Table of Contents

Tools and Data Management

Standard Boilers

In [1]:
```
# Import basic libraries
import matplotlib.pyplot as plt
import numpy as np # for linear algebra eg np.log, np.where, np.mean, np.std
import pandas as pd # for import data eg pd.read_csv, pd.DataFrame or pd.Series, pd.get_dummies(col_name,drop_first=True)
import seaborn as sns # mainly for visualisation
import plotly.express as px #plotly visualisation (aka ggplot2 in R)
from tabulate import tabulate #construct table
from datetime import datetime# if dealing with date and time or time series data
sns.set() # Optional:: this just make the plot nicer by changing the color, fontsize etc

#This syntax helps to display inline within frontends like the Jupyter notebook,
#directly below the code cell that produced it.
#The resulting plots will published nicedly within the notebook document.
%matplotlib inline
#display all float to 3dp
pd.set_option('display.float','{:.3f}'.format)
pd.set_option('display.max_columns',50)
pd.set_option('display.max_rows',100)

#Warning messages are typically issued in situations where it is useful to alert the user of some condition in a program,
#where that condition (normally) doesn't warrant raising an exception and terminating the program.
#For example, one might want to issue a warning when a program uses an obsolete module.
from warnings import filterwarnings
filterwarnings('ignore')
```

Scikit-Learn Parts

In [354]:
```
# Imports relevant scikit-learn to this project
# tools for data management
import missingno as msno
from fancyimpute import KNN
from sklearn.impute import SimpleImputer, IterativeImputer
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split, learning_curve, KFold # i
mport KFold
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_selection import RFE, RFECV  ##Recursive Feature Eliminati
on with Cross Validation
from sklearn.pipeline import Pipeline, FeatureUnion

#for Decision Tree Visualisation
from sklearn import tree
#from sklearn.tree import export_graphviz
import pydotplus
from scipy import stats
from statsmodels.api import qqplot
# unsupervised learnings
from sklearn.cluster import KMeans

# supervised learnings
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, Lasso
from sklearn.linear_model import ElasticNet
from sklearn.kernel_ridge import KernelRidge
#from sklearn.svm import SVR
from sklearn import svm
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesRegressor, ExtraTreesClassifier
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.ensemble import GradientBoostingRegressor, GradientBoostingClassif
ier
from sklearn.ensemble import AdaBoostRegressor, AdaBoostClassifier
from xgboost.sklearn import XGBClassifier
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostRegressor, Pool, cv
from keras.wrappers.scikit_learn import KerasClassifier
from mlxtend.regressor import StackingCVRegressor
from keras import models,layers

# Model accuracy measures
import sklearn.metrics as metrics
from sklearn.metrics import log_loss
from sklearn.metrics import mean_squared_error as MSE, r2_score, confusion_matr
ix
from sklearn.metrics import classification_report, recall_score, precision_scor
e, accuracy_score, f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
import scikitplot as skplt #Lift Curve / Gain Chart
```

Acquire Data

Data Ingestion

In [3]:
```
#Import files/data
file_path = '../PGDADS_Capstone Assignment/Office Supply Campaign ResultS 7-23-19.xlsx'
df = pd.read_excel(file_path) ##,index_col=0)
print('Data is loaded successfully!')
```

```
Data is loaded successfully!
```

In [4]:
```
print('There are {} observations and {} features in the dataset:'.format(*df.shape))
print('\t Of which, {} numeric features'.format(df.select_dtypes(include='float64').shape[1]))
print('\t And, {} non-numeric/object features.'.format(df.select_dtypes(include='object').shape[1]))
print('\t And, {} non-numeric/non-object features.'.format(df.select_dtypes(exclude=['float64','object']).shape[1]))
```

```
There are 16173 observations and 21 features in the dataset:
         Of which, 7 numeric features
         And, 13 non-numeric/object features.
         And, 1 non-numeric/non-object features.
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16173 entries, 0 to 16172
Data columns (total 21 columns):
Customer Number                      16172 non-null float64
Campaign Period Sales                16172 non-null float64
Historical Sales Volume              16172 non-null float64
Date of First Purchase               16172 non-null datetime64[ns]
Number of Prior Year Transactions    16172 non-null float64
Do Not Direct Mail Solicit           16172 non-null float64
Do Not Email                         16172 non-null float64
Do Not Telemarket                    16172 non-null float64
Repurchase Method                    16172 non-null object
Last Transaction Channel             15730 non-null object
Desk                                 16173 non-null object
Executive Chair                      16171 non-null object
Standard Chair                       16171 non-null object
Monitor                              16171 non-null object
Printer                              16171 non-null object
Computer                             16172 non-null object
Insurance                            16170 non-null object
Toner                                16170 non-null object
Office Supplies                      16172 non-null object
Number of Employees                  16170 non-null object
Language                             11701 non-null object
dtypes: datetime64[ns](1), float64(7), object(13)
memory usage: 2.6+ MB
```

Exploratory Analysis

In [6]:
```
df.rename(columns=lambda col: col.replace(' ', '_'), inplace=True)
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16173 entries, 0 to 16172
Data columns (total 21 columns):
Customer_Number                      16172 non-null float64
Campaign_Period_Sales                16172 non-null float64
Historical_Sales_Volume              16172 non-null float64
Date_of_First_Purchase               16172 non-null datetime64[ns]
Number_of_Prior_Year_Transactions    16172 non-null float64
Do_Not_Direct_Mail_Solicit           16172 non-null float64
Do_Not_Email                         16172 non-null float64
Do_Not_Telemarket                    16172 non-null float64
Repurchase_Method                    16172 non-null object
Last_Transaction_Channel             15730 non-null object
Desk                                 16173 non-null object
Executive_Chair                      16171 non-null object
Standard_Chair                       16171 non-null object
Monitor                              16171 non-null object
Printer                              16171 non-null object
Computer                             16172 non-null object
Insurance                            16170 non-null object
Toner                                16170 non-null object
Office_Supplies                      16172 non-null object
Number_of_Employees                  16170 non-null object
Language                             11701 non-null object
dtypes: datetime64[ns](1), float64(7), object(13)
memory usage: 2.6+ MB
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16173 entries, 0 to 16172
Data columns (total 21 columns):
Customer_Number                      16172 non-null float64
Campaign_Period_Sales                16172 non-null float64
Historical_Sales_Volume              16172 non-null float64
Date_of_First_Purchase               16172 non-null datetime64[ns]
Number_of_Prior_Year_Transactions    16172 non-null float64
Do_Not_Direct_Mail_Solicit           16172 non-null float64
Do_Not_Email                         16172 non-null float64
Do_Not_Telemarket                    16172 non-null float64
Repurchase_Method                    16172 non-null object
Last_Transaction_Channel             15730 non-null object
Desk                                 16173 non-null object
Executive_Chair                      16171 non-null object
Standard_Chair                       16171 non-null object
Monitor                              16171 non-null object
Printer                              16171 non-null object
Computer                             16172 non-null object
Insurance                            16170 non-null object
Toner                                16170 non-null object
Office_Supplies                      16172 non-null object
Number_of_Employees                  16170 non-null object
Language                             11701 non-null object
dtypes: datetime64[ns](1), float64(7), object(13)
memory usage: 2.6+ MB
```

Identify Areas of Problem

Invalid Records | Type Errors | Missing Values

In [9]:
```
print(f"Unique values in each non-numeric column:\t")
for col in df.select_dtypes(include=[object]):
    print(col,"           :\n", df[col].unique())
```

```
Unique values in each non-numeric column:
Repurchase_Method            :
 ['AUTO RENEW' 'NOTICE' 'PAYMENT PLAN' nan]
Last_Transaction_Channel            :
 ['AUTO RENEW' 'MAIL' 'PHONE' 'BRANCH (PHONE)' 'WEB' nan 'BRANCH (POS)'
 'IT' 'BILLING']
Desk            :
 ['N' 'Y' 'YY' 911]
Executive_Chair            :
 ['N' 'Y' nan]
Standard_Chair            :
 ['N' 'Y' nan]
Monitor            :
 ['N' 'Y' nan]
Printer            :
 ['N' 'Y' nan]
Computer            :
 ['N' 'Y' 'YY' nan]
Insurance            :
 ['Y' 'N' 'YY' nan]
Toner            :
 ['N' 'Y' nan 'YY']
Office_Supplies            :
 ['Y' 'N' nan]
Number_of_Employees            :
 ['6-10' '11-50' '1-5' ' ' '51-100' '101-500' '500+' nan]
Language            :
 ['English' nan 'Hindi' 'Italian' 'French' 'Chinese' 'Portuguese' 'Russian'
 'Spanish' 'Hebrew' 'Japanese' 'German' 'Polish' 'Arabic' 'Greek'
 'Vietnamese' 'Korean' 'Thai' 'Pashto']
```

In [10]:
```
print("Unique values in each non-numeric column:\t")
for col in df.select_dtypes(include=[object]):
    print(col,":", df[col].nunique())
```

```
Unique values in each non-numeric column:
Repurchase_Method : 3
Last_Transaction_Channel : 8
Desk : 4
Executive_Chair : 2
Standard_Chair : 2
Monitor : 2
Printer : 2
Computer : 3
Insurance : 3
Toner : 3
Office_Supplies : 2
Number_of_Employees : 7
Language : 18
```

Remove Invalid Record(s)

In [11]:
```
df_updated = df[df['Desk'] != 911].copy()
```

In [12]:
```
df_updated.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16172 entries, 0 to 16171
Data columns (total 21 columns):
Customer_Number                         16172 non-null float64
Campaign_Period_Sales                   16172 non-null float64
Historical_Sales_Volume                 16172 non-null float64
Date_of_First_Purchase                  16172 non-null datetime64[ns]
Number_of_Prior_Year_Transactions       16172 non-null float64
Do_Not_Direct_Mail_Solicit              16172 non-null float64
Do_Not_Email                            16172 non-null float64
Do_Not_Telemarket                       16172 non-null float64
Repurchase_Method                       16172 non-null object
Last_Transaction_Channel                15730 non-null object
Desk                                    16172 non-null object
Executive_Chair                         16171 non-null object
Standard_Chair                          16171 non-null object
Monitor                                 16171 non-null object
Printer                                 16171 non-null object
Computer                                16172 non-null object
Insurance                               16170 non-null object
Toner                                   16170 non-null object
Office_Supplies                         16172 non-null object
Number_of_Employees                     16170 non-null object
Language                                11701 non-null object
dtypes: datetime64[ns](1), float64(7), object(13)
memory usage: 2.7+ MB
```

In [13]:
```
print('There are {} observations and {} features in the dataset:'.format(*df_up
dated.shape))
print('\t Of which, {} numeric features'.format(df_updated.select_dtypes(includ
e='float64').shape[1]))
print('\t And, {} non-numeric/object features.'.format(df_updated.select_dtypes
(include='object').shape[1]))
print('\t And, {} non-numeric/non-object features.'.format(df_updated.select_dt
ypes(exclude=['float64','object']).shape[1]))
print('\t Removed {} feature(s).'.format(abs(df_updated.shape[1] - df.shape
[1])))
print('\t Removed {} observation(s) (i.e. Desk = 911).'.format(abs(df_updated.s
hape[0] - df.shape[0])))
```

```
There are 16172 observations and 21 features in the dataset:
         Of which, 7 numeric features
         And, 13 non-numeric/object features.
         And, 1 non-numeric/non-object features.
         Removed 0 feature(s).
         Removed 1 observation(s) (i.e. Desk = 911).
```

Remove Negative Sales Values

In [14]:
```
neg_bool = ((df_updated['Campaign_Period_Sales'] < 0) | (df_updated['Historical
_Sales_Volume'] < 0))
print(f"Number of negative values:\t{sum(neg_bool)}")
display(df_updated.loc[neg_bool, :])
```

```
Number of negative values:      10
```

	Customer_Number	Campaign_Period_Sales	Historical_Sales_Volume	Date_of_First_Purchase	Numbe
2216	5125266.000	0.000	-126638.400	1995-10-01	
5783	12970830.000	-119.333	69223.560	1972-04-15	
5919	13276523.000	-140.960	500408.000	1976-10-15	
6135	13753769.000	-14.320	24057.600	1969-01-01	
6139	13758714.000	0.000	-20021.760	2015-07-22	
9300	20726836.000	0.000	-9184.000	2011-01-05	
10689	23849132.000	-421.833	13543.210	2006-05-15	
11934	26677728.000	-161.167	27398.333	1993-01-15	
12548	28176445.000	0.000	-164303.100	2012-10-31	
15840	56184607.000	-566.500	48152.670	2014-10-31	

In [15]:
```
df_vers = df_updated.copy()
df_vers = df_vers[~df_vers['Customer_Number'].isin(df_updated['Customer_Number
'][neg_bool])]
df_vers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16162 entries, 0 to 16171
Data columns (total 21 columns):
Customer_Number                       16162 non-null float64
Campaign_Period_Sales                 16162 non-null float64
Historical_Sales_Volume               16162 non-null float64
Date_of_First_Purchase                16162 non-null datetime64[ns]
Number_of_Prior_Year_Transactions     16162 non-null float64
Do_Not_Direct_Mail_Solicit            16162 non-null float64
Do_Not_Email                          16162 non-null float64
Do_Not_Telemarket                     16162 non-null float64
Repurchase_Method                     16162 non-null object
Last_Transaction_Channel              15720 non-null object
Desk                                  16162 non-null object
Executive_Chair                       16161 non-null object
Standard_Chair                        16161 non-null object
Monitor                               16161 non-null object
Printer                               16161 non-null object
Computer                              16162 non-null object
Insurance                             16160 non-null object
Toner                                 16160 non-null object
Office_Supplies                       16162 non-null object
Number_of_Employees                   16160 non-null object
Language                              11695 non-null object
dtypes: datetime64[ns](1), float64(7), object(13)
memory usage: 2.7+ MB
```

In [16]:
```
print('There are {} observations and {} features in the dataset:'.format(*df_ve
rs.shape))
print('\t Of which, {} numeric features'.format(df_vers.select_dtypes(include='
float64').shape[1]))
print('\t And, {} non-numeric/object features.'.format(df_vers.select_dtypes(in
clude='object').shape[1]))
print('\t And, {} non-numeric/non-object features.'.format(df_vers.select_dtype
s(exclude=['float64','object']).shape[1]))
print('\t Removed {} feature(s).'.format(abs(df_vers.shape[1] - df_updated.shap
e[1])))
print('\t Removed {} observation(s) (i.e. Negative Sales Values).'.format(abs(d
f_vers.shape[0] - df_updated.shape[0])))
```

```
There are 16162 observations and 21 features in the dataset:
         Of which, 7 numeric features
         And, 13 non-numeric/object features.
         And, 1 non-numeric/non-object features.
         Removed 0 feature(s).
         Removed 10 observation(s) (i.e. Negative Sales Values).
```

Fix Typo Error(s) & Change from Y/N to Binary

In [17]:
```
df_vers['Desk'] = df_vers['Desk'].map({'YY':1.,'Y':1.,'N':0.})
df_vers['Executive_Chair'] = df_vers['Executive_Chair'].map({'YY':1.,'Y':1.,'N
':0.})
df_vers['Standard_Chair'] = df_vers['Standard_Chair'].map({'YY':1.,'Y':1.,'N':
0.})
df_vers['Monitor'] = df_vers['Monitor'].map({'YY':1.,'Y':1.,'N':0.})
df_vers['Printer'] = df_vers['Printer'].map({'YY':1.,'Y':1.,'N':0.})
df_vers['Computer'] = df_vers['Computer'].map({'YY':1.,'Y':1.,'N':0.})
df_vers['Insurance'] = df_vers['Insurance'].map({'YY':1.,'Y':1.,'N':0.})
df_vers['Toner'] = df_vers['Toner'].map({'YY':1.,'Y':1.,'N':0.})
df_vers['Office_Supplies'] = df_vers['Office_Supplies'].map({'YY':1.,'Y':1.,'N
':0.})
```

Impute '' as NaN

In [18]:
```
df_vers['Number_of_Employees'] = np.where(df_vers['Number_of_Employees']== ' ',
np.nan,df_vers['Number_of_Employees'])
```

In [19]:
```
df_vers1 = df_vers.copy()
```

In [20]:
```
df_vers1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16162 entries, 0 to 16171
Data columns (total 21 columns):
Customer_Number                         16162 non-null float64
Campaign_Period_Sales                   16162 non-null float64
Historical_Sales_Volume                 16162 non-null float64
Date_of_First_Purchase                  16162 non-null datetime64[ns]
Number_of_Prior_Year_Transactions       16162 non-null float64
Do_Not_Direct_Mail_Solicit              16162 non-null float64
Do_Not_Email                            16162 non-null float64
Do_Not_Telemarket                       16162 non-null float64
Repurchase_Method                       16162 non-null object
Last_Transaction_Channel                15720 non-null object
Desk                                    16162 non-null float64
Executive_Chair                         16161 non-null float64
Standard_Chair                          16161 non-null float64
Monitor                                 16161 non-null float64
Printer                                 16161 non-null float64
Computer                                16162 non-null float64
Insurance                               16160 non-null float64
Toner                                   16160 non-null float64
Office_Supplies                         16162 non-null float64
Number_of_Employees                     12418 non-null object
Language                                11695 non-null object
dtypes: datetime64[ns](1), float64(16), object(4)
memory usage: 2.7+ MB
```

In [21]:
```
print('There are {} observations and {} features in the dataset:'.format(*df_ve
rs.shape))
print('\t Of which, {} numeric features'.format(df_vers.select_dtypes(include='
float64').shape[1]))
print('\t And, {} non-numeric/object features.'.format(df_vers.select_dtypes(in
clude='object').shape[1]))
print('\t And, {} non-numeric/non-object features.'.format(df_vers.select_dtype
s(exclude=['float64','object']).shape[1]))
print('\t Added {} feature(s).'.format(abs(df_vers.shape[1] - df_vers1.shape
[1])))
print('\t Removed {} observation(s).'.format(abs(df_vers1.shape[0] - df_vers.sh
ape[0])))
```

```
There are 16162 observations and 21 features in the dataset:
         Of which, 16 numeric features
         And, 4 non-numeric/object features.
         And, 1 non-numeric/non-object features.
         Added 0 feature(s).
         Removed 0 observation(s).
```

In [22]:
```
print("Unique values in each non-numeric column:\t")
for col in df_vers1.select_dtypes(include=[object]):
    print(col,":", df_vers1[col].unique())
```

```
Unique values in each non-numeric column:
Repurchase_Method : ['AUTO RENEW' 'NOTICE' 'PAYMENT PLAN']
Last_Transaction_Channel : ['AUTO RENEW' 'MAIL' 'PHONE' 'BRANCH (PHONE)' 'WEB'
nan 'BRANCH (POS)'
 'IT' 'BILLING']
Number_of_Employees : ['6-10' '11-50' '1-5' nan '51-100' '101-500' '500+']
Language : ['English' nan 'Hindi' 'Italian' 'French' 'Chinese' 'Portuguese' 'R
ussian'
 'Spanish' 'Hebrew' 'Japanese' 'German' 'Polish' 'Arabic' 'Greek'
 'Vietnamese' 'Korean' 'Thai' 'Pashto']
```

In [23]:
```
print("Unique values in each numeric column:\t")
for col in df_vers1.select_dtypes(exclude=[object]):
    print(col,":", df_vers1[col].nunique())
```

```
Unique values in each numeric column:
Customer_Number : 16162
Campaign_Period_Sales : 4316
Historical_Sales_Volume : 16152
Date_of_First_Purchase : 3490
Number_of_Prior_Year_Transactions : 61
Do_Not_Direct_Mail_Solicit : 2
Do_Not_Email : 2
Do_Not_Telemarket : 2
Desk : 2
Executive_Chair : 2
Standard_Chair : 2
Monitor : 2
Printer : 2
Computer : 2
Insurance : 2
Toner : 2
Office_Supplies : 2
```

Impute Missing Values

In [24]:
```
df_vers2 = df_vers1.copy()
```

In [25]:
```
total_obs = df_vers2.isnull().count()
calculate_missing = df_vers2.isnull().sum()
total_missing = calculate_missing[calculate_missing.values>0].sort_values(ascen
ding=False)
perc_missing = round(total_missing/total_obs*100,2)
perc_missing = perc_missing[perc_missing.values>0].sort_values(ascending=False)
col_missing_data = pd.concat([total_missing,perc_missing],axis=1,keys=['#
','%']).sort_values('#',ascending=False)

summary_table = tabulate(col_missing_data, headers=["Features", "No of Obs Miss
ing #", "No of Obs Missing %"])
print('There are {} columns with missing data.'.format(col_missing_data.shape
[0]))
print('These columns are as follows:\n')
print(summary_table)
```

```
There are 9 columns with missing data.
These columns are as follows:

Features                    No of Obs Missing #    No of Obs Missing %
------------------------  ---------------------  ---------------------
Language                                   4467                  27.64
Number_of_Employees                        3744                  23.17
Last_Transaction_Channel                    442                   2.73
Insurance                                     2                   0.01
Toner                                         2                   0.01
Executive_Chair                               1                   0.01
Monitor                                       1                   0.01
Printer                                       1                   0.01
Standard_Chair                                1                   0.01
```

Impute using Mode

In [26]:
```
apply_mode = df_vers2[total_missing.index].mode()
apply_mode
```

Out[26]:

	Language	Number_of_Employees	Last_Transaction_Channel	Toner	Insurance	Printer	Monitor	Standar
0	English	6-10	MAIL	0.000	0.000	0.000	0.000	

In [27]:
```
for col in apply_mode[['Toner','Insurance','Printer','Monitor','Standard_Chair
','Executive_Chair']]:
    df_vers2[col] = df_vers2[col].fillna(apply_mode[col][0])
```

In [28]:
```
df_vers2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16162 entries, 0 to 16171
Data columns (total 21 columns):
Customer_Number                      16162 non-null float64
Campaign_Period_Sales                16162 non-null float64
Historical_Sales_Volume              16162 non-null float64
Date_of_First_Purchase               16162 non-null datetime64[ns]
Number_of_Prior_Year_Transactions    16162 non-null float64
Do_Not_Direct_Mail_Solicit           16162 non-null float64
Do_Not_Email                         16162 non-null float64
Do_Not_Telemarket                    16162 non-null float64
Repurchase_Method                    16162 non-null object
Last_Transaction_Channel             15720 non-null object
Desk                                 16162 non-null float64
Executive_Chair                      16162 non-null float64
Standard_Chair                       16162 non-null float64
Monitor                              16162 non-null float64
Printer                              16162 non-null float64
Computer                             16162 non-null float64
Insurance                            16162 non-null float64
Toner                                16162 non-null float64
Office_Supplies                      16162 non-null float64
Number_of_Employees                  12418 non-null object
Language                             11695 non-null object
dtypes: datetime64[ns](1), float64(16), object(4)
memory usage: 2.7+ MB
```

Impute via IterativeImputer

In [29]:
```
def encode(data):
    df_enc = df.copy()
    encoder = OrdinalEncoder()
    #retains only non-null values
    nonulls = np.array(data.dropna())
    #reshapes the data for encoding
    impute_reshape = nonulls.reshape(-1,1)
    #encode date
    impute_ordinal = encoder.fit_transform(impute_reshape)
    #Assign back encoded values to non-null values
    data.loc[data.notnull()] = np.squeeze(impute_ordinal)
    return data
```

In [30]:
```
df_vers3 = df_vers2.copy()
df_vers3.head()
```

Out[30]:

	Customer_Number	Campaign_Period_Sales	Historical_Sales_Volume	Date_of_First_Purchase	Number_of_
0	86734.000	238.705	146803.429	1968-10-01	
1	97098.000	281.680	439984.160	1981-06-15	
2	100836.000	432.857	970465.714	1962-11-01	
3	116390.000	0.000	230193.600	1983-06-01	
4	127914.000	1370.167	27403.333	1987-07-15	

In [31]:
```
df_vers3.select_dtypes(include='object').columns
```

Out[31]:
```
Index(['Repurchase_Method', 'Last_Transaction_Channel', 'Number_of_Employees',
       'Language'],
      dtype='object')
```

In [32]:
```
for col in df_vers3.select_dtypes(include='object').drop('Repurchase_Method',ax
is=1).columns:
    encode(df_vers3[col])
```

In [33]:
```
df_vers3.head()
```

Out[33]:

	Customer_Number	Campaign_Period_Sales	Historical_Sales_Volume	Date_of_First_Purchase	Number_of_
0	86734.000	238.705	146803.429	1968-10-01	
1	97098.000	281.680	439984.160	1981-06-15	
2	100836.000	432.857	970465.714	1962-11-01	
3	116390.000	0.000	230193.600	1983-06-01	
4	127914.000	1370.167	27403.333	1987-07-15	

In [34]:
```
im_br = IterativeImputer(random_state=24, estimator=ExtraTreesClassifier(n_esti
mators=10))
cat_df_imp = pd.DataFrame(im_br.fit_transform(df_vers3.select_dtypes(include='o
bject').drop('Repurchase_Method',axis=1)),
                          index=df_vers3.index,
                          columns=df_vers3.select_dtypes(include='object').drop
('Repurchase_Method',axis=1).columns)
cat_df_imp.head(10)
```

Out[34]:

	Last_Transaction_Channel	Number_of_Employees	Language
0	0.000	5.000	2.000
1	5.000	2.000	2.000
2	5.000	5.000	2.000
3	6.000	0.000	2.000
4	2.000	5.000	2.000
5	5.000	5.000	2.000
6	5.000	4.000	2.000
7	5.000	4.000	2.000
8	5.000	5.000	2.000
9	5.000	0.000	2.000

In [35]:
```
df_vers3.select_dtypes(include='object').drop('Repurchase_Method',axis=1).head
(10)
```

Out[35]:

	Last_Transaction_Channel	Number_of_Employees	Language
0	0.000	5.000	2.000
1	5.000	2.000	2.000
2	5.000	5.000	2.000
3	6.000	0.000	2.000
4	2.000	NaN	NaN
5	5.000	5.000	2.000
6	5.000	4.000	2.000
7	5.000	4.000	NaN
8	5.000	NaN	NaN
9	5.000	0.000	2.000

In [36]:
```
print("Unique values in each non-numeric column:\t")
for col in df_vers3.select_dtypes(include=[object]).drop('Repurchase_Method',ax
is=1):
    print(col,":", df_vers3[col].unique())
```

```
Unique values in each non-numeric column:
Last_Transaction_Channel : [0.0 5.0 6.0 2.0 7.0 nan 3.0 4.0 1.0]
Number_of_Employees : [5.0 2.0 0.0 nan 4.0 1.0 3.0]
Language : [2.0 nan 7.0 8.0 3.0 1.0 13.0 14.0 15.0 6.0 9.0 4.0 12.0 0.0 5.0 1
7.0 10.0
 16.0 11.0]
```

In [37]:
```
print("Unique values in each non-numeric column:\t")
for col in cat_df_imp:
    print(col,":", cat_df_imp[col].unique())
```

```
Unique values in each non-numeric column:
Last_Transaction_Channel : [0. 5. 6. 2. 7. 3. 4. 1.]
Number_of_Employees : [5. 2. 0. 4. 1. 3.]
Language : [ 2.  7.  8.  3.  1. 13. 14. 15.  6.  9.  4. 12.  0.  5. 17. 10. 1
6. 11.]
```

In [38]:
```
list_of_cats = []
for col in df_vers2.drop('Repurchase_Method',axis=1).dropna().select_dtypes(inc
lude='object').columns:
    uniq_values = df_vers2.drop('Repurchase_Method',axis=1).dropna().select_dty
pes(include='object')[col].unique().tolist()
    try:
        nan_idx = uniq_values.index(np.nan)
        uniq_values.pop(nan_idx)
        list_of_cats.append(uniq_values)
    except ValueError as ve:
        list_of_cats.append(uniq_values)
list_of_cats
```

Out[38]:
```
[['AUTO RENEW',
  'MAIL',
  'PHONE',
  'WEB',
  'BRANCH (POS)',
  'BRANCH (PHONE)',
  'BILLING',
  'IT'],
 ['6-10', '11-50', '1-5', '51-100', '101-500', '500+'],
 ['English',
  'Hindi',
  'Italian',
  'French',
  'Chinese',
  'Portuguese',
  'Russian',
  'Spanish',
  'Hebrew',
  'Japanese',
  'German',
  'Polish',
  'Arabic',
  'Greek',
  'Vietnamese',
  'Korean',
  'Thai',
  'Pashto']]
```

In [39]:
```
def decode_cat_encodings(df_encoded):
    df = pd.DataFrame(df_encoded, columns=df_vers3.drop('Repurchase_Method',axi
s=1).select_dtypes(include='object').columns)
    for col_idx, col in enumerate(df_vers3.drop('Repurchase_Method',axis=1).sel
ect_dtypes(include='object').columns):
        uniq_lst_enc = df[col].unique().tolist()
        level_mapper = {num: cat for num, cat in zip(uniq_lst_enc, list_of_cats
[col_idx])}
        df[col] = df[col].map(level_mapper)
    return df
```

In [40]:
```
imputed_col = decode_cat_encodings(cat_df_imp)
imputed_col.isnull().sum()
```

Out[40]:
```
Last_Transaction_Channel    0
Number_of_Employees         0
Language                    0
dtype: int64
```

In [41]:
```
df_vers4 = pd.concat([df_vers3.drop(['Last_Transaction_Channel','Number_of_Empl
oyees','Language'],axis=1), imputed_col],axis = 1)
```

In [42]:
```
df_vers4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16162 entries, 0 to 16171
Data columns (total 21 columns):
Customer_Number                      16162 non-null float64
Campaign_Period_Sales                16162 non-null float64
Historical_Sales_Volume              16162 non-null float64
Date_of_First_Purchase               16162 non-null datetime64[ns]
Number_of_Prior_Year_Transactions    16162 non-null float64
Do_Not_Direct_Mail_Solicit           16162 non-null float64
Do_Not_Email                         16162 non-null float64
Do_Not_Telemarket                    16162 non-null float64
Repurchase_Method                    16162 non-null object
Desk                                 16162 non-null float64
Executive_Chair                      16162 non-null float64
Standard_Chair                       16162 non-null float64
Monitor                              16162 non-null float64
Printer                              16162 non-null float64
Computer                             16162 non-null float64
Insurance                            16162 non-null float64
Toner                                16162 non-null float64
Office_Supplies                      16162 non-null float64
Last_Transaction_Channel             16162 non-null object
Number_of_Employees                  16162 non-null object
Language                             16162 non-null object
dtypes: datetime64[ns](1), float64(16), object(4)
memory usage: 2.7+ MB
```

In [43]:
```
df_vers4
```

Out[43]:

	Customer_Number	Campaign_Period_Sales	Historical_Sales_Volume	Date_of_First_Purchase	Numbe
0	86734.000	238.705	146803.429	1968-10-01	
1	97098.000	281.680	439984.160	1981-06-15	
2	100836.000	432.857	970465.714	1962-11-01	
3	116390.000	0.000	230193.600	1983-06-01	
4	127914.000	1370.167	27403.333	1987-07-15	
...	...	...	...	...	
16167	166988514.000	0.000	701295.400	1995-12-15	
16168	167014041.000	0.000	2558801.000	1994-06-01	
16169	167077817.000	0.000	2355030.000	1995-01-01	
16170	167089540.000	0.000	584570.000	1996-02-01	
16171	167235933.000	0.000	1949425.333	1998-03-15	

16162 rows × 21 columns

In [44]:
```
#Recheck if all missing data are imputed correctly.
total_obs = df_vers4.isnull().count()
calculate_missing = df_vers4.isnull().sum()
total_missing = calculate_missing[calculate_missing.values>0].sort_values(ascen
ding=False)
perc_missing = round(total_missing/total_obs*100,2)
perc_missing = perc_missing[perc_missing.values>0].sort_values(ascending=False)
col_missing_data = pd.concat([total_missing,perc_missing],axis=1,keys=['#
','%']).sort_values('#',ascending=False)

summary_table = tabulate(col_missing_data, headers=["Features", "No of Obs Miss
ing #", "No of Obs Missing %"])
print('There are {} columns with missing data.'.format(col_missing_data.shape
[0]))
print('These columns are as follows:\n')
print(summary_table)
print('\n')
print('Since there are {} columns with missing data, \
which means we have succesfully imputed with mode accordingly.'.format(col_miss
ing_data.shape[0]))
```

```
There are 0 columns with missing data.
These columns are as follows:

Features    No of Obs Missing #    No of Obs Missing %
----------  ---------------------  ---------------------


Since there are 0 columns with missing data, which means we have succesfully i
mputed with mode accordingly.
```

Simple Features Engineering

In [45]:
```
df_vers4.shape
```

Out[45]: (16162, 21)

Feature 1 - Response

In [46]:
```
df_vers4['Response']=np.where(df_vers4['Campaign_Period_Sales']<=0,0.,1.)
```

Feature 2 - Tenure (in Years)

In [47]:
```
df_vers4['TenureYrs']=((datetime.now().year - pd.Series(df_vers4["Date_of_First
_Purchase"]).apply(lambda x: x.year)) * 12 + \
(datetime.now().month-pd.Series(df_vers4["Date_of_First_Purchase"]).apply(lambd
a x: x.month)))/12
```

Feature 3 - Product Purchase Mix

In [48]:
```
df_vers4['ProductMix'] = df_vers4['Desk']+df_vers4['Executive_Chair']+df_vers4
['Standard_Chair']+\
                         df_vers4['Monitor']+df_vers4['Printer']+df_vers4['Compu
ter']+\
                         df_vers4['Insurance']+df_vers4['Toner']+df_vers4['Offic
e_Supplies']
```

Feature 4 - Contact Channel

In [49]:
```
df_vers4['Last_Transaction_Channel'].unique()
```

Out[49]:
```
array(['AUTO RENEW', 'MAIL', 'PHONE', 'WEB', 'BRANCH (POS)',
       'BRANCH (PHONE)', 'BILLING', 'IT'], dtype=object)
```

In [50]:
```
df_vers4['Contact_Channel'] = np.where(df_vers4['Last_Transaction_Channel']=='AUTO RENEW','eDM',\
                               np.where(df_vers4['Last_Transaction_Channel']=='BILLING','eDM',\
                               np.where(df_vers4['Last_Transaction_Channel']=='MAIL','DM',\
                               np.where(df_vers4['Last_Transaction_Channel']=='PHONE','Telemarket',\
                               np.where(df_vers4['Last_Transaction_Channel']=='WEB','Digital',\
                               np.where(df_vers4['Last_Transaction_Channel']=='None','All',\
                               np.where(df_vers4['Last_Transaction_Channel']=='BRANCH (POS)','DM',\
                               np.where(df_vers4['Last_Transaction_Channel']=='BRANCH (PHONE)','Telemarket','Digital')))))))
```

Feature 5 - Average Sales

In [51]:
```
df_vers4['Avg_Sales_PU'] = df_vers4['Historical_Sales_Volume']/df_vers4['Number_of_Prior_Year_Transactions']
```

In [52]:
```
df_vers4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16162 entries, 0 to 16171
Data columns (total 26 columns):
Customer_Number                      16162 non-null float64
Campaign_Period_Sales                16162 non-null float64
Historical_Sales_Volume              16162 non-null float64
Date_of_First_Purchase               16162 non-null datetime64[ns]
Number_of_Prior_Year_Transactions    16162 non-null float64
Do_Not_Direct_Mail_Solicit           16162 non-null float64
Do_Not_Email                         16162 non-null float64
Do_Not_Telemarket                    16162 non-null float64
Repurchase_Method                    16162 non-null object
Desk                                 16162 non-null float64
Executive_Chair                      16162 non-null float64
Standard_Chair                       16162 non-null float64
Monitor                              16162 non-null float64
Printer                              16162 non-null float64
Computer                             16162 non-null float64
Insurance                            16162 non-null float64
Toner                                16162 non-null float64
Office_Supplies                      16162 non-null float64
Last_Transaction_Channel             16162 non-null object
Number_of_Employees                  16162 non-null object
Language                             16162 non-null object
Response                             16162 non-null float64
TenureYrs                            16162 non-null float64
ProductMix                           16162 non-null float64
Contact_Channel                      16162 non-null object
Avg_Sales_PU                         16162 non-null float64
dtypes: datetime64[ns](1), float64(20), object(5)
memory usage: 3.3+ MB
```

In [53]:
```
df_vers4.shape
```

Out[53]:
```
(16162, 26)
```

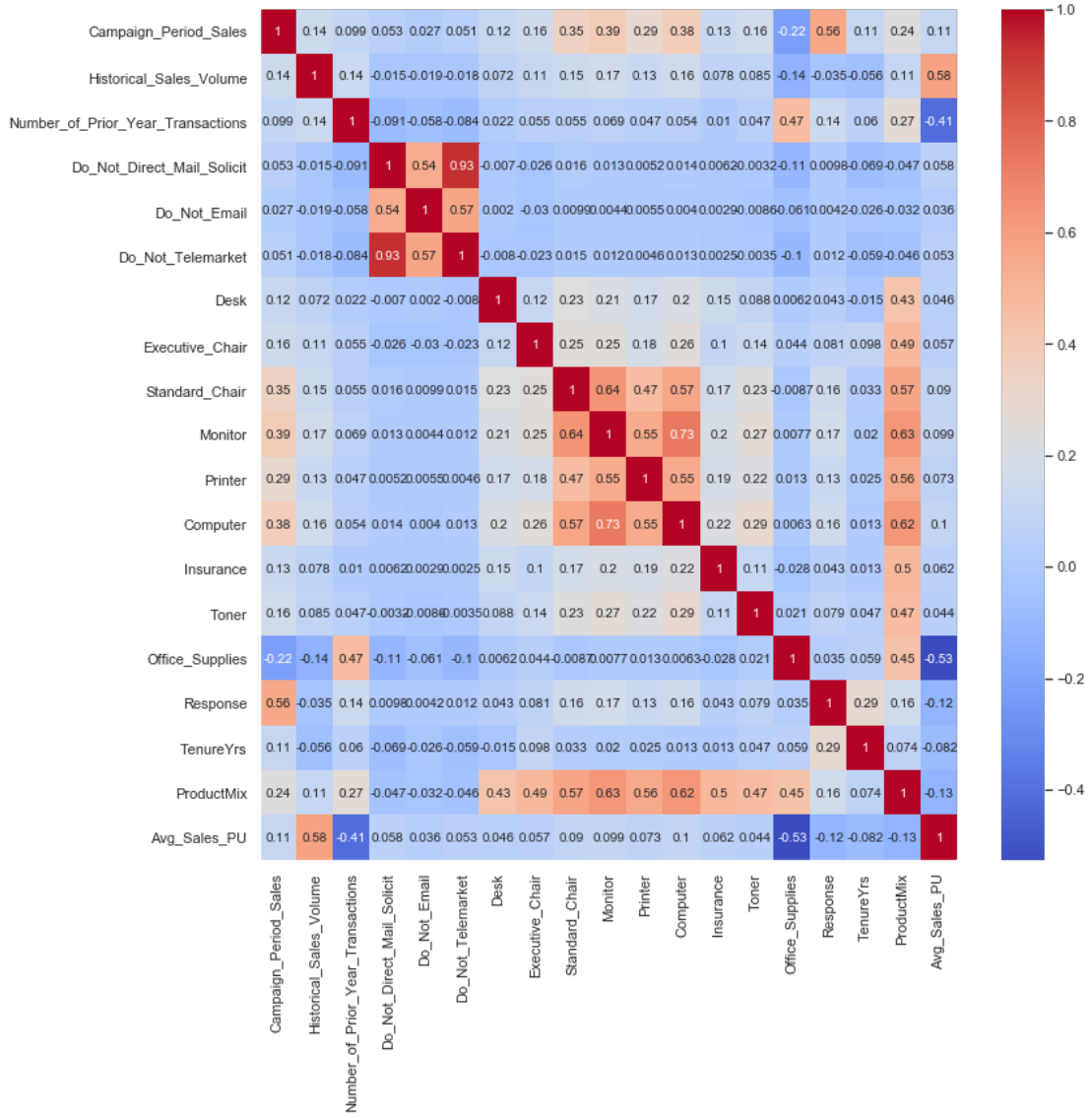
In [54]:
```
print("Unique values in each non-numeric column:\t")
for col in df_vers4.select_dtypes(include=[object]):
    print(col,":", df_vers4[col].unique())
```

```
Unique values in each non-numeric column:
Repurchase_Method : ['AUTO RENEW' 'NOTICE' 'PAYMENT PLAN']
Last_Transaction_Channel : ['AUTO RENEW' 'MAIL' 'PHONE' 'WEB' 'BRANCH (POS)' '
BRANCH (PHONE)'
 'BILLING' 'IT']
Number_of_Employees : ['6-10' '11-50' '1-5' '51-100' '101-500' '500+']
Language : ['English' 'Hindi' 'Italian' 'French' 'Chinese' 'Portuguese' 'Russi
an'
 'Spanish' 'Hebrew' 'Japanese' 'German' 'Polish' 'Arabic' 'Greek'
 'Vietnamese' 'Korean' 'Thai' 'Pashto']
Contact_Channel : ['eDM' 'DM' 'Telemarket' 'Digital']
```

Collinearity Check

In [55]:
```
new_df = df_vers4.copy()
```

In [56]:
```
corr = new_df.drop('Customer_Number',axis=1).corr()
plt.figure(figsize=(12,12))
sns.heatmap(corr, annot=True, cmap='coolwarm')
sns.despine()
```



In [57]:
```
def cramers_v(x, y):
    from scipy.stats import chi2_contingency
    confusion_matrix = pd.crosstab(x,y)
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2/n
    r,k = confusion_matrix.shape
    phi2corr = max(0, phi2-((k-1)*(r-1))/(n-1))
    rcorr = r-((r-1)**2)/(n-1)
    kcorr = k-((k-1)**2)/(n-1)
    return round(np.sqrt(phi2corr/min((kcorr-1),(rcorr-1))),4)
```

In [58]:
```
new_df.columns
```

Out[58]:
```
Index(['Customer_Number', 'Campaign_Period_Sales', 'Historical_Sales_Volume',
       'Date_of_First_Purchase', 'Number_of_Prior_Year_Transactions',
       'Do_Not_Direct_Mail_Solicit', 'Do_Not_Email', 'Do_Not_Telemarket',
       'Repurchase_Method', 'Desk', 'Executive_Chair', 'Standard_Chair',
       'Monitor', 'Printer', 'Computer', 'Insurance', 'Toner',
       'Office_Supplies', 'Last_Transaction_Channel', 'Number_of_Employees',
       'Language', 'Response', 'TenureYrs', 'ProductMix', 'Contact_Channel',
       'Avg_Sales_PU'],
      dtype='object')
```

In [59]:
```
print("Unique values in each non-numeric column:\t")
for col in new_df.select_dtypes(include=[object]):
    print(col,":", new_df[col].unique())
```

```
Unique values in each non-numeric column:
Repurchase_Method : ['AUTO RENEW' 'NOTICE' 'PAYMENT PLAN']
Last_Transaction_Channel : ['AUTO RENEW' 'MAIL' 'PHONE' 'WEB' 'BRANCH (POS)' '
BRANCH (PHONE)'
 'BILLING' 'IT']
Number_of_Employees : ['6-10' '11-50' '1-5' '51-100' '101-500' '500+']
Language : ['English' 'Hindi' 'Italian' 'French' 'Chinese' 'Portuguese' 'Russi
an'
 'Spanish' 'Hebrew' 'Japanese' 'German' 'Polish' 'Arabic' 'Greek'
 'Vietnamese' 'Korean' 'Thai' 'Pashto']
Contact_Channel : ['eDM' 'DM' 'Telemarket' 'Digital']
```

In [60]:
```
group1 = ['Monitor','Printer','Computer','Standard_Chair','ProductMix']
group2 = ['Do_Not_Direct_Mail_Solicit', 'Do_Not_Email', 'Do_Not_Telemarket']
group3 = ['Repurchase_Method', 'Last_Transaction_Channel']
```

In [61]:
```
X_PCA1 = pd.DataFrame(new_df, columns=group1)
X_PCA2 = pd.DataFrame(new_df, columns=group2)
X_PCA3 = pd.DataFrame(new_df, columns=group3)
y_PCA1 = pd.Series(new_df['Response'], name='label')
```

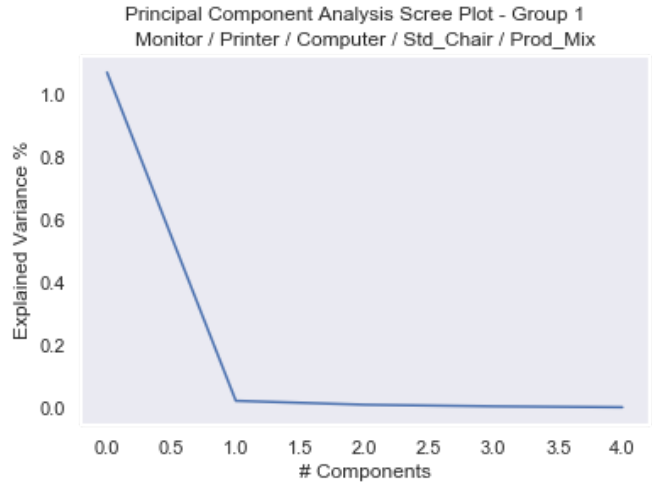
Area 1 | Monitor, Printer, Computer, Standard_Chair, ProductMix

In [62]:
```
new_df[['Desk','Executive_Chair','Standard_Chair','Monitor','Printer','Computer
','Insurance','Toner','Office_Supplies','ProductMix']].corr()
```

Out[62]:

	Desk	Executive_Chair	Standard_Chair	Monitor	Printer	Computer	Insurance	Toner	Offi
Desk	1.000	0.115	0.232	0.208	0.171	0.201	0.147	0.088	
Executive_Chair	0.115	1.000	0.250	0.251	0.178	0.260	0.105	0.136	
Standard_Chair	0.232	0.250	1.000	0.640	0.466	0.570	0.174	0.229	
Monitor	0.208	0.251	0.640	1.000	0.549	0.727	0.198	0.270	
Printer	0.171	0.178	0.466	0.549	1.000	0.550	0.193	0.222	
Computer	0.201	0.260	0.570	0.727	0.550	1.000	0.219	0.295	
Insurance	0.147	0.105	0.174	0.198	0.193	0.219	1.000	0.111	
Toner	0.088	0.136	0.229	0.270	0.222	0.295	0.111	1.000	
Office_Supplies	0.006	0.044	-0.009	0.008	0.013	0.006	-0.028	0.021	
ProductMix	0.432	0.486	0.573	0.625	0.556	0.625	0.495	0.474	

In [63]:
```
#f,axes = plt.subplots(1, 1, figsize=(15,10))
pca1 = PCA(n_components=5).fit(X_PCA1)
#plt.figure(figsize=(15,10))
plt.plot(pca1.explained_variance_)
plt.title('Monitor / Printer / Computer / Std_Chair / Prod_Mix')
plt.suptitle('Principal Component Analysis Scree Plot - Group 1')
plt.xlabel('# Components')
plt.ylabel('Explained Variance %')
plt.grid()
```

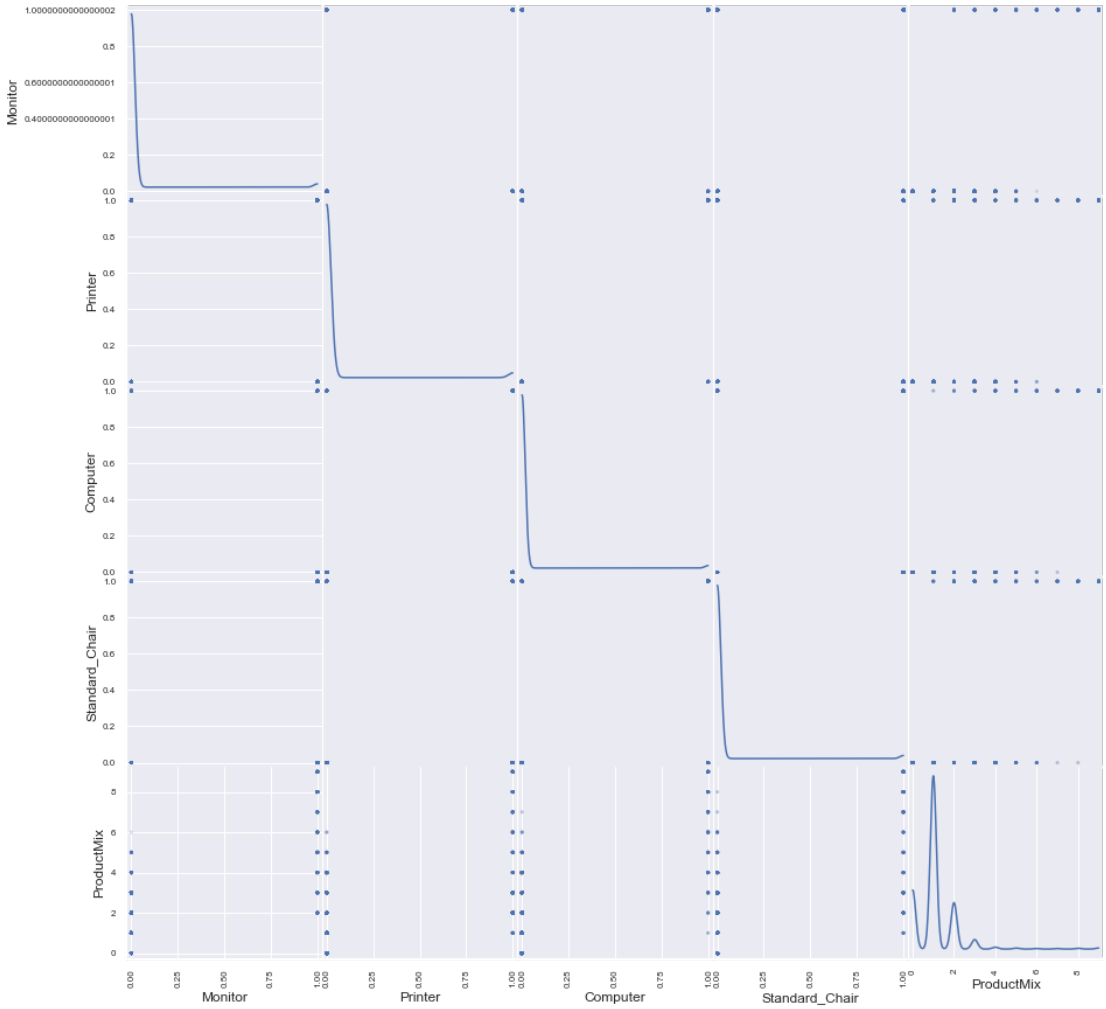


In [64]:
```
new_df[group1].corr()
```

Out[64]:

	Monitor	Printer	Computer	Standard_Chair	ProductMix
Monitor	1.000	0.549	0.727	0.640	0.625
Printer	0.549	1.000	0.550	0.466	0.556
Computer	0.727	0.550	1.000	0.570	0.625
Standard_Chair	0.640	0.466	0.570	1.000	0.573
ProductMix	0.625	0.556	0.625	0.573	1.000

In [65]:
```
feature1 = pd.DataFrame(new_df[group1])
from pandas.plotting import scatter_matrix
scatter_matrix(feature1,alpha=0.2,figsize=(15,15),diagonal='kde')
sns.despine()
```



In [66]:
```
feature_scaled1 = feature1/feature1.std()
pca_scaled1 =PCA(n_components=1).fit_transform(feature_scaled1)
components_scaled1 = PCA(n_components=1).fit(feature_scaled1).components_

print('Dimension of projected feature: {}'.format(pca_scaled1.shape[1]))
print('\nPCA Components:\n1st{}'.format(np.round(components_scaled1[0],4)))
#,np.round(components_scaled1[1],4)))
```

```
Dimension of projected feature: 1

PCA Components:
1st[0.4752 0.4102 0.4655 0.4316 0.4505]
```

In [67]:
```
display(pd.DataFrame(abs(components_scaled1),\
                     index=['PCA_1'], \
                     columns=[group1]))
```

	Monitor	Printer	Computer	Standard_Chair	ProductMix
PCA_1	0.475	0.410	0.466	0.432	0.451

In [68]:
```
response_rows = new_df['Response'] == 1
not_response_rows = new_df['Response'] == 0
```

In [69]:
```
#plt.scatter((pca_scaled1)[response_rows,0]),(pca_scaled1)[response_rows,1])
#plt.scatter((pca_scaled1)[not_response_rows,0]),(pca_scaled1)[not_response_row
s,1])
#plt.xlabel('Principal Component 1')
#plt.ylabel('Principal Component 2')
#plt.suptitle('PCA Scaled: Customers by Response')
#plt.title('PCA #1 Explained 55% & PCA #2 Explained 45%')
#plt.legend(['Response','Not Response'])
#plt.show()
```

In [70]:
```
exp_var_ratio1 = abs(components_scaled1[0])/np.sum(abs(components_scaled1[0]))
#exp_var_ratio2 = abs(components_scaled1[1])/np.sum(abs(components_scaled1[1]))
print(np.round(exp_var_ratio1,3))
#print(np.round(exp_var_ratio2,4))
```

```
[0.213 0.184 0.208 0.193 0.202]
```

In [71]:
```
#exp_var_ratio1 = np.vstack([exp_var_ratio1, exp_var_ratio2])
display(pd.DataFrame(exp_var_ratio1[np.newaxis],\
                     index=['PCA_1'], \
                     columns=[group1]))
```

	Monitor	Printer	Computer	Standard_Chair	ProductMix
PCA_1	0.213	0.184	0.208	0.193	0.202

In [72]:
```
#print('PCA_1: {:.2f}'.format(np.sum(abs(components_scaled1[0]))/(np.sum(abs(co
mponents_scaled1[0]))+np.sum(abs(components_scaled1[1])))))
#print('PCA_2: {:.2f}'.format(np.sum(abs(components_scaled1[1]))/(np.sum(abs(co
mponents_scaled1[0]))+np.sum(abs(components_scaled1[1])))))
```

Area 2 | Do_Not_Direct_Mail_Solicit, Do_Not_Email, Do_Not_Telemarket

In [73]:
```
new_df[['Do_Not_Direct_Mail_Solicit','Do_Not_Email','Do_Not_Telemarket']].corr
()
```

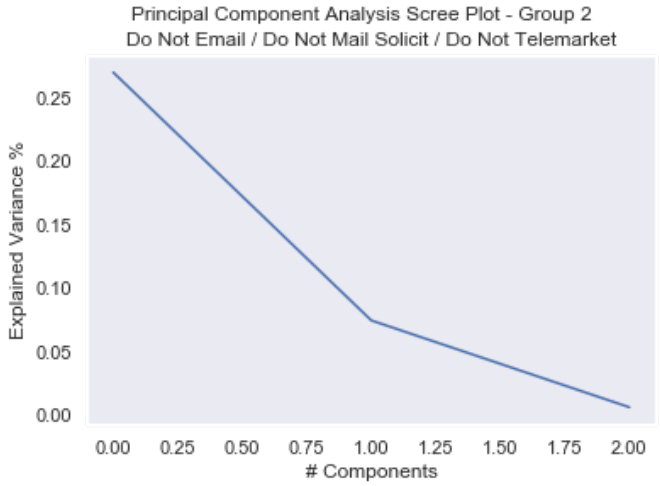
Out[73]:

	Do_Not_Direct_Mail_Solicit	Do_Not_Email	Do_Not_Telemarket
Do_Not_Direct_Mail_Solicit	1.000	0.544	0.933
Do_Not_Email	0.544	1.000	0.572
Do_Not_Telemarket	0.933	0.572	1.000

In [74]:
```
print('Cramers V between Do_Not_Direct_Mail_Solicit and Do_Not_Telemarket :
{}'.format(cramers_v(new_df.Do_Not_Telemarket, new_df.Do_Not_Direct_Mail_Solici
t)))
print('Cramers V between Do_Not_Direct_Mail_Solicit and Do_Not_Email      :
{}'.format(cramers_v(new_df.Do_Not_Email, new_df.Do_Not_Direct_Mail_Solicit)))
print('Cramers V between Do_Not_Telemarket and Do_Not_Email               :
{}'.format(cramers_v(new_df.Do_Not_Email, new_df.Do_Not_Telemarket)))
```

```
Cramers V between Do_Not_Direct_Mail_Solicit and Do_Not_Telemarket : 0.9322
Cramers V between Do_Not_Direct_Mail_Solicit and Do_Not_Email      : 0.5438
Cramers V between Do_Not_Telemarket and Do_Not_Email               : 0.5714
```

In [75]:
```
#f,axes = plt.subplots(1, 1, figsize=(15,10))
pca2 = PCA(n_components=3).fit(X_PCA2)
#plt.figure(figsize=(15,10))
plt.plot(pca2.explained_variance_)
plt.title('Do Not Email / Do Not Mail Solicit / Do Not Telemarket')
plt.suptitle('Principal Component Analysis Scree Plot - Group 2')
plt.xlabel('# Components')
plt.ylabel('Explained Variance %')
plt.grid()
```



In [76]:
```
new_df[group2].corr()
```

Out[76]:

	Do_Not_Direct_Mail_Solicit	Do_Not_Email	Do_Not_Telemarket
Do_Not_Direct_Mail_Solicit	1.000	0.544	0.933
Do_Not_Email	0.544	1.000	0.572
Do_Not_Telemarket	0.933	0.572	1.000

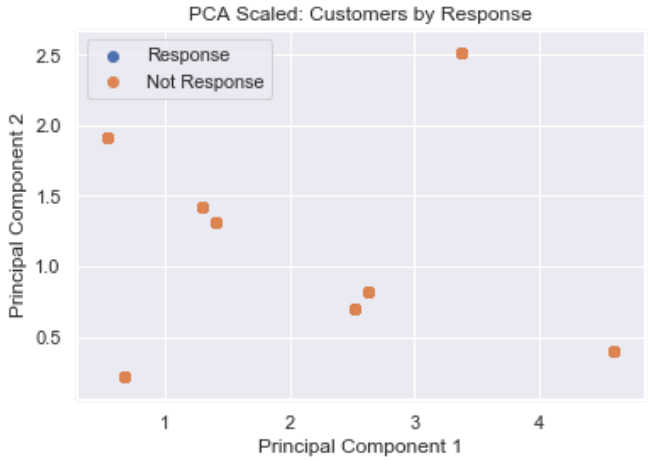
In [77]:
```
feature2 = pd.DataFrame(new_df[group2])
feature_scaled2 = feature2/feature2.std()
pca_scaled2 =PCA(n_components=2).fit_transform(feature_scaled2)
components_scaled2 = PCA(n_components=2).fit(feature_scaled2).components_

print('Dimension of projected feature: {}'.format(pca_scaled2.shape[1]))
print('\nPCA Components:\n1st{}\n 2nd{}'.format(np.round(components_scaled2[0],
4),np.round(components_scaled2[1],4)))
```

```
Dimension of projected feature: 2

PCA Components:
1st[0.6112 0.4956 0.6171]
 2nd[ 0.3751 -0.8679  0.3256]
```

In [78]:
```
display(pd.DataFrame(abs(components_scaled2),\
                 index=['PCA_1','PCA_2'], \
                 columns=[group2]))
```

	Do_Not_Direct_Mail_Solicit	Do_Not_Email	Do_Not_Telemarket
PCA_1	0.611	0.496	0.617
PCA_2	0.375	0.868	0.326

In [79]:
```
plt.scatter(abs(pca_scaled2)[response_rows,0],abs(pca_scaled2)[response_rows,
1])
plt.scatter(abs(pca_scaled2)[not_response_rows,0],abs(pca_scaled2)[not_response
_rows,1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Scaled: Customers by Response')
plt.legend(['Response','Not Response'])
plt.show()
```



In [80]:
```
exp_var_ratio3 = abs(components_scaled2[0])/np.sum(abs(components_scaled2[0]))
exp_var_ratio4 = abs(components_scaled2[1])/np.sum(abs(components_scaled2[1]))
print(np.round(exp_var_ratio3,4))
print(np.round(exp_var_ratio4,4))
```

```
[0.3545 0.2875 0.358 ]
[0.2392 0.5533 0.2075]
```

In [81]:
```
exp_var_ratio3 = np.vstack([exp_var_ratio3, exp_var_ratio4])
display(pd.DataFrame(abs(exp_var_ratio3),\
                 index=['PCA_1','PCA_2'], \
                 columns=[group2]))
```

	Do_Not_Direct_Mail_Solicit	Do_Not_Email	Do_Not_Telemarket
PCA_1	0.355	0.288	0.358
PCA_2	0.239	0.553	0.208

In [82]:
```
print('PCA_1: {:.2f}'.format(np.sum(abs(components_scaled2[0]))/(np.sum(abs(com
ponents_scaled2[0]))+np.sum(abs(components_scaled2[1])))))
print('PCA_2: {:.2f}'.format(np.sum(abs(components_scaled2[1]))/(np.sum(abs(com
ponents_scaled2[0]))+np.sum(abs(components_scaled2[1])))))
```

```
PCA_1: 0.52
PCA_2: 0.48
```

Area 3 | Repurchase Method, Last Trans Channel

In [83]:
```
print(new_df.groupby(['Repurchase_Method','Last_Transaction_Channel']).size())
```

```
Repurchase_Method  Last_Transaction_Channel
AUTO RENEW         AUTO RENEW                 3070
                   BILLING                       1
                   BRANCH (PHONE)               16
                   BRANCH (POS)                524
                   IT                            4
                   MAIL                        169
                   PHONE                       405
                   WEB                          78
NOTICE             AUTO RENEW                   72
                   BILLING                      13
                   BRANCH (PHONE)              617
                   BRANCH (POS)               2123
                   IT                           14
                   MAIL                       8031
                   PHONE                       691
                   WEB                         333
PAYMENT PLAN       MAIL                          1
dtype: int64
```

In [84]:
```
print('Cramers V between RepurchMth and LastTranChannel: {}'.format(cramers_v(n
ew_df.Repurchase_Method, new_df.Last_Transaction_Channel)))
print('Cramers V between CtcChannel and LastTranChannel: {}'.format(cramers_v(n
ew_df.Contact_Channel, new_df.Last_Transaction_Channel)))
```

```
Cramers V between RepurchMth and LastTranChannel: 0.5842
Cramers V between CtcChannel and LastTranChannel: 0.9999
```

Dimensionality Reduction using PCA

In [85]:
```
#Monitor        Printer         Computer        Office_Supplies          Standar
d_Chair         ProdMix
#principalDf1 = pd.DataFrame(data=pca_scaled1,columns=['PCA_1_Prt_Mon_Com_Chr
','PCA_2_OffSupplies'])
principalDf1 = pd.DataFrame(data=pca_scaled1,columns=['PCA_1_Prt_Mon_Com_Chr'])
principalDf1
```

Out[85]:

	PCA_1_Prt_Mon_Com_Chr
0	0.142
1	-0.299
2	-0.299
3	-0.299
4	-0.299
...	...
16157	-0.299
16158	-0.739
16159	-0.739
16160	-0.739
16161	-0.739

16162 rows × 1 columns

In [86]:
```
#Do_Not_Direct_Mail_Solicit      Do_Not_Email     Do_Not_Telemarket
principalDf2 = pd.DataFrame(data=pca_scaled2,columns=['PCA_1_DM_Tele','PCA_2_Email'])
principalDf2
```

Out[86]:

	PCA_1_DM_Tele	PCA_2_Email
0	-0.672	0.212
1	-0.672	0.212
2	-0.672	0.212
3	-0.672	0.212
4	-0.672	0.212
...	...	...
16157	0.542	-1.912
16158	-0.672	0.212
16159	-0.672	0.212
16160	0.542	-1.912
16161	-0.672	0.212

16162 rows × 2 columns

In [87]:
```
final_df = pd.concat([new_df.reset_index(), principalDf1],axis = 1)
final_df = pd.concat([final_df, principalDf2],axis = 1)
```

In [88]:
```
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16162 entries, 0 to 16161
Data columns (total 30 columns):
index                                16162 non-null int64
Customer_Number                      16162 non-null float64
Campaign_Period_Sales                16162 non-null float64
Historical_Sales_Volume              16162 non-null float64
Date_of_First_Purchase               16162 non-null datetime64[ns]
Number_of_Prior_Year_Transactions    16162 non-null float64
Do_Not_Direct_Mail_Solicit           16162 non-null float64
Do_Not_Email                         16162 non-null float64
Do_Not_Telemarket                    16162 non-null float64
Repurchase_Method                    16162 non-null object
Desk                                 16162 non-null float64
Executive_Chair                      16162 non-null float64
Standard_Chair                       16162 non-null float64
Monitor                              16162 non-null float64
Printer                              16162 non-null float64
Computer                             16162 non-null float64
Insurance                            16162 non-null float64
Toner                                16162 non-null float64
Office_Supplies                      16162 non-null float64
Last_Transaction_Channel             16162 non-null object
Number_of_Employees                  16162 non-null object
Language                             16162 non-null object
Response                             16162 non-null float64
TenureYrs                            16162 non-null float64
ProductMix                           16162 non-null float64
Contact_Channel                      16162 non-null object
Avg_Sales_PU                         16162 non-null float64
PCA_1_Prt_Mon_Com_Chr                16162 non-null float64
PCA_1_DM_Tele                        16162 non-null float64
PCA_2_Email                          16162 non-null float64
dtypes: datetime64[ns](1), float64(23), int64(1), object(5)
memory usage: 3.7+ MB
```

In [89]:
```
print("Unique values in each non-numeric column:\t")
for col in final_df.select_dtypes(include=[object]):
    print(col,":", final_df[col].unique())
```

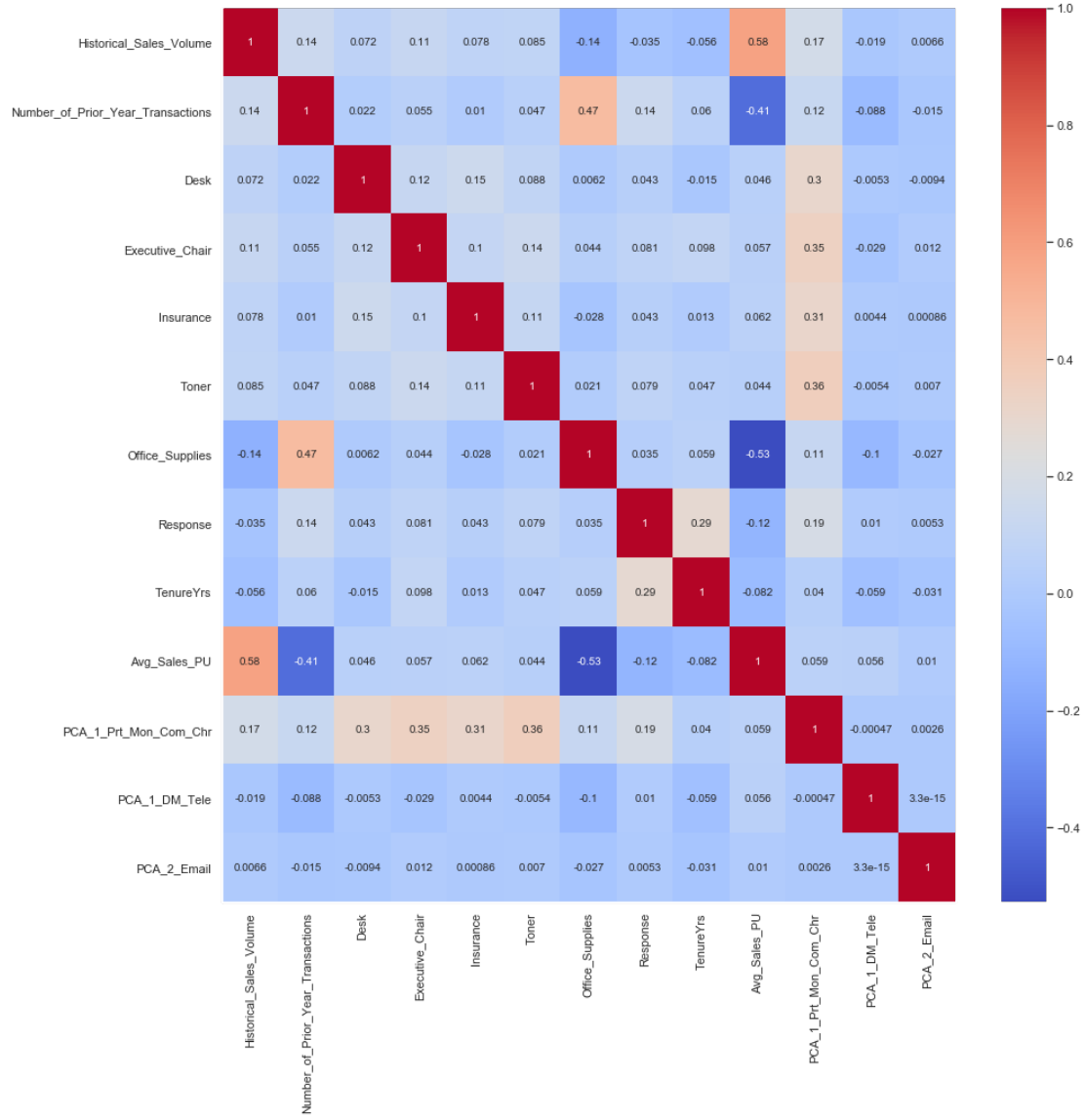
```
Unique values in each non-numeric column:
Repurchase_Method : ['AUTO RENEW' 'NOTICE' 'PAYMENT PLAN']
Last_Transaction_Channel : ['AUTO RENEW' 'MAIL' 'PHONE' 'WEB' 'BRANCH (POS)' '
BRANCH (PHONE)'
 'BILLING' 'IT']
Number_of_Employees : ['6-10' '11-50' '1-5' '51-100' '101-500' '500+']
Language : ['English' 'Hindi' 'Italian' 'French' 'Chinese' 'Portuguese' 'Russi
an'
 'Spanish' 'Hebrew' 'Japanese' 'German' 'Polish' 'Arabic' 'Greek'
 'Vietnamese' 'Korean' 'Thai' 'Pashto']
Contact_Channel : ['eDM' 'DM' 'Telemarket' 'Digital']
```

In [90]:
```
print('There are {} observations and {} features in the dataset:'.format(*final
_df.shape))
print('\t Of which, {} numeric features'.format(final_df.select_dtypes(include=
['int64','float64']).shape[1]))
print('\t And, {} non-numeric/object features.'.format(final_df.select_dtypes(i
nclude='object').shape[1]))
print('\t And, {} non-numeric/non-object features.'.format(final_df.select_dtyp
es(exclude=['int64','float64','object']).shape[1]))
print('\t Add {} new feature(s).'.format(abs(new_df.shape[1] - final_df.shape
[1])))
print('\t Removed {} observation(s).'.format(abs(df_vers4.shape[0] - final_df.s
hape[0])))
```

```
There are 16162 observations and 30 features in the dataset:
         Of which, 24 numeric features
         And, 5 non-numeric/object features.
         And, 1 non-numeric/non-object features.
         Add 4 new feature(s).
         Removed 0 observation(s).
```

In [91]:
```
list_combined = ['Monitor','Printer','Computer','Standard_Chair','ProductMix',\
                 'Do_Not_Direct_Mail_Solicit', 'Do_Not_Email', 'Do_Not_Telemark
et',\
                 'Repurchase_Method', 'Contact_Channel']
```

In [92]:
```
corr1 = final_df.drop(['index','Customer_Number','Campaign_Period_Sales'],axis=
1).drop(list_combined,axis=1).corr()
plt.figure(figsize=(15,15))
sns.heatmap(corr1, annot=True, cmap='coolwarm')
sns.despine()
```



Managing Outliers

Remove Outliers using Z-Score

In [93]:
```
df_vers5 = final_df.copy()
df_vers5.describe()
```

Out[93]:

	index	Customer_Number	Campaign_Period_Sales	Historical_Sales_Volume	Number_of_Prior_Yea
count	16162.000	16162.000	16162.000	16162.000	
mean	8085.151	20703599.809	246.137	672069.449	
std	4669.015	18988771.137	720.460	956782.988	
min	0.000	86734.000	0.000	1319.314	
25%	4041.250	9096278.750	0.000	190286.750	
50%	8085.500	18071963.500	0.000	396597.143	
75%	12128.750	27121957.250	145.787	788705.719	
max	16171.000	167235933.000	8936.850	34412125.800	

In [94]:
```
f = df_vers5[['Historical_Sales_Volume']]
z1 = np.abs(stats.zscore(f))
print(z1)
```

```
[[0.54900879]
 [0.24257587]
 [0.3118842 ]
 ...
 [1.75903276]
 [0.09145455]
 [1.33509418]]
```

In [95]:
```
g = df_vers5[['Number_of_Prior_Year_Transactions']]
z2 = np.abs(stats.zscore(g))
print(z2)
```

```
[[0.06425995]
 [0.93585631]
 [0.56231501]
 ...
 [1.05636394]
 [1.55441901]
 [1.30539147]]
```

In [96]:
```
df_z1_score = df_vers5.copy()
df_z1_score = f[(z1 < 3).all(axis=1)]
df_z1_score.info()
print('Total Observations To Be Removed: {}'.format(df_vers2.shape[0]-df_z1_sco
re.shape[0]))
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15894 entries, 0 to 16161
Data columns (total 1 columns):
Historical_Sales_Volume    15894 non-null float64
dtypes: float64(1)
memory usage: 248.3 KB
Total Observations To Be Removed: 268
```

In [97]:
```
df_z2_score = df_vers5.copy()
df_z2_score =g[(z2 < 3).all(axis=1)]
df_z2_score.info()
print('Total Observations To Be Removed: {}'.format(df_vers2.shape[0]-df_z2_sco
re.shape[0]))
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16059 entries, 0 to 16161
Data columns (total 1 columns):
Number_of_Prior_Year_Transactions    16059 non-null float64
dtypes: float64(1)
memory usage: 250.9 KB
Total Observations To Be Removed: 103
```

In [98]:
```
clean_df1 = df_vers5[df_vers5.index.isin(df_z1_score.index)]
#clean_df2 = clean_df1[clean_df1.index.isin(df_z2_score.index)]
clean_df2 = clean_df1[clean_df1['Number_of_Prior_Year_Transactions']<300]
```

In [99]:
```
print('Total outliers removed from Historical_Sales_Volume           : {}'.form
at(abs(df_vers5.shape[0]-clean_df1.shape[0])))
print('Total outliers removed from Number_of_Prior_Year_Transactions : {}'.form
at(abs(clean_df1.shape[0]-clean_df2.shape[0])))
print('In summary, total outliers removed                            : {}'.form
at(abs(df_vers4.shape[0]-clean_df2.shape[0])))
print('From original {} observations to {}.'.format(df_vers4.shape[0],clean_df
2.shape[0]))
```

```
Total outliers removed from Historical_Sales_Volume           : 268
Total outliers removed from Number_of_Prior_Year_Transactions : 1
In summary, total outliers removed                            : 269
From original 16162 observations to 15893.
```

In [100]:
```
clean_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15893 entries, 0 to 16161
Data columns (total 30 columns):
index                                15893 non-null int64
Customer_Number                      15893 non-null float64
Campaign_Period_Sales                15893 non-null float64
Historical_Sales_Volume              15893 non-null float64
Date_of_First_Purchase               15893 non-null datetime64[ns]
Number_of_Prior_Year_Transactions    15893 non-null float64
Do_Not_Direct_Mail_Solicit           15893 non-null float64
Do_Not_Email                         15893 non-null float64
Do_Not_Telemarket                    15893 non-null float64
Repurchase_Method                    15893 non-null object
Desk                                 15893 non-null float64
Executive_Chair                      15893 non-null float64
Standard_Chair                       15893 non-null float64
Monitor                              15893 non-null float64
Printer                              15893 non-null float64
Computer                             15893 non-null float64
Insurance                            15893 non-null float64
Toner                                15893 non-null float64
Office_Supplies                      15893 non-null float64
Last_Transaction_Channel             15893 non-null object
Number_of_Employees                  15893 non-null object
Language                             15893 non-null object
Response                             15893 non-null float64
TenureYrs                            15893 non-null float64
ProductMix                           15893 non-null float64
Contact_Channel                      15893 non-null object
Avg_Sales_PU                         15893 non-null float64
PCA_1_Prt_Mon_Com_Chr                15893 non-null float64
PCA_1_DM_Tele                        15893 non-null float64
PCA_2_Email                          15893 non-null float64
dtypes: datetime64[ns](1), float64(23), int64(1), object(5)
memory usage: 3.8+ MB
```

In [101]:
```
print('After removing outliers, there are now {} observations and {} feature
s.'.format(*clean_df2.shape))
print('In summary, we have removed {} observations due to outliers.'.format(df_
vers2.shape[0]-clean_df2.shape[0]))
print('Which means, we still have {} of the data retained, which is still reaso
nable.'.format(str(round(len(clean_df2)/len(df_vers2)*100,2))+str('%')))
```

```
After removing outliers, there are now 15893 observations and 30 features.
In summary, we have removed 269 observations due to outliers.
Which means, we still have 98.34% of the data retained, which is still reasona
ble.
```

Manage Outliers using Binning

In [102]:
```
final_df_new = final_df.copy()
```

In [103]:
```
final_df_new.describe()
```

Out[103]:

	index	Customer_Number	Campaign_Period_Sales	Historical_Sales_Volume	Number_of_Prior_Yea
count	16162.000	16162.000	16162.000	16162.000	
mean	8085.151	20703599.809	246.137	672069.449	
std	4669.015	18988771.137	720.460	956782.988	
min	0.000	86734.000	0.000	1319.314	
25%	4041.250	9096278.750	0.000	190286.750	
50%	8085.500	18071963.500	0.000	396597.143	
75%	12128.750	27121957.250	145.787	788705.719	
max	16171.000	167235933.000	8936.850	34412125.800	

In [104]:
```
final_df_new.select_dtypes(include=['int64','float64']).columns
```

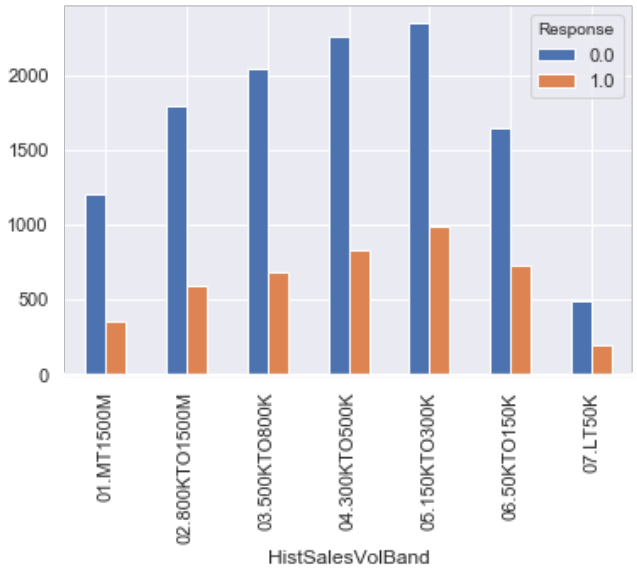
Out[104]:
```
Index(['index', 'Customer_Number', 'Campaign_Period_Sales',
       'Historical_Sales_Volume', 'Number_of_Prior_Year_Transactions',
       'Do_Not_Direct_Mail_Solicit', 'Do_Not_Email', 'Do_Not_Telemarket',
       'Desk', 'Executive_Chair', 'Standard_Chair', 'Monitor', 'Printer',
       'Computer', 'Insurance', 'Toner', 'Office_Supplies', 'Response',
       'TenureYrs', 'ProductMix', 'Avg_Sales_PU', 'PCA_1_Prt_Mon_Com_Chr',
       'PCA_1_DM_Tele', 'PCA_2_Email'],
      dtype='object')
```

In [105]:
```
final_df_new['HistSalesVolBand'] = np.where(final_df_new['Historical_Sales_Volu
me'] >= 1500000,'01.MT1500M',\
                                            np.where(final_df_new['Historical_Sales_V
olume'] >= 800000,'02.800KTO1500M',\
                                            np.where(final_df_new['Historical_Sales_V
olume'] >= 500000,'03.500KTO800K',\
                                            np.where(final_df_new['Historical_Sales_V
olume'] >= 300000,'04.300KTO500K',\
                                            np.where(final_df_new['Historical_Sales_V
olume'] >= 150000,'05.150KTO300K',\
                                            np.where(final_df_new['Historical_Sales_V
olume'] >=  50000,'06.50KTO150K','07.LT50K'))))))
```

In [106]:
```
final_df_new.groupby(['HistSalesVolBand', 'Response']).size()
```

Out[106]:
```
HistSalesVolBand  Response
01.MT1500M        0.000       1205
                  1.000        355
02.800KTO1500M    0.000       1797
                  1.000        589
03.500KTO800K     0.000       2044
                  1.000        686
04.300KTO500K     0.000       2256
                  1.000        833
05.150KTO300K     0.000       2345
                  1.000        985
06.50KTO150K      0.000       1649
                  1.000        730
07.LT50K          0.000        487
                  1.000        201
dtype: int64
```

In [107]:
```
final_df_new.groupby(['HistSalesVolBand', 'Response']).size().unstack(level=1).
plot(kind='bar')
sns.despine(left=True, right=True)
```



In [108]:
```
#final_df_new['Historical_Sales_Bin'] = pd.qcut(final_df_new['Historical_Sales_
Volume'], 8)
#final_df_new.groupby(['Historical_Sales_Bin', 'Response']).size()
```

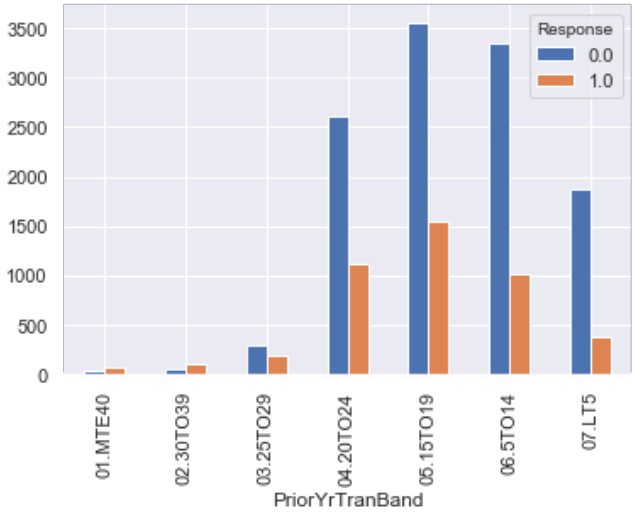
In [109]:
```
#final_df_new.groupby(['Historical_Sales_Bin', 'Response']).size().unstack(leve
l=1).plot(kind='bar')
#sns.despine(left=True, right=True)
```

In [110]:
```
final_df_new['PriorYrTranBand'] = np.where(final_df_new['Number_of_Prior_Year_T
ransactions'] >= 40,'01.MTE40',\
                                           np.where(final_df_new['Number_of_Prior_Ye
ar_Transactions'] >= 30,'02.30TO39',\
                                           np.where(final_df_new['Number_of_Prior_Ye
ar_Transactions'] >= 25,'03.25TO29',\
                                           np.where(final_df_new['Number_of_Prior_Ye
ar_Transactions'] >= 20,'04.20TO24',\
                                           np.where(final_df_new['Number_of_Prior_Ye
ar_Transactions'] >= 15,'05.15TO19',\
                                           np.where(final_df_new['Number_of_Prior_Ye
ar_Transactions'] >= 5,'06.5TO14','07.LT5'))))))
```

In [110]:
```
final_df_new.groupby(['PriorYrTranBand', 'Response']).size()
```

Out[110]:
```
PriorYrTranBand  Response
01.MTE40         0.000          38
                 1.000          59
02.30TO39        0.000          51
                 1.000          94
03.25TO29        0.000         296
                 1.000         182
04.20TO24        0.000        2611
                 1.000        1110
05.15TO19        0.000        3560
                 1.000        1543
06.5TO14         0.000        3356
                 1.000        1019
07.LT5           0.000        1871
                 1.000         372
dtype: int64
```

In [112]:
```
final_df_new.groupby(['PriorYrTranBand', 'Response']).size().unstack(level=1).plot(kind='bar')
sns.despine(left=True, right=True)
```



In [113]:
```
#final_df_new['Num_of_Prior_Trans_Bin'] = pd.qcut(final_df_new['Number_of_Prior_Year_Transactions'], 8)
#final_df_new.groupby(['Num_of_Prior_Trans_Bin', 'Response']).size()
```

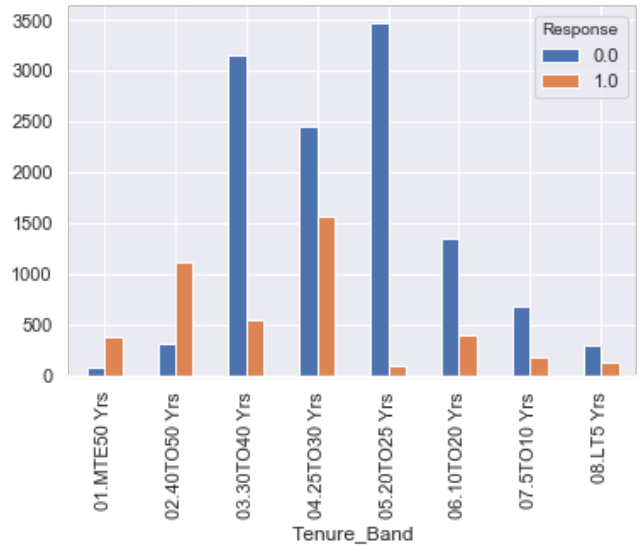
In [114]:
```
#final_df_new.groupby(['Num_of_Prior_Trans_Bin', 'Response']).size().unstack(level=1).plot(kind='bar')
#sns.despine(left=True, right=True)
```

In [115]:
```
final_df_new['Tenure_Band'] = np.where(final_df_new['TenureYrs'] >= 50,'01.MTE50 Yrs',\
                                       np.where(final_df_new['TenureYrs'] >= 40,'02.40TO50 Yrs',\
                                       np.where(final_df_new['TenureYrs'] >= 30,'03.30TO40 Yrs',\
                                       np.where(final_df_new['TenureYrs'] >= 25,'04.25TO30 Yrs',\
                                       np.where(final_df_new['TenureYrs'] >= 20,'05.20TO25 Yrs',\
                                       np.where(final_df_new['TenureYrs'] >= 10,'06.10TO20 Yrs',\
                                       np.where(final_df_new['TenureYrs'] >= 5,'07.5TO10 Yrs','08.LT5 Yrs')))))))
```

In [116]:
```
final_df_new.groupby(['Tenure_Band', 'Response']).size()
```

Out[116]:
```
Tenure_Band    Response
01.MTE50 Yrs   0.000          68
               1.000         382
02.40TO50 Yrs  0.000         305
               1.000        1106
03.30TO40 Yrs  0.000        3154
               1.000         542
04.25TO30 Yrs  0.000        2456
               1.000        1564
05.20TO25 Yrs  0.000        3466
               1.000          94
06.10TO20 Yrs  0.000        1355
               1.000         393
07.5TO10 Yrs   0.000         681
               1.000         177
08.LT5 Yrs     0.000         298
               1.000         121
dtype: int64
```

In [117]:
```
final_df_new.groupby(['Tenure_Band', 'Response']).size().unstack(level=1).plot
(kind='bar')
sns.despine(left=True, right=True)
```



In [118]:
```
#final_df_new['Tenure_Bin'] = pd.qcut(final_df_new['TenureYrs'], 8)
#final_df_new.groupby(['Tenure_Bin', 'Response']).size()
```

In [119]:
```
#final_df_new.groupby(['Tenure_Bin', 'Response']).size().unstack(level=1).plot
(kind='bar')
#sns.despine(left=True, right=True)
```

In [120]:
```
print('Option 1: Remove Outliers: {} observations with {} features'.format(clea
n_df2.shape[0],clean_df2.shape[1]))
print('Option 2: Binning        : {} observations with {} features'.format(fina
l_df_new.shape[0],final_df_new.shape[1]))
```

```
Option 1: Remove Outliers: 15893 observations with 30 features
Option 2: Binning        : 16162 observations with 33 features
```

Drop Uninformative Features

In [122]:
```
# Option 1
model_df = clean_df2.copy()
print('Original Dataset after removed outliers : {} observations with {} featur
es.'.format(model_df.shape[0],model_df.shape[1]),'\n')
list_to_remove = ['Contact_Channel','Repurchase_Method','index','Do_Not_Direct_
Mail_Solicit','Do_Not_Email','Do_Not_Telemarket','Response','Date_of_First_Purc
hase','Customer_Number','Campaign_Period_Sales','Standard_Chair','Monitor','Pri
nter','Computer','ProductMix']
print('Model Dataset if using Option 1:')
print(model_df.drop(list_to_remove, axis=1).info())
```

```
Original Dataset after removed outliers : 15893 observations with 30 features.

Model Dataset if using Option 1:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15893 entries, 0 to 16161
Data columns (total 15 columns):
Historical_Sales_Volume               15893 non-null float64
Number_of_Prior_Year_Transactions     15893 non-null float64
Desk                                  15893 non-null float64
Executive_Chair                       15893 non-null float64
Insurance                             15893 non-null float64
Toner                                 15893 non-null float64
Office_Supplies                       15893 non-null float64
Last_Transaction_Channel              15893 non-null object
Number_of_Employees                   15893 non-null object
Language                              15893 non-null object
TenureYrs                             15893 non-null float64
Avg_Sales_PU                          15893 non-null float64
PCA_1_Prt_Mon_Com_Chr                 15893 non-null float64
PCA_1_DM_Tele                         15893 non-null float64
PCA_2_Email                           15893 non-null float64
dtypes: float64(12), object(3)
memory usage: 1.9+ MB
None
```

In [121]:
```
# Option 2
model_df = final_df_new.copy()
print('Original Dataset if using binnings : {} observations with {} features.'.
format(model_df.shape[0],model_df.shape[1]),'\n')
list_to_remove = ['Historical_Sales_Volume','Number_of_Prior_Year_Transactions
','TenureYrs','Contact_Channel','Repurchase_Method','index','Do_Not_Direct_Mail
_Solicit','Do_Not_Email','Do_Not_Telemarket','Response','Date_of_First_Purchase
','Customer_Number','Campaign_Period_Sales','Standard_Chair','Monitor','Printer
','Computer','ProductMix']
print('Model Dataset if using Option 2:')
print(model_df.drop(list_to_remove, axis=1).info())
```

```
Original Dataset if using binnings : 16162 observations with 32 features.

Model Dataset if using Option 2:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16162 entries, 0 to 16161
Data columns (total 14 columns):
Desk                          16162 non-null float64
Executive_Chair               16162 non-null float64
Insurance                     16162 non-null float64
Toner                         16162 non-null float64
Office_Supplies               16162 non-null float64
Last_Transaction_Channel      16162 non-null object
Number_of_Employees           16162 non-null object
Language                      16162 non-null object
PCA_1_Prt_Mon_Com_Chr         16162 non-null float64
PCA_1_DM_Tele                 16162 non-null float64
PCA_2_Email                   16162 non-null float64
HistSalesVolBand              16162 non-null object
PriorYrTranBand               16162 non-null object
Tenure_Band                   16162 non-null object
dtypes: float64(8), object(6)
memory usage: 1.7+ MB
None
```

In [122]:
```
# Option 3
model_df = clean_df2[['Response','Historical_Sales_Volume','Last_Transaction_Ch
annel','Number_of_Prior_Year_Transactions','Toner','Executive_Chair','Insurance
','Desk','Office_Supplies','PCA_1_Prt_Mon_Com_Chr','PCA_1_DM_Tele','PCA_2_Email
']].copy()
print('Dataset with KSelect Features : {} observations with {} features.'.forma
t(model_df.shape[0],model_df.shape[1]),'\n')
print('Model Dataset if using Option 3:')
print(model_df.info())
```

```
Dataset with KSelect Features : 15893 observations with 12 features.

Model Dataset if using Option 3:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15893 entries, 0 to 16161
Data columns (total 12 columns):
Response                              15893 non-null float64
Historical_Sales_Volume               15893 non-null float64
Last_Transaction_Channel              15893 non-null object
Number_of_Prior_Year_Transactions     15893 non-null float64
Toner                                 15893 non-null float64
Executive_Chair                       15893 non-null float64
Insurance                             15893 non-null float64
Desk                                  15893 non-null float64
Office_Supplies                       15893 non-null float64
PCA_1_Prt_Mon_Com_Chr                 15893 non-null float64
PCA_1_DM_Tele                         15893 non-null float64
PCA_2_Email                           15893 non-null float64
dtypes: float64(11), object(1)
memory usage: 1.6+ MB
None
```

In [123]:
```
for col in model_df.drop(list_to_remove, axis=1).select_dtypes(include=[objec
t]):
    print(col,":\n",model_df[col].unique())
```

```
Last_Transaction_Channel :
 ['AUTO RENEW' 'MAIL' 'PHONE' 'WEB' 'BRANCH (POS)' 'BRANCH (PHONE)'
 'BILLING' 'IT']
Number_of_Employees :
 ['6-10' '11-50' '1-5' '51-100' '101-500' '500+']
Language :
 ['English' 'Hindi' 'Italian' 'French' 'Chinese' 'Portuguese' 'Russian'
 'Spanish' 'Hebrew' 'Japanese' 'German' 'Polish' 'Arabic' 'Greek'
 'Vietnamese' 'Korean' 'Thai' 'Pashto']
```

In [124]:
```
for col in model_df.drop(list_to_remove, axis=1).select_dtypes(include=[objec
t]):
    print(col,":",model_df[col].nunique())
```

```
Last_Transaction_Channel : 8
Number_of_Employees : 6
Language : 18
```

So far, Option 1 provides the best outcome.

Train / Test Split & Cross Validation

In [125]:
```
model_df.shape
```

Out[125]:
```
(15893, 30)
```

In [127]:
```
X = model_df.drop(list_to_remove, axis=1)
y = model_df['Response']
```

In [128]:
```
# Apply train, test split
print(X.shape, y.shape)
print('So in X dataframe, there are {} independent variables and in Y datafram
e, only 1 target variable.'.format(X.shape[1], 1))
print('Also there are {} number of observations.'.format(X.shape[0]))
```

```
(15893, 15) (15893,)
So in X dataframe, there are 15 independent variables and in Y dataframe, only
1 target variable.
Also there are 15893 number of observations.
```

In [129]:
```
X.select_dtypes(exclude='object').info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15893 entries, 0 to 16161
Data columns (total 12 columns):
Historical_Sales_Volume               15893 non-null float64
Number_of_Prior_Year_Transactions     15893 non-null float64
Desk                                  15893 non-null float64
Executive_Chair                       15893 non-null float64
Insurance                             15893 non-null float64
Toner                                 15893 non-null float64
Office_Supplies                       15893 non-null float64
TenureYrs                             15893 non-null float64
Avg_Sales_PU                          15893 non-null float64
PCA_1_Prt_Mon_Com_Chr                 15893 non-null float64
PCA_1_DM_Tele                         15893 non-null float64
PCA_2_Email                           15893 non-null float64
dtypes: float64(12)
memory usage: 1.6 MB
```

In [130]:
```
X.select_dtypes(include='object').info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15893 entries, 0 to 16161
Data columns (total 3 columns):
Last_Transaction_Channel    15893 non-null object
Number_of_Employees         15893 non-null object
Language                    15893 non-null object
dtypes: object(3)
memory usage: 496.7+ KB
```

In [131]:
```
# One-hot encode the data using pandas get_dummies
X_onehot = pd.get_dummies(X)
X_onehot
```

Out[131]:

	Historical_Sales_Volume	Number_of_Prior_Year_Transactions	Desk	Executive_Chair	Insurance	Tone
0	146803.429	15.000	0.000	0.000	1.000	0.00
1	439984.160	22.000	0.000	0.000	0.000	0.00
2	970465.714	19.000	0.000	0.000	0.000	0.00
3	230193.600	17.000	0.000	0.000	0.000	0.00
4	27403.333	21.000	0.000	0.000	1.000	0.00
...	...	...	...	...	...	...
16157	701295.400	3.000	0.000	0.000	1.000	0.00
16158	2558801.000	7.000	0.000	0.000	0.000	0.00
16159	2355030.000	6.000	0.000	0.000	0.000	0.00
16160	584570.000	2.000	0.000	0.000	0.000	0.00
16161	1949425.333	4.000	0.000	0.000	0.000	0.00

15893 rows × 44 columns

In [132]:
```
X_train, X_validation, y_train, y_validation = train_test_split(X_onehot, y, st
ratify=y, test_size = 0.50, random_state = 42)
print('OneHot_Encoded Training examples: {}'.format(X_train.shape[0]))
print('So in X_train dataframe, there are {} independent variables and only 1 t
arget variable.'.format(X_train.shape[1],+1))
print('Also there are {} number of observations.'.format(X_train.shape[0]))
print('==' * 50)
print('OneHot_Encoded Validation examples:  {}'.format(X_validation.shape[0]))
print('So in X_validation dataframe, there are {} independent variables and onl
y 1 target variable.'.format(X_validation.shape[1],+1))
print('Also there are {} number of observations.'.format(X_validation.shape
[0]))
```

```
OneHot_Encoded Training examples: 7946
So in X_train dataframe, there are 44 independent variables and only 1 target
variable.
Also there are 7946 number of observations.
================================================================================
======================
OneHot_Encoded Validation examples:  7947
So in X_validation dataframe, there are 44 independent variables and only 1 ta
rget variable.
Also there are 7947 number of observations.
```

In [133]:
```
def score(confusion_matrix):
    TP = confusion_matrix[0,0]
    FP = confusion_matrix[0,1]
    FN = confusion_matrix[1,0]
    TN = confusion_matrix[1,1]
    accuracy_score = (TP+TN)/(TN+TP+FP+FN)
    recall_score = TP/(TP+FN)
    precision = TP/(TP+FP)
    auc = (recall_score + precision)/2
    F1=(2*precision*recall_score)/(precision+recall_score)
    return accuracy_score, auc, recall_score, precision, F1
```

Model Development & Hyperparams Tuning

In [231]:
```
print('Linear | Logistic Regression:')
logit = LogisticRegression(random_state=24)
# Construct the hyperparameter grid
param_grid = {'penalty':['l1', 'l2', 'elasticnet', 'none'], 'C':np.logspace(0,
4, 10),
              'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
              'l1_ratio' : [0,1,None], 'fit_intercept': [True,False]
             }
# Instantiate the GridSearchCV object using the estimator object,
# the hyperparameter grid, & 5-fold cross-validation
grid_logit = GridSearchCV(logit, param_grid=param_grid, cv=5)
grid_logit.fit(X_train, y_train)
y_pred_logit = grid_logit.predict(X_validation)

cm_logit_train = confusion_matrix(y_train, grid_logit.predict(X_train), labels=
None, sample_weight=None)
print('\nAccuracy Measures on Train:')
print('Confusion_matrix  : \n{}'.format(cm_logit_train))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_train, grid_logit.p
redict(X_train))))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_train, grid_logit.pr
edict(X_train))))
print('Classification Report:')
print(classification_report(y_train, grid_logit.predict(X_train)))

cm_logit = confusion_matrix(y_validation, y_pred_logit, labels=None, sample_wei
ght=None)
print('\nAccuracy Measures on Test:')
print('Confusion_matrix  : \n{}'.format(cm_logit))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_validation, y_pred_
logit)))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_validation, y_pred_l
ogit)))
print('Classification Report:')
print(classification_report(y_validation, y_pred_logit))
print('\n')
print('Optimal hyperparameter(s): {}.'.format(dict(grid_logit.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_logit.best_estimator_))
```

```
Linear | Logistic Regression:

Accuracy Measures on Train:
Confusion_matrix  :
[[5616  176]
 [1503  651]]
Accuracy Score    : 0.789
AUC Score         : 0.636
Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.97      0.87      5792
         1.0       0.79      0.30      0.44      2154

    accuracy                           0.79      7946
   macro avg       0.79      0.64      0.65      7946
weighted avg       0.79      0.79      0.75      7946


Accuracy Measures on Test:
Confusion_matrix  :
[[5645  148]
 [1537  617]]
Accuracy Score    : 0.788
AUC Score         : 0.630
Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.97      0.87      5793
         1.0       0.81      0.29      0.42      2154

    accuracy                           0.79      7947
   macro avg       0.80      0.63      0.65      7947
weighted avg       0.79      0.79      0.75      7947



Optimal hyperparameter(s): {'C': 1.0, 'fit_intercept': True, 'l1_ratio': 0, 'p
enalty': 'l1', 'solver': 'liblinear'}.
Optimal Estimator:
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=0, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l1',
                   random_state=24, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [134]:
```
print('Linear | Logistic Regression:')
#Ridge
cls = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tr
ue,
                   intercept_scaling=1, l1_ratio=0, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l1',
                   random_state=24, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
cls.fit(X_train, y_train)
y_pred_cls = cls.predict(X_validation)

cm_logit_t = confusion_matrix(y_train, cls.predict(X_train), labels=None, sampl
e_weight=None)
cm_logit_test = confusion_matrix(y_validation, y_pred_cls, labels=None, sample_
weight=None)
```

```
Linear | Logistic Regression:
```

In [235]:
```
print('Tree | Decision Tree Classification:')
dec_tree = DecisionTreeClassifier(random_state=24)
# Construct the hyperparameter grid
param_grid = {'criterion': ["gini"],#,"mse"],
              'min_samples_split': [2, 3, 4],
              'max_depth': [4],#list(range(2, 21, 2)),
              'min_samples_leaf': [1, 2, 3],
              #'max_leaf_nodes': [5, 20, 100],
             }
# Instantiate the GridSearchCV object using the estimator object,
# the hyperparameter grid, & 5-fold cross-validation
grid_dec = GridSearchCV(dec_tree, param_grid=param_grid, cv=5)
grid_dec.fit(X_train, y_train)
y_pred_dec_tree = grid_dec.predict(X_validation)

cm_dt_train = confusion_matrix(y_train, grid_dec.predict(X_train), labels=None,
sample_weight=None)
cm_dt_test = confusion_matrix(y_validation, y_pred_dec_tree, labels=None, sampl
e_weight=None)
```

```
Tree | Decision Tree Classification:
```

In [135]:
```
X_encode = X.copy()
for col in X_encode.select_dtypes(include='object').columns:
    encode(X_encode[col])
```

In [136]:
```
X.select_dtypes(include='object').columns
```

Out[136]:
```
Index(['Last_Transaction_Channel', 'Number_of_Employees', 'Language'], dtype='
object')
```

In [137]:
```
def decode(df_encoded):
    df = pd.DataFrame(df_encoded, columns=X.select_dtypes(include='object').col
umns)
    for col_idx, col in enumerate(X[['Last_Transaction_Channel', 'Number_of_Emp
loyees', 'Language']].columns):
        uniq_lst_enc = df[col].unique().tolist()
        level_mapper = {num: cat for num, cat in zip(uniq_lst_enc, list_of_cats
[col_idx])}
        df[col] = df[col].map(level_mapper)
    return df
```

In [138]:
```
X_train_enc, X_validation_enc, y_train_enc, y_validation_enc = train_test_split
(X_encode, y, test_size = 0.50, random_state = 24)
print('Label_Encoded Training examples: {}'.format(X_train_enc.shape[0]))
print('So in X_train dataframe, there are {} independent variables and only 1 t
arget variable.'.format(X_train_enc.shape[1],+1))
print('Also there are {} number of observations.'.format(X_train_enc.shape[0]))
print('==' * 50)
print('Label_Encoded Validation examples:  {}'.format(X_validation_enc.shape
[0]))
print('So in X_validation dataframe, there are {} independent variables and onl
y 1 target variable.'.format(X_validation_enc.shape[1],+1))
print('Also there are {} number of observations.'.format(X_validation_enc.shape
[0]))
```

```
Label_Encoded Training examples: 7946
So in X_train dataframe, there are 15 independent variables and only 1 target
variable.
Also there are 7946 number of observations.
================================================================================
======================
Label_Encoded Validation examples:  7947
So in X_validation dataframe, there are 15 independent variables and only 1 ta
rget variable.
Also there are 7947 number of observations.
```

In [144]:
```
print('Ensemble | Random Forest Classification:')
rf_clf = RandomForestClassifier(
                        bootstrap=True, ccp_alpha=0.0, class_weight=None,
                        criterion='gini', max_depth=7, max_features=10,
                        max_leaf_nodes=None, max_samples=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=2000,
                        n_jobs=None, oob_score=True, random_state=24, verbose=0,
                        warm_start=False)

# Construct the hyperparameter grid
param_grid = {}#'n_estimators':[2000, 3000],
              #'max_features':[5,10,15],
              #'max_depth'   :[3,5,7,8],
              #'criterion'   :['gini', 'mse', 'entropy'],
              #'oob_score'   :[True]
# Instantiate the GridSearchCV object using the estimator object,
# the hyperparameter grid, & 5-fold cross-validation
grid_rf = GridSearchCV(rf_clf, param_grid=param_grid, cv=5)
grid_rf.fit(X_train_enc, y_train_enc)
y_pred_rf = grid_rf.predict(X_validation_enc)
cm_rf_train = confusion_matrix(y_train_enc, grid_rf.predict(X_train_enc), label
s=None, sample_weight=None)
cm_rf_test = confusion_matrix(y_validation_enc, y_pred_rf, labels=None, sample_
weight=None)
```

```
Ensemble | Random Forest Classification:
```

In [915]:
```
print('Ensemble | GradientBoost-Tree Classification:')
gbm = GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=
None,
                           learning_rate=0.01, loss='deviance', max_depth=3,
                           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, n_estimators=3000,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=24, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
#xgb = XGBClassifier(random_state=24)
# Construct the hyperparameter grid
#param_grid =  {'n_estimators': [500, 1000, 2000, 3000, 4000],
#               'max_depth' : [3,4,5, 6, 7, 8],
#               'learning_rate' : [0.01,0.02,0.05,0.10]
#               'obj': ['binary:logistic'] #['reg:linear','reg:squarederror']
#                }
param_grid =   {
               #'n_estimators': [3000],
               #'max_depth' : [3,8],
               #'learning_rate' : [0.01,0.02]
                }
# Instantiate the GridSearchCV object using the estimator object,
# the hyperparameter grid, & 5-fold cross-validation
grid_gbm = GridSearchCV(gbm, param_grid=param_grid, cv=5)
grid_gbm.fit(X_train_enc, y_train_enc)
y_pred_gbm = grid_gbm.predict(X_validation_enc)

cm_gbm_train = confusion_matrix(y_train_enc, grid_gbm.predict(X_train_enc), lab
els=None, sample_weight=None)
cm_gbm_test = confusion_matrix(y_validation_enc, y_pred_gbm, labels=None, sampl
e_weight=None)
```

```
Ensemble | GradientBoost-Tree Classification:
```

In [145]:
```
print('Ensemble | XGBoost Classification:\n')
xgb = XGBClassifier(
              base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.01, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=3000, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=24,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
# Construct the hyperparameter grid
param_grid =  {}#'n_estimators': [1000, 2000, 3000, 4000],
               #'max_depth' : [3,5,7,8],
               #'learning_rate' : [0.01,0.02,0.05]
               #'obj': ['binary:logistic'] #['reg:linear','reg:squarederror']}
# Instantiate the GridSearchCV object using the estimator object,
# the hyperparameter grid, & 10-fold cross-validation
grid_xgb = GridSearchCV(xgb, param_grid=param_grid, cv=10)
grid_xgb.fit(X_train_enc, y_train_enc)
y_pred_xgb = grid_xgb.predict(X_validation_enc)
cm_xgb_train = confusion_matrix(y_train_enc, grid_xgb.predict(X_train_enc), lab
els=None, sample_weight=None)
cm_xgb_test = confusion_matrix(y_validation_enc, y_pred_xgb, labels=None, sampl
e_weight=None)
```

```
Ensemble | XGBoost Classification:
```

In [143]:
```
def create_model(lyrs=[4], act='linear', opt='Adam', dr=0.0):
    # set random seed for reproducibility
    from numpy.random import seed
    seed(42)

    model = models.Sequential()
    # create first hidden layer
    model.add(layers.Dense(lyrs[0], input_dim=X_train.shape[1], activation=ac
t))
    # create additional hidden layers
    for i in range(1,len(lyrs)):
        model.add(layers.Dense(lyrs[i], activation=act))
    # create output layers
    #model.add(layers.Dense(4, activation='sigmoid'))  # output layer
    #model.add(layers.Dense(2, activation='relu'))  # output layer
    model.add(layers.Dense(12, input_dim=8, activation='relu'))
    model.add(layers.Dense(1, activation='sigmoid'))  # output layer
    #model.add(layers.Dense(1, activation='softmax'))  # output layer
    model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy
'])
    return model

model = create_model()
print(model.summary())
```

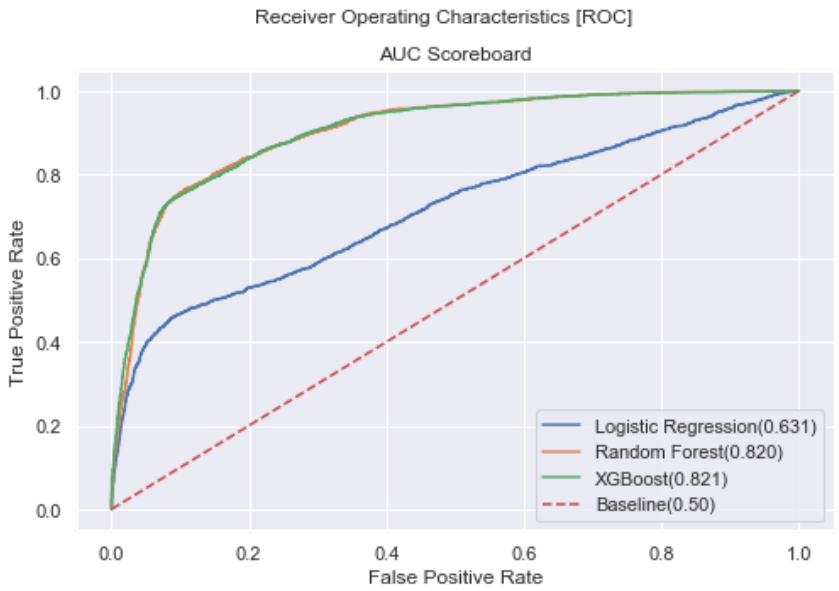
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 4)                 176
_________________________________________________________________
dense_4 (Dense)              (None, 12)                60
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 13
=================================================================
Total params: 249
Trainable params: 249
Non-trainable params: 0
_________________________________________________________________
None
```

In [933]:
```
model = KerasClassifier(build_fn=create_model, verbose=0)
# Construct the hyperparameter grid
batch_size = [16, 32]
epochs = [50]
#optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam
']
#activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'h
ard_sigmoid', 'linear']
param_grid = dict(batch_size=batch_size, epochs=epochs) #optimizer=optimizer, ,
activation=activation
# Instantiate the GridSearchCV object using the estimator object,
# the hyperparameter grid, & 5-fold cross-validation
grid_nn = GridSearchCV(estimator=model,param_grid=param_grid,n_jobs=-1,cv=5,ver
bose=2)# include n_jobs=-1 if you are using CPU
grid_nn.fit(X_train, y_train)
y_pred_neu = grid_nn.predict(X_validation)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  1.7min finished
```

In [385]:
```
print('Receiver Operating Characteristics:\n')
logit_roc_auc = roc_auc_score(y_validation, y_pred_cls)
#dectree_roc_auc = roc_auc_score(y_validation, y_pred_dec_tree)
rand_roc_auc = roc_auc_score(y_validation_enc, y_pred_rf)
#grad_roc_auc = roc_auc_score(y_validation_enc, y_pred_gbm)
#neural_roc_auc = roc_auc_score(y_validation, y_pred_neu)
xgb_roc_auc = roc_auc_score(y_validation_enc, y_pred_xgb)
fpr1, tpr1, thresholds1 = roc_curve(y_validation, cls.predict_proba(X_validatio
n)[:,1])
#fpr2, tpr2, thresholds2 = roc_curve(y_validation, grid_dec.predict_proba(X_val
idation)[:,1])
fpr3, tpr3, thresholds3 = roc_curve(y_validation_enc, grid_rf.predict_proba(X_v
alidation_enc)[:,1])
#fpr4, tpr4, thresholds4 = roc_curve(y_validation_enc, grid_gbm.predict_proba(X
_validation_enc)[:,1])
#fpr5, tpr5, thresholds5 = roc_curve(y_validation, grid_nn.predict_proba(X_vali
dation)[:,1])
fpr6, tpr6, thresholds6 = roc_curve(y_validation_enc, grid_xgb.predict_proba(X_
validation_enc)[:,1])
plt.figure(figsize=(8,5))
plt.plot(fpr1, tpr1, label='Logistic Regression(%0.3f)' % logit_roc_auc)
#plt.plot(fpr2, tpr2, label='Decision Tree(%0.3f)' % dectree_roc_auc)
plt.plot(fpr3, tpr3, label='Random Forest(%0.3f)' % rand_roc_auc)
#plt.plot(fpr4, tpr4, label='GradientBoosted Tree(%0.3f)' % grad_roc_auc)
#plt.plot(fpr5, tpr5, label='Neural Network(%0.3f)' % neural_roc_auc)
plt.plot(fpr6, tpr6, label='XGBoost(%0.3f)' % xgb_roc_auc)
plt.plot([0, 1], [0, 1],'r--',label='Baseline(0.50)')
plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.suptitle('Receiver Operating Characteristics [ROC]')
plt.title('AUC Scoreboard')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```

```
Receiver Operating Characteristics:
```



In [148]:
```
print('Linear | Logistic Regression:')
print('\nAccuracy Measures on Train:')
print('Confusion_matrix  : \n{}'.format(cm_logit_t))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_train, cls.predict
(X_train))))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_train, cls.predict(X
_train))))
print('Log Loss          : {:5.3f}'.format(log_loss(y_train, cls.predict(X_trai
n))))
print('Classification Report:')
print(classification_report(y_train, cls.predict(X_train)))

print('\nAccuracy Measures on Test:')
print('Confusion_matrix  : \n{}'.format(cm_logit_test))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_validation, y_pred_
cls)))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_validation, y_pred_c
ls)))
print('Log Loss          : {:5.3f}'.format(log_loss(y_validation, y_pred_cls)))
print('Classification Report:')
print(classification_report(y_validation, y_pred_cls))
```

```
Linear | Logistic Regression:

Accuracy Measures on Train:
Confusion_matrix  :
[[5618  174]
 [1503  651]]
Accuracy Score    : 0.789
AUC Score         : 0.636
Log Loss          : 7.289
Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.97      0.87      5792
         1.0       0.79      0.30      0.44      2154

    accuracy                           0.79      7946
   macro avg       0.79      0.64      0.65      7946
weighted avg       0.79      0.79      0.75      7946


Accuracy Measures on Test:
Confusion_matrix  :
[[5645  148]
 [1535  619]]
Accuracy Score    : 0.788
AUC Score         : 0.631
Log Loss          : 7.315
Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.97      0.87      5793
         1.0       0.81      0.29      0.42      2154

    accuracy                           0.79      7947
   macro avg       0.80      0.63      0.65      7947
weighted avg       0.79      0.79      0.75      7947
```

In [931]:
```
print('Tree | Decision Tree Classification:')
print('\nAccuracy Measures on Train:')
print('Confusion_matrix  : \n{}'.format(cm_dt_train))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_train, grid_dec.pre
dict(X_train))))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_train, grid_dec.pred
ict(X_train))))
print('Log Loss          : {:5.3f}'.format(log_loss(y_train, grid_dec.predict(X
_train))))
print('Classification Report:')
print(classification_report(y_train, grid_dec.predict(X_train)))

print('\nAccuracy Measures on Test:')
print('Confusion_matrix  : \n{}'.format(cm_dt_test))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_validation, y_pred_
dec_tree)))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_validation, y_pred_d
ec_tree)))
print('Log Loss          : {:5.3f}'.format(log_loss(y_validation, y_pred_dec_tr
ee)))
print('Classification Report:')
print(classification_report(y_validation, y_pred_dec_tree))
print('\n')
print('Optimal hyperparameter(s): {}.'.format(dict(grid_dec.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_dec.best_estimator_))
```

```
Tree | Decision Tree Classification:

Accuracy Measures on Train:
Confusion_matrix  :
[[5089  703]
 [ 558 1596]]
Accuracy Score    : 0.841
AUC Score         : 0.810
Log Loss          : 5.481
Classification Report:
              precision    recall  f1-score   support

         0.0       0.90      0.88      0.89      5792
         1.0       0.69      0.74      0.72      2154

    accuracy                           0.84      7946
   macro avg       0.80      0.81      0.80      7946
weighted avg       0.85      0.84      0.84      7946


Accuracy Measures on Test:
Confusion_matrix  :
[[5051  742]
 [ 578 1576]]
Accuracy Score    : 0.834
AUC Score         : 0.802
Log Loss          : 5.737
Classification Report:
              precision    recall  f1-score   support

         0.0       0.90      0.87      0.88      5793
         1.0       0.68      0.73      0.70      2154

    accuracy                           0.83      7947
   macro avg       0.79      0.80      0.79      7947
weighted avg       0.84      0.83      0.84      7947


Optimal hyperparameter(s): {'criterion': 'gini', 'max_depth': 4, 'min_samples_
leaf': 1, 'min_samples_split': 2}.
Optimal Estimator:
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, 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='deprecated',
                       random_state=24, splitter='best')
```

In [149]:
```
print('Ensemble | Random Forest Classification:')
print('\nAccuracy Measures on Train:')
print('Confusion_matrix  : \n{}'.format(cm_rf_train))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_train_enc, grid_rf.
predict(X_train_enc))))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_train_enc, grid_rf.p
redict(X_train_enc))))
print('Log Loss          : {:5.3f}'.format(log_loss(y_train_enc, grid_rf.predic
t(X_train_enc))))
print('Classification Report:')
print(classification_report(y_train_enc, grid_rf.predict(X_train_enc)))

print('\nAccuracy Measures on Test:')
print('Confusion_matrix  : \n{}'.format(cm_rf_test))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_validation_enc, y_p
red_rf)))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_validation_enc, y_pr
ed_rf)))
print('Log Loss          : {:5.3f}'.format(log_loss(y_validation_enc, y_pred_r
f)))
print('Classification Report:')
print(classification_report(y_validation_enc, y_pred_rf))
print('\n')
print('Optimal hyperparameter(s): {}.'.format(dict(grid_rf.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_rf.best_estimator_))
```

```
Ensemble | Random Forest Classification:

Accuracy Measures on Train:
Confusion_matrix  :
[[5366  379]
 [ 613 1588]]
Accuracy Score    : 0.875
AUC Score         : 0.828
Log Loss          : 4.312
Classification Report:
              precision    recall  f1-score   support

         0.0       0.90      0.93      0.92      5745
         1.0       0.81      0.72      0.76      2201

    accuracy                           0.88      7946
   macro avg       0.85      0.83      0.84      7946
weighted avg       0.87      0.88      0.87      7946


Accuracy Measures on Test:
Confusion_matrix  :
[[5395  445]
 [ 600 1507]]
Accuracy Score    : 0.869
AUC Score         : 0.820
Log Loss          : 4.542
Classification Report:
              precision    recall  f1-score   support

         0.0       0.90      0.92      0.91      5840
         1.0       0.77      0.72      0.74      2107

    accuracy                           0.87      7947
   macro avg       0.84      0.82      0.83      7947
weighted avg       0.87      0.87      0.87      7947



Optimal hyperparameter(s): {}.
Optimal Estimator:
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=7, max_features=10,
                       max_leaf_nodes=None, max_samples=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=2000,
                       n_jobs=None, oob_score=True, random_state=24, verbose=
0,
                       warm_start=False)
```

In [149]:
```
print('Ensemble | GradientBoost-Tree Classification:')
print('\nAccuracy Measures on Train:')
print('Confusion_matrix  : \n{}'.format(cm_gbm_train))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_train_enc, grid_gb
m.predict(X_train_enc))))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_train_enc, grid_gbm.
predict(X_train_enc))))
print('Log Loss          : {:5.3f}'.format(log_loss(y_train_enc, grid_gbm.predi
ct(X_train_enc))))
print('Classification Report:')
print(classification_report(y_train_enc, grid_gbm.predict(X_train_enc)))

print('\nAccuracy Measures on Test:')
print('Confusion_matrix  : \n{}'.format(cm_gbm_test))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_validation_enc, y_p
red_gbm)))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_validation_enc, y_pr
ed_gbm)))
print('Log Loss          : {:5.3f}'.format(log_loss(y_validation_enc, y_pred_gb
m)))
print('Classification Report:')
print(classification_report(y_validation_enc, y_pred_gbm))
print('\n')
print('Optimal hyperparameter(s): {}.'.format(dict(grid_gbm.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_gbm.best_estimator_))
```

```
Ensemble | GradientBoost-Tree Classification:

Accuracy Measures on Train:
Confusion_matrix  :
[[5395  350]
 [ 550 1651]]
Accuracy Score    : 0.887
AUC Score         : 0.845
Log Loss          : 3.912
Classification Report:
              precision    recall  f1-score   support

         0.0       0.91      0.94      0.92      5745
         1.0       0.83      0.75      0.79      2201

    accuracy                           0.89      7946
   macro avg       0.87      0.84      0.85      7946
weighted avg       0.88      0.89      0.89      7946


Accuracy Measures on Test:
Confusion_matrix  :
[[5393  447]
 [ 591 1516]]
Accuracy Score    : 0.869
AUC Score         : 0.821
Log Loss          : 4.511
Classification Report:
              precision    recall  f1-score   support

         0.0       0.90      0.92      0.91      5840
         1.0       0.77      0.72      0.74      2107

    accuracy                           0.87      7947
   macro avg       0.84      0.82      0.83      7947
weighted avg       0.87      0.87      0.87      7947



Optimal hyperparameter(s): {}.
Optimal Estimator:
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.01, loss='deviance', max_depth=3,
                           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, n_estimators=3000,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=24, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [153]:
```
print('Ensemble | XGBoost Classification:')
print('\nAccuracy Measures on Train:')
print('Confusion_matrix  : \n{}'.format(cm_xgb_train))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_train_enc, grid_xg
b.predict(X_train_enc))))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_train_enc, grid_xgb.
predict(X_train_enc))))
print('Log Loss          : {:5.3f}'.format(log_loss(y_train_enc, grid_xgb.predi
ct(X_train_enc))))
print('Classification Report:')
print(classification_report(y_train_enc, grid_xgb.predict(X_train_enc)))

print('\nAccuracy Measures on Test:')
print('Confusion_matrix  : \n{}'.format(cm_xgb_test))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_validation_enc, y_p
red_xgb)))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_validation_enc, y_pr
ed_xgb)))
print('Log Loss          : {:5.3f}'.format(log_loss(y_validation_enc, y_pred_xg
b)))
print('Classification Report:')
print(classification_report(y_validation_enc, y_pred_xgb))
print('\n')
print('Optimal hyperparameter(s): {}.'.format(dict(grid_xgb.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_xgb.best_estimator_))
```

```
Ensemble | XGBoost Classification:

Accuracy Measures on Train:
Confusion_matrix  :
[[5384  361]
 [ 568 1633]]
Accuracy Score    : 0.883
AUC Score         : 0.840
Log Loss          : 4.038
Classification Report:
              precision    recall  f1-score   support

         0.0       0.90      0.94      0.92      5745
         1.0       0.82      0.74      0.78      2201

    accuracy                           0.88      7946
   macro avg       0.86      0.84      0.85      7946
weighted avg       0.88      0.88      0.88      7946


Accuracy Measures on Test:
Confusion_matrix  :
[[5406  434]
 [ 596 1511]]
Accuracy Score    : 0.870
AUC Score         : 0.821
Log Loss          : 4.477
Classification Report:
              precision    recall  f1-score   support

         0.0       0.90      0.93      0.91      5840
         1.0       0.78      0.72      0.75      2107

    accuracy                           0.87      7947
   macro avg       0.84      0.82      0.83      7947
weighted avg       0.87      0.87      0.87      7947



Optimal hyperparameter(s): {}.
Optimal Estimator:
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.01, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=3000, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=24,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [151]:
```
print('NeuralNet | Neural Network:')
cm_nn_train = confusion_matrix(y_train, grid_nn.predict(X_train), labels=None,
sample_weight=None)
print('\nAccuracy Measures on Train:')
print('Confusion_matrix  : \n{}'.format(cm_nn_train))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_train, grid_nn.pred
ict(X_train))))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_train, grid_nn.predi
ct(X_train))))
print('Log Loss          : {:5.3f}'.format(log_loss(y_train, grid_nn.predict(X_
train))))
print('Classification Report:')
print(classification_report(y_train, grid_nn.predict(X_train)))

cm_nn_test = confusion_matrix(y_validation, y_pred_neu, labels=None, sample_wei
ght=None)
print('\nAccuracy Measures on Test:')
print('Confusion_matrix  : \n{}'.format(cm_gbm_test))
print('Accuracy Score    : {:5.3f}'.format(accuracy_score(y_validation, y_pred_
neu)))
print('AUC Score         : {:5.3f}'.format(roc_auc_score(y_validation, y_pred_n
eu)))
print('Log Loss          : {:5.3f}'.format(log_loss(y_validation, y_pred_neu)))
print('Classification Report:')
print(classification_report(y_validation, y_pred_neu))
print('\n')
print('Optimal hyperparameter(s): {}.'.format(dict(grid_nn.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_nn.best_estimator_))
```

```
NeuralNet | Neural Network:

Accuracy Measures on Train:
Confusion_matrix  :
[[4663 1129]
 [1345  809]]
Accuracy Score    : 0.689
AUC Score         : 0.590
Log Loss          : 10.754
Classification Report:
              precision    recall  f1-score   support

         0.0       0.78      0.81      0.79      5792
         1.0       0.42      0.38      0.40      2154

    accuracy                           0.69      7946
   macro avg       0.60      0.59      0.59      7946
weighted avg       0.68      0.69      0.68      7946


Accuracy Measures on Test:
Confusion_matrix  :
[[5393  447]
 [ 591 1516]]
Accuracy Score    : 0.686
AUC Score         : 0.593
Log Loss          : 10.857
Classification Report:
              precision    recall  f1-score   support

         0.0       0.78      0.80      0.79      5793
         1.0       0.42      0.39      0.40      2154

    accuracy                           0.69      7947
   macro avg       0.60      0.59      0.59      7947
weighted avg       0.68      0.69      0.68      7947



Optimal hyperparameter(s): {'batch_size': 16, 'epochs': 50}.
Optimal Estimator:
<keras.wrappers.scikit_learn.KerasClassifier object at 0x1a3c7ae1d0>
```

In [381]:

```
# training classification preds
y_train_pred = grid_xgb.predict_proba(X_train_enc)[:,1]
# possible thresholds
thresholds_train = np.linspace(min(y_train_pred),
                               max(y_train_pred) - 0.1, 100)
# list to hold scores on train and test
train_f1_scores = []
for t in thresholds_train:
    train_f1_scores.append(f1_score(y_train_enc, y_train_pred > t, pos_label=
1))
#find optimal precision score
optimal_t = float(thresholds_train[train_f1_scores==max(train_f1_scores)])
print("Threshold maximising f1 score - {}".format(optimal_t))
```

```
Threshold maximising f1 score - 0.2917735211598226
```

In [356]:
```
print(classification_report(y_train_enc, y_train_pred > optimal_t))
```

```
              precision    recall  f1-score   support

         0.0       0.93      0.91      0.92      5745
         1.0       0.77      0.81      0.79      2201

    accuracy                           0.88      7946
   macro avg       0.85      0.86      0.85      7946
weighted avg       0.88      0.88      0.88      7946
```

Features Selection Techniques

Univariate Selection

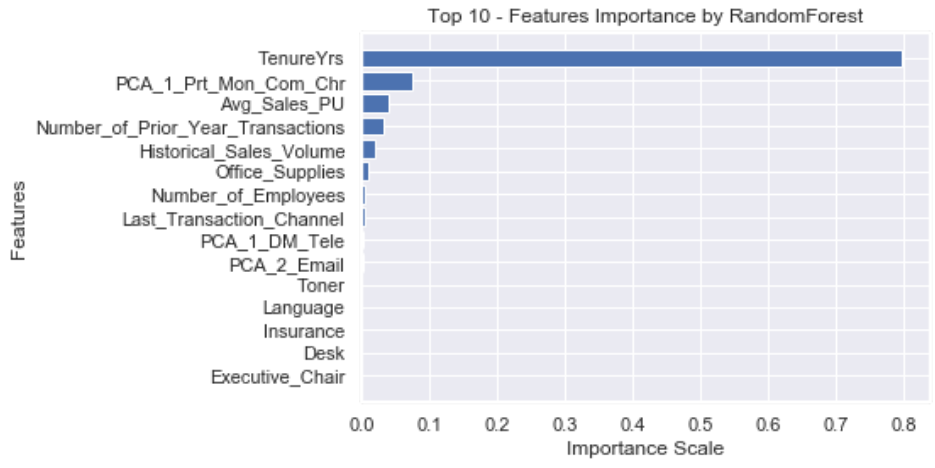
In [ ]:
```
#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X_train.drop(['PCA_1_Prt_Mon_Com_Chr', 'PCA_1_DM_Tele',
'PCA_2_Email'],axis=1),y_train)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Features','Score']  #naming the dataframe columns
print(featureScores.nlargest(15,'Score'))  #print 10 best features
```

Features Importance

In [154]:
```
RF = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                        criterion='gini', max_depth=7, max_features=10,
                        max_leaf_nodes=None, max_samples=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=2000,
                        n_jobs=None, oob_score=True, random_state=24, verbose=0,
                        warm_start=False)

RF.fit(X_train_enc,y_train_enc)
RF_dffeatures = pd.DataFrame(RF.feature_importances_)
RF_dfcolumns = pd.DataFrame(X_train_enc.columns)
RF_featureScores = pd.concat([RF_dfcolumns, RF_dffeatures], axis=1)
RF_featureScores.columns = ['Features','Importance'] # naming the dataframe col
umns
print(RF_featureScores.nlargest(15,'Importance')) # provide top features
```

	Features	Importance
10	TenureYrs	0.796
12	PCA_1_Prt_Mon_Com_Chr	0.076
11	Avg_Sales_PU	0.040
1	Number_of_Prior_Year_Transactions	0.033
0	Historical_Sales_Volume	0.022
6	Office_Supplies	0.010
8	Number_of_Employees	0.006
7	Last_Transaction_Channel	0.005
13	PCA_1_DM_Tele	0.003
14	PCA_2_Email	0.003
5	Toner	0.002
9	Language	0.002
4	Insurance	0.001
2	Desk	0.001
3	Executive_Chair	0.001

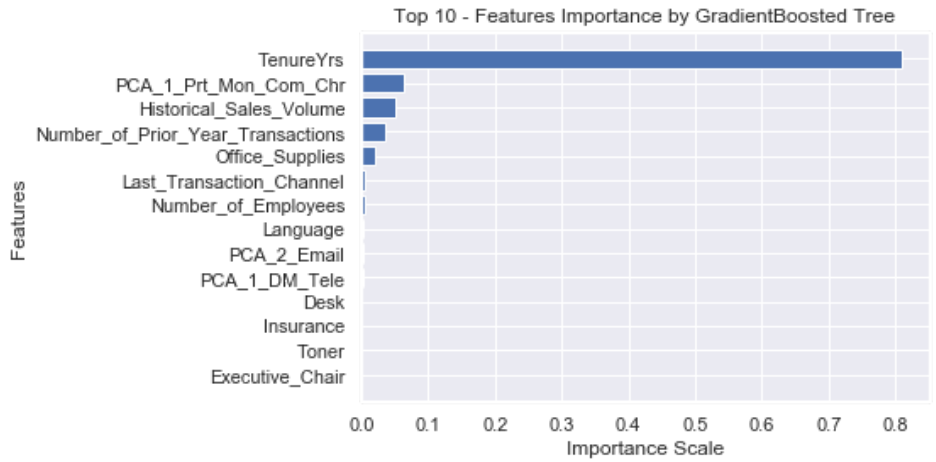
In [155]:
```
plot_RF = RF_featureScores.nlargest(15,'Importance').reset_index().drop('index
',axis=1).sort_values('Importance', ascending=True)
plt.barh(plot_RF['Features'],plot_RF['Importance'])
plt.title("Top 10 - Features Importance by RandomForest")
plt.ylabel("Features")
plt.xlabel("Importance Scale")
plt.show()
```



In [154]:
```
g = GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=No
ne,
                               learning_rate=0.01, loss='deviance', max_depth=3,
                               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, n_estimators=3000,
                               n_iter_no_change=None, presort='deprecated',
                               random_state=24, subsample=1.0, tol=0.0001,
                               validation_fraction=0.1, verbose=0,
                               warm_start=False)

g.fit(X_train_enc,y_train_enc)
g_dffeatures = pd.DataFrame(g.feature_importances_)
g_dfcolumns = pd.DataFrame(X_train_enc.columns)
g_featureScores = pd.concat([g_dfcolumns, g_dffeatures], axis=1)
g_featureScores.columns = ['Features','Importance'] # naming the dataframe colu
mns
print(g_featureScores.nlargest(15,'Importance')) # provide top features
```

```
                            Features  Importance
10                         TenureYrs       0.808
11             PCA_1_Prt_Mon_Com_Chr       0.063
0             Historical_Sales_Volume       0.051
1   Number_of_Prior_Year_Transactions       0.036
6                     Office_Supplies       0.021
7           Last_Transaction_Channel       0.006
8                Number_of_Employees       0.005
9                           Language       0.002
13                       PCA_2_Email       0.002
12                    PCA_1_DM_Tele       0.002
2                             Desk       0.001
4                        Insurance       0.001
5                            Toner       0.001
3                    Executive_Chair       0.000
```

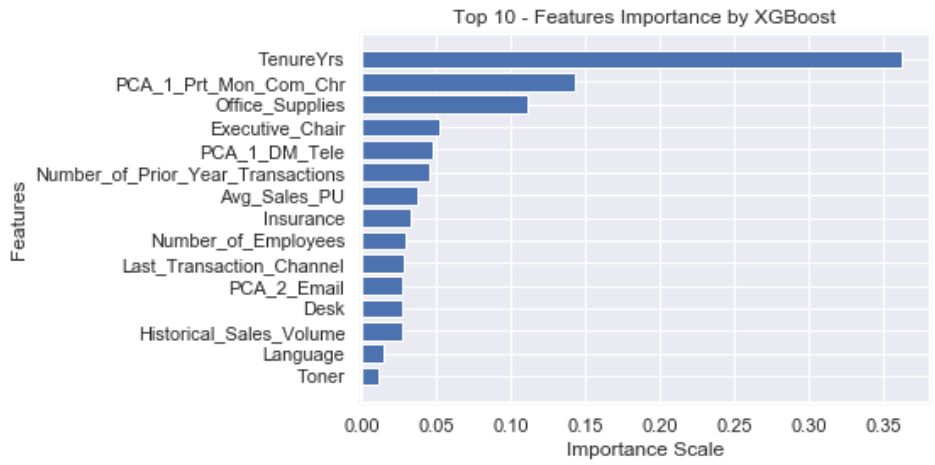
In [155]:
```
plot_g = g_featureScores.nlargest(15,'Importance').reset_index().drop('index',a
xis=1).sort_values('Importance', ascending=True)
plt.barh(plot_g['Features'],plot_g['Importance'])
plt.title("Top 10 - Features Importance by GradientBoosted Tree")
plt.ylabel("Features")
plt.xlabel("Importance Scale")
plt.show()
```



In [156]:
```
xgc = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.01, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=3000, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=24,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
xgc.fit(X_train_enc,y_train_enc)
xgc_dffeatures = pd.DataFrame(xgc.feature_importances_)
xgc_dfcolumns = pd.DataFrame(X_train_enc.columns)
xgc_featureScores = pd.concat([xgc_dfcolumns, xgc_dffeatures], axis=1)
xgc_featureScores.columns = ['Features','Importance'] # naming the dataframe co
lumns
print(xgc_featureScores.nlargest(15,'Importance')) # provide top features
```

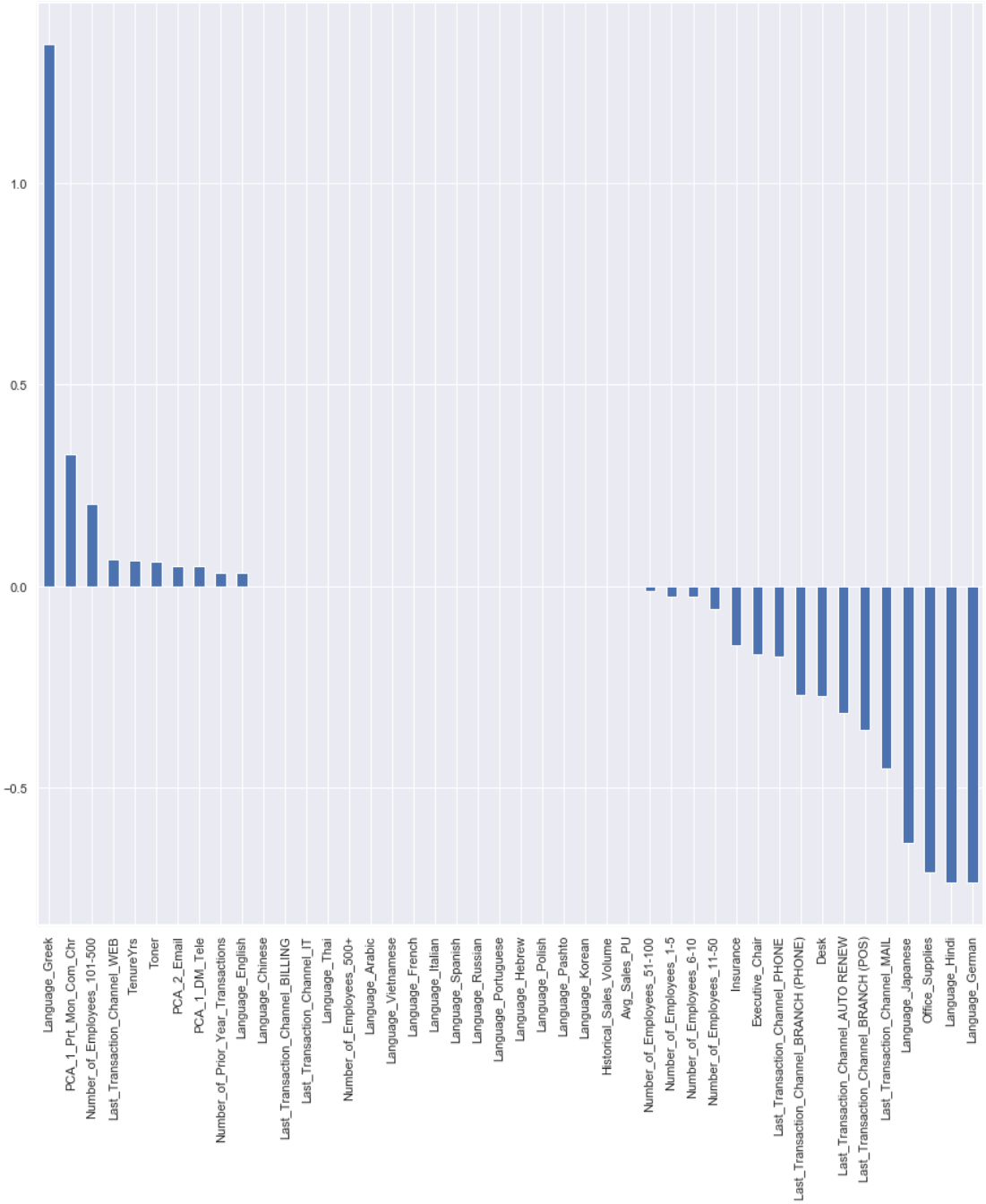
	Features	Importance
10	TenureYrs	0.362
12	PCA_1_Prt_Mon_Com_Chr	0.143
6	Office_Supplies	0.112
3	Executive_Chair	0.052
13	PCA_1_DM_Tele	0.048
1	Number_of_Prior_Year_Transactions	0.045
11	Avg_Sales_PU	0.038
4	Insurance	0.033
8	Number_of_Employees	0.030
7	Last_Transaction_Channel	0.028
14	PCA_2_Email	0.028
2	Desk	0.028
0	Historical_Sales_Volume	0.027
9	Language	0.016
5	Toner	0.011

In [157]:
```
plot_xgc = xgc_featureScores.nlargest(15,'Importance').reset_index().drop('inde
x',axis=1).sort_values('Importance', ascending=True)
plt.barh(plot_xgc['Features'],plot_xgc['Importance'])
plt.title("Top 10 - Features Importance by XGBoost")
plt.ylabel("Features")
plt.xlabel("Importance Scale")
plt.show()
```



In [158]:
```
print('Logistic Regression:\n')
coefs = pd.Series(cls.coef_[0],index=X_train.columns)
coefs = coefs.sort_values(ascending=False)
plt.figure(figsize=(15,15))
plt.subplot(1,1,1)
coefs.plot(kind='bar')
plt.show()
print(coefs.sort_values(ascending=False))
```

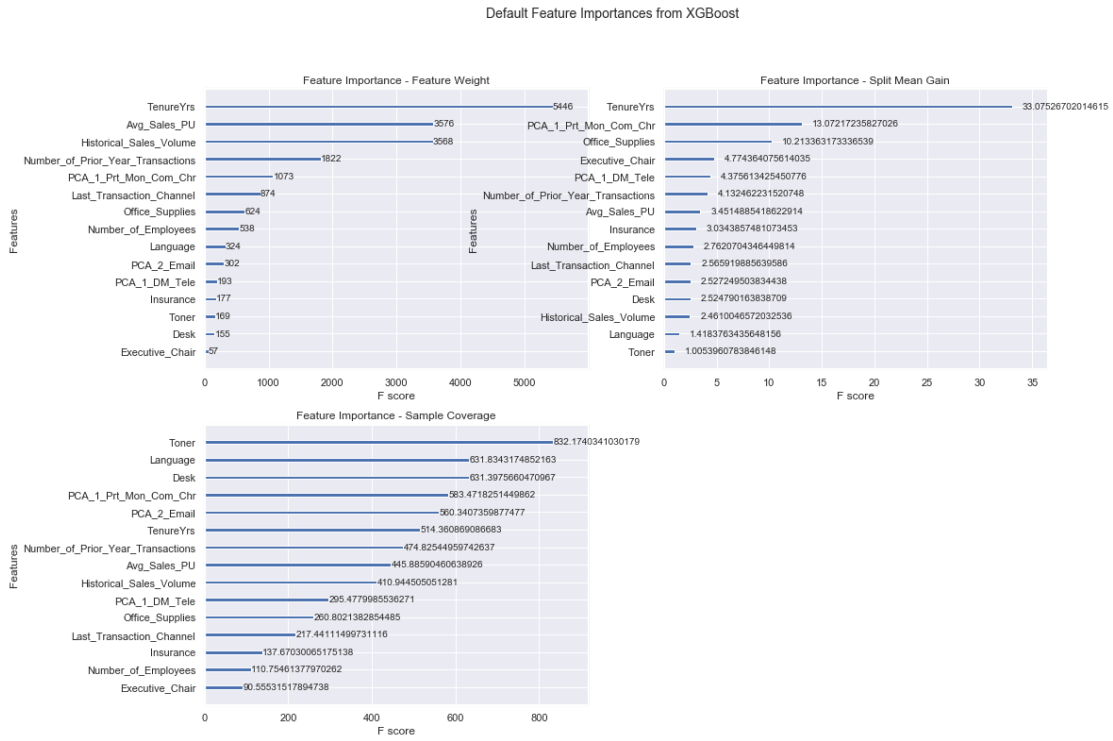
```
Logistic Regression:
```



```
Language_Greek                              1.344
PCA_1_Prt_Mon_Com_Chr                       0.326
Number_of_Employees_101-500                 0.203
Last_Transaction_Channel_WEB                0.066
TenureYrs                                   0.064
Toner                                       0.061
PCA_2_Email                                 0.051
PCA_1_DM_Tele                               0.051
Number_of_Prior_Year_Transactions           0.034
Language_English                            0.032
Language_Russian                            0.000
Language_Chinese                            0.000
Last_Transaction_Channel_BILLING            0.000
Last_Transaction_Channel_IT                 0.000
Language_Thai                               0.000
Number_of_Employees_500+                    0.000
Language_Arabic                             0.000
Language_Vietnamese                         0.000
Language_French                             0.000
Language_Italian                            0.000
Language_Spanish                            0.000
Language_Portuguese                         0.000
Language_Hebrew                             0.000
Language_Polish                             0.000
Language_Pashto                             0.000
Language_Korean                             0.000
Historical_Sales_Volume                    -0.000
Avg_Sales_PU                               -0.000
Number_of_Employees_51-100                 -0.013
Number_of_Employees_1-5                    -0.026
Number_of_Employees_6-10                   -0.027
Number_of_Employees_11-50                  -0.055
Insurance                                  -0.147
Executive_Chair                            -0.170
Last_Transaction_Channel_PHONE             -0.175
Last_Transaction_Channel_BRANCH (PHONE)    -0.269
Desk                                       -0.272
Last_Transaction_Channel_AUTO RENEW        -0.314
Last_Transaction_Channel_BRANCH (POS)      -0.355
Last_Transaction_Channel_MAIL              -0.451
Language_Japanese                          -0.636
Office_Supplies                            -0.710
Language_Hindi                             -0.733
Language_German                            -0.734
dtype: float64
```

SNAP (SHapley Additive exPlanations)

In [159]:
```
import xgboost as xgb
fig = plt.figure(figsize = (16, 12))
title = fig.suptitle("Default Feature Importances from XGBoost", fontsize=14)

ax1 = fig.add_subplot(2,2,1)
xgb.plot_importance(xgc, importance_type='weight', ax=ax1)
t=ax1.set_title("Feature Importance - Feature Weight")

ax2 = fig.add_subplot(2,2,2)
xgb.plot_importance(xgc, importance_type='gain', ax=ax2)
t=ax2.set_title("Feature Importance - Split Mean Gain")

ax3 = fig.add_subplot(2,2,3)
xgb.plot_importance(xgc, importance_type='cover', ax=ax3)
t=ax3.set_title("Feature Importance - Sample Coverage")
```



In [162]:
```
import shap
explainer = shap.TreeExplainer(xgc)
shap_values = explainer.shap_values(X_validation_enc)#.drop(['LastTransactionChannel','NumberofEmployees','Languager'],axis=1))
print('Expected Value:', explainer.expected_value)
pd.DataFrame(shap_values).head()
```

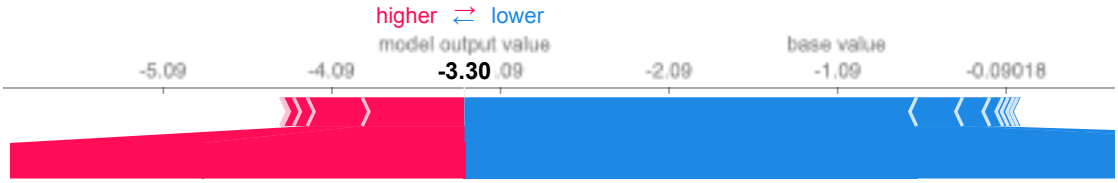
```
Expected Value: -1.0901808
```

Out[162]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.081	-0.272	-0.017	0.000	-0.032	-0.025	0.325	0.065	0.011	0.001	-2.682	0.607	-0.077	-0.032
1	0.037	-0.027	0.232	0.001	0.005	0.000	-0.091	0.041	0.004	0.000	-2.641	0.215	-0.094	-0.029
2	0.099	0.025	0.239	0.001	-0.055	0.001	-0.036	-0.010	0.013	0.010	3.618	0.430	0.039	0.151
3	-0.242	0.050	0.312	0.012	-0.204	-0.013	0.181	-0.017	-0.024	0.010	-2.388	0.142	0.140	-0.025
4	-0.027	-0.002	-0.014	0.001	0.004	-0.004	-0.049	-0.049	-0.002	0.001	-0.258	0.008	-0.069	-0.011

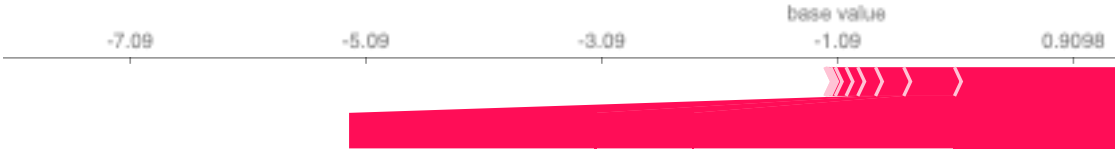
In [164]:
```
shap.initjs()
shap.force_plot(explainer.expected_value,shap_values[0,:], X_validation_enc.iloc[0,:]) #_disp.drop(['LastTransactionChannel','NumberofEmployees','Languager'], axis=1)
```

Out[164]:



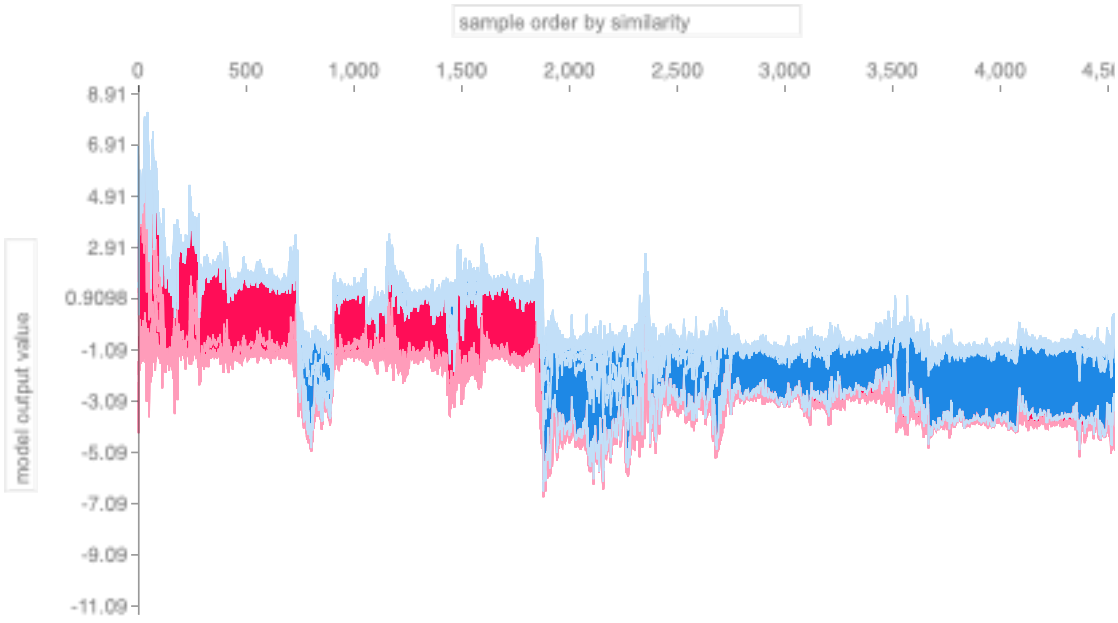
In [165]:
```
shap.force_plot(explainer.expected_value,shap_values[2,:], X_validation_enc.iloc[2,:]) #_disp.drop(['LastTransactionChannel','NumberofEmployees','Languager'], axis=1)
```

Out[165]:

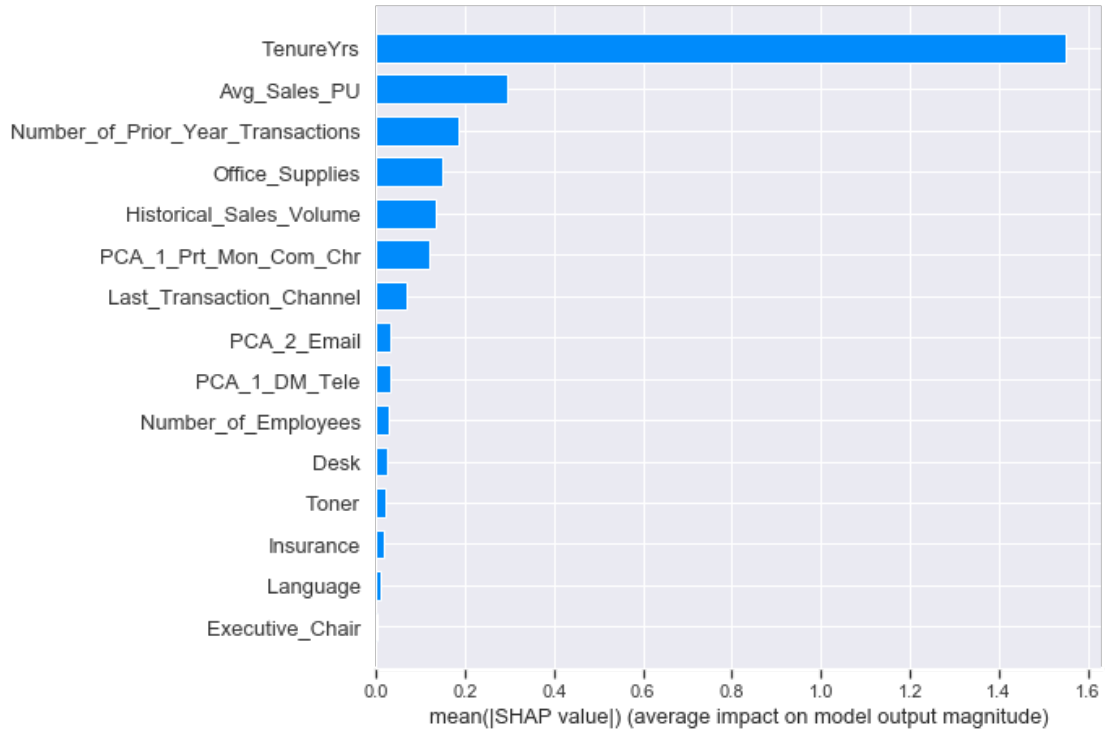


In [166]:
```
shap.force_plot(explainer.expected_value, shap_values[:7000,:], X_validation_enc.iloc[:7000,:])
```

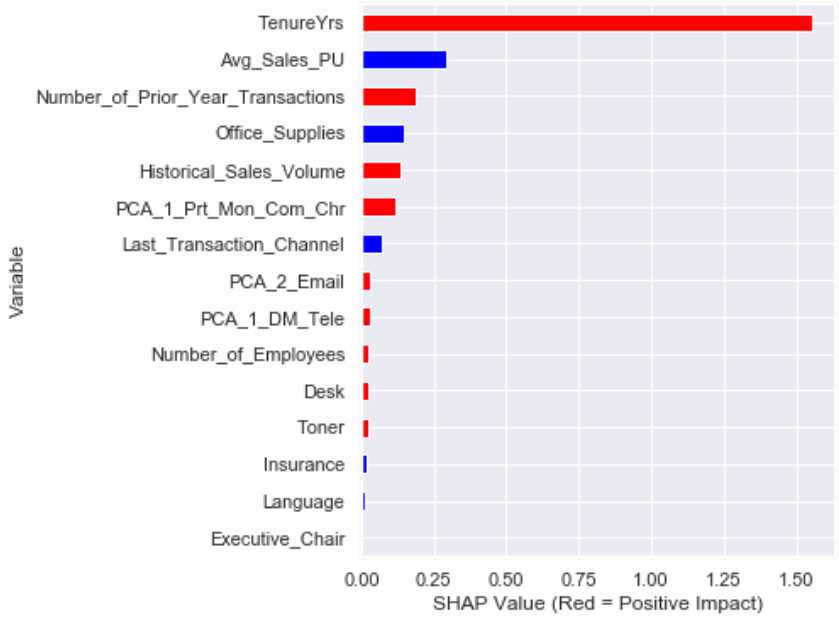
Out[166]:



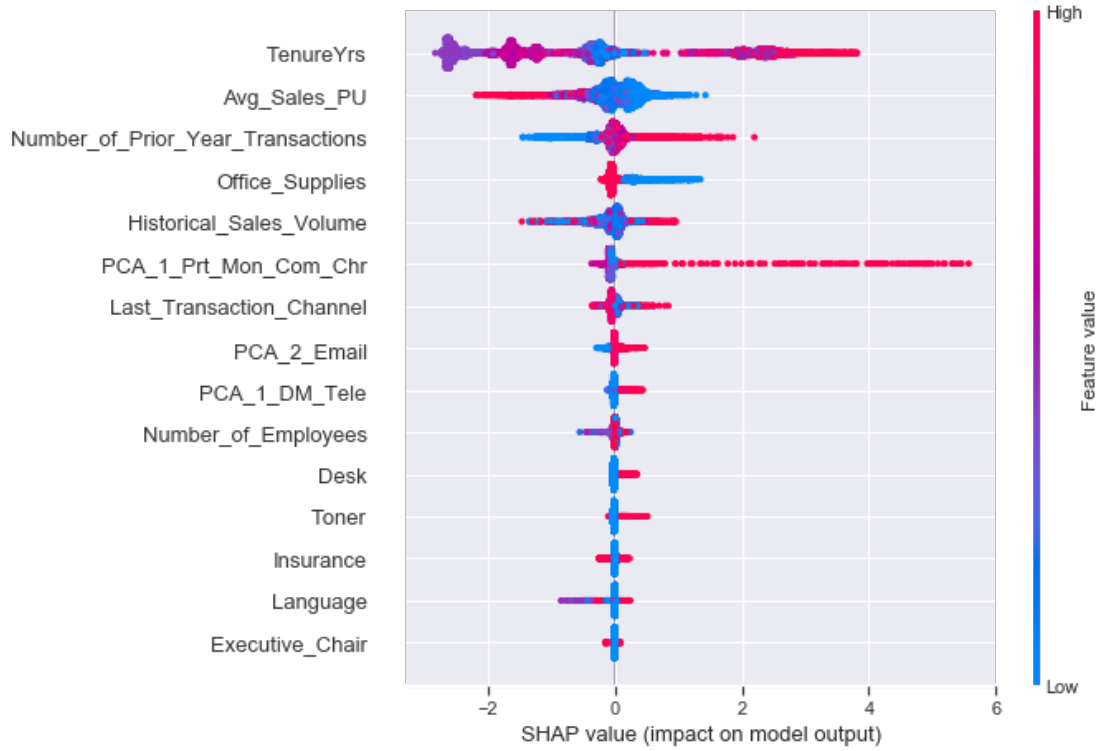
In [167]:
```
shap.summary_plot(shap_values, X_validation_enc, plot_type="bar")
```



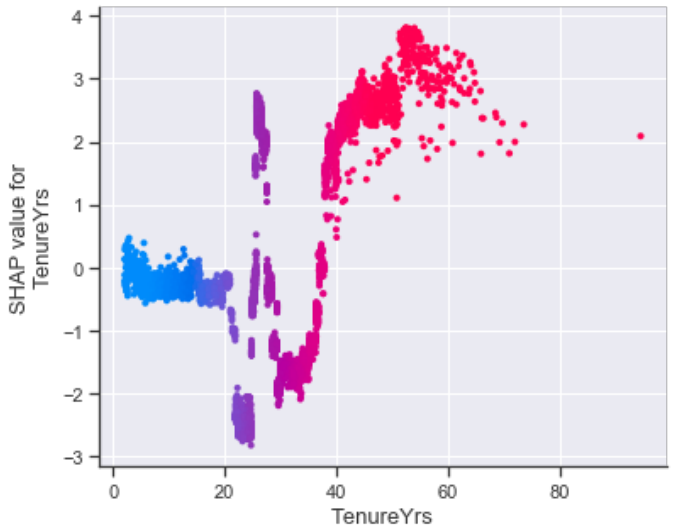
In [168]:
```
def ABS_SHAP(df_shap,df):
    #import matplotlib as plt
    # Make a copy of the input data
    shap_v = pd.DataFrame(df_shap)
    feature_list = df.columns
    shap_v.columns = feature_list
    df_v = df.copy().reset_index().drop('index',axis=1)

    # Determine the correlation in order to plot with different colors
    corr_list = list()
    for i in feature_list:
        b = np.corrcoef(shap_v[i],df_v[i])[1][0]
        corr_list.append(b)
    corr_df = pd.concat([pd.Series(feature_list),pd.Series(corr_list)],axis=1).
fillna(0)
    # Make a data frame. Column 1 is the feature, and Column 2 is the correlati
on coefficient
    corr_df.columns  = ['Variable','Corr']
    corr_df['Sign'] = np.where(corr_df['Corr']>0,'red','blue')

    # Plot it
    shap_abs = np.abs(shap_v)
    k=pd.DataFrame(shap_abs.mean()).reset_index()
    k.columns = ['Variable','SHAP_abs']
    k2 = k.merge(corr_df,left_on = 'Variable',right_on='Variable',how='inner')
    k2 = k2.sort_values(by='SHAP_abs',ascending = True)
    colorlist = k2['Sign']
    ax = k2.plot.barh(x='Variable',y='SHAP_abs',color = colorlist, figsize=(5,
6),legend=False)
    ax.set_xlabel("SHAP Value (Red = Positive Impact)")

ABS_SHAP(shap_values,X_validation_enc)
```



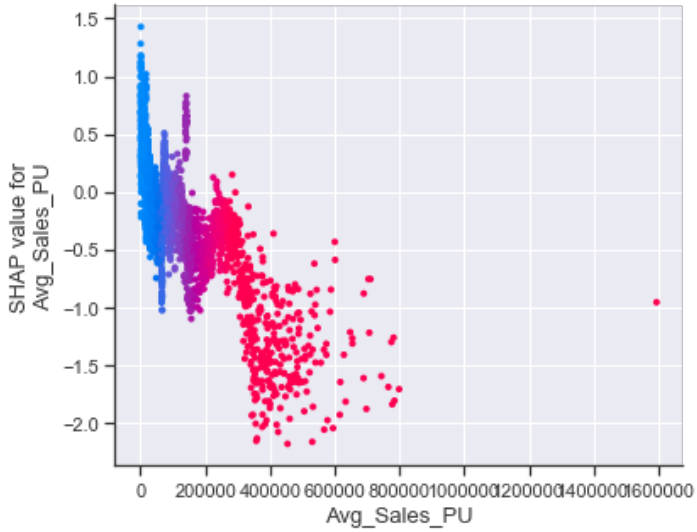
In [169]:
```
shap.summary_plot(shap_values, X_validation_enc)
```



In [171]:
```
shap.dependence_plot(ind='TenureYrs', interaction_index='TenureYrs',
                     shap_values=shap_values,
                     features=X_validation_enc,
                     display_features=X_validation_enc)
```

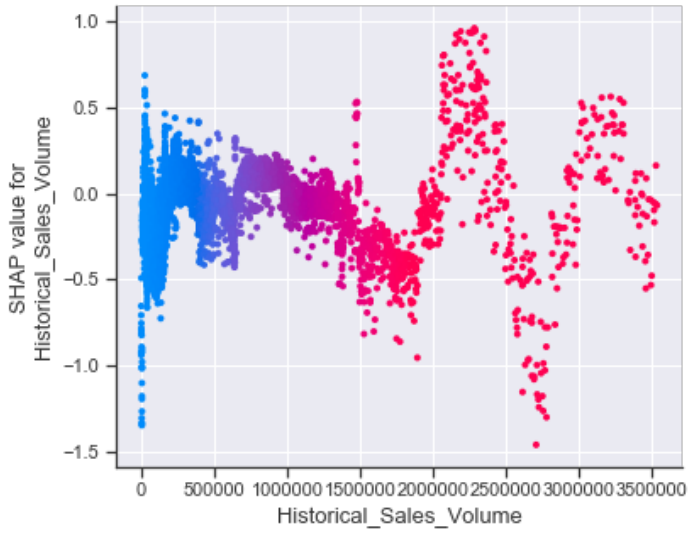


In [173]:
```
shap.dependence_plot(ind='Avg_Sales_PU', interaction_index='Avg_Sales_PU',
                     shap_values=shap_values,
                     features=X_validation_enc,
                     display_features=X_validation_enc)
```

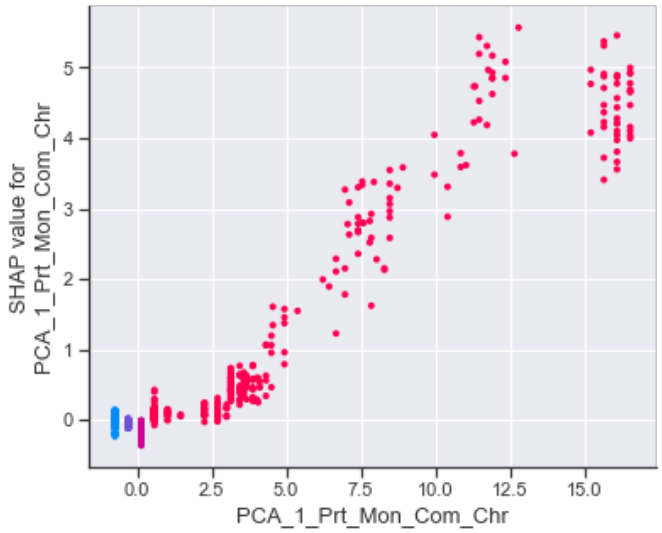


In [172]:
```
shap.dependence_plot(ind='Number_of_Prior_Year_Transactions', interaction_index
='Number_of_Prior_Year_Transactions',
                     shap_values=shap_values,
                     features=X_validation_enc,
                     display_features=X_validation_enc)
```

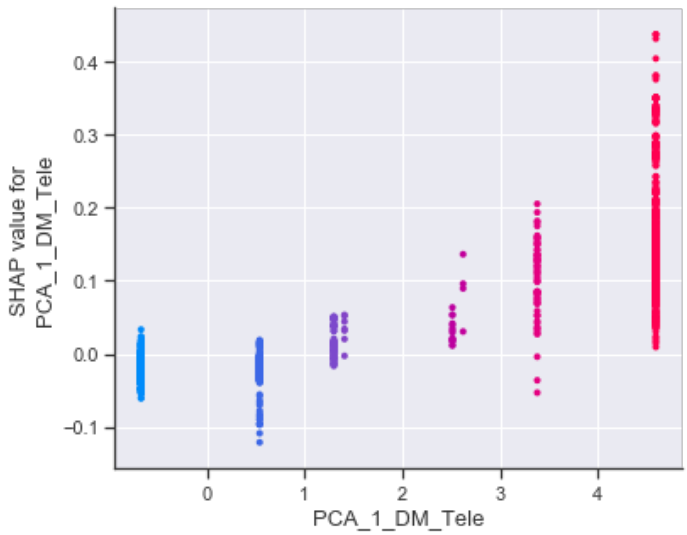
In [175]:
```
shap.dependence_plot(ind='Historical_Sales_Volume', interaction_index='Historic
al_Sales_Volume',
                     shap_values=shap_values,
                     features=X_validation_enc,
                     display_features=X_validation_enc)
```



In [176]:
```
shap.dependence_plot(ind='PCA_1_Prt_Mon_Com_Chr', interaction_index='PCA_1_Prt_
Mon_Com_Chr',
                     shap_values=shap_values,
                     features=X_validation_enc,
                     display_features=X_validation_enc)
```



In [181]:
```
shap.dependence_plot(ind='PCA_1_DM_Tele', interaction_index='PCA_1_DM_Tele',
                     shap_values=shap_values,
                     features=X_validation_enc,
                     display_features=X_validation_enc)
```



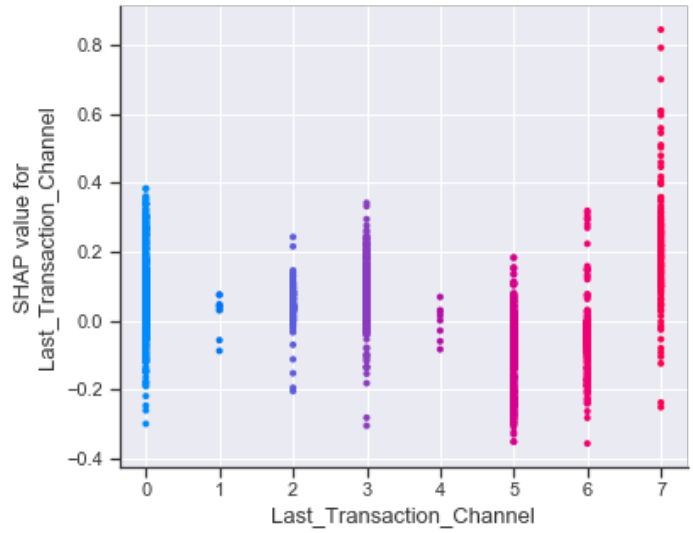
In [182]:
```
a = X_validation_enc
b = decode(X_validation_enc)
b.rename(columns=lambda col: col.replace('_', ''), inplace=True)
b.rename(columns={'Language': 'Languager'}, inplace=True)
b.columns
```

Out[182]:
```
Index(['LastTransactionChannel', 'NumberofEmployees', 'Languager'], dtype='obj
ect')
```

In [183]:
```
X_validation_disp = pd.concat([a, b],axis = 1)
X_validation_disp.columns
```

Out[183]:
```
Index(['Historical_Sales_Volume', 'Number_of_Prior_Year_Transactions', 'Desk',
       'Executive_Chair', 'Insurance', 'Toner', 'Office_Supplies',
       'Last_Transaction_Channel', 'Number_of_Employees', 'Language',
       'TenureYrs', 'Avg_Sales_PU', 'PCA_1_Prt_Mon_Com_Chr', 'PCA_1_DM_Tele',
       'PCA_2_Email', 'LastTransactionChannel', 'NumberofEmployees',
       'Languager'],
      dtype='object')
```

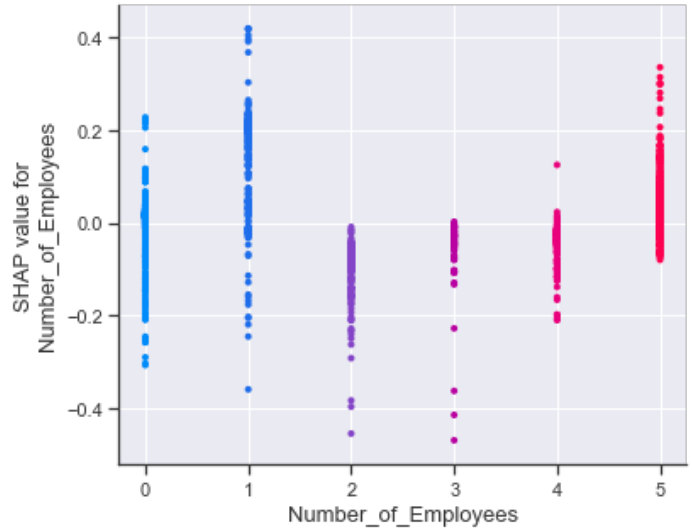
In [184]:
```
shap.dependence_plot(ind='Last_Transaction_Channel', interaction_index='Last_Transaction_Channel',
                     shap_values=shap_values,
                     features=X_validation_enc,
                     display_features=X_validation_enc)
```



In [185]:
```
print(X_validation_disp.groupby(['Last_Transaction_Channel','LastTransactionChannel']).size())
```

```
Last_Transaction_Channel  LastTransactionChannel
0                         MAIL                    1514
1                         BILLING                    9
2                         WEB                      313
3                         AUTO RENEW              1258
4                         IT                         8
5                         PHONE                   4116
6                         BRANCH (POS)             536
7                         BRANCH (PHONE)           193
dtype: int64
```

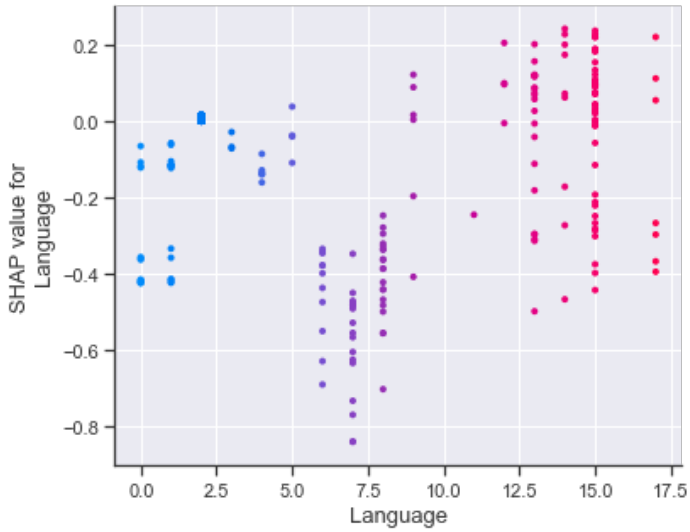
In [332]:
```
shap.dependence_plot(ind='Number_of_Employees', interaction_index='Number_of_Employees',
                     shap_values=shap_values,
                     features=X_validation_enc,
                     display_features=X_validation_enc)
```



In [334]:
```
print(X_validation_disp.groupby(['Number_of_Employees','NumberofEmployees']).size())
```

```
Number_of_Employees  NumberofEmployees
0                    1-5                  1389
1                    11-50                 304
2                    101-500              1077
3                    51-100                 76
4                    500+                  682
5                    6-10                 4419
dtype: int64
```

In [186]:
```
shap.dependence_plot(ind='Language', interaction_index='Language',
                     shap_values=shap_values,
                     features=X_validation_enc,
                     display_features=X_validation_enc)
```

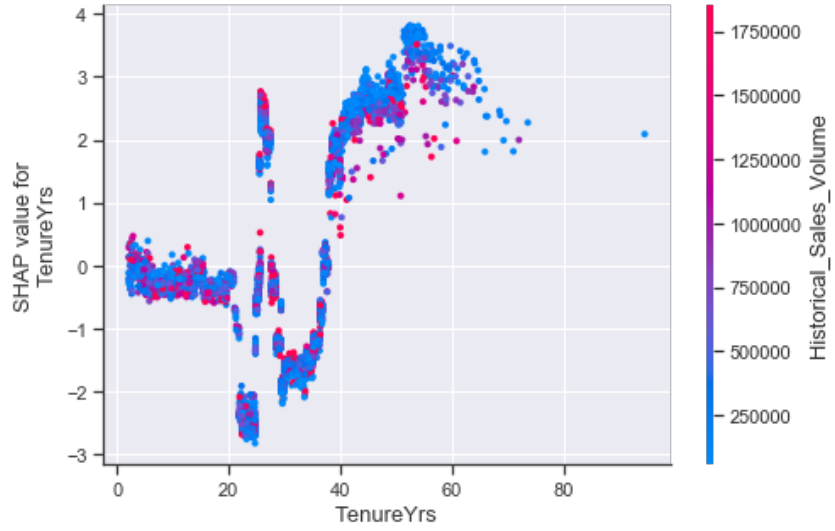


In [187]:
```
print(X_validation_disp.groupby(['Language','Languager']).size())
```

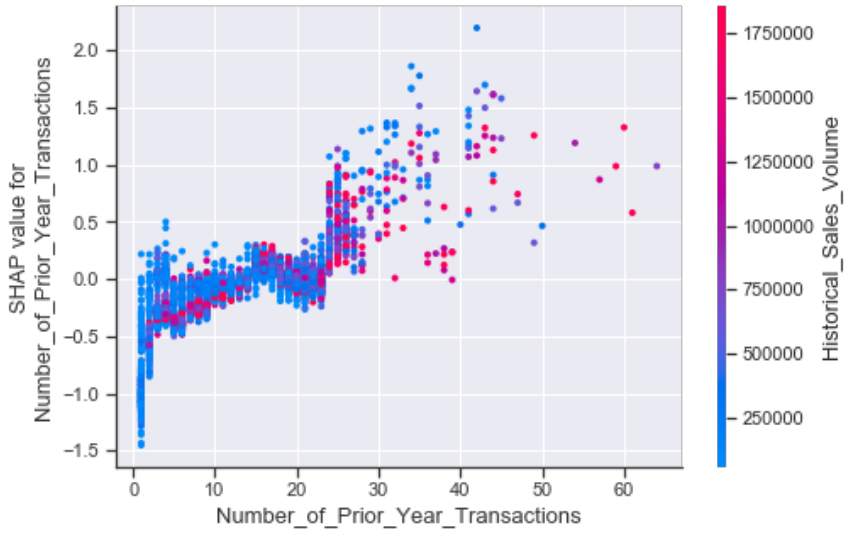
```
Language  Languager
0         Vietnamese      13
1         Portuguese      15
2         English       7766
3         Greek            3
4         Spanish          6
5         Polish           4
6         Italian         11
7         German          17
8         Hindi           20
9         Russian          6
11        Korean           1
12        Arabic           4
13        French          21
14        Japanese         9
15        Chinese         44
17        Hebrew           7
dtype: int64
```

In [188]:
```
shap.dependence_plot(ind='Historical_Sales_Volume', interaction_index='Number_of_Prior_Year_Transactions',
                     shap_values=shap_values, features=X_validation_enc,
                     display_features=X_validation_enc)
```

In [189]:
```
shap.dependence_plot(ind='TenureYrs', interaction_index='Historical_Sales_Volume',
                     shap_values=shap_values, features=X_validation_enc,
                     display_features=X_validation_enc)
```



In [190]:
```
shap.dependence_plot(ind='Number_of_Prior_Year_Transactions', interaction_index
='Historical_Sales_Volume',
                     shap_values=shap_values, features=X_validation_enc,
                     display_features=X_validation_enc)
```



Propensity Files Exportation

In [376]:
```
file_path = '../PGDADS_Capstone Assignment/model_df.csv'
model_df.to_csv(file_path)
print('Data exported successfully!')
```

```
Data exported successfully!
```

In [192]:
```
model_df.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15893 entries, 0 to 16161
Columns: 30 entries, index to PCA_2_Email
dtypes: datetime64[ns](1), float64(23), int64(1), object(5)
memory usage: 3.8+ MB
```

In [193]:
```
xgc.predict_proba(X_encode)[:,1]
```

Out[193]:
```
array([0.9308839 , 0.67296773, 0.8343765 , ..., 0.15478101, 0.00940854,
       0.00364285], dtype=float32)
```

In [194]:
```
predict_prob = pd.DataFrame(data=xgc.predict_proba(X_encode)[:,1],columns=['Est
_Prob'])
predict_prob.tail(3)
```

Out[194]:

	Est_Prob
15890	0.155
15891	0.009
15892	0.004

In [195]:
```
proba_df = pd.concat([model_df.reset_index(), predict_prob],axis = 1)
proba_df.tail(10)
```

Out[195]:

	level_0	index	Customer_Number	Campaign_Period_Sales	Historical_Sales_Volume	Date_of_First_Pu
15883	16152	16162	166802395.000	0.000	1391314.167	198
15884	16153	16163	166873914.000	0.000	3046590.300	199
15885	16154	16164	166893156.000	0.000	1971091.200	198
15886	16155	16165	166911510.000	2566.167	1108584.000	199
15887	16156	16166	166955549.000	0.000	49595.000	201
15888	16157	16167	166988514.000	0.000	701295.400	199
15889	16158	16168	167014041.000	0.000	2558801.000	199
15890	16159	16169	167077817.000	0.000	2355030.000	199
15891	16160	16170	167089540.000	0.000	584570.000	199
15892	16161	16171	167235933.000	0.000	1949425.333	199

In [196]:
```
proba_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15893 entries, 0 to 15892
Data columns (total 32 columns):
level_0                                15893 non-null int64
index                                  15893 non-null int64
Customer_Number                        15893 non-null float64
Campaign_Period_Sales                  15893 non-null float64
Historical_Sales_Volume                15893 non-null float64
Date_of_First_Purchase                 15893 non-null datetime64[ns]
Number_of_Prior_Year_Transactions      15893 non-null float64
Do_Not_Direct_Mail_Solicit             15893 non-null float64
Do_Not_Email                           15893 non-null float64
Do_Not_Telemarket                      15893 non-null float64
Repurchase_Method                      15893 non-null object
Desk                                   15893 non-null float64
Executive_Chair                        15893 non-null float64
Standard_Chair                         15893 non-null float64
Monitor                                15893 non-null float64
Printer                                15893 non-null float64
Computer                               15893 non-null float64
Insurance                              15893 non-null float64
Toner                                  15893 non-null float64
Office_Supplies                        15893 non-null float64
Last_Transaction_Channel               15893 non-null object
Number_of_Employees                    15893 non-null object
Language                               15893 non-null object
Response                               15893 non-null float64
TenureYrs                              15893 non-null float64
ProductMix                             15893 non-null float64
Contact_Channel                        15893 non-null object
Avg_Sales_PU                           15893 non-null float64
PCA_1_Prt_Mon_Com_Chr                  15893 non-null float64
PCA_1_DM_Tele                          15893 non-null float64
PCA_2_Email                            15893 non-null float64
Est_Prob                               15893 non-null float32
dtypes: datetime64[ns](1), float32(1), float64(23), int64(2), object(5)
memory usage: 3.8+ MB
```

In [375]:
```
file_path = '../PGDADS_Capstone Assignment/propensitymodel.csv'
proba_df.to_csv(file_path)
print('Data exported successfully!')
```

```
Data exported successfully!
```

In [197]:
```
proba_df.rename(columns={'level_0': 'Aft_Propensity_Index','index': 'Original_I
ndex'}, inplace=True)
```

Regression Models

In [198]:
```
regress_df = proba_df.copy()
regress_df['log_act_y'] = np.log(regress_df['Campaign_Period_Sales'])
```

In [213]:
```
regress_df['log_act_y'].describe()
```

Out[213]:
```
count   15893.000
mean         -inf
std           nan
min          -inf
25%           nan
50%           nan
75%         4.980
max         9.098
Name: log_act_y, dtype: float64
```

In [199]:
```
regress_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15893 entries, 0 to 15892
Data columns (total 33 columns):
Aft_Propensity_Index                15893 non-null int64
Original_Index                      15893 non-null int64
Customer_Number                     15893 non-null float64
Campaign_Period_Sales               15893 non-null float64
Historical_Sales_Volume             15893 non-null float64
Date_of_First_Purchase              15893 non-null datetime64[ns]
Number_of_Prior_Year_Transactions   15893 non-null float64
Do_Not_Direct_Mail_Solicit          15893 non-null float64
Do_Not_Email                        15893 non-null float64
Do_Not_Telemarket                   15893 non-null float64
Repurchase_Method                   15893 non-null object
Desk                                15893 non-null float64
Executive_Chair                     15893 non-null float64
Standard_Chair                      15893 non-null float64
Monitor                             15893 non-null float64
Printer                             15893 non-null float64
Computer                            15893 non-null float64
Insurance                           15893 non-null float64
Toner                               15893 non-null float64
Office_Supplies                     15893 non-null float64
Last_Transaction_Channel            15893 non-null object
Number_of_Employees                 15893 non-null object
Language                            15893 non-null object
Response                            15893 non-null float64
TenureYrs                           15893 non-null float64
ProductMix                          15893 non-null float64
Contact_Channel                     15893 non-null object
Avg_Sales_PU                        15893 non-null float64
PCA_1_Prt_Mon_Com_Chr               15893 non-null float64
PCA_1_DM_Tele                       15893 non-null float64
PCA_2_Email                         15893 non-null float64
Est_Prob                            15893 non-null float32
log_act_y                           15893 non-null float64
dtypes: datetime64[ns](1), float32(1), float64(24), int64(2), object(5)
memory usage: 3.9+ MB
```

In [200]:
```
list_to_remove2 = ['log_act_y','Aft_Propensity_Index','Customer_Number','Contac
t_Channel','Repurchase_Method','Original_Index','Do_Not_Direct_Mail_Solicit','D
o_Not_Email','Do_Not_Telemarket','Response','Date_of_First_Purchase','Campaign_
Period_Sales','Standard_Chair','Monitor','Printer','Computer','ProductMix']
print('Model Dataset if using Option 1:')
print(regress_df.drop(list_to_remove2, axis=1).info())
```

```
Model Dataset if using Option 1:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15893 entries, 0 to 15892
Data columns (total 16 columns):
Historical_Sales_Volume              15893 non-null float64
Number_of_Prior_Year_Transactions    15893 non-null float64
Desk                                 15893 non-null float64
Executive_Chair                      15893 non-null float64
Insurance                            15893 non-null float64
Toner                                15893 non-null float64
Office_Supplies                      15893 non-null float64
Last_Transaction_Channel             15893 non-null object
Number_of_Employees                  15893 non-null object
Language                             15893 non-null object
TenureYrs                            15893 non-null float64
Avg_Sales_PU                         15893 non-null float64
PCA_1_Prt_Mon_Com_Chr                15893 non-null float64
PCA_1_DM_Tele                        15893 non-null float64
PCA_2_Email                          15893 non-null float64
Est_Prob                             15893 non-null float32
dtypes: float32(1), float64(12), object(3)
memory usage: 1.9+ MB
None
```

In [254]:
```
a = regress_df[regress_df['Est_Prob']>=0.5]
a[['Est_Prob','Campaign_Period_Sales']].describe()
```

Out[254]:

	Est_Prob	Campaign_Period_Sales
count	3939.000	3939.000
mean	0.784	724.921
std	0.118	1091.354
min	0.500	0.000
25%	0.706	134.073
50%	0.784	281.680
75%	0.866	687.760
max	1.000	8936.850

In [257]:
```
regress_df_resp = regress_df[regress_df['Response']==1]
#regress_df_resp = regress_df_resp[regress_df_resp['Est_Prob']>=0.5]
#regress_df_resp = regress_df_resp[regress_df_resp['Campaign_Period_Sales']>0]
regress_df_resp.info()
```

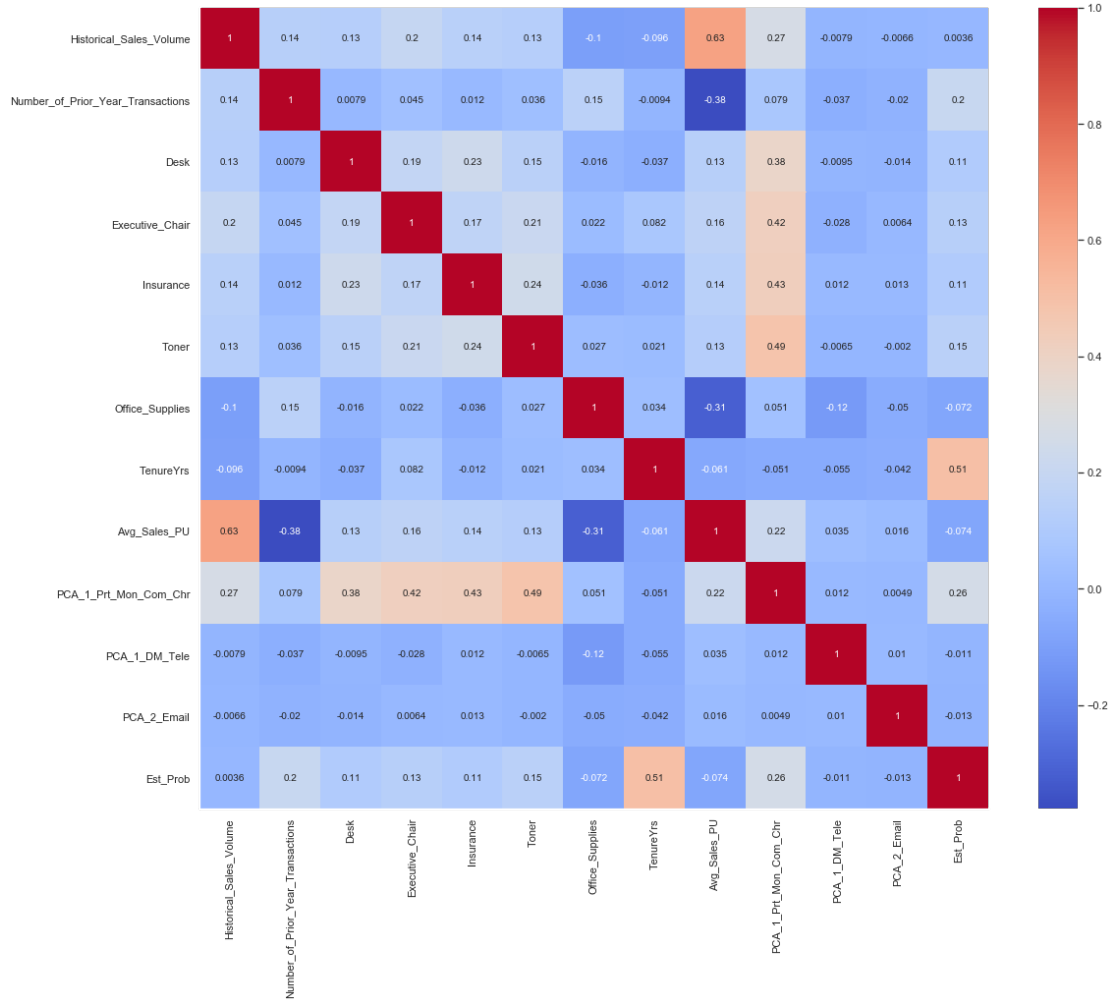
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4308 entries, 0 to 15886
Data columns (total 33 columns):
Aft_Propensity_Index                  4308 non-null int64
Original_Index                        4308 non-null int64
Customer_Number                       4308 non-null float64
Campaign_Period_Sales                 4308 non-null float64
Historical_Sales_Volume               4308 non-null float64
Date_of_First_Purchase                4308 non-null datetime64[ns]
Number_of_Prior_Year_Transactions     4308 non-null float64
Do_Not_Direct_Mail_Solicit            4308 non-null float64
Do_Not_Email                          4308 non-null float64
Do_Not_Telemarket                     4308 non-null float64
Repurchase_Method                     4308 non-null object
Desk                                  4308 non-null float64
Executive_Chair                       4308 non-null float64
Standard_Chair                        4308 non-null float64
Monitor                               4308 non-null float64
Printer                               4308 non-null float64
Computer                              4308 non-null float64
Insurance                             4308 non-null float64
Toner                                 4308 non-null float64
Office_Supplies                       4308 non-null float64
Last_Transaction_Channel              4308 non-null object
Number_of_Employees                   4308 non-null object
Language                              4308 non-null object
Response                              4308 non-null float64
TenureYrs                             4308 non-null float64
ProductMix                            4308 non-null float64
Contact_Channel                       4308 non-null object
Avg_Sales_PU                          4308 non-null float64
PCA_1_Prt_Mon_Com_Chr                 4308 non-null float64
PCA_1_DM_Tele                         4308 non-null float64
PCA_2_Email                           4308 non-null float64
Est_Prob                              4308 non-null float32
log_act_y                             4308 non-null float64
dtypes: datetime64[ns](1), float32(1), float64(24), int64(2), object(5)
memory usage: 1.1+ MB
```

In [258]:
```
X_model_df = regress_df_resp.drop(list_to_remove2,axis=1)
y_model_df = regress_df_resp['log_act_y']
```

In [259]:
```
y_model_df.describe()
```

Out[259]:
```
count   4308.000
mean       6.182
std        1.035
min        1.883
25%        5.415
50%        5.912
75%        6.979
max        9.098
Name: log_act_y, dtype: float64
```

In [260]:
```
corr_reg= regress_df_resp.drop(list_to_remove2,axis=1).select_dtypes(exclude='o
bject').corr()
plt.figure(figsize=(18,15))
sns.heatmap(corr_reg, annot=True, cmap='coolwarm')
sns.despine()
```



In [261]:
```
# Apply train, test split
print(X_model_df.shape, y_model_df.shape)
print('So in X dataframe, there are {} independent variables and in Y datafram
e,\nonly 1 target variable.'.format(X_model_df.shape[1], 1))
print('Also there are {} number of observations.'.format(X_model_df.shape[0]))
```

```
(4308, 16) (4308,)
So in X dataframe, there are 16 independent variables and in Y dataframe,
only 1 target variable.
Also there are 4308 number of observations.
```

In [262]:
```
X_model_df_enc = X_model_df.copy()
for col in X_model_df_enc.select_dtypes(include='object').columns:
    encode(X_model_df_enc[col])
```

In [263]:
```
X_model_df_enc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4308 entries, 0 to 15886
Data columns (total 16 columns):
Historical_Sales_Volume               4308 non-null float64
Number_of_Prior_Year_Transactions     4308 non-null float64
Desk                                  4308 non-null float64
Executive_Chair                       4308 non-null float64
Insurance                             4308 non-null float64
Toner                                 4308 non-null float64
Office_Supplies                       4308 non-null float64
Last_Transaction_Channel              4308 non-null int64
Number_of_Employees                   4308 non-null int64
Language                              4308 non-null int64
TenureYrs                             4308 non-null float64
Avg_Sales_PU                          4308 non-null float64
PCA_1_Prt_Mon_Com_Chr                 4308 non-null float64
PCA_1_DM_Tele                         4308 non-null float64
PCA_2_Email                           4308 non-null float64
Est_Prob                              4308 non-null float32
dtypes: float32(1), float64(12), int64(3)
memory usage: 555.3 KB
```

In [264]:
```
X_model_df_OHE = pd.get_dummies(X_model_df)
X_model_df_OHE.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4308 entries, 0 to 15886
Data columns (total 44 columns):
Historical_Sales_Volume                      4308 non-null float64
Number_of_Prior_Year_Transactions            4308 non-null float64
Desk                                         4308 non-null float64
Executive_Chair                              4308 non-null float64
Insurance                                    4308 non-null float64
Toner                                        4308 non-null float64
Office_Supplies                              4308 non-null float64
TenureYrs                                    4308 non-null float64
Avg_Sales_PU                                 4308 non-null float64
PCA_1_Prt_Mon_Com_Chr                        4308 non-null float64
PCA_1_DM_Tele                                4308 non-null float64
PCA_2_Email                                  4308 non-null float64
Est_Prob                                     4308 non-null float32
Last_Transaction_Channel_AUTO RENEW          4308 non-null uint8
Last_Transaction_Channel_BILLING             4308 non-null uint8
Last_Transaction_Channel_BRANCH (PHONE)      4308 non-null uint8
Last_Transaction_Channel_BRANCH (POS)        4308 non-null uint8
Last_Transaction_Channel_IT                  4308 non-null uint8
Last_Transaction_Channel_MAIL                4308 non-null uint8
Last_Transaction_Channel_PHONE               4308 non-null uint8
Last_Transaction_Channel_WEB                 4308 non-null uint8
Number_of_Employees_1-5                      4308 non-null uint8
Number_of_Employees_101-500                  4308 non-null uint8
Number_of_Employees_11-50                    4308 non-null uint8
Number_of_Employees_500+                     4308 non-null uint8
Number_of_Employees_51-100                   4308 non-null uint8
Number_of_Employees_6-10                     4308 non-null uint8
Language_Arabic                              4308 non-null uint8
Language_Chinese                             4308 non-null uint8
Language_English                             4308 non-null uint8
Language_French                              4308 non-null uint8
Language_German                              4308 non-null uint8
Language_Greek                               4308 non-null uint8
Language_Hebrew                              4308 non-null uint8
Language_Hindi                               4308 non-null uint8
Language_Italian                             4308 non-null uint8
Language_Japanese                            4308 non-null uint8
Language_Pashto                              4308 non-null uint8
Language_Polish                              4308 non-null uint8
Language_Portuguese                          4308 non-null uint8
Language_Russian                             4308 non-null uint8
Language_Spanish                             4308 non-null uint8
Language_Thai                                4308 non-null uint8
Language_Vietnamese                          4308 non-null uint8
dtypes: float32(1), float64(12), uint8(31)
memory usage: 584.8 KB
```

In [265]:
```
X_train_enc2, X_validation_enc2, y_train_enc2, y_validation_enc2 = train_test_s
plit(X_model_df_enc,y_model_df,test_size = 0.50,random_state = 42)
print('Ordinal Encoder Training examples: {}'.format(X_train_enc2.shape[0]))
print('So in X_train dataframe, there are {} independent variables and only 1 t
arget variable.'.format(X_train_enc2.shape[1],+1))
print('Also there are {} number of observations.'.format(X_train_enc2.shape
[0]))
print('==' * 50)
print('Ordinal Encoder Validation examples:  {}'.format(X_validation_enc2.shape
[0]))
print('So in X_validation dataframe, there are {} independent variables and onl
y 1 target variable.'.format(X_validation_enc2.shape[1],+1))
print('Also there are {} number of observations.'.format(X_validation_enc2.shap
e[0]))
```

```
Ordinal Encoder Training examples: 2154
So in X_train dataframe, there are 16 independent variables and only 1 target
variable.
Also there are 2154 number of observations.
==========================================================================
======================
Ordinal Encoder Validation examples:  2154
So in X_validation dataframe, there are 16 independent variables and only 1 ta
rget variable.
Also there are 2154 number of observations.
```

In [266]:
```
X_train_ohe, X_validation_ohe, y_train_ohe, y_validation_ohe = train_test_split
(X_model_df_OHE,y_model_df,test_size = 0.50,random_state = 42)
print('Ordinal Encoder Training examples: {}'.format(X_train_ohe.shape[0]))
print('So in X_train dataframe, there are {} independent variables and only 1 t
arget variable.'.format(X_train_ohe.shape[1],+1))
print('Also there are {} number of observations.'.format(X_train_ohe.shape[0]))
print('==' * 50)
print('Ordinal Encoder Validation examples:  {}'.format(X_validation_ohe.shape
[0]))
print('So in X_validation dataframe, there are {} independent variables and onl
y 1 target variable.'.format(X_validation_ohe.shape[1],+1))
print('Also there are {} number of observations.'.format(X_validation_ohe.shape
[0]))
```

```
Ordinal Encoder Training examples: 2154
So in X_train dataframe, there are 44 independent variables and only 1 target
variable.
Also there are 2154 number of observations.
==========================================================================
======================
Ordinal Encoder Validation examples:  2154
So in X_validation dataframe, there are 44 independent variables and only 1 ta
rget variable.
Also there are 2154 number of observations.
```

In [267]:
```
linear_mod = LinearRegression()

# Construct the hyperparameter grid
param_grid = {'fit_intercept':[True,False], 'normalize':[True,False], 'copy_X
':[True, False]}

# Instantiate the GridSearchCV object using the estimator object,
# the hyperparameter grid, & 5-fold cross-validation
grid_linear = GridSearchCV(linear_mod, param_grid=param_grid, cv=5)
grid_linear.fit(X_train_ohe, y_train_ohe)
y_pred_linear = grid_linear.predict(X_validation_ohe)
print('R2 Score   :', round(r2_score(y_validation_ohe, y_pred_linear),4))
print('MSE        :', round(metrics.mean_squared_error(y_validation_ohe, y_pred
_linear),4))
print('RMSE       :', round(np.sqrt(metrics.mean_squared_error(y_validation_oh
e, y_pred_linear)),4))
print('Optimal hyperparameter(s): {}.'.format(dict(grid_linear.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_linear.best_estimator_))
```

```
R2 Score   : 0.7037
MSE        : 0.3185
RMSE       : 0.5644
Optimal hyperparameter(s): {'copy_X': True, 'fit_intercept': True, 'normalize
': False}.
Optimal Estimator:
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

In [268]:
```
rr = Ridge(random_state=24)

# Construct the hyperparameter grid
param_grid = {'alpha': [0.0001, 0.001, 0.01, 0.02, 0.1, 1, 10, 100, 1000],
              'fit_intercept':[True,False],'normalize':[True,False], 'copy_X
':[True, False],
              'max_iter' :[100,1000,10e3,10e5]}
#higher the alpha value, more restriction on the coefficients;
#low alpha > more generalization, coefficients are barely

#Instantiate the GridSearchCV object using the estimator object,
# the hyperparameter grid, & 5-fold cross-validation
grid_ridge = GridSearchCV(rr, param_grid=param_grid, cv=5)
grid_ridge.fit(X_train_ohe, y_train_ohe)
y_pred_rr = grid_ridge.predict(X_validation_ohe)
print('R2 Score   :', round(r2_score(y_validation_ohe, y_pred_rr),4))
print('MSE        :', round(metrics.mean_squared_error(y_validation_ohe, y_pred
_rr),4))
print('RMSE       :', round(np.sqrt(metrics.mean_squared_error(y_validation_oh
e, y_pred_rr)),4))
print('Optimal hyperparameter(s): {}.'.format(dict(grid_ridge.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_ridge.best_estimator_))
```

```
R2 Score   : 0.7047
MSE        : 0.3175
RMSE       : 0.5634
Optimal hyperparameter(s): {'alpha': 1, 'copy_X': True, 'fit_intercept': True,
'max_iter': 100, 'normalize': False}.
Optimal Estimator:
Ridge(alpha=1, copy_X=True, fit_intercept=True, max_iter=100, normalize=False,
      random_state=24, solver='auto', tol=0.001)
```

In [269]:
```
elastic = ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True,
           l1_ratio=0.30000000000000004, max_iter=10, normalize=False,
           positive=False, precompute=False, random_state=24,
           selection='cyclic', tol=0.0001, warm_start=False)
# Construct the hyperparameter grid
param_grid = {#'alpha': [0.0001, 0.001, 0.01, 0.02, 0.1, 1, 10, 100, 1000],
              #  'l1_ratio': np.arange(0.0, 1.0, 0.1),
              #   'max_iter' :[10, 100, 1000]
             }

#Instantiate the GridSearchCV object using the estimator object,
# the hyperparameter grid, & 5-fold cross-validation
grid_elastic= GridSearchCV(elastic, param_grid=param_grid, cv=5)
grid_elastic.fit(X_train_ohe, y_train_ohe)
y_pred_elastic = grid_elastic.predict(X_validation_ohe)
print('R2 Score   :', round(r2_score(y_validation_ohe, y_pred_elastic),4))
print('MSE        :', round(metrics.mean_squared_error(y_validation_ohe, y_pred
_elastic),4))
print('RMSE       :', round(np.sqrt(metrics.mean_squared_error(y_validation_oh
e, y_pred_elastic)),4))
print('Optimal hyperparameter(s): {}.'.format(dict(grid_elastic.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_elastic.best_estimator_))
```

```
R2 Score   : 0.6971
MSE        : 0.3257
RMSE       : 0.5707
Optimal hyperparameter(s): {}.
Optimal Estimator:
ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True,
           l1_ratio=0.30000000000000004, max_iter=10, normalize=False,
           positive=False, precompute=False, random_state=24,
           selection='cyclic', tol=0.0001, warm_start=False)
```

In [270]:
```
rf = RandomForestRegressor(random_state=24)
# Construct the hyperparameter grid
param_grid = {'n_estimators':[1000, 3000],
                  'max_features': ['auto', 'sqrt', 'log2'],
                  'max_depth' : [3,5,7],
                  'criterion' : ['mse', 'entropy'],
                  'oob_score' :[True, False]
             }
# Instantiate the GridSearchCV object using the estimator object,
# the hyperparameter grid, & 5-fold cross-validation
grid_rand = GridSearchCV(rf, param_grid=param_grid, cv=5)
grid_rand.fit(X_train_enc2, y_train_enc2)
y_pred_rand = grid_rand.predict(X_validation_enc2)
print('R2 Score   :', round(r2_score(y_validation_enc2, y_pred_rand),4))
print('MSE        :', round(metrics.mean_squared_error(y_validation_enc2, y_pre
d_rand),4))
print('RMSE       :', round(np.sqrt(metrics.mean_squared_error(y_validation_enc
2, y_pred_rand)),4))
print('Optimal hyperparameter(s): {}.'.format(dict(grid_rand.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_rand.best_estimator_))
```

```
R2 Score   : 0.8034
MSE        : 0.2114
RMSE       : 0.4598
Optimal hyperparameter(s): {'criterion': 'mse', 'max_depth': 7, 'max_features
': 'auto', 'n_estimators': 3000, 'oob_score': True}.
Optimal Estimator:
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=7, max_features='auto', max_leaf_nodes=None,
                      max_samples=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=3000, n_jobs=None, oob_score=True,
                      random_state=24, verbose=0, warm_start=False)
```

In [271]:
```
gbm = GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_m
se',
                                init=None, learning_rate=0.01, loss='ls', max_depth=
5,
                                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, n_estimators=1000,
                                n_iter_no_change=None, presort='deprecated',
                                random_state=24, subsample=1.0, tol=0.0001,
                                validation_fraction=0.1, verbose=0, warm_start=False)

# Construct the hyperparameter grid
param_grid = {#'n_estimators': [1000],#, 2000, 3000],
              #'max_features': ['auto', 'sqrt', 'log2'],
              #'criterion' :['friedman_mse'],#'gini', 'entropy'],
              #'max_depth' : [5],#[3,5,7,8],
              #'learning_rate' : [0.005,0.01]#[0.01,0.02,0.03,0.05]
             }
# Instantiate the GridSearchCV object using the estimator object,
# the hyperparameter grid, & 5-fold cross-validation
grid_gbmr = GridSearchCV(gbm, param_grid=param_grid, cv=5)
grid_gbmr.fit(X_train_enc2, y_train_enc2)
y_pred_gbmr = grid_gbmr.predict(X_validation_enc2)
print('R2 Score   :', round(r2_score(y_validation_enc2, y_pred_gbmr),4))
print('MSE        :', round(metrics.mean_squared_error(y_validation_enc2, y_pre
d_gbmr),4))
print('RMSE       :', round(np.sqrt(metrics.mean_squared_error(y_validation_enc
2, y_pred_gbmr)),4))
print('Optimal hyperparameter(s): {}.'.format(dict(grid_gbmr.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_gbmr.best_estimator_))
```

```
R2 Score   : 0.8152
MSE        : 0.1987
RMSE       : 0.4457
Optimal hyperparameter(s): {}.
Optimal Estimator:
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.01, loss='ls', max_depth=
5,
                          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, n_estimators=1000,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=24, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=Fals
e)
```

Final Regression Model

In [273]:
```
xgbr = xgb.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, criterion='friedman_mse',
             gamma=0, importance_type='gain', learning_rate=0.005,
             max_delta_step=0, max_depth=5, max_features='auto',
             min_child_weight=1, missing=None, n_estimators=2000, n_jobs=1,
             nthread=None, objective='reg:linear', random_state=24, reg_alpha=
0,
             reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
             subsample=1, verbosity=1)
xgbr.fit(X_train_enc2, y_train_enc2)
y_pred_gbm_regr = xgbr.predict(X_validation_enc2)
```

```
[18:36:23] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
```

In [274]:
```
print('R2 Score   :', round(r2_score(y_validation_enc2, y_pred_gbm_regr),4))
print('MSE        :', round(metrics.mean_squared_error(y_validation_enc2, y_pre
d_gbm_regr),4))
print('RMSE       :', round(np.sqrt(metrics.mean_squared_error(y_validation_enc
2, y_pred_gbm_regr)),4))
print('Optimal hyperparameter(s): {}.'.format(dict(grid_xgbr.best_params_)))
print('Optimal Estimator:\n{}'.format(grid_xgbr.best_estimator_))
```

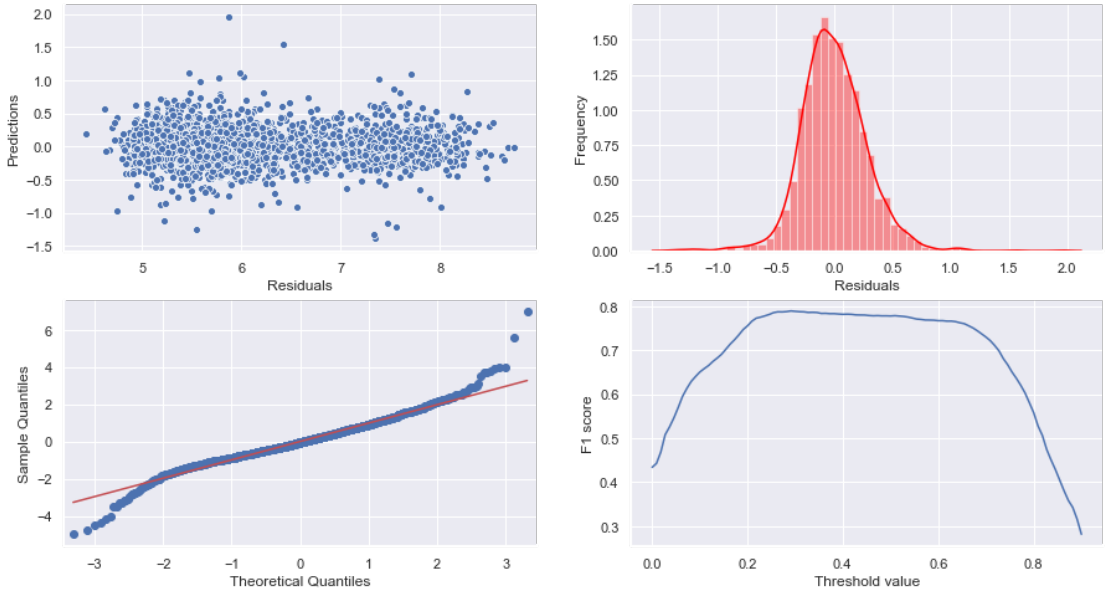
```
R2 Score   : 0.816
MSE        : 0.1978
RMSE       : 0.4448
Optimal hyperparameter(s): {'criterion': 'friedman_mse', 'learning_rate': 0.00
5, 'max_depth': 5, 'max_features': 'auto', 'n_estimators': 2000}.
Optimal Estimator:
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, criterion='friedman_mse',
             gamma=0, importance_type='gain', learning_rate=0.005,
             max_delta_step=0, max_depth=5, max_features='auto',
             min_child_weight=1, missing=None, n_estimators=2000, n_jobs=1,
             nthread=None, objective='reg:linear', random_state=24, reg_alpha=
0,
             reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
             subsample=1, verbosity=1)
```

In [278]:
```
xgbr.fit(X_train_enc2, y_train_enc2)
xgbr_dffeatures = pd.DataFrame(xgbr.feature_importances_)
xgbr_dfcolumns = pd.DataFrame(X_train_enc2.columns)
xgbr_featureScores = pd.concat([xgbr_dfcolumns, xgbr_dffeatures], axis=1)
xgbr_featureScores.columns = ['Features','Importance'] # naming the dataframe c
olumns
print(xgbr_featureScores.nlargest(10,'Importance')) # provide top features
```

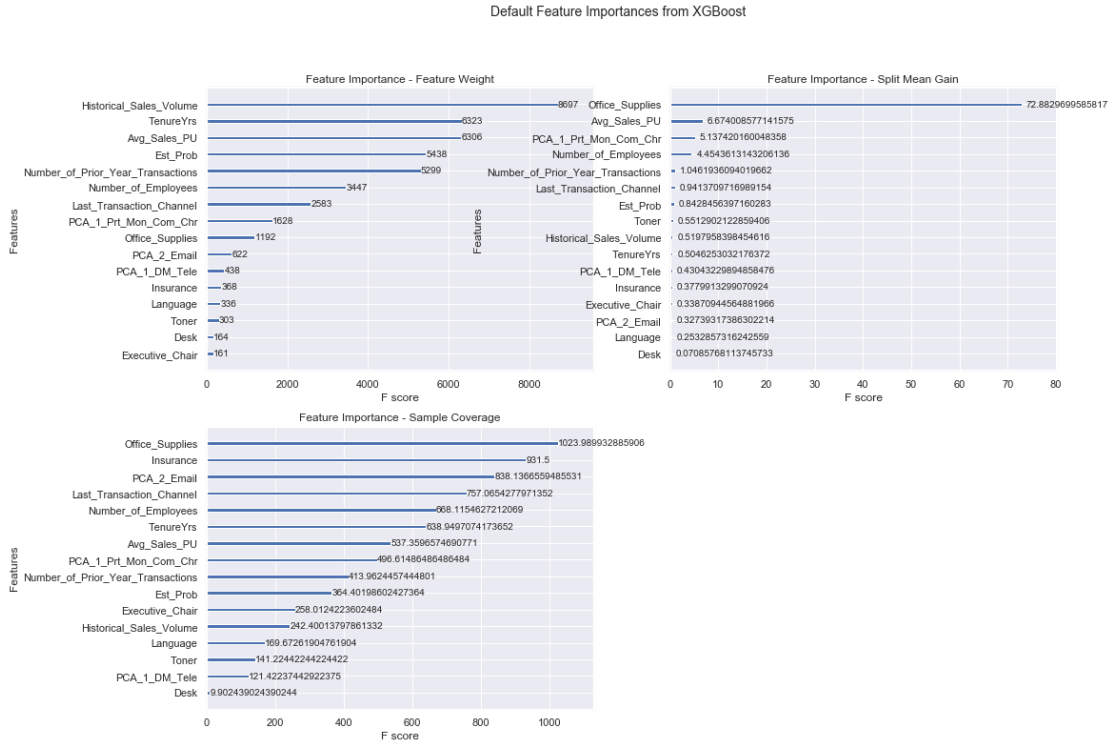
```
[18:42:09] WARNING: src/objective/regression_obj.cu:152: reg:linear is now dep
recated in favor of reg:squarederror.
                            Features  Importance
6                      Office_Supplies       0.764
11                        Avg_Sales_PU       0.070
12               PCA_1_Prt_Mon_Com_Chr       0.054
8                  Number_of_Employees       0.047
1   Number_of_Prior_Year_Transactions       0.011
7             Last_Transaction_Channel       0.010
15                            Est_Prob       0.009
5                                Toner       0.006
0              Historical_Sales_Volume       0.005
10                          TenureYrs       0.005
```

In [364]:
```
residuals = y_train_enc2 - grid_gbmr.predict(X_train_enc2)
```

In [371]:
```
f,axes = plt.subplots(2, 2, figsize=(15,8))
# plot scatter on upper right plot
g=sns.scatterplot(x=y_pred_gbm_regr, y =residuals,ax=axes[0,0])
g.set(xlabel="Residuals",ylabel="Predictions")
# plot hist on upper left plot
h=sns.distplot(residuals, color='red',ax=axes[0,1])
h.set(xlabel='Residuals', ylabel='Frequency')
# plot qq plot on lower left
qqplot(residuals, fit=True, line='r', ax=axes[1,0])
# classifier thresholds on lower right
axes[1,1].plot(thresholds_train, train_f1_scores)
axes[1,1].set(xlabel="Threshold value",ylabel="F1 score");
sns.despine()
```



In [279]:
```
fig = plt.figure(figsize = (16, 12))
title = fig.suptitle("Default Feature Importances from XGBoost", fontsize=14)

ax1 = fig.add_subplot(2,2,1)
xgb.plot_importance(xgbr, importance_type='weight', ax=ax1)
t=ax1.set_title("Feature Importance - Feature Weight")

ax2 = fig.add_subplot(2,2,2)
xgb.plot_importance(xgbr, importance_type='gain', ax=ax2)
t=ax2.set_title("Feature Importance - Split Mean Gain")

ax3 = fig.add_subplot(2,2,3)
xgb.plot_importance(xgbr, importance_type='cover', ax=ax3)
t=ax3.set_title("Feature Importance - Sample Coverage")
```



In [280]:
```
import shap
explainer = shap.TreeExplainer(xgbr)
shap_values = explainer.shap_values(X_validation_enc2)
print('Expected Value:', explainer.expected_value)
pd.DataFrame(shap_values).head()
```

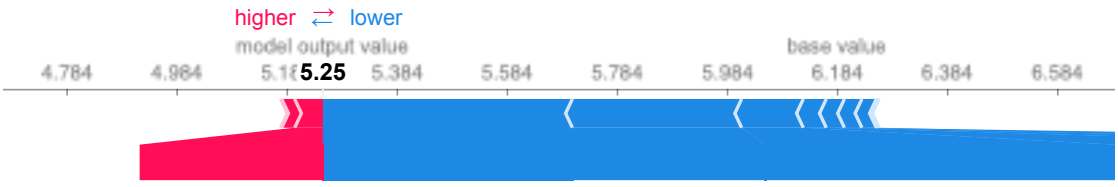
```
Expected Value: 6.1838126
```

Out[280]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	-0.112	-0.026	-0.000	-0.002	0.001	0.001	-0.308	-0.042	0.024	0.000	0.051	-0.453	-0.032	-0.001
1	0.025	-0.022	-0.000	-0.000	-0.006	0.001	0.489	-0.043	-1.049	-0.000	0.028	-0.093	-0.031	-0.003
2	-0.067	-0.020	0.000	-0.002	-0.003	0.001	-0.188	-0.037	-0.452	-0.000	-0.012	-0.295	-0.026	-0.001
3	-0.068	-0.045	0.000	-0.000	0.005	0.007	-0.314	0.119	0.040	0.000	0.014	-0.246	-0.039	-0.001
4	-0.007	-0.033	0.000	-0.000	0.002	0.001	-0.190	-0.028	-0.435	-0.001	0.038	-0.302	-0.025	0.012

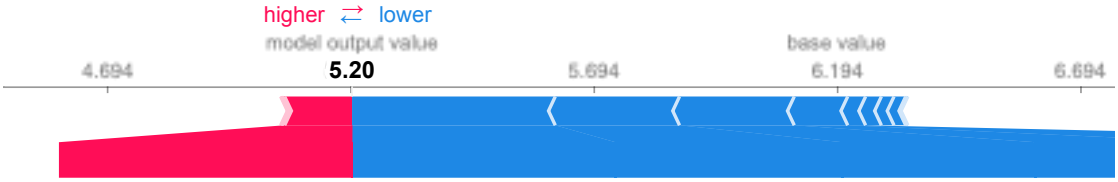
In [281]:
```
shap.initjs()
shap.force_plot(explainer.expected_value,shap_values[0,:], X_validation_enc2.il
oc[0,:])
```

Out[281]:



In [249]:
```
shap.force_plot(explainer.expected_value,shap_values[2,:], X_validation_enc2.il
oc[2,:])
```

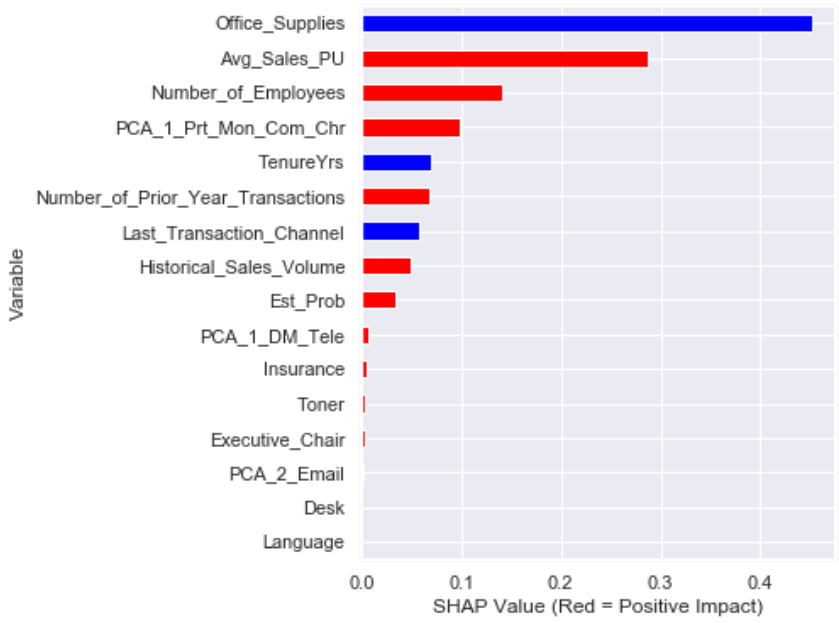
Out[249]:



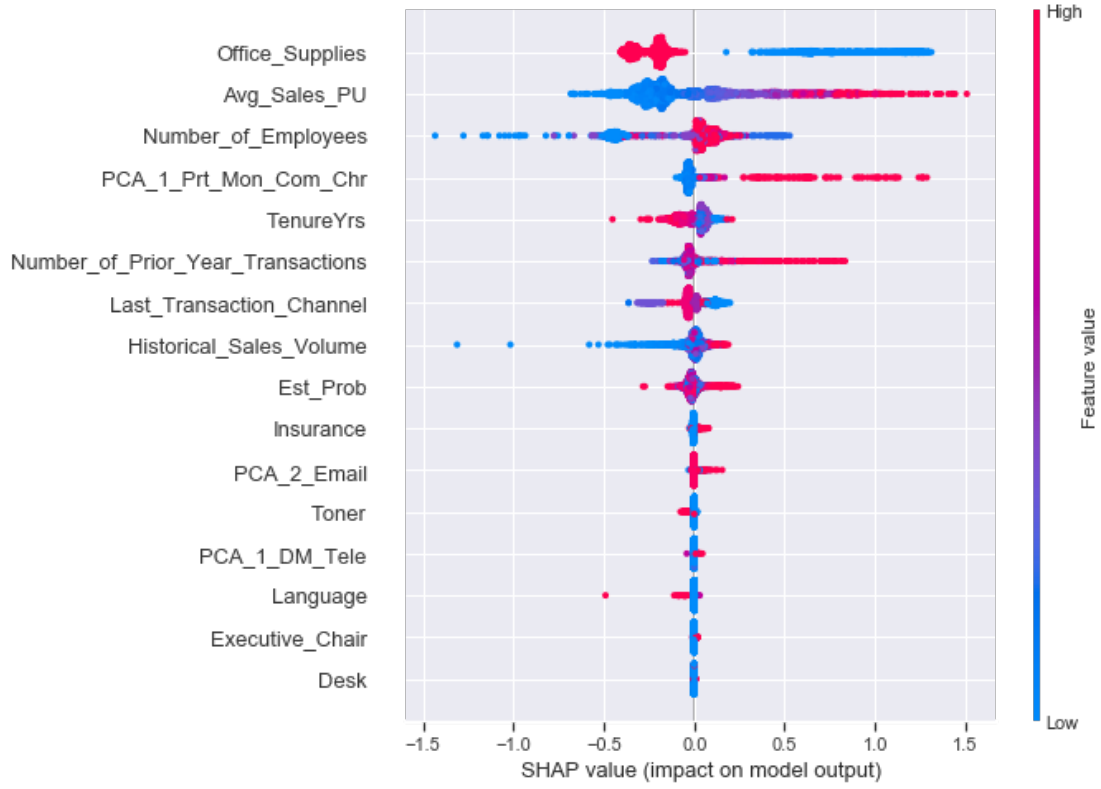
In [282]:
```
shap.force_plot(explainer.expected_value, shap_values[:2000,:], X_validation_en
c.iloc[:2000,:])
```

Out[282]:

In [251]:
```
ABS_SHAP(shap_values,X_validation_enc2)
```



In [283]:
```
shap.summary_plot(shap_values, X_validation_enc2)
```



Data Merging Process

In [284]:
```
regress_df_enc = regress_df.copy()
for col in regress_df_enc.select_dtypes(include='object').columns:
    encode(regress_df_enc[col])
```

In [285]:
```
X_train_enc2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2154 entries, 9832 to 3094
Data columns (total 16 columns):
Historical_Sales_Volume               2154 non-null float64
Number_of_Prior_Year_Transactions     2154 non-null float64
Desk                                  2154 non-null float64
Executive_Chair                       2154 non-null float64
Insurance                             2154 non-null float64
Toner                                 2154 non-null float64
Office_Supplies                       2154 non-null float64
Last_Transaction_Channel              2154 non-null int64
Number_of_Employees                   2154 non-null int64
Language                              2154 non-null int64
TenureYrs                             2154 non-null float64
Avg_Sales_PU                          2154 non-null float64
PCA_1_Prt_Mon_Com_Chr                 2154 non-null float64
PCA_1_DM_Tele                         2154 non-null float64
PCA_2_Email                           2154 non-null float64
Est_Prob                              2154 non-null float32
dtypes: float32(1), float64(12), int64(3)
memory usage: 277.7 KB
```

In [286]:
```
regress_df_enc.drop(list_to_remove2,axis=1).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15893 entries, 0 to 15892
Data columns (total 16 columns):
Historical_Sales_Volume               15893 non-null float64
Number_of_Prior_Year_Transactions     15893 non-null float64
Desk                                  15893 non-null float64
Executive_Chair                       15893 non-null float64
Insurance                             15893 non-null float64
Toner                                 15893 non-null float64
Office_Supplies                       15893 non-null float64
Last_Transaction_Channel              15893 non-null int64
Number_of_Employees                   15893 non-null int64
Language                              15893 non-null int64
TenureYrs                             15893 non-null float64
Avg_Sales_PU                          15893 non-null float64
PCA_1_Prt_Mon_Com_Chr                 15893 non-null float64
PCA_1_DM_Tele                         15893 non-null float64
PCA_2_Email                           15893 non-null float64
Est_Prob                              15893 non-null float32
dtypes: float32(1), float64(12), int64(3)
memory usage: 1.9 MB
```

In [287]:
```
proba_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15893 entries, 0 to 15892
Data columns (total 32 columns):
Aft_Propensity_Index                 15893 non-null int64
Original_Index                       15893 non-null int64
Customer_Number                      15893 non-null float64
Campaign_Period_Sales                15893 non-null float64
Historical_Sales_Volume              15893 non-null float64
Date_of_First_Purchase               15893 non-null datetime64[ns]
Number_of_Prior_Year_Transactions    15893 non-null float64
Do_Not_Direct_Mail_Solicit           15893 non-null float64
Do_Not_Email                         15893 non-null float64
Do_Not_Telemarket                    15893 non-null float64
Repurchase_Method                    15893 non-null object
Desk                                 15893 non-null float64
Executive_Chair                      15893 non-null float64
Standard_Chair                       15893 non-null float64
Monitor                              15893 non-null float64
Printer                              15893 non-null float64
Computer                             15893 non-null float64
Insurance                            15893 non-null float64
Toner                                15893 non-null float64
Office_Supplies                      15893 non-null float64
Last_Transaction_Channel             15893 non-null object
Number_of_Employees                  15893 non-null object
Language                             15893 non-null object
Response                             15893 non-null float64
TenureYrs                            15893 non-null float64
ProductMix                           15893 non-null float64
Contact_Channel                      15893 non-null object
Avg_Sales_PU                         15893 non-null float64
PCA_1_Prt_Mon_Com_Chr                15893 non-null float64
PCA_1_DM_Tele                        15893 non-null float64
PCA_2_Email                          15893 non-null float64
Est_Prob                             15893 non-null float32
dtypes: datetime64[ns](1), float32(1), float64(23), int64(2), object(5)
memory usage: 3.8+ MB
```

In [288]:
```
regress_df_enc[regress_df_enc['Response']==1].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4308 entries, 0 to 15886
Data columns (total 33 columns):
Aft_Propensity_Index                  4308 non-null int64
Original_Index                        4308 non-null int64
Customer_Number                       4308 non-null float64
Campaign_Period_Sales                 4308 non-null float64
Historical_Sales_Volume               4308 non-null float64
Date_of_First_Purchase                4308 non-null datetime64[ns]
Number_of_Prior_Year_Transactions     4308 non-null float64
Do_Not_Direct_Mail_Solicit            4308 non-null float64
Do_Not_Email                          4308 non-null float64
Do_Not_Telemarket                     4308 non-null float64
Repurchase_Method                     4308 non-null int64
Desk                                  4308 non-null float64
Executive_Chair                       4308 non-null float64
Standard_Chair                        4308 non-null float64
Monitor                               4308 non-null float64
Printer                               4308 non-null float64
Computer                              4308 non-null float64
Insurance                             4308 non-null float64
Toner                                 4308 non-null float64
Office_Supplies                       4308 non-null float64
Last_Transaction_Channel              4308 non-null int64
Number_of_Employees                   4308 non-null int64
Language                              4308 non-null int64
Response                              4308 non-null float64
TenureYrs                             4308 non-null float64
ProductMix                            4308 non-null float64
Contact_Channel                       4308 non-null int64
Avg_Sales_PU                          4308 non-null float64
PCA_1_Prt_Mon_Com_Chr                 4308 non-null float64
PCA_1_DM_Tele                         4308 non-null float64
PCA_2_Email                           4308 non-null float64
Est_Prob                              4308 non-null float32
log_act_y                             4308 non-null float64
dtypes: datetime64[ns](1), float32(1), float64(24), int64(7)
memory usage: 1.1 MB
```

In [289]:
```
regress_est = grid_gbmr.predict(regress_df_enc[regress_df_enc['Response']==1].d
rop(list_to_remove2,axis=1))
```

In [290]:
```
regress_est_log = pd.DataFrame(data=regress_est,columns=['log_hats_y'])
regress_est_log['Est_Sale']=np.exp(regress_est_log['log_hats_y'])
regress_est_log
```

Out[290]:

	log_hats_y	Est_Sale
0	5.717	303.897
1	5.741	311.524
2	5.981	395.976
3	7.267	1431.795
4	5.386	218.260
...	...	...
4303	8.192	3613.644
4304	7.075	1181.578
4305	7.049	1151.535
4306	7.604	2006.648
4307	7.907	2715.563

4308 rows × 2 columns

In [291]:
```
regress_df_resp
```

Out[291]:

	Aft_Propensity_Index	Original_Index	Customer_Number	Campaign_Period_Sales	Historical_Sales_Vol
0	0	0	86734.000	238.705	14680
1	1	1	97098.000	281.680	43998
2	2	2	100836.000	432.857	97046
4	4	4	127914.000	1370.167	2740
5	5	5	148529.000	308.419	18258
...	...	...	...	...	
15851	16120	16130	164280518.000	3187.667	34108
15878	16147	16157	165550176.000	1216.500	20437
15879	16148	16158	166618255.000	2407.167	28886
15880	16149	16159	166635531.000	2179.233	22228
15886	16155	16165	166911510.000	2566.167	11085

4308 rows × 33 columns

In [292]:
```
regress_df = pd.concat([regress_df_resp.reset_index(), regress_est_log],axis = 1)
regress_df.head(10)
```

Out[292]:

	index	Aft_Propensity_Index	Original_Index	Customer_Number	Campaign_Period_Sales	Historical_Sales_
0	0	0	0	86734.000	238.705	146
1	1	1	1	97098.000	281.680	439
2	2	2	2	100836.000	432.857	970
3	4	4	4	127914.000	1370.167	27
4	5	5	5	148529.000	308.419	182
5	8	8	8	166859.000	2241.500	609
6	9	9	9	170336.000	223.440	277
7	11	11	11	186013.000	124.760	199
8	15	15	15	212131.000	110.333	148
9	17	17	17	234724.000	141.712	637

In [293]:
```
regress_df.rename(columns={'index':'Aft_Regression_Index'})
```

Out[293]:

	Aft_Regression_Index	Aft_Propensity_Index	Original_Index	Customer_Number	Campaign_Period_Sales
0	0	0	0	86734.000	238.705
1	1	1	1	97098.000	281.680
2	2	2	2	100836.000	432.857
3	4	4	4	127914.000	1370.167
4	5	5	5	148529.000	308.419
...	...	...	...	...	..
4303	15851	16120	16130	164280518.000	3187.667
4304	15878	16147	16157	165550176.000	1216.500
4305	15879	16148	16158	166618255.000	2407.167
4306	15880	16149	16159	166635531.000	2179.233
4307	15886	16155	16165	166911510.000	2566.167

4308 rows × 36 columns

In [294]:
```
regress_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4308 entries, 0 to 4307
Data columns (total 36 columns):
index                                4308 non-null int64
Aft_Propensity_Index                 4308 non-null int64
Original_Index                       4308 non-null int64
Customer_Number                      4308 non-null float64
Campaign_Period_Sales                4308 non-null float64
Historical_Sales_Volume              4308 non-null float64
Date_of_First_Purchase               4308 non-null datetime64[ns]
Number_of_Prior_Year_Transactions    4308 non-null float64
Do_Not_Direct_Mail_Solicit           4308 non-null float64
Do_Not_Email                         4308 non-null float64
Do_Not_Telemarket                    4308 non-null float64
Repurchase_Method                    4308 non-null object
Desk                                 4308 non-null float64
Executive_Chair                      4308 non-null float64
Standard_Chair                       4308 non-null float64
Monitor                              4308 non-null float64
Printer                              4308 non-null float64
Computer                             4308 non-null float64
Insurance                            4308 non-null float64
Toner                                4308 non-null float64
Office_Supplies                      4308 non-null float64
Last_Transaction_Channel             4308 non-null object
Number_of_Employees                  4308 non-null object
Language                             4308 non-null object
Response                             4308 non-null float64
TenureYrs                            4308 non-null float64
ProductMix                           4308 non-null float64
Contact_Channel                      4308 non-null object
Avg_Sales_PU                         4308 non-null float64
PCA_1_Prt_Mon_Com_Chr                4308 non-null float64
PCA_1_DM_Tele                        4308 non-null float64
PCA_2_Email                          4308 non-null float64
Est_Prob                             4308 non-null float32
log_act_y                            4308 non-null float64
log_hats_y                           4308 non-null float64
Est_Sale                             4308 non-null float64
dtypes: datetime64[ns](1), float32(1), float64(26), int64(3), object(5)
memory usage: 1.2+ MB
```

In [297]:
```
small_df = regress_df[['Customer_Number','Est_Prob','log_act_y','log_hats_y','E
st_Sale']]
```

In [298]:
```
proba_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15893 entries, 0 to 15892
Data columns (total 32 columns):
Aft_Propensity_Index                  15893 non-null int64
Original_Index                        15893 non-null int64
Customer_Number                       15893 non-null float64
Campaign_Period_Sales                 15893 non-null float64
Historical_Sales_Volume               15893 non-null float64
Date_of_First_Purchase                15893 non-null datetime64[ns]
Number_of_Prior_Year_Transactions     15893 non-null float64
Do_Not_Direct_Mail_Solicit            15893 non-null float64
Do_Not_Email                          15893 non-null float64
Do_Not_Telemarket                     15893 non-null float64
Repurchase_Method                     15893 non-null object
Desk                                  15893 non-null float64
Executive_Chair                       15893 non-null float64
Standard_Chair                        15893 non-null float64
Monitor                               15893 non-null float64
Printer                               15893 non-null float64
Computer                              15893 non-null float64
Insurance                             15893 non-null float64
Toner                                 15893 non-null float64
Office_Supplies                       15893 non-null float64
Last_Transaction_Channel              15893 non-null object
Number_of_Employees                   15893 non-null object
Language                              15893 non-null object
Response                              15893 non-null float64
TenureYrs                             15893 non-null float64
ProductMix                            15893 non-null float64
Contact_Channel                       15893 non-null object
Avg_Sales_PU                          15893 non-null float64
PCA_1_Prt_Mon_Com_Chr                 15893 non-null float64
PCA_1_DM_Tele                         15893 non-null float64
PCA_2_Email                           15893 non-null float64
Est_Prob                              15893 non-null float32
dtypes: datetime64[ns](1), float32(1), float64(23), int64(2), object(5)
memory usage: 3.8+ MB
```

In [299]:
```
final_df_regression = proba_df.merge(small_df, left_on='Customer_Number', right
_on='Customer_Number', how='left')
```

In [300]:
```
final_df_regression.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15893 entries, 0 to 15892
Data columns (total 36 columns):
Aft_Propensity_Index                  15893 non-null int64
Original_Index                        15893 non-null int64
Customer_Number                       15893 non-null float64
Campaign_Period_Sales                 15893 non-null float64
Historical_Sales_Volume               15893 non-null float64
Date_of_First_Purchase                15893 non-null datetime64[ns]
Number_of_Prior_Year_Transactions     15893 non-null float64
Do_Not_Direct_Mail_Solicit            15893 non-null float64
Do_Not_Email                          15893 non-null float64
Do_Not_Telemarket                     15893 non-null float64
Repurchase_Method                     15893 non-null object
Desk                                  15893 non-null float64
Executive_Chair                       15893 non-null float64
Standard_Chair                        15893 non-null float64
Monitor                               15893 non-null float64
Printer                               15893 non-null float64
Computer                              15893 non-null float64
Insurance                             15893 non-null float64
Toner                                 15893 non-null float64
Office_Supplies                       15893 non-null float64
Last_Transaction_Channel              15893 non-null object
Number_of_Employees                   15893 non-null object
Language                              15893 non-null object
Response                              15893 non-null float64
TenureYrs                             15893 non-null float64
ProductMix                            15893 non-null float64
Contact_Channel                       15893 non-null object
Avg_Sales_PU                          15893 non-null float64
PCA_1_Prt_Mon_Com_Chr                 15893 non-null float64
PCA_1_DM_Tele                         15893 non-null float64
PCA_2_Email                           15893 non-null float64
Est_Prob_x                            15893 non-null float32
Est_Prob_y                            4308 non-null float32
log_act_y                             4308 non-null float64
log_hats_y                            4308 non-null float64
Est_Sale                              4308 non-null float64
dtypes: datetime64[ns](1), float32(2), float64(26), int64(2), object(5)
memory usage: 4.4+ MB
```

In [301]:
```
final_df_regression
```

Out[301]:

	Aft_Propensity_Index	Original_Index	Customer_Number	Campaign_Period_Sales	Historical_Sales_Vol
0	0	0	86734.000	238.705	146803
1	1	1	97098.000	281.680	439984
2	2	2	100836.000	432.857	970465
3	3	3	116390.000	0.000	230193
4	4	4	127914.000	1370.167	27403
...	...	...	...	...	
15888	16157	16167	166988514.000	0.000	701295
15889	16158	16168	167014041.000	0.000	2558801
15890	16159	16169	167077817.000	0.000	235503(
15891	16160	16170	167089540.000	0.000	58457(
15892	16161	16171	167235933.000	0.000	1949425

15893 rows × 36 columns

In [302]:
```
final_df_regression['Est_Prob_y'] = np.where(final_df_regression['Est_Prob_y'].
isna(),0.00,final_df_regression['Est_Prob_y'])
final_df_regression['log_act_y'] = np.where(final_df_regression['log_act_y'].is
na(),0.00,final_df_regression['log_act_y'])
final_df_regression['log_hats_y'] = np.where(final_df_regression['log_hats_y'].
isna(),0.00,final_df_regression['log_hats_y'])
final_df_regression['Est_Sale'] = np.where(final_df_regression['Est_Sale'].isna
(),0.00,final_df_regression['Est_Sale'])
```

In [304]:
```
final_df_regression = final_df_regression.drop('Est_Prob_y',axis=1)
```

In [308]:
```
final_df_regression = final_df_regression.rename(columns={'Est_Prob_x':'Est_Pro
b'})
```

In [309]:
```
final_df_regression.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15893 entries, 0 to 15892
Data columns (total 35 columns):
Aft_Propensity_Index                15893 non-null int64
Original_Index                      15893 non-null int64
Customer_Number                     15893 non-null float64
Campaign_Period_Sales               15893 non-null float64
Historical_Sales_Volume             15893 non-null float64
Date_of_First_Purchase              15893 non-null datetime64[ns]
Number_of_Prior_Year_Transactions   15893 non-null float64
Do_Not_Direct_Mail_Solicit          15893 non-null float64
Do_Not_Email                        15893 non-null float64
Do_Not_Telemarket                   15893 non-null float64
Repurchase_Method                   15893 non-null object
Desk                                15893 non-null float64
Executive_Chair                     15893 non-null float64
Standard_Chair                      15893 non-null float64
Monitor                             15893 non-null float64
Printer                             15893 non-null float64
Computer                            15893 non-null float64
Insurance                           15893 non-null float64
Toner                               15893 non-null float64
Office_Supplies                     15893 non-null float64
Last_Transaction_Channel            15893 non-null object
Number_of_Employees                 15893 non-null object
Language                            15893 non-null object
Response                            15893 non-null float64
TenureYrs                           15893 non-null float64
ProductMix                          15893 non-null float64
Contact_Channel                     15893 non-null object
Avg_Sales_PU                        15893 non-null float64
PCA_1_Prt_Mon_Com_Chr               15893 non-null float64
PCA_1_DM_Tele                       15893 non-null float64
PCA_2_Email                         15893 non-null float64
Est_Prob                            15893 non-null float32
log_act_y                           15893 non-null float64
log_hats_y                          15893 non-null float64
Est_Sale                            15893 non-null float64
dtypes: datetime64[ns](1), float32(1), float64(26), int64(2), object(5)
memory usage: 4.3+ MB
```

In [312]:
```
def profit_calculator(data):
    data['Exp_Profit'] = ((((0.22*data['Est_Prob']*data['Est_Sale']) - (8.40*dat
a['Est_Prob']) - 45.65))
    return data
```

In [313]:
```
profit_calculator(final_df_regression)
```

Out[313]:

	Aft_Propensity_Index	Original_Index	Customer_Number	Campaign_Period_Sales	Historical_Sales_Vol
0	0	0	86734.000	238.705	146803
1	1	1	97098.000	281.680	439984
2	2	2	100836.000	432.857	970465
3	3	3	116390.000	0.000	230193
4	4	4	127914.000	1370.167	27403
...	...	...	...	...	
15888	16157	16167	166988514.000	0.000	701295
15889	16158	16168	167014041.000	0.000	2558801
15890	16159	16169	167077817.000	0.000	235503(
15891	16160	16170	167089540.000	0.000	58457(
15892	16161	16171	167235933.000	0.000	1949425

15893 rows × 36 columns

In [314]:
```
final_df_regression.sort_values(by='Exp_Profit',ascending=False)
```

Out[314]:

	Aft_Propensity_Index	Original_Index	Customer_Number	Campaign_Period_Sales	Historical_Sales_Vol
12969	13179	13188	29676025.000	7693.400	1023222
2959	3006	3007	6807415.000	6421.000	107872(
6688	6798	6803	15215071.000	7382.700	996664
8458	8593	8598	19112091.000	7458.750	89505(
15270	15529	15538	53053980.000	6775.080	995936
...	...	...	...	...	
11614	11794	11801	26373322.000	0.000	231974
12729	12932	12941	29118237.000	0.000	106451(
673	682	682	1877429.000	0.000	94172
13056	13267	13276	29865170.000	0.000	779709
13794	14019	14028	31722303.000	0.000	264785(

15893 rows × 36 columns

Regression File Exportation

In [1015]:
```
X_batch1, X_batch2 = train_test_split(final_df_regression, stratify=y, test_si
ze = 0.50, random_state = 42)
```

In [1016]:
```
print(X_batch1.shape, X_batch2.shape)
```

```
(7946, 36) (7947, 36)
```

In [1017]:
```
file_path = '../PGDADS_Capstone Assignment/GBTregressionmodel.csv'
final_df_regression.to_csv(file_path)
print('Data exported successfully!')
```

```
Data exported successfully!
```

In [1018]:
```
file_path = '../PGDADS_Capstone Assignment/X_batch1regressionmodel.csv'
X_batch1.to_csv(file_path)
print('Data exported successfully!')
```

```
Data exported successfully!
```

In [1019]:
```
file_path = '../PGDADS_Capstone Assignment/X_batch2regressionmodel.csv'
X_batch2.to_csv(file_path)
print('Data exported successfully!')
```

```
Data exported successfully!
```

THE END