LI_BFSI_01 – LIFE INSURANCE SALES CAPSTONE FINAL REPORT

Prepared by

PARTHIBAN PALANISAMY

20-11-2022

Batch: G5

List of Contents

List of Tables

List of Equations

No table of figures entries found.

List of Figures

Introduction of the business problem

Problem statement

The major objective of this data set is to extract actionable insights from the leading life insurance company data and make strategic changes to make the company grow. Primary objective is to create Machine Learning models which correctly predicts the bonus for its agents so that it may provide information regarding high performing agents and low performing agents. Once a model is developed then it can extract actionable insights and recommendation, so based of which the company may design appropriate engagement activity and up skill programs for their agents as required.

Need of the study/project

Based on their agents to sell the policies, the insurance companies are heavily dependent on their success. So, it becomes very crucial to find and design engagement activity for their high performing agents giving them more and more incentives to keep up their performance and achieve more and also, up skill programs for their low performing agents to get better and perform better, and such that all together their agents are more able to sell the quality insurance to their customers and add more greater value to the company. And through this project with the help of data and its analysis help the insurance company to make data-driven business decisions. It empowers companies with high-level data and information that is leveraged into improved insurance processes and new opportunities.

Basically, the need of this data study here is Bonus prediction of the employees. Help the company to conduct proper skill engaging activities for well performing agents. Help the company to conduct proper upskill activities for underperforming agents. These programs will help the company to increase skilled employment.

Scope

To build machine learning regression model and choose the best model for the agent bonus by analyzing 18 different factors of customers data.

Constraints

The available data are customer centric information whereas agent related information is missing for prediction of Agent bonus. High dispersion of target column.

EDA and Business Implication

Univariate Analysis

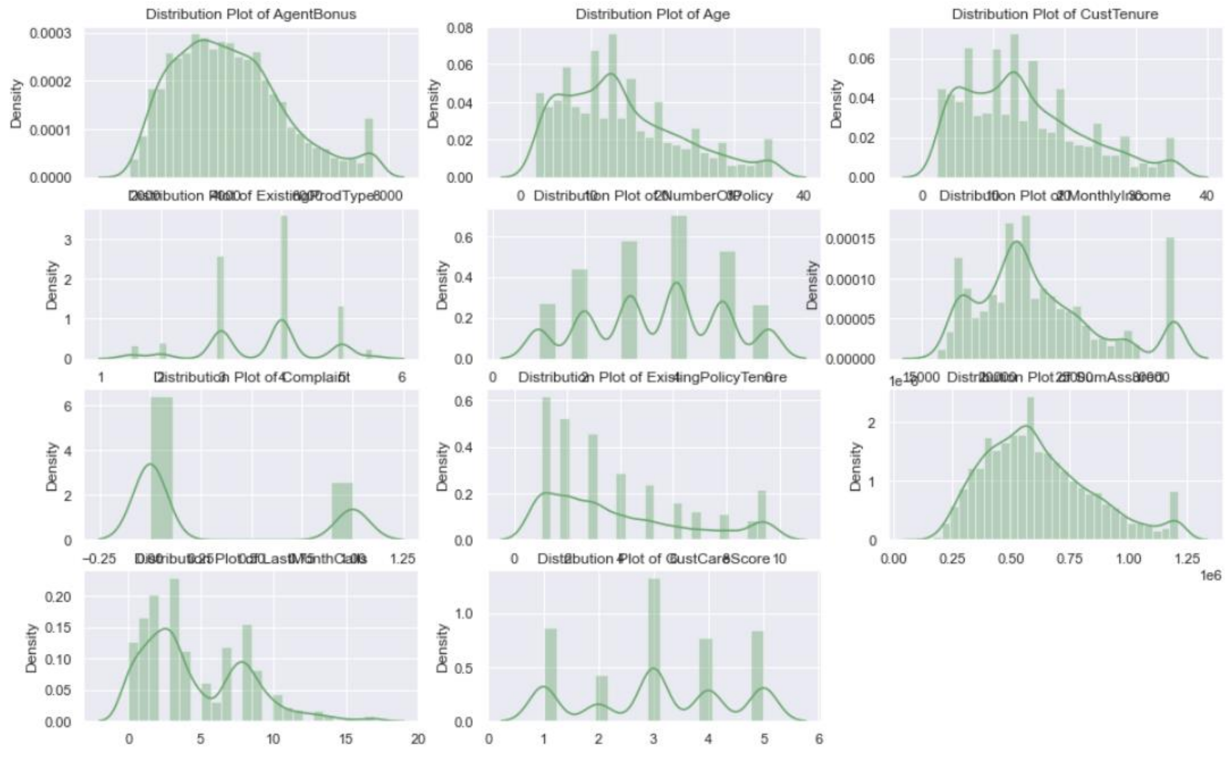

Figure 1 Univariate Analysis

Observation:

- Agentbonus, Age, cust Tenure, Sum Assured, customer Score plots are right skewed distributions.
- Complaint is discrete kind of data and 3 is the most frequent.
- Existing policy tenure is discrete kind of data and 6 is the frequent.
- LastMonthlyCalls is discrete kind of data with 2 peaks values in the range.

Bivariate analysis

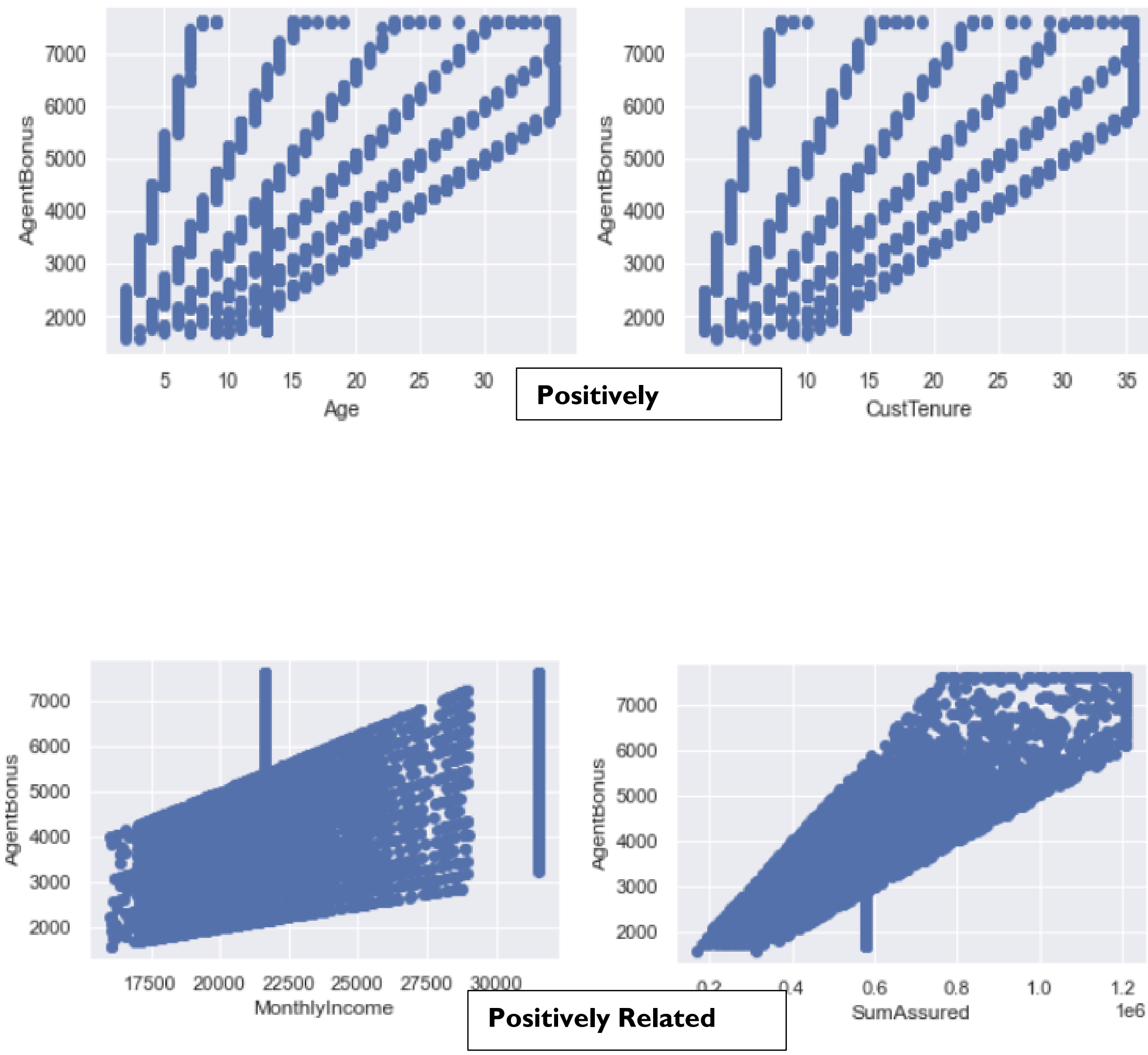

Figure 2 Bivariate analysis for correlated variables

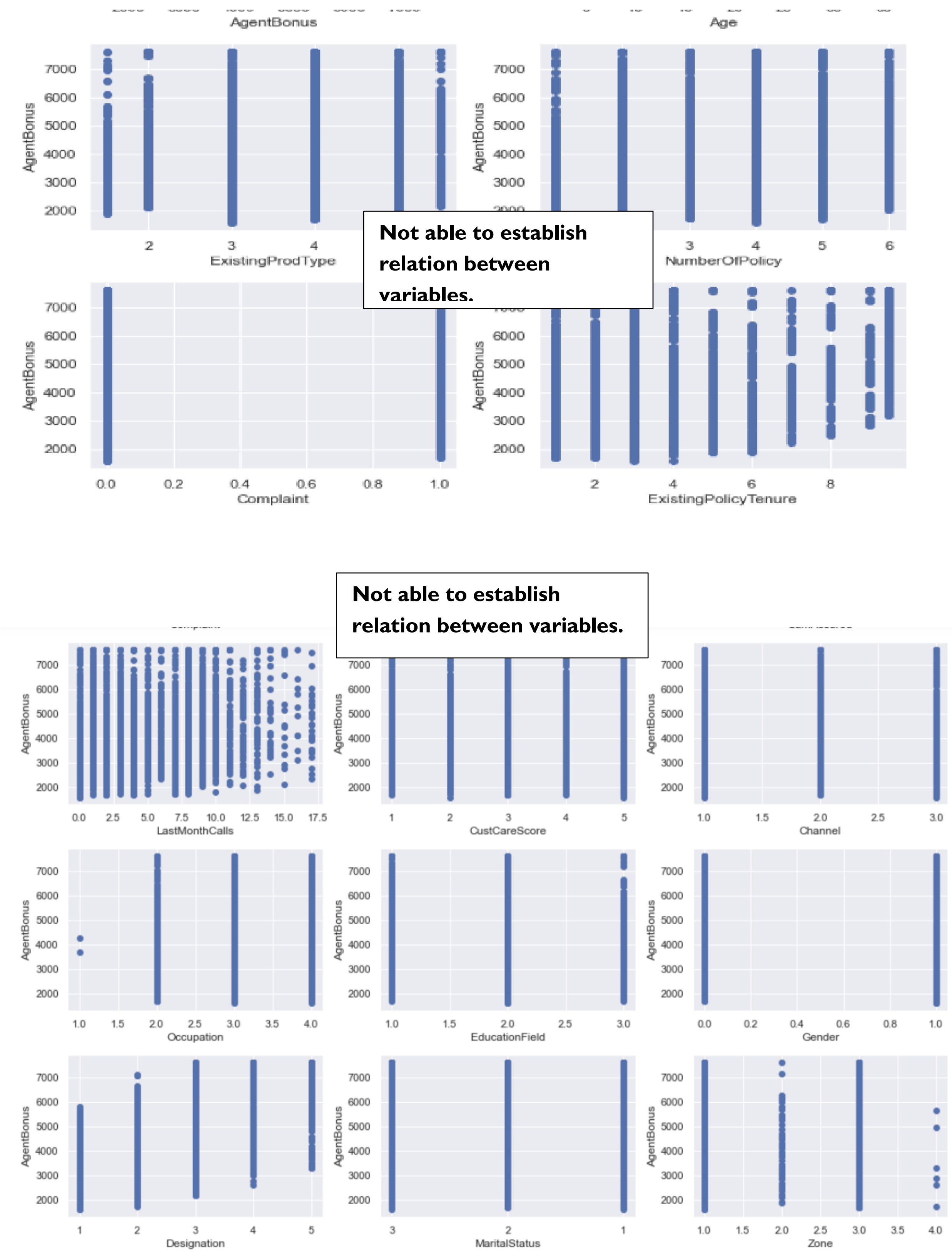

Figure 3 Bivariate analysis for non- corelated variables

Most of the variables don't seem to be related closely to each other which means there is low multi-collinearity in the data and each feature would have its importance in building the right model because of this we have not dropped any columns and would want to build the model to see the variable importance.

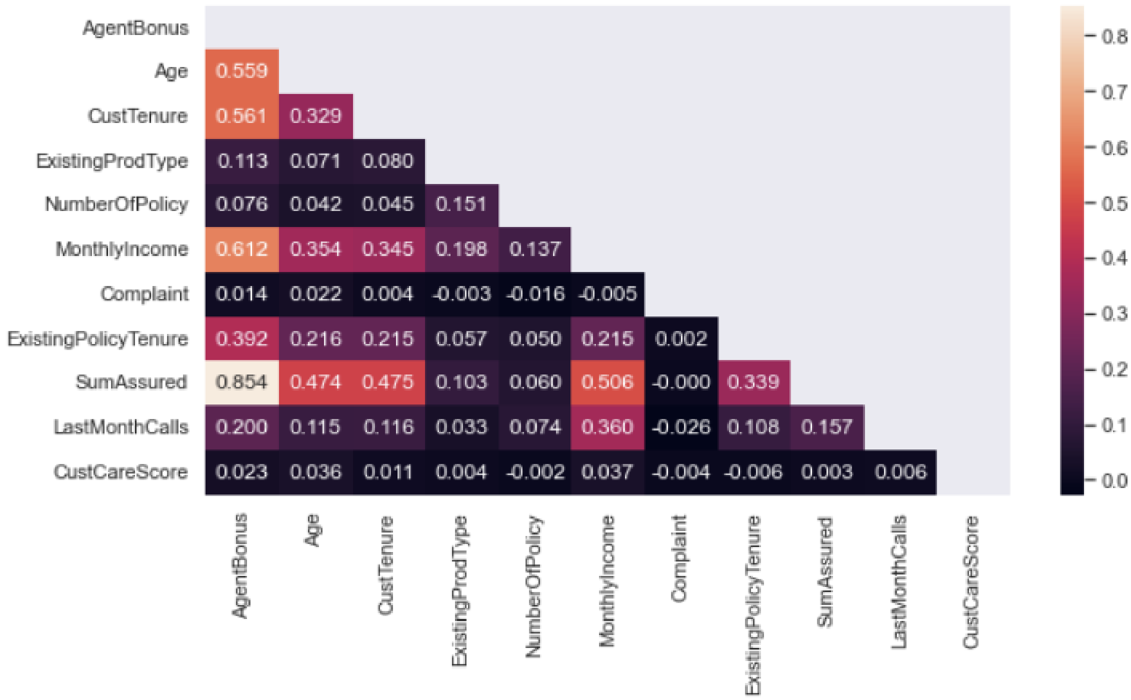

Figure 4 Heatmap for numerical variables.

Multivariate analysis

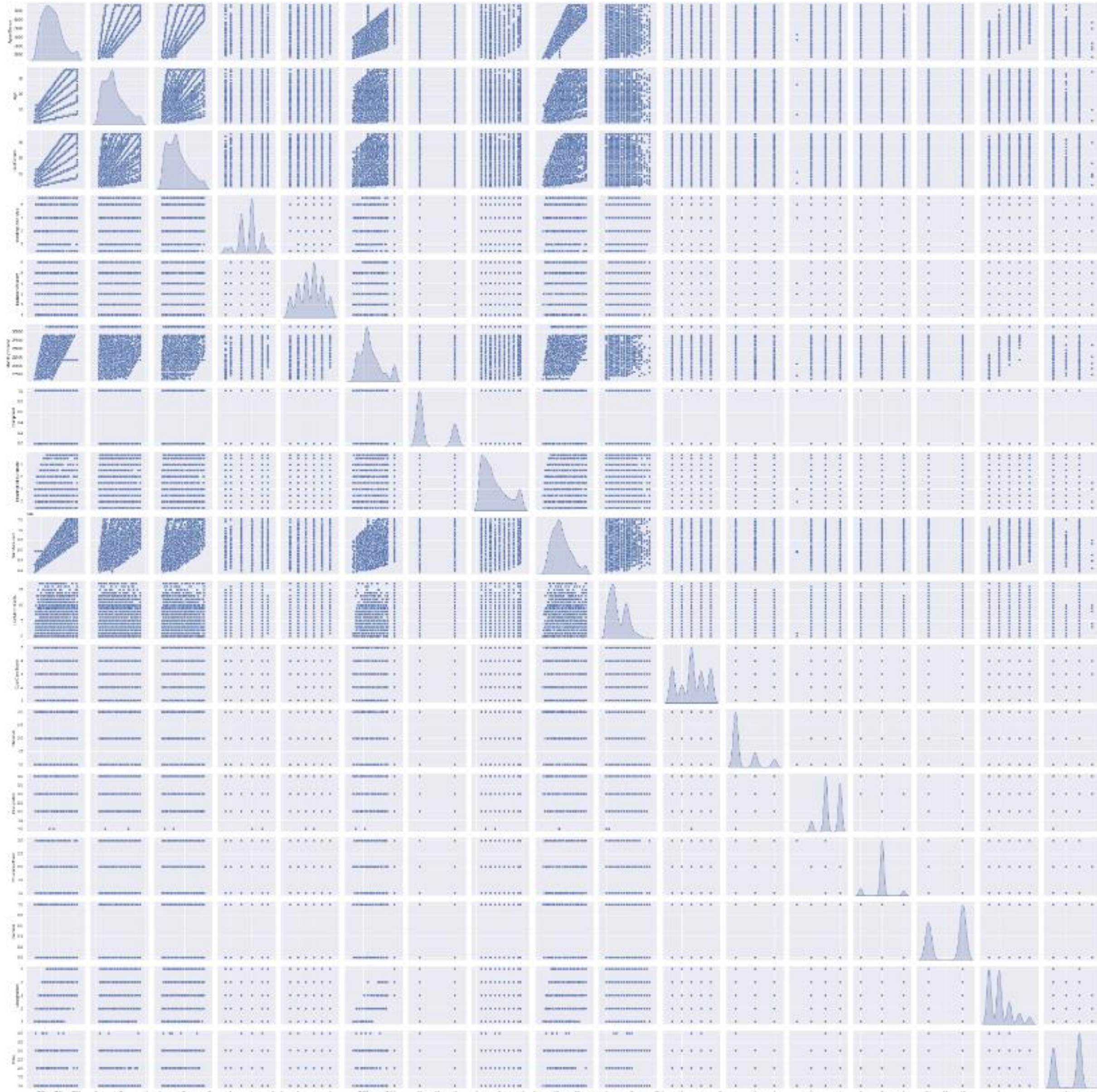

Figure 5 Pairplot

The pair plot also seems to suggest the same thing. But due to the huge number of columns pair plot was not providing very clear insight and hence resorted to bivariate plots with every combination possible.

From the above pair plot, we observe that there is less multicollinearity between the data. Some of the variables do have multicollinearity. This means some the data is not related to each other, and some of the data is significant individually. Hence all these columns should be used for building models.

Business insights

Figure 6 Catplot for AgentBonus Vs Channel

From the above cat plot, we observed that the more number agent bonus is through agent and second most is third-party partner. Online is very less in count.

Figure 7 Swarm plot for Channel vs Age

We observed that the more number of people brought premium through agent channel and second most is 3rd party and little least is in online.

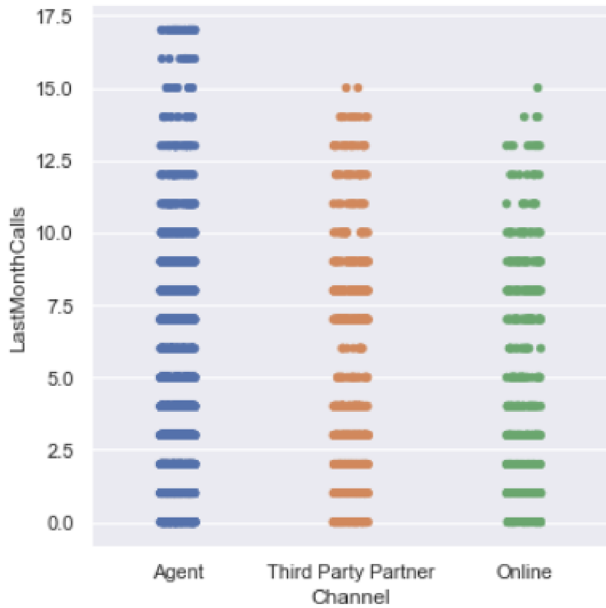

Figure 8 Cat plot for Channel vs Lastmonthcalls

From the above plot, most of the calls received to customers by agent channel and 3rd party, Online is little less comparatively to agent.

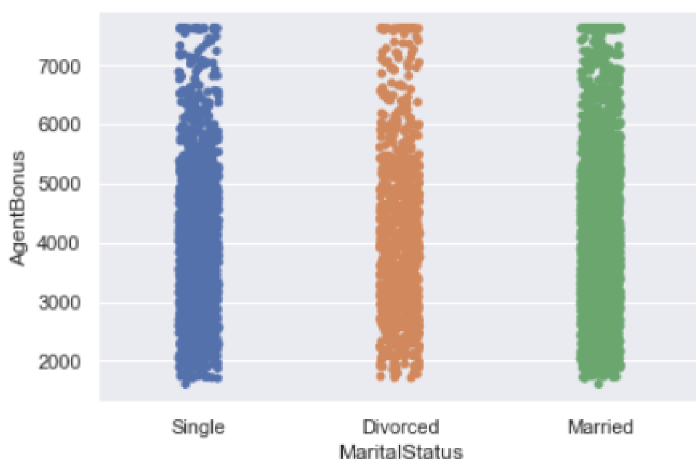

Figure 9 Stripplot for AgentBonus VS Marital status

From the above strip plot, first most is Married customers agents received more agent bonus and second most is single and third most is divorced.

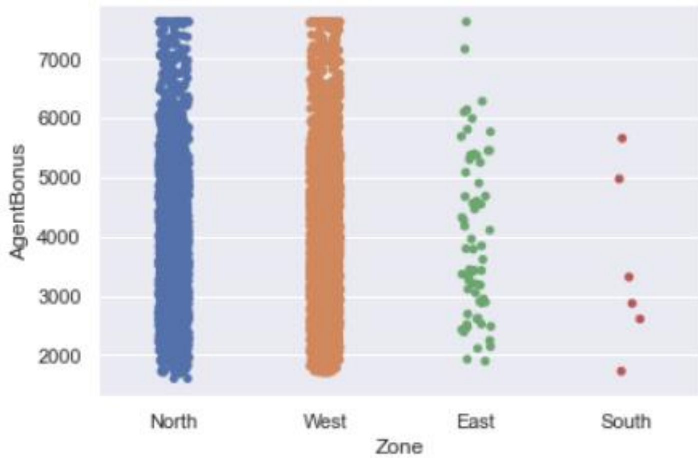

Figure 10 Strip plot for Zone vs Agentbonus

From the above strip plot, we observe that customers in north are high in number and contribute more towards bonus, followed by West in second, East with considerably very less in both and South with only five customers.

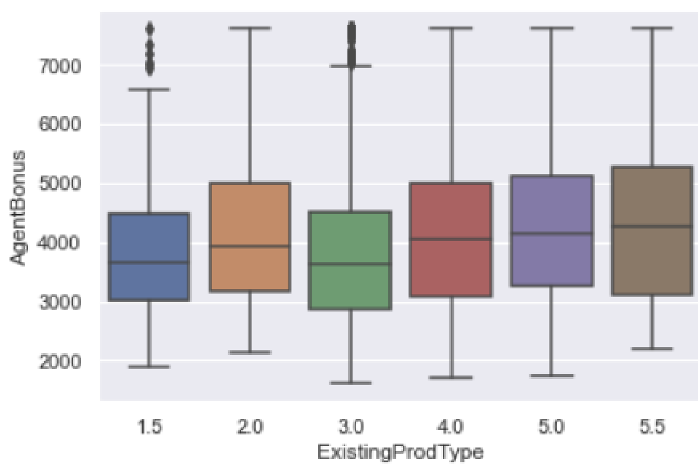

Figure 11 Box plot for AgentBonus vs Existing Prod type

From the above boxplot, customers having product type 6 contribute more towards agent bonus, next followed by 5, 4, 2, 3 and 1 respectively.

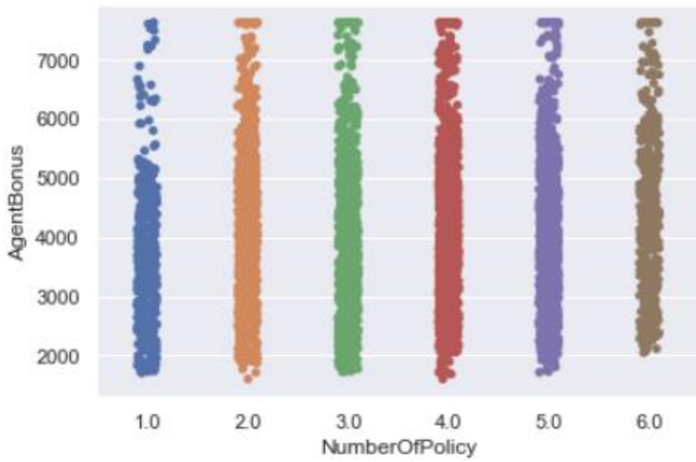

Figure 12 Strip plot for AgentBonus vs NumberOfPolicy

From the above plot, in the number of policies customers have, most customers have 4 policies and contribute more towards bonus, next comes customers with 3 policies, third comes customers with 5 policies followed customers with 2, 6 and 1 policy respectively.

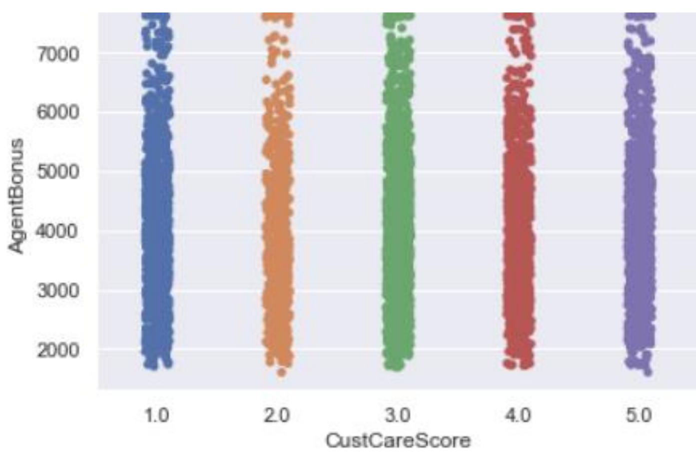

Figure 13 Strip plot for Agentbonus Vs CustcCareScore

From the above plot, we observed that the number 3.0 is high whose customer agents got more high bonus and second most is 5.0. And other three values little less but not low.

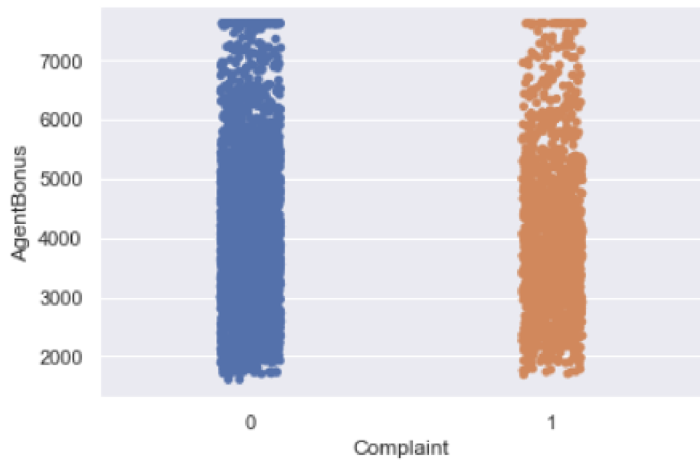

Figure 14 Strip plot for AgentBonus vs Complaint

From the above plot, we observed that the agent who got 0 complaint registered in a month by customer is receiving more agent bonus to agents and second most is 1.

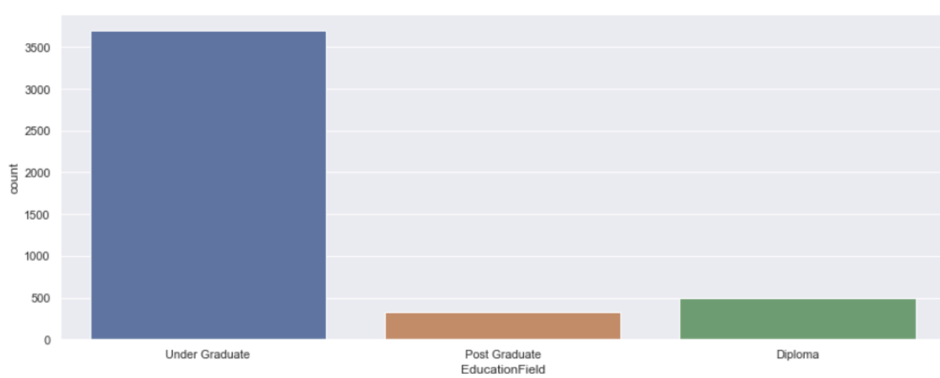

Figure 15 Education wise

Under Graduate customers are more contributors for purchasing insurance.

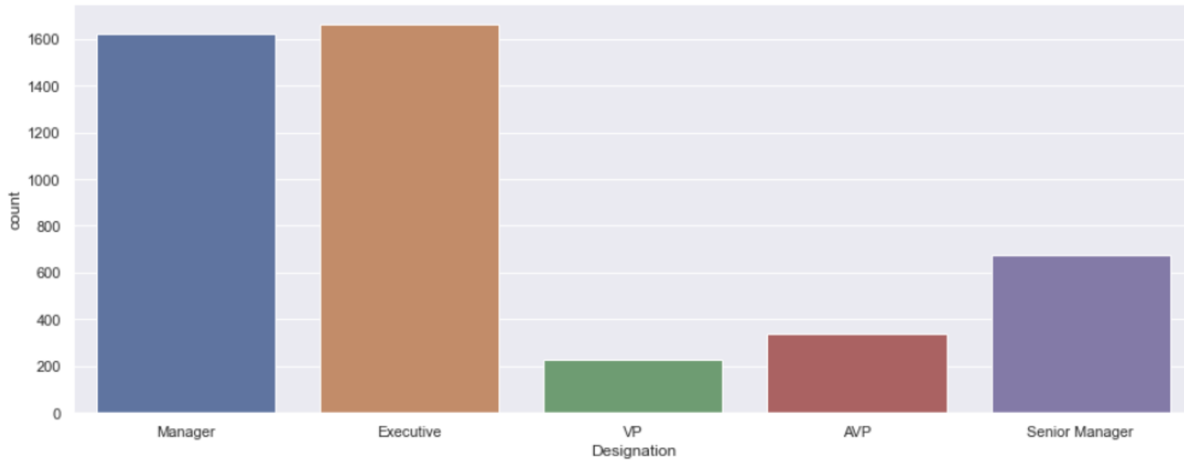

Figure 16 Designation wise

Executive and Manager are more contributors for purchasing insurance.

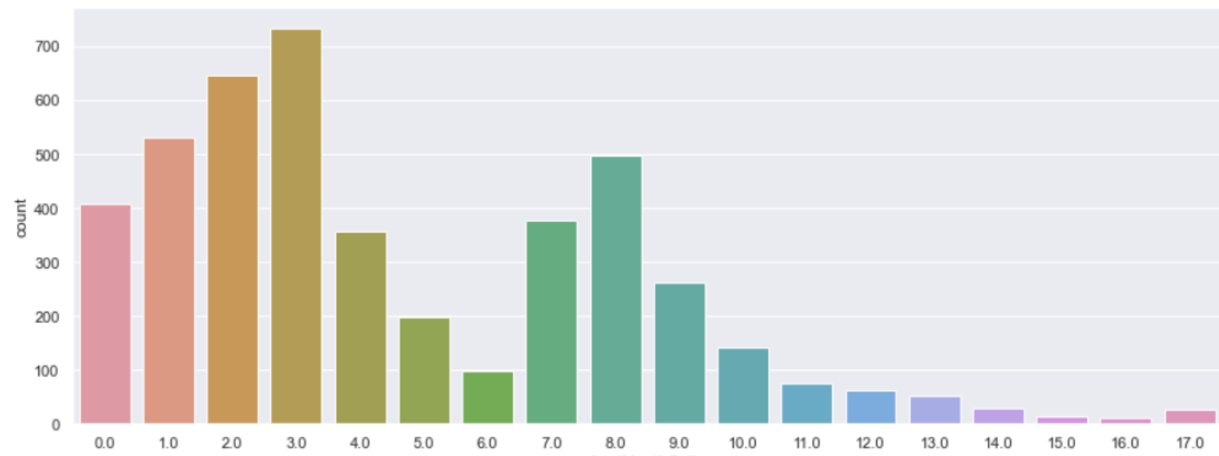

Figure 17 Lastmonth calls

Last month calls had happened 3 is top most and second most is 2.

Data Cleaning and Pre-processing

Removal of unwanted variables

In the dataset **CustID** and **PaymentMethod** are unique and not significant columns and thus have been removed. Choose not to remove any other columns and left to the model phase where the variable importance would be judged.

Missing Value treatment

```
CustID                  0
AgentBonus              0
Age                   269
CustTenure            226
Channel                 0
Occupation              0
EducationField          0
Gender                  0
ExistingProdType        0
Designation             0
NumberOfPolicy         45
MaritalStatus           0
MonthlyIncome         236
Complaint               0
ExistingPolicyTenure  184
SumAssured            154
Zone                    0
PaymentMethod           0
LastMonthCalls          0
CustCareScore          52
dtype: int64
```

Table 1 Missing Values of the columns

```
AgentBonus             0
Age                    0
CustTenure             0
ExistingProdType       0
NumberOfPolicy         0
MonthlyIncome          0
Complaint              0
ExistingPolicyTenure   0
SumAssured             0
LastMonthCalls         0
CustCareScore          0
dtype: int64
```

Table 2 Missing Values post treated

The missing values have been treated with most frequent values than median for numeric data including. The main reason of choosing median of the data is numeric and skewed for numeric columns.

Outlier treatment

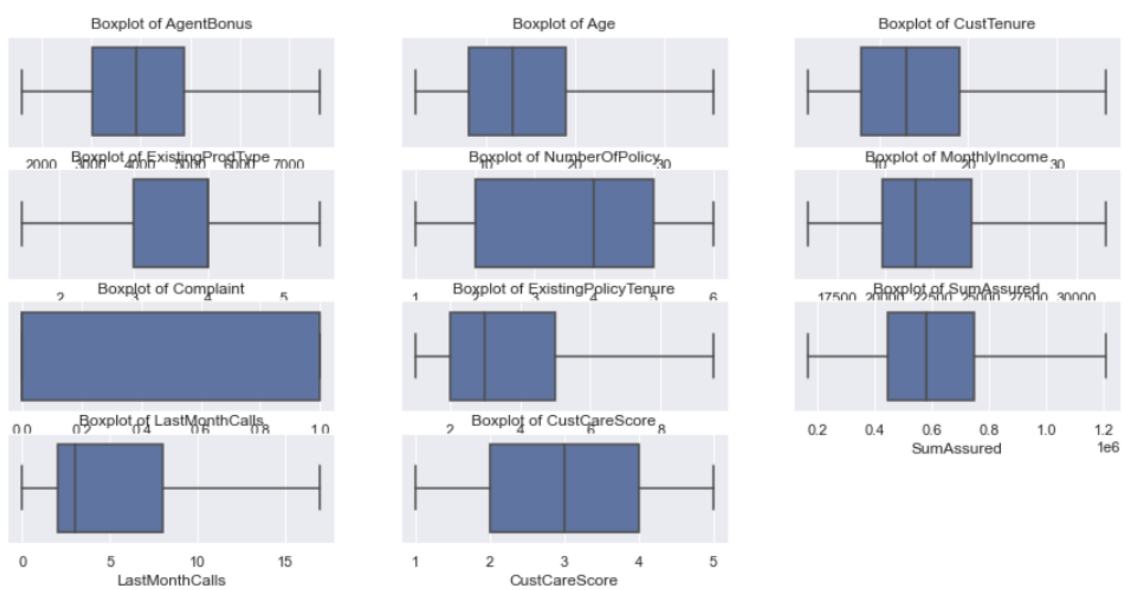

Figure 18 Post outlier treatment

We don’t see any outliers post treated.

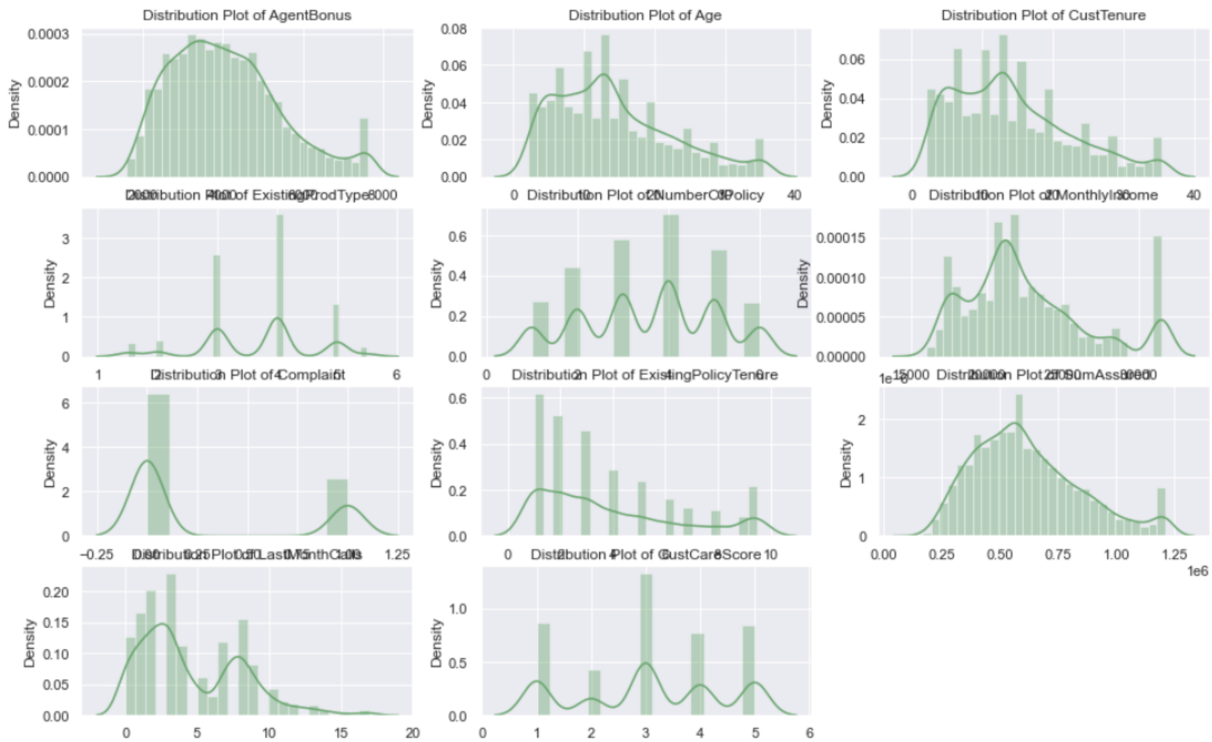

Figure 19 Dataset numerical variables post outlier treatment

We don’t see much impact in distribution post treating outliers.

Variable transformation

We do have categorical variables in the data hence we need to encode them into numeric variables.

	Channel	Occupation	EducationField	Gender	Designation	MaritalStatus	Zone
0	Agent	Salaried	Under Graduate	Female	Manager	Single	North
1	Third Party Partner	Salaried	Under Graduate	Male	Manager	Divorced	North
2	Agent	Free Lancer	Post Graduate	Male	Executive	Single	North
3	Third Party Partner	Salaried	Under Graduate	Female	Executive	Divorced	West
4	Agent	Small Business	Under Graduate	Male	Executive	Divorced	West

Table 3 Actual Categorical variables

	Channel	Occupation	EducationField	Gender	Designation	MaritalStatus	Zone
0	1	3	2	0	2	3	1
1	2	3	2	1	2	2	1
2	1	1	3	1	1	3	1
3	2	3	2	0	1	2	3
4	1	4	2	1	1	2	3

Table 4 Encoded Categorical variables

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4520 entries, 0 to 4519
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Channel         4520 non-null   int64
 1   Occupation      4520 non-null   int64
 2   EducationField  4520 non-null   int64
 3   Gender          4520 non-null   int64
 4   Designation     4520 non-null   int64
 5   MaritalStatus   4520 non-null   object
 6   Zone            4520 non-null   int64
dtypes: int64(6), object(1)
memory usage: 247.3+ KB
```

Table 5 Categorical dataset information

From the above two tables we observe that the categorical variables are encoded into numerical variables, which will be suitable for model building.

Model building

Clear on why was a particular model(s) chosen

Since it is regression problem, we have used list of models to build Regression models.

1. Linear Regression
2. Lasso Regression
3. Ridge Regression
4. KNN

5. ANN

Model1:

Linear Regression

Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

In the first iteration towards building linear regression model, we used all of the independent variables.

```
Index(['Age', 'CustTenure', 'ExistingProdType', 'NumberOfPolicy',
       'MonthlyIncome', 'Complaint', 'ExistingPolicyTenure', 'SumAssured',
       'LastMonthCalls', 'CustCareScore', 'Channel', 'Occupation',
       'EducationField', 'Gender', 'Designation', 'MaritalStatus', 'Zone'],
      dtype='object')
```

Intercept Value:

```
The intercept for our model is -0.0038166911956584995
```

Coefficient values:

	Feature	Coefficients
7	SumAssured	0.602073
1	CustTenure	0.141159
0	Age	0.137074
4	MonthlyIncome	0.118705
14	Designation	0.092926
6	ExistingPolicyTenure	0.081893
15	MaritalStatus	0.016609
3	NumberOfPolicy	0.014511
5	Complaint	0.011107
9	CustCareScore	0.008641
13	Gender	0.007380
10	Channel	0.003341
12	EducationField	0.003058
11	Occupation	-0.000649
16	Zone	-0.005106
8	LastMonthCalls	-0.007953
2	ExistingProdType	-0.009031

Table 6 Coefficients

From the above coefficient values, we could see the top 4 significant variables to use build model building such as **SumAssuresd, CustTenure, Age and MonthlyIncome.**

Train and Test RMSE values:

```
Linear Regression RMSE training data  0.44591533249562926
Linear Regression RMSE testing data  0.45960450380308937
```

Table 7 Linear Regression RMSE values

Train and Test Scores:

```
The Regression model train Score: 0.8012537647340943

The Regression model test Score: 0.7825826284866033
```

Table 8 Linear Regression R^2 scores

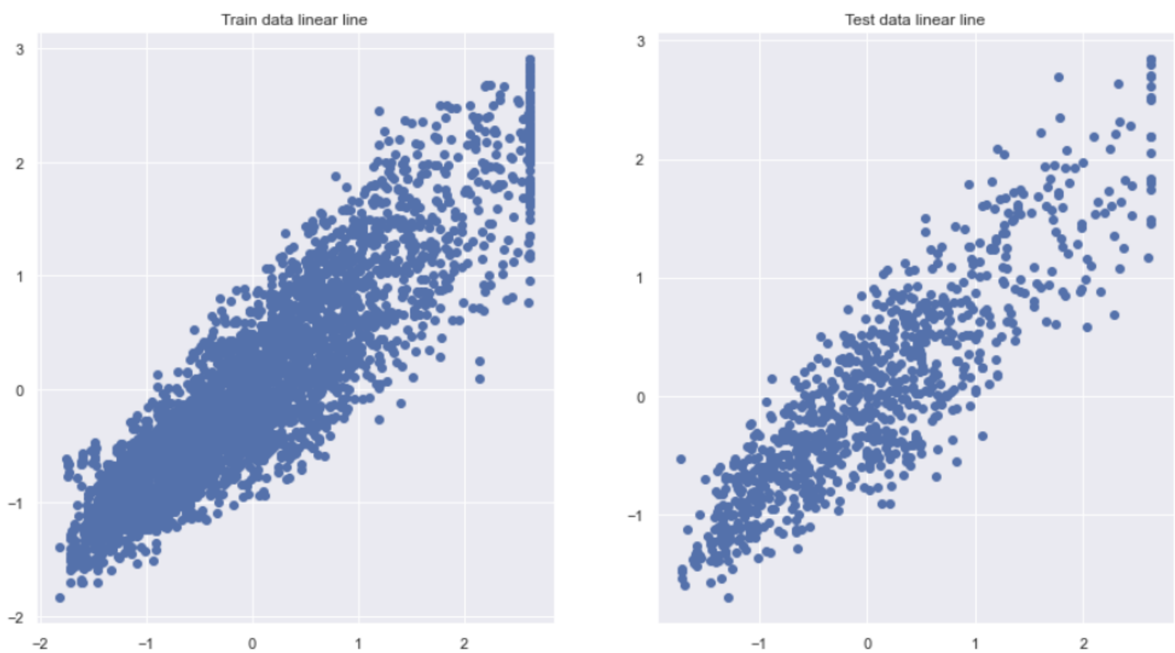

Figure 20 Train and Test linear line

LM1 Summary:

Variation Inflation Factor:

Multicollinearity can be detected using various techniques, one such technique being the **Variance Inflation Factor** (**VIF**).

```
Age ---> 1.306343801235912
CustTenure ---> 1.3081005869598255
ExistingProdType ---> 1.1319129544674138
NumberOfPolicy ---> 1.064813795320549
MonthlyIncome ---> 3.360382018629411
Complaint ---> 1.0035460655620618
ExistingPolicyTenure ---> 1.1061057357321789
SumAssured ---> 1.7125267913080493
LastMonthCalls ---> 1.1776227025654817
CustCareScore ---> 1.0069733280806898
Channel ---> 1.0057618349447914
Occupation ---> 1.1556122831988924
EducationField ---> 1.163518849164721
Gender ---> 1.014339342394037
Designation ---> 3.2230202200578293
MaritalStatus ---> 1.0175584589155833
Zone ---> 1.010381040885463
```

Table 9 VIF

As we can see, **Designation** and **MonthlyIncome** have very high values of VIF, indicating that these two variables are highly correlated. We can drop these variables and generate the L2 summary results again.

LM Summary:

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:             AgentBonus   R-squared:                       0.775
Model:                            OLS   Adj. R-squared:                  0.774
Method:                 Least Squares   F-statistic:                     825.0
Date:                Sat, 29 Oct 2022   Prob (F-statistic):               0.00
Time:                        10:47:20   Log-Likelihood:                -2442.4
No. Observations:                3616   AIC:                             4917.
Df Residuals:                    3600   BIC:                             5016.
Df Model:                          15                                         
Covariance Type:            nonrobust                                         
========================================================================================
                           coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
Intercept               -0.0039      0.008     -0.498      0.619      -0.019       0.012
Age                      0.1580      0.009     17.488      0.000       0.140       0.176
CustTenure               0.1603      0.009     17.719      0.000       0.143       0.178
Complaint                0.0101      0.008      1.268      0.205      -0.006       0.026
ExistingPolicyTenure     0.0752      0.008      9.017      0.000       0.059       0.092
ExistingProdType         0.0060      0.008      0.736      0.462      -0.010       0.022
SumAssured               0.6670      0.010     68.132      0.000       0.648       0.686
Channel                 -0.0011      0.008     -0.133      0.894      -0.017       0.014
NumberOfPolicy           0.0231      0.008      2.865      0.004       0.007       0.039
Occupation               0.0050      0.009      0.584      0.559      -0.012       0.022
EducationField          -0.0071      0.009     -0.821      0.412      -0.024       0.010
Gender                   0.0092      0.008      1.155      0.248      -0.006       0.025
MaritalStatus            0.0058      0.008      0.718      0.473      -0.010       0.021
Zone                    -0.0014      0.008     -0.177      0.860      -0.017       0.014
CustCareScore            0.0147      0.008      1.843      0.065      -0.001       0.030
LastMonthCalls           0.0503      0.008      6.259      0.000       0.035       0.066
==============================================================================
Omnibus:                      241.759   Durbin-Watson:                   2.004
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              313.522
Skew:                           0.610   Prob(JB):                     8.31e-69
Kurtosis:                       3.771   Cond. No.                         2.09
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Table 10 L2 Summary

- The above results R^2 and adj R^2 is not much difference even after dropping independent variables.
- From above p-value and coefficient we can conclude that there are some variables having significant p-value i.e., p-value<0.05 and higher coefficient value. These variables have significant influence on the prediction of Agent bonus

Model 2

Ridge Regression

Ridge Regression is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.

```
Ridge model: [[ 0.15797611  0.16026059  0.00595121  0.02307195  0.01008895  0.07517539
   0.66699371  0.0502581   0.01467367 -0.00105136  0.00500293 -0.0070708
   0.00919496  0.00575271 -0.0014052 ]]
```

Table 11 Ridge coefficient Model

```
Ridge Train: 0.7746477309254053
Ridge Test: 0.7567599766565156
```

Table 12 Ridge scores

The above results tell us that it gets good metrices but let's try other also.

Model 3

Lasso Regression

```
Lasso model: [ 0.10461929  0.10603694  0.          0.          0.          0.00547267
  0.64582182  0.          0.         -0.          0.         -0.
 -0.         -0.         -0.        ]
```

Table 13 Lasso Coeff model

```
Lasso Train: 0.7496025132002047
Lasso Test: 0.730431686877846
```

Table 14 Lasso Scores

On comparing both Ridge and Lasso regressions, Ridge is good model. Let's try other algorithms as well.

Model 4

k-nearest neighbors

The goal of the k-nearest neighbor algorithm is to identify the nearest neighbors of a given query point, so that we can assign a class label to that point.

In order to determine which data points are closest to a given query point, the distance between the query point and the other data points will need to be calculated. These distance metrics help to form decision boundaries, which partitions query points into different regions.

```
KNeighborsRegressor()
```

```
The KNN RMSE train: 0.49043481007836387
The KNN RMSE test: 0.6200312184329655
```

Table 15 KNN RMSE

```
The KNN train score is: 0.7602194370065416
The KNN test score is: 0.6102026994801534
```

Table 16 KNN Scores

As per the above observation, the metric tells that higher RMSE for both Train and Test then little lower the score values. This model cannot be fit.

Model 5

Random Forest

The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement, called the bootstrap sample.

```
RandomForestRegressor(random_state=1)
```

```
The RF RMSE train: 0.1479793709074061
The RF RMSE test: 0.3995135489321969
```

Table 17 Random Forest RMSE

```
The RF train score is: 0.978169999127187
The RF test score is: 0.8381641479967743
```

Table 18 Random Forest Score

The above result tells the train score is good but test score is little difference and need to explore the more model.

Model 6

Artificial Neural Network

Neural Network is a series of algorithms that are trying to mimic the human brain and find the relationship between the sets of data.

```
MLPRegressor(activation='logistic', hidden_layer_sizes=(20, 20), max_iter=1000)
```

```
ANN RMSE train: 0.4295924197313411
ANN RMSE test: 0.4576963405693563
```

Table 19 ANN RMSE

```
The Ann train score is: 0.8160225399909837
The Ann test score is: 0.7875940911716729
```

Table 20 ANN Score

As per the above observation, Train and Test scores are better but RMSE is little high. We need to do some tunning for the above models.

	Train RMSE	Test RMSE	Train Mape	Test Mape	Training Score	Test Score
Linear Regression	0.474375	0.488065	1.873268	4.562135	0.775082	0.761496
KNeighborsRegressor	0.489178	0.619898	1.752559	6.281935	0.760826	0.615247
Random Forest Regressor	0.149670	0.405643	0.668296	2.969166	0.977610	0.835249
ANN Regressor	0.477344	0.490564	2.026788	4.998988	0.772258	0.759047
Lasso Regression	0.500291	0.513149	1.760362	4.369116	0.749835	0.736350
Ridge Regression	0.474375	0.488063	1.873142	4.561692	0.775082	0.761497

Table 21 RMSE, MAPE and R^2 Scores without tunning

As we can see the above table, Random Forest MAPE is less error while comparing other models. We will do hyperparameter tunning to see whether it is gives best score.

Effort to improve model performance.

Hyperparameter tuning is the best method to determine optimal model as it try many different combinations to evaluate performance of the model.

Random Forest Grid Search Best Parameter:

A model has to match the business objectives hence various permutation and combination has been carried on to refine the model.

```
GridSearchCV(cv=3, estimator=RandomForestRegressor(random_state=123),
             param_grid={'max_depth': [7, 10], 'max_features': [5, 7],
                         'min_samples_leaf': [3, 10, 25],
                         'min_samples_split': [30, 50, 100],
                         'n_estimators': [250, 500]})
```

KNN Grid search Best Parameter:

```
GridSearchCV(cv=5, estimator=KNeighborsRegressor(),
             param_grid={'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9]})
```

ANN Grid Search Best Parameter:

```
GridSearchCV(cv=3, estimator=MLPRegressor(max_iter=10000, random_state=123),
             param_grid={'activation': ['tanh', 'relu', 'logistic'],
                         'hidden_layer_sizes': [500, (100, 100)],
                         'solver': ['sgd', 'adam']})
```

```
{'activation': 'relu', 'hidden_layer_sizes': 500, 'solver': 'sgd'}
```

Gradient Boosting Grid Search Best Parameter:

```
GridSearchCV(cv=3, estimator=GradientBoostingRegressor(random_state=100),
             param_grid={'learning_rate': [0.01, 1.0], 'max_depth': [7, 10],
                         'min_samples_split': [5, 10],
                         'n_estimators': [500, 1000]})
```

```
{'learning_rate': 0.01,
 'max_depth': 7,
 'min_samples_split': 10,
 'n_estimators': 500}
```

AdaBoosting Grid Search Best Parameter:

```
AdaBoostRegressor(n_estimators=200, random_state=1)
```

	Train RMSE	Test RMSE	Train Mape	Test Mape	Training Score	Test Score
KNN Regressor Gridsearch	0.526592	0.602661	2.008631	4.376970	0.722841	0.636347
Random Forest Regressor Gridsearch	0.356992	0.415353	1.582283	2.651499	0.872621	0.827266
ANN Regressor Gridsearch	0.442777	0.473439	1.937007	4.478961	0.804047	0.775576
Gradient Boosting	0.264278	0.403339	1.114021	3.369419	0.930192	0.837115
Ada Boosting	0.503975	0.515588	2.708170	6.439419	0.746137	0.733838

Table 22 RMSE, MAPE and R^2 scores hyperparameter tunning

- **Linear Regression, Lasso Regression, Ridge Regression** models are good for R^2 scores but RMSE values are high hence we cannot conclude this model.
- **KNN** model is underfitting and RMSE value also high.
- **Random Forest** model is overfit model without tunning and greater differences for RMSE values.

- **ANN** model is good with and without Tuning but RMSE value is high on both.
- But among these three model **Random Forest with Grid search** has the highest R squared or model score value, whereas lowest RMSE value than all other model. That means, predicted value is closer to actual value. Random forest grid search tuning is performing well for both training and test data.
- Hence **Random Forest** with hyper parameter tuning using Grid search is the best optimum model.
- The Insurance company can use this model for prediction of Agent Bonus.

Model validation

	Train RMSE	Test RMSE	Train Mape	Test Mape	Training Score	Test Score
Linear Regression	0.474375	0.488065	1.873268	4.562135	0.775082	0.761496
KNeighborsRegressor	0.489178	0.619898	1.752559	6.281935	0.760826	0.615247
Random Forest Regressor	0.149670	0.405643	0.668296	2.969166	0.977610	0.835249
ANN Regressor	0.479851	0.493697	2.066262	5.339615	0.769859	0.755959
Lasso Regression	0.500291	0.513149	1.760362	4.369116	0.749835	0.736350
Ridge Regression	0.474375	0.488063	1.873142	4.561692	0.775082	0.761497
KNN Regressor Gridsearch	0.526592	0.602661	2.008631	4.376970	0.722841	0.636347
Random Forest Regressor Gridsearch	0.356992	0.415353	1.582283	2.651499	0.872621	0.827266
ANN Regressor Gridsearch	0.442777	0.473439	1.937007	4.478961	0.804047	0.775576
Gradient Boosting	0.264278	0.403339	1.114021	3.369419	0.930192	0.837115
Ada Boosting	0.503975	0.515588	2.708170	6.439419	0.746137	0.733838

Table 23 Model Metrics

From the above metrics, Random Forest Grid Search RMSE value is little less and MAPE percentage also less while comparing other models. R^2 is good score. Hence we consider the Random Forest Grid Search model is best optimum model.

R Square: It determines how much of the total variation in Y (dependent variable) is explained by the variation in X (independent variable). Higher value R Squared is better fit.

RMSE: It is defined as the square root of the average squared error. RMSE is an absolute measure of fit. RMSE can be interpreted as the standard deviation of the unexplained variance. Lower values of RMSE indicate better fit.

MAPE: MAPE is the sum of the individual absolute errors divided by the demand (each period separately). It is the average of the percentage errors which expresses accuracy as a percentage of the error. Lower values of MAPE indicate better fit.

Final interpretation / recommendation

- Agent is the primary channel, company should take all necessary steps to retain high performing Agent, as they are the key factor for the company Growth.
- To increase the number of Female customers they must introduce special scheme to attract them by the way agent bonus will get increase
- To attract younger customer, introduce long term policy having higher sum assured value with lower premium by attracting offers which leads agent bonus will be paid high.
- To increase sale in South and East zone, they should find out the pain area and can also design Upskill program for the agent of these zone to enhance their skill so the agent bonus will get increase.
- Company must introduce new schemes for Diploma customers with less premium and good benefits and for PG customers whom they must introduce new scheme with best policies irrespective of price by the way agent bonus will get increase.
- They must introduce higher sum assured value with lower premium policy to them as they have average monthly income by the way agent bonus will get increase.
- VP and AVP customers have higher monthly income. They have potential to buy good policy and to increase the sale among them. Agent should understand their needs and requirement and suggest scheme accordingly which says the agent bonus will be paid high.
- Agents must concentrate on customers who is having less existing policies and make them purchase more number of policies with lowest premium by getting more offers which will increase the sales as well by the way agent bonus will pe paid high.

Appendix

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(color_codes=True)
color = sns.color_palette()
import warnings
warnings.filterwarnings("ignore")
import math

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from scipy.stats import zscore
from sklearn import metrics
import statsmodels.formula.api as smf
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import BaggingRegressor, AdaBoostRegressor, GradientBoostingRegressor
```

Checking the data

```
df.head()
```

```
df.tail()
```

Find out unique values in each categorical column

```
df["Occupation"].unique()
```

```
df["EducationField"].unique()
```

```
df["Gender"].unique()
```

```
df["MaritalStatus"].unique()
```

```
df["Designation"].unique()
```

Making spelling changes and update in dataframe

```
df['Occupation'] = df['Occupation'].replace(to_replace='Laarge Business', value='Large Business')
df['EducationField'] = df['EducationField'].replace(to_replace='UG', value='Under Graduate')
df['EducationField'] = df['EducationField'].replace(to_replace='Graduate', value='Under Graduate')
df['EducationField'] = df['EducationField'].replace(to_replace='Engineer', value='Under Graduate')
df['EducationField'] = df['EducationField'].replace(to_replace='MBA', value='Post Graduate')
df['Gender'] = df['Gender'].replace(to_replace='Fe male', value='Female')
df['MaritalStatus'] = df['MaritalStatus'].replace(to_replace='Unmarried', value='Single')
df['Designation'] = df['Designation'].replace(to_replace='Exe', value='Executive')
```

Missing Values

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

```
df.shape
```

Checking the categorical value counts

```
df.Channel.value_counts()
```

```
df.EducationField.value_counts()
```

```
df.Gender.value_counts()
```

```
df.Designation.value_counts()
```

```
df.MaritalStatus.value_counts()
```

```
df.Zone.value_counts()
```

```
df.PaymentMethod.value_counts()
```

Summary of data

```
df.describe().T
```

Checking Duplicate values

```
dups = df.duplicated()
print('Number of duplicate rows = %d' % (dups.sum()))
df[dups]
```

Dropping customer id column

```
df = df.drop(['CustID','PaymentMethod'], axis=1)
```

```
df.head()
```

```
df_numercial = df[['AgentBonus','Age','CustTenure','ExistingProdType','NumberOfPolicy','MonthlyIncome','Complaint','ExistingPoli
```

```
df_numercial
```

```
for column in df_numercial.columns:
    if df_numercial[column].dtype != 'object':
        median = df_numercial[column].median()
        df_numercial[column] = df_numercial[column].fillna(median)

df_numercial.isnull().sum()
```

```
plt.figure(figsize = (15,7))
feature = df_numercial.columns
for i in range(len(feature)):
    plt.subplot(4,3,i+1)
    sns.boxplot(x=df_numercial[feature[i]], data=df_numercial, color='b');
    plt.title('Boxplot of {}'.format(feature[i]))
```

```
plt.figure(figsize=(16,10))
feature = df_numercial.columns
for i in range(len(feature)):
    plt.subplot(4,3,i+1)
    sns.distplot(x=df_numercial[feature[i]], color='g', kde=True)
    plt.title('Distribution Plot of {}'.format(feature[i]))
```

Treating outliers

```
def remove_outlier(col):
    sorted(col)
    Q1,Q3=col.quantile([0.25,0.75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

```
Outlier = ['AgentBonus','Age','CustTenure','ExistingProdType','NumberOfPolicy','MonthlyIncome','ExistingPolicyTenure','SumAssurec
for i in Outlier:
    LL,UL = remove_outlier(df_numercial[i])
    df_numercial[i]=np.where(df_numercial[i]>UL,UL,df_numercial[i])
    df_numercial[i]=np.where(df_numercial[i]<LL,LL,df_numercial[i])
```

```
plt.figure(figsize = (15,7))
feature = df_numercial.columns
for i in range(len(feature)):
    plt.subplot(4,3,i+1)
    sns.boxplot(x=df_numercial[feature[i]], data=df_numercial, color='b');
    plt.title('Boxplot of {}'.format(feature[i]))
```

```
plt.figure(figsize=(16,10))
feature = df_numercial.columns
for i in range(len(feature)):
    plt.subplot(4,3,i+1)
    sns.distplot(x=df_numercial[feature[i]], color='g', kde=True)
    plt.title('Distribution Plot of {}'.format(feature[i]))
```

```
df_numercial.head()
```

```
df_numercial.tail()
```

```
df_cat = df[['Channel','Occupation','EducationField','Gender','Designation','MaritalStatus','Zone']]
```

Splitting categorical datasets

```
df_categorical = df[['Channel','Occupation','EducationField','Gender','Designation','MaritalStatus','Zone']]
```

```
df_categorical.head()
```

Converting object into categorical codes

```
df_categorical['Channel']=np.where(df_categorical['Channel'] =='Agent', '1', df_categorical['Channel'])
df_categorical['Channel']=np.where(df_categorical['Channel'] =='Third Party Partner', '2', df_categorical['Channel'])
df_categorical['Channel']=np.where(df_categorical['Channel'] =='Online', '3', df_categorical['Channel'])
```

```
df_categorical['Occupation']=np.where(df_categorical['Occupation'] =='Free Lancer', '1', df_categorical['Occupation'])
df_categorical['Occupation']=np.where(df_categorical['Occupation'] =='Large Business', '2', df_categorical['Occupation'])
df_categorical['Occupation']=np.where(df_categorical['Occupation'] =='Salaried', '3', df_categorical['Occupation'])
df_categorical['Occupation']=np.where(df_categorical['Occupation'] =='Small Business', '4', df_categorical['Occupation'])
```

```
df_categorical['EducationField']=np.where(df_categorical['EducationField'] =='Diploma', '1', df_categorical['EducationField'])
df_categorical['EducationField']=np.where(df_categorical['EducationField'] =='Under Graduate', '2', df_categorical['EducationFie]
df_categorical['EducationField']=np.where(df_categorical['EducationField'] =='Post Graduate', '3', df_categorical['EducationFielc
```

```
df_categorical['Gender']=np.where(df_categorical['Gender'] =='Female', '0', df_categorical['Gender'])
df_categorical['Gender']=np.where(df_categorical['Gender'] =='Male', '1', df_categorical['Gender'])
```

```
df_categorical['Designation']=np.where(df_categorical['Designation'] =='Executive', '1', df_categorical['Designation'])
df_categorical['Designation']=np.where(df_categorical['Designation'] =='Manager', '2', df_categorical['Designation'])
df_categorical['Designation']=np.where(df_categorical['Designation'] =='Senior Manager', '3', df_categorical['Designation'])
df_categorical['Designation']=np.where(df_categorical['Designation'] =='AVP', '4', df_categorical['Designation'])
df_categorical['Designation']=np.where(df_categorical['Designation'] =='VP', '5', df_categorical['Designation'])
```

```
df_categorical['MaritalStatus']=np.where(df_categorical['MaritalStatus'] =='Married', '1', df_categorical['MaritalStatus'])
df_categorical['MaritalStatus']=np.where(df_categorical['MaritalStatus'] =='Divorced', '2', df_categorical['MaritalStatus'])
df_categorical['MaritalStatus']=np.where(df_categorical['MaritalStatus'] =='Single', '3', df_categorical['MaritalStatus'])
```

```
df_categorical['Zone']=np.where(df_categorical['Zone'] =='North', '1', df_categorical['Zone'])
df_categorical['Zone']=np.where(df_categorical['Zone'] =='East', '2', df_categorical['Zone'])
df_categorical['Zone']=np.where(df_categorical['Zone'] =='West', '3', df_categorical['Zone'])
df_categorical['Zone']=np.where(df_categorical['Zone'] =='South', '4', df_categorical['Zone'])
```

```
df_categorical.head()
```

```
df_categorical.info()
```

```
df_categorical['Channel'] = df_categorical['Channel'].astype('int64')
df_categorical['Occupation'] = df_categorical['Occupation'].astype('int64')
df_categorical['EducationField'] = df_categorical['EducationField'].astype('int64')
df_categorical['Gender'] = df_categorical['Gender'].astype('int64')
df_categorical['Designation'] = df_categorical['Designation'].astype('int64')
df_categorical['Zone'] = df_categorical['Zone'].astype('int64')
df_categorical['MaritalStatus'] = df_categorical['MaritalStatus'].astype('int64')
```

```
df_categorical.info()
```

```
df1 = pd.concat([df_numercial,  df_categorical], axis=1)
df1.head()
```

Univariate Analysis

```
plt.figure(figsize=(16,10))
feature = df1.columns
for i in range(len(feature)):
    plt.subplot(5,4,i+1)
    sns.distplot(x=df1[feature[i]], color='g', kde=True)
    plt.title('Distribution Plot of {}'.format(feature[i]))
```

```
plt.figure(figsize= (15,10))
plt.subplot(3,1,1)
sns.boxplot(x= df_categorical.Channel, color='lightblue')
```

```
plt.figure(figsize= (15,10))
plt.subplot(3,1,1)
sns.boxplot(x= df_categorical.EducationField, color='lightblue')
```

```
plt.figure(figsize= (15,10))
plt.subplot(3,1,1)
sns.boxplot(x= df_categorical.Gender , color='lightblue')
```

```
plt.figure(figsize= (15,10))
plt.subplot(3,1,1)
sns.boxplot(x= df_categorical.Occupation , color='lightblue')
```

```
plt.figure(figsize= (15,10))
plt.subplot(3,1,1)
sns.boxplot(x= df_numercial.NumberOfPolicy , color='lightblue')
```

```
df2 = pd.concat([df_numercial,  df_cat], axis=1)
df2.head()
```

Bivariate analysis

```
sns.catplot(x='Channel', y='AgentBonus', data=df2)
```

```
sns.swarmplot(df_cat['Channel'], df1['Age']);
```

```
sns.catplot(x='Channel', y='LastMonthCalls', data=df2)
```

```
sns.stripplot(df2['MaritalStatus'], df2['AgentBonus'], jitter=True);
```

```
sns.catplot('EducationField', data=df2, kind='count', aspect=2.5)
```

```
sns.catplot('Designation', data=df2, kind='count', aspect=2.5)
```

```
sns.catplot('LastMonthCalls', data=df2, kind='count', aspect=2.5)
```

```
sns.swarmplot(df_cat['Channel'], df1['Age']);
```

```
sns.barplot(df1['Channel'], df1['MonthlyIncome'], hue=df_cat['Gender']);
```

```
sns.countplot(df_cat['Channel'],hue=df_cat['Designation']);
```

```
sns.boxplot(x='Occupation', y='AgentBonus', data=df1)
```

```
sns.boxplot(x='ExistingProdType', y='AgentBonus', data=df1)
```

```
sns.stripplot(df_cat['Zone'], df2['AgentBonus'], jitter=True);
```

```
sns.stripplot(df2['CustCareScore'], df2['AgentBonus'], jitter=True);
```

```
sns.stripplot(df2['NumberOfPolicy'], df2['AgentBonus'], jitter=True);
```

HeatMap

```
mask = np.zeros_like(df1.corr()) #Creates an array of the same size as df.corr()
mask[np.triu_indices_from(mask)] = True #Returns the indices of the Upper Triangle (if you use tril_indices, it will return Lower
plt.figure(figsize = (10,5))
sns.heatmap(df1.corr(), annot=True,fmt='.3f',mask=mask);
```

```
mask = np.zeros_like(df.corr()) #Creates an array of the same size as df.corr()
mask[np.triu_indices_from(mask)] = True #Returns the indices of the Upper Triangle (if you use tril_indices, it will return Lower
plt.figure(figsize = (10,5))
sns.heatmap(df.corr(), annot=True,fmt='.3f',mask=mask);
```

```
#mask = np.zeros_like(df_LinReg.corr()) #Creates an array of the same size as df.corr()
#mask[np.triu_indices_from(mask)] = True #Returns the indices of the Upper Triangle (if you use tril_indices, it will return Lowe
plt.figure(figsize = (10,5))
sns.heatmap(df.corr(), annot=True,fmt='.3f');
```

```
sns.pairplot(df1,diag_kind='kde')
plt.show()
```

```
plt.figure(figsize=(14,21))
for i in range(len(df1.columns)):
    plt.subplot(7, 3, i+1)
    #sns.scatterplot(y=df.AgentBonus,x=df1[col_val])
    plt.scatter(df1[df1.columns[i]], df1['AgentBonus'])
    plt.xlabel(df1.columns[i])
    plt.ylabel("AgentBonus")

plt.tight_layout()
```

```
plt.figure(figsize=(15,10))
sns.heatmap(df1.corr(), annot=True, fmt=".1f",square=False)
```

```
df_scaled = df1.apply(zscore)
```

```
X = df_scaled.drop('AgentBonus', axis=1)

y = df_scaled[['AgentBonus']]
```

```
X.head()
```

```
y.head()
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30 , random_state=1)
```

```
display(X_train.shape)
display(y_train.shape)
display(X_test.shape)
display(y_test.shape)
```

```
print(X_train.columns)
```

```
print(y_train.columns)
```

```
print(X_test.columns)
```

```
print(y_test.columns)
```

LinearRegression

```
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)
```

```
intercept=regression_model.intercept_[0]

print("The intercept for our model is {}".format(intercept))
```

```
#for idx, col_name in enumerate(X_train.columns):
#    print("The coefficient for {} is {}".format(col_name, regression_model.coef_[0][idx]))

coefficients = pd.DataFrame({"Feature":X_train.columns,"Coefficients": np.transpose(regression_model.coef_[0])})
coefficients.sort_values("Coefficients", axis=0, ascending=False)
```

```
#R^2 Training and Test data
print('The Regression model train Score:',regression_model.score(X_train, y_train))
print('')
print('The Regression model test Score:',regression_model.score(X_test, y_test))
```

```
#Training MSE
mse = np.mean((regression_model.predict(X_train)-y_train)**2)
math.sqrt(mse)
```

```
#Test MSE
mse = np.mean((regression_model.predict(X_test)-y_test)**2)
math.sqrt(mse)
```

```
ytrain_predict = regression_model.predict(X_train)
ytest_predict = regression_model.predict(X_test)
```

```
#RMSE on Training data
predicted_train=regression_model.fit(X_train, y_train).predict(X_train)
print('Linear Regression RMSE training data ',np.sqrt(metrics.mean_squared_error(y_train,ytrain_predict)))
```

```
#RMSE on Testing data
predicted_test=regression_model.fit(X_train, y_train).predict(X_test)
print('Linear Regression RMSE testing data ',np.sqrt(metrics.mean_squared_error(y_test,ytest_predict)))
```

```
figure, axis = plt.subplots(1, 2)
axis[0].scatter(y_train['AgentBonus'], ytrain_predict)
axis[0].set_title("Train data linear line")
axis[1].scatter(y_test['AgentBonus'], ytest_predict)
axis[1].set_title("Test data linear line")

figure.set_figheight(8)
figure.set_figwidth(15)
```

```
# concatenate X and y into a single dataframe
data_train = pd.concat([X_train, y_train], axis=1)
data_test=pd.concat([X_test,y_test],axis=1)
data_train.head()
```

```
data_test.head()
```

```
exp ='AgentBonus ~ Age + CustTenure + ExistingProdType + NumberOfPolicy + MonthlyIncome + Complaint + ExistingPolicyTenure + Sum
```

```
lm1 = smf.ols(formula= exp, data = data_train).fit()
lm1.params
```

```
print(lm1.summary())
```

```
vif = [variance_inflation_factor(X.values, ix) for ix in range(X.shape[1])]
```

```
i=0
for column in X.columns:
    if i < 18:
        print (column ,"--->",  vif[i])
        i = i+1
```

```
exp1 ='AgentBonus ~ Age + CustTenure + ExistingProdType + NumberOfPolicy + Complaint + ExistingPolicyTenure + SumAssured + Channe
```

```
lm2 = smf.ols(formula= exp1, data = data_train).fit()
lm2.params
```

```
print(lm2.summary())
```

```
X_train = X_train.drop(['Designation', 'MonthlyIncome'], axis=1)
X_test = X_test.drop(['Designation', 'MonthlyIncome'], axis=1)
```

```
ridge = Ridge(alpha=.3)
ridge.fit(X_train,y_train)
print ("Ridge model:", (ridge.coef_))
```

```
lasso = Lasso(alpha=0.1)
lasso.fit(X_train,y_train)
print ("Lasso model:", (lasso.coef_))
```

```
print(ridge.score(X_train, y_train))
print(ridge.score(X_test, y_test))
```

```
print(lasso.score(X_train, y_train))
print(lasso.score(X_test, y_test))
```

KNN Model

```
knnmodel=KNeighborsRegressor(n_neighbors=5, weights='uniform')
knnmodel.fit(X_train, y_train)
```

```
# Calculate MSE train

from sklearn.metrics import mean_squared_error
ypred_train = knnmodel.predict(X_train)
knn_mse_train = (mean_squared_error(y_train,ypred_train))
print('The train MSE KNN:',knn_mse_train)

# Calculate MSE test

ypred_test = knnmodel.predict(X_test)
knn_mse_test = (mean_squared_error(y_test,ypred_test))
print('The test MSE KNN:',knn_mse_test)
```

```
# Calculate RMSE train

knn_rmse_train = math.sqrt(knn_mse_train)
print('The KNN RMSE train:',knn_rmse_train)

# Calculate RMSE test

knn_rmse_test = math.sqrt(knn_mse_test)
print('The KNN RMSE test:',knn_rmse_test)
```

```
# R square train

knn_r2_train=knnmodel.score(X_train,y_train)
print('The KNN train score is:',knn_r2_train)

# R square test

knn_r2_test=knnmodel.score(X_test,y_test)
print('The KNN test score is:',knn_r2_test)
```

Random Forest

```
Randomregressor = RandomForestRegressor(n_estimators = 100, random_state = 1)
Randomregressor.fit(X_train,y_train)
```

```
ypred_train = Randomregressor.predict(X_train)
ypred_test = Randomregressor.predict(X_test)
```

```
RF_mse_train = (mean_squared_error(y_train,ypred_train))
print('Random Forest train MSE:',RF_mse_train)

RF_mse_test = (mean_squared_error(y_test,ypred_test))
print('Random Forest test MSE :',RF_mse_test)
```

```
# Calculate RMSE train

RF_rmse_train = math.sqrt(RF_mse_train)
print('The RF RMSE train:',RF_rmse_train)

# Calculate RMSE test

RF_rmse_test = math.sqrt(RF_mse_test)
print('The RF RMSE test:',RF_rmse_test)
```

```
# R square train

RF_r2_train=Randomregressor.score(X_train,y_train)
print('The RF train score is:',RF_r2_train)

# R square test

RF_r2_test=Randomregressor.score(X_test,y_test)
print('The RF test score is:',RF_r2_test)
```

Artificial Neural Network (ANN)

```
Annmodel=MLPRegressor(hidden_layer_sizes=(20,20,), activation='logistic', max_iter=1000)
```

```
Annmodel.fit(X_train, y_train)
```

```
ypred_train = Annmodel.predict(X_train)
ypred_test = Annmodel.predict(X_test)
```

```
Ann_mse_train = (mean_squared_error(y_train,ypred_train))
print('ANN train MSE:',Ann_mse_train)

Ann_mse_test = (mean_squared_error(y_test,ypred_test))
print('ANN test MSE :',Ann_mse_test)
```

```
# Calculate RMSE train

Ann_rmse_train = math.sqrt(Ann_mse_train)
print('ANN RMSE train:',Ann_rmse_train)

# Calculate RMSE test

Ann_rmse_test = math.sqrt(Ann_mse_test)
print('ANN RMSE test:',Ann_rmse_test)
```

```
# R square train

Ann_r2_train=Annmodel.score(X_train,y_train)
print('The Ann train score is:',Ann_r2_train)

# R square test

Ann_r2_test=Annmodel.score(X_test,y_test)
print('The Ann test score is:',Ann_r2_test)
```

Making 6 models

```
ann = MLPRegressor(hidden_layer_sizes=(20,20,), activation='logistic', max_iter=1000)
rfr = RandomForestRegressor(random_state=123)
Knn = KNeighborsRegressor(n_neighbors=5, weights='uniform')
regression_model = LinearRegression()
lasso = Lasso(alpha=0.1)
ridge = Ridge(alpha=.3)

models=[regression_model,Knn,rfr,ann,lasso,ridge]

rmse_train=[]
rmse_test=[]
scores_train=[]
scores_test=[]
mape_train=[]
mape_test=[]
for i in models:  # we are scaling the data for ANN. Without scaling it will give very poor results. Computations becomes easier
        i.fit(X_train,y_train)
        scores_train.append(i.score(X_train, y_train))
        scores_test.append(i.score(X_test, y_test))
        rmse_train.append(np.sqrt(mean_squared_error(y_train,i.predict(X_train))))
        rmse_test.append(np.sqrt(mean_squared_error(y_test,i.predict(X_test))))
        mape_train.append(mean_absolute_percentage_error(y_train,i.predict(X_train)))
        mape_test.append(mean_absolute_percentage_error(y_test,i.predict(X_test)))

a = pd.DataFrame({'Train RMSE': rmse_train,'Test RMSE': rmse_test,'Train Mape':mape_train,'Test Mape':mape_test,'Training Score':
            index=['Linear Regression','KNeighborsRegressor','Random Forest Regressor', 'ANN Regressor','Lasso Regression','Ridge
a
```

RandomForest Using Grid search

```
param_grid = {
    'max_depth': [7,10],
    'max_features': [5, 7],
    'min_samples_leaf': [3, 10,25],
    'min_samples_split': [30, 50,100],
    'n_estimators': [250, 500]
}

rfrg = RandomForestRegressor(random_state=123)

grid_search = GridSearchCV(estimator = rfrg, param_grid = param_grid, cv = 3)
```

```
grid_search.fit(X_train,y_train)
```

```
print(grid_search.best_params_)
```

ANN using Grid search

```
param_grid = {
    'hidden_layer_sizes':[(500),(100,100)],
    # keeping these simple because it would take too much time to run on low-end computers
    "activation": ["tanh", "relu","logistic"],
    "solver": ["sgd", "adam"]}

annrg = MLPRegressor(max_iter=10000, random_state=123)

grid_search = GridSearchCV(estimator = annrg, param_grid = param_grid, cv = 3)
```

```
grid_search.fit(X_train,y_train)
```

```
print(grid_search.best_params_)
```

KNN using Gridsearch

```
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}
knng = KNeighborsRegressor()

knnGridsearch = GridSearchCV(knng, params, cv=5)
```

```
knnGridsearch.fit(X_train,y_train)
```

```
knnGridsearch.best_params_
```

```
anng = MLPRegressor(activation='relu', hidden_layer_sizes=(500),
                    solver='sgd',random_state=123,
                    max_iter=10000)
rfrg = RandomForestRegressor(max_depth=10, max_features=7,
                            min_samples_leaf= 3,
                            min_samples_split= 30, n_estimators= 500,
                            random_state=123)
knng = KNeighborsRegressor(n_neighbors= 9)
models=[knng,rfrg,anng]

rmse_train=[]
rmse_test=[]
scores_train=[]
scores_test=[]
mape_train=[]
mape_test=[]

for i in models:
        i.fit(X_train, y_train)
        scores_train.append(i.score(X_train, y_train))
        scores_test.append(i.score(X_test, y_test))
        rmse_train.append(np.sqrt(mean_squared_error(y_train,i.predict(X_train))))
        rmse_test.append(np.sqrt(mean_squared_error(y_test,i.predict(X_test))))
        mape_train.append(mean_absolute_percentage_error(y_train,i.predict(X_train)))
        mape_test.append(mean_absolute_percentage_error(y_test,i.predict(X_test)))



a1 = pd.DataFrame({'Train RMSE': rmse_train,'Test RMSE': rmse_test,'Train Mape': mape_train,'Test Mape': mape_test,'Training Scor
            index=['KNN Regressor Gridsearch','Random Forest Regressor Gridsearch', 'ANN Regressor Gridsearch'])
a1
```

Gradient Boosting

```
gbr_params = {'n_estimators': [500, 1000],
          'max_depth': [7,10],
          'min_samples_split': [5, 10],
          'learning_rate': [0.01, 1.0]
             }

grad = GradientBoostingRegressor(random_state= 100)

grid_search_grad = GridSearchCV(estimator = grad, param_grid = gbr_params, cv= 3)
```

```
grid_search_grad.fit(X_train, y_train)
```

```
grid_search_grad.best_params_
```

```
gB_train_score = grid_search_grad.score(X_train, y_train)
print('Train score:',gB_train_score)
gB_test_score = grid_search_grad.score(X_test, y_test)
print('Test Score:',gB_test_score)
```

```
ypred_train = grid_search_grad.predict(X_train)
ypred_test = grid_search_grad.predict(X_test)
```

```
GBoost_mse_train = (mean_squared_error(y_train,ypred_train))
print('GradBoosting train MSE:',GBoost_mse_train)

GBoost_mse_test = (mean_squared_error(y_test,ypred_test))
print('GradBoosting test MSE :',GBoost_mse_test)
```

```
# Calculate RMSE train

GBoost_rmse_train = math.sqrt(GBoost_mse_train)
print('GBoosting RMSE train:',GBoost_rmse_train)

# Calculate RMSE test

GBoost_rmse_test = math.sqrt(GBoost_mse_test)
print('GBoosting RMSE test:',GBoost_rmse_test)
```

```
GBoost_mape_train = mean_absolute_percentage_error(y_train,ypred_train)
print('GBoost train MAPE :',GBoost_mape_train)

GBoost_mape_test = mean_absolute_percentage_error(y_test,ypred_test)
print('GBoost train MAPE :',GBoost_mape_test)
```

```
gb = pd.DataFrame({'Train RMSE': GBoost_rmse_train,'Test RMSE': GBoost_rmse_test,'Train Mape': GBoost_mape_train,'Test Mape': GBo
            index=['Gradient Boosting'])
gb
```

AdaBoosting

```
ada = AdaBoostRegressor(n_estimators=200, learning_rate=1.0, random_state= 1)
```

```
ada.fit(X_train, y_train)
```

```
ada_train_score = ada.score(X_train, y_train)
print('Train score:',ada_train_score)
ada_test_score = ada.score(X_test, y_test)
print('Test Score:',ada_test_score)
```

```
ypred_train = ada.predict(X_train)
ypred_test = ada.predict(X_test)

adaBoost_mse_train = (mean_squared_error(y_train,ypred_train))
print('GradBoosting train MSE:',adaBoost_mse_train)

adaBoost_mse_test = (mean_squared_error(y_test,ypred_test))
print('GradBoosting test MSE :',adaBoost_mse_test)

# Calculate RMSE train

adaBoost_rmse_train = math.sqrt(adaBoost_mse_train)
print('GBoosting RMSE train:',adaBoost_rmse_train)

# Calculate RMSE test

adaBoost_rmse_test = math.sqrt(adaBoost_mse_test)
print('GBoosting RMSE test:',adaBoost_rmse_test)
```

```
adaBoost_mape_train = mean_absolute_percentage_error(y_train,ypred_train)
print('adaBoost train MAPE :',adaBoost_mape_train)

adaBoost_mape_test = mean_absolute_percentage_error(y_test,ypred_test)
print('adaBoost train MAPE :',adaBoost_mape_test)
```

```
ada = pd.DataFrame({'Train RMSE': adaBoost_rmse_train,'Test RMSE': adaBoost_rmse_test,'Train Mape': adaBoost_mape_train,'Test Map
            index=['Ada Boosting'])
ada
```

```
tunning = pd.concat([a1,gb,ada],axis=0)
tunning
```

Interpretation models metrices

```
concat_ds = pd.concat([a,a1,gb,ada],axis=0)
concat_ds
```

Sample table:

	CustID	AgentBonus	Age	CustTenure	Channel	Occupation	EducationField	Gender	ExistingProdType	Designation	NumberOfPolicy	MaritalStatus	Montl
0	7000000	4409	22.0	4.0	Agent	Salaried	Graduate	Female	3	Manager	2.0	Single	
1	7000001	2214	11.0	2.0	Third Party Partner	Salaried	Graduate	Male	4	Manager	4.0	Divorced	
2	7000002	4273	26.0	4.0	Agent	Free Lancer	Post Graduate	Male	4	Exe	3.0	Unmarried	
3	7000003	1791	11.0	NaN	Third Party Partner	Salaried	Graduate	Fe male	3	Executive	3.0	Divorced	
4	7000004	2955	6.0	NaN	Agent	Small Business	UG	Male	3	Executive	4.0	Divorced	

Data Dictionary

Variable	Description
CustID	Unique customer ID
AgentBonus	Bonus amount given to each agents in last month
Age	Age of customer
CustTenure	Tenure of customer in organization
Channel	Channel through which acquisition of customer is done
Occupation	Occupation of customer
EducationField	Field of education of customer
Gender	Gender of customer
ExistingProdType	Existing product type of customer
Designation	Designation of customer in their organization
NumberOfPolicy	Total number of existing policy of a customer
MaritalStatus	Marital status of customer
MonthlyIncome	Gross monthly income of customer
Complaint	Indicator of complaint registered in last one month by customer
ExistingPolicyTenure	Max tenure in all existing policies of customer
SumAssured	Max of sum assured in all existing policies of customer
Zone	Customer belongs to which zone in India. Like East, West, North and South
PaymentMethod	Frequency of payment selected by customer like Monthly, quarterly, half yearly and yearly
LastMonthCalls	Total calls attempted by company to a customer for cross sell
CustCareScore	Customer satisfaction score given by customer in previous service call

Summary of the dataset:

	count	mean	std	min	25%	50%	75%	max
CustID	4520.0	7.002260e+06	1304.955938	7000000.0	7001129.75	7002259.5	7003389.25	7004519.0
AgentBonus	4520.0	4.077838e+03	1403.321711	1605.0	3027.75	3911.5	4867.25	9608.0
Age	4251.0	1.449471e+01	9.037629	2.0	7.00	13.0	20.00	58.0
CustTenure	4294.0	1.446903e+01	8.963671	2.0	7.00	13.0	20.00	57.0
ExistingProdType	4520.0	3.688938e+00	1.015769	1.0	3.00	4.0	4.00	6.0
NumberOfPolicy	4475.0	3.565363e+00	1.455926	1.0	2.00	4.0	5.00	6.0
MonthlyIncome	4284.0	2.289031e+04	4885.600757	16009.0	19683.50	21606.0	24725.00	38456.0
Complaint	4520.0	2.871681e-01	0.452491	0.0	0.00	0.0	1.00	1.0
ExistingPolicyTenure	4336.0	4.130074e+00	3.346386	1.0	2.00	3.0	6.00	25.0
SumAssured	4366.0	6.199997e+05	246234.822140	168536.0	439443.25	578976.5	758236.00	1838496.0
LastMonthCalls	4520.0	4.626991e+00	3.620132	0.0	2.00	3.0	8.00	18.0
CustCareScore	4468.0	3.067592e+00	1.382968	1.0	2.00	3.0	4.00	5.0