Gallagher Case Study - Product Type Prediction

Case Study Abstract

Situation – This is a case study for Gallagher . At Gallagher, one of our biggest challenges is deciding which insurance products are right for our clients' needs. Using data on Gallagher's clients, the goal of the case study is to build a model which will predict the type of products that the company should sell to a client and deploy this model so that the applications and systems can use it. The objective through this case study is to explore and identify an optimized model that can predict the product type which can then be utilized by the internal business unit to make informed decision.

Task – In this case study, we're tasked to explore the client purchase history and client census datasets to build a model that can predict the product type. Hopefully, the outcome will offer actionable insights and enable the company to make informed decision on coverage/product selection for different customer segment with help of data analysis.

Action – There are two datasets (client_census_data.csv and client_purchase_history.csv) that contain client policy/coverage/premium history and the associated census data for each of those businesses, the census data contains business specific information like the number of employees, the annual payroll, the county where the business is being operated with the total population etc. Initially, we will conduct exploratory data analysis on the datasets – investigate the statistics of data, identify missing values on each data column, explore target variable distribution, correlation between predictor variables, identify columns that may not be necessary, and then preprocess data to convert categorical variables to numeric variables, remove columns that may be not be needed, impute missing values as needed. The preprocessed data will be scaled, trained using a baseline model and evaluated through K-Fold cross validation with basic parameter setting using all the features. We will conduct further experimentation by using different parameter settings, drop features of low importance and determine the optimized model with best accuracy score.

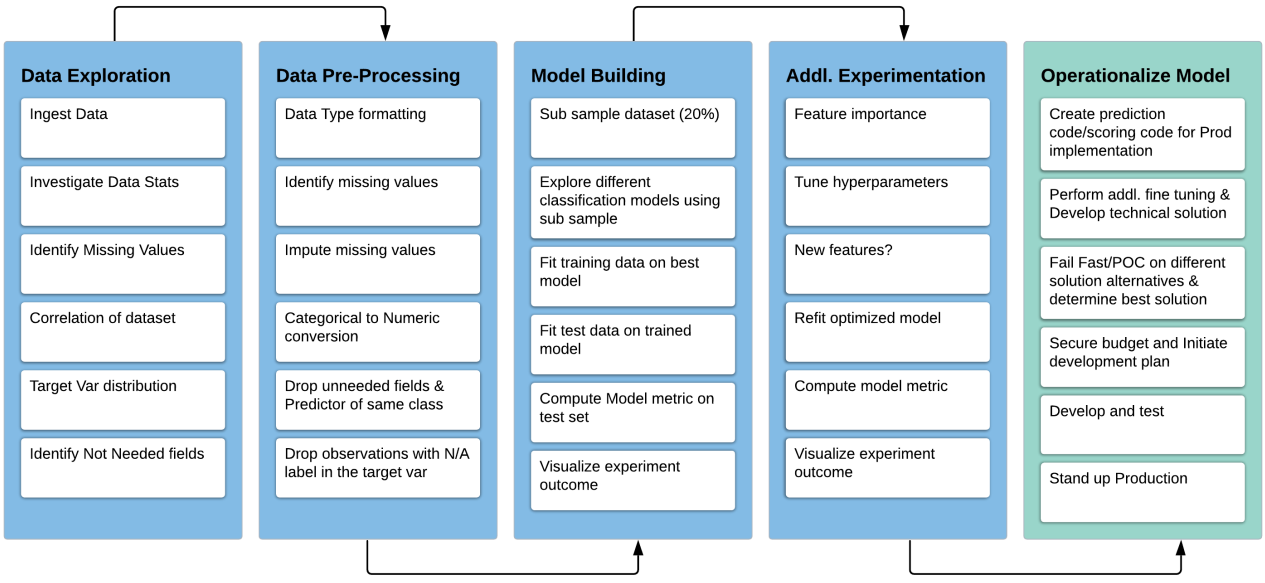
Result – The best model will be used to predict product type in the test dataset. Our scoring will be mainly based on accuracy and will try to evaluate using other metrics too. For each client in the test set, we will predict the product type.

Model Pipeline

In [3]:

```
from IPython.display import Image
Image(filename='/users/kousik/ajg_case_study/model_pipeline.png')
```

Out[3]:



Problem Setting

Our objective was to predict the product type for different customers which appears to be a classification problem and we will explore different classification models for such a setting to predict the labels using a set of features given the basic assumption that the examples are independent and identically distributed. The problem seems to have only two classes (binary classification) and it is important to compare the performance of different machine learning algorithms consistently.

The client purchase history dataset contains 1314922 records and the client census data dataset contains 514976 records. Considering the size of the datasets, we will plan on evaluating different models using a stratified sub sample (20%) of the data and based on the outcome, we will train a model with the full dataset. We will perform cleaning and pre-processing of the data before training the model and the initial training/evaluation would be done using basic parameter setting and upon observing the outcome of this initial model, we will probably conduct further experimentation to tune the hyper-parameters if the required accuracy (> 80%) is not achieved through any of the models and if needed engineer additional features from the provided input data to improve the accuracy of the final model.

Metrics

The models will be basically scored on the accuracy. We will explore the following metrics on our trained model and calculate these metrics on the test set.

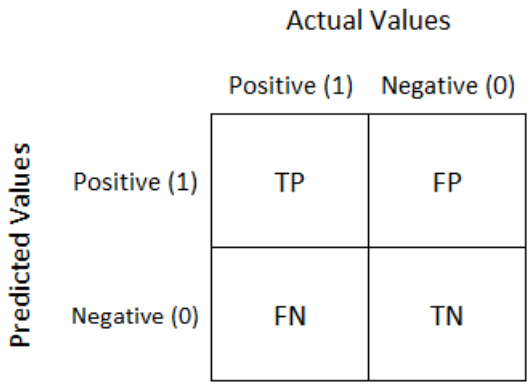
Confusion Matrix

The confusion matrix is used to describe the performance of a classification model on a set of test data for which true values are known.

In [4]:

```
from IPython.display import Image
Image(filename='/users/kousik/ajg_case_study/cfm.png')
```

Out[4]:



From the confusion matrix the following information can be extracted :

True Positive (TP) : This shows that a model correctly predicted Positive cases as Positive.

False Positive (FP) : This shows that a model incorrectly predicted Negative cases as Positive.

False Negative (FN) : This shows that an incorrectly model predicted Positive cases as Negative.

True Negative (TN) : This shows that a model correctly predicted Negative cases as Positive.

F1-Score

This comes from the confusion matrix. Based on the above confusion matrix above, we can calculate the precision and the recall scores.

Precision score: this is the measure of the accuracy, provided that a class label has been predicted. Simply put, it answers the following question, of all the classes, how many were correctly predicted? The answer to this question should be as high as possible.

It can be calculated as follows:

In [11]:

```
from IPython.display import Image
Image(filename='/users/kousik/ajg_case_study/prec.png')
```

Out[11]:

$$Precision = \frac{TP}{TP + FP}$$

Recall score(Sensitivity): This is the true positive rate that is if it predicts positive then how often does this take place

In [12]:

```
from IPython.display import Image
Image(filename='/users/kousik/ajg_case_study/rec.png')
```

Out[12]:

$$TP = \frac{TP}{TP + FN}$$

The F1 score is calculated based on the precision and recall of each class. It is the weighted average of the Precision and the recall scores. The F1 score reaches its perfect value at one and worst at 0.It is a very good way to show that a classifies has a good recall and precision values.

We can calculate it using this formula:

In [13]:

```
from IPython.display import Image
Image(filename='/users/kousik/ajg_case_study/f1.png')
```

Out[13]:

$$F1Score = 2(\frac{Precision \times Recal}{Precision + Recal})$$

Accuracy

Accuracy = (TP + TN)/Total Observations

Data Load

In [138]:

```
import os
import time
import gc
import warnings
warnings.filterwarnings("ignore")
# data manipulation
import numpy as np
import pandas as pd
# plot
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()
# model
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Imputer
from sklearn.metrics import confusion_matrix
import time
from sklearn.metrics import accuracy_score
```

In [2]:

```
%%time
client_purchase_df = pd.read_csv('/users/kousik/ajg_case_study/client_purchase_history.csv')
client_census_df = pd.read_csv('/users/kousik/ajg_case_study/client_census_data.csv')
gc.collect()
```

```
CPU times: user 9.27 s, sys: 643 ms, total: 9.92 s
Wall time: 9.09 s
```

Merge dataset

In [3]:

```
client_df = pd.merge(client_purchase_df, client_census_df, on="Client_Key")
print("Shape of client purchase history dataset: ",client_purchase_df.shape)
print("Shape of client census dataset: ",client_census_df.shape)
print("Shape of final merged dataset: ",client_df.shape)
```

```
Shape of client purchase history dataset:  (1314922, 36)
Shape of client census dataset:  (514976, 8)
Shape of final merged dataset:  (1314922, 43)
```

In [4]:

```
client_df.head()
```

Out[4]:

	Client_Key	Global_Market_Tier	DUNS_Number	Client_Offices	Placement_Market_Group_Key	Placement_Market_Company_Key	Plac
0	36680389	NaN	NaN	NaN	5971484.0	5127032	
1	36680389	Tier 1	95582010.0	NaN	7245911.0	6221237	
2	36680389	Tier 1	NaN	NaN	8280758.0	7109742	
3	78136034	Tier 2	107329587.0	3.0	35395670.0	30390222	
4	78136034	Tier 3	NaN	2.0	30534869.0	26216807	

5 rows × 43 columns

Data Exploration

Preliminary Data Statistics

In [5]:

```
client_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1314922 entries, 0 to 1314921
```

```
Data columns (total 43 columns):
Client_Key                        1314922 non-null int64
Global_Market_Tier                1312399 non-null object
DUNS_Number                       707971 non-null float64
Client_Offices                    700796 non-null float64
Placement_Market_Group_Key        1314919 non-null float64
Placement_Market_Company_Key      1314922 non-null int64
Placement_Market_Type             1314922 non-null object
Placement_Market_Focus            1314922 non-null object
Intermediary_Market_Type          1314922 non-null object
Intermediary_Detail               1314922 non-null object
SIC_Level_2                       1314922 non-null object
SIC_Level_2_Name                  1314922 non-null object
SIC_Level_8                       1314922 non-null int64
SIC_Level_8_Name                  1314922 non-null object
IndustryNicheConventionDesc       1314922 non-null object
NAICSLevel1Code                   1314922 non-null int64
NAICSLevel1Name                   1314922 non-null object
Effective_Year                    1314922 non-null int64
Management_Niche                  1314922 non-null object
Client_Premium_Segmentation       1314922 non-null object
Client_Revenue_Segmentation       1314922 non-null object
Satellite                         1314922 non-null object
Region                            1314922 non-null object
Product_Coverage                  1314922 non-null object
Product_Line                      1314922 non-null object
Product_Group                     1287051 non-null object
Source_Product_Line               1314922 non-null int64
PolicyID                          1314922 non-null object
Bill_Type                         1314922 non-null object
Source_System_Name                1314922 non-null object
contact_count                     1314922 non-null int64
Premium                           1314922 non-null float64
Brokerage_Expense                 1314922 non-null float64
Commission_Revenue                1314922 non-null float64
Fee_Revenue                       296522 non-null float64
dcf_revenue                       1313600 non-null float64
zip_code                          1314922 non-null int64
cnty_pop_2010                     1311765 non-null float64
RUCC_2013                         1311765 non-null float64
count_estabs_in_cnty              1314922 non-null int64
num_employees_fill                1313153 non-null float64
annual_payroll_fill               1313404 non-null float64
census_match_ind                  1314922 non-null int64
dtypes: float64(12), int64(10), object(21)
memory usage: 441.4+ MB
```

In [6]:

```
client_df.describe()
```

Out[6]:

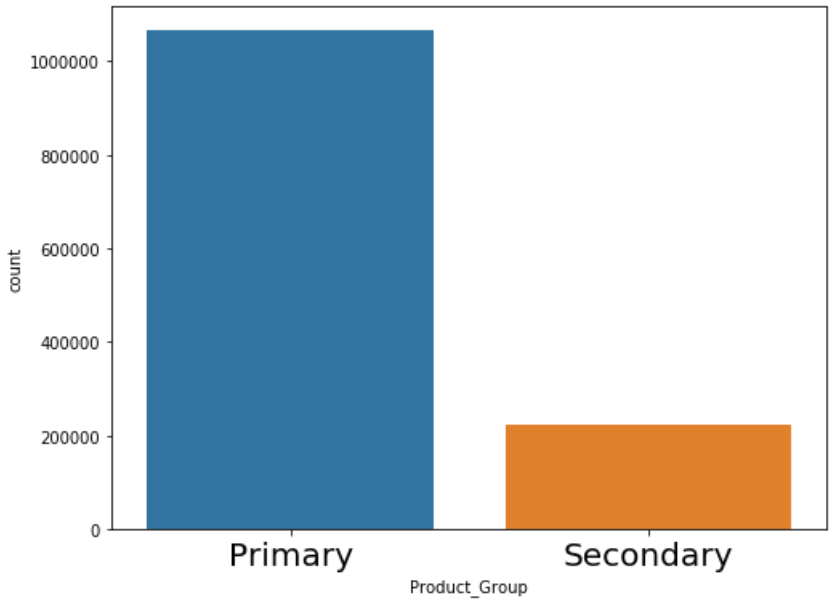
	Client_Key	DUNS_Number	Client_Offices	Placement_Market_Group_Key	Placement_Market_Company_Key	SIC_Level_8	NA
count	1.314922e+06	7.079710e+05	7.007960e+05	1.314919e+06	1.314922e+06	1.314922e+06	
mean	5.499920e+07	3.108191e+08	7.692595e+02	2.123115e+07	1.822873e+07	3.280657e+07	
std	2.597431e+07	3.959115e+08	7.626904e+03	1.237944e+07	1.062883e+07	3.599015e+07	
min	1.000002e+07	1.200367e+06	0.000000e+00	1.010000e+02	8.700000e+01	-1.000000e+00	
25%	3.250123e+07	4.947454e+07	3.000000e+00	1.036223e+07	8.896782e+06	0.000000e+00	
50%	5.501202e+07	9.603736e+07	1.200000e+01	2.146092e+07	1.842596e+07	1.542010e+07	
75%	7.747333e+07	6.682975e+08	9.500000e+01	3.198339e+07	2.746044e+07	7.311000e+07	
max	9.999989e+07	1.182770e+09	2.300000e+06	4.236923e+07	3.637762e+07	9.999900e+07	

8 rows × 22 columns

Target Variable (Class) Frequency

In [7]:

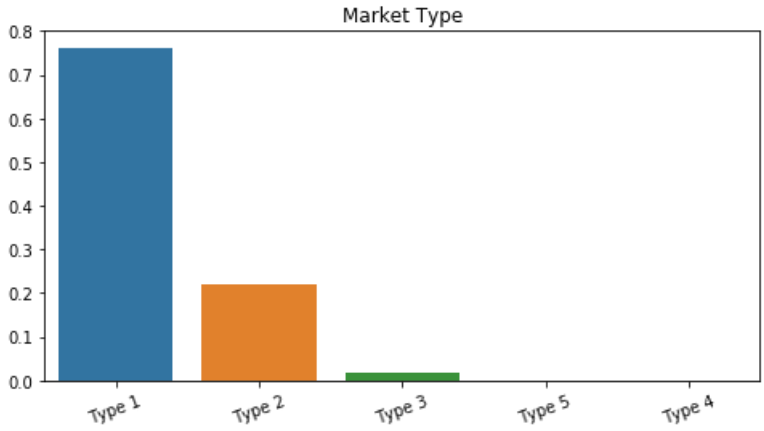
```
plt.figure(figsize=(8,6))
sns.countplot(client_df['Product_Group'])
plt.xticks(fontsize=20)
plt.show()
```



Market Type Distribution

In [8]:

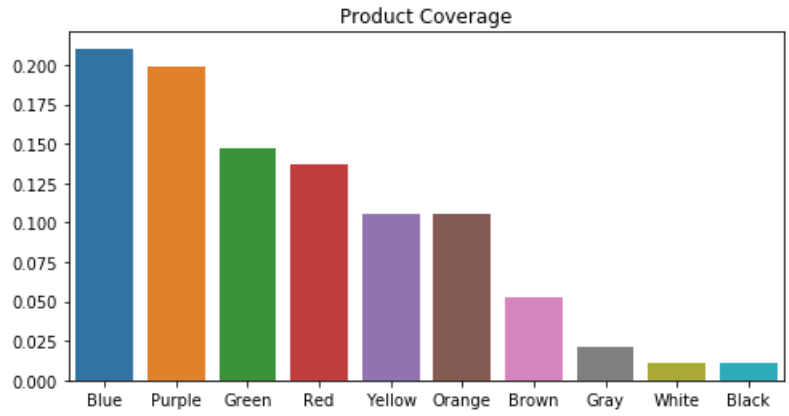
```
def plot_categorical(data, col, size=[8 ,4], xlabel_angle=0, title='', max_cat = None):
    '''use this for ploting the count of categorical features'''
    plotdata = data[col].value_counts() / len(data)
    if max_cat != None:
        plotdata = plotdata[max_cat[0]:max_cat[1]]
    plt.figure(figsize = size)
    sns.barplot(x = plotdata.index, y=plotdata.values)
    plt.title(title)
    if xlabel_angle!=0:
        plt.xticks(rotation=xlabel_angle)
    plt.show()
plot_categorical(data=client_df, col='Placement_Market_Type', size=[8 ,4], xlabel_angle=20, title='
Market Type', max_cat=[0, 6])
```



Product Coverage Distribution

In [9]:

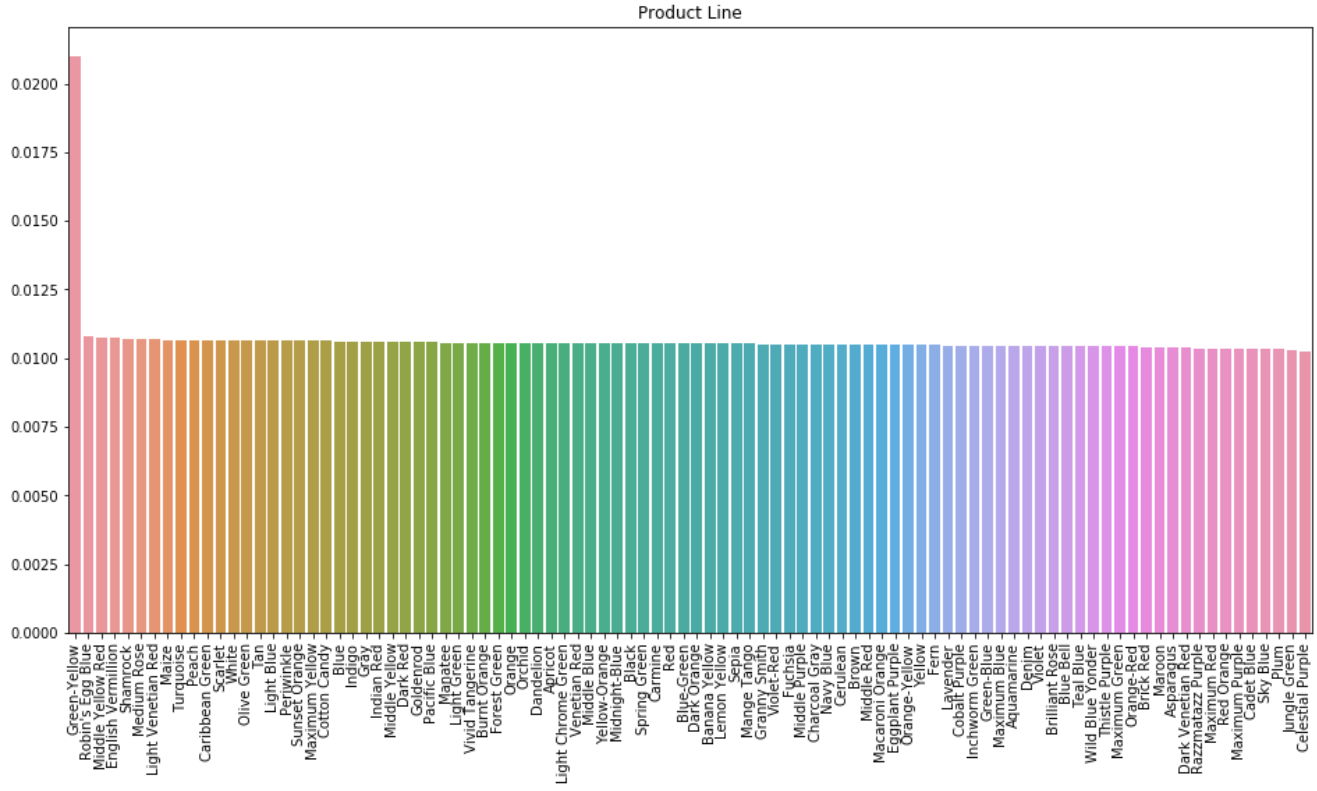
```
plot_categorical(data=client_df, col='Product_Coverage', size=[8 ,4], xlabel_angle=0, title='Produc
t Coverage')
```



Product Line Distribution

In [10]:

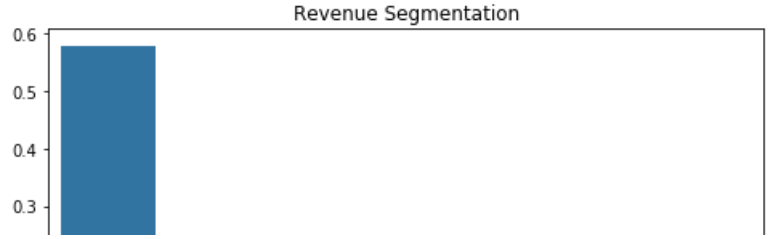
```
plot_categorical(data=client_df, col='Product_Line', size=[16 ,8], xlabel_angle=90, title='Product Line')
```

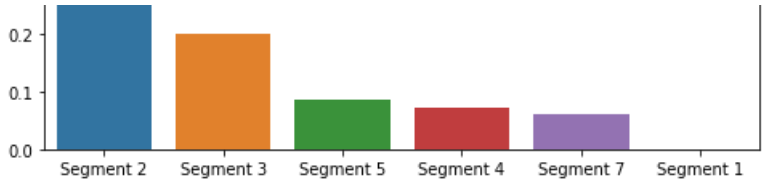


Revenue Segmentation Distribution

In [11]:

```
plot_categorical(data=client_df, col='Client_Revenue_Segmentation', size=[8 ,4], xlabel_angle=0, title='Revenue Segmentation')
```

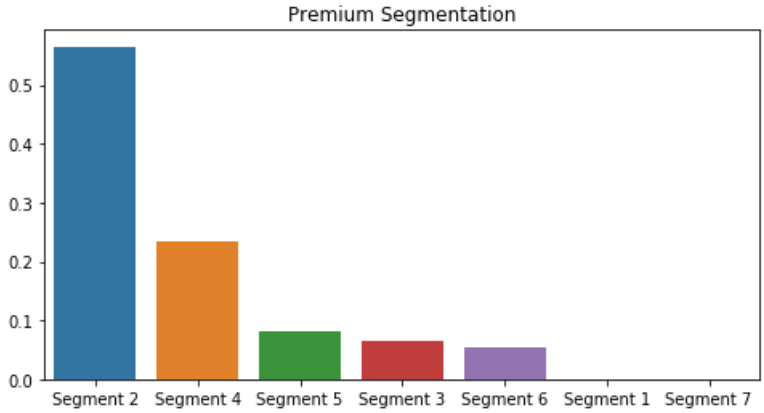




Premium Segmentation Distribution

In [12]:

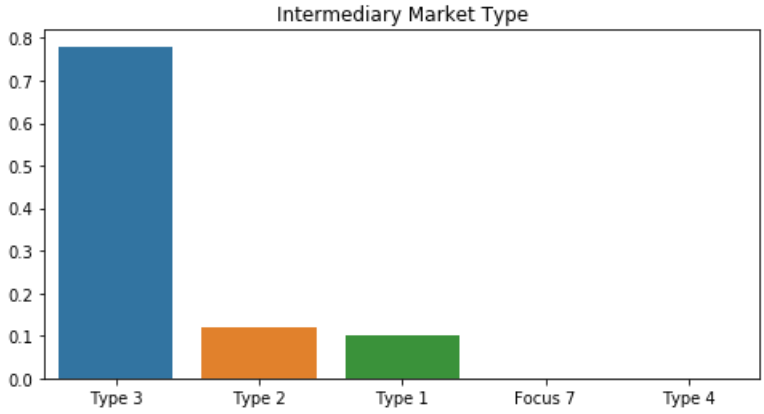
```
plot_categorical(data=client_df, col='Client_Premium_Segmentation', size=[8 ,4], xlabel_angle=0, ti
tle='Premium Segmentation')
```



Intermediary Market Type Dist

In [13]:

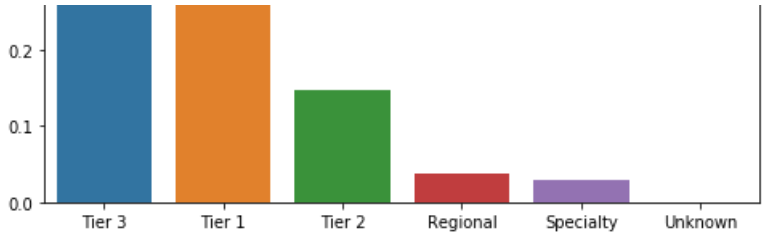
```
plot_categorical(data=client_df, col='Intermediary_Market_Type', size=[8 ,4], xlabel_angle=0, title
='Intermediary Market Type')
```



Global Market Tier Dist

In [14]:

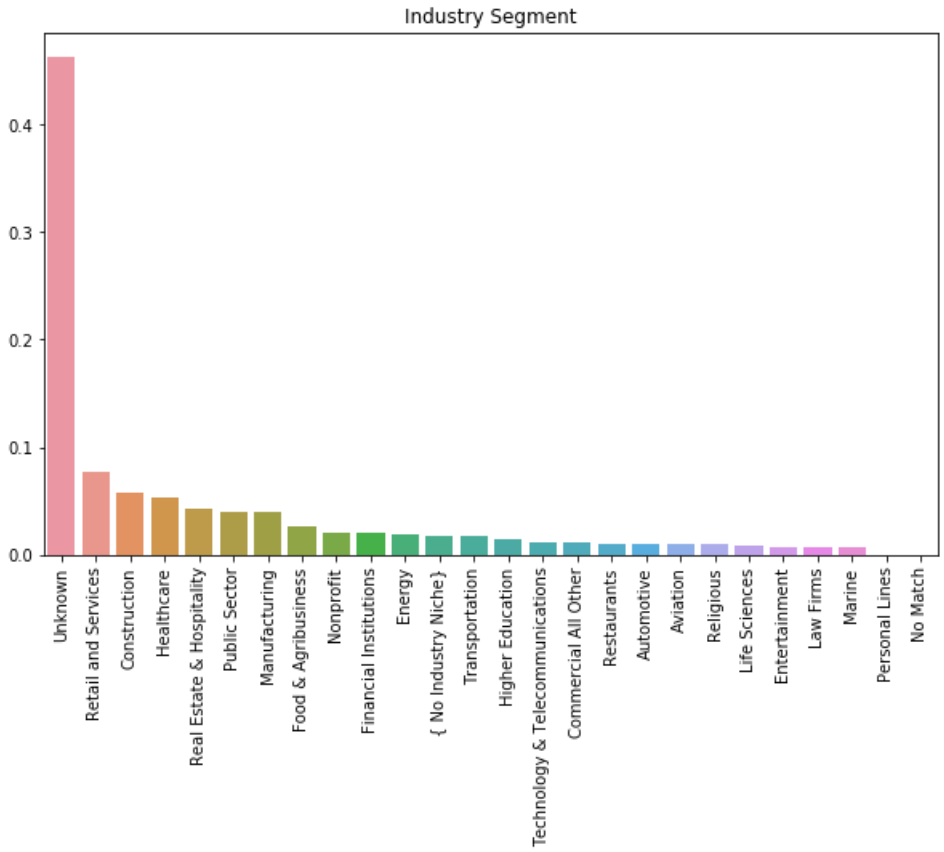
```
plot_categorical(data=client_df, col='Global_Market_Tier', size=[8 ,4], xlabel_angle=0, title='Glob
al Market Tier')
```



Industry Segment Distribution

In [15]:

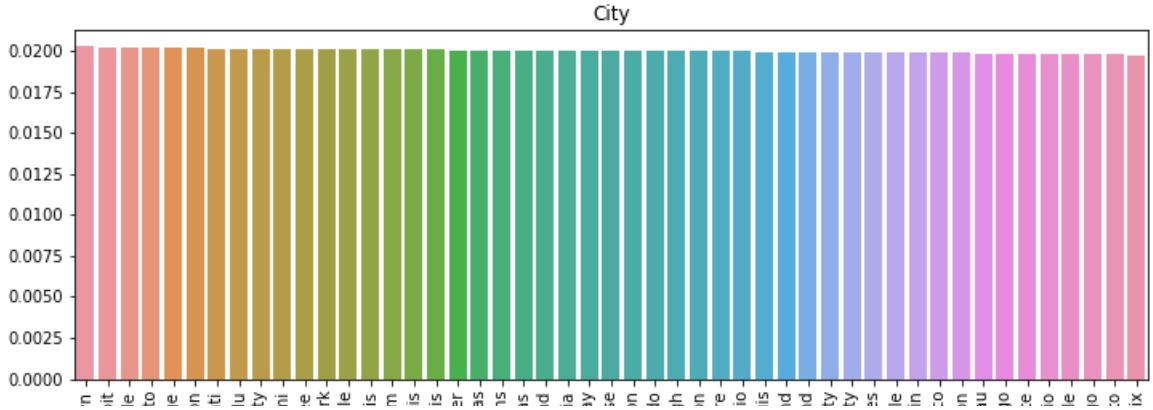
```
plot_categorical(data=client_df, col='Management_Niche', size=[10 ,6], xlabel_angle=90, title='Indu
stry Segment')
```



City Distribution

In [42]:

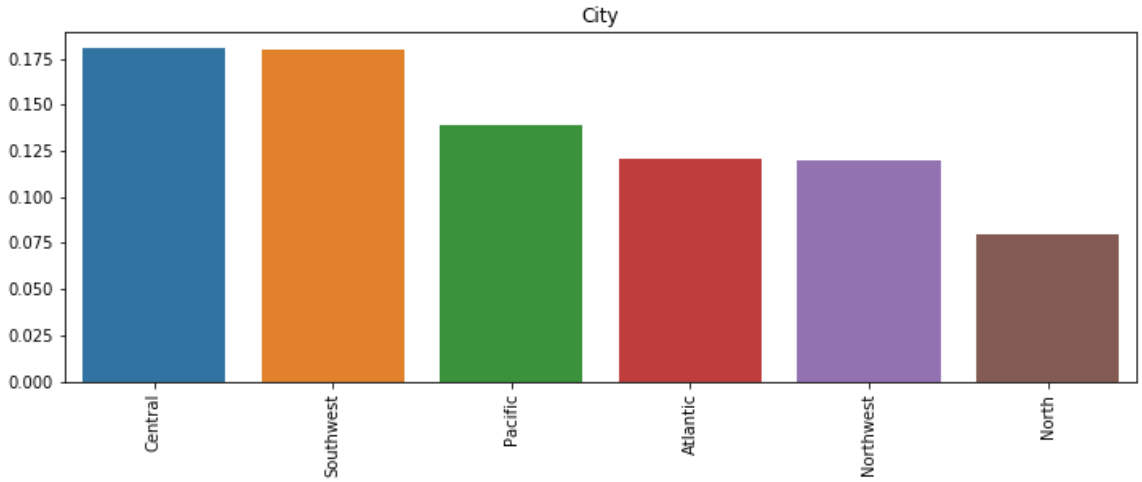
```
plot_categorical(data=client_df, col='Satellite', size=[12 ,4], xlabel_angle=90,
                 title='City', max_cat = [1, 1000])
```



Region Distribution

In [43]:

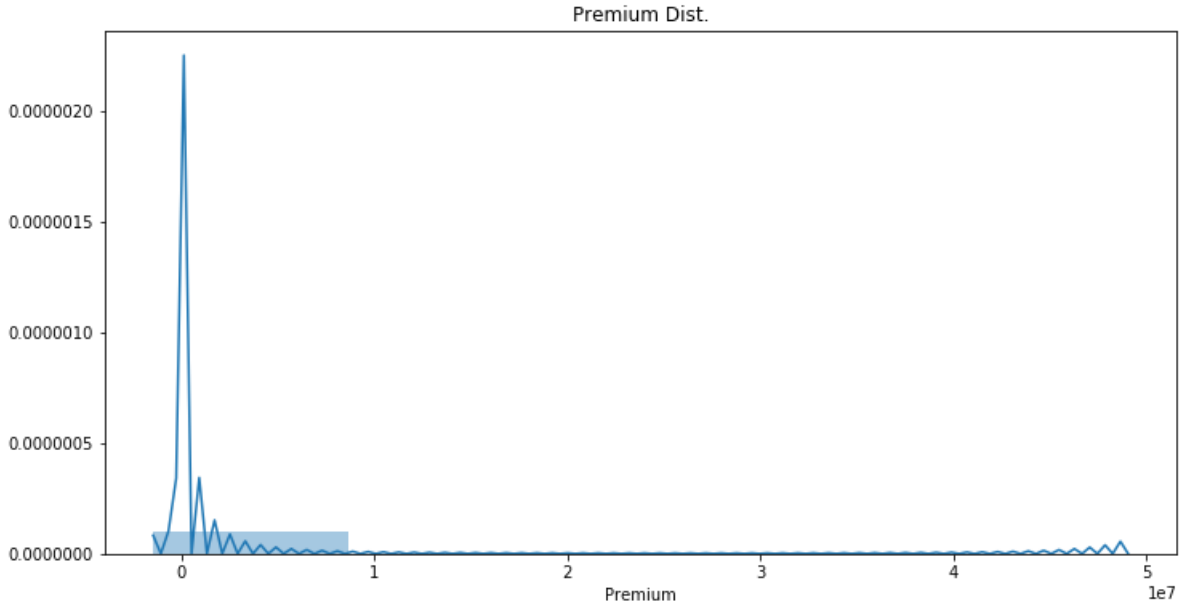
```
plot_categorical(data=client_df, col='Region', size=[12 ,4], xlabel_angle=90,
                 title='City', max_cat = [1, 1000])
```



Premium, Payroll and Employee Count Dist.

In [146]:

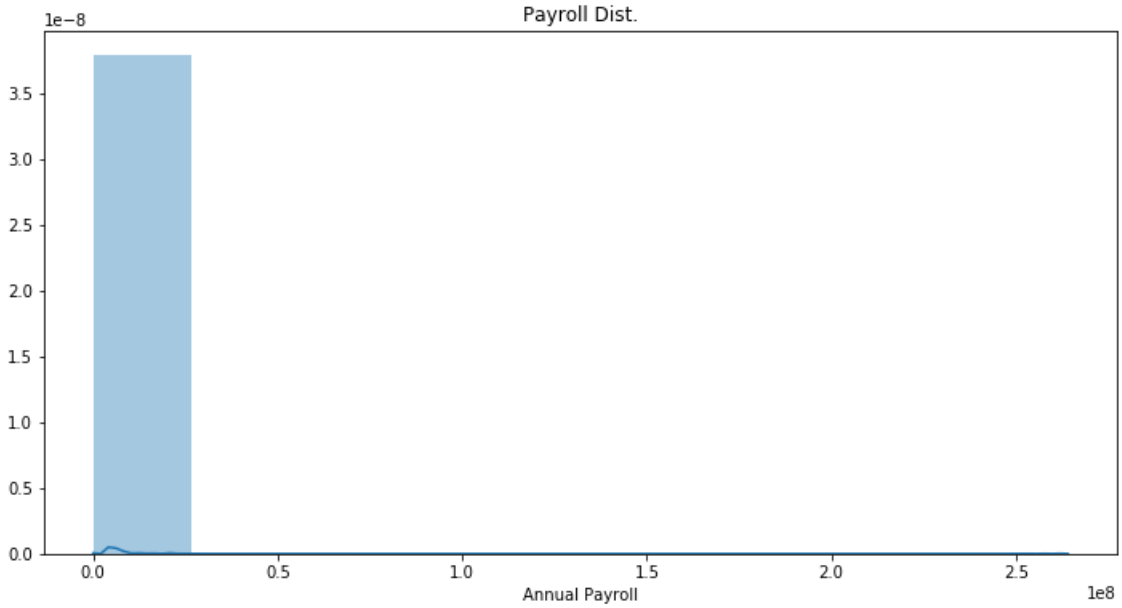
```
plt.figure(figsize=[12, 6])
sns.distplot(client_df['Premium'].fillna(0).astype('int'),kde=True,bins=5)
plt.xlabel('Premium')
plt.title('Premium Dist.')
plt.show()
```



In [64]:

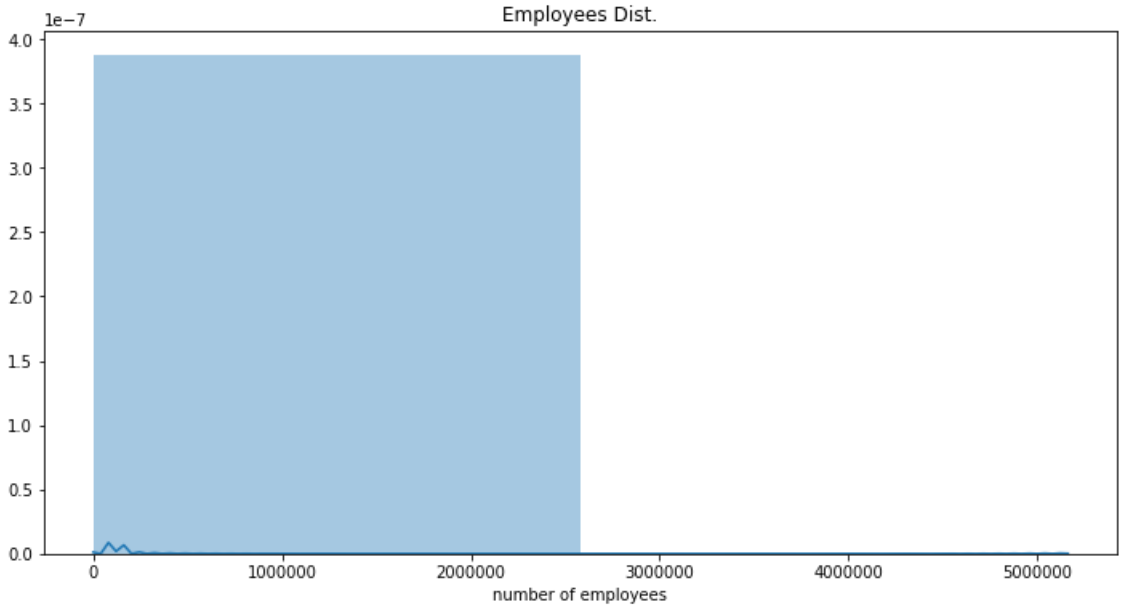
```
plt.figure(figsize=[12, 6])
sns.distplot(client_df['annual_payroll_fill'].fillna(0).astype('int'),kde=True,bins=10)
plt.xlabel('Annual Payroll')
plt.title('Payroll Dist.')
```

```
plt.show()
```



In [101]:

```
plt.figure(figsize=[12, 6])
sns.distplot(client_df['num_employees_fill'].fillna(0).astype('int'),bins=2)
plt.xlabel('number of employees')
plt.title('Employees Dist.')
plt.show()
```

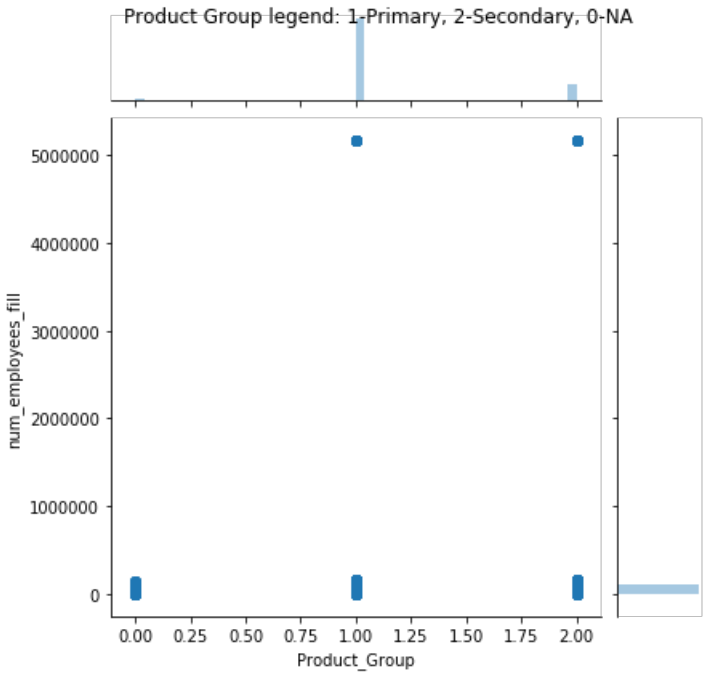


Bivariate Distribution - Employee Count vs Product Group

In [148]:

```
plt.figure(figsize=[20, 10])
df_1 = pd.DataFrame(client_df, columns=["Product_Group", "num_employees_fill"])
df_1['Product_Group'] = df_1['Product_Group'].map({'Primary':1, 'Secondary':2})
df_1.fillna(0, inplace=True)
sns.jointplot(x="Product_Group", y="num_employees_fill", data=df_1);
plt.suptitle('Product Group legend: 1-Primary, 2-Secondary, 0-NA')
plt.show()
```

```
<Figure size 1440x720 with 0 Axes>
```

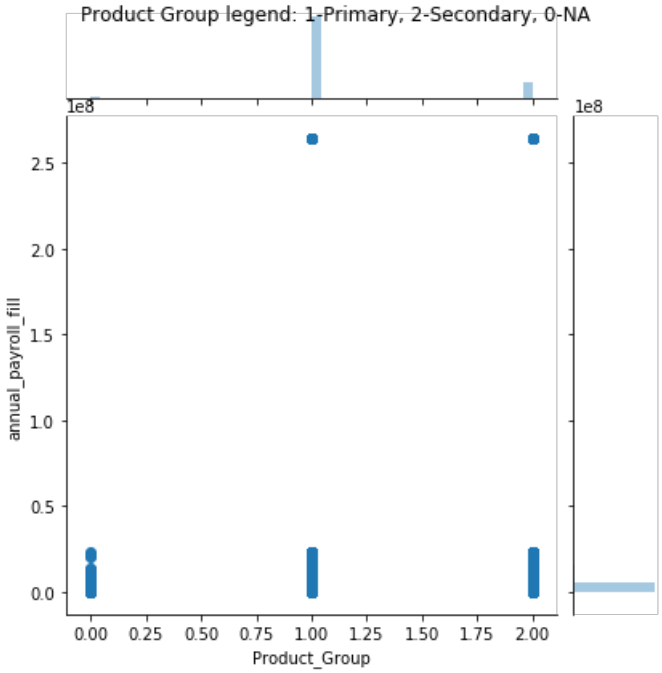


Bivariate Distribution - Annual Payroll vs Product Group

In [149]:

```
plt.figure(figsize=[20, 10])
df_1 = pd.DataFrame(client_df, columns=["Product_Group", "annual_payroll_fill"])
df_1['Product_Group'] = df_1['Product_Group'].map({'Primary':1, 'Secondary':2})
df_1.fillna(0, inplace=True)
sns.jointplot(x="Product_Group", y="annual_payroll_fill", data=df_1);
plt.suptitle('Product Group legend: 1-Primary, 2-Secondary, 0-NA')
plt.show()
```

```
<Figure size 1440x720 with 0 Axes>
```



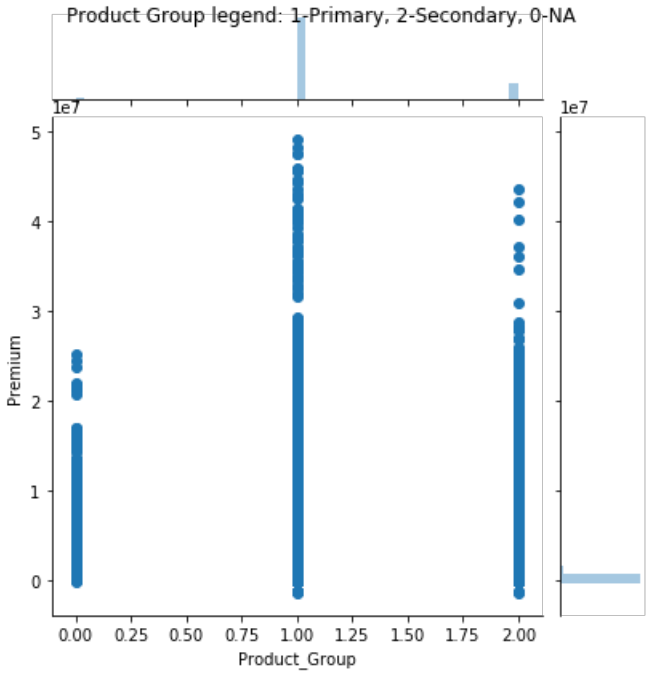
Bivariate Distribution - Premium vs Product Group

In [152]:

```
plt.figure(figsize=[20, 10])
```

```
df_1 = pd.DataFrame(client_df, columns=["Product_Group", "Premium"])
df_1['Product_Group'] = df_1['Product_Group'].map({'Primary':1, 'Secondary':2})
df_1.fillna(0, inplace=True)
sns.jointplot(x="Product_Group", y="Premium", data=df_1);
plt.suptitle('Product Group legend: 1-Primary, 2-Secondary, 0-NA')
plt.show()
```

```
<Figure size 1440x720 with 0 Axes>
```



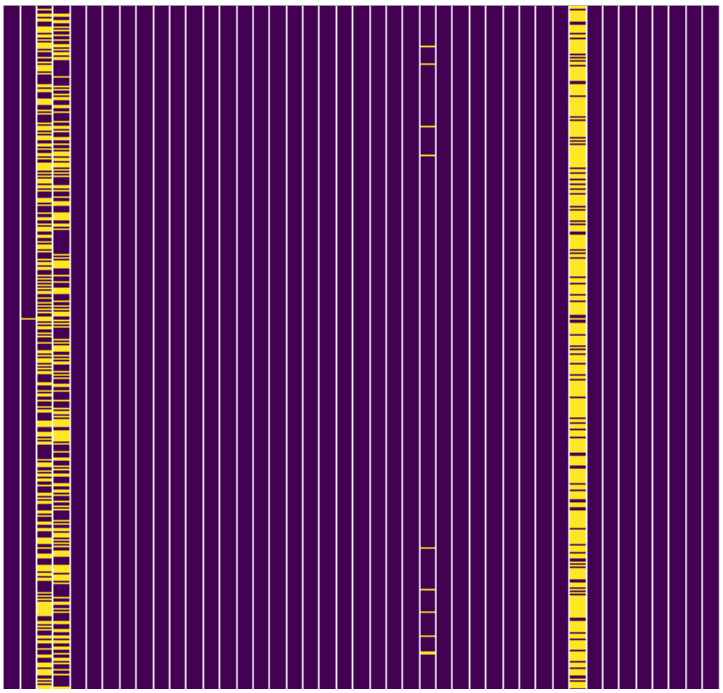
Null values Heatmap

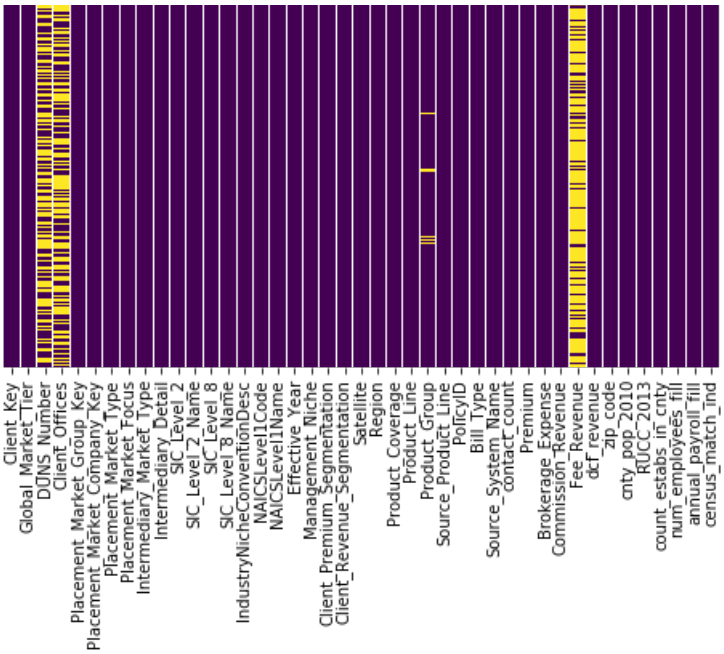
In [108]:

```
plt.figure(figsize=(8, 12))
#df_2 = pd.DataFrame(client_df, columns=["Product_Group", "num_employees_fill"])
sns.heatmap(client_df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[108]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a199f9e10>
```





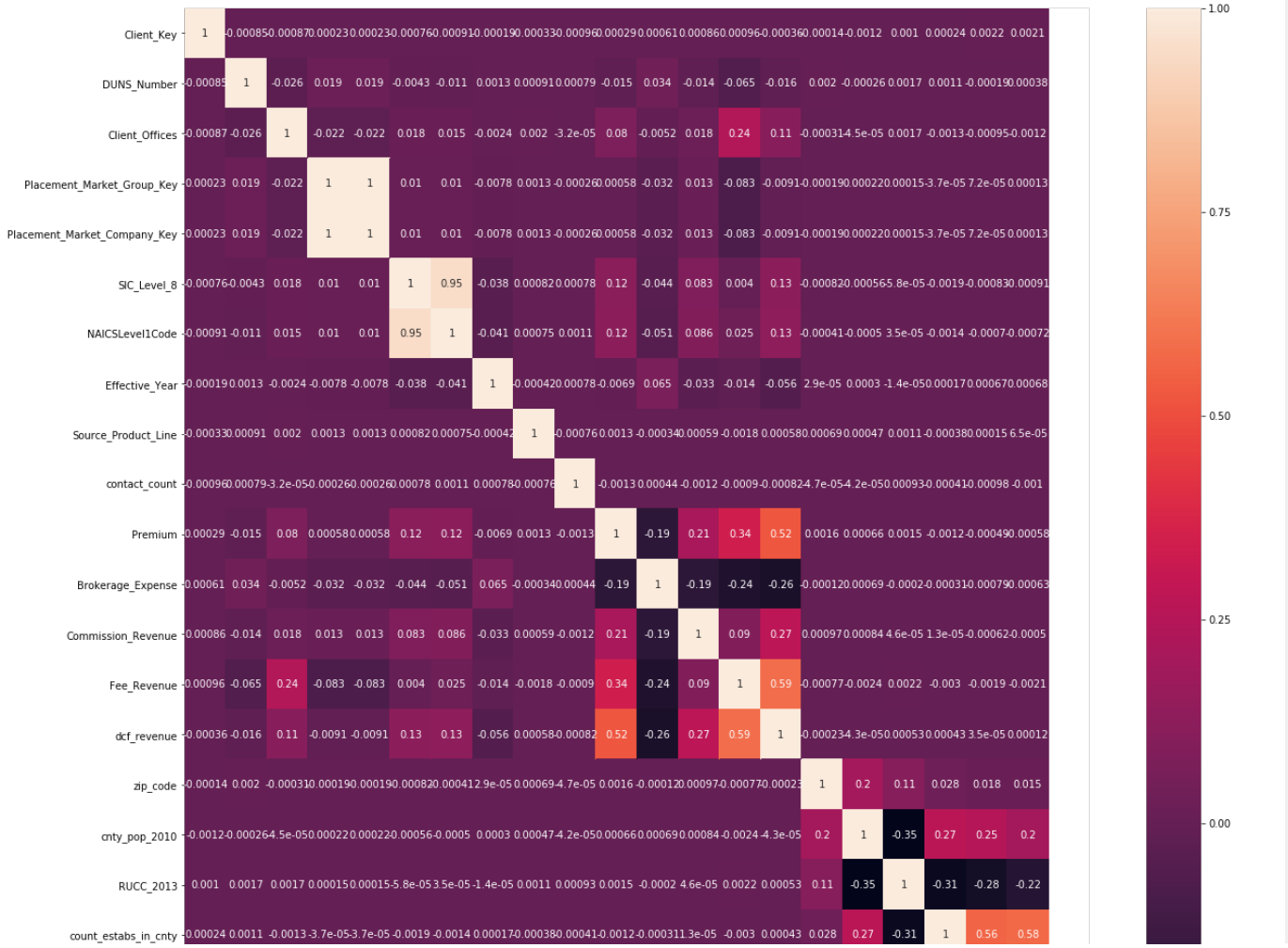
Predictor Variables Correlation Heat Map

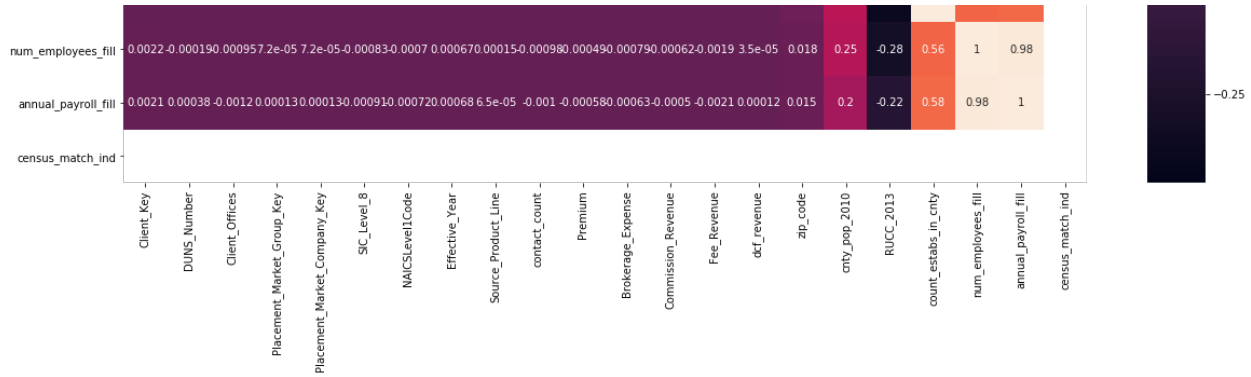
In [13]:

```
#Built this heat map after data pre-processing but keeping here as it more relevant to EDA#
plt.figure(figsize=(20,20))
sns.heatmap(client_df.corr(),annot=True)
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1078d4f28>
```



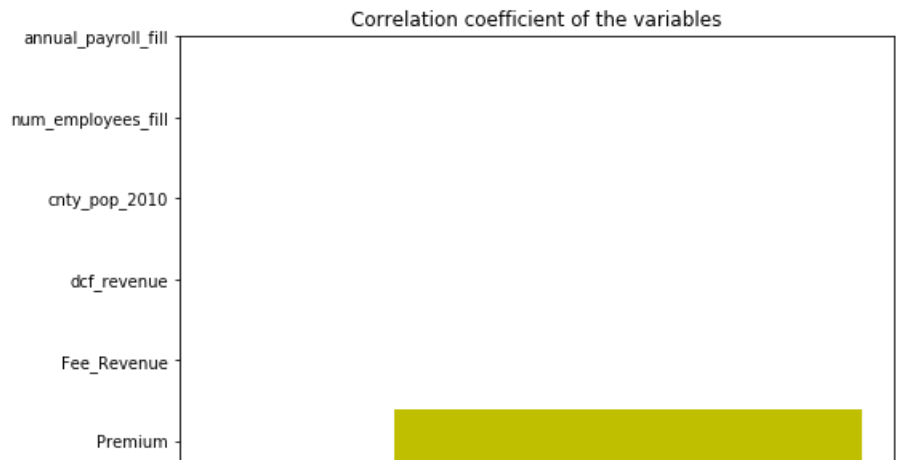


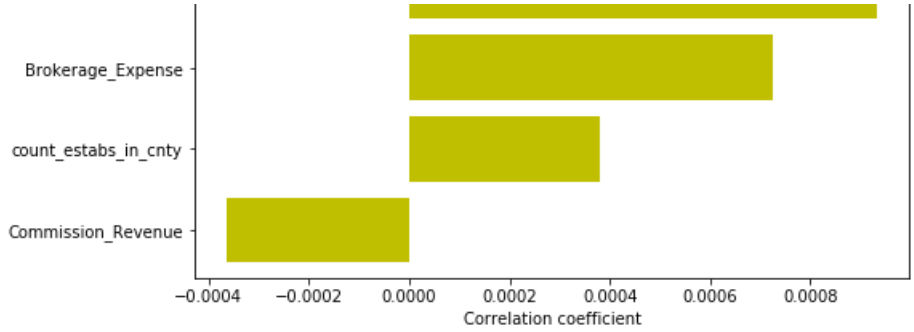
Correlation of Numeric Attributes in Dataset

In [139]:

```
num_cols = ['Premium','Brokerage_Expense','Commission_Revenue','Fee_Revenue',
            'dcf_revenue','cnty_pop_2010','count_estabs_in_cnty','num_employees_fill','annual_payro
ll_fill']
client_df_new = client_df.copy()
client_df_new = client_df_new[pd.notnull(client_df_new['Product_Group'])]
client_df_new['Product_Group_Int'] = (client_df_new['Product_Group']=='Primary').astype(int)
#client_df_new.fillna(0,inplace=True)
client_df_new_x = client_df_new[num_cols]
client_df_new_y = client_df_new['Product_Group_Int']

#print(train_df_new.head())

#Now let us look at the correlation coefficient of each of these variables #
x_cols = [col for col in client_df_new_x.columns if col not in ['Product_Group']]

print(x_cols)
labels = []
values = []
for col in x_cols:
    labels.append(col)
    values.append(np.corrcoef(client_df_new_x[col].values, client_df_new_y.values)[0,1])
corr_df = pd.DataFrame({'col_labels':labels, 'corr_values':values})
corr_df = corr_df.sort_values(by='corr_values')

ind = np.arange(len(labels))
width = 0.9
fig, ax = plt.subplots(figsize=(8,8))
rects = ax.barh(ind, np.array(corr_df.corr_values.values), color='y')
ax.set_yticks(ind)
ax.set_yticklabels(corr_df.col_labels.values, rotation='horizontal')
ax.set_xlabel("Correlation coefficient")
ax.set_title("Correlation coefficient of the variables")
#autolabel(rects)
plt.show()
```

```
['Premium', 'Brokerage_Expense', 'Commission_Revenue', 'Fee_Revenue', 'dcf_revenue',
'cnty_pop_2010', 'count_estabs_in_cnty', 'num_employees_fill', 'annual_payroll_fill']
```





Check Missing values

In [142]:

```
def find_missing(data):
    # number of missing values
    count_missing = data.isnull().sum().values
    # total records
    total = data.shape[0]
    # percentage of missing
    ratio_missing = count_missing/total
    # return a dataframe to show: feature name, # of missing and % of missing
    return pd.DataFrame(data={'missing_count':count_missing, 'missing_ratio':ratio_missing}, index=
data.columns.values)
data_missing = find_missing(client_df)

data_missing.reset_index()[['index', 'missing_ratio']]\
    .rename(columns={'index':'columns', 'missing_ratio':'data_missing_ratio'})\
    .sort_values(['data_missing_ratio'], ascending=False)\
    .query('data_missing_ratio>0')
```

Out[142]:

	columns	data_missing_ratio
34	Fee_Revenue	0.774495
3	Client_Offices	0.467044
2	DUNS_Number	0.461587
25	Product_Group	0.021196
37	cnty_pop_2010	0.002401
38	RUCC_2013	0.002401
1	Global_Market_Tier	0.001919
40	num_employees_fill	0.001345
41	annual_payroll_fill	0.001154
35	dcf_revenue	0.001005
4	Placement_Market_Group_Key	0.000002

Outlier Detection

In [315]:

```
client_df.sort_values(['annual_payroll_fill'], ascending=False).head(50)
```

Out[315]:

	Client_Key	Global_Market_Tier	DUNS_Number	Client_Offices	Placement_Market_Group_Key	Placement_Market_Company_Key
496302	65652876	Tier 1	NaN	NaN	33092732.0	28412952
496303	65652876	Tier 2	NaN	4.0	39754739.0	34132857
699326	52973330	Tier 2	9.360749e+07	1500.0	8250761.0	7083987

	Client_Key	Global_Market_Tier	DUNS_Number	Client_Offices	Placement_Market_Group_Key	Placement_Market_Company_Key
1243297	64513067	Tier 1	9.913370e+06	8.0	37026004.0	31790002
1243296	64513067	Tier 3	5.253686e+07	65.0	30740690.0	26393522
699325	52973330	Tier 3	NaN	NaN	7006430.0	6015622
545611	23747449	Tier 1	4.311392e+07	28.0	7973561.0	6845987
614081	29378880	Tier 1	2.050448e+07	99.0	16424795.0	14102097
1287751	33135167	Tier 3	9.612076e+08	NaN	37344980.0	32063872
941524	59711595	Tier 3	NaN	NaN	21409940.0	18382272
545610	23747449	Tier 1	9.918551e+08	60.0	6697748.0	5750592
699772	98572486	Tier 3	2.540179e+07	NaN	37603865.0	32286147
1281470	93558673	Tier 3	7.253932e+08	NaN	16467860.0	14139072
614082	29378880	Tier 2	6.680399e+08	NaN	23416076.0	20104712
614083	29378880	Tier 3	NaN	94.0	19969589.0	17145607
1281469	93558673	Tier 1	NaN	NaN	19326485.0	16593447
941523	59711595	Tier 3	4.176744e+07	NaN	14988305.0	12868747
941522	59711595	Tier 2	NaN	NaN	18110072.0	15549052
818502	47026921	Tier 3	NaN	NaN	40298645.0	34599847
1281468	93558673	Tier 2	9.516627e+07	550.0	13634084.0	11706032
699327	52973330	Tier 1	NaN	NaN	9719030.0	8344622
627367	99686963	Tier 2	NaN	0.0	32416265.0	27832147
603439	97124738	Tier 3	NaN	60.0	18906131.0	16232537
614641	80723914	Tier 3	5.011148e+07	0.0	37294193.0	32020267
1045039	39491235	Tier 2	1.159571e+09	NaN	11702198.0	10047342
1045038	39491235	Tier 1	9.776031e+07	3.0	10082162.0	8656402
568234	59282721	Tier 2	NaN	NaN	8906141.0	7646687
568233	59282721	Tier 3	NaN	NaN	10899605.0	9358247
603440	97124738	Tier 3	8.560603e+07	NaN	13343222.0	11456302
627366	99686963	Tier 3	NaN	NaN	39022832.0	33504452
603438	97124738	Specialty	NaN	7.0	16170563.0	13883817
496277	92894866	Tier 2	NaN	NaN	20185112.0	17330652
614642	80723914	Tier 3	1.625344e+08	NaN	30986012.0	26604152
568232	59282721	Tier 3	2.079221e+08	434.0	12614087.0	10830277
927761	86287587	Tier 1	NaN	NaN	38929574.0	33424382
496278	92894866	Tier 3	7.429625e+07	NaN	24502700.0	21037672
496279	92894866	Tier 3	8.659114e+07	NaN	28341029.0	24333207
545612	23747449	Tier 1	NaN	NaN	9320852.0	8002752
1045040	39491235	Tier 1	1.075316e+07	20.0	8261552.0	7093252
1308104	29819436	Tier 1	7.864887e+07	85.0	3194237.0	2742527
1031968	45608770	Tier 1	2.161281e+08	6.0	11143937.0	9568027
373518	75453914	Tier 3	1.493485e+08	1315.0	35413985.0	30405947
1155183	67026967	Tier 1	NaN	2.0	35322113.0	30327067
373517	75453914	Tier 1	3.137749e+07	5.0	42302999.0	36320757
1308105	29819436	Tier 3	NaN	NaN	3789524.0	3253632
346807	86371562	Tier 3	9.441375e+08	NaN	34576841.0	29687187
993215	16350518	Tier 3	NaN	NaN	27247376.0	23394212
1031970	45608770	Tier 1	1.155173e+09	NaN	15648932.0	13435952
1156276	78654591	Tier 2	NaN	1.0	947036.0	813112
1156277	78654591	Tier 3	1.033308e+09	3.0	1324820.0	1137472

50 rows × 43 columns

30 rows × 40 columns

Observations from EDA

1. Following are some of the important data metrics extracted from the dataset that are interesting to me.
    a. Total count in client purchase history - 1314922
    b. Total count in client census dataset   - 514976
    c. Total count of missing Product_Group   - 27871
    d. Total count of missing Industry Segment - 607814
    e. Total count of missing Fee Revenue   - 1018400
    f. Total count of missing DUNS Number   - 606951
    g. Total count of missing Client Offices   - 614126

2. There are several NULL values in some of the predictor variables like Fee_Revenue, Client_Offices, DUNS_Number and the percentage of missing values ranges from 45-80%. I do not think these variables hold any statistical significance for our prediction problem and I am anyways planning to drop these variables and do not see any need to impute the missing values on these variables.

3. The target variable (Product Group) has a very small percentage of missing values and we will drop those values from Model training and evaluation. I believe the missing values on the target variable requires manual involvement and input from the business area to include them in the supervised learning process. There is a potential for these missing values to turn into a new label which may force our problem to be updated to a multiclass classification problem. For now, we will ignore the missing values and proceed with the assumption that this is a binary classification problem.

4. There are some variables with very strong positive and negative correlation which should technically be excluded from the model training (like for example, zip code & cnty population), I will drop some of them and leave the remaining to determine the importance of those features. There is a potential risk of overfitting but we should be able to re-train the model after observing the importance of those high correlation features.

5. The numeric variables that are missing values and part of our feature space will be imputed to the mean value.

6. Based on class frequency 80% of the observations have been classified as the "Primary" Product_Group which is significant observation. Will that mean most of the predictions would end up being classified as "Primary"? May be.

7. City does not seem to be having a large effect on the Product_Group classification. The distribution is pretty much the same across all the cities.

8. Bivariate distribution was not so informative on how some of the numeric variables that I was interested in were distributed against the target variable.

9. Market Type, Revenue Segmentation, Intermediary Market Type, Region may have effect on our final prediction. Majority of the observations fall under "Type-1" Market Type category, Central and South West regions have large number of observations and Segment-2 seems to have huge number of observations categorized under it.

10. Product line distribution seems to be pretty much evenly distributed except for one specific category. Industry segment has several Unknown values and need to determine how this feature impacts the model with the unknown values.

11. Some of the feature like the Client Key, Source_System_Name, census_match_id, Policy_ID etc. will be dropped because they have a very low significance or impact to our model and the problem we are trying to solve.

12. In attempt to make use of annual payroll, population and number of employees, I will group them under logical buckets as new features and use those derived features in model training & evaluation.

13. Industry segment appears in different shape and form in the dataset. There is sic_level_2, sic_level_2_name, sic_level_8, sic_level_8_name, management_niche, industrynicheconvention_desc, NAICSlevel1_Name, NAICSLevel1_Code. I dont think we would all these different levels of categorization and will choose only sic_level_8 and NAICSLevel1_Code for the model building and evaluation.

14. The outlier records will be dropped because they are missing several features and may s kew the model. The number of outlier records are minimal and I believe it should be ok to d rop them.

15. Features like zip code, effective year etc. are actually categorical values and need to be transformed using some kind of encoding but the risk is that these features have a very high cardinality. We will try using one hot encoding/label encoder for these features and o bserve the results.

Data Pre-Processing and Feature Engineering

Drop unneeded features

In [90]:

```
drop_cols =
['Client_Key','DUNS_Number','Placement_Market_Group_Key','Placement_Market_Company_Key','SIC_Level_
2',
             'SIC_Level_2_Name','SIC_Level_8_Name','IndustryNicheConventionDesc','NAICSLevel1Name',
             'Source_Product_Line','Brokerage_Expense','Commission_Revenue','Fee_Revenue','cnty_pop
_2010','RUCC_2013',
             'dcf_revenue','census_match_ind','PolicyID','Bill_Type','Source_System_Name','Manageme
nt_Niche',
             'Intermediary_Detail','Client_Offices']
client_full_df = client_df.drop(drop_cols,axis=1).copy()
client_full_df.head()
```

Out[90]:

	Global_Market_Tier	Placement_Market_Type	Placement_Market_Focus	Intermediary_Market_Type	SIC_Level_8	NAICSLevel1Code	Ef
0	NaN	Type 1	Focus 4	Type 3	0	0	
1	Tier 1	Type 1	Focus 2	Type 3	91110204	921110	
2	Tier 1	Type 1	Focus 2	Type 3	0	0	
3	Tier 2	Type 1	Focus 2	Type 3	70110302	721110	
4	Tier 3	Type 2	Focus 2	Type 1	0	0	

Semantic adjustments to features

In [91]:

```
client_full_df.rename({'Satellite': 'city'}, axis=1, inplace=True)
client_full_df.columns = map(str.lower, client_full_df.columns)
```

Impute missing value for Payroll & Employee count with mean

In [92]:

```
client_full_df = client_full_df[client_df.zip_code != 99999]
```

In [93]:

```
client_full_df['annual_payroll_fill'].fillna(client_full_df['annual_payroll_fill'].mean(skipna=True
```

```
), inplace=True)
```

In [94]:

```
client_full_df['num_employees_fill'].fillna(client_full_df['num_employees_fill'].mean(skipna=True),
inplace=True)
```

Drop records with missing values

In [95]:

```
client_full_df = client_full_df[pd.notnull(client_full_df['product_group'])]
```

In [96]:

```
client_full_df = client_full_df[pd.notnull(client_full_df['global_market_tier'])]
```

In [97]:

```
client_full_df['product_group_int']=client_full_df['product_group'].map({'Primary': 0, 'Secondary':
1})
client_full_df.drop(['product_group'],axis=1,inplace=True)
```

In [98]:

```
client_pipeline_df = client_full_df.copy()
client_full_df.shape
```

Out[98]:

```
(1284533, 20)
```

Convert features to string for Categorical Encoding

In [99]:

```
client_full_df.zip_code = client_full_df.zip_code.astype(str)
client_full_df.effective_year = client_full_df.effective_year.astype(str)
client_full_df.naicslevel1code = client_full_df.naicslevel1code.astype(str)
client_full_df.sic_level_8 = client_full_df.sic_level_8.astype(str)
```

In [100]:

```
# instantiate labelencoder object
categorical_cols=['global_market_tier','placement_market_type','placement_market_focus','intermedia
ry_market_type',
'client_premium_segmentation','client_revenue_segmentation','city','region','product_coverage','pro
duct_line']
le = LabelEncoder()
client_full_df[categorical_cols] = client_full_df[categorical_cols].apply(lambda col: le.fit_transf
orm(col))
client_full_df[categorical_cols].head(10)
```

Out[100]:

	global_market_tier	placement_market_type	placement_market_focus	intermediary_market_type	client_premium_segmentation	client_
2	2	0	1	3	1	
3	3	0	1	3	4	
4	4	1	1	1	1	
5	3	0	1	3	1	
6	4	1	2	2	2	
7	2	0	1	3	1	

	global_market_tier	placement_market_type	placement_market_focus	intermediary_market_type	client_premium_segmentation	client_
8	3	0	1	3	4	
9	4	0	1	3	5	
11	4	1	1	1	5	
12	4	2	3	3	1	

Split the dataset

In [103]:

```
client_full_x = client_full_df.copy()
client_full_x.drop(['product_group_int'],axis=1,inplace=True)
print(client_full_x.shape)

client_full_y = client_full_df['product_group_int'].copy()
print(client_full_y.shape)
```

```
(1284533, 19)
(1284533,)
```

In [104]:

```
train_x,test_x,train_y,test_y = train_test_split(client_full_x, client_full_y, random_state=42)
print(train_x.shape)
print(train_y.shape)
print(test_x.shape)
print(test_y.shape)
```

```
(963399, 19)
(963399,)
(321134, 19)
(321134,)
```

Create Sub Sample for Model Comparison

In [105]:

```
subsample_rate = 0.2
X_train, _, y_train, _ = train_test_split(train_x, train_y, stratify=train_y, train_size=subsample_
rate, random_state=42)
X_test, _, y_test, _ = train_test_split(test_x, test_y, stratify=test_y, train_size=subsample_rate,
random_state=42)
```

In [106]:

```
unique_train = np.unique(y_train, return_counts=True)
classes = [0,1]
list(zip(np.array(classes)[unique_train[0]], unique_train[1]))
```

Out[106]:

```
[(0, 159437), (1, 33242)]
```

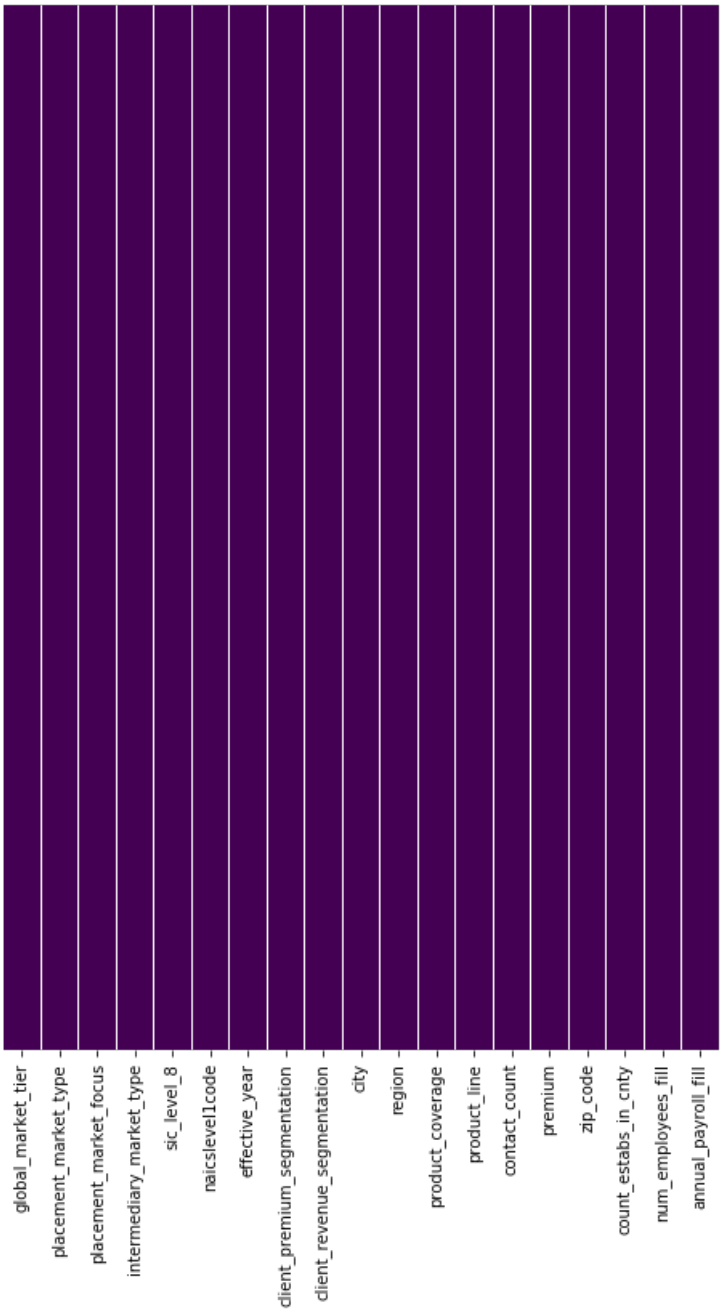
Heatmap - No Nulls

In [351]:

```
plt.figure(figsize=(8, 12))
#df_2 = pd.DataFrame(client_df, columns=["Product_Group", "num_employees_fill"])
sns.heatmap(X_train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[351]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1b169518>
```



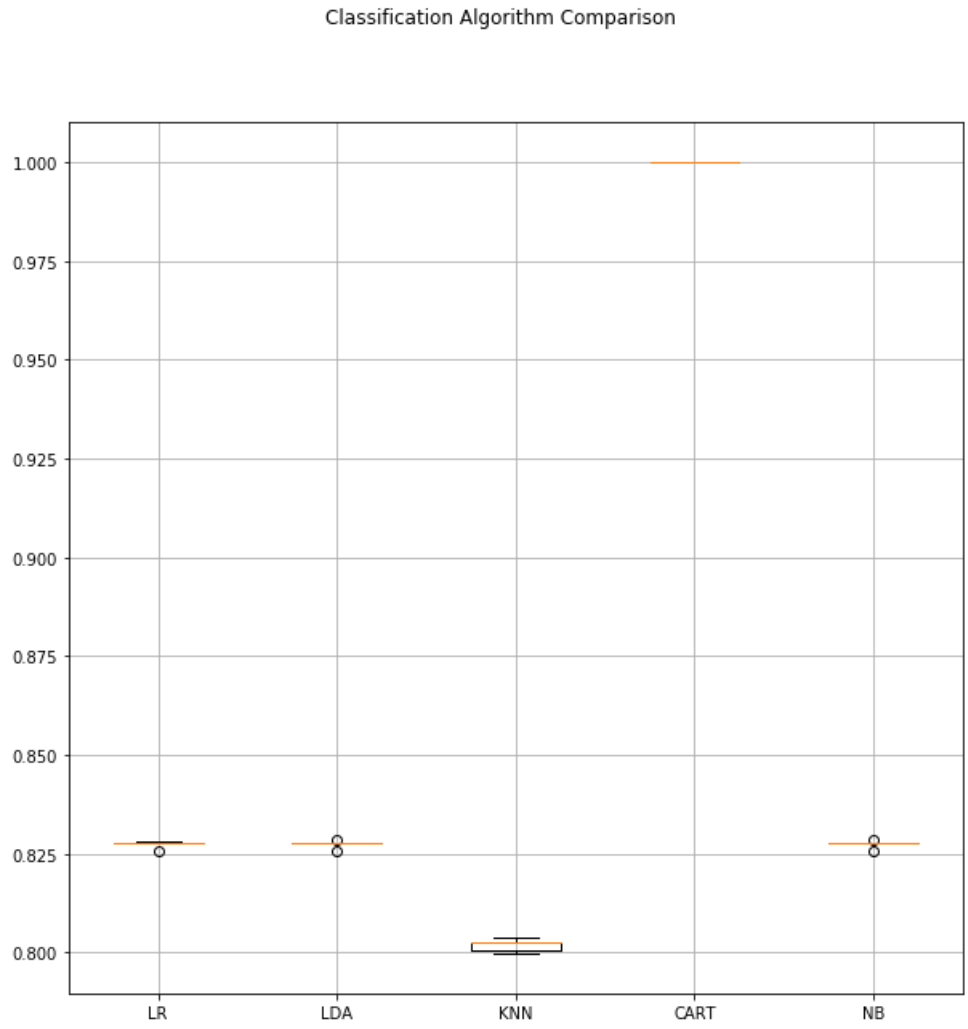
Model Comparison

In [33]:

```
# prepare models
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
#models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
#Standardize the data
for name, model in models:
    kfold = KFold(n_splits=5, random_state=7)
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
# boxplot algorithm comparison
```

```
fig = plt.figure(figsize=(10,10))
fig.suptitle('Classification Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.grid()
plt.show()
```

```
LR: 0.827392 (0.000953)
LDA: 0.827475 (0.001015)
KNN: 0.801753 (0.001421)
CART: 1.000000 (0.000000)
NB: 0.827475 (0.001015)
```



Model Building and Cross Validation using Grid Search

In [45]:

```
from sklearn.base import BaseEstimator, TransformerMixin

# Create a class to select numerical or categorical columns
# since Scikit-Learn doesn't handle DataFrames yet
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values

class CategoricalEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, cat_cols):
        self.cat_cols=cat_cols
        return None
    def fit(self, X, y=None):
```

```
        self.X=X
        return self
    def transform(self, X):
        X = pd.DataFrame(X)
        X.columns = self.cat_cols
        for col in self.cat_cols:
            lbl = LabelEncoder()
            lbl.fit(list(X[col].values.astype('str')))
            X[col] = lbl.transform(list(X[col].values.astype('str')))
        return X
```

In [46]:

```
num_cols =
['premium','contact_count','num_employees_fill','annual_payroll_fill','count_estabs_in_cnty']
cat_cols =
['global_market_tier','placement_market_type','placement_market_focus','intermediary_market_type',
'client_premium_segmentation','client_revenue_segmentation','city','region','product_coverage','pro
duct_line',
'zip_code','effective_year','naicslevel1code','sic_level_8']

num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_cols)),
        ('std_scaler', StandardScaler())
        #('std_scaler', MinMaxScaler())
    ])

cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_cols)),
        ('label_encoder', CategoricalEncoder(cat_cols))
    ])

full_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline)
    ])
```

Split the data using Sklearn

In [107]:

```
client_full_x = client_pipeline_df.copy()
client_full_x.drop(['product_group_int'],axis=1,inplace=True)
print(client_full_x.shape)

client_full_y = client_full_df['product_group_int'].copy()
print(client_full_y.shape)

train_x,test_x,train_y,test_y = train_test_split(client_full_x, client_full_y, random_state=42)
print(train_x.shape)
print(train_y.shape)
print(test_x.shape)
print(test_y.shape)

X_train_split, X_test_split, y_train_split, y_test_split = train_test_split(train_x, train_y, test_
size=0.2, random_state=42)
print(X_train_split.shape)
print(X_test_split.shape)
```

```
(1284533, 19)
(1284533,)
(963399, 19)
(963399,)
(321134, 19)
(321134,)
(770719, 19)
(192680, 19)
```

In [163]:

```
X_train_prepared = full_pipeline.fit_transform(X_train_split)
X_test_prepared = full_pipeline.fit_transform(X_test_split)
```

In [164]:

```
X_train_prepared.shape
```

Out[164]:

```
(770719, 19)
```

Train Decision Tree Classifier

In [165]:

```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train_prepared, y_train_split)
```

Out[165]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [166]:

```
results = pd.DataFrame(columns=["Model description", "Train Accuracy Score", "Test Accuracy Score"
, "Train Pred Time", "Test Pred Time"])
pd.set_option('display.max_colwidth', 0)

start = time.time()
y_train_pred = dt_model.predict(X_train_prepared)
train_time = time.time() - start
start = time.time()
y_test_pred = dt_model.predict(X_test_prepared)
test_time = time.time() - start

# add the result of this experiment ot the log book
results.loc[len(results)] = ["SKLearn Decision Tree Classifier Model",
                             accuracy_score(y_train_pred, y_train_split),
                             accuracy_score(y_test_pred, y_test_split),
                             train_time,
                             test_time]
results
```

Out[166]:

	Model description	Train Accuracy Score	Test Accuracy Score	Train Pred Time	Test Pred Time
0	SKLearn Decision Tree Classifier Model	1.0	1.0	0.054175	0.014146

Train Gaussian NB Classifier

In [167]:

```
nb_model = GaussianNB()
nb_model.fit(X_train_prepared, y_train_split)
```

Out[167]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [168]:

```
start = time.time()
y_train_pred = nb_model.predict(X_train_prepared)
train_time = time.time() - start
```

```
start = time.time()
y_test_pred = nb_model.predict(X_test_prepared)
test_time = time.time() - start

# add the result of this experiment ot the log book
results.loc[len(results)] = ["SKLearn Gaussian NB Classifier Model",
                             accuracy_score(y_train_pred, y_train_split),
                             accuracy_score(y_test_pred, y_test_split),
                             train_time,
                             test_time]
results
```

Out[168]:

	Model description	Train Accuracy Score	Test Accuracy Score	Train Pred Time	Test Pred Time
0	SKLearn Decision Tree Classifier Model	1.000000	1.000000	0.054175	0.014146
1	SKLearn Gaussian NB Classifier Model	0.827706	0.826547	0.345611	0.088409

Train KNN Classifier

In [128]:

```
knn_model = KNeighborsClassifier()
knn_model.fit(X_train_prepared, y_train_split)
```

Out[128]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

In [132]:

```
start = time.time()
y_train_pred = knn_model.predict(X_train_prepared)
train_time = time.time() - start
start = time.time()
y_test_pred = knn_model.predict(X_test_prepared)
test_time = time.time() - start

# add the result of this experiment ot the log book
results.loc[len(results)] = ["SKLearn KNN Classifier Model",
                             accuracy_score(y_train_pred, y_train_split),
                             accuracy_score(y_test_pred, y_test_split),
                             train_time,
                             test_time]
results
```

Out[132]:

	Model description	Train Accuracy Score	Test Accuracy Score	Train Pred Time	Test Pred Time
0	SKLearn Decision Tree Classifier Model	1.000000	1.000000	0.070314	0.017204
1	SKLearn Gaussian NB Classifier Model	0.827706	0.826547	0.559050	0.081415
2	SKLearn KNN Classifier Model	0.842978	0.800670	24.232563	6.826487

Train Random Forest Classifier

In [135]:

```
rf_model = RandomForestClassifier()
rf_model.fit(X_train_prepared, y_train_split)
```

Out[135]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [136]:

```
start = time.time()
y_train_pred = rf_model.predict(X_train_prepared)
train_time = time.time() - start
start = time.time()
y_test_pred = rf_model.predict(X_test_prepared)
test_time = time.time() - start

# add the result of this experiment ot the log book
results.loc[len(results)] = ["SKLearn Random Forest Classifier Model",
                             accuracy_score(y_train_pred, y_train_split),
                             accuracy_score(y_test_pred, y_test_split),
                             train_time,
                             test_time]
results
```

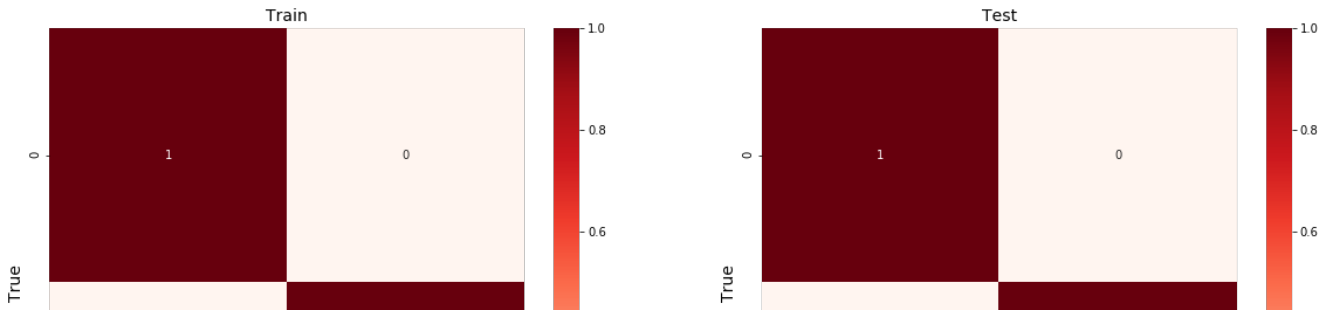
Out[136]:

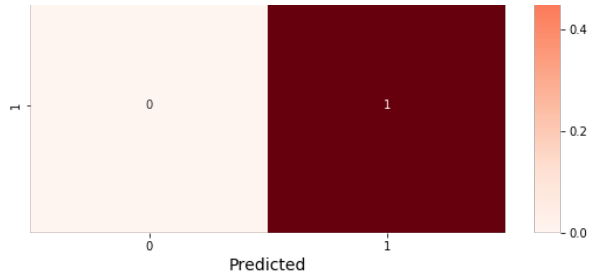
	Model description	Train Accuracy Score	Test Accuracy Score	Train Pred Time	Test Pred Time
0	SKLearn Decision Tree Classifier Model	1.000000	1.000000	0.070314	0.017204
1	SKLearn Gaussian NB Classifier Model	0.827706	0.826547	0.559050	0.081415
2	SKLearn KNN Classifier Model	0.842978	0.800670	24.232563	6.826487
3	SKLearn Random Forest Classifier Model	1.000000	0.999834	0.908342	0.212016

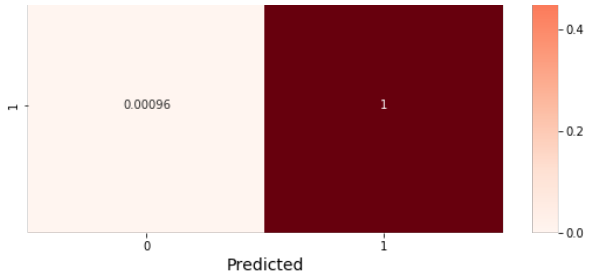
Confusion Matrix for Random Forest Tree Classifier (one example)

In [142]:

```
cm_train = confusion_matrix(y_train_split, y_train_pred).astype(np.float32)
cm_train /= cm_train.sum(axis=1)[:, np.newaxis]

cm_test = confusion_matrix(y_test_split, y_test_pred).astype(np.float32)
cm_test /= cm_test.sum(axis=1)[:, np.newaxis]

plt.figure(figsize=(20, 8))
plt.subplot(121)
g = sns.heatmap(cm_train, vmin=0, vmax=1, annot=True, cmap="Reds")
plt.xlabel("Predicted", fontsize=14)
plt.ylabel("True", fontsize=14)
g.set(xticklabels=[0,1], yticklabels=[0,1])
plt.title("Train", fontsize=14)

plt.subplot(122)
g = sns.heatmap(cm_test, vmin=0, vmax=1, annot=True, cmap="Reds")
plt.xlabel("Predicted", fontsize=14)
plt.ylabel("True", fontsize=14)
g.set(xticklabels=[0,1], yticklabels=[0,1])
plt.title("Test", fontsize=14);
```







Train LR Classifier

In [151]:

```
lr_model = LogisticRegression()
lr_model.fit(X_train_prepared, y_train_split)
```

Out[151]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [152]:

```
start = time.time()
y_train_pred = lr_model.predict(X_train_prepared)
train_time = time.time() - start
start = time.time()
y_test_pred = lr_model.predict(X_test_prepared)
test_time = time.time() - start

# add the result of this experiment ot the log book
results.loc[len(results)] = ["SKLearn LR Classifier Model",
                             accuracy_score(y_train_pred, y_train_split),
                             accuracy_score(y_test_pred, y_test_split),
                             train_time,
                             test_time]
results
```

Out[152]:

	Model description	Train Accuracy Score	Test Accuracy Score	Train Pred Time	Test Pred Time
0	SKLearn Decision Tree Classifier Model	1.000000	1.000000	0.070314	0.017204
1	SKLearn Gaussian NB Classifier Model	0.827706	0.826547	0.559050	0.081415
2	SKLearn KNN Classifier Model	0.842978	0.800670	24.232563	6.826487
3	SKLearn Random Forest Classifier Model	1.000000	0.999834	0.908342	0.212016
4	SKLearn LR Classifier Model	0.827696	0.826541	0.044512	0.010002

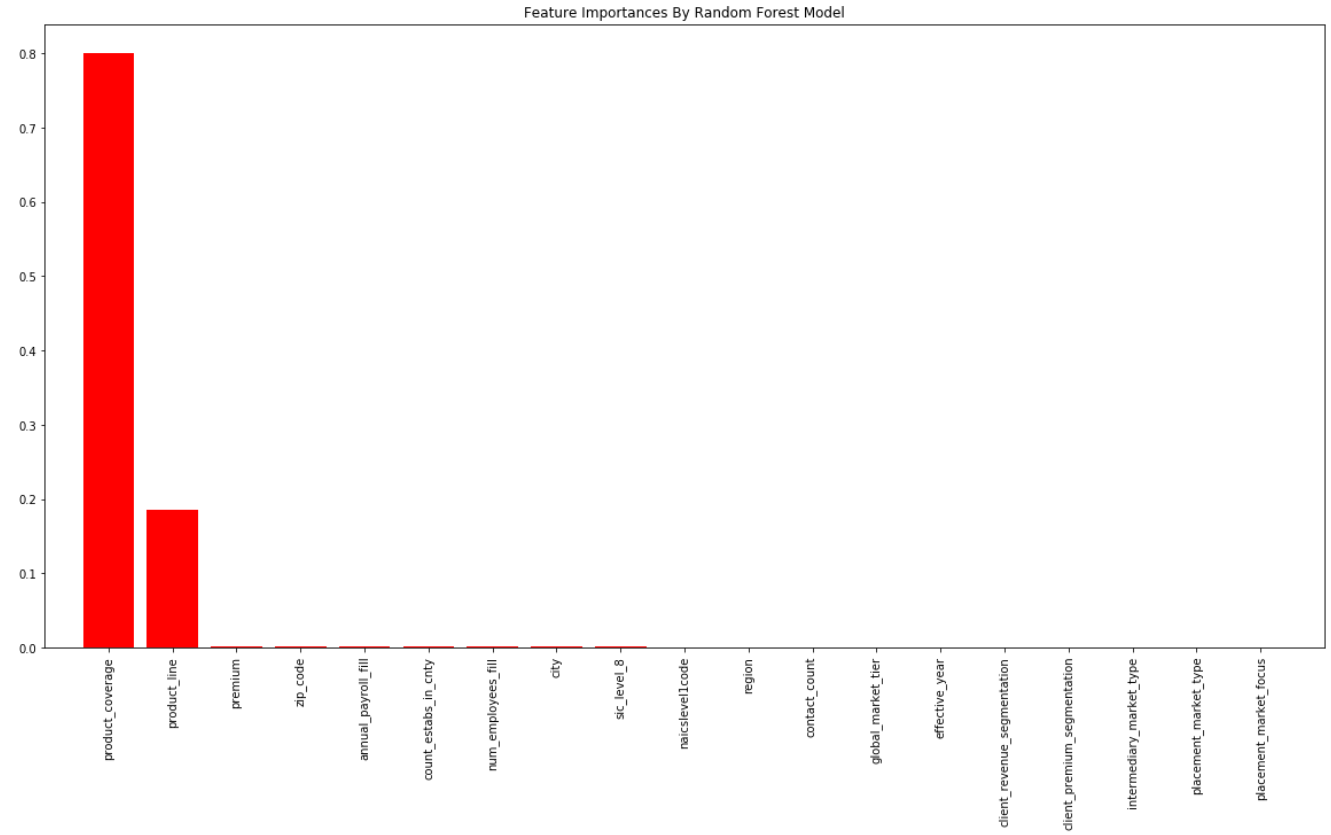
Feature Importance

Random Forest Classifier

In [144]:

```
importances=rf_model.feature_importances_
imp_con = np.array(importances[0:50])
std = np.std([imp_con for tree in rf_model],
             axis=0)
indices = np.argsort(imp_con)[::-1]
sorted_important_features=[]
predictors=list(num_cols+cat_cols)
for i in indices:
```

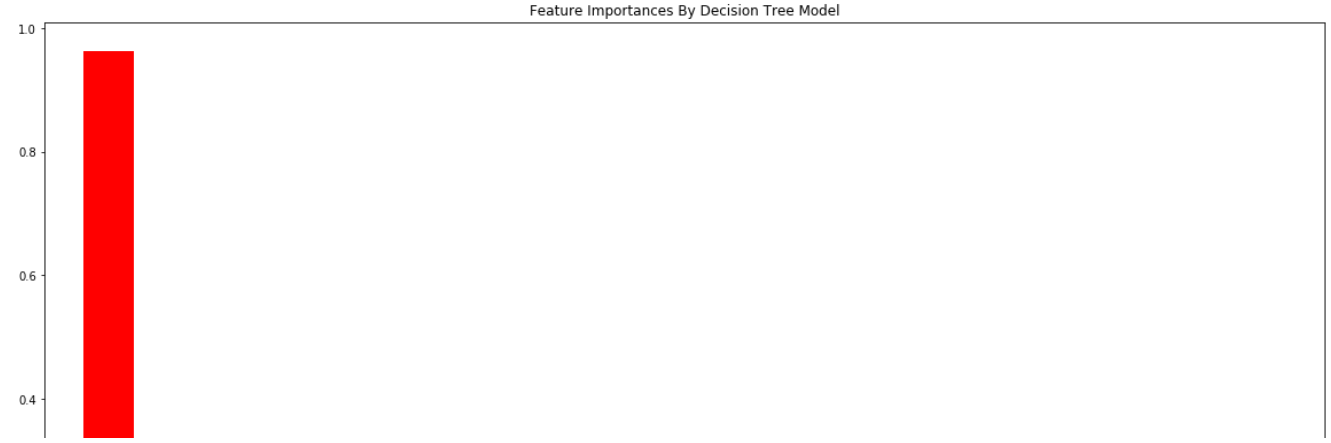
```
    sorted_important_features.append(predictors[i])
plt.figure(figsize=(20,10))
plt.title("Feature Importances By Random Forest Model")
plt.bar(range(np.size(predictors)), imp_con[indices], color="r", yerr=std[indices], align="center")
plt.xticks(range(np.size(predictors)), sorted_important_features, rotation='vertical')

plt.xlim([-1, np.size(predictors)]);
```

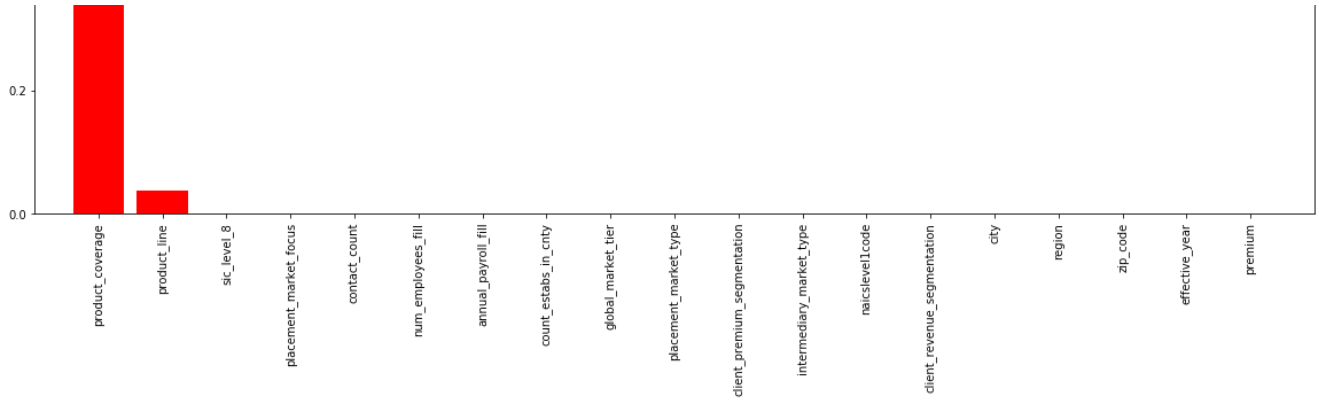


Decision Tree Classifier

In [148]:

```
importances=dt_model.feature_importances_
imp_con = np.array(importances[0:50])
indices = np.argsort(imp_con)[::-1]
sorted_important_features=[]
predictors=list(num_cols+cat_cols)
for i in indices:
    sorted_important_features.append(predictors[i])
plt.figure(figsize=(20,10))
plt.title("Feature Importances By Decision Tree Model")
plt.bar(range(np.size(predictors)), imp_con[indices], color="r", yerr=std[indices], align="center")
plt.xticks(range(np.size(predictors)), sorted_important_features, rotation='vertical')

plt.xlim([-1, np.size(predictors)]);
```





Feature Importance Scores with frequency

Create a table with the feature importance scores and values with max occurrence for each of the features along with their frequencies and save to a database table. This table can be used as the reason code determination table and the prediction rationale can be developed from this table

Final Results and Discussion

Model Pipeline

As part of this case study, we built multiple classification models with different parameter setting using algorithms from SKlearn library mainly but without any major hyper parameter tuning. Our objective as we explored the data, pre-processing & feature extraction techniques further was to conduct multiple experiments using different algorithms to build an optimized model that predicts with high accuracy. Have we explored different hyper parameter setting for the the models that came up with a low accuracy score? We did not do it yet but this will be the next action and will continue to explore the hyper parameter space for these low scoring models which will help understand deeply on what features are impacting the accuracy and may be learning for future implementations.

Pre-processing and Feature Engineering

In the earlier sections we attempted to build new features/encoded features and were able to visualize the importance of these new features in the different models that we built, I also have ideas for new features which can be explored in the phase and will be part of my future extension/study. I was able to improve the model processing through pre-processing and excluded features from the model training that were least important, feature importance information would help to even further reduce the feature space and will be of great significance during operationalization, we also explored different imputation strategies, dropped unneeded features, scaled numeric data so they have uniform variance, data type conversions, semantic adjustments, excluded highly correlated features, tried to identify outliers and tried different encoding strategy for categorical attributes and specifically on the categorical attributes we resorted to regular label encoder which seemed to perform better. One hot encoding did not generate the same level of accuracy as the regular label encoder. Some of the features like industry segment had very little impact on the model which is interesting.

Outcome of Experiments

As described in the model pipeline section for the problem setting, we explored multiple models including Decision Treem, Random Forest, LDA, Gaussian NB, KNN etc with basic parameter setting and used cross validation to identify the best estimator that produced the best accuracy score. We performed cross validation on our training set and explored the results of cross validation. I also attempted to fit support vector machine classifier which performed poorly and wont be recommending for the problem that we are trying to solve. The KNN classifier although produced a reasonable accuracy will not be an ideal candidate for operationalization because of the performance. Tree/Ensemble methods seem to be recommended models for real time implementation but I would also need to understand the difference in performance if more observations are included in the training set. The experimental results for each of the algorithms that we explored can be found below.

Model Algorithms

Following are the different algorithms that we explored in our project with different parameter setting. I am also looking to build home grown optimizers from the knowledge that we gathered in the course as part of future extension to this case study.

1. KNN Classifier
2. Gaussian NB

3. Random Forest Classifier
4. Decision Tree Classifier
5. LDA
6. SVC (Support Vector Machine)
7. Logistic Regression Classifier

Results Table

In [153]:

```
results
```

Out[153]:

	Model description	Train Accuracy Score	Test Accuracy Score	Train Pred Time	Test Pred Time
0	SKLearn Decision Tree Classifier Model	1.000000	1.000000	0.070314	0.017204
1	SKLearn Gaussian NB Classifier Model	0.827706	0.826547	0.559050	0.081415
2	SKLearn KNN Classifier Model	0.842978	0.800670	24.232563	6.826487
3	SKLearn Random Forest Classifier Model	1.000000	0.999834	0.908342	0.212016
4	SKLearn LR Classifier Model	0.827696	0.826541	0.044512	0.010002

Save Model to Pickle file

In [169]:

```
import pickle
from sklearn.externals import joblib

joblib.dump(dt_model, 'dt_model.pkl')
joblib.dump(rf_model, 'rf_model.pkl')
joblib.dump(lr_model, 'lr_model.pkl')
joblib.dump(knn_model, 'knn_model.pkl')
joblib.dump(nb_model, 'nb_model.pkl')

joblib.dump(dt_model, 'dt_model_1.sav')
joblib.dump(rf_model, 'rf_model_1.sav')
joblib.dump(full_pipeline, 'full_pipeline.sav')
# Save the trained model as a pickle string.
saved_dt_model = pickle.dumps(dt_model)
saved_rf_model = pickle.dumps(rf_model)
```