RuntimeWarning: More than 20 figures have been opened.
Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained unt
il explicitly closed and may consume too much memory. (To control this warning, see the rc
Param `figure.max_open_warning`).
  plt.figure(i)
```

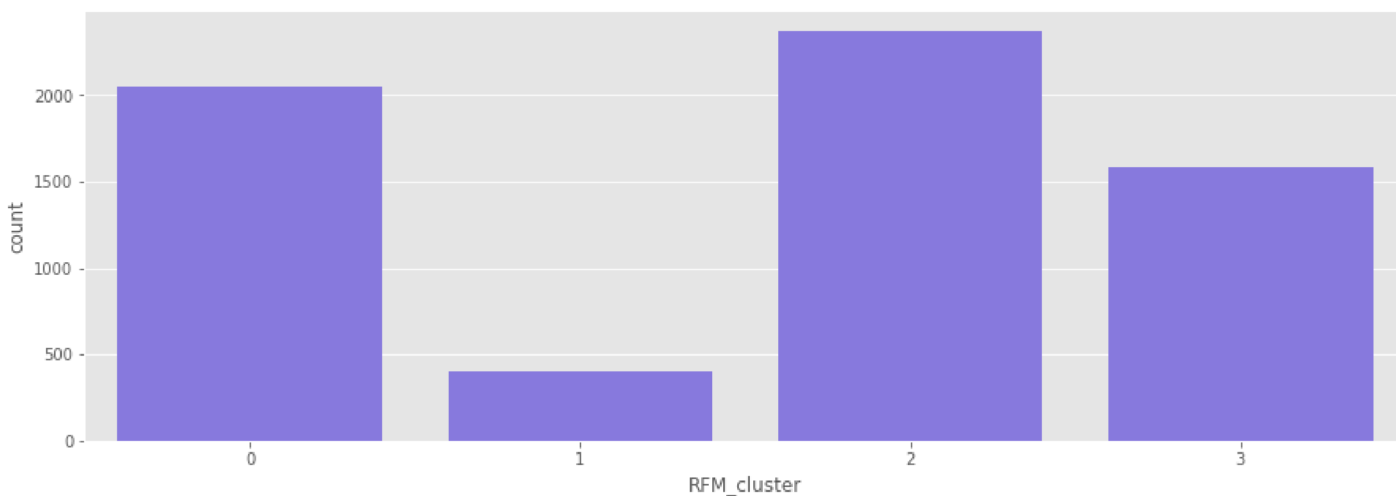




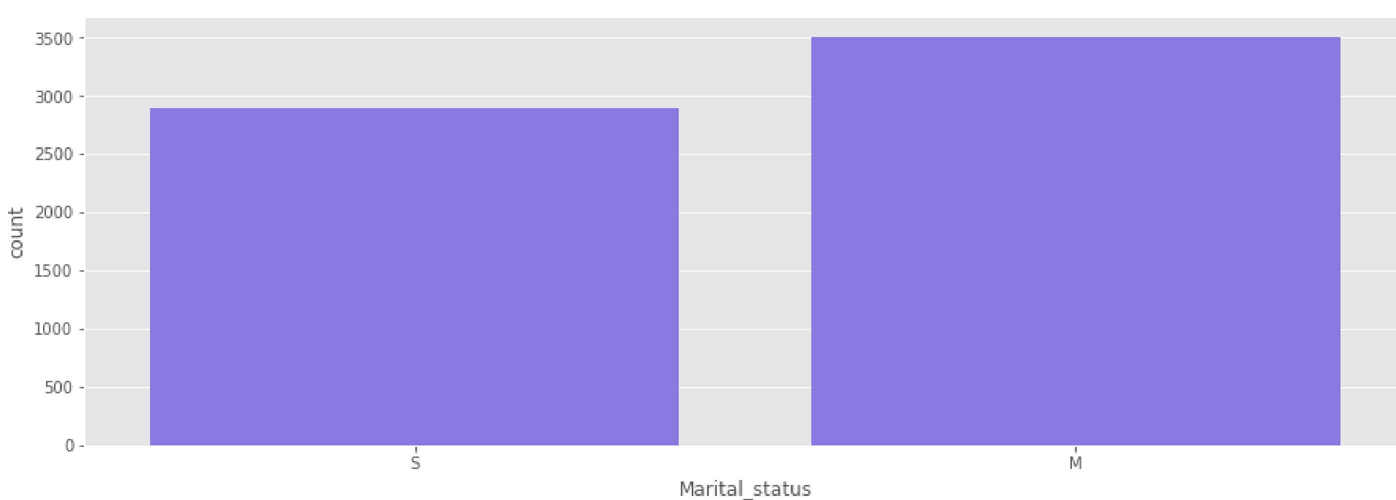
count
0
200
400
600
800
1000
1200
3
4
5
6
7
8
9
10
11
RFM_score



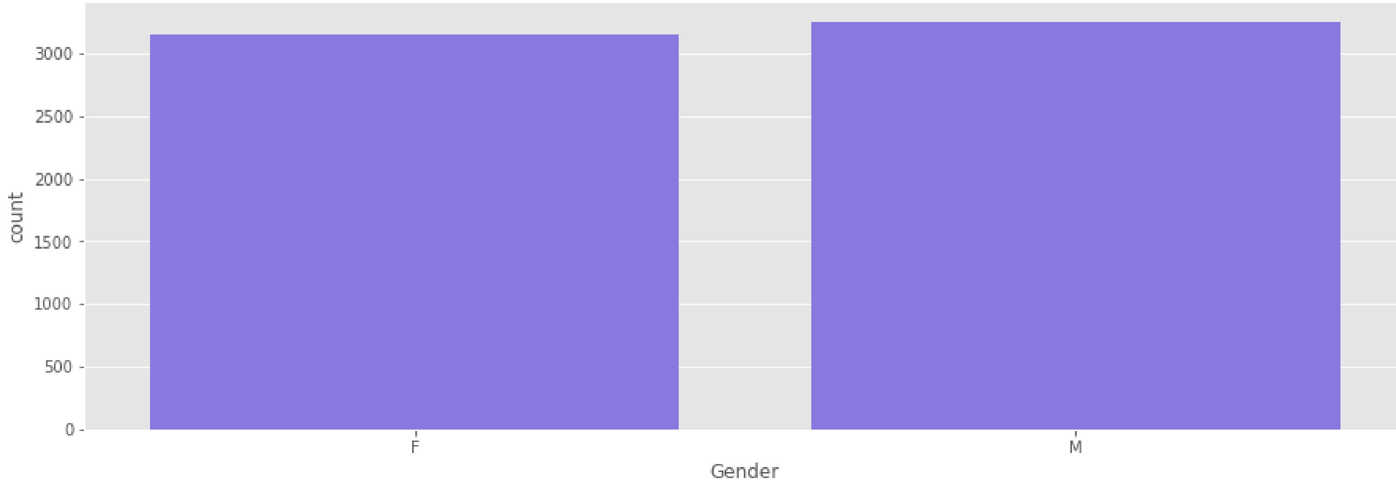
count
0
500
1000
1500
2000
2500
3000
Gold
Silver
Bronze
Green
RFM_status

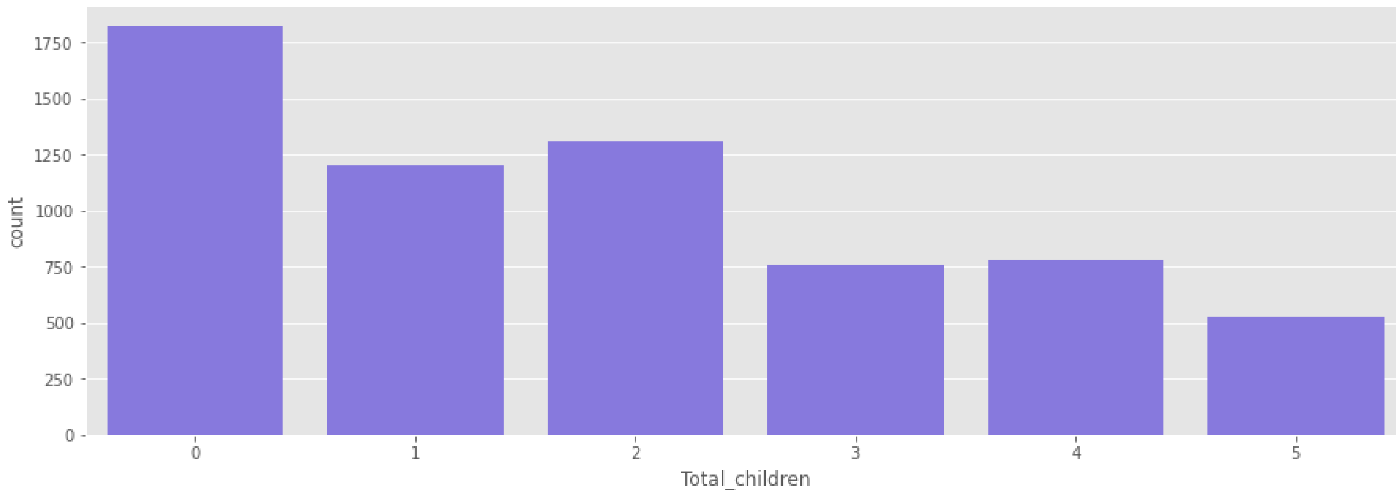


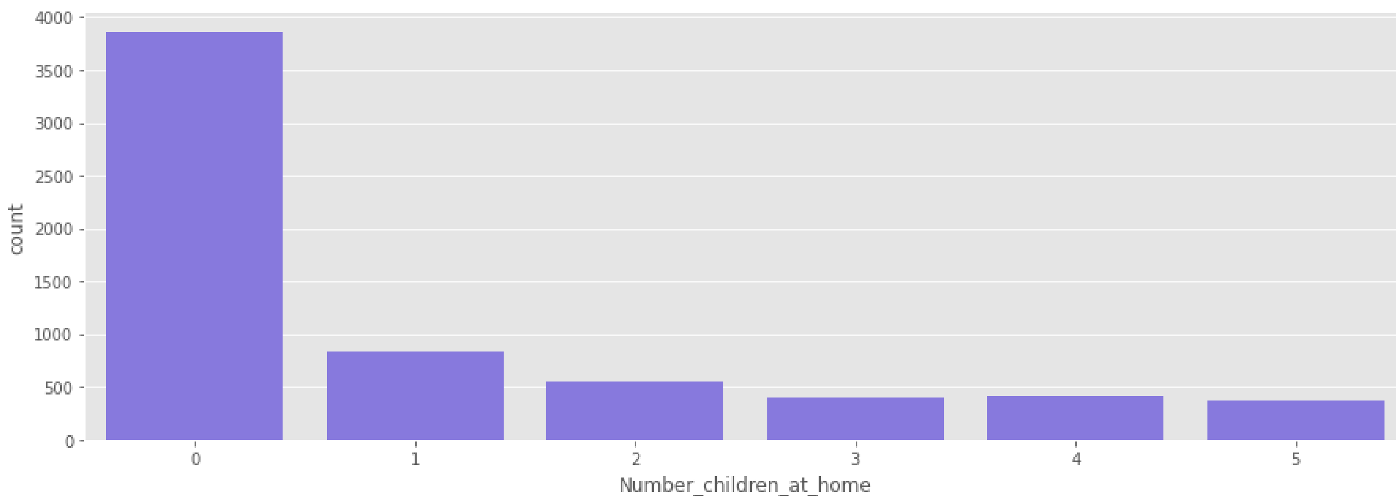
count
0
500
1000
1500
2000
0
1
2
3
RFM_cluster

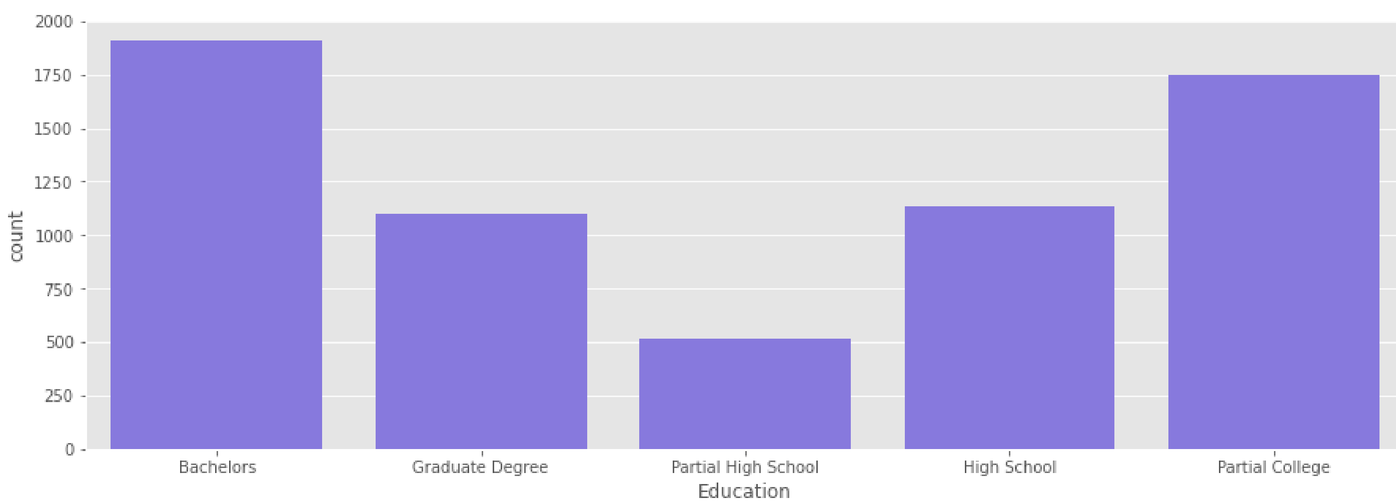


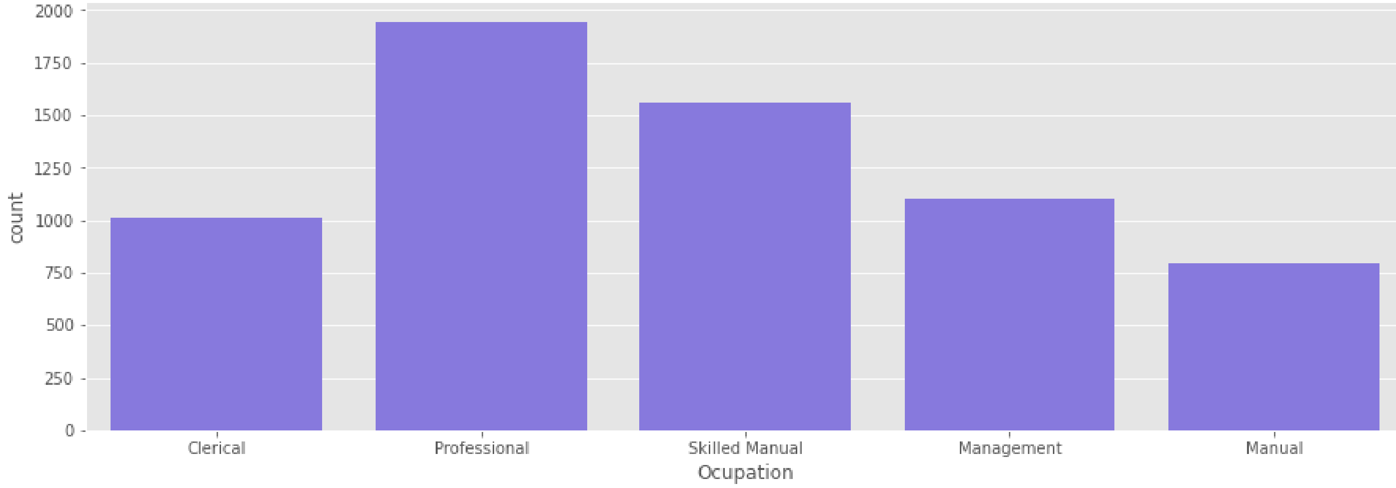
count
0
500
1000
1500
2000
2500
3000
3500
S
M
Marital_status

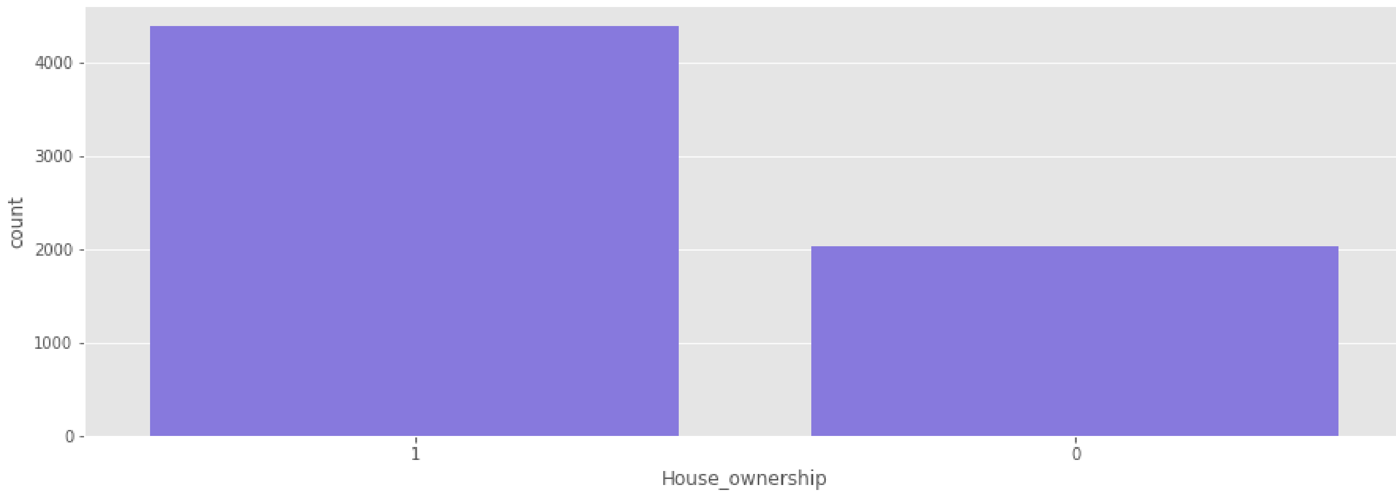

count
0
500
1000
1500
2000
2500
3000
F
M
Gender

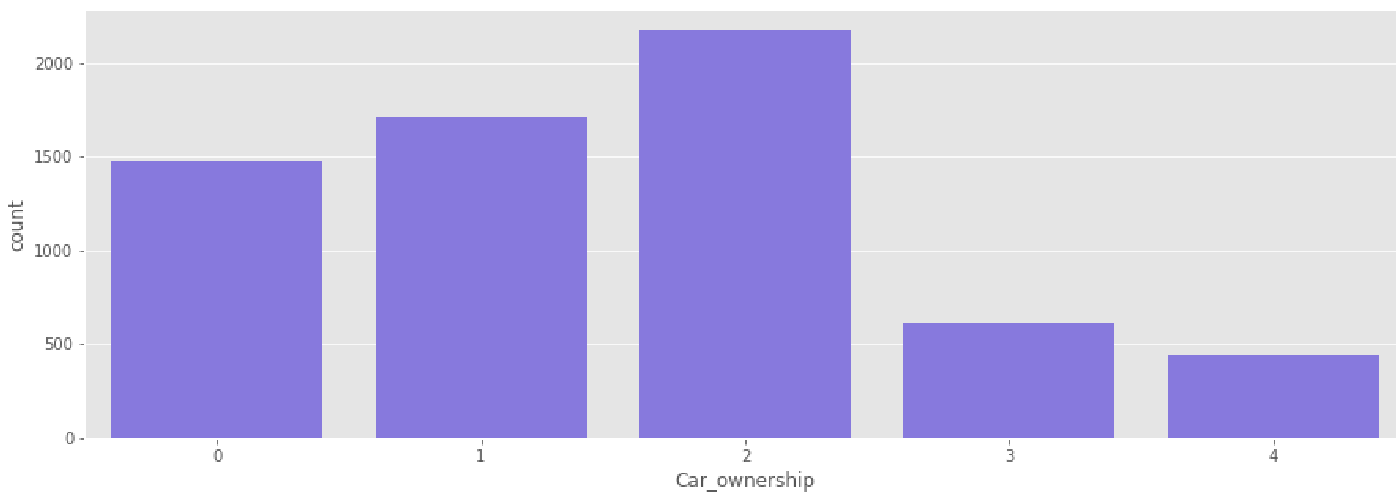

count
0
250
500
750
1000
1250
1500
1750
0
1
2
3
4
5
Total_children

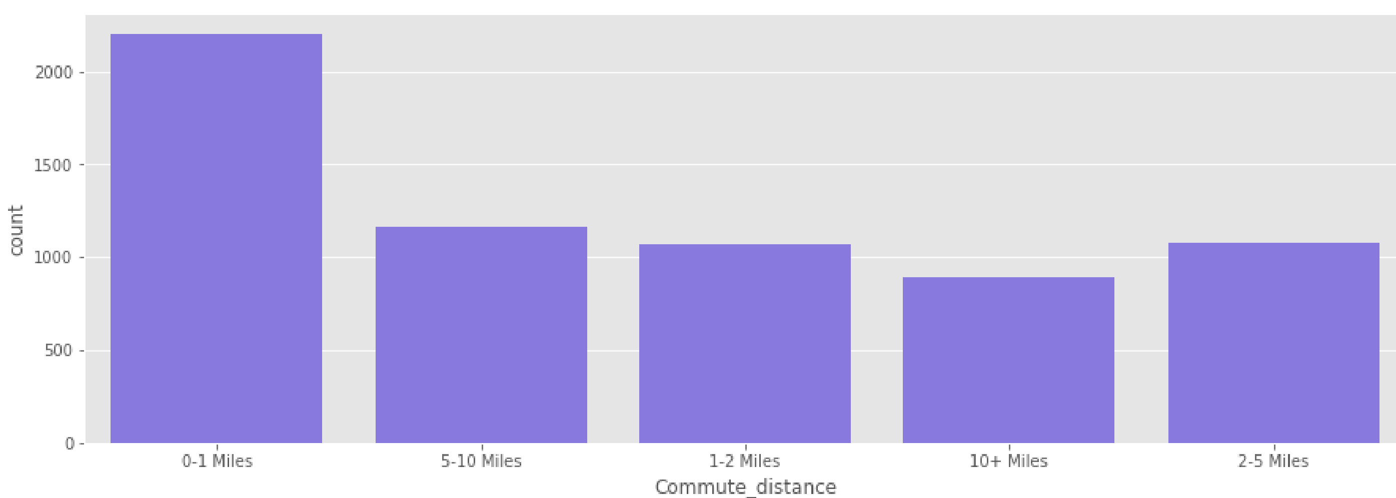

count
0
500
1000
1500
2000
2500
3000
3500
4000
0
1
2
3
4
5
Number_children_at_home

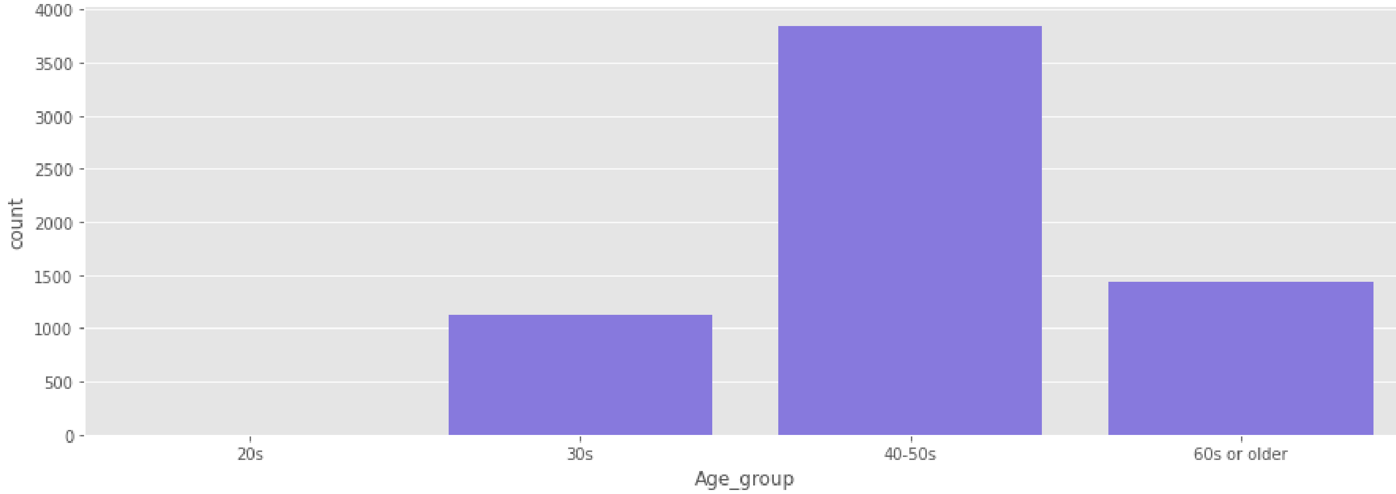

count
0
250
500
750
1000
1250
1500
1750
2000
Bachelors
Graduate Degree
Partial High School
High School
Partial College
Education

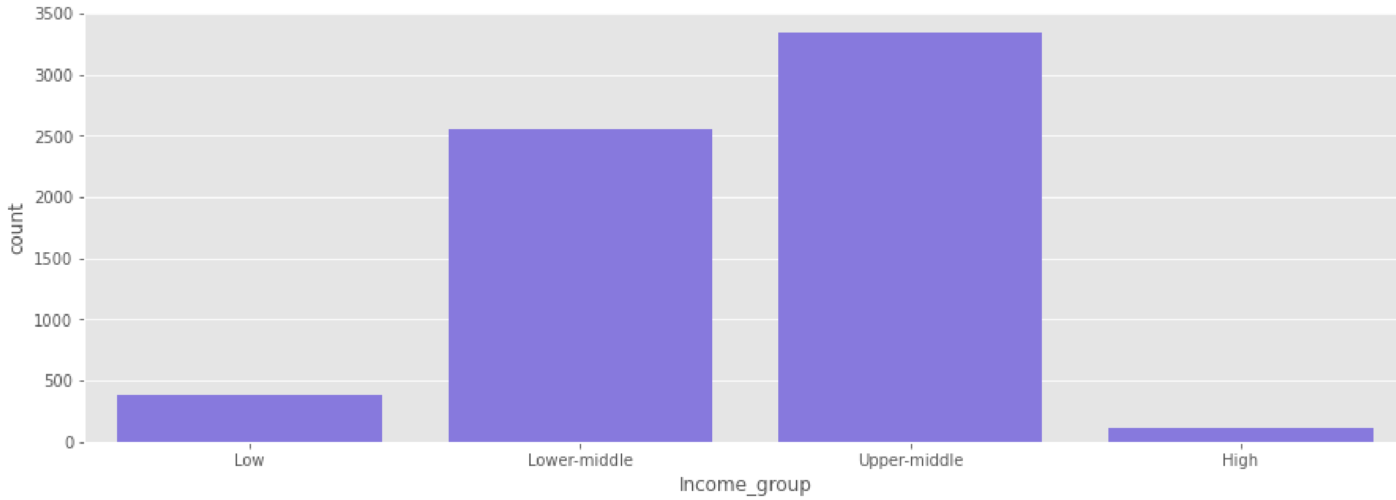

count
2000
1750
1500
1250
1000
750
500
250
0
Clerical
Professional
Skilled Manual
Management
Manual
Ocupation

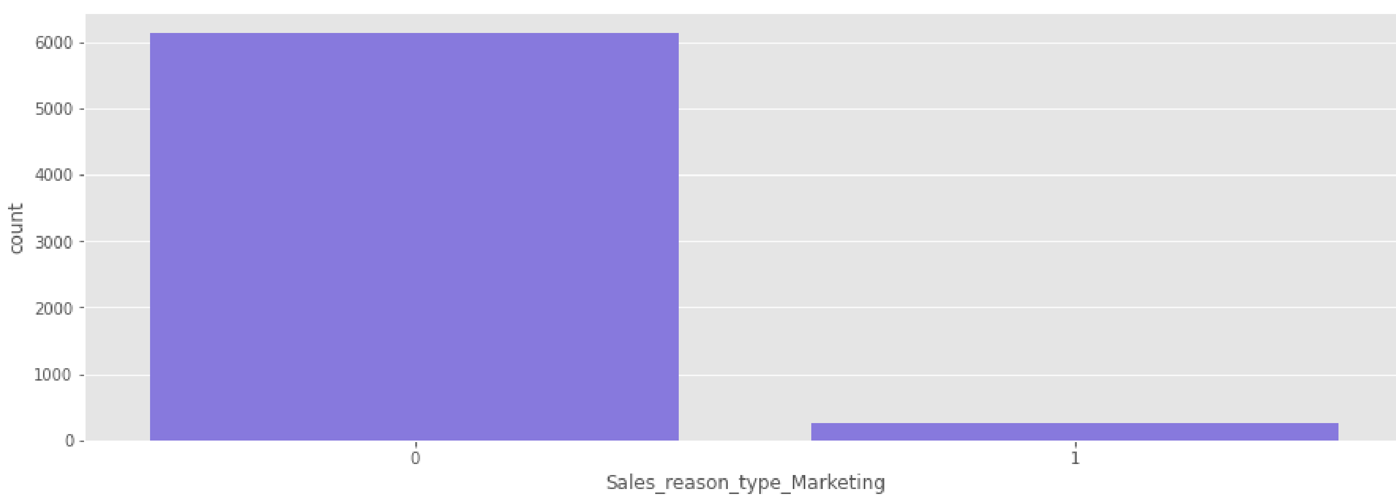

count
4000
3000
2000
1000
0
1
0
House_ownership

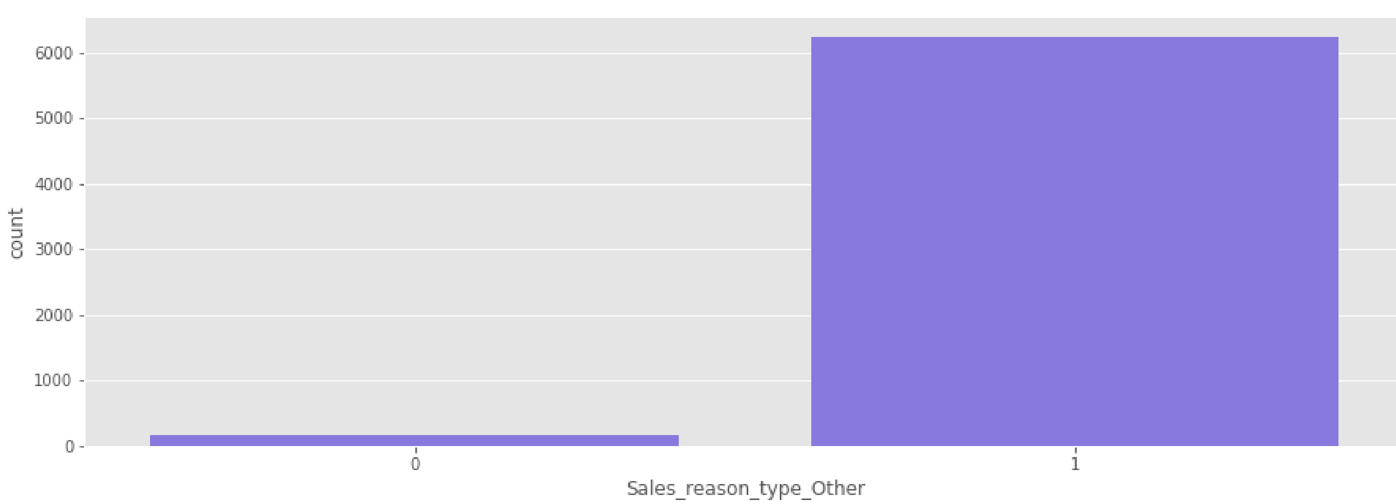

count
2000
1500
1000
500
0
0
1
2
3
4
Car_ownership

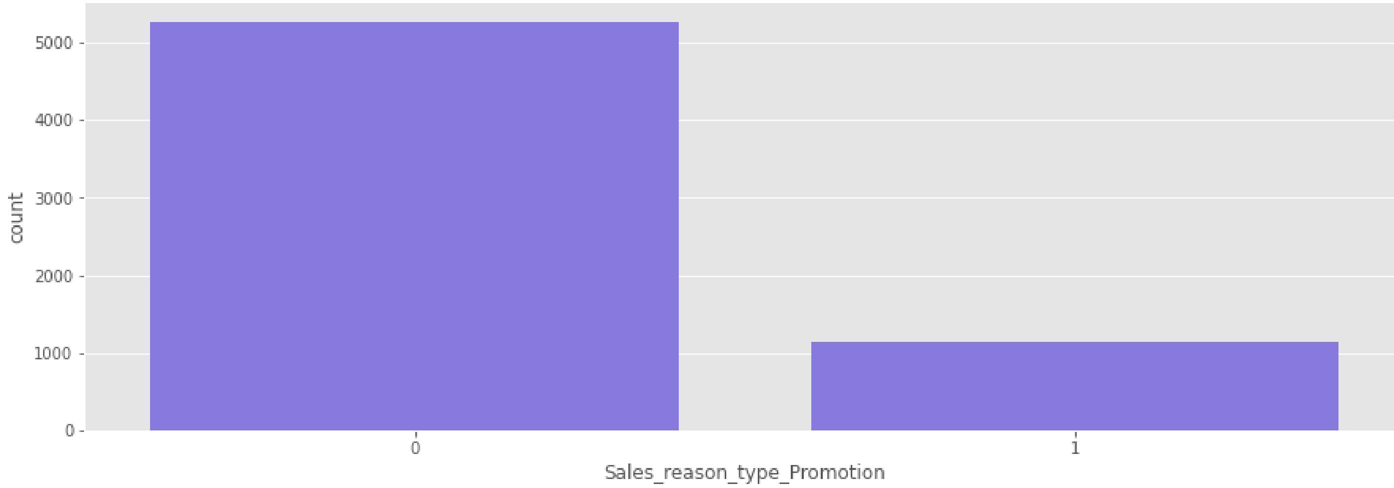
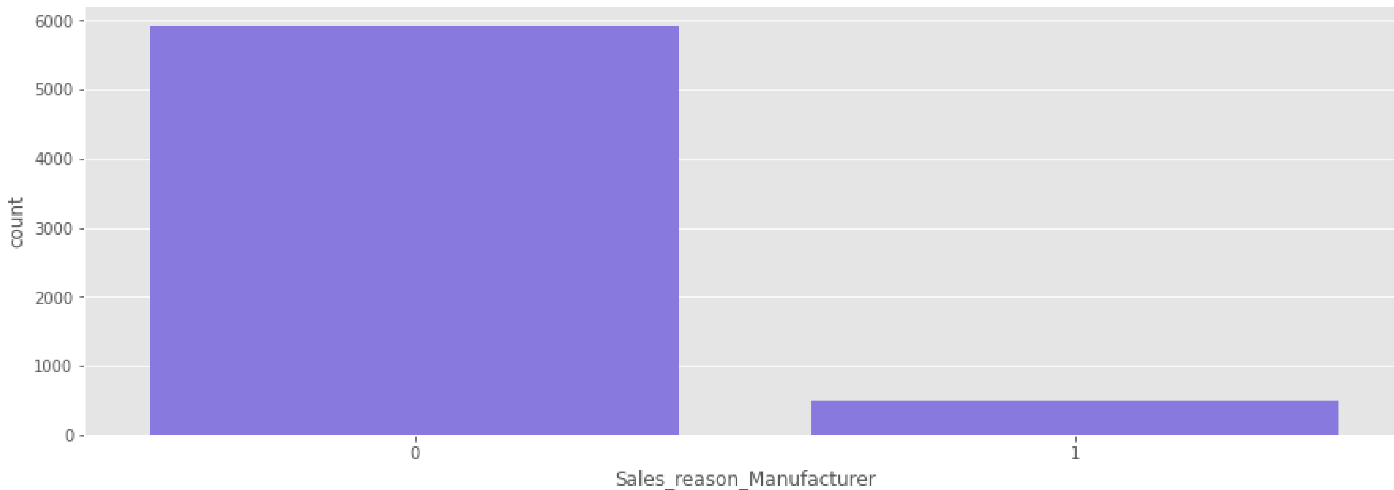
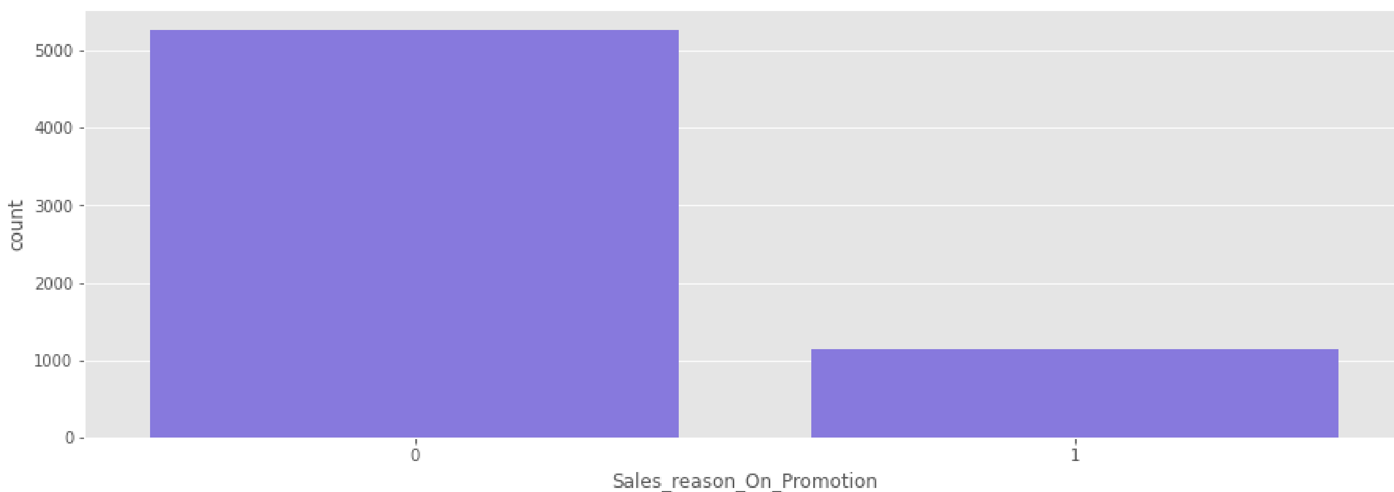
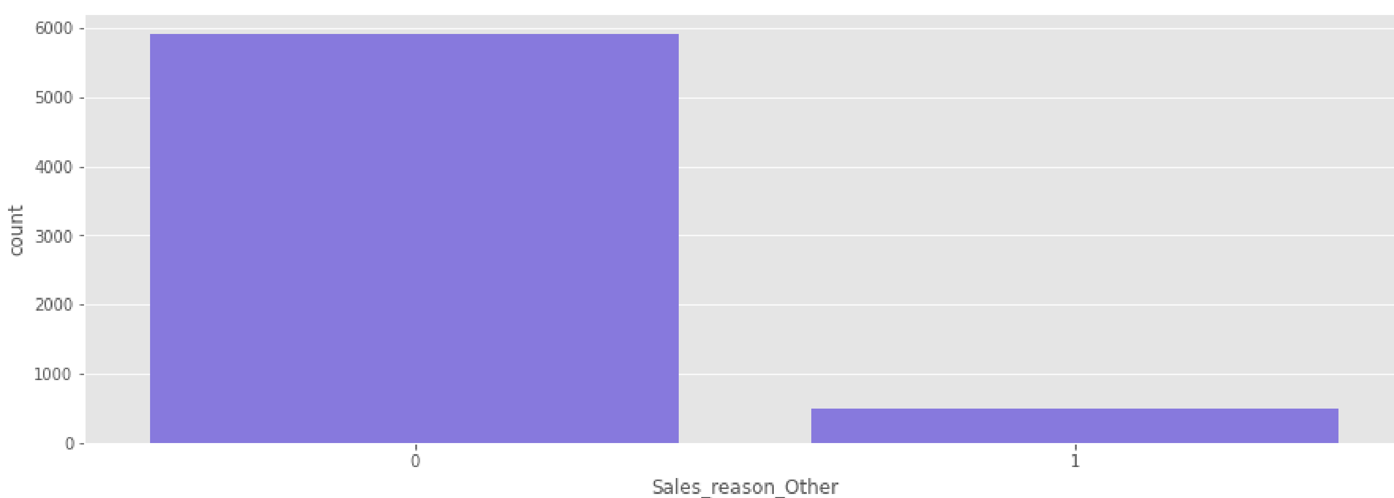

count
2000
1500
1000
500
0
0-1 Miles
5-10 Miles
1-2 Miles
10+ Miles
2-5 Miles
Commute_distance

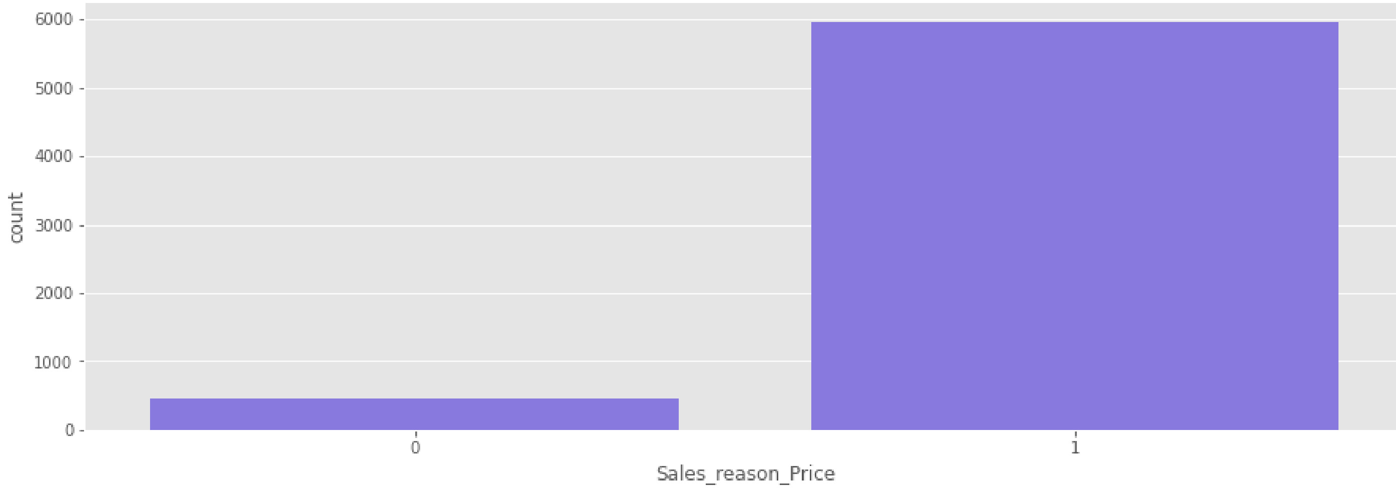

count
0
500
1000
1500
2000
2500
3000
3500
4000
20s
30s
40-50s
60s or older
Age_group

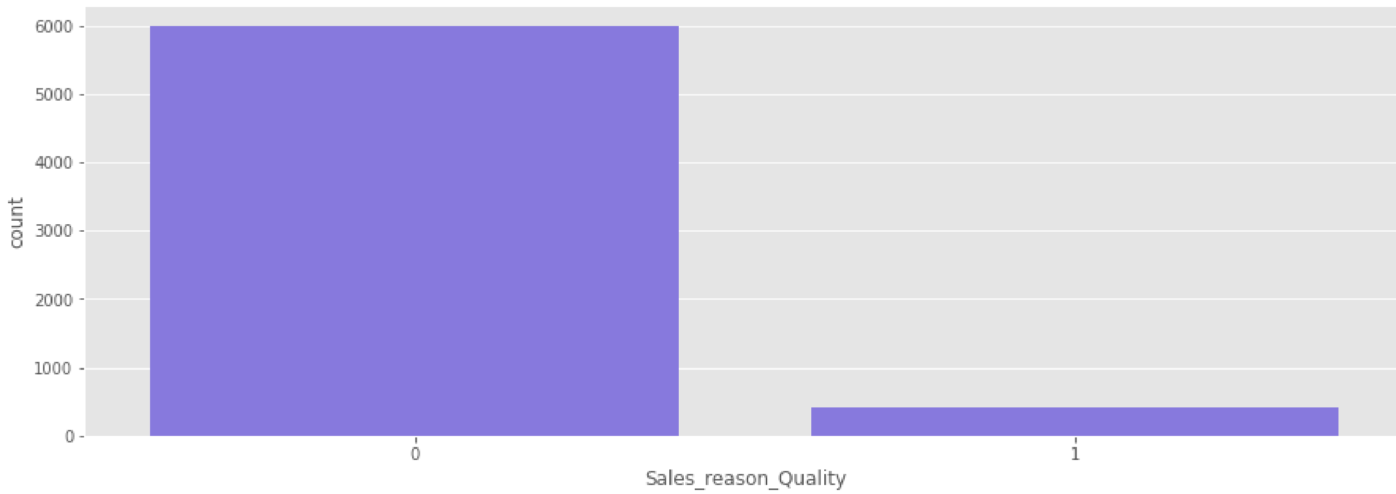

count
0
500
1000
1500
2000
2500
3000
3500
Low
Lower-middle
Upper-middle
High
Income_group

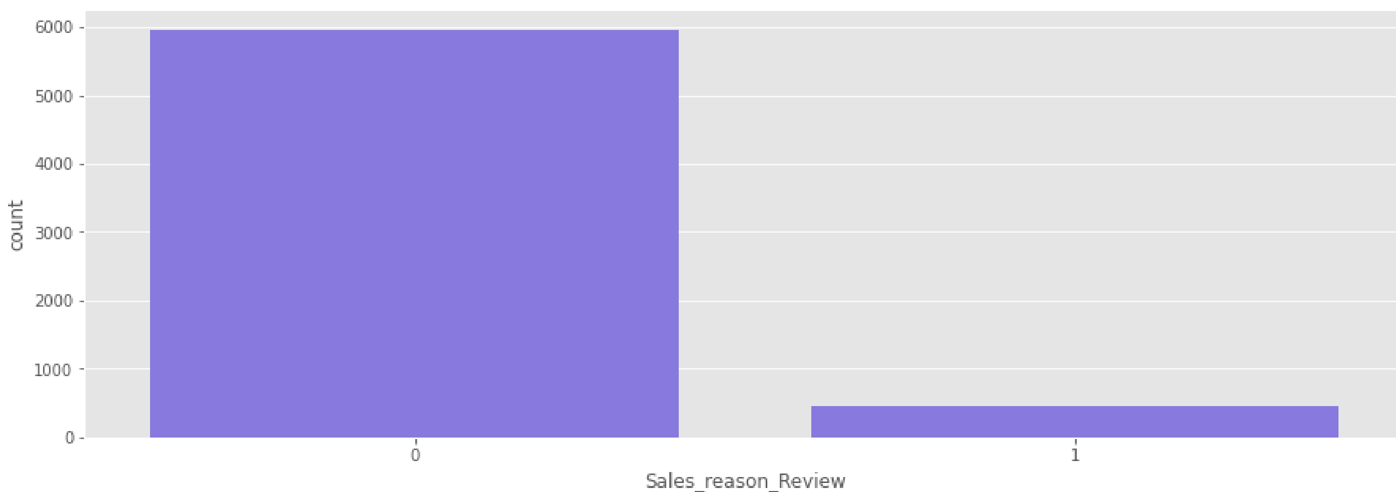

count
0
1000
2000
3000
4000
5000
6000
0
1
Sales_reason_type_Marketing

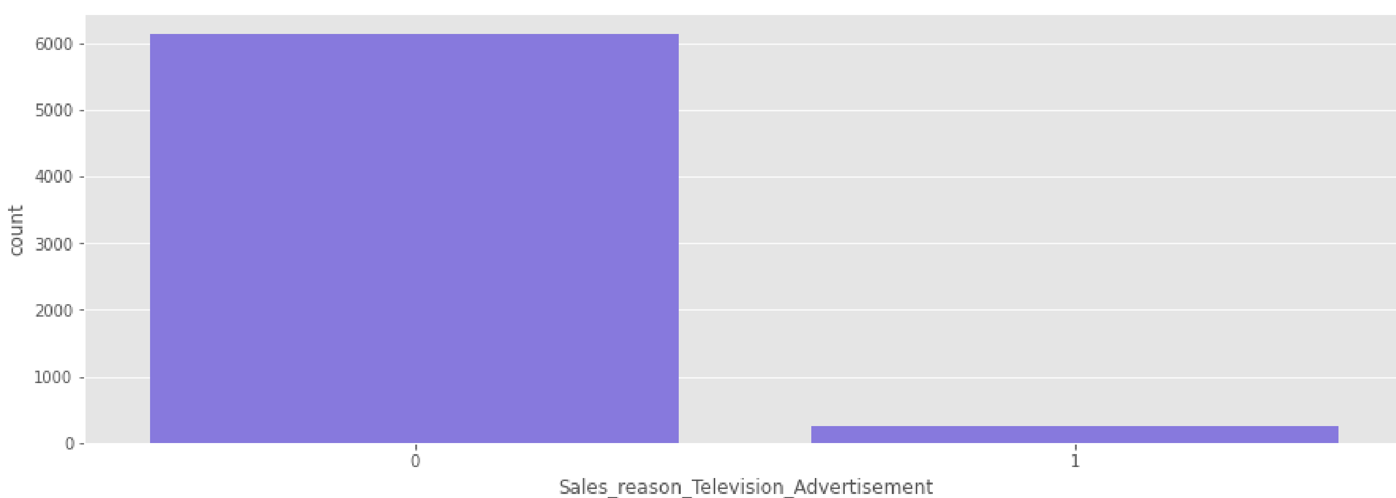

count
0
1000
2000
3000
4000
5000
6000
0
1
Sales_reason_type_Other


count
0
1000
2000
3000
4000
5000
0
1
Sales_reason_type_Promotion

count
0
1000
2000
3000
4000
5000
6000
0
1
Sales_reason_Manufacturer

count
0
1000
2000
3000
4000
5000
0
1
Sales_reason_On_Promotion

count
0
1000
2000
3000
4000
5000
6000
0
1
Sales_reason_Other


6000
5000
4000
3000
2000
1000
0
count
0
1
Sales_reason_Price


6000
5000
4000
3000
2000
1000
0
count
0
1
Sales_reason_Quality


6000
5000
4000
3000
2000
1000
0
count
0
1
Sales_reason_Review


6000
5000
4000
3000
2000
1000
0
count
0
1
Sales_reason_Television_Advertisement

In [493…

```
# Most machine learning models only accept numerical variables
# Some of our variables contains string or letter
# for example Gender = M or F
# Further more a more restricted number of model require boolean variables
# Below we transform all variables to boolean by default

# Encode categorical data (besides drivers which are already binary)
dummies = ['RFM_segment', 'RFM_score', 'RFM_status', 'RFM_cluster', 'Marital_status', 'Ger
           'Total_children', 'Number_children_at_home', 'Education', 'Ocupation', 'House_c
           'Car_ownership', 'Commute_distance', 'Age_group', 'Income_group']

encoded_score_data = pd.get_dummies(score_data, columns = [v for v in dummies], drop_first
```

In [494…

```
encoded_score_var= list(encoded_score_data.columns)
encoded_score_var
```

Out[494…

```
['Customer_id',
 'Quantity_sum',
 'Quantity_mean',
 'Quantity_median',
 'Quantity_max',
 'Revenue_mean',
 'Revenue_median',
 'Revenue_min',
 'Revenue_max',
 'Profit_sum',
 'Profit_mean',
 'Profit_median',
 'Profit_min',
 'Profit_max',
 'Recency',
 'Frequency',
 'Monetary',
 'Tenure_months',
 'Yearly_income',
 'Age',
 'Sales_reason_type_Marketing',
 'Sales_reason_type_Other',
 'Sales_reason_type_Promotion',
 'Sales_reason_Manufacturer',
 'Sales_reason_On_Promotion',
 'Sales_reason_Other',
 'Sales_reason_Price',
 'Sales_reason_Quality',
 'Sales_reason_Review',
 'Sales_reason_Television_Advertisement',
 'RFM_segment_111',
 'RFM_segment_112',
 'RFM_segment_113',
 'RFM_segment_114',
 'RFM_segment_121',
 'RFM_segment_122',
 'RFM_segment_123',
 'RFM_segment_124',
 'RFM_segment_141',
 'RFM_segment_142',
 'RFM_segment_143',
 'RFM_segment_144',
 'RFM_segment_211',
 'RFM_segment_212',
 'RFM_segment_213',
 'RFM_segment_214',
 'RFM_segment_221',
 'RFM_segment_222',
```

```
'RFM_segment_223',
'RFM_segment_224',
'RFM_segment_241',
'RFM_segment_242',
'RFM_segment_243',
'RFM_segment_244',
'RFM_segment_311',
'RFM_segment_312',
'RFM_segment_313',
'RFM_segment_321',
'RFM_segment_322',
'RFM_segment_323',
'RFM_segment_324',
'RFM_segment_341',
'RFM_segment_342',
'RFM_segment_343',
'RFM_segment_344',
'RFM_score_3',
'RFM_score_4',
'RFM_score_5',
'RFM_score_6',
'RFM_score_7',
'RFM_score_8',
'RFM_score_9',
'RFM_score_10',
'RFM_score_11',
'RFM_status_Gold',
'RFM_status_Silver',
'RFM_status_Bronze',
'RFM_status_Green',
'RFM_cluster_0',
'RFM_cluster_1',
'RFM_cluster_2',
'RFM_cluster_3',
'Marital_status_M',
'Marital_status_S',
'Gender_F',
'Gender_M',
'Total_children_0',
'Total_children_1',
'Total_children_2',
'Total_children_3',
'Total_children_4',
'Total_children_5',
'Number_children_at_home_0',
'Number_children_at_home_1',
'Number_children_at_home_2',
'Number_children_at_home_3',
'Number_children_at_home_4',
'Number_children_at_home_5',
'Education_Bachelors',
'Education_Graduate Degree',
'Education_High School',
'Education_Partial College',
'Education_Partial High School',
'Ocupation_Clerical',
'Ocupation_Management',
'Ocupation_Manual',
'Ocupation_Professional',
'Ocupation_Skilled Manual',
'House_ownership_0',
'House_ownership_1',
'Car_ownership_0',
'Car_ownership_1',
'Car_ownership_2',
'Car_ownership_3',
```

```
'Car_ownership_4',
'Commute_distance_0-1 Miles',
'Commute_distance_1-2 Miles',
'Commute_distance_10+ Miles',
'Commute_distance_2-5 Miles',
'Commute_distance_5-10 Miles',
'Age_group_20s',
'Age_group_30s',
'Age_group_40-50s',
'Age_group_60s or older',
'Income_group_Low',
'Income_group_Lower-middle',
'Income_group_Upper-middle',
'Income_group_High']
```

In [495…

```
# Filter continuous variables
# Filter continuous variables
cont = [v for v in cols if v not in cat_var and v not in del_var and v != 'Customer_id']
```

In [496…

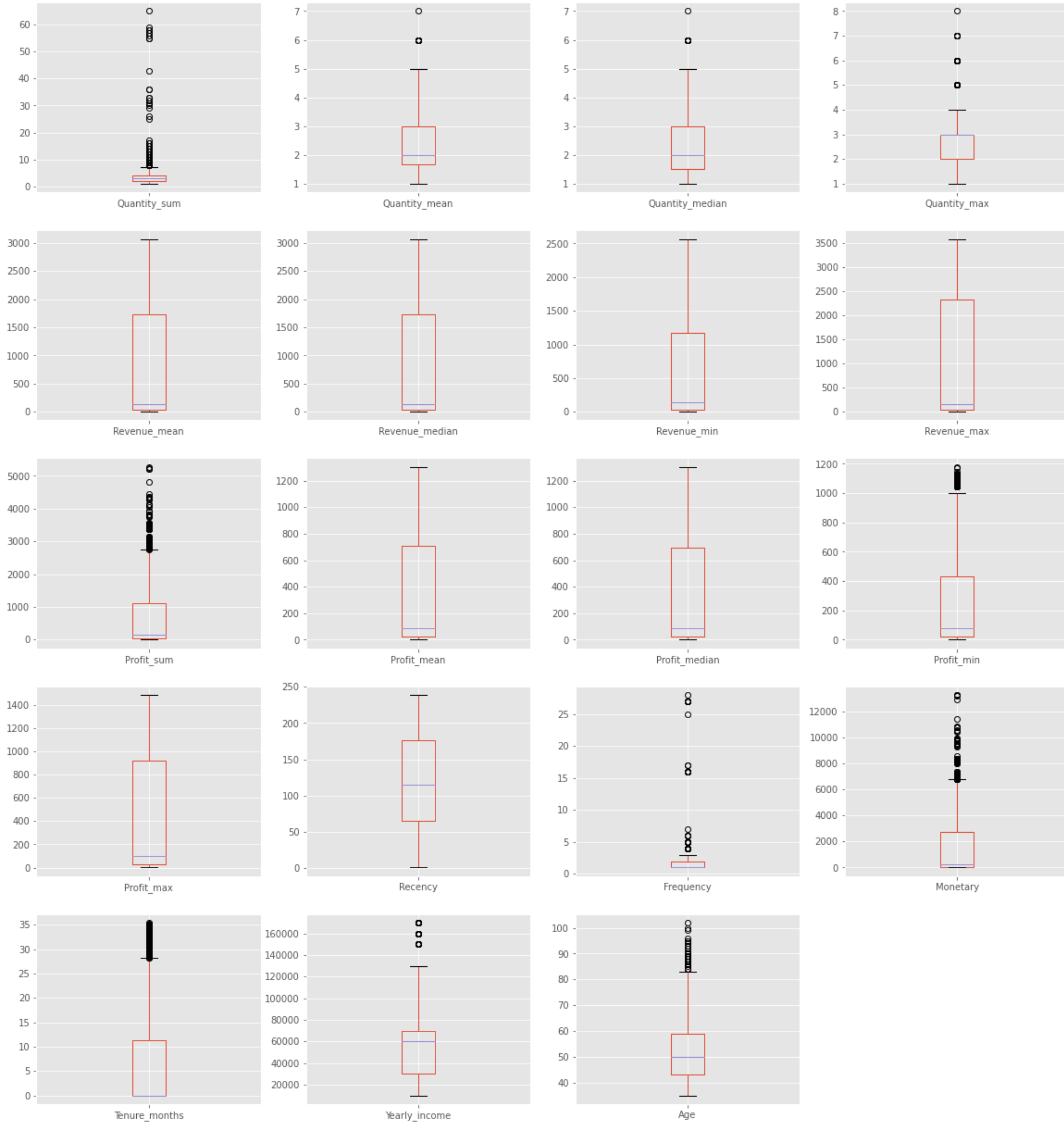
```
# Get quick stats
encoded_score_data[cont].describe().transpose()
```

Out[496…

	count	mean	std	min	25%	50%	75%	max
Quantity_sum	6409.00	3.55	3.07	1.00	2.00	3.00	4.00	65.00
Quantity_mean	6409.00	2.26	0.89	1.00	1.67	2.00	3.00	7.00
Quantity_median	6409.00	2.25	0.91	1.00	1.50	2.00	3.00	7.00
Quantity_max	6409.00	2.65	1.10	1.00	2.00	3.00	3.00	8.00
Revenue_mean	6409.00	874.78	991.47	2.29	40.63	135.46	1735.98	3072.53
Revenue_median	6409.00	873.49	991.09	2.29	39.98	135.46	1739.46	3072.53
Revenue_min	6409.00	714.92	852.50	2.29	36.59	134.93	1173.96	2566.80
Revenue_max	6409.00	1036.60	1208.36	2.29	49.97	161.29	2319.99	3578.27
Profit_sum	6409.00	686.89	925.47	1.43	30.66	145.37	1120.90	5273.81
Profit_mean	6409.00	361.45	407.01	1.43	24.71	83.55	707.35	1303.19
Profit_median	6409.00	361.37	407.95	1.43	24.40	83.55	697.23	1303.19
Profit_min	6409.00	291.50	347.85	1.43	21.80	79.68	429.31	1178.94
Profit_max	6409.00	431.78	499.10	1.43	25.03	100.45	924.56	1487.84
Recency	6409.00	120.27	65.58	1.00	65.00	115.00	176.00	239.00
Frequency	6409.00	1.61	1.35	1.00	1.00	1.00	2.00	28.00
Monetary	6409.00	1655.13	2231.88	2.29	58.05	252.32	2749.32	13295.38
Tenure_months	6409.00	6.20	9.15	0.00	0.00	0.00	11.30	35.42
Yearly_income	6409.00	58180.68	32627.80	10000.00	30000.00	60000.00	70000.00	170000.00
Age	6409.00	51.77	11.45	35.00	43.00	50.00	59.00	102.00

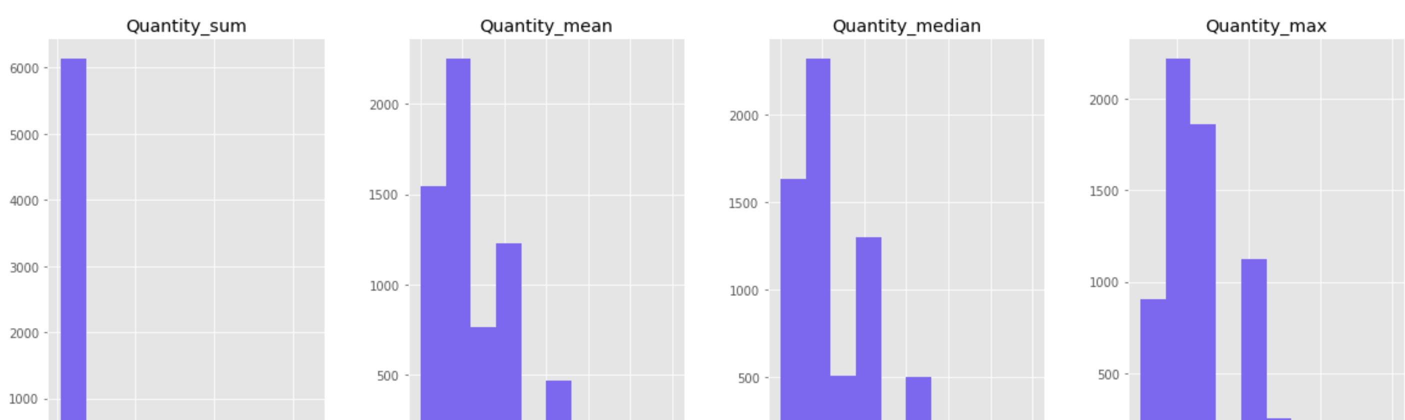
In [497…

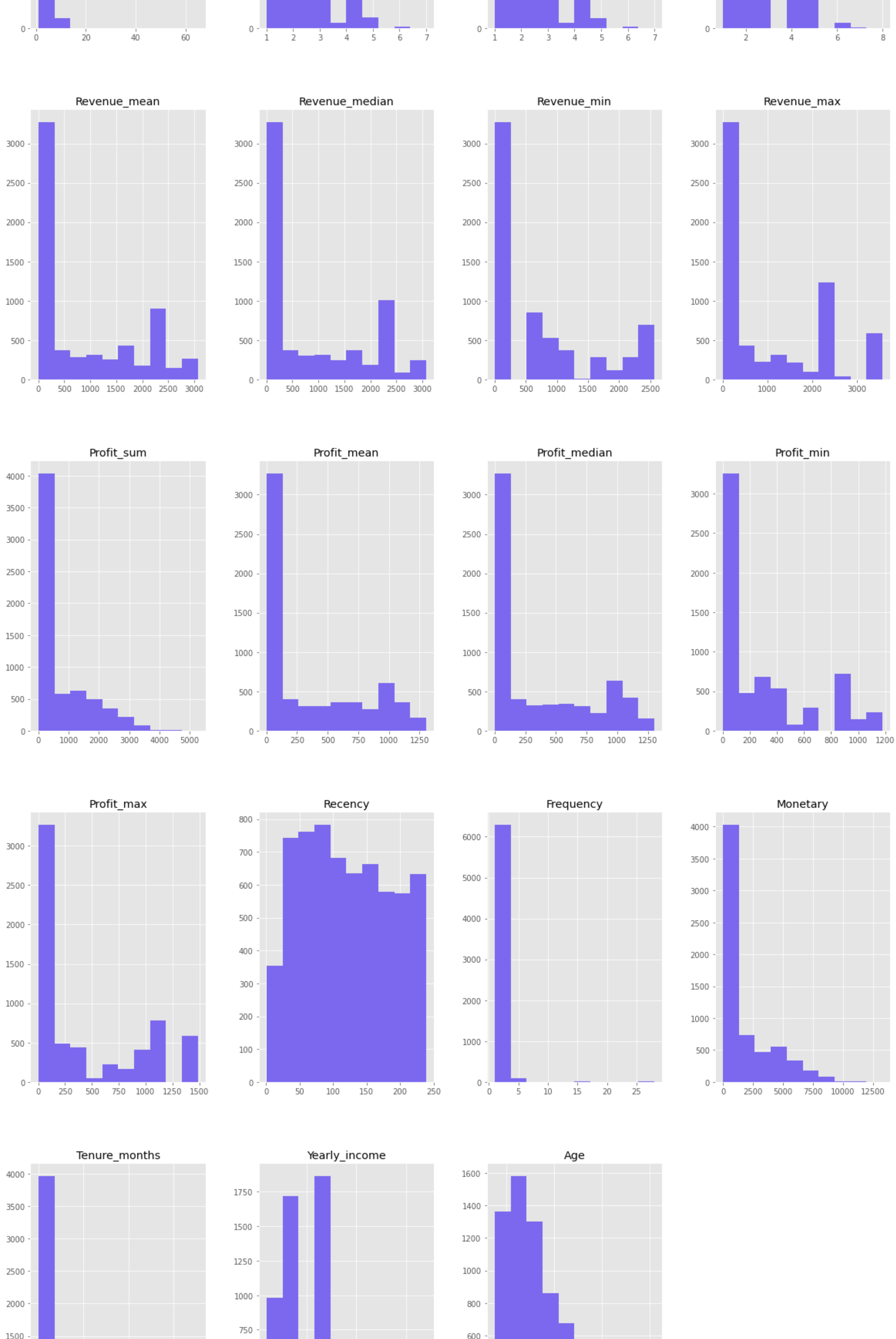
```
# Plot boxplots
encoded_score_data[cont].plot(kind='box', subplots=True, figsize=(20,40),
                      layout=(9,4), sharex=False, sharey=False)
plt.show()
```

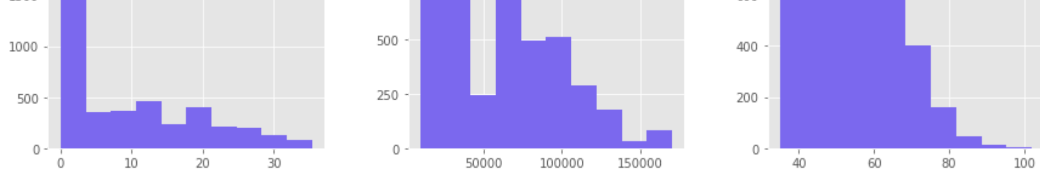


In [498…

```
# PLot histograms
encoded_score_data[cont].hist(figsize=(20,40), color = 'mediumslateblue')
pass
```




Revenue_mean
Revenue_median
Revenue_min
Revenue_max
Profit_sum
Profit_mean
Profit_median
Profit_min
Profit_max
Recency
Frequency
Monetary
Tenure_months
Yearly_income
Age



In [499…

```
# Check if all encoded features of model dataset are included in score dataset
result = all(v in encoded_score_var for v in final_features)
if result:
    print('All encoded features of model dataset are included in score dataset.')
else:
    print('Not all encoded features of model dataset are included in score dataset.')
```

```
Not all encoded features of model dataset are included in score dataset.
```

In [500…

```
# Find which encoded features are not present in score dataset
# because we must score our model based on features used in building it.
missing_var = set(final_features) - set(encoded_score_var)
missing_var
```

Out[500…

```
{'RFM_segment_442'}
```

In [501…

```
# Create missing fields and assign 0 to all rows
for v in missing_var:
    encoded_score_data[v] = 0
```

In [502…

```
# Create the final score dataframe with all selected features and Customer id
score_df = encoded_score_data[final_features]
score_df.insert(0, 'Customer_id', encoded_score_data['Customer_id'])
score_df.head()
```

Out[502…

	Customer_id	RFM_cluster_3	RFM_status_Green	RFM_status_Gold	RFM_score_5	RFM_score_8	RFM_cluster_0
4071	15071	0	0	1	1	0	0
14598	25598	0	0	0	0	0	1
11148	22148	1	0	0	0	0	0
10998	21998	1	0	0	0	0	0
8239	19239	0	0	0	0	1	1

Model deployment

In [503…

```
loaded_model = joblib.load('final_churning_model')
loaded_model
```

Out[503…

```
GradientBoostingClassifier(max_depth=4, max_features='sqrt',
                           min_samples_split=10, random_state=53)
```

In [504…

```
# Make predictions
features = score_df.loc[:, score_df.columns != 'Customer_id']
predictions = loaded_model.predict(features)
# Add a new column to the dataframe with the predictions
score_df.insert(1, "Churn", predictions, True)
score_df.head()
```

Out[504...

	Customer_id	Churn	RFM_cluster_3	RFM_status_Green	RFM_status_Gold	RFM_score_5	RFM_score_8	RFM_cl
4071	15071	0	0	0	1	1	0	
14598	25598	0	0	0	0	0	0	
11148	22148	0	1	0	0	0	0	
10998	21998	0	1	0	0	0	0	
8239	19239	0	0	0	0	0	1	

In [505...

```
# Show how many customers are predicted to churn (1) and not to churn (0)
score_df.groupby('Churn').size()
```

Out[505...

```
Churn
0    5837
1     572
dtype: int64
```

In [506...

```
# Filter potential churners
predicted_churners = score_df[score_df['Churn'] == 1]
```

In [507...

```
# get the percentage of churner predicted by our model
100* len(predicted_churners )/len(score_df)
```

Out[507...

```
8.924949290060852
```

In [508...

```
#### only 9% of the score_ data is predicted to churn by our model
```

In [509...

```
predicted_churners.head()
```

Out[509...

	Customer_id	Churn	RFM_cluster_3	RFM_status_Green	RFM_status_Gold	RFM_score_5	RFM_score_8	RFM_cl
3543	14543	1	1	1	0	0	0	
8273	19273	1	1	1	0	0	0	
7310	18310	1	1	1	0	0	0	
14745	25745	1	1	0	0	0	0	
18469	29469	1	1	0	0	0	0	

In [510...

```
# Estimate commercial value if our strategy is successful in preventing churning
int(annual_customer_rev * len(predicted_churners))
```

Out[510...

```
1129897
```

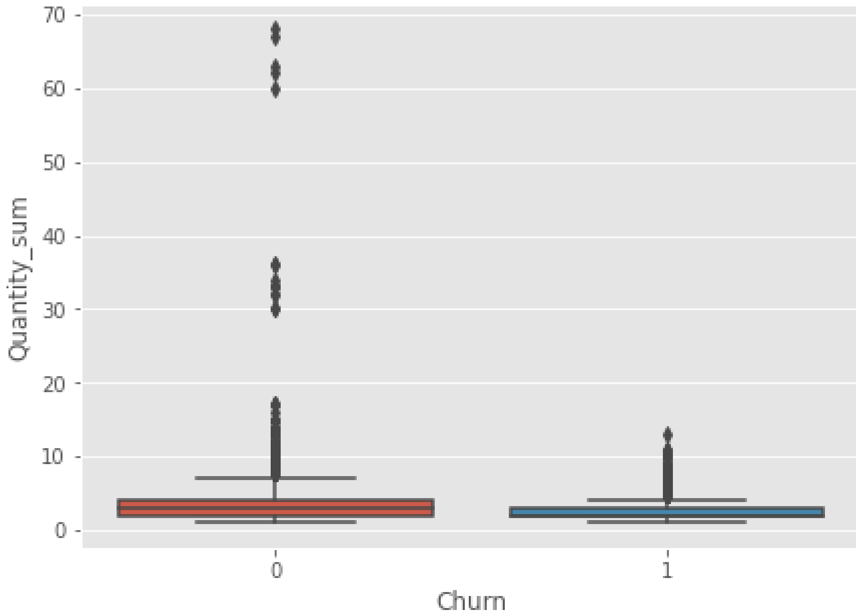
Customer profiling

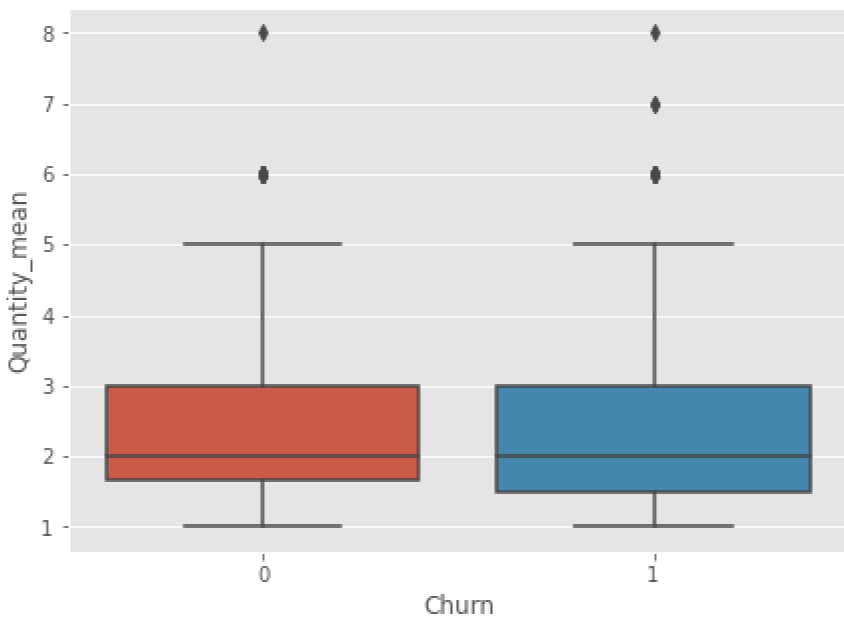
Desciptive statistics

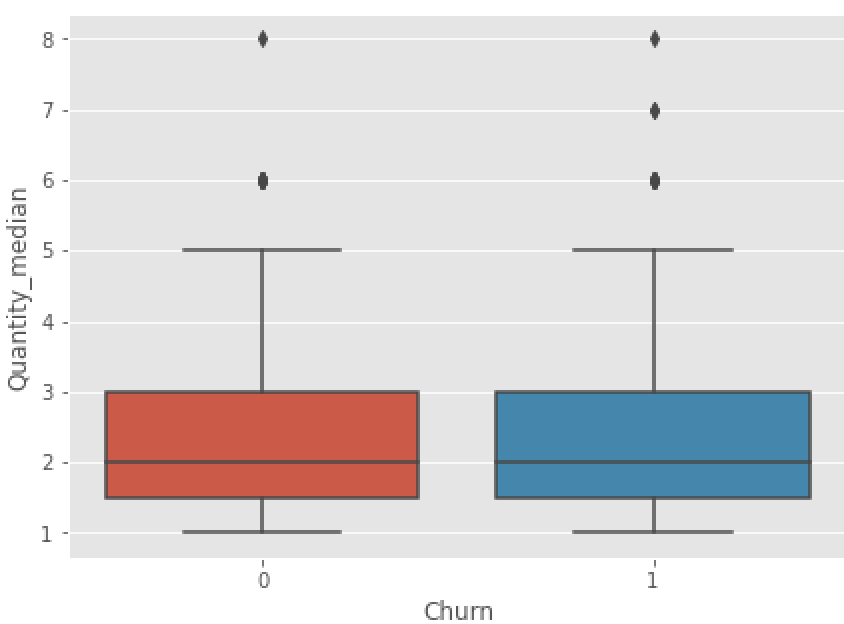
In [511...

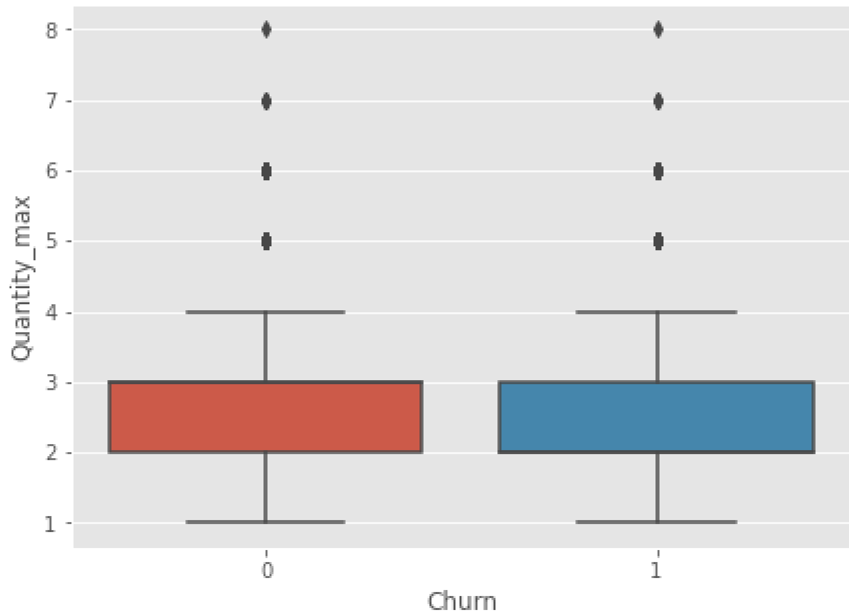
```
# Plot boxplots by Churn
plt.rcParams['figure.figsize'] = (7, 5)
cont_df = model_data[cont]
```

```python
for i, col in enumerate(cont_df .columns):
    plt.figure(i)
    sns.boxplot(data = cont_df, y = col , x = model_data['Churn'])
```

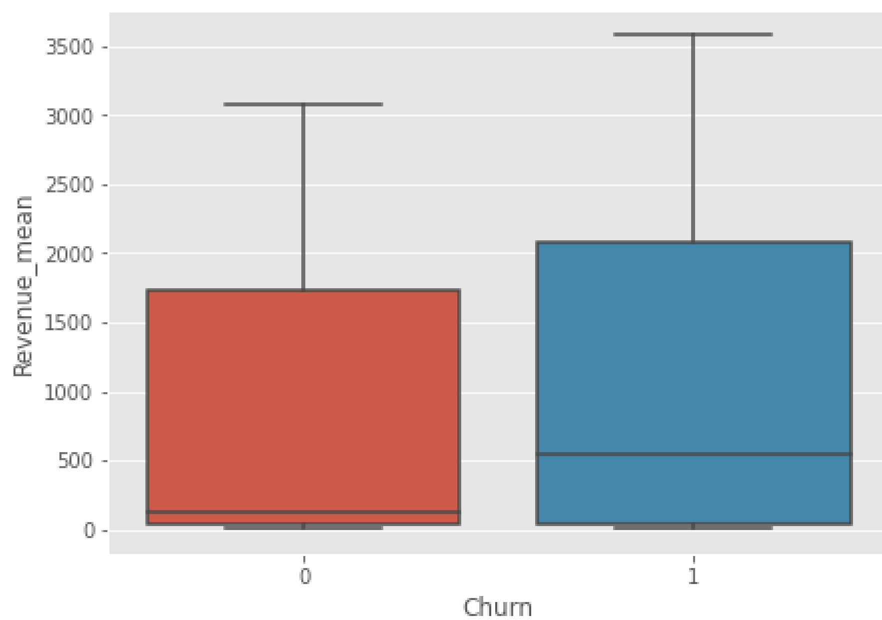




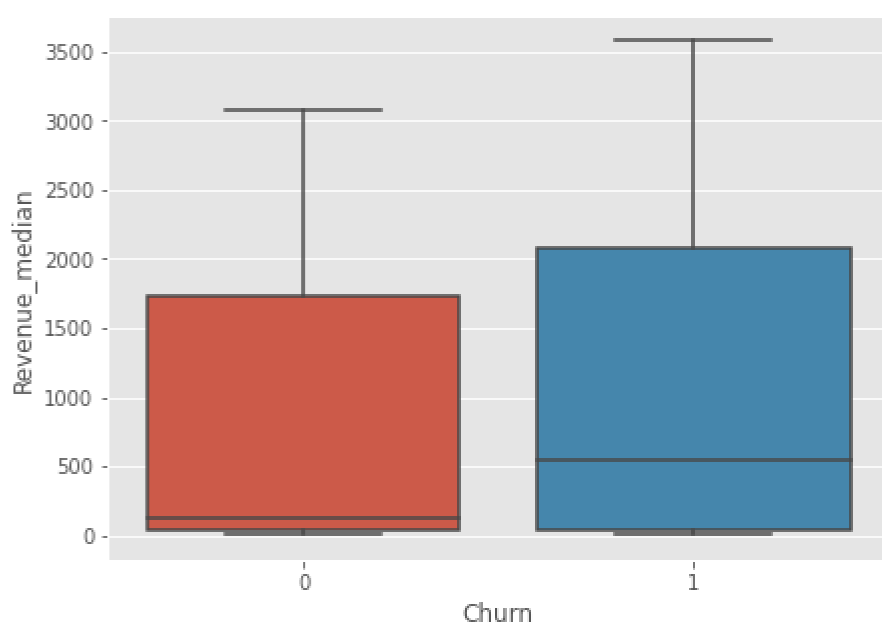




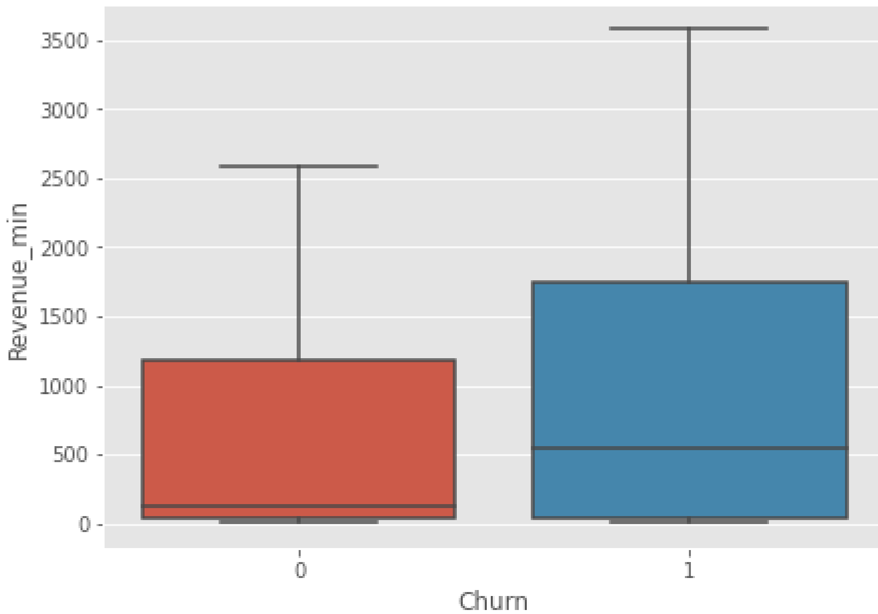
Quantity_max
8
7
6
5
4
3
2
1
0
1
Churn

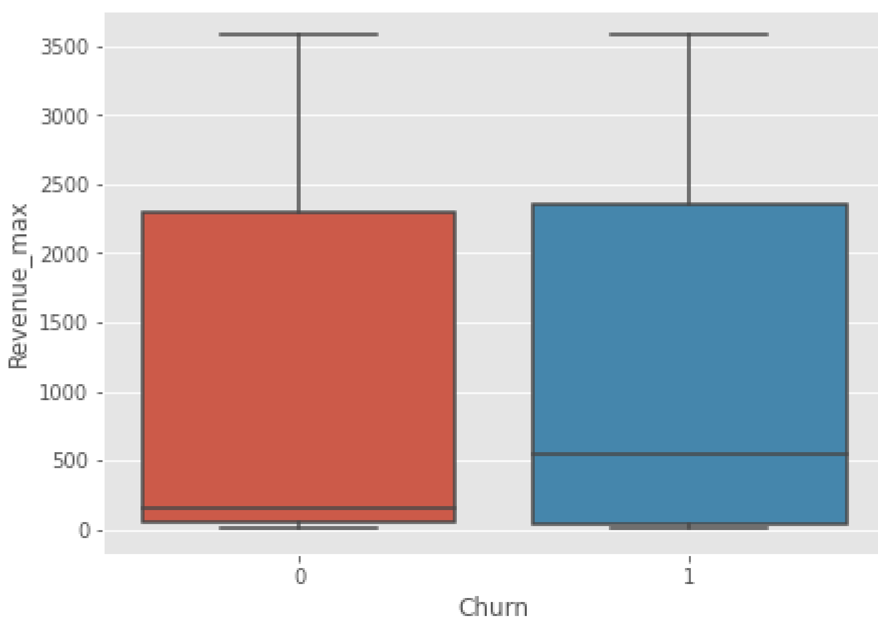


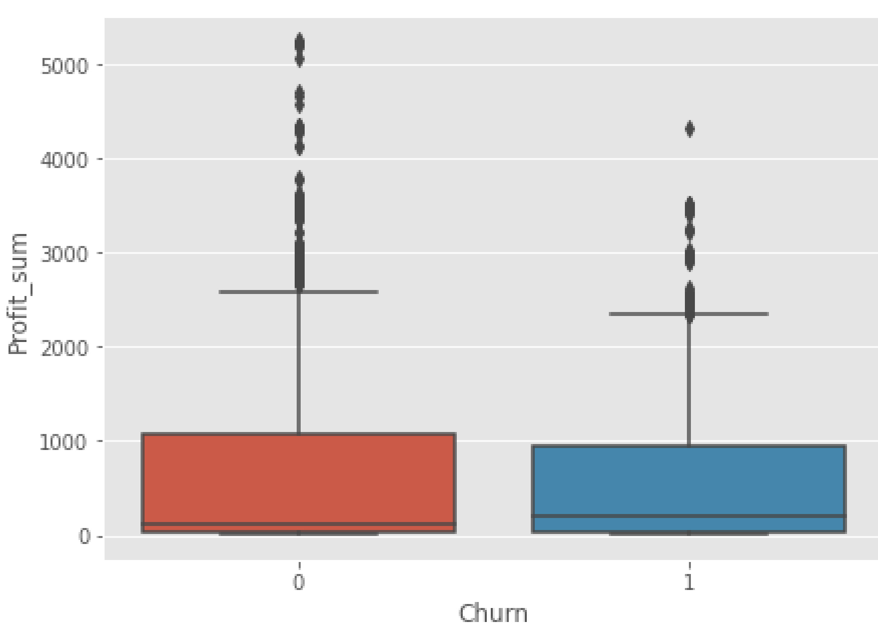
Revenue_mean
3500
3000
2500
2000
1500
1000
500
0
0
1
Churn

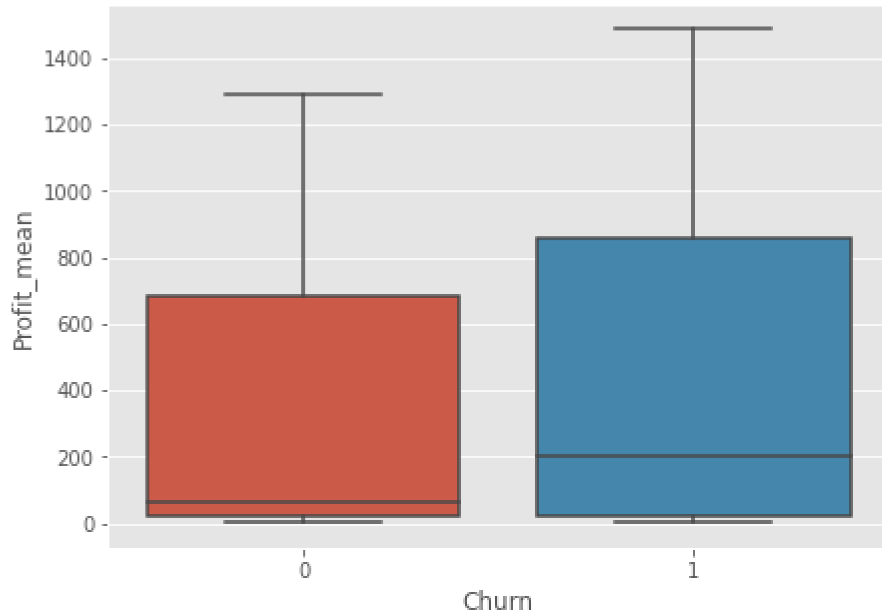


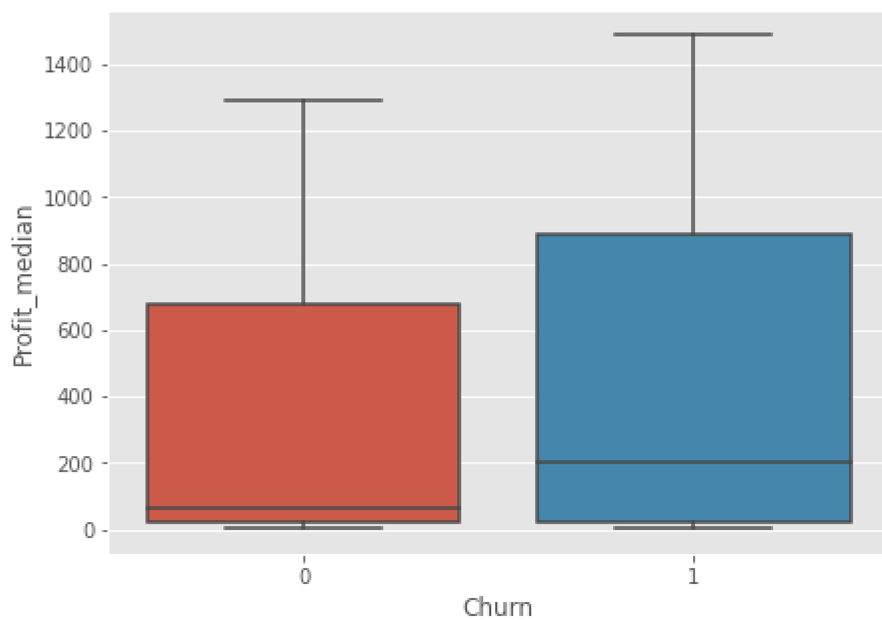
Revenue_median
3500
3000
2500
2000
1500
1000
500
0
0
1
Churn

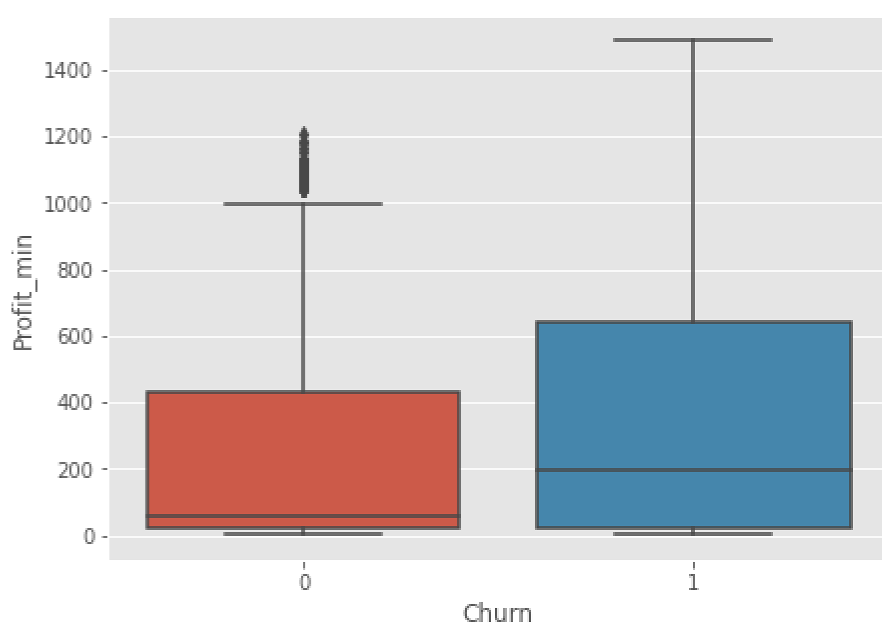

Revenue_min
0
500
1000
1500
2000
2500
3000
3500
0
1
Churn

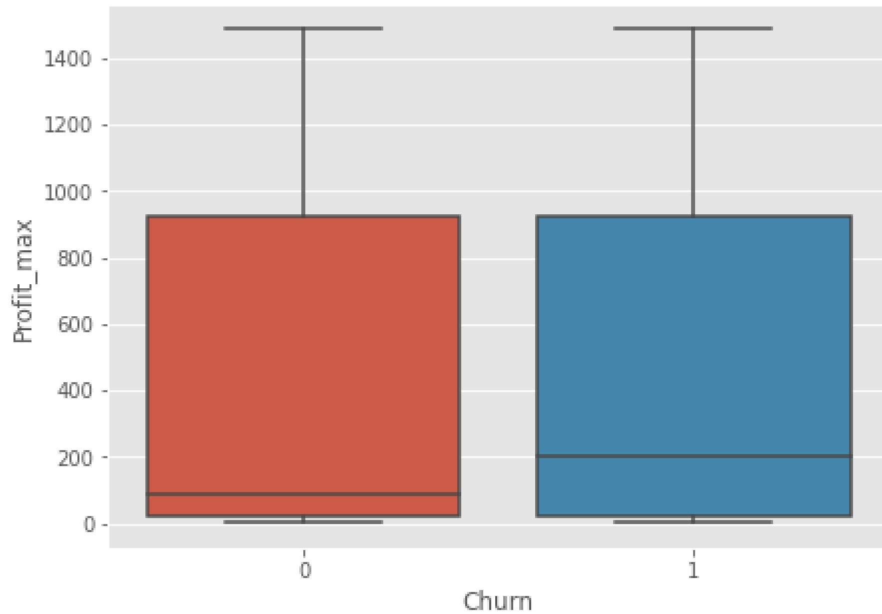

Revenue_max
0
500
1000
1500
2000
2500
3000
3500
0
1
Churn

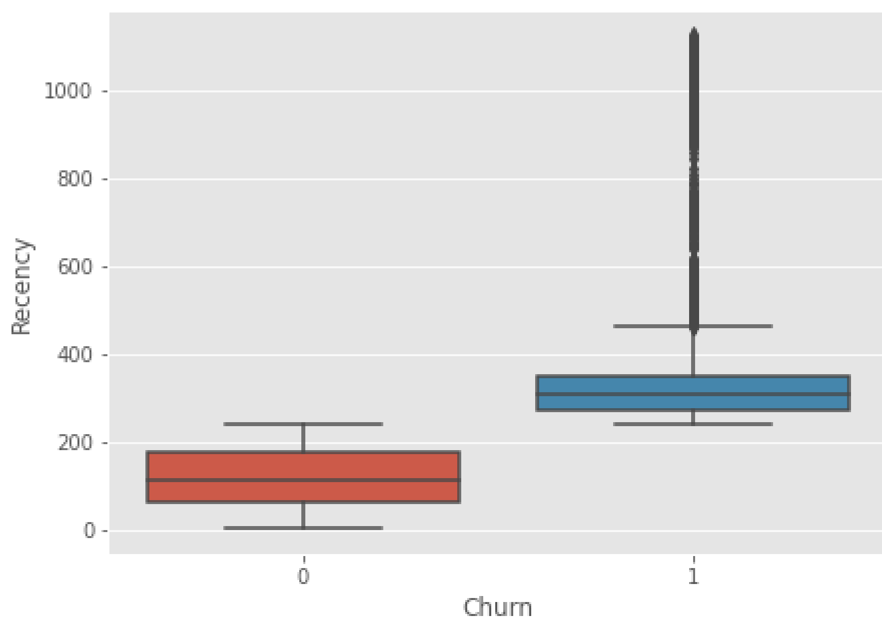

Profit_sum
0
1000
2000
3000
4000
5000
0
1
Churn

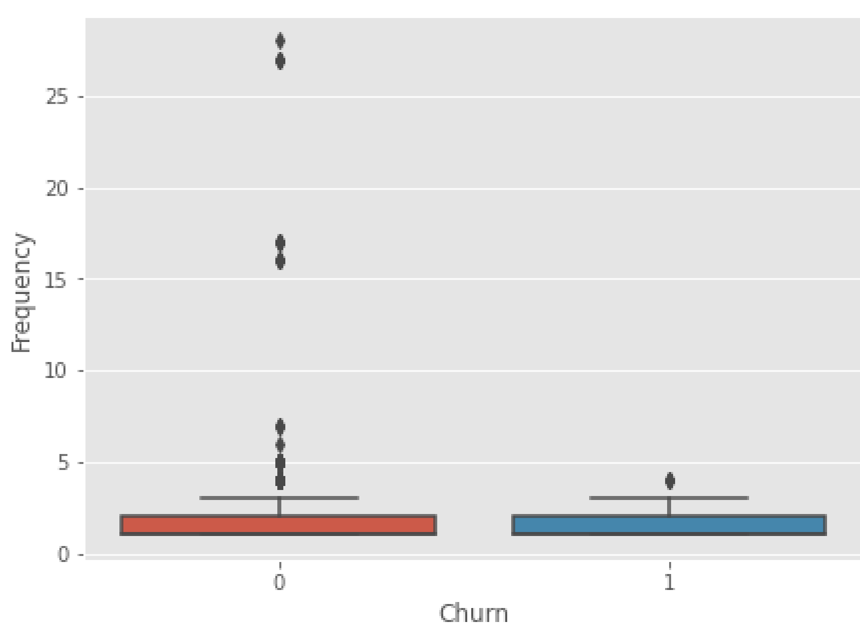

1400
1200
1000
800
600
400
200
0
Profit_mean
0
1
Churn

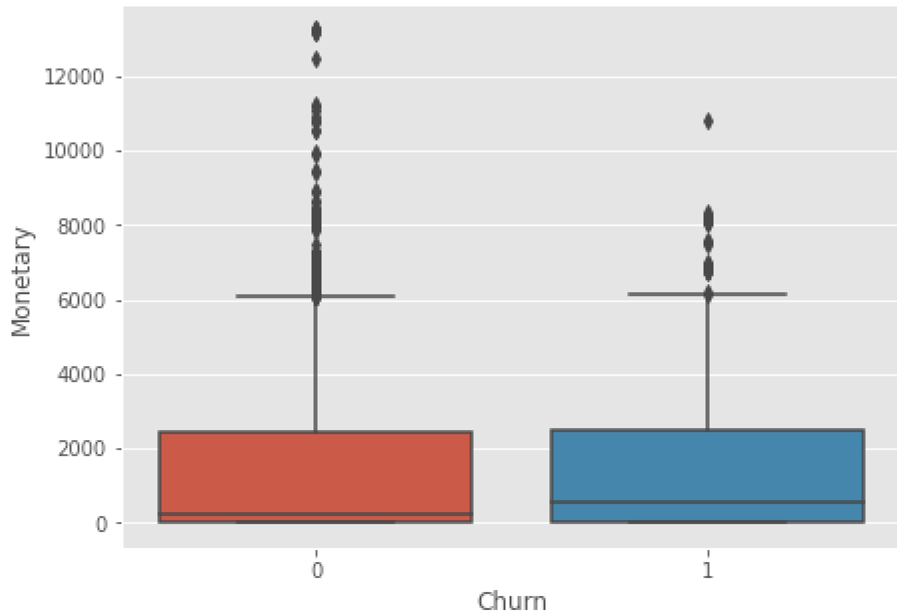

1400
1200
1000
800
600
400
200
0
Profit_median
0
1
Churn

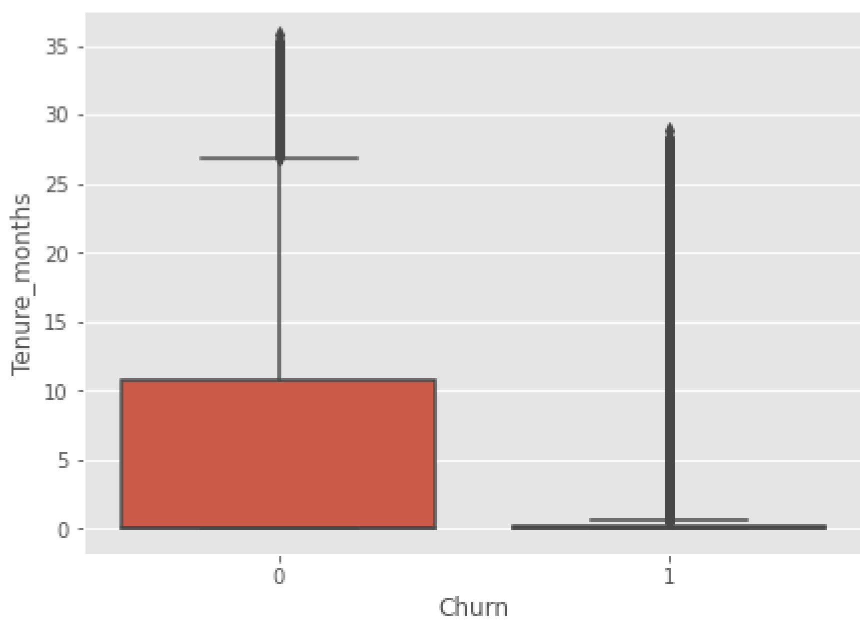

1400
1200
1000
800
600
400
200
0
Profit_min
0
1
Churn

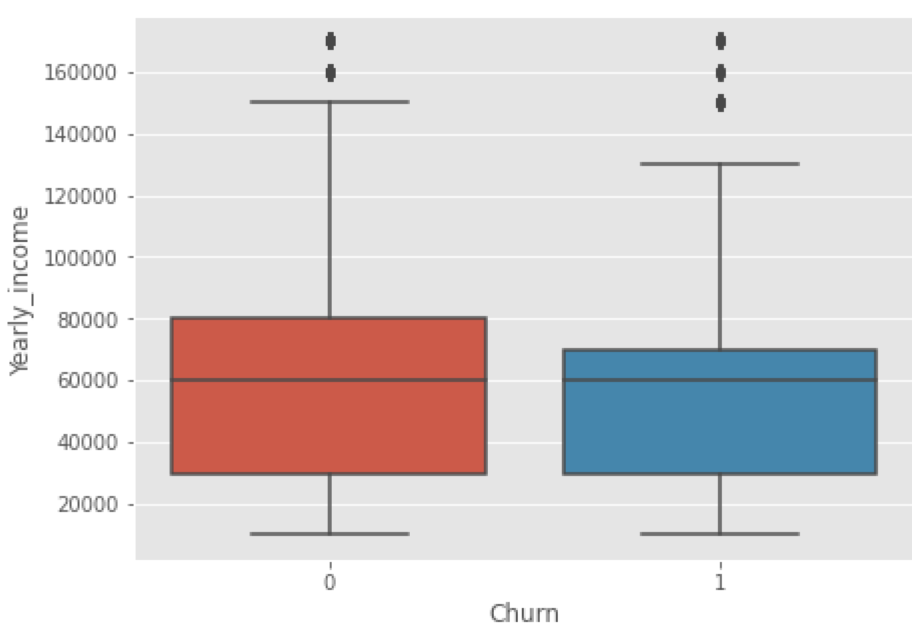

Profit_max
Churn
0
1
0
200
400
600
800
1000
1200
1400

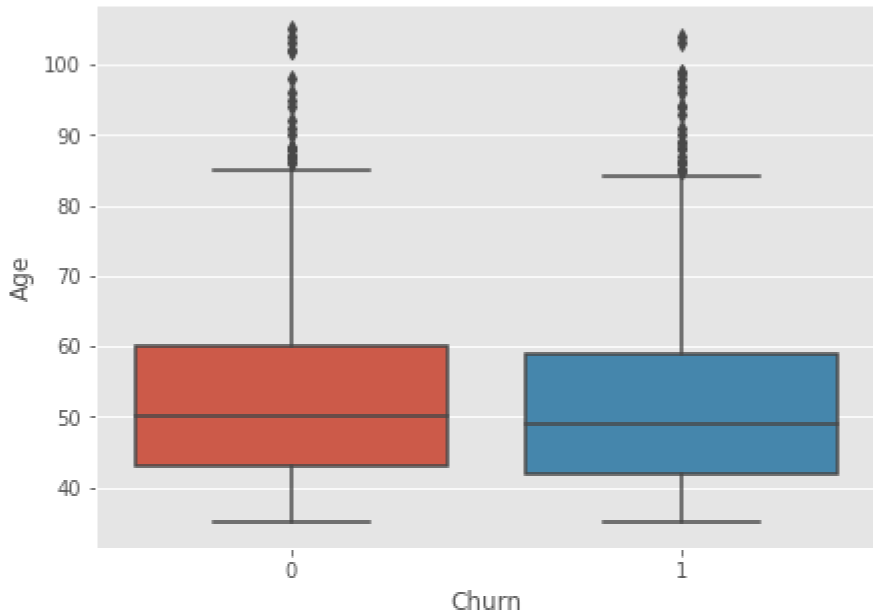

Recency
Churn
0
1
0
200
400
600
800
1000


Frequency
Churn
0
1
0
5
10
15
20
25


Monetary
12000
10000
8000
6000
4000
2000
0
0
1
Churn


Tenure_months
35
30
25
20
15
10
5
0
0
1
Churn


Yearly_income
160000
140000
120000
100000
80000
60000
40000
20000
0
1
Churn

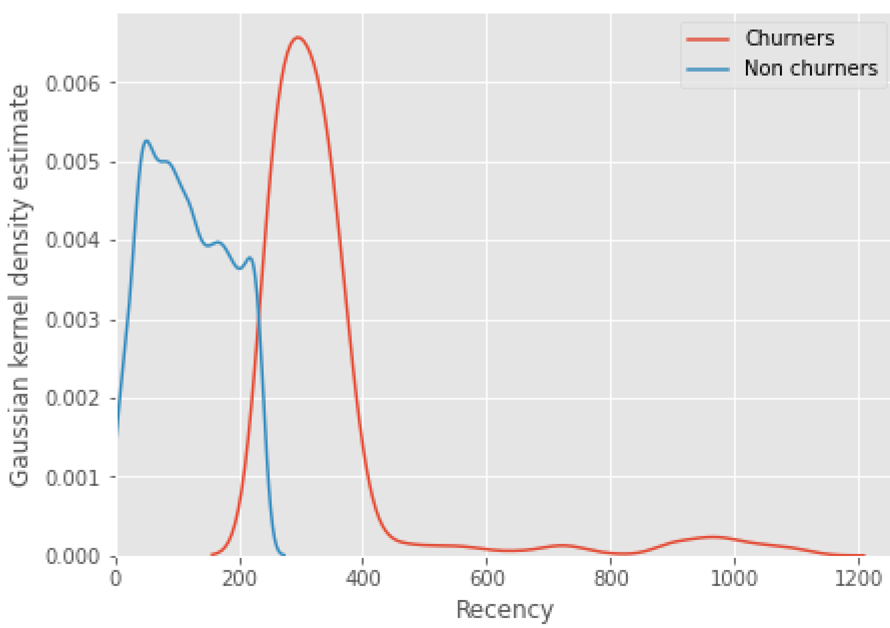


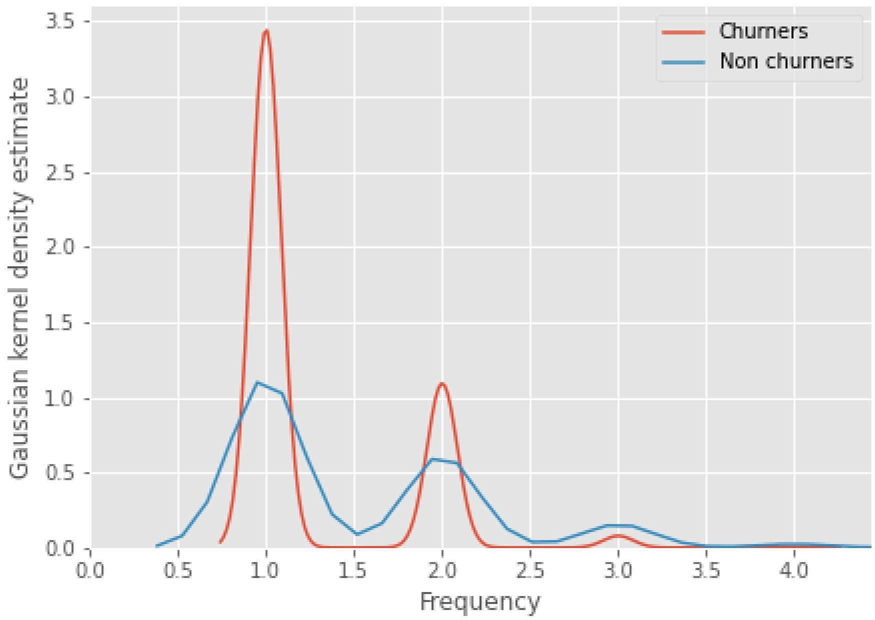
In [512…

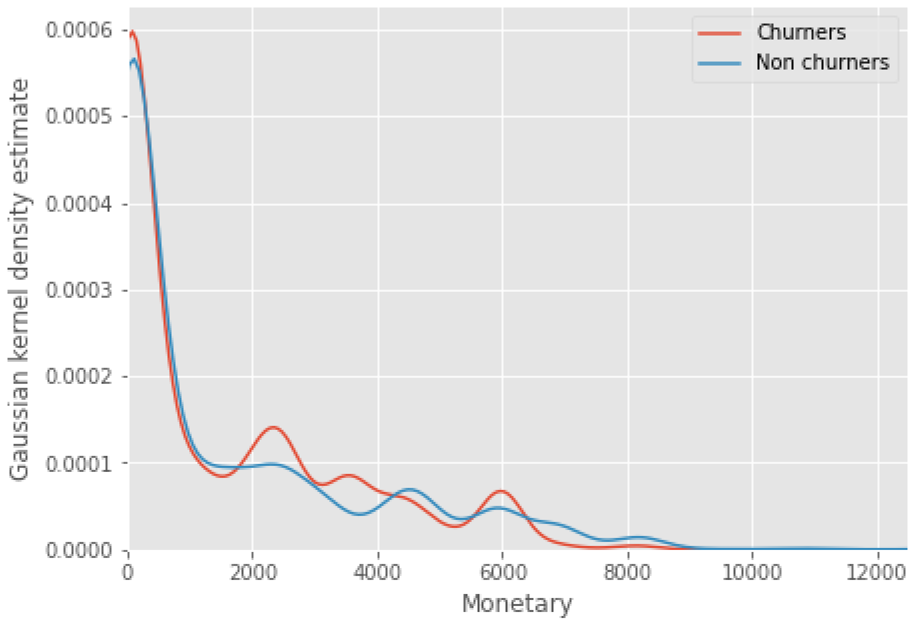
```
# There are notable differences in revenue, profit, tenure and -unsuprisingly- recency bet
```

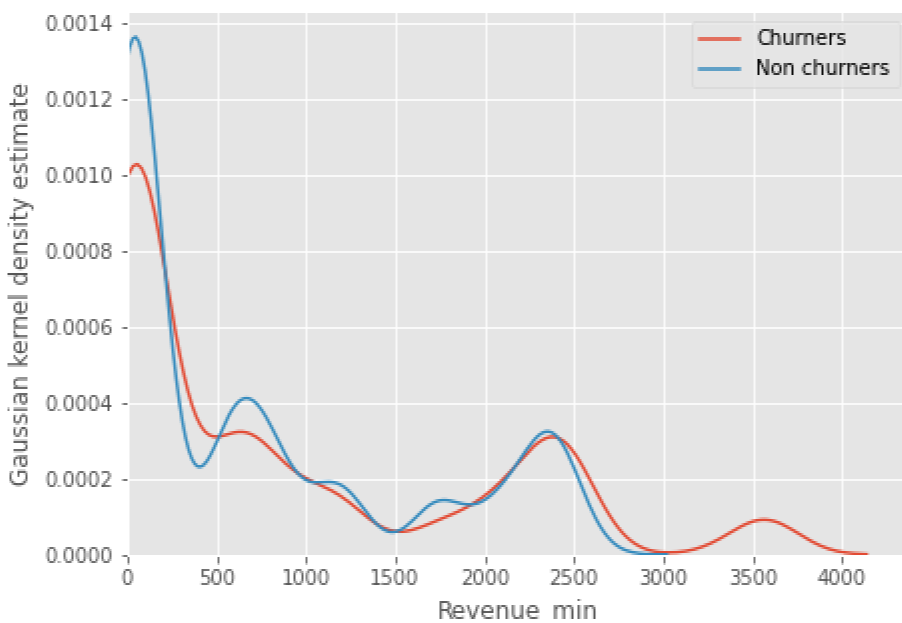
In [513…

```
# Plot KDE plot (Gaussian kernel density estimate) by Churn for select variables
select_var = ['Recency', 'Frequency', 'Monetary', 'Revenue_min']
for i, v in enumerate(select_var):
    plt.figure(i)
    sns.distplot(model_data[model_data['Churn']==1][v], label="Churners", hist= False).set
    sns.distplot(model_data[model_data['Churn']==0][v], label="Non churners", hist = False
    plt.ylabel('Gaussian kernel density estimate')
    plt.legend();
```





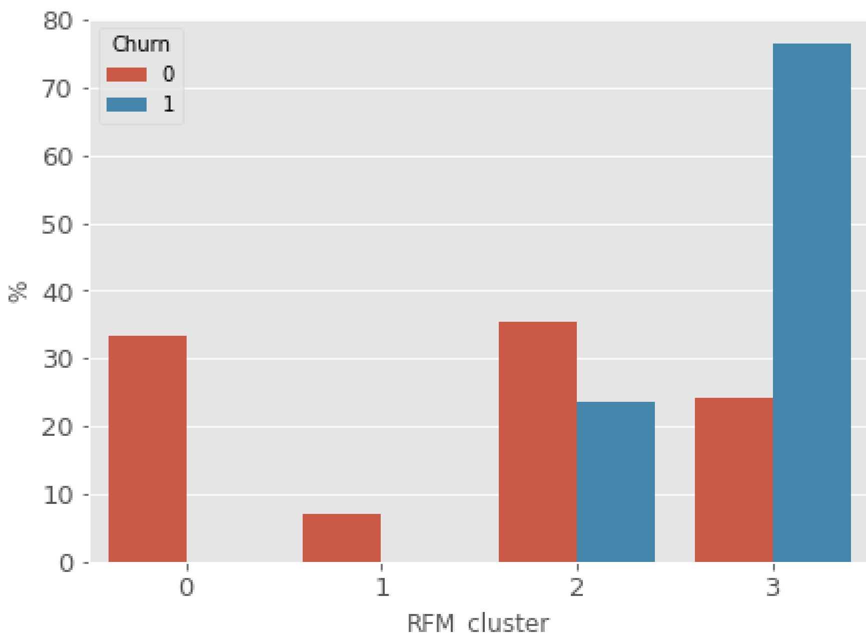


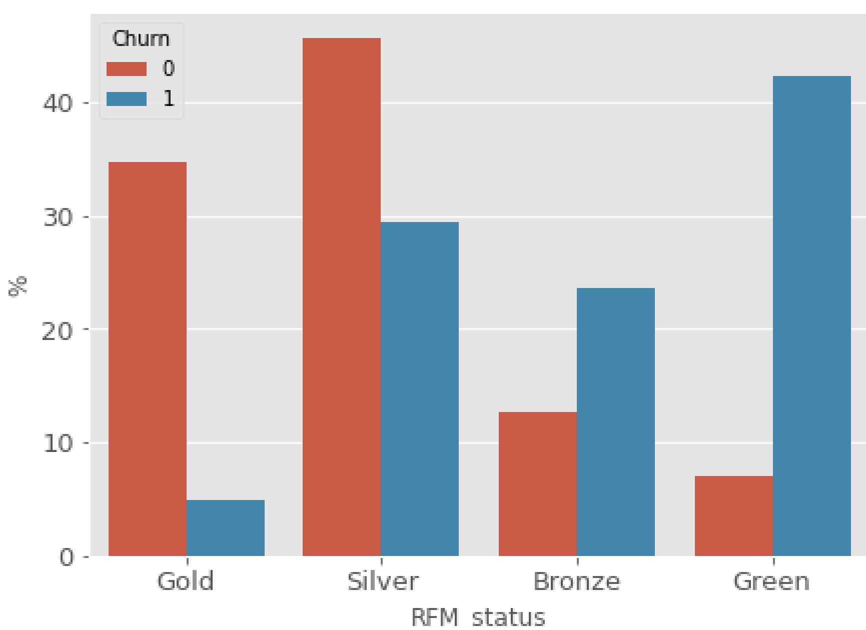


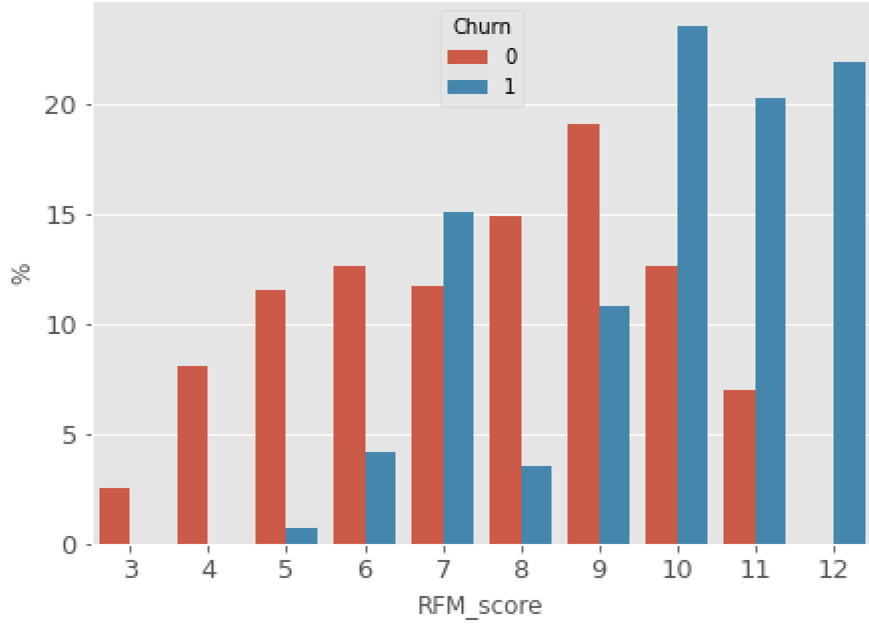
In [514…

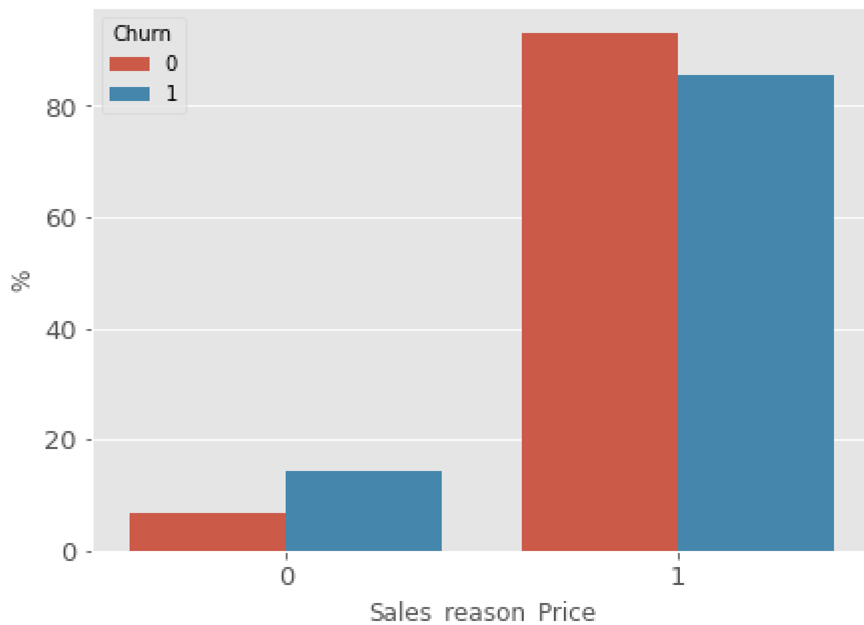
```
# Plot distribution by Churn of select categorical variables as a proportion
select_var = ["RFM_cluster", "RFM_status", "RFM_score", "Sales_reason_Price", "Income_grou
y = "%"
```

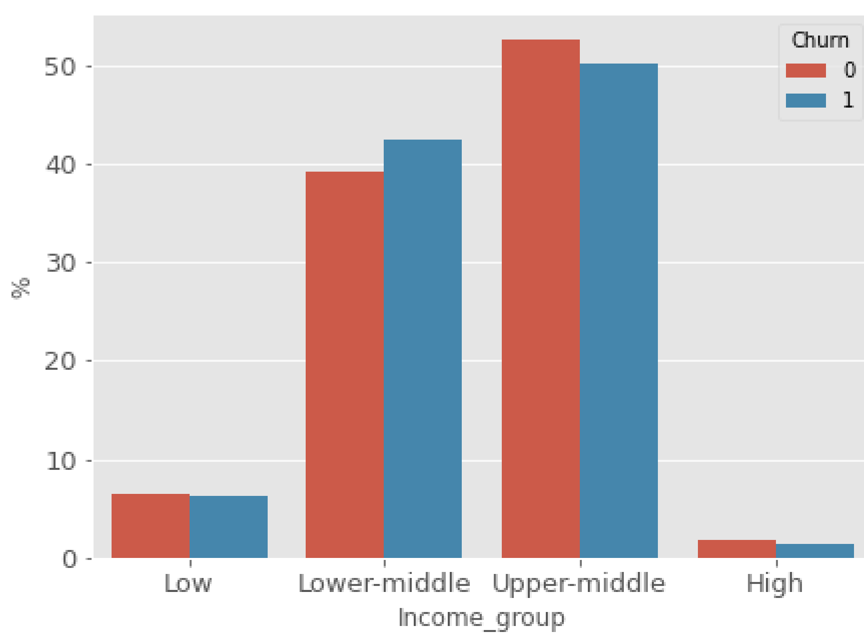
```
hue = 'Churn'

for i, v in enumerate(select_var):
    plt.figure(i)
    prop_df = (model_data[v]
               .groupby(model_data[hue])
               .value_counts(normalize=True)
               .mul(100)
               .rename(y)
               .reset_index())

    sns.barplot(x=v, y=y, hue=hue, data=prop_df)
    plt.yticks(fontsize=13)
    plt.xticks(fontsize=13);
```











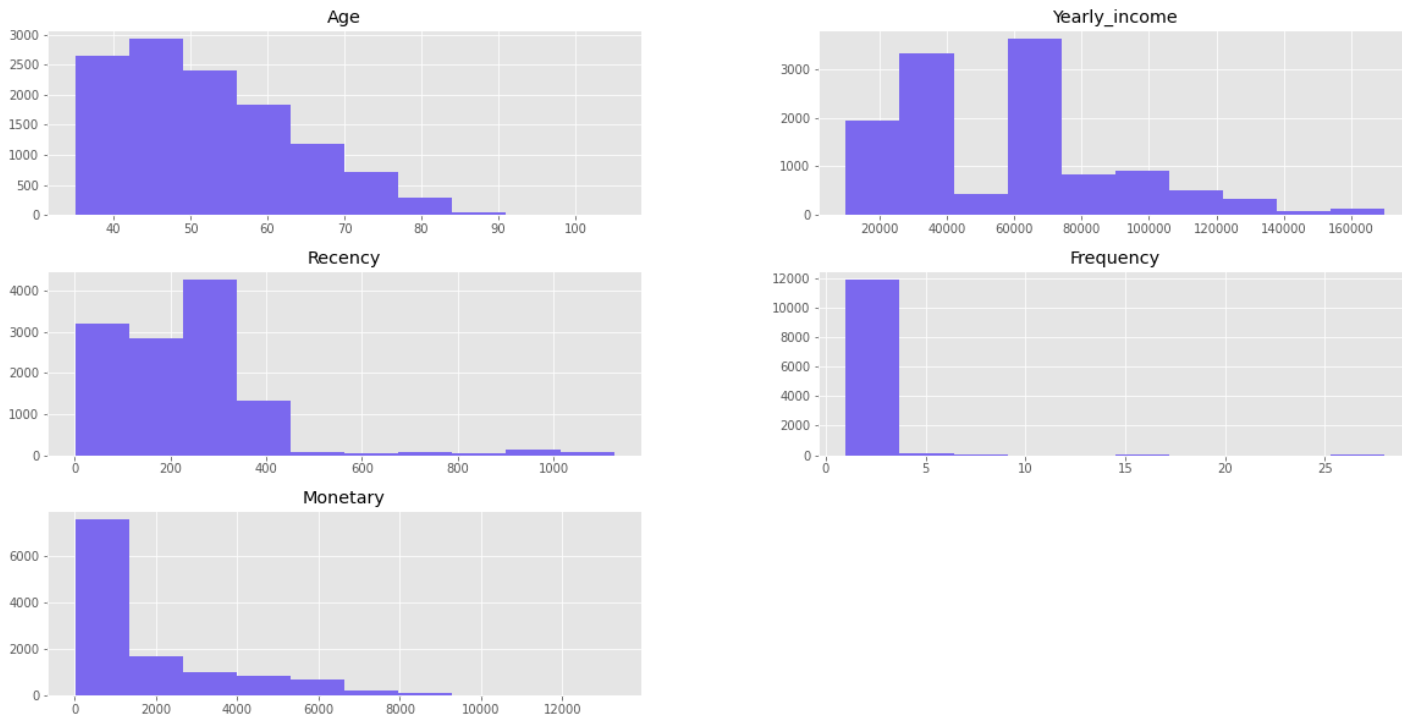
K-means clustering

In [515...

```python
# Cluster churners based on age, yearly income, recency, frequency and monetary value
feats = ['Age', 'Yearly_income', 'Recency', 'Frequency', 'Monetary']
feats_df = model_data[feats]
```

In [516…

```python
# Plot data
feats_df.hist(figsize=(20,10), color = 'mediumslateblue');
```



In [517…

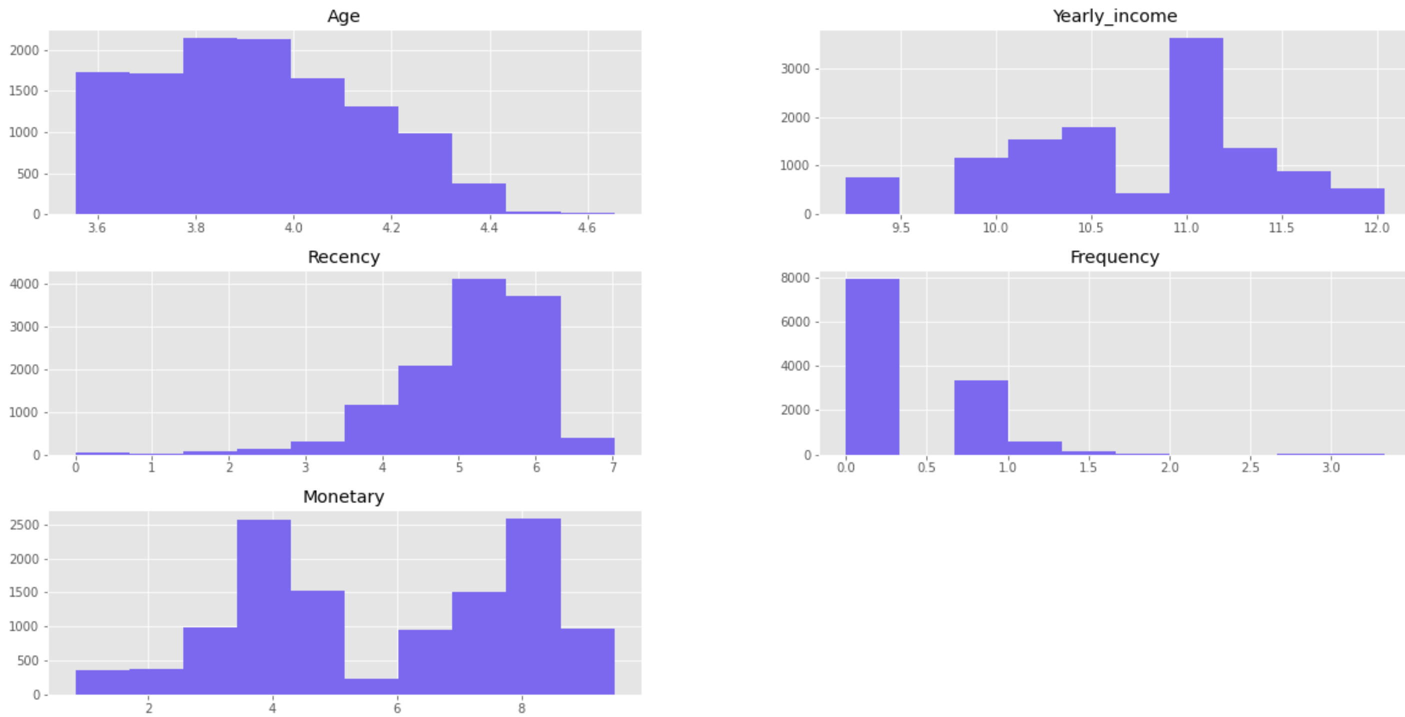
```python
feats_df.describe()
```

Out[517…

	Age	Yearly_income	Recency	Frequency	Monetary
count	12075.00	12075.00	12075.00	12075.00	12075.00
mean	51.77	56841.41	225.99	1.43	1552.88
std	11.56	32094.53	163.10	0.94	2064.03
min	35.00	10000.00	1.00	1.00	2.29
25%	42.00	30000.00	107.00	1.00	48.97
50%	50.00	60000.00	226.00	1.00	293.40
75%	59.00	70000.00	305.00	2.00	2477.78
max	105.00	170000.00	1126.00	28.00	13269.27

In [518…

```python
# No zero values, we need to normalise and scale data for the K-means model
feats_log = feats_df.apply(np.log, axis = 1).round(3)
# PLot logged data
feats_log.hist(figsize=(20,10), color = 'mediumslateblue');
```

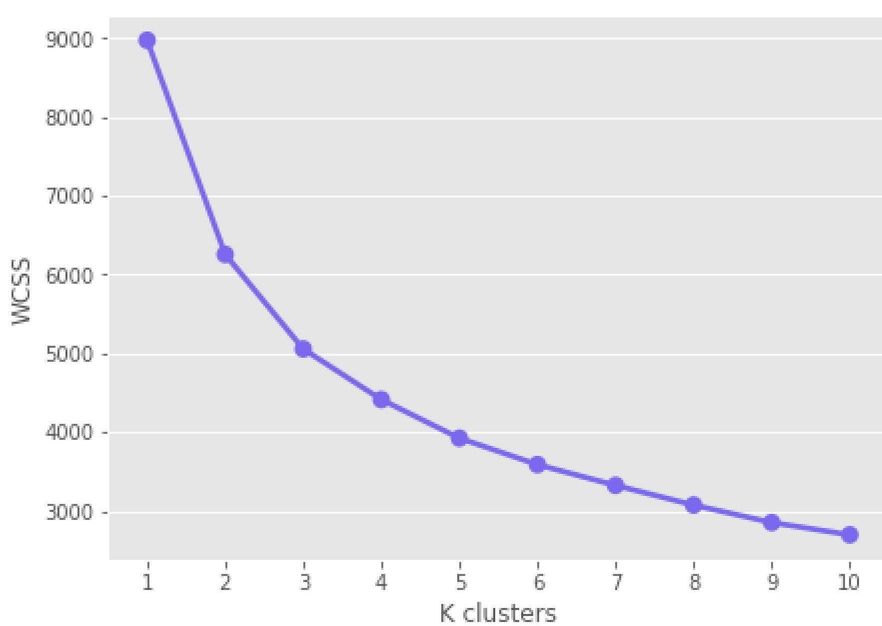


In [519…

```
# Scale data
scaler = MinMaxScaler(feature_range=(-1, 1))
feats_scaled = scaler.fit_transform(feats_log)
# Transform into a dataframe
feats_scaled = pd.DataFrame(feats_scaled, index = feats_df.index, columns = feats_log.colu
```

In [520…

```
# Call elbow_plot function to determine optimal k
elbow_plot(feats_scaled)
```



In [521…

```
kmeans = KMeans(n_clusters = 4, init= 'k-means++', max_iter= 300, random_state = seed)
kmeans.fit(feats_scaled)
# Assign the clusters to rfm dataframe
feats_df['Churn_cluster'] = kmeans.labels_
feats_df.head()
```

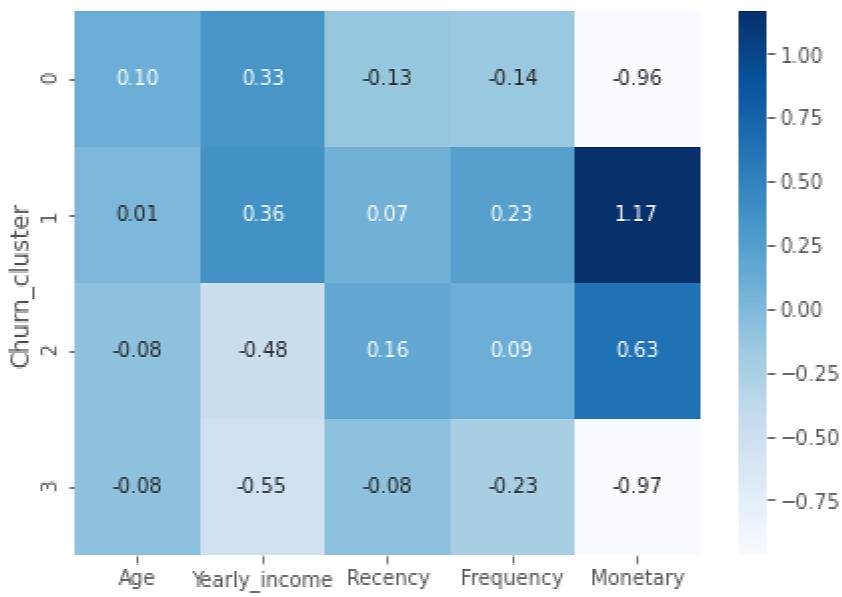
Out[521…

Age	Yearly_income	Recency	Frequency	Monetary	Churn_cluster

	Age	Yearly_income	Recency	Frequency	Monetary	Churn_cluster
0	73	70000.00	239.00	2	3351.40	1
1	36	30000.00	187.00	2	4366.41	2
2	65	70000.00	230.00	2	3373.91	1
3	58	100000.00	237.00	1	96.46	0
4	68	40000.00	29.00	1	4.99	0

In [522…

```
# Visualise clusters with heatmap
# Calculate the mean value in total
total_avg = feats_df.iloc[:, 0:5].mean()
total_avg

# Calculate the proportional gap with total mean
cluster_avg_K = feats_df.groupby('Churn_cluster').mean().iloc[:, 0:5]
prop_churners_K = cluster_avg_K/total_avg - 1

# Plot heatmap
sns.heatmap(prop_churners_K, cmap= 'Blues', fmt= '.2f', annot = True);
```

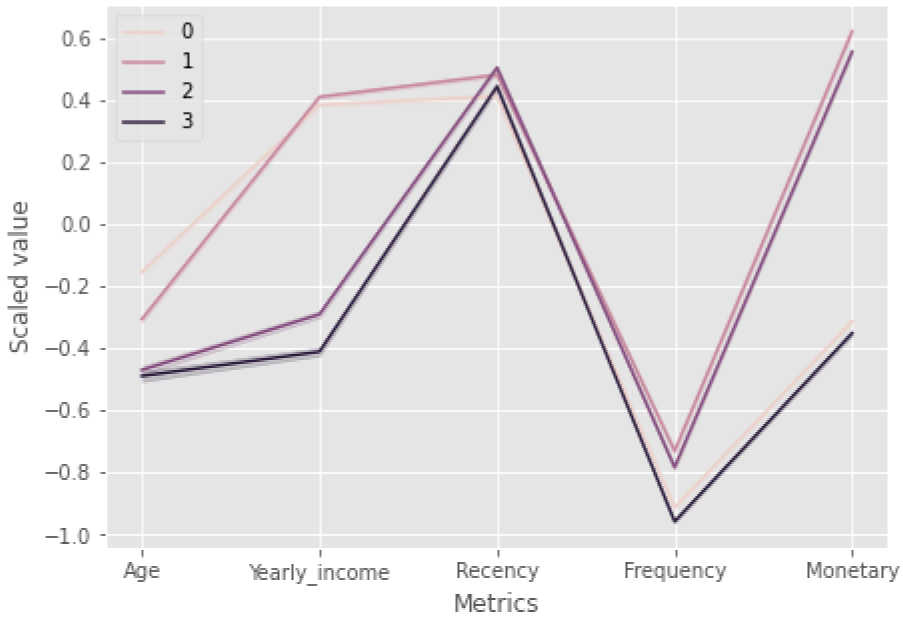


In [523…

```
# Clusters 0 and 1 have similar yearly income but very different monetary value
# The company should place particular focus to re-gain cluster 1 churners
```

In [524…

```
# Alternative way to visualise 3D data: Snake plot
# Assign cluster column
feats_scaled['Churn_cluster'] = kmeans.labels_

# Melt the dataframe
clusters_melted = pd.melt(frame= feats_scaled,
                          id_vars= ['Churn_cluster'],
                          var_name = 'Metrics',
                          value_name = 'Scaled value')

# PLot snake plot with K-Means
sns.lineplot(x = 'Metrics', y = 'Scaled value', hue = 'Churn_cluster', data = clusters_mel
plt.legend(loc = 'upper left');
```



Market basket analysis

In [ ]:

```

```

In [525…

```
# We will perform MBA for churning customers using data of their last transaction
# What's the most frequently purchased items among these transactions?
# Perhaps the company should re-assess their quality. Were churning customers unhappy with
```

In [526…

```
# Filter churners
churners = mba_data[mba_data['Churn'] == 1]
churners.shape
```

Out[526…

```
(5667, 54)
```

In [527…

```
# Get customer ids of churners
ids = churners['Customer_id'].tolist()
# Remove Order_date from index and make it a column again
transaction_df = transaction_df.reset_index()
# Filter churners of original dataframe based on customer id
mba = transaction_df[transaction_df['Customer_id'].isin(ids)]
mba.head()
```

Out[527…

	Order_date	Customer_id	Sales_order_number	Sales_order_line_number	Product	Quantity	Revenue	Cost	
0	2010-12-29	21768	SO43697	1	Road-150 Red, 62	1.00	3578.27	2171.29	1
1	2010-12-29	28389	SO43698	1	Mountain-100 Silver, 44	1.00	3399.99	1912.15	1
4	2010-12-29	11003	SO43701	1	Mountain-100 Silver, 44	1.00	3399.99	1912.15	1
7	2010-12-30	11005	SO43704	1	Mountain-100 Black, 48	1.00	3374.99	1898.09	1

	Order_date	Customer_id	Sales_order_number	Sales_order_line_number	Product	Quantity	Revenue	Cost	
8	2010-12-30	11011	SO43705	1	Mountain-100 Silver, 38	1.00	3399.99	1912.15	1

In [528…
```python
# Filter rows that contain data of the last transaction of each customer
mba = mba[mba.groupby('Customer_id')['Order_date'].transform('max') == mba['Order_date']]
mba.head()
```

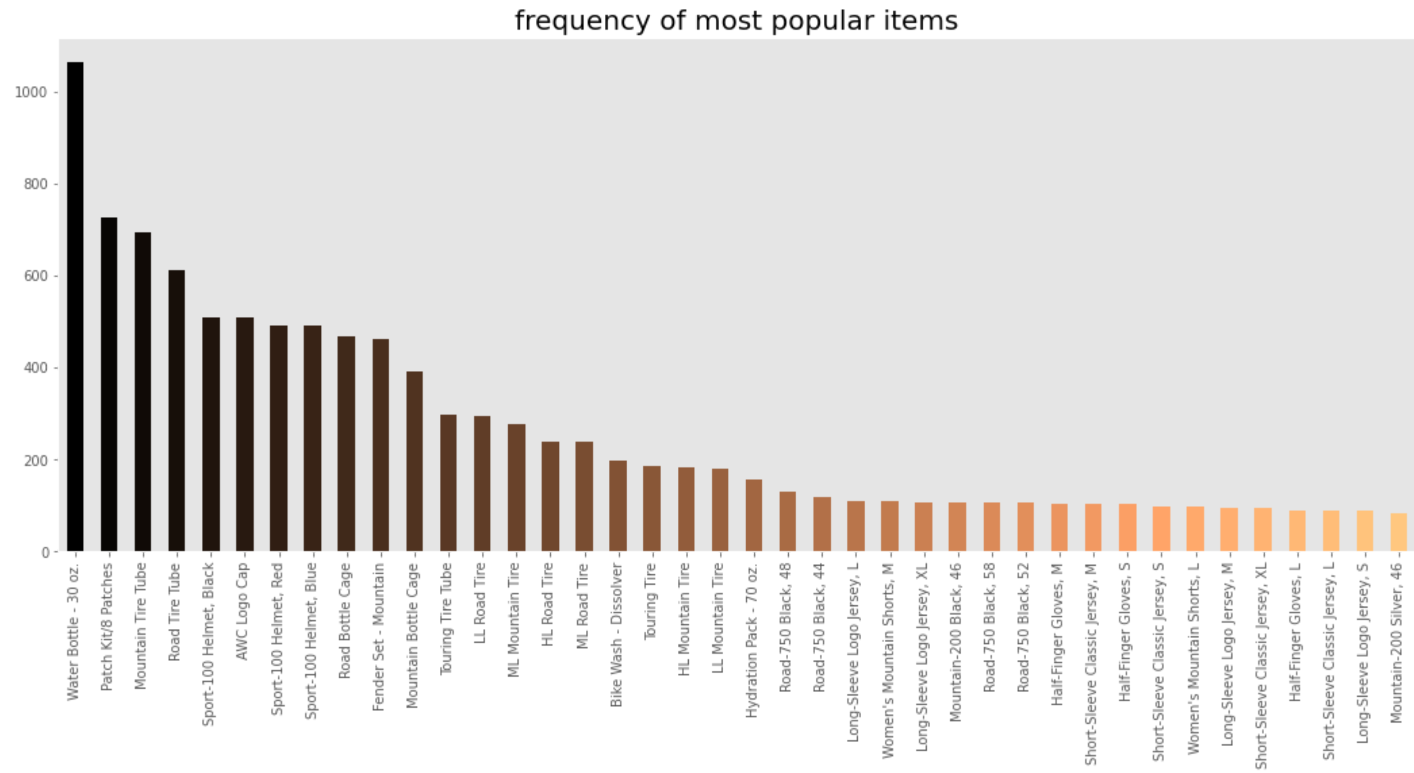
Out[528…

	Order_date	Customer_id	Sales_order_number	Sales_order_line_number	Product	Quantity	Revenue	Cost
1	2010-12-29	28389	SO43698	1	Mountain-100 Silver, 44	1.00	3399.99	1912.15
16	2011-01-02	27601	SO43713	1	Road-150 Red, 62	1.00	3578.27	2171.29
22	2011-01-03	27612	SO43719	1	Road-150 Red, 48	1.00	3578.27	2171.29
31	2011-01-06	27666	SO43728	1	Road-150 Red, 52	1.00	3578.27	2171.29
33	2011-01-06	25861	SO43730	1	Mountain-100 Silver, 44	1.00	3399.99	1912.15

In [529…
```python
from wordcloud import WordCloud

plt.rcParams['figure.figsize'] = (20, 20)
wordcloud = WordCloud(background_color = 'white', width = 1200,  height = 1200, max_words
plt.imshow(wordcloud)
plt.axis('off')
plt.title('Most Popular Items by the churners',fontsize = 20)
plt.show()
```

In [530…

```
# looking at most popular item
# looking at the frequency of most popular items

plt.rcParams['figure.figsize'] = (18, 7)
color = plt.cm.copper(np.linspace(0, 1, 40))
mba['Product'].value_counts().head(5).plot.bar(color = color)
plt.title('frequency of most popular items', fontsize = 20)
plt.xticks(rotation = 90 )
plt.grid()
plt.show()
```



In [531…

```
mba['Product'].value_counts()
```

Out[531…

```
Water Bottle - 30 oz.       1064
Patch Kit/8 Patches          725
Mountain Tire Tube           693
Road Tire Tube               610
Sport-100 Helmet, Black      508
                            ...
Mountain-100 Black, 48         2
Mountain-100 Silver, 42        1
Mountain-100 Silver, 48        1
Road-650 Red, 48               1
Road-650 Black, 62             1
Name: Product, Length: 130, dtype: int64
```

In [ ]:

```

```

In [532…

```
# Re-arrange rows and columns; replace NA values with 0
mba = (mba.groupby(['Sales_order_number', 'Product'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('Sales_order_number'))
mba.head()
```

Out[532…

Product	AWC Logo Cap	All-Purpose Bike Stand	Bike Wash - Dissolver	Classic Vest, L	Classic Vest, M	Classic Vest, S	Fender Set - Mountain	HL Mountain Tire	HL Road Tire	Half-Finger Gloves, L	...	To Bl
Sales_order_number												
SO43698	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	
SO43713	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	
SO43719	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	
SO43728	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	

Product	AWC Logo Cap	All-Purpose Bike Stand	Bike Wash - Dissolver	Classic Vest, L	Classic Vest, M	Classic Vest, S	Fender Set - Mountain	HL Mountain Tire	HL Road Tire	Half-Finger Gloves, L	...	To Bl
Sales_order_number												
SO43730	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	

5 rows × 130 columns

In [533…

```
# Encoding function
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

baskets = mba.applymap(encode_units)
```

Association rules

Generate Frequent Itemsets

Now, we are ready to generate the frequent item sets. We will set the minimum-support threshold at 7 %

In [534…

```
# Retrieve frequent items or itemsets with min frequency 3%
frequent_itemsets = apriori(baskets, min_support=0.03, use_colnames=True)
# Sort the dataframe by support
frequent_itemsets.sort_values('support', ascending = False, inplace = True)
frequent_itemsets.head(10)
```

Out[534…

	support	itemsets
19	0.19	(Water Bottle - 30 oz.)
11	0.13	(Patch Kit/8 Patches)
10	0.12	(Mountain Tire Tube)
13	0.11	(Road Tire Tube)
14	0.09	(Sport-100 Helmet, Black)
0	0.09	(AWC Logo Cap)
16	0.09	(Sport-100 Helmet, Red)
15	0.09	(Sport-100 Helmet, Blue)
12	0.08	(Road Bottle Cage)
2	0.08	(Fender Set - Mountain)

In [535…

```
# Bonus: Calculate confidence and lift among frequent item sets
# Generate rules; min threshold for lift to be 1
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.sort_values('confidence', ascending = False, inplace = True)
rules.head(10)
```

Out[535…

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
1	(Road Bottle Cage)	(Water Bottle - 30 oz.)	0.08	0.19	0.07	0.89	4.72	0.06	7.16
3	(Mountain Bottle Cage)	(Water Bottle - 30 oz.)	0.07	0.19	0.06	0.84	4.50	0.05	5.22
7	(ML Mountain Tire)	(Mountain Tire Tube)	0.05	0.12	0.03	0.65	5.32	0.03	2.51
0	(Water Bottle - 30 oz.)	(Road Bottle Cage)	0.19	0.08	0.07	0.39	4.72	0.06	1.50
2	(Water Bottle - 30 oz.)	(Mountain Bottle Cage)	0.19	0.07	0.06	0.31	4.50	0.05	1.35
4	(Mountain Tire Tube)	(Patch Kit/8 Patches)	0.12	0.13	0.03	0.27	2.08	0.02	1.19
6	(Mountain Tire Tube)	(ML Mountain Tire)	0.12	0.05	0.03	0.26	5.32	0.03	1.28
5	(Patch Kit/8 Patches)	(Mountain Tire Tube)	0.13	0.12	0.03	0.25	2.08	0.02	1.18

MBA for Non- Churner

In [536…

```
# now let get the items that are frequently purchase by non churner,this means they are ha
# may be we can reccommend those items for the churner
```

In [537…

```
# Filter non_churners
non_churners = mba_data[mba_data['Churn'] == 0]
non_churners.shape
```

Out[537…

```
(6408, 54)
```

In [538…

```
# Get customer ids of churners
ids = churners['Customer_id'].tolist()
# Remove Order_date from index and make it a column again
transaction_df = transaction_df.reset_index()
# Filter non_churners of original dataframe based on customer id
mba_non_churner = transaction_df[transaction_df['Customer_id'].isin(ids)]
mba_non_churner .head()
```

Out[538…

	index	Order_date	Customer_id	Sales_order_number	Sales_order_line_number	Product	Quantity	Revenue	
0	0	2010-12-29	21768	SO43697	1	Road-150 Red, 62	1.00	3578.27	21
1	1	2010-12-29	28389	SO43698	1	Mountain-100 Silver, 44	1.00	3399.99	19
4	4	2010-12-29	11003	SO43701	1	Mountain-100 Silver, 44	1.00	3399.99	19
7	7	2010-12-30	11005	SO43704	1	Mountain-100 Black, 48	1.00	3374.99	18

	index	Order_date	Customer_id	Sales_order_number	Sales_order_line_number	Product	Quantity	Revenue	
8	8	2010-12-30	11011	SO43705	1	Mountain-100 Silver, 38	1.00	3399.99	19

In [539…

```
# Re-arrange rows and columns; replace NA values with 0
mba_non_churner = (mba_non_churner.groupby(['Sales_order_number', 'Product'])['Quantity']
        .sum().unstack().reset_index().fillna(0)
        .set_index('Sales_order_number'))
mba_non_churner.head()
```

Out[539…

Product	AWC Logo Cap	All-Purpose Bike Stand	Bike Wash - Dissolver	Classic Vest, L	Classic Vest, M	Classic Vest, S	Fender Set - Mountain	HL Mountain Tire	HL Road Tire	Half-Finger Gloves, L	...	To Bl
Sales_order_number												
SO43697	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	
SO43698	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	
SO43701	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	
SO43704	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	
SO43705	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	

5 rows × 130 columns

Generate Frequent Itemsets

Now, we are ready to generate the frequent item sets. We will set the minimum-support threshold at 7 %

In [540…

```
# Retrieve frequent items or itemsets with min frequency 3%
frequent_itemsets = apriori(baskets, min_support=0.03, use_colnames=True)
# Sort the dataframe by support
frequent_itemsets.sort_values('support', ascending = False, inplace = True)
frequent_itemsets.head(10)
```

Out[540…

	support	itemsets
19	0.19	(Water Bottle - 30 oz.)
11	0.13	(Patch Kit/8 Patches)
10	0.12	(Mountain Tire Tube)
13	0.11	(Road Tire Tube)
14	0.09	(Sport-100 Helmet, Black)
0	0.09	(AWC Logo Cap)
16	0.09	(Sport-100 Helmet, Red)
15	0.09	(Sport-100 Helmet, Blue)
12	0.08	(Road Bottle Cage)
2	0.08	(Fender Set - Mountain)

In [541…

```python
# Bonus: Calculate confidence and lift among frequent item sets
# Generate rules; min threshold for lift to be 1
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
rules.sort_values('confidence', ascending = False, inplace = True)
rules.head(10)
```

Out[541…

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
1	(Road Bottle Cage)	(Water Bottle - 30 oz.)	0.08	0.19	0.07	0.89	4.72	0.06	7.16
3	(Mountain Bottle Cage)	(Water Bottle - 30 oz.)	0.07	0.19	0.06	0.84	4.50	0.05	5.22
7	(ML Mountain Tire)	(Mountain Tire Tube)	0.05	0.12	0.03	0.65	5.32	0.03	2.51
0	(Water Bottle - 30 oz.)	(Road Bottle Cage)	0.19	0.08	0.07	0.39	4.72	0.06	1.50
2	(Water Bottle - 30 oz.)	(Mountain Bottle Cage)	0.19	0.07	0.06	0.31	4.50	0.05	1.35
4	(Mountain Tire Tube)	(Patch Kit/8 Patches)	0.12	0.13	0.03	0.27	2.08	0.02	1.19
6	(Mountain Tire Tube)	(ML Mountain Tire)	0.12	0.05	0.03	0.26	5.32	0.03	1.28
5	(Patch Kit/8 Patches)	(Mountain Tire Tube)	0.13	0.12	0.03	0.25	2.08	0.02	1.18

In [542…

```python
# we can urge our churners to consider these items buy  the non_churner
rules['antecedents']
```

Out[542…

```
1          (Road Bottle Cage)
3      (Mountain Bottle Cage)
7          (ML Mountain Tire)
0     (Water Bottle - 30 oz.)
2     (Water Bottle - 30 oz.)
4        (Mountain Tire Tube)
6        (Mountain Tire Tube)
5       (Patch Kit/8 Patches)
Name: antecedents, dtype: object
```

1. python
    A. the list
    B. pandas
    C. red
2. mathplot lib
    A. google
3. keyboard
    A. like

- python
    - the
    - pandas
    - red
- mathplot lib

        ▪ google
- keyboard
    ▪ like

References

In [543…

```
#https://www.scikit-yb.org/en/latest/api/cluster/elbow.htm
#https://towardsdatascience.com/find-your-best-customers-with-customer-segmentation-in-pyt
# https://towardsdatascience.com/who-is-your-golden-goose-cohort-analysis-50c9de5dbd31
# https://medium.com/@sundarstyles89/weight-of-evidence-and-information-value-using-python
# https://medium.com/@sundarstyles89/variable-selection-using-python-vote-based-approach-f
# https://github.com/Sundar0989/XuniVerse/blob/master/Xverse.ipynb
# https://towardsdatascience.com/end-to-end-python-framework-for-predictive-modeling-b8052
# https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classificatio
# https://pbpython.com/market-basket-analysis.html
#https://towardsdatascience.com/handling-multi-collinearity-6579eb99fd81
#https://www.scikit-yb.org/en/latest/
#https://www.tutorialspoint.com/jupyter/jupyter_notebook_markdown_cells.htm
#https://medium.com/analytics-vidhya/feature-selection-by-using-voting-approach-e0d1c7182a
```

In [ ]: