Capstone Project: Life Insurance Sales

Defining problem statement / Context

The dataset belongs to a leading life insurance company. The company wants to predict the bonus for its agents so that it may design appropriate engagement activity for their high performing agents and upskill programs for low performing agents.

Need of the study/project / Objective

- The company wants to predict bonus for its agents based on their performance.
- The company wants to design engagement activities for their high performing agents.
- The company wants to design upskill programs for their low performing agents.

Understanding business/social opportunity

The company want to improve the performance and engagement levels of agents by providing them with incentives based on their predicted bonus. This study provides a business opportunity for the life insurance company to optimize its agent engagement and performance.

Importing required libraries

In [163…

```
# Import libraries for data manipulation
import numpy as np
import pandas as pd

# Import libraries for data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# to impute using KNNImputer
from sklearn.impute import KNNImputer

# to scale the data using zscore
from scipy.stats import zscore

# to perform KMeans clustring
from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_samples, silhouette_score

from sklearn.model_selection import train_test_split,GridSearchCV
```

```python
# to perform Linear Regression
from sklearn.linear_model import LinearRegression

# to perform Lasso Regression
from sklearn.linear_model import Lasso

# to perform Ridge Regression
from sklearn.linear_model import Ridge

# to perform Decision Tree Regression

from sklearn.tree import DecisionTreeRegressor

# to perform Random Forest Regression
from sklearn.ensemble import RandomForestRegressor

# to perform XGBoost Regression
import xgboost as xg

# to perform AdaBoost Regression
from sklearn.ensemble import AdaBoostRegressor

# to perform SVR Regression
from sklearn.svm import SVR

# to check model performance
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, mean

from sklearn.model_selection import KFold

import warnings
warnings.filterwarnings( "ignore")
```

Data Report / Description

Data Dictionary

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

Understanding the structure of dataset

In [164…
```
df = pd.read_excel('Sales.xlsx',sheet_name = 'Sales') # Importing the data
```

In [165…
```
df.head() # Returns first 5 rows
```

Out[165…

	CustID	AgentBonus	Age	CustTenure	Channel	Occupation	EducationField	Gender
0	7000000	4409	22.0	4.0	Agent	Salaried	Graduate	Female
1	7000001	2214	11.0	2.0	Third Party Partner	Salaried	Graduate	Male
2	7000002	4273	26.0	4.0	Agent	Free Lancer	Post Graduate	Male
3	7000003	1791	11.0	NaN	Third Party Partner	Salaried	Graduate	Fe male
4	7000004	2955	6.0	NaN	Agent	Small Business	UG	Male

Number of rows and columns in the dataset

In [166…
```
# checking shape of the data

rows = str(df.shape[0])
columns = str(df.shape[1])

print(f"There are \033[1m" + rows + "\033[0m rows and \033[1m" + columns + "\033[0m
```

```
There are 4520 rows and 20 columns in the dataset.
```

Datatypes of the different columns in the dataset

In [167…
```
df.info() # Concise summary of dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4520 entries, 0 to 4519
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CustID                4520 non-null   int64  
 1   AgentBonus            4520 non-null   int64  
 2   Age                   4251 non-null   float64
 3   CustTenure            4294 non-null   float64
 4   Channel               4520 non-null   object 
 5   Occupation            4520 non-null   object 
 6   EducationField        4520 non-null   object 
 7   Gender                4520 non-null   object 
 8   ExistingProdType      4520 non-null   int64  
 9   Designation           4520 non-null   object 
 10  NumberOfPolicy        4475 non-null   float64
 11  MaritalStatus         4520 non-null   object 
 12  MonthlyIncome         4284 non-null   float64
 13  Complaint             4520 non-null   int64  
 14  ExistingPolicyTenure  4336 non-null   float64
 15  SumAssured            4366 non-null   float64
 16  Zone                  4520 non-null   object 
 17  PaymentMethod         4520 non-null   object 
 18  LastMonthCalls        4520 non-null   int64  
 19  CustCareScore         4468 non-null   float64
dtypes: float64(7), int64(5), object(8)
memory usage: 706.4+ KB
```

There are 20 columns in the dataset. Out of which 7 have float data type, 5 have integer data type and 8 have object data type.

Check duplicate records

In [168…

```
df.duplicated().sum() # Check duplicate records
```

Out[168…

```
0
```

There are no duplicate rows in the dataset.

Statistical summary of the data

In [169…

```
df.describe().T # Summary statistics of the numerical and categorial data
```

Out[169…

	count	mean	std	min	25%	50%
CustID	4520.0	7.002260e+06	1304.955938	7000000.0	7001129.75	7002259.5
AgentBonus	4520.0	4.077838e+03	1403.321711	1605.0	3027.75	3911.5
Age	4251.0	1.449471e+01	9.037629	2.0	7.00	13.0
CustTenure	4294.0	1.446903e+01	8.963671	2.0	7.00	13.0
ExistingProdType	4520.0	3.688938e+00	1.015769	1.0	3.00	4.0
NumberOfPolicy	4475.0	3.565363e+00	1.455926	1.0	2.00	4.0
MonthlyIncome	4284.0	2.289031e+04	4885.600757	16009.0	19683.50	21606.0
Complaint	4520.0	2.871681e-01	0.452491	0.0	0.00	0.0
ExistingPolicyTenure	4336.0	4.130074e+00	3.346386	1.0	2.00	3.0
SumAssured	4366.0	6.199997e+05	246234.822140	168536.0	439443.25	578976.5
LastMonthCalls	4520.0	4.626991e+00	3.620132	0.0	2.00	3.0
CustCareScore	4468.0	3.067592e+00	1.382968	1.0	2.00	3.0

Observations and Insights:

- Minimum bonus amount for agent is 1605.0 INR and maximum bonus amount for agent is 9608.0 INR.
- Minimum age of customer is 2 years and maximum age of customer is 58 years.
- Minimum tenure for customer in organization is 2 years and maximum tenure for customer in organization is 57 years.
- Existing product type of customer is in range 1 and 6.
- Total number of existing policies of a customer is in range 1 and 6.
- Minimum gross monthly income of customer is 16009.0 INR and maximum gross monthly income of customer is 38456.0 INR.
- Complaint indicator is in range 0 (No) and 1 (Yes).
- Max tenure in all existing policies of customer is in range 1 and 25.
- Max of sum assured in all existing policies of customer is in range 168536.0 INR and 1838496.0 INR.
- Total calls attempted by company to a customer for cross sell is in range 0 and 18.
- Customer satisfaction score given by customer in previous service call is in range 1 and 5.

Exploratory Data Analysis (EDA)

Univariate Analysis

Channel

In [170…

```
# Check unique Gender
df['Channel'].value_counts() # Frequency of each distinct value in the Channel colu
```

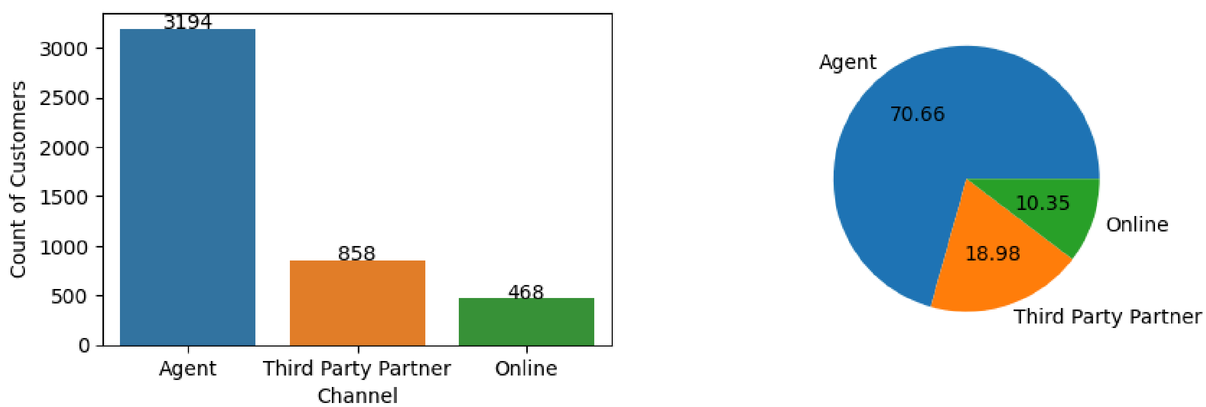
Out[170…

```
Channel
Agent                  3194
Third Party Partner     858
Online                  468
Name: count, dtype: int64
```

In [171…

```
# Count Plot and Pie Chart - Distribution of Channel across customers

fig, ax = plt.subplots(1,2, figsize=(10,3))
sns.countplot(data=df, x='Channel', order = df['Channel'].value_counts().index, ax=
ax[0].set(xlabel = 'Channel', ylabel = 'Count of Customers')

# Looping over entire dataset:
for p in ax[0].patches:
    height = p.get_height()
    ax[0].text(p.get_x()+p.get_width()/2., height + 3, '{:1.0f}'.format(height), ha

ax[1]=plt.pie(df['Channel'].value_counts(), labels=['Agent', 'Third Party Partner',
fig.suptitle('Fig 1: Distribution of Channel Across Customers')
plt.show()
```

Fig 1: Distribution of Channel Across Customers



Occupation

In [172…

```
# Check unique Occupation
df['Occupation'].value_counts() # Frequency of each distinct value in the Occupatio
```

Out[172…

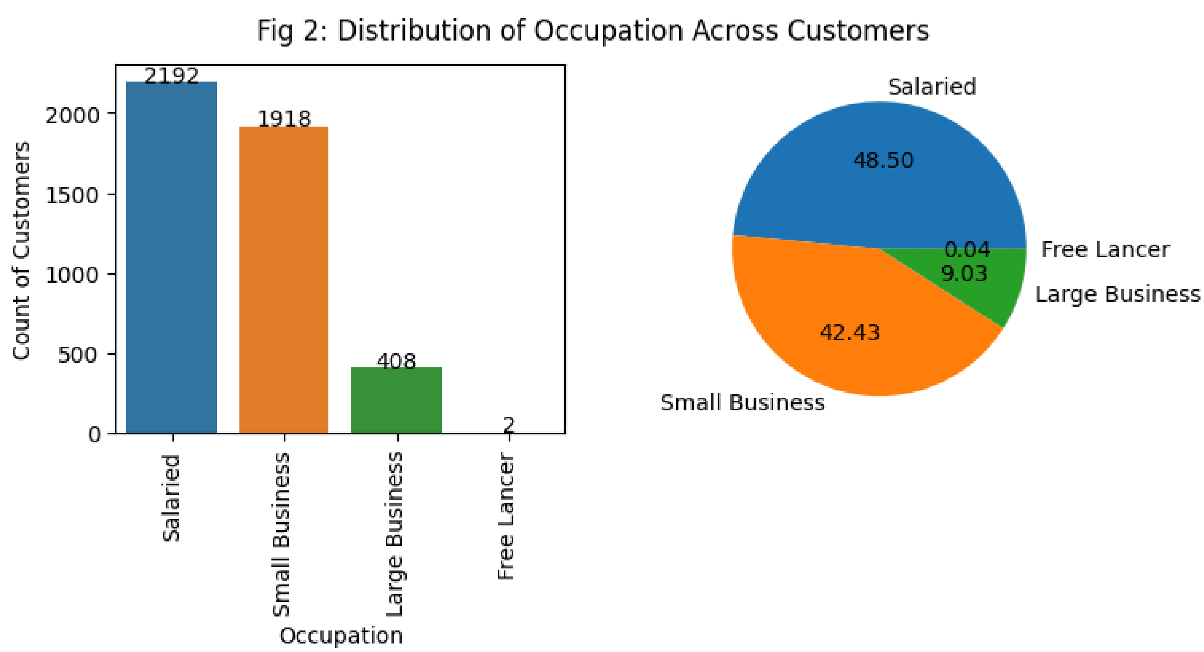
```
Occupation
Salaried          2192
Small Business    1918
Large Business     255
Laarge Business    153
Free Lancer          2
Name: count, dtype: int64
```

In [173…
```
df['Occupation'].replace('Laarge Business', 'Large Business', inplace=True) # Repla
```

In [174…
```
# Check unique Occupation (after replacement)
df['Occupation'].value_counts() # Frequency of each distinct value in the Occupatio
```

Out[174…
```
Occupation
Salaried          2192
Small Business    1918
Large Business     408
Free Lancer          2
Name: count, dtype: int64
```

In [175…
```
# Count Plot and Pie Chart - Distribution of Occupation across customers

fig, ax = plt.subplots(1,2, figsize=(8,3))
sns.countplot(data=df, x='Occupation', order = df['Occupation'].value_counts().inde
ax[0].set(xlabel = 'Occupation', ylabel = 'Count of Customers')

# Looping over entire dataset:
for p in ax[0].patches:
    height = p.get_height()
    ax[0].text(p.get_x()+p.get_width()/2., height + 3, '{:1.0f}'.format(height), ha

ax[0].set_xticklabels(ax[0].get_xticklabels(), rotation=90)
ax[1]=plt.pie(df['Occupation'].value_counts(), labels=['Salaried', 'Small Business'
fig.suptitle('Fig 2: Distribution of Occupation Across Customers')
plt.show()
```



EducationField

In [176…
```
# Check unique EducationField
df['EducationField'].value_counts() # Frequency of each distinct value in the Educa
```

Out[176…

```
EducationField
Graduate          1870
Under Graduate    1190
Diploma            496
Engineer           408
Post Graduate      252
UG                 230
MBA                 74
Name: count, dtype: int64
```

In [177…

```
df['EducationField'].replace('UG', 'Under Graduate', inplace=True) # Replace UG val
```

In [178…

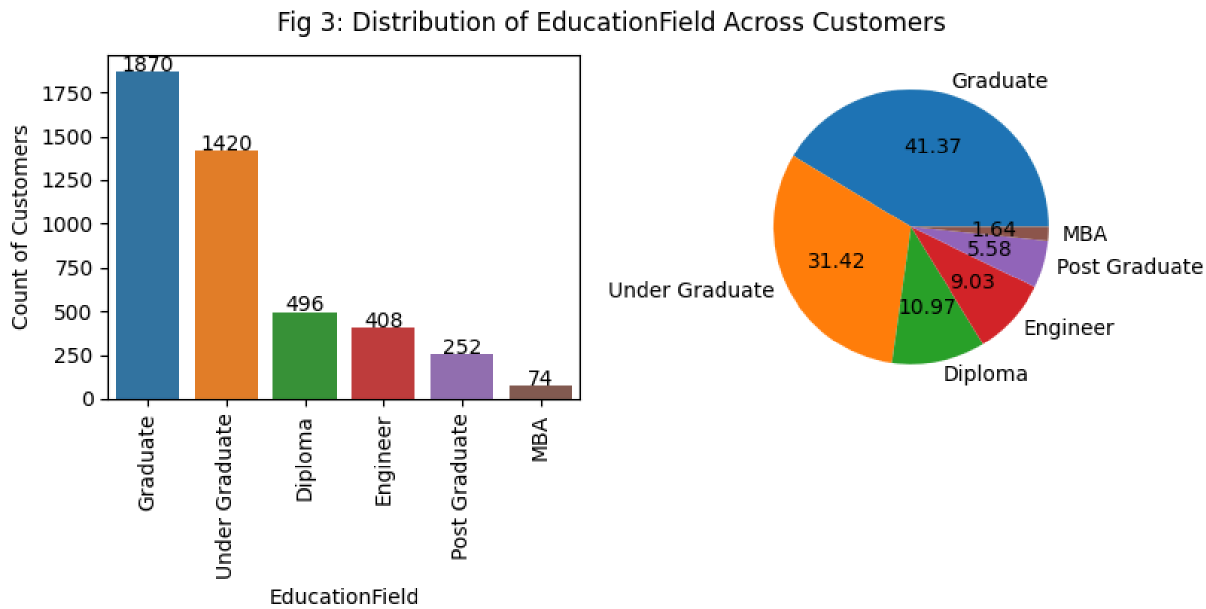
```
# Check unique EducationField (after replacement)
df['EducationField'].value_counts() # Frequency of each distinct value in the Educa
```

Out[178…

```
EducationField
Graduate          1870
Under Graduate    1420
Diploma            496
Engineer           408
Post Graduate      252
MBA                 74
Name: count, dtype: int64
```

In [179…

```
# Count Plot and Pie Chart - Distribution of EducationField across customers

fig, ax = plt.subplots(1,2, figsize=(9,3))
sns.countplot(data=df, x='EducationField', order = df['EducationField'].value_count
ax[0].set(xlabel = 'EducationField', ylabel = 'Count of Customers')

# Looping over entire dataset:
for p in ax[0].patches:
    height = p.get_height()
    ax[0].text(p.get_x()+p.get_width()/2., height + 3, '{:1.0f}'.format(height), ha

ax[0].set_xticklabels(ax[0].get_xticklabels(), rotation=90)
ax[1]=plt.pie(df['EducationField'].value_counts(), labels=['Graduate', 'Under Gradu
fig.suptitle('Fig 3: Distribution of EducationField Across Customers')
plt.show()
```



Gender

In [180…

```
# Check unique Gender
df['Gender'].value_counts() # Frequency of each distinct value in the Gender column
```

Out[180…

```
Gender
Male        2688
Female      1507
Fe male      325
Name: count, dtype: int64
```

In [181…

```
df['Gender'].replace('Fe male', 'Female', inplace=True) # Replace Fe male value wit
```

In [182…

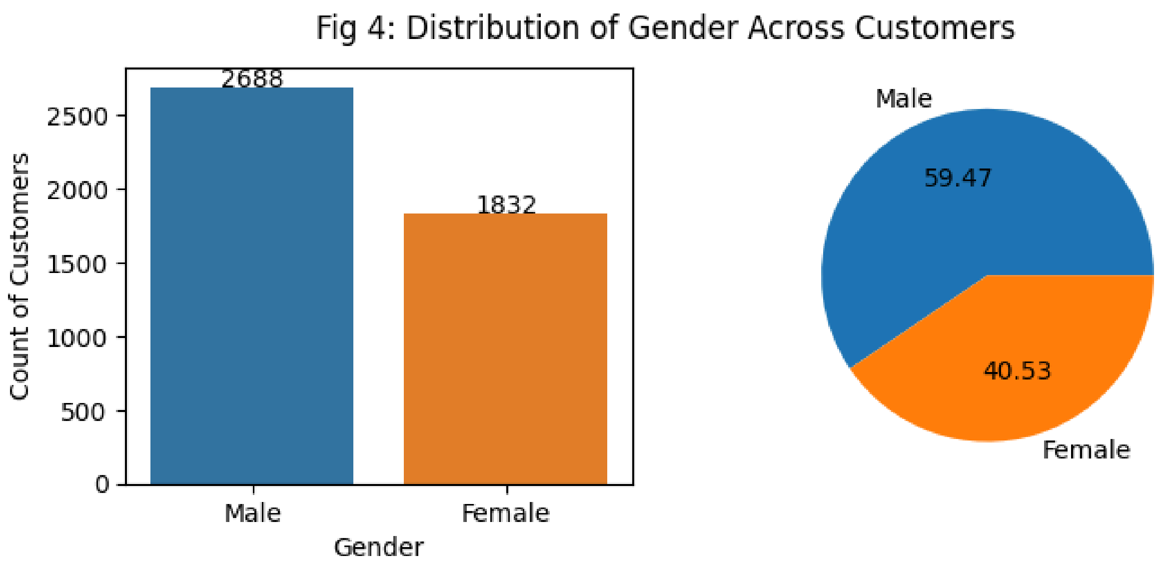
```
# Check unique Gender (after replacement)
df['Gender'].value_counts() # Frequency of each distinct value in the Gender column
```

Out[182…

```
Gender
Male      2688
Female    1832
Name: count, dtype: int64
```

In [183…

```
# Count Plot and Pie Chart - Distribution of Gender across customers

fig, ax = plt.subplots(1,2, figsize=(8,3))
sns.countplot(data=df, x='Gender', order = df['Gender'].value_counts().index, ax=ax
ax[0].set(xlabel = 'Gender', ylabel = 'Count of Customers')

# Looping over entire dataset:
for p in ax[0].patches:
    height = p.get_height()
    ax[0].text(p.get_x()+p.get_width()/2., height + 3, '{:1.0f}'.format(height), ha

ax[1]=plt.pie(df['Gender'].value_counts(), labels=['Male', 'Female'], autopct='%.2f
fig.suptitle('Fig 4: Distribution of Gender Across Customers')
plt.show()
```



Designation

In [184…

```
# Check unique Designation
df['Designation'].value_counts() # Frequency of each distinct value in the Designat
```

Out[184…

```
Designation
Manager           1620
Executive         1535
Senior Manager     676
AVP                336
VP                 226
Exe                127
Name: count, dtype: int64
```

In [185…

```
df['Designation'].replace('Exe', 'Executive', inplace=True) # Replace Exe value wit
```

In [186…

```
# Check unique Designation (after replacement)
df['Designation'].value_counts() # Frequency of each distinct value in the Designat
```

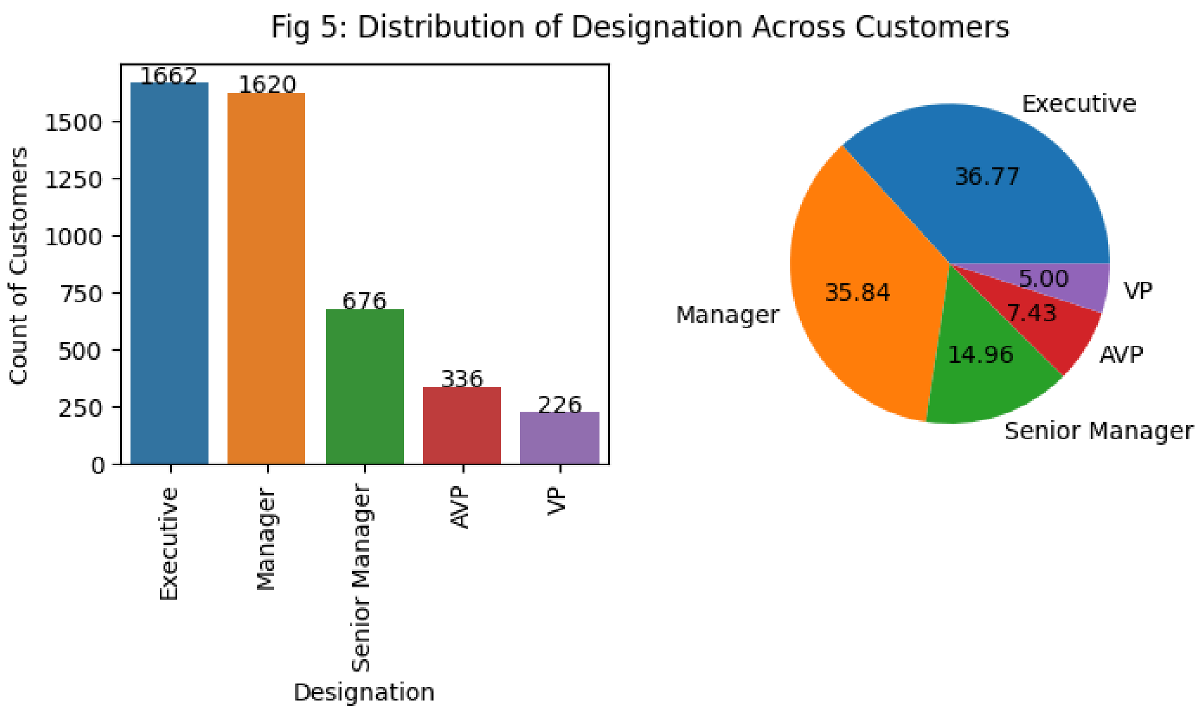
Out[186…

```
Designation
Executive         1662
Manager           1620
Senior Manager     676
AVP                336
VP                 226
Name: count, dtype: int64
```

In [187…

```
# Count Plot and Pie Chart - Distribution of Designation across customers

fig, ax = plt.subplots(1,2, figsize=(8,3))
sns.countplot(data=df, x='Designation', order = df['Designation'].value_counts().in
ax[0].set(xlabel = 'Designation', ylabel = 'Count of Customers')

# Looping over entire dataset:
for p in ax[0].patches:
    height = p.get_height()
    ax[0].text(p.get_x()+p.get_width()/2., height + 3, '{:1.0f}'.format(height), ha
```

```
ax[0].set_xticklabels(ax[0].get_xticklabels(), rotation=90)
ax[1]=plt.pie(df['Designation'].value_counts(), labels=['Executive', 'Manager', 'Se
fig.suptitle('Fig 5: Distribution of Designation Across Customers')
plt.show()
```



MaritalStatus

In [188…

```
# Check unique Designation
df['MaritalStatus'].value_counts() # Frequency of each distinct value in the Marita
```

Out[188…

```
MaritalStatus
Married      2268
Single       1254
Divorced      804
Unmarried     194
Name: count, dtype: int64
```

In [189…

```
df['MaritalStatus'].replace('Unmarried', 'Single', inplace=True) # Replace Unmarrie
```

In [190…

```
# Check unique MaritalStatus (after replacement)
df['MaritalStatus'].value_counts() # Frequency of each distinct value in the Marita
```

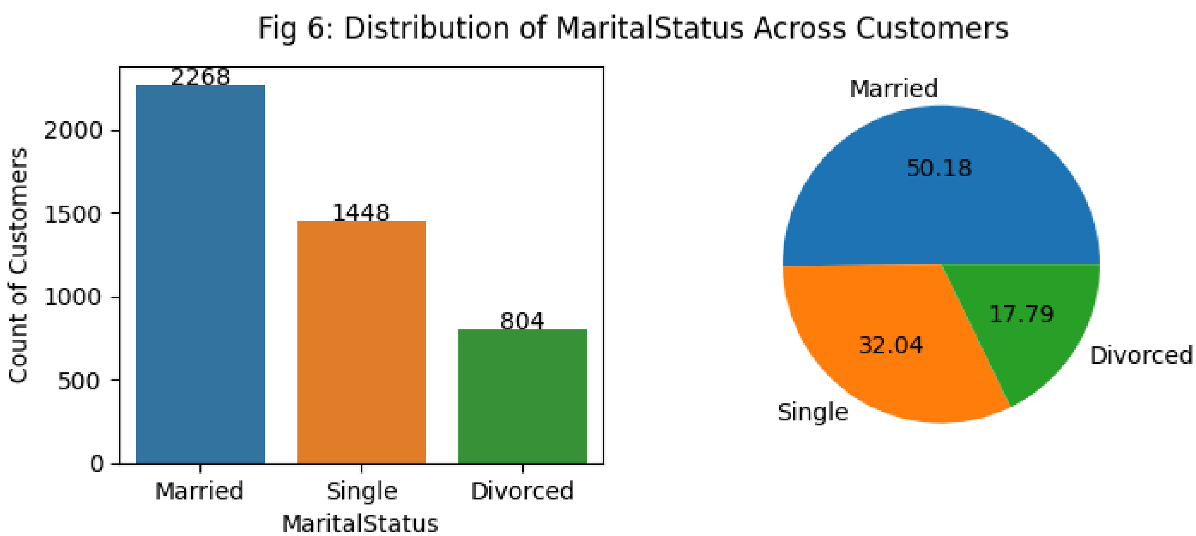
Out[190…

```
MaritalStatus
Married     2268
Single      1448
Divorced     804
Name: count, dtype: int64
```

In [191…

```
# Count Plot and Pie Chart - Distribution of MaritalStatus across customers

fig, ax = plt.subplots(1,2, figsize=(8,3))
sns.countplot(data=df, x='MaritalStatus', order = df['MaritalStatus'].value_counts(
```

```
ax[0].set(xlabel = 'MaritalStatus', ylabel = 'Count of Customers')

# Looping over entire dataset:
for p in ax[0].patches:
    height = p.get_height()
    ax[0].text(p.get_x()+p.get_width()/2., height + 3, '{:1.0f}'.format(height), ha

ax[1]=plt.pie(df['MaritalStatus'].value_counts(), labels=['Married', 'Single' ,'Div
fig.suptitle('Fig 6: Distribution of MaritalStatus Across Customers')
plt.show()
```



Zone

In [192…

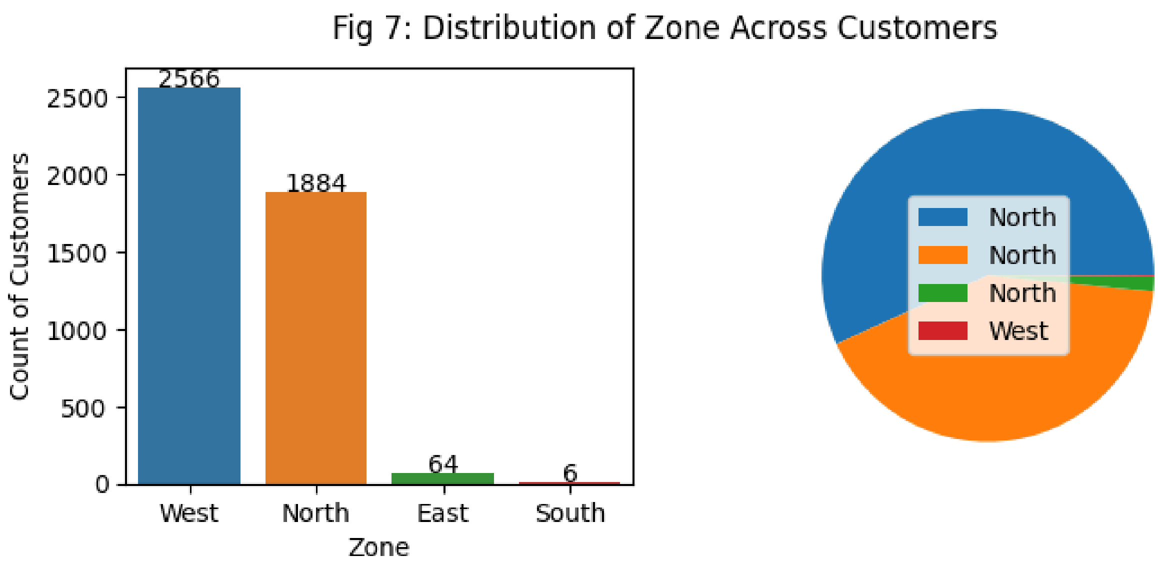
```
# Check unique Zone
df['Zone'].value_counts() # Frequency of each distinct value in the Zone column
```

Out[192…

```
Zone
West     2566
North    1884
East       64
South       6
Name: count, dtype: int64
```

In [193…

```
# Count Plot and Pie Chart - Distribution of Zone across customers

fig, ax = plt.subplots(1,2, figsize=(8,3))
sns.countplot(data=df, x='Zone', order = df['Zone'].value_counts().index, ax=ax[0])
ax[0].set(xlabel = 'Zone', ylabel = 'Count of Customers')

# Looping over entire dataset:
for p in ax[0].patches:
    height = p.get_height()
    ax[0].text(p.get_x()+p.get_width()/2., height + 3, '{:1.0f}'.format(height), ha

ax[1]=plt.pie(df['Zone'].value_counts())
plt.legend(df['Zone'], loc='center')
fig.suptitle('Fig 7: Distribution of Zone Across Customers')
plt.show()
```



PaymentMethod

In [194…

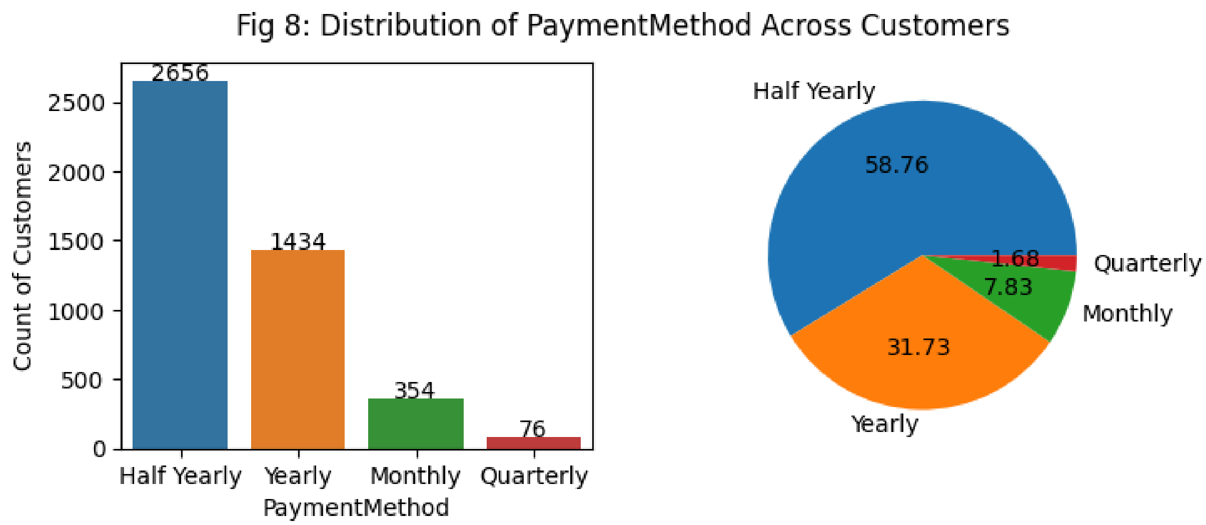
```
# Check unique PaymentMethod
df['PaymentMethod'].value_counts() # Frequency of each distinct value in the Paymen
```

Out[194…

```
PaymentMethod
Half Yearly    2656
Yearly         1434
Monthly         354
Quarterly        76
Name: count, dtype: int64
```

In [195…

```
# Count Plot and Pie Chart - Distribution of PaymentMethod across customers

fig, ax = plt.subplots(1,2, figsize=(8,3))
sns.countplot(data=df, x='PaymentMethod', order = df['PaymentMethod'].value_counts(
ax[0].set(xlabel = 'PaymentMethod', ylabel = 'Count of Customers')

# Looping over entire dataset:
for p in ax[0].patches:
    height = p.get_height()
    ax[0].text(p.get_x()+p.get_width()/2., height + 3, '{:1.0f}'.format(height), ha

ax[1]=plt.pie(df['PaymentMethod'].value_counts(), labels=['Half Yearly', 'Yearly' ,
fig.suptitle('Fig 8: Distribution of PaymentMethod Across Customers')
plt.show()
```



Observations and Insights:

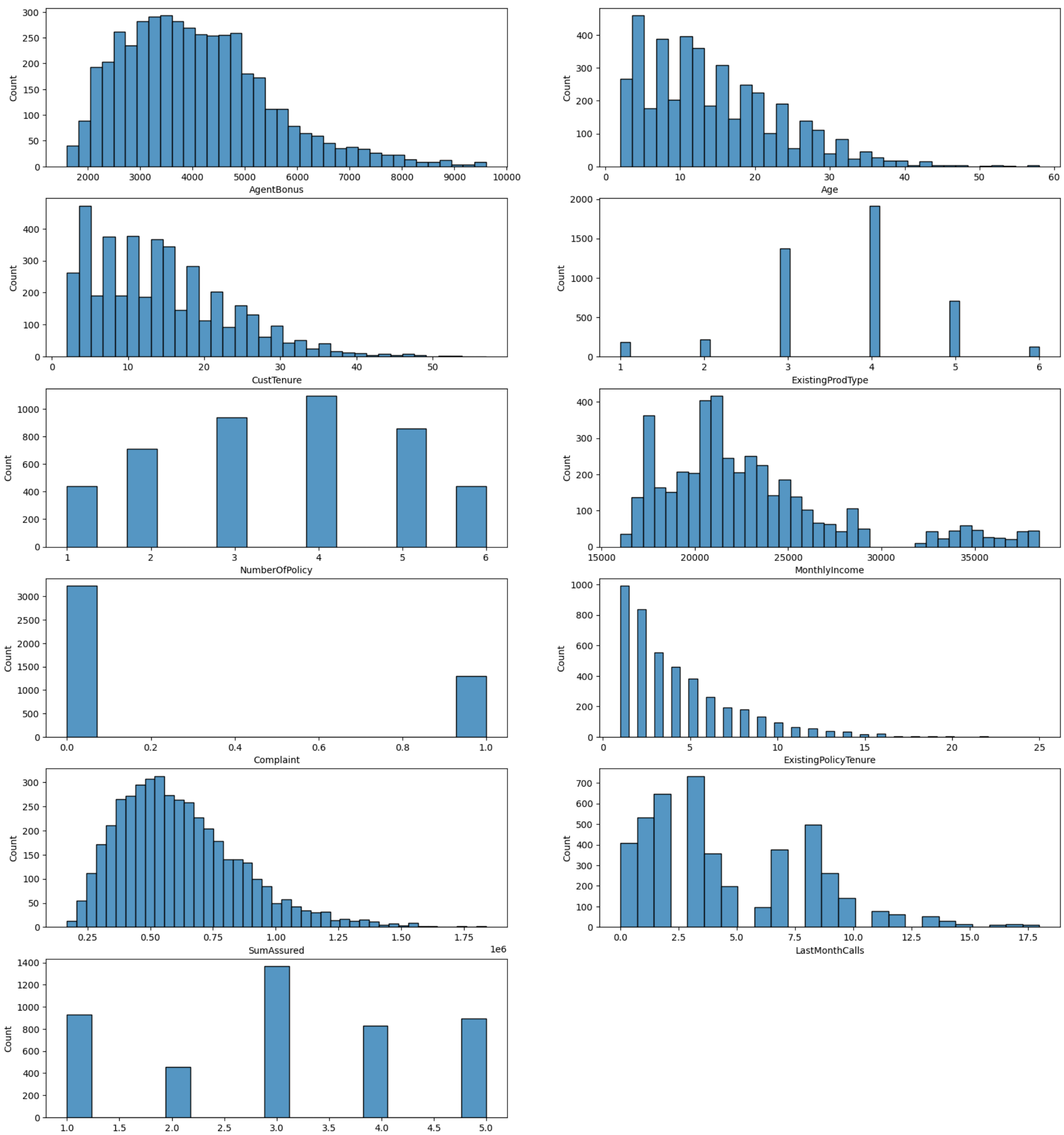
- Distinct values for channel through which acquisition of customer are: Agent, Third Party Partner and Online (count of customers - highest to lowest).
- Distinct values for occupation of customer are: Salaried, Small Business, Large Business and Free Lancer (count of customers - highest to lowest).
- Distinct values for field of education of customer are: Graduate, Under Graduate, Diploma, Engineer, Post Graduate and MBA (count of customers - highest to lowest).
- Distinct values for gender of customer are: Male and Female (count of customers - highest to lowest).
- Distinct values for designation of customer in their organization are: Executive, Manager, Senior Manager, AVP and VP (count of customers - highest to lowest).
- Distinct values for marital status of customer are: Married, Single, Divorced (count of customers - highest to lowest).
- Distinct values for customer belong to which zone in India are: West, North, East, South (count of customers - highest to lowest).
- Distinct values for frequency of payment selected by customer are: Half Yearly, Yearly, Monthly, Quarterly (count of customers - highest to lowest).

In [196…

```
# Hist Plots for AgentBonus, Age, CustTenure, ExistingProdType, NumberOfPolicy, Mon

fig, axes = plt.subplots(6,2, figsize=(20, 22))

sns.histplot(ax=axes[0, 0], data=df, x='AgentBonus')
sns.histplot(ax=axes[0, 1], data=df, x='Age')
sns.histplot(ax=axes[1, 0], data=df, x='CustTenure')
sns.histplot(ax=axes[1, 1], data=df, x='ExistingProdType')
sns.histplot(ax=axes[2, 0], data=df, x='NumberOfPolicy')
sns.histplot(ax=axes[2, 1], data=df, x='MonthlyIncome')
sns.histplot(ax=axes[3, 0], data=df, x='Complaint')
sns.histplot(ax=axes[3, 1], data=df, x='ExistingPolicyTenure')
sns.histplot(ax=axes[4, 0], data=df, x='SumAssured')
sns.histplot(ax=axes[4, 1], data=df, x='LastMonthCalls')
sns.histplot(ax=axes[5, 0], data=df, x='CustCareScore')
```

```python
axes[5,1].axis("off")

plt.suptitle('Fig 9: Hist Plots: AgentBonus, Age, CustTenure, ExistingProdType, Num

plt.show()
```

Fig 9: Hist Plots: AgentBonus, Age, CustTenure, ExistingProdType, NumberOfPolicy, MonthlyIncome, Complaint, ExistingPolicyTenure, SumAssured, LastMonthCalls, CustCareScore

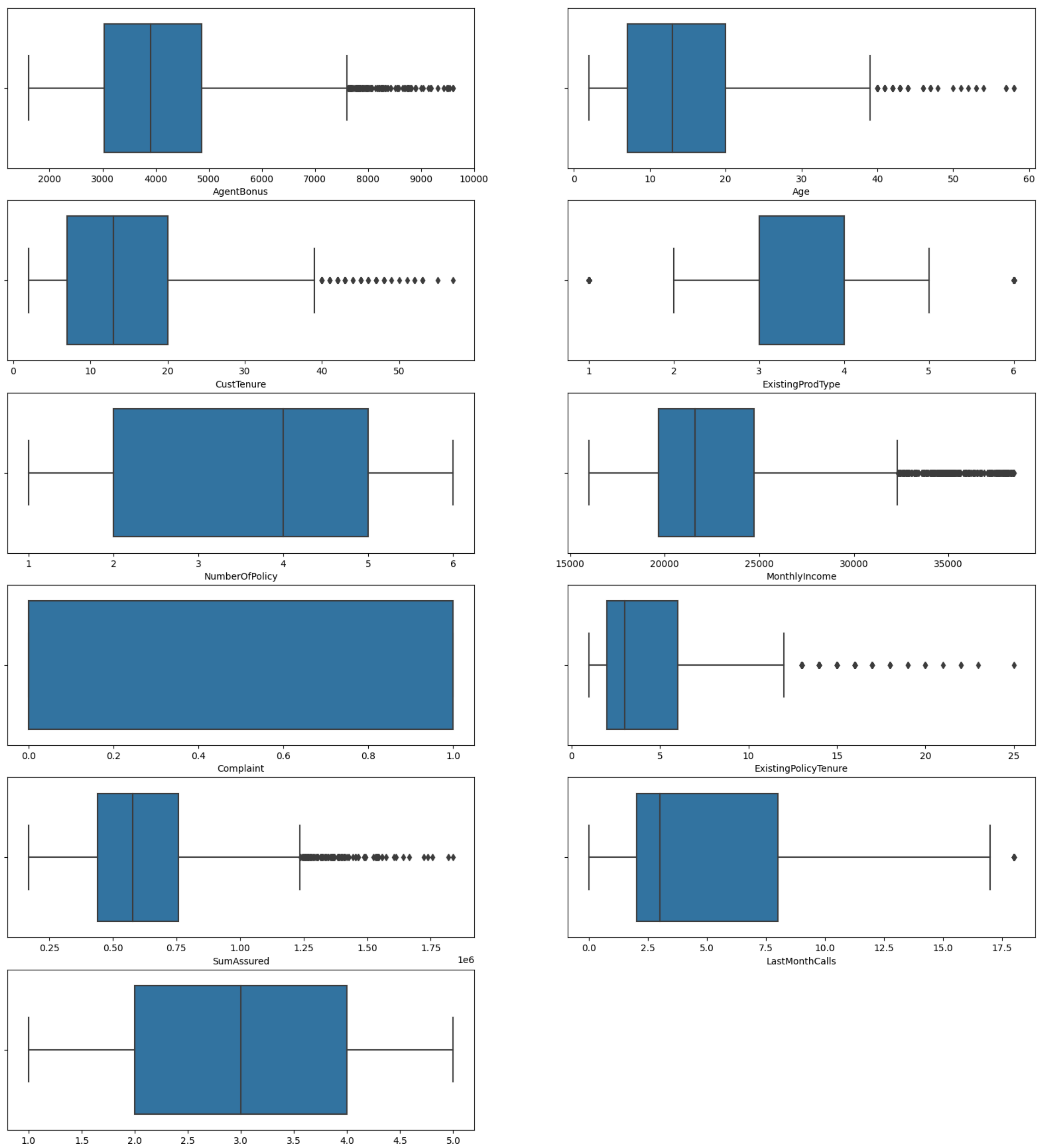


In [197…

```python
# Box Plots for AgentBonus, Age, CustTenure, ExistingProdType, NumberOfPolicy, Mont

fig, axes = plt.subplots(6,2, figsize=(20, 22))

sns.boxplot(ax=axes[0, 0], data=df, x='AgentBonus')
sns.boxplot(ax=axes[0, 1], data=df, x='Age')
sns.boxplot(ax=axes[1, 0], data=df, x='CustTenure')
sns.boxplot(ax=axes[1, 1], data=df, x='ExistingProdType')
sns.boxplot(ax=axes[2, 0], data=df, x='NumberOfPolicy')
```

```
sns.boxplot(ax=axes[2, 1], data=df, x='MonthlyIncome')
sns.boxplot(ax=axes[3, 0], data=df, x='Complaint')
sns.boxplot(ax=axes[3, 1], data=df, x='ExistingPolicyTenure')
sns.boxplot(ax=axes[4, 0], data=df, x='SumAssured')
sns.boxplot(ax=axes[4, 1], data=df, x='LastMonthCalls')
sns.boxplot(ax=axes[5, 0], data=df, x='CustCareScore')
axes[5,1].axis("off")

plt.suptitle('Fig 10: Box Plots: AgentBonus, Age, CustTenure, ExistingProdType, Num

plt.show()
```

Fig 10: Box Plots: AgentBonus, Age, CustTenure, ExistingProdType, NumberOfPolicy, MonthlyIncome, Complaint, ExistingPolicyTenure, SumAssured, LastMonthCalls, CustCareScore



Observations and Insights:

- No distribution is evenly distributed (symmetric).
- Some distributions are Positively Skewed (mean is more than the mode).
- AgentBonus, Age, CustTenure, ExistingProdType, MonthlyIncome, ExistingPolicyTenure, SumAssured and LastMonthCalls columns are having outliers.

Bivariate Analysis

Correlation among variables

In [198…

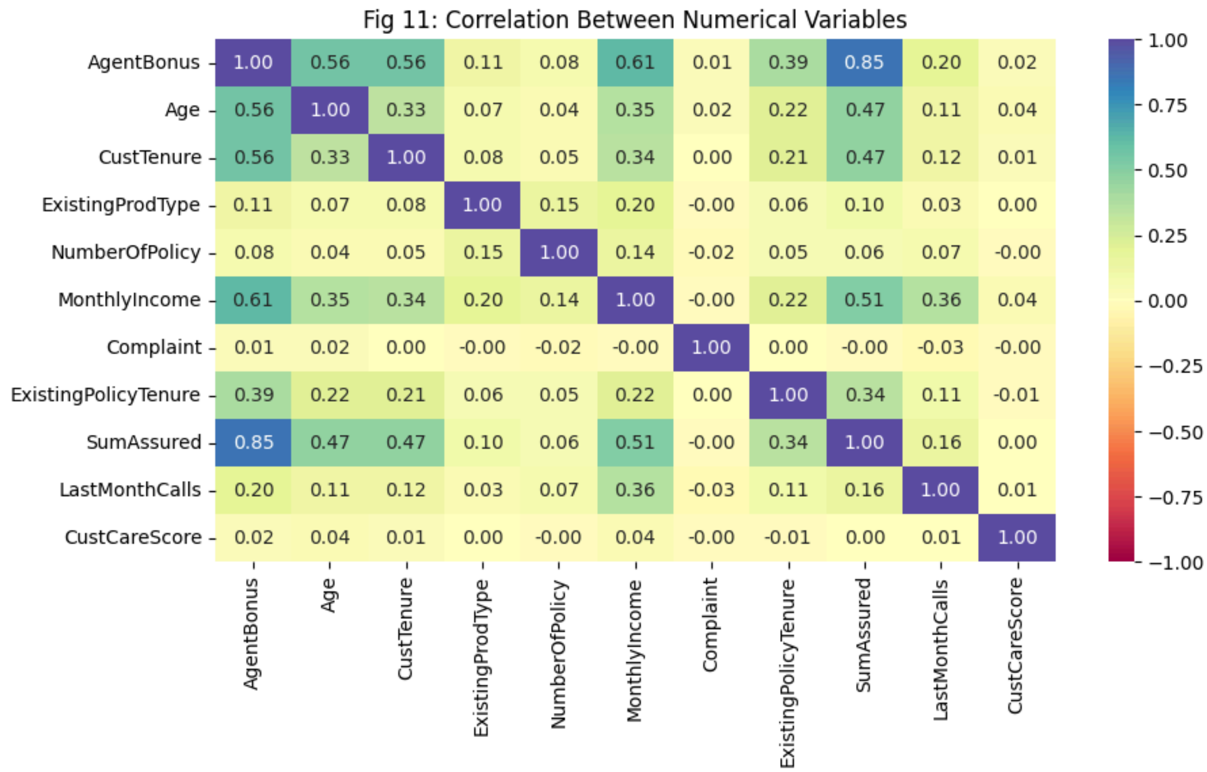
```
# Correlation between all numerical variables in the dataset

df_corr = df.drop(df.columns[[0]], axis=1)
df_corr = df_corr.select_dtypes(include=[np.number])
df_corr.corr()
```

Out[198…

	AgentBonus	Age	CustTenure	ExistingProdType	NumberOfPolicy
AgentBonus	1.000000	0.559481	0.561344	0.113023	0.076448
Age	0.559481	1.000000	0.328627	0.070555	0.042143
CustTenure	0.561344	0.328627	1.000000	0.079891	0.045021
ExistingProdType	0.113023	0.070555	0.079891	1.000000	0.150923
NumberOfPolicy	0.076448	0.042143	0.045021	0.150923	1.000000
MonthlyIncome	0.612196	0.354162	0.344911	0.198468	0.136518
Complaint	0.014281	0.021888	0.003807	-0.003486	-0.016416
ExistingPolicyTenure	0.392415	0.216259	0.214984	0.057066	0.049673
SumAssured	0.854257	0.474434	0.474610	0.102597	0.060359
LastMonthCalls	0.199708	0.114670	0.115993	0.033191	0.074069
CustCareScore	0.022860	0.035694	0.011145	0.003813	-0.002265

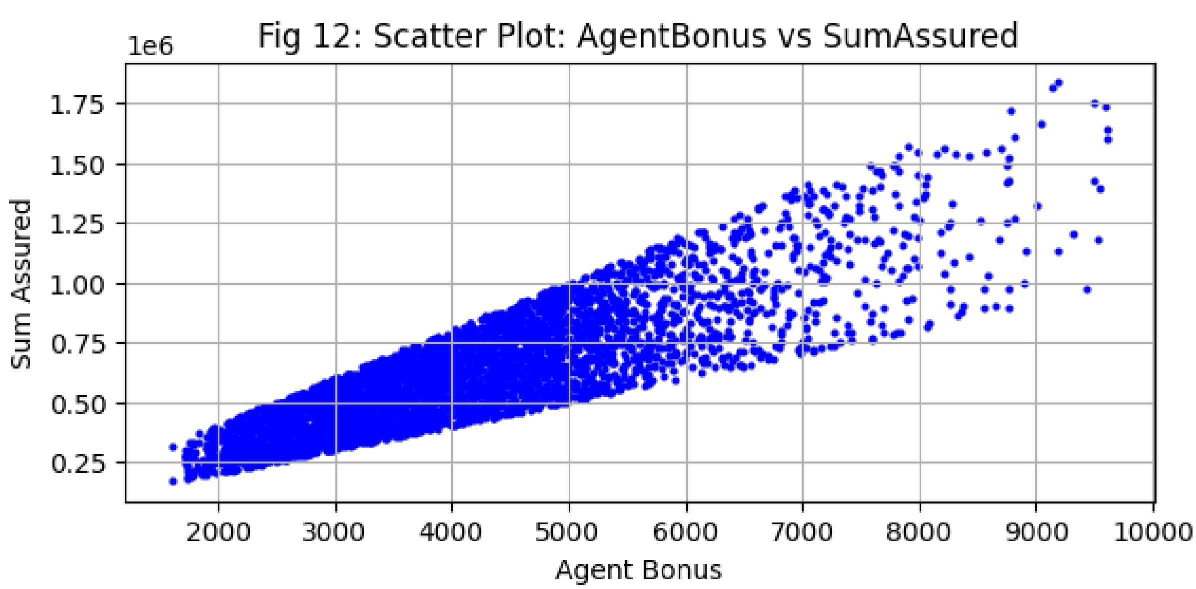
In [199…

```
# Heatmap to plot correlation between all numerical variables in the dataset

plt.figure(figsize=(10, 5))
sns.heatmap(df_corr.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
plt.title('Fig 11: Correlation Between Numerical Variables')
plt.show()
```



In [200…

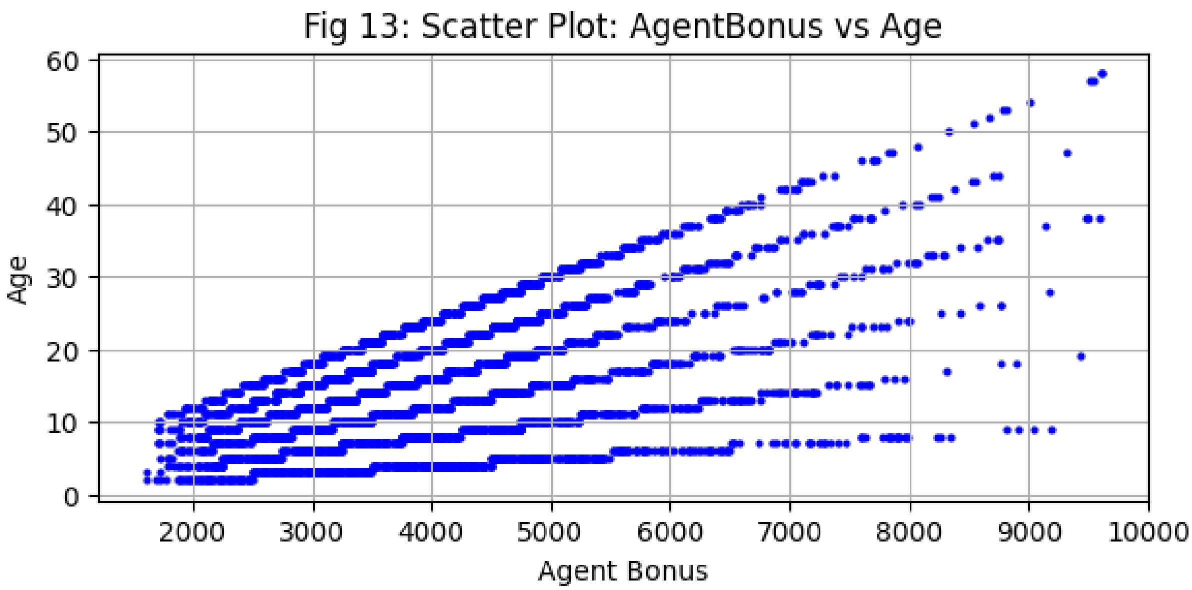
```
# Scatter Plot between AgentBonus and SumAssured

plt.figure(figsize=(7, 3))
plt.scatter(df.AgentBonus, df.SumAssured, s=4, c="blue")
plt.xlabel('Agent Bonus')
plt.ylabel('Sum Assured')
plt.title('Fig 12: Scatter Plot: AgentBonus vs SumAssured')
plt.grid()
plt.show()
```



In [201…

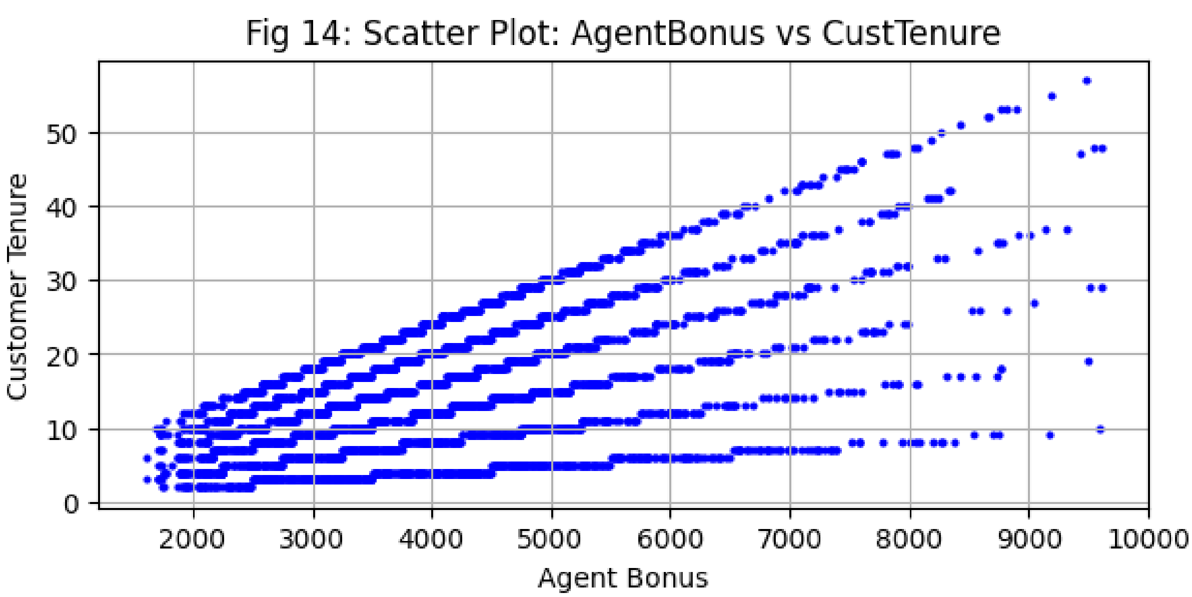
```
# Scatter Plot between AgentBonus and Age

plt.figure(figsize=(7, 3))
```

```
plt.scatter(df.AgentBonus, df.Age, s=4, c="blue")
plt.xlabel('Agent Bonus')
plt.ylabel('Age')
plt.title('Fig 13: Scatter Plot: AgentBonus vs Age')
plt.grid()
plt.show()
```



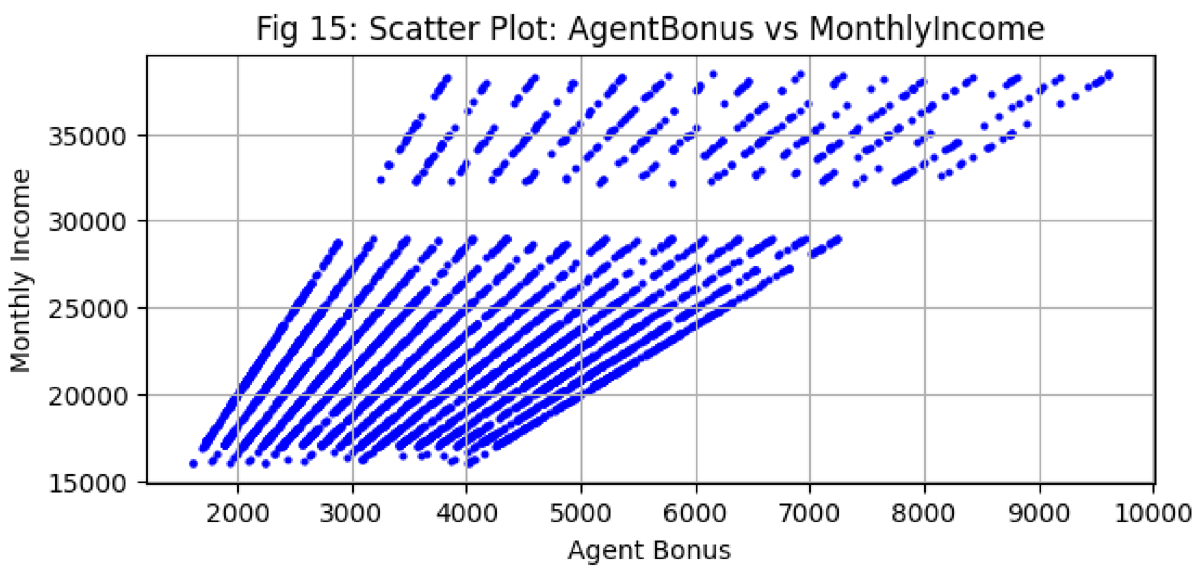
In [202…

```
# Scatter Plot between AgentBonus and CustTenure

plt.figure(figsize=(7, 3))
plt.scatter(df.AgentBonus, df.CustTenure, s=4, c="blue")
plt.xlabel('Agent Bonus')
plt.ylabel('Customer Tenure')
plt.title('Fig 14: Scatter Plot: AgentBonus vs CustTenure')
plt.grid()
plt.show()
```



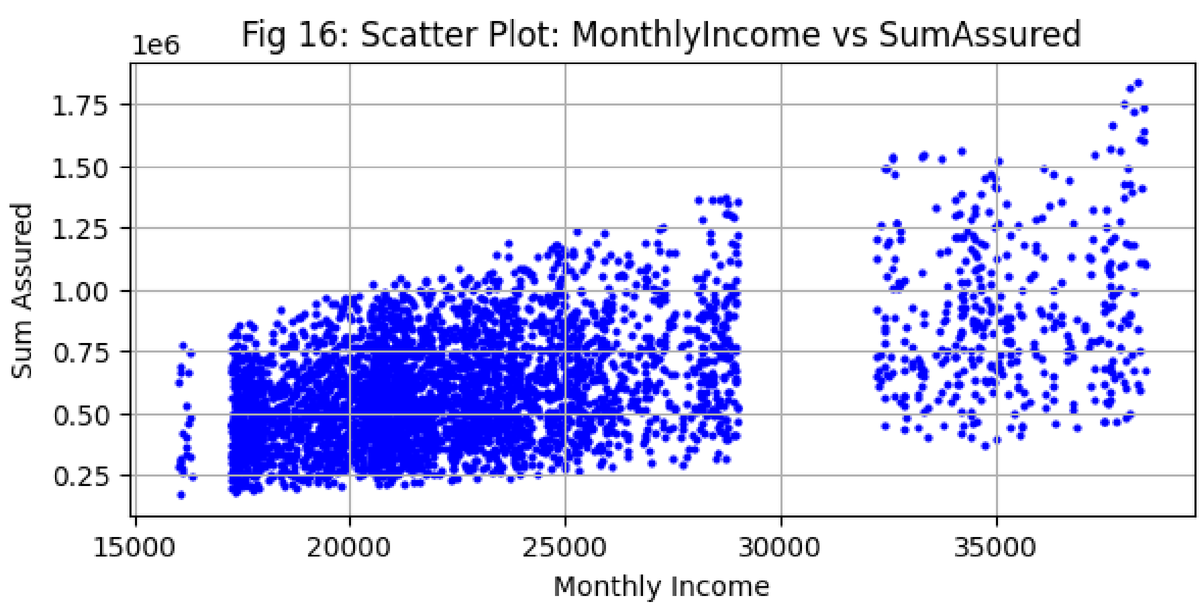
In [203…

```python
# Scatter Plot between AgentBonus and MonthlyIncome

plt.figure(figsize=(7, 3))
plt.scatter(df.AgentBonus, df.MonthlyIncome, s=4, c="blue")
plt.xlabel('Agent Bonus')
plt.ylabel('Monthly Income')
plt.title('Fig 15: Scatter Plot: AgentBonus vs MonthlyIncome')
plt.grid()
plt.show()
```



In [204…

```python
plt.figure(figsize=(7, 3))
plt.scatter(df.MonthlyIncome, df.SumAssured, s=4, c="blue")
plt.xlabel('Monthly Income')
plt.ylabel('Sum Assured')
plt.title('Fig 16: Scatter Plot: MonthlyIncome vs SumAssured')
plt.grid()
plt.show()
```



Observations and Insights:

- There is strong correlation between AgentBonus and SumAssured.
- There is moderate correlation between AgentBonus and Age.
- There is moderate correlation between AgentBonus and CustTenure.
- There is moderate correlation between AgentBonus and MonthlyIncome.
- There is moderate correlation between MonthlyIncome and SumAssured.
- AgentBonus increases when SumAssured increases.
- AgentBonus increases when Age increases.
- AgentBonus increases when CustTenure increases.
- AgentBonus increases when MonthlyIncome increases.
- SumAssured increases when MonthlyIncome increases.

Data Pre-processing

Removing first column (CustID) in the dataset

In [205…

```
# Removing first column in the dataset as it is a auto generated number

df.drop('CustID', axis=1, inplace=True)
```

Removed first column from the dataset as it is an auto generated number.

In [206…

```
df.head() # Returns first 5 rows
```

Out[206…

	AgentBonus	Age	CustTenure	Channel	Occupation	EducationField	Gender	ExistingPr
0	4409	22.0	4.0	Agent	Salaried	Graduate	Female	
1	2214	11.0	2.0	Third Party Partner	Salaried	Graduate	Male	
2	4273	26.0	4.0	Agent	Free Lancer	Post Graduate	Male	
3	1791	11.0	NaN	Third Party Partner	Salaried	Graduate	Female	
4	2955	6.0	NaN	Agent	Small Business	Under Graduate	Male	

Finding missing values in the dataset

In [207…

```
df.isna().sum()  # Count NaN values in all columns of dataset
```

Out[207…
```
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
```

Imputing the missing values - KNNImputer

KNN Imputer is a powerful and versatile method for handling missing data, offering advantages such as data retention, relationship preservation, and adaptability to different data types. It is particularly useful when dealing with non-random missingness and can lead to more accurate and reliable machine-learning models.

In [208…
```
imputer = KNNImputer(n_neighbors=5) # KNNImputer
```

In [209…
```
# Total number of Null values before imputation in the dataset

print('Total number of Null values before imputation in the dataset:', df.isnull().
```

```
Total number of Null values before imputation in the dataset: 1166
```

In [210…
```
df['Age'] = imputer.fit_transform(df[['Age']]) # Impute Age column
df['CustTenure'] = imputer.fit_transform(df[['CustTenure']]) # Impute CustTenure co
df['NumberOfPolicy'] = imputer.fit_transform(df[['NumberOfPolicy']]) # Impute Numbe
df['MonthlyIncome'] = imputer.fit_transform(df[['MonthlyIncome']]) # Impute Monthly
df['ExistingPolicyTenure'] = imputer.fit_transform(df[['ExistingPolicyTenure']]) #
df['SumAssured'] = imputer.fit_transform(df[['SumAssured']]) # Impute SumAssured co
df['CustCareScore'] = imputer.fit_transform(df[['CustCareScore']]) # Impute CustCar
```

In [211…
```
# Total number of Null values after imputation in the dataset

print('Total number of Null values after imputation in the dataset:', df.isnull().s
```

```
Total number of Null values after imputation in the dataset: 0
```

Outliers Detection and Treatment - IQR Method

IQR method is robust to skewed data distributions. It identifies outliers based on percentiles, making it less sensitive to extreme values. IQR method is easy to implement and interpret. It provides a clear range within which most data points should fall, making it a valuable tool for data analysis and quality control.

In [212…

```
# Outliers count

num = ['AgentBonus', 'Age', 'CustTenure', 'ExistingProdType', 'NumberOfPolicy', 'Mo

Q1 = df[num].quantile(0.25)
Q3 = df[num].quantile(0.75)
IQR = Q3 - Q1
((df[num] < (Q1 - 1.5 * IQR)) | (df[num] > (Q3 + 1.5 * IQR))).sum()
```

Out[212…

```
AgentBonus              100
Age                     105
CustTenure               97
ExistingProdType        306
NumberOfPolicy            0
MonthlyIncome           384
Complaint                 0
ExistingPolicyTenure    345
SumAssured              110
LastMonthCalls           12
CustCareScore             0
dtype: int64
```

In [213…

```
# User Defined Function (UDF) to treat outliers

def treat_outlier(x):

    # taking 25,75 percentile of column
    q25=np.percentile(x,25)
    q75=np.percentile(x,75)

    #calculationg IQR range
    IQR=q75-q25
    #Calculating minimum threshold
    lower_bound=q25-(1.5*IQR)
    upper_bound=q75+(1.5*IQR)
    #Capping outliers
    return x.apply(lambda y: upper_bound if y > upper_bound else y).apply(lambda y:
```

In [214…

```
outlier_list = ['AgentBonus', 'Age', 'CustTenure', 'MonthlyIncome', 'ExistingPolicy

# Using for loop to iterate over numerical columns and calling treat_outlier UDF to

for i in df[outlier_list]:
    df[i]=treat_outlier(df[i])
```

In [215…

```
# Outliers count (after treatment)

Q1 = df[outlier_list].quantile(0.25)
Q3 = df[outlier_list].quantile(0.75)
```

```
IQR = Q3 - Q1
((df[outlier_list] < (Q1 - 1.5 * IQR)) | (df[outlier_list] > (Q3 + 1.5 * IQR))).sum
```

Out[215…

```
AgentBonus              0
Age                     0
CustTenure              0
MonthlyIncome           0
ExistingPolicyTenure    0
SumAssured              0
LastMonthCalls          0
dtype: int64
```

Variables Transformation (Feature Encoding)

Machine learning models require numerical input. However, real-world data often contains non-numeric or categorical data. Transformations, especially encoding techniques, convert this data into a format that models can interpret. For instance, one-hot encoding transforms categorical variables into binary vectors.

In [216…

```
cat = ['Channel', 'Occupation', 'EducationField', 'Gender', 'Designation', 'Marital
```

In [217…

```
df_enc = pd.get_dummies(df, columns=cat, dtype=int, drop_first=True) # One-Hot Enco
```

In [218…

```
df_enc.head() # Returns first 5 rows
```

Out[218…

	AgentBonus	Age	CustTenure	ExistingProdType	NumberOfPolicy	MonthlyIncome	Cor
0	4409.0	22.0	4.000000	3	2.0	20993.0	
1	2214.0	11.0	2.000000	4	4.0	20130.0	
2	4273.0	26.0	4.000000	4	3.0	17090.0	
3	1791.0	11.0	14.469027	3	3.0	17909.0	
4	2955.0	6.0	14.469027	3	4.0	18468.0	

5 rows × 34 columns

Business Insights (EDA)

- Channel through which acquisition of customer is done - highest is Agent and lowest is Online.
- Occupation of customer - highest is Salaried and lowest is Free Lancer.
- Field of education of customer - highest is Graduate and lowest is MBA.
- Gender of customer - highest is Male and lowest is Female.
- Designation of customer in their organization - highest is Executive and lowest is VP.
- Marital status of customer - highest is Married and lowest is Divorced.
- Customer belongs to which zone in India - highest is West and lowest is South.

- Frequency of payment selected by customer - highest is Half Yearly and lowest is Quarterly.
- Bonus amount for agent increase when max of sum assured in all existing policies of customer increase.

Scaling Data

In [219…

```
# Copy all the predictor variables into X dataframe
X = df_enc.drop('AgentBonus', axis=1)

# Copy target into y dataframe
y = df_enc['AgentBonus']
```

In [220…

```
df_scaled = X.apply(zscore) # scaling the dataset
```

In [221…

```
df_scaled.head() # Returns first 5 rows
```

Out[221…

	Age	CustTenure	ExistingProdType	NumberOfPolicy	MonthlyIncome	Complaint
0	0.912567	-1.241130	-0.678318	-1.08068	-0.400493	1.575525
1	-0.402305	-1.481257	0.306267	0.30006	-0.619100	-0.634709
2	1.390703	-1.241130	0.306267	-0.39031	-1.389166	1.575525
3	-0.402305	0.015818	-0.678318	-0.39031	-1.181704	1.575525
4	-0.999974	0.015818	-0.678318	0.30006	-1.040103	-0.634709

5 rows × 33 columns

K-Means Clustering

In [222…

```
# KMeans clustring

k_means = KMeans(n_clusters = 2, random_state=1)
k_means.fit(df_scaled)
```

Out[222…

```
▾ KMeans
KMeans(n_clusters=2, random_state=1)
```

In [223…

```
print('Sum of Squares for K = 2:', k_means.inertia_)
```

```
Sum of Squares for K = 2: 135363.5831902912
```

In [224…

```
# Calculating WSS for other values of K - Elbow Method

wss = []
for i in range(1,11):
    KM = KMeans(n_clusters=i, random_state=1)
```

```
    KM.fit(df_scaled)
    wss.append(KM.inertia_)
    print('wss for '+ str(i)+ ' clusters is : ' +str(KM.inertia_))
```

```
wss for 1 clusters is : 149160.00000000003
wss for 2 clusters is : 135363.5831902912
wss for 3 clusters is : 125164.60825551946
wss for 4 clusters is : 119226.96508352408
wss for 5 clusters is : 114348.52558298888
wss for 6 clusters is : 110362.2308405131
wss for 7 clusters is : 105580.78855054671
wss for 8 clusters is : 102513.02717312284
wss for 9 clusters is : 98345.8187467369
wss for 10 clusters is : 94459.3565914122
```

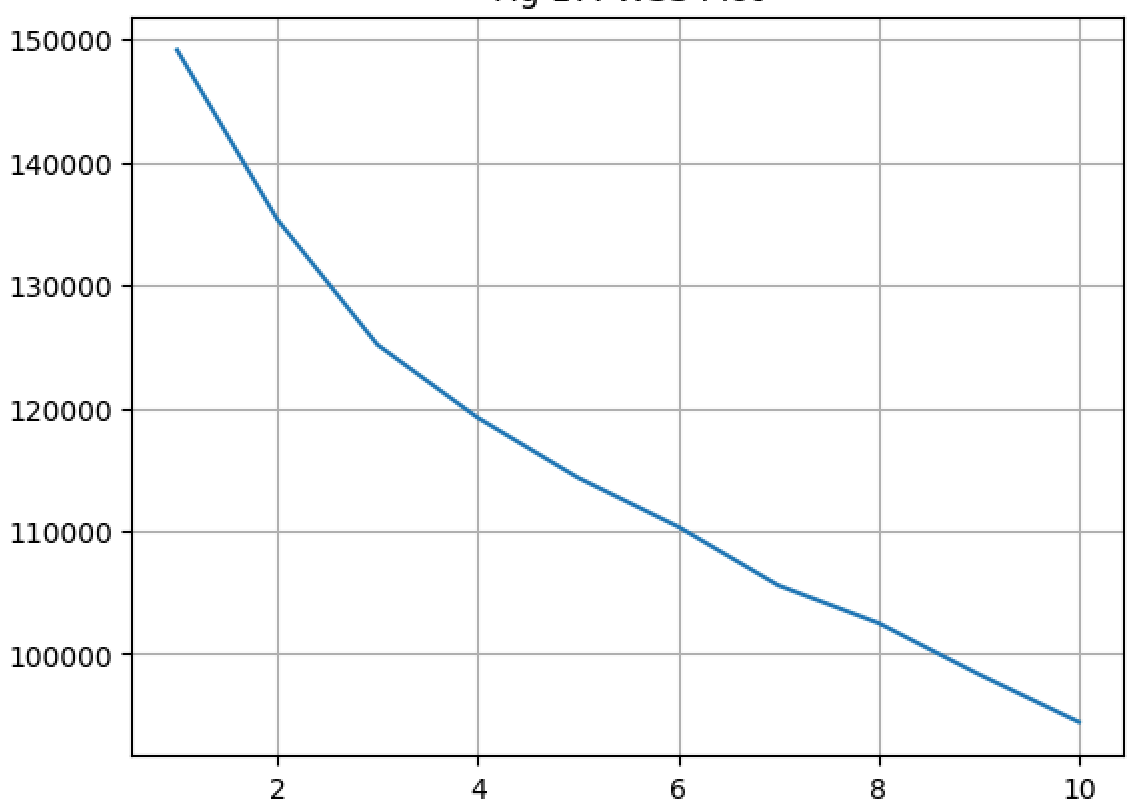
WSS reduces as K keeps increasing.

WSS Plot

In [225…

```
# WSS Plot

plt.plot(range(1,11), wss)
plt.title('Fig 17: WSS Plot')
plt.grid()
plt.show()
```

Fig 17: WSS Plot



K-Means clustering with K=3

In [226…

```
k_means = KMeans(n_clusters = 3,random_state=1)
k_means.fit(df_scaled)
labels = k_means.labels_
```

In [227…

```
# Calculating silhouette_score
print('Cluster evaluation for 3 clusters:', silhouette_score(df_scaled,labels,rando
```

```
Cluster evaluation for 3 clusters: 0.12632426310595582
```

K-Means clustering with K=4

In [228…

```
k_means = KMeans(n_clusters = 4,random_state=1)
k_means.fit(df_scaled)
labels = k_means.labels_
```

In [229…

```
# Calculating silhouette_score
print('Cluster evaluation for 4 clusters:', silhouette_score(df_scaled,labels,rando
```

```
Cluster evaluation for 4 clusters: 0.1336284979220062
```

K-Means clustering with K=5

In [230…

```
k_means = KMeans(n_clusters = 5,random_state=1)
k_means.fit(df_scaled)
labels = k_means.labels_
```

In [231…

```
# Calculating silhouette_score
print('Cluster evaluation for 5 clusters:', silhouette_score(df_scaled,labels,rando
```

```
Cluster evaluation for 5 clusters: 0.1466067192609726
```

Silhouette score is highest for K = 5, among all values of K considered.

Silhouette score is better for 5 clusters than for 4 clusters. So, final clusters will be 5.

In [232…

```
df_enc_clust = df_enc.copy()

df_enc_clust["Clus_kmeans5"] = labels # Appending Clusters to the original dataset

df_enc_clust.head() # Returns first 5 rows
```

Out[232…

	AgentBonus	Age	CustTenure	ExistingProdType	NumberOfPolicy	MonthlyIncome	Cor
0	4409.0	22.0	4.000000	3	2.0	20993.0	
1	2214.0	11.0	2.000000	4	4.0	20130.0	
2	4273.0	26.0	4.000000	4	3.0	17090.0	
3	1791.0	11.0	14.469027	3	3.0	17909.0	
4	2955.0	6.0	14.469027	3	4.0	18468.0	

5 rows × 35 columns

Silhouette Score (K from 2 to 6)

In [233…

```
# Silhouette Analysis

range_n_clusters=[2,3,4,5,6]
for num_clusters in range_n_clusters:

    # initialize K means
    kmeans=KMeans(n_clusters=num_clusters, random_state=1)
    kmeans.fit(df_scaled)
    cluster_labels=kmeans.labels_

    #Silhouette Score
    silhouette_avg = silhouette_score(df_scaled,cluster_labels)
    print("For n_clusters={0}, the silhouette score is {1}".format(num_clusters, si
```

```
For n_clusters=2, the silhouette score is 0.10150975905145038
For n_clusters=3, the silhouette score is 0.12632426310595582
For n_clusters=4, the silhouette score is 0.1336284979220062
For n_clusters=5, the silhouette score is 0.1466067192609726
For n_clusters=6, the silhouette score is 0.11751499705810707
```

It can be observed that the maximum Silhouette Score is obtained for K=5, followed by K=4.

Cluster Profiling

In [234…

```
df_enc_clust.Clus_kmeans5.value_counts().sort_index() # Frequency of each distinct
```

Out[234…

```
Clus_kmeans5
0    1833
1     225
2     241
3    1822
4     399
Name: count, dtype: int64
```

In [235…

```
clust_profile = df_enc_clust.groupby('Clus_kmeans5').mean()
clust_profile['Freq'] = df_enc_clust.Clus_kmeans5.value_counts().sort_index()
clust_profile
```

Out[235…

Clus_kmeans5	AgentBonus	Age	CustTenure	ExistingProdType	NumberOfPolicy	Mo
0	3970.774414	13.891407	14.010411	3.671577	3.615078	
1	6071.346667	21.604444	20.764444	3.826667	3.617778	
2	3896.259336	14.184384	13.766350	3.771784	3.560166	
3	3977.178924	14.188175	14.134193	3.664654	3.486758	
4	3844.203008	13.381794	13.486781	3.751880	3.669501	

5 rows × 35 columns

Business Insights (K-Means Clustering)

- Cluster 0: Large size of customers with max of sum assured in all existing policies of customer, age and tenure of customer in organization, gross monthly income of customer and max tenure in all existing policies of customer
- Cluster 1: Small size of customers with highest max of sum assured in all existing policies of customer, age, tenure of customer in organization, gross monthly income of customer and max tenure in all existing policies of customer
- Cluster 2: Small size of customers with max of sum assured in all existing policies of customer, age and tenure of customer in organization, gross monthly income of customer and max tenure in all existing policies of customer
- Cluster 3: Large size of customers with max of sum assured in all existing policies of customer, age and tenure of customer in organization, gross monthly income of customer and max tenure in all existing policies of customer
- Cluster 4: Medium size of customers with max of sum assured in all existing policies of customer, age and tenure of customer in organization, gross monthly income of customer and max tenure in all existing policies of customer

Model Building

Modeling Approach Used: Regression analysis

Reason: Regression analysis is used to predict a continuous target variable from one or multiple independent variables. Typically, regression analysis is used with naturally-occurring variables, rather than variables that have been manipulated through experimentation.

In this project, AgentBonus is considered as a continuous target variable and remanning variables (Age, CustTenure, Channel, Occupation, EducationField, Gender, ExistingProdType, Designation, NumberOfPolicy, MaritalStatus, MonthlyIncome, Complaint, ExistingPolicyTenure, SumAssured, Zone, PaymentMethod, LastMonthCalls, CustCareScore) are considered as independent variables.

Splitting the data into Train and Test sets

In [236…
```
X = df_enc.drop('AgentBonus', axis=1)
y = df_enc[['AgentBonus']]
```

In [237…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
```

In [238…
```
print('Shape of X_train set:',X_train.shape)
print('Shape of X_test set:',X_test.shape)
print('Shape of y_train set:',y_train.shape)
print('Shape of y_test set:',y_test.shape)
```

```
Shape of X_train set: (3164, 33)
Shape of X_test set: (1356, 33)
Shape of y_train set: (3164, 1)
Shape of y_test set: (1356, 1)
```

In [239…
```
# Scaling Train and Test dataset

X_train_scaled  = X_train.apply(zscore)
X_test_scaled = X_test.apply(zscore)
y_train_scaled = y_train.apply(zscore)
y_test_scaled = y_test.apply(zscore)
```

Linear Regression Model

In [240…
```
LinReg = LinearRegression()

LinReg.fit(X_train_scaled,y_train_scaled)
```

Out[240…
```
▾ LinearRegression
LinearRegression()
```

In [241…
```
# Model coefficients

coefficients = LinReg.coef_[0]
print('Model coefficients:', coefficients)
```

```
Model coefficients: [ 1.33240578e-01  1.41773748e-01  3.42110803e-02  7.87770029e-03
  1.27473088e-01  1.13602512e-02  8.20197096e-02  5.94517209e-01
 -4.80170446e-03  9.53005593e-03  5.76978986e-03  1.80972270e-03
 -1.08404228e-01 -1.36791777e-01 -1.81463992e-01  1.20211858e-03
 -4.20170459e-02 -1.48576192e-02 -1.81747582e-02  1.66799294e-05
  1.21088337e-02 -1.39876871e-01 -1.28944524e-01 -5.44836249e-02
 -4.77649119e-03 -1.88321853e-02  1.05712401e-03  2.12674810e-02
  6.44221680e-03  1.89562493e-02  3.18324897e-02  1.13552114e-02
 -2.83960821e-02]
```

In [242…
```
# Model intercept
```

```
intercept = LinReg.intercept_[0]
print('Model intercept:', intercept)
```

Model intercept: 2.8926479685822997e-16

In [243…

```
# Predictions on the Train and Test dataset

y_pred_train = LinReg.predict(X_train_scaled)
y_pred_test = LinReg.predict(X_test_scaled)
```

In [244…

```
# Calculate performance metrics on the Train dataset

MSE_train = mean_squared_error(y_train_scaled,y_pred_train)
RMSE_train = np.sqrt(mean_squared_error(y_train_scaled,y_pred_train))
R_squared_train = r2_score(y_train_scaled, y_pred_train)
Adj_R_squared_train = 1 - (1-LinReg.score(X_train_scaled, y_train_scaled))*(len(y_t
MAE_train = mean_absolute_error(y_train_scaled,y_pred_train)
MAPE_train = mean_absolute_percentage_error(y_train_scaled,y_pred_train)
```

In [245…

```
resultsDf_train = pd.DataFrame({'MSE': [MSE_train], 'RMSE': [RMSE_train], 'R-square
resultsDf_train
```

Out[245…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Linear Regression - Train set	0.19822	0.44522	0.80178	0.79969	0.351211	1.926033

In [246…

```
# Calculate performance metrics on the Test dataset

MSE_test = mean_squared_error(y_test_scaled,y_pred_test)
RMSE_test = np.sqrt(mean_squared_error(y_test_scaled,y_pred_test))
R_squared_test = r2_score(y_test_scaled, y_pred_test)
Adj_R_squared_test = 1 - (1-LinReg.score(X_test_scaled, y_test_scaled))*(len(y_test
MAE_test = mean_absolute_error(y_test_scaled,y_pred_test)
MAPE_test = mean_absolute_percentage_error(y_test_scaled,y_pred_test)
```

In [247…

```
resultsDf_test = pd.DataFrame({'MSE': [MSE_test], 'RMSE': [RMSE_test], 'R-squared':
resultsDf_test
```

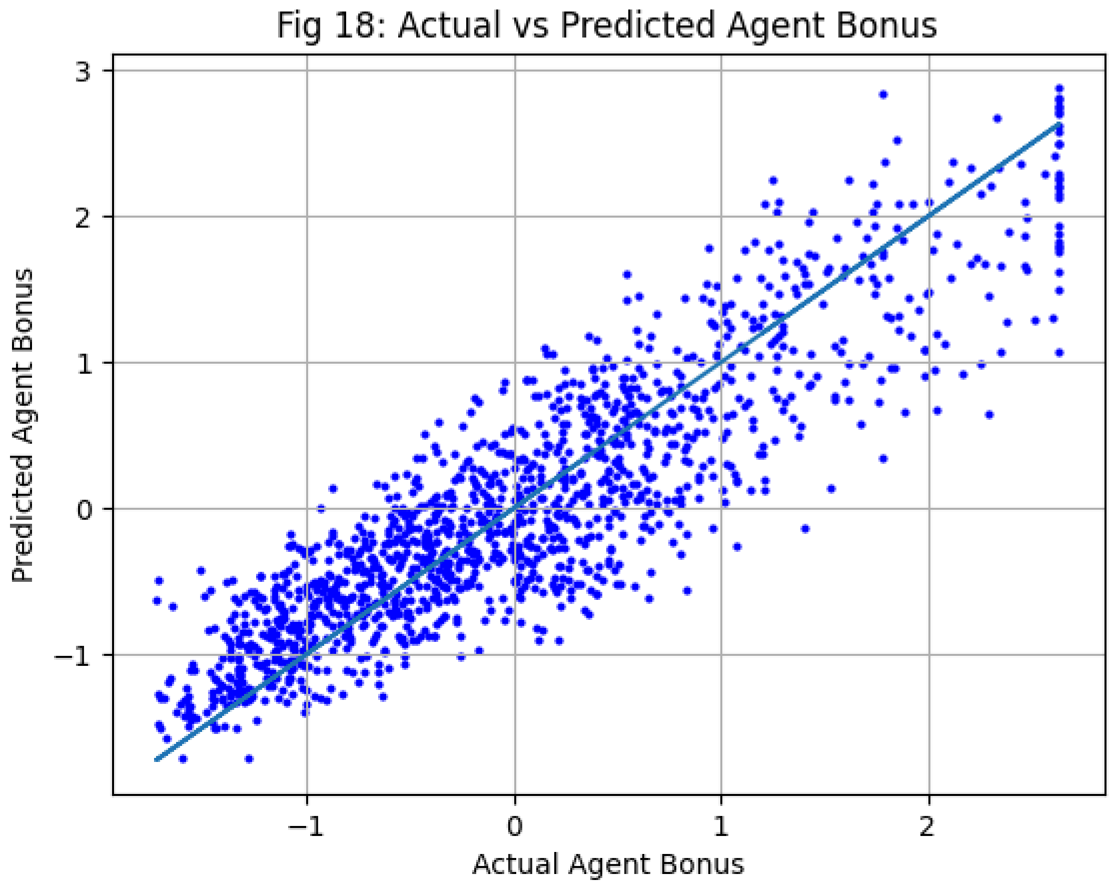
Out[247…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Linear Regression - Test set	0.210326	0.458613	0.789674	0.784424	0.363047	2.42708

In [248…

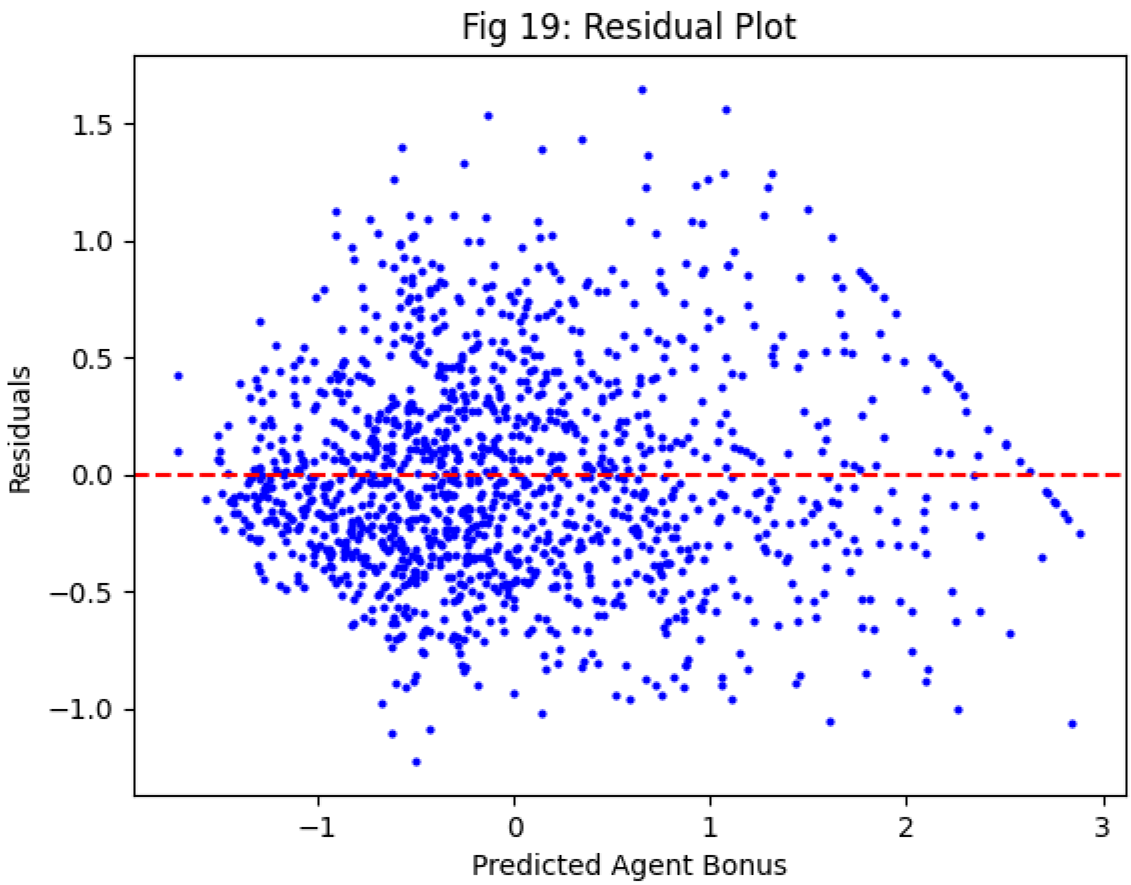
```
# Actual vs Predicted Plot

plt.plot(y_test_scaled, y_test_scaled)
plt.scatter(y_test_scaled, y_pred_test, s=4, c="blue")
plt.xlabel('Actual Agent Bonus')
plt.ylabel('Predicted Agent Bonus')
plt.title('Fig 18: Actual vs Predicted Agent Bonus')
```

```
plt.grid()
plt.show()
```



In [249…

```
# Residual Plot

residuals = y_test_scaled - y_pred_test
plt.scatter(y_pred_test, residuals, s=4, c="blue")
plt.xlabel('Predicted Agent Bonus')
plt.ylabel('Residuals')
plt.title('Fig 19: Residual Plot')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



In [250…

```
resultsDf = pd.concat([resultsDf_train, resultsDf_test])
resultsDf
```

Out[250…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Linear Regression - Train set	0.198220	0.445220	0.801780	0.799690	0.351211	1.926033
Linear Regression - Test set	0.210326	0.458613	0.789674	0.784424	0.363047	2.427080

Model Interpretation:

- Predicted Agent Bonus increases when Actual Agent Bonus increase.
- Residuals are scattered randomly around zero.
- Lower value of MSE/RMSE, MAE/MAPE of regression model (test) indicates that it can predict the value of a response variable in absolute terms.
- Higher value of R-squared / Adj. R-squared of regression model (test) indicates that the predictor variables can explain the variation in the response variable.

Lasso Regression Model

In [251…
```
LasReg = Lasso(alpha=0.1, random_state=1)
LasReg.fit(X_train_scaled,y_train_scaled)
```

Out[251…
```
▾ Lasso
Lasso(alpha=0.1, random_state=1)
```

In [252…
```
# Model coefficients

coefficients = LasReg.coef_
print('Model coefficients:', coefficients)
```

```
Model coefficients: [ 0.08742831  0.09283326  0.          0.          0.14781751  0.
  0.01342333  0.58963989  0.          0.          0.         -0.
 -0.          0.         -0.         -0.          0.         -0.
 -0.         -0.          0.         -0.         -0.          0.
  0.         -0.          0.         -0.          0.          0.
  0.          0.         -0.        ]
```

In [253…
```
# Model intercept

intercept = LasReg.intercept_[0]
print('Model intercept:', intercept)
```

```
Model intercept: 3.3953133863007113e-16
```

In [254…
```
# Predictions on the Train and Test dataset

y_pred_train = LasReg.predict(X_train_scaled)
y_pred_test = LasReg.predict(X_test_scaled)
```

In [255…
```
# Calculate performance metrics on the Train dataset

MSE_train = mean_squared_error(y_train_scaled,y_pred_train)
RMSE_train = np.sqrt(mean_squared_error(y_train_scaled,y_pred_train))
R_squared_train = r2_score(y_train_scaled, y_pred_train)
Adj_R_squared_train = 1 - (1-LasReg.score(X_train_scaled, y_train_scaled))*(len(y_t
MAE_train = mean_absolute_error(y_train_scaled,y_pred_train)
MAPE_train = mean_absolute_percentage_error(y_train_scaled,y_pred_train)
```

In [256…
```
resultsDf_train = pd.DataFrame({'MSE': [MSE_train], 'RMSE': [RMSE_train], 'R-square
resultsDf_train
```

Out[256…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Lasso Regression - Train set	0.225945	0.475336	0.774055	0.771673	0.375781	1.753539

In [257…
```
# Calculate performance metrics on the Test dataset

MSE_test = mean_squared_error(y_test_scaled,y_pred_test)
RMSE_test = np.sqrt(mean_squared_error(y_test_scaled,y_pred_test))
R_squared_test = r2_score(y_test_scaled, y_pred_test)
```

```
Adj_R_squared_test = 1 - (1-LasReg.score(X_test_scaled, y_test_scaled))*(len(y_test
MAE_test = mean_absolute_error(y_test_scaled,y_pred_test)
MAPE_test = mean_absolute_percentage_error(y_test_scaled,y_pred_test)
```

In [258…

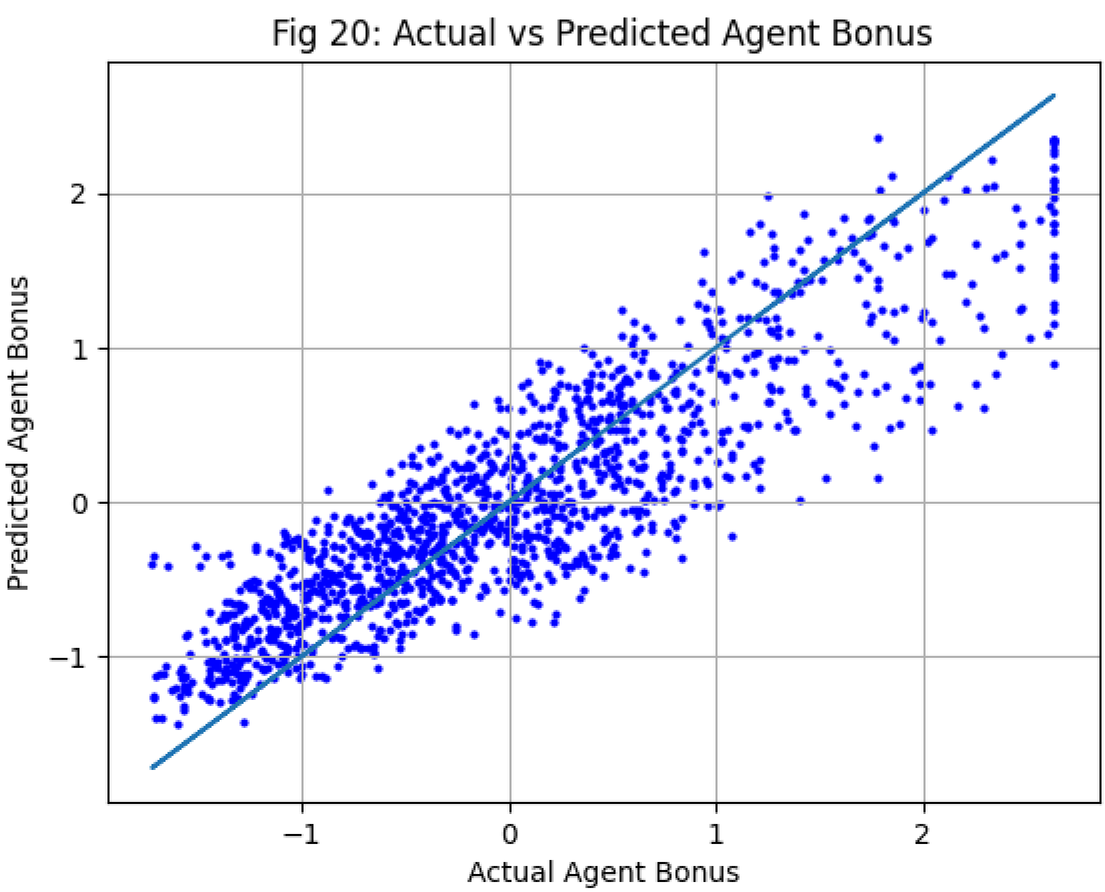
```
resultsDf_test = pd.DataFrame({'MSE': [MSE_test], 'RMSE': [RMSE_test], 'R-squared':
resultsDf_test
```

Out[258…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Lasso Regression - Test set	0.237153	0.486984	0.762847	0.756927	0.387169	1.769022

In [259…

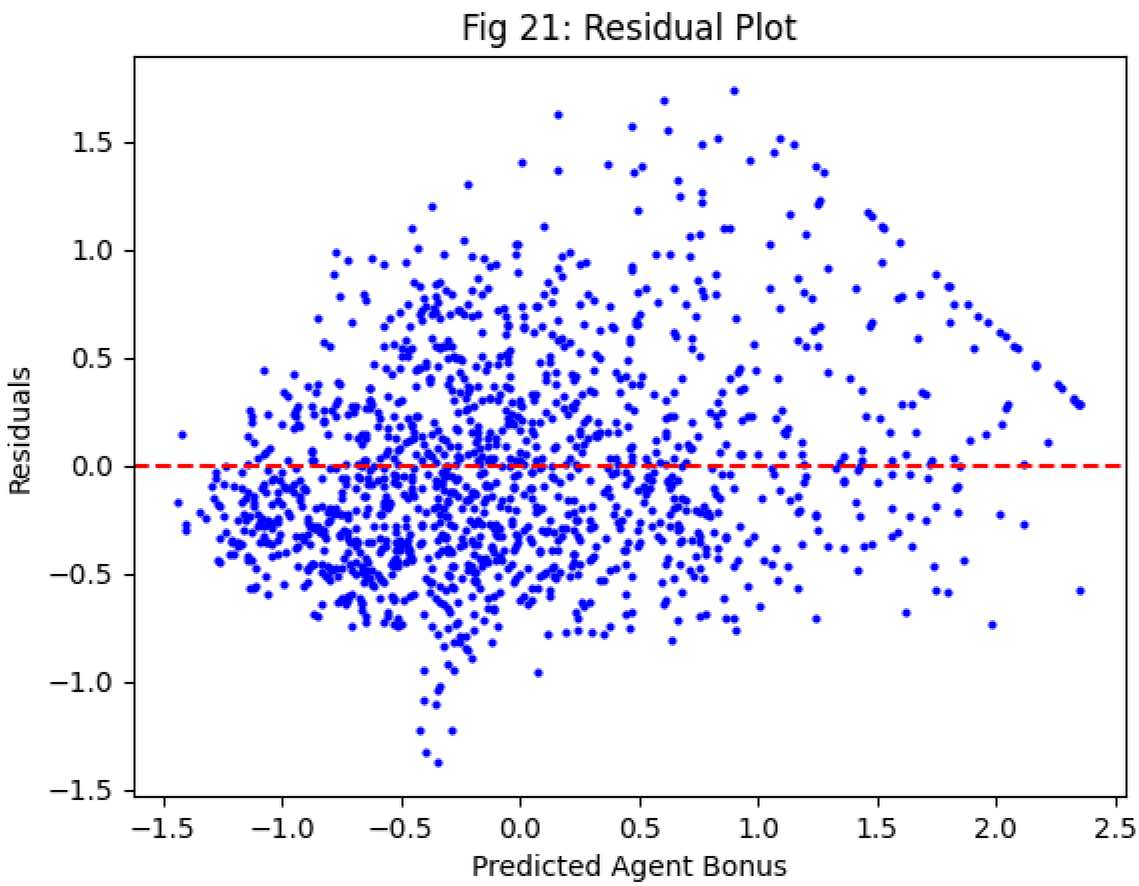
```
# Actual vs Predicted Plot

plt.plot(y_test_scaled, y_test_scaled)
plt.scatter(y_test_scaled, y_pred_test, s=4, c="blue")
plt.xlabel('Actual Agent Bonus')
plt.ylabel('Predicted Agent Bonus')
plt.title('Fig 20: Actual vs Predicted Agent Bonus')
plt.grid()
plt.show()
```



In [260…

```
# Residual Plot

y_pred_test=y_pred_test.reshape(1356,1)
```

```
residuals = y_test_scaled - y_pred_test
plt.scatter(y_pred_test, residuals, s=4, c="blue")
plt.xlabel('Predicted Agent Bonus')
plt.ylabel('Residuals')
plt.title('Fig 21: Residual Plot')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



In [261…

```
resultsDf = pd.concat([resultsDf, resultsDf_train, resultsDf_test])
resultsDf
```

Out[261…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Linear Regression - Train set	0.198220	0.445220	0.801780	0.799690	0.351211	1.926033
Linear Regression - Test set	0.210326	0.458613	0.789674	0.784424	0.363047	2.427080
Lasso Regression - Train set	0.225945	0.475336	0.774055	0.771673	0.375781	1.753539
Lasso Regression - Test set	0.237153	0.486984	0.762847	0.756927	0.387169	1.769022

Model Interpretation:

- Predicted Agent Bonus increases when Actual Agent Bonus increase.
- Residuals are scattered randomly around zero.
- Lower value of MSE/RMSE, MAE/MAPE of regression model (test) indicates that it can predict the value of a response variable in absolute terms.
- Higher value of R-squared / Adj. R-squared of regression model (test) indicates that the predictor variables can explain the variation in the response variable.

Ridge Regression Model

In [262...

```
RidReg = Ridge(random_state=1)
RidReg.fit(X_train_scaled,y_train_scaled)
```

Out[262...

```
▾ Ridge
Ridge(random_state=1)
```

In [263...

```
# Model coefficients

coefficients = RidReg.coef_[0]
print('Model coefficients:', coefficients)
```

```
Model coefficients: [ 1.33315866e-01  1.41780747e-01  3.39800529e-02  7.88996602e-03
  1.27743739e-01  1.14251293e-02  8.20497328e-02  5.94294556e-01
 -4.80393191e-03  9.53097625e-03  5.75863958e-03  1.77970416e-03
 -8.68497882e-02 -9.97729950e-02 -1.43684195e-01  1.13979487e-03
 -4.09466635e-02 -1.45996792e-02 -1.74923578e-02  6.59151290e-05
  1.21123382e-02 -1.39337848e-01 -1.28521569e-01 -5.42359601e-02
 -4.67509910e-03 -1.88318755e-02  1.10110656e-03  2.11389095e-02
  6.42724905e-03  1.87644946e-02  3.17066019e-02  1.13276782e-02
 -2.82147273e-02]
```

In [264...

```
# Model intercept

intercept = RidReg.intercept_[0]
print('Model intercept:', intercept)
```

```
Model intercept: 2.9340654885638657e-16
```

In [265...

```
# Predictions on the Train and Test dataset

y_pred_train = RidReg.predict(X_train_scaled)
y_pred_test = RidReg.predict(X_test_scaled)
```

In [266...

```
# Calculate performance metrics on the Train dataset

MSE_train = mean_squared_error(y_train_scaled,y_pred_train)
RMSE_train = np.sqrt(mean_squared_error(y_train_scaled,y_pred_train))
R_squared_train = r2_score(y_train_scaled, y_pred_train)
Adj_R_squared_train = 1 - (1-RidReg.score(X_train_scaled, y_train_scaled))*(len(y_t
MAE_train = mean_absolute_error(y_train_scaled,y_pred_train)
MAPE_train = mean_absolute_percentage_error(y_train_scaled,y_pred_train)
```

In [267…

```
resultsDf_train = pd.DataFrame({'MSE': [MSE_train], 'RMSE': [RMSE_train], 'R-square
resultsDf_train
```

Out[267…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Ridge Regression - Train set	0.198224	0.445224	0.801776	0.799686	0.351218	1.925817

In [268…

```
# Calculate performance metrics on the Test dataset

MSE_test = mean_squared_error(y_test_scaled,y_pred_test)
RMSE_test = np.sqrt(mean_squared_error(y_test_scaled,y_pred_test))
R_squared_test = r2_score(y_test_scaled, y_pred_test)
Adj_R_squared_test = 1 - (1-RidReg.score(X_test_scaled, y_test_scaled))*(len(y_test
MAE_test = mean_absolute_error(y_test_scaled,y_pred_test)
MAPE_test = mean_absolute_percentage_error(y_test_scaled,y_pred_test)
```

In [269…

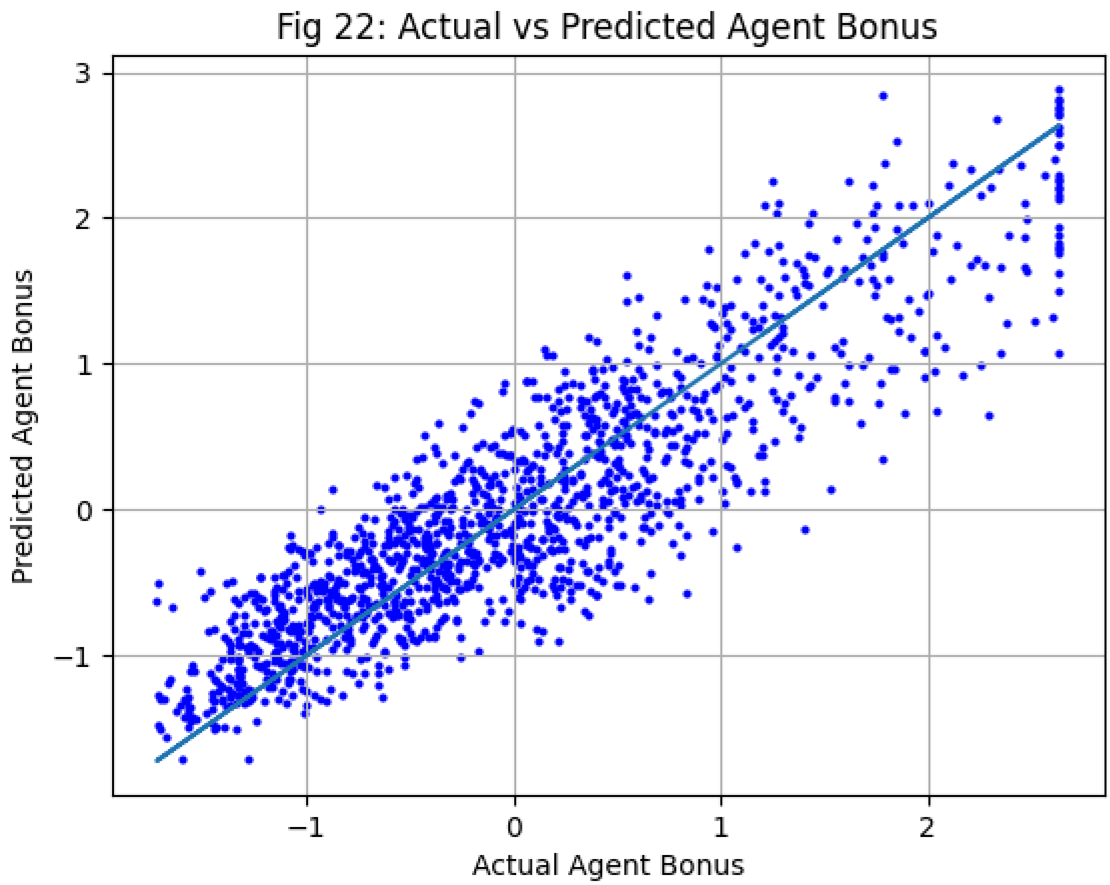
```
resultsDf_test = pd.DataFrame({'MSE': [MSE_test], 'RMSE': [RMSE_test], 'R-squared':
resultsDf_test
```

Out[269…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Ridge Regression - Test set	0.210318	0.458604	0.789682	0.784432	0.363065	2.425757

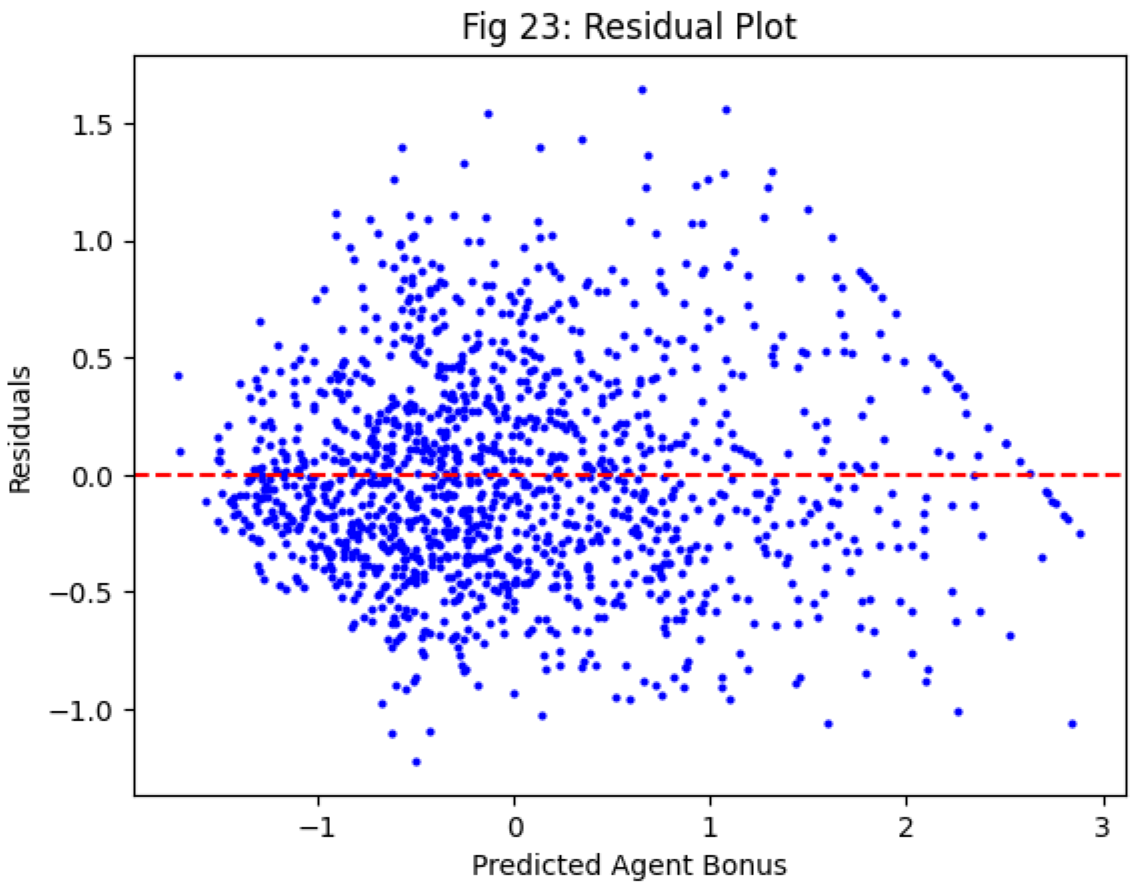
In [270…

```
# Actual vs Predicted Plot

plt.plot(y_test_scaled, y_test_scaled)
plt.scatter(y_test_scaled, y_pred_test, s=4, c="blue")
plt.xlabel('Actual Agent Bonus')
plt.ylabel('Predicted Agent Bonus')
plt.title('Fig 22: Actual vs Predicted Agent Bonus')
plt.grid()
plt.show()
```



In [271…

```
# Residual Plot

residuals = y_test_scaled - y_pred_test
plt.scatter(y_pred_test, residuals, s=4, c="blue")
plt.xlabel('Predicted Agent Bonus')
plt.ylabel('Residuals')
plt.title('Fig 23: Residual Plot')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



In [272…

```
resultsDf = pd.concat([resultsDf, resultsDf_train, resultsDf_test])
resultsDf
```

Out[272…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Linear Regression - Train set	0.198220	0.445220	0.801780	0.799690	0.351211	1.926033
Linear Regression - Test set	0.210326	0.458613	0.789674	0.784424	0.363047	2.427080
Lasso Regression - Train set	0.225945	0.475336	0.774055	0.771673	0.375781	1.753539
Lasso Regression - Test set	0.237153	0.486984	0.762847	0.756927	0.387169	1.769022
Ridge Regression - Train set	0.198224	0.445224	0.801776	0.799686	0.351218	1.925817
Ridge Regression - Test set	0.210318	0.458604	0.789682	0.784432	0.363065	2.425757

Model Interpretation:

- Predicted Agent Bonus increases when Actual Agent Bonus increase.
- Residuals are scattered randomly around zero.

- Lower value of MSE/RMSE, MAE/MAPE of regression model (test) indicates that it can predict the value of a response variable in absolute terms.
- Higher value of R-squared / Adj. R-squared of regression model (test) indicates that the predictor variables can explain the variation in the response variable.

Random Forest Regression Model

In [273…

```
# Initialise a Random Forest Classifier

RFReg = RandomForestRegressor(random_state=1)
RFReg.fit(X_train_scaled,y_train_scaled)
```

Out[273…

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

In [274…

```
# Predictions on the Train and Test dataset

y_pred_train = RFReg.predict(X_train_scaled)
y_pred_test = RFReg.predict(X_test_scaled)
```

In [275…

```
# Calculate performance metrics on the Train dataset

MSE_train = mean_squared_error(y_train_scaled,y_pred_train)
RMSE_train = np.sqrt(mean_squared_error(y_train_scaled,y_pred_train))
R_squared_train = r2_score(y_train_scaled, y_pred_train)
Adj_R_squared_train = 1 - (1-RFReg.score(X_train_scaled, y_train_scaled))*(len(y_tr
MAE_train = mean_absolute_error(y_train_scaled,y_pred_train)
MAPE_train = mean_absolute_percentage_error(y_train_scaled,y_pred_train)
```

In [276…

```
resultsDf_train = pd.DataFrame({'MSE': [MSE_train], 'RMSE': [RMSE_train], 'R-square
resultsDf_train
```

Out[276…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Random Forest Regression - Train set	0.01952	0.139714	0.98048	0.980274	0.105904	0.615818

In [277…

```
# Calculate performance metrics on the Test dataset

MSE_test = mean_squared_error(y_test_scaled,y_pred_test)
RMSE_test = np.sqrt(mean_squared_error(y_test_scaled,y_pred_test))
R_squared_test = r2_score(y_test_scaled, y_pred_test)
Adj_R_squared_test = 1 - (1-RFReg.score(X_test_scaled, y_test_scaled))*(len(y_test_
MAE_test = mean_absolute_error(y_test_scaled,y_pred_test)
MAPE_test = mean_absolute_percentage_error(y_test_scaled,y_pred_test)
```

In [278…

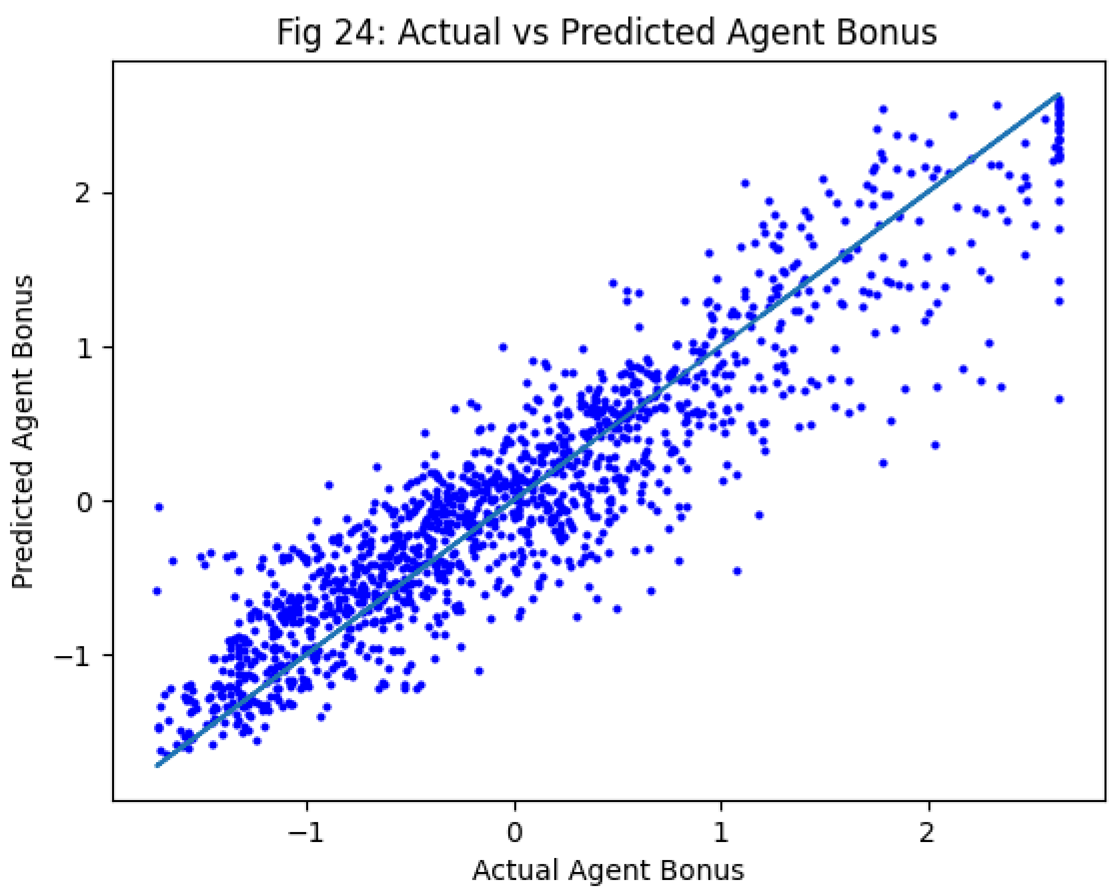
```
resultsDf_test = pd.DataFrame({'MSE': [MSE_test], 'RMSE': [RMSE_test], 'R-squared':
resultsDf_test
```

Out[278...

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Random Forest Regression - Test set	0.159737	0.399672	0.840263	0.836275	0.304728	1.327642

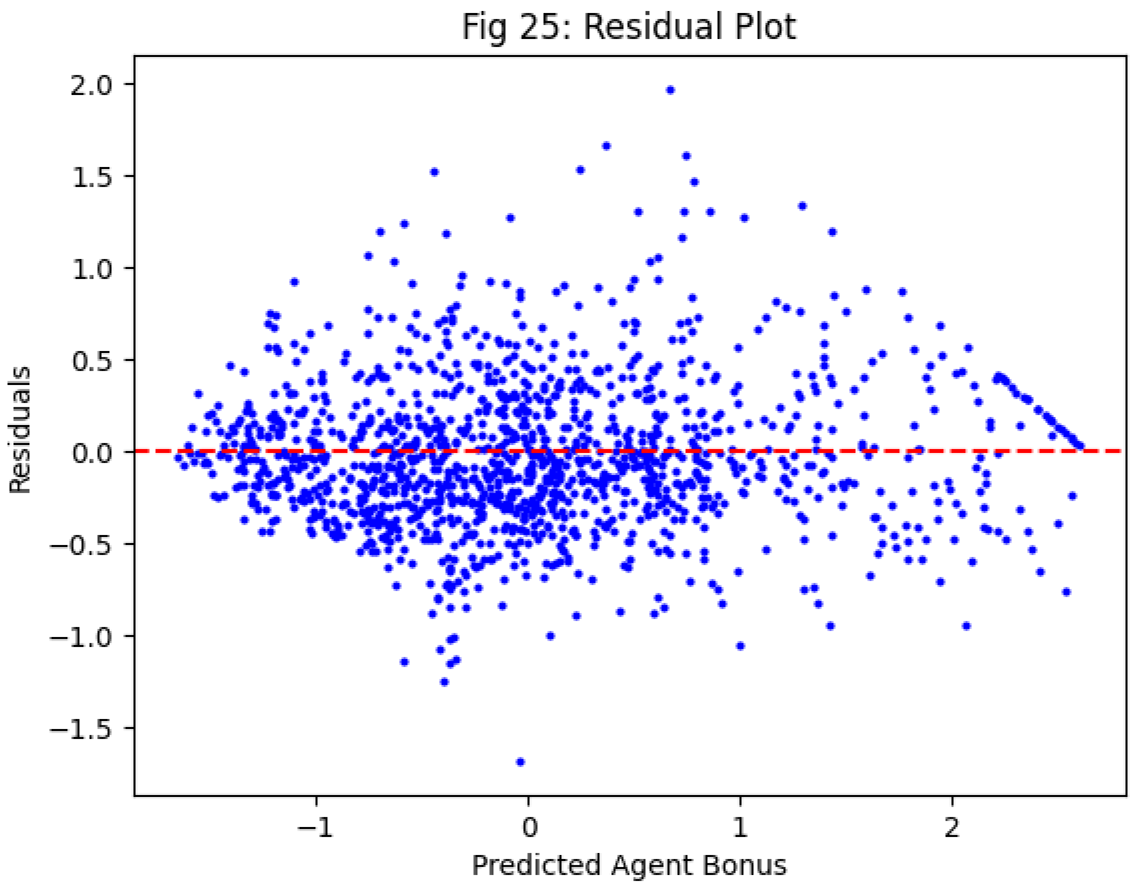
In [279...

```
# Actual vs Predicted Plot

plt.plot(y_test_scaled, y_test_scaled)
plt.scatter(y_test_scaled, y_pred_test, s=4, c="blue")
plt.xlabel('Actual Agent Bonus')
plt.ylabel('Predicted Agent Bonus')
plt.title('Fig 24: Actual vs Predicted Agent Bonus')
#plt.grid()
plt.show()
```



In [280...

```
# Residual Plot

y_pred_test=y_pred_test.reshape(1356,1)

residuals = y_test_scaled - y_pred_test
plt.scatter(y_pred_test, residuals, s=4, c="blue")
plt.xlabel('Predicted Agent Bonus')
plt.ylabel('Residuals')
plt.title('Fig 25: Residual Plot')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



In [281…

```
resultsDf = pd.concat([resultsDf, resultsDf_train, resultsDf_test])
resultsDf
```

Out[281…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Linear Regression - Train set	0.198220	0.445220	0.801780	0.799690	0.351211	1.926033
Linear Regression - Test set	0.210326	0.458613	0.789674	0.784424	0.363047	2.427080
Lasso Regression - Train set	0.225945	0.475336	0.774055	0.771673	0.375781	1.753539
Lasso Regression - Test set	0.237153	0.486984	0.762847	0.756927	0.387169	1.769022
Ridge Regression - Train set	0.198224	0.445224	0.801776	0.799686	0.351218	1.925817
Ridge Regression - Test set	0.210318	0.458604	0.789682	0.784432	0.363065	2.425757
Random Forest Regression - Train set	0.019520	0.139714	0.980480	0.980274	0.105904	0.615818
Random Forest Regression - Test set	0.159737	0.399672	0.840263	0.836275	0.304728	1.327642

Model Interpretation:

- Predicted Agent Bonus increases when Actual Agent Bonus increase.
- Residuals are scattered randomly around zero.
- Lower value of MSE/RMSE, MAE/MAPE of regression model (test) indicates that it can predict the value of a response variable in absolute terms.
- Higher value of R-squared / Adj. R-squared of regression model (test) indicates that the predictor variables can explain the variation in the response variable.

XGB Regression Model

In [282…

```
XGBReg = xg.XGBRegressor(random_state=1)
XGBReg.fit(X_train_scaled,y_train_scaled)
```

Out[282...

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=Non
e,
             enable_categorical=False, eval_metric=None, feature_types=Non
e,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=Non
e,
```

In [283...

```python
# Model intercept

intercept = XGBReg.intercept_[0]
print('Model intercept:', intercept)
```

```
Model intercept: -1.9703956e-09
```

In [284...

```python
# Predictions on the Train and Test dataset

y_pred_train = XGBReg.predict(X_train_scaled)
y_pred_test = XGBReg.predict(X_test_scaled)
```

In [285...

```python
# Calculate performance metrics on the Train dataset

MSE_train = mean_squared_error(y_train_scaled,y_pred_train)
RMSE_train = np.sqrt(mean_squared_error(y_train_scaled,y_pred_train))
R_squared_train = r2_score(y_train_scaled, y_pred_train)
Adj_R_squared_train = 1 - (1-XGBReg.score(X_train_scaled, y_train_scaled))*(len(y_t
MAE_train = mean_absolute_error(y_train_scaled,y_pred_train)
MAPE_train = mean_absolute_percentage_error(y_train_scaled,y_pred_train)
```

In [286...

```python
resultsDf_train = pd.DataFrame({'MSE': [MSE_train], 'RMSE': [RMSE_train], 'R-square
resultsDf_train
```

Out[286...

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
XGB Regression - Train set	0.010667	0.103281	0.989333	0.989221	0.073909	0.424946

In [287...

```python
# Calculate performance metrics on the Test dataset

MSE_test = mean_squared_error(y_test_scaled,y_pred_test)
RMSE_test = np.sqrt(mean_squared_error(y_test_scaled,y_pred_test))
R_squared_test = r2_score(y_test_scaled, y_pred_test)
Adj_R_squared_test = 1 - (1-XGBReg.score(X_test_scaled, y_test_scaled))*(len(y_test
MAE_test = mean_absolute_error(y_test_scaled,y_pred_test)
MAPE_test = mean_absolute_percentage_error(y_test_scaled,y_pred_test)
```

In [288…

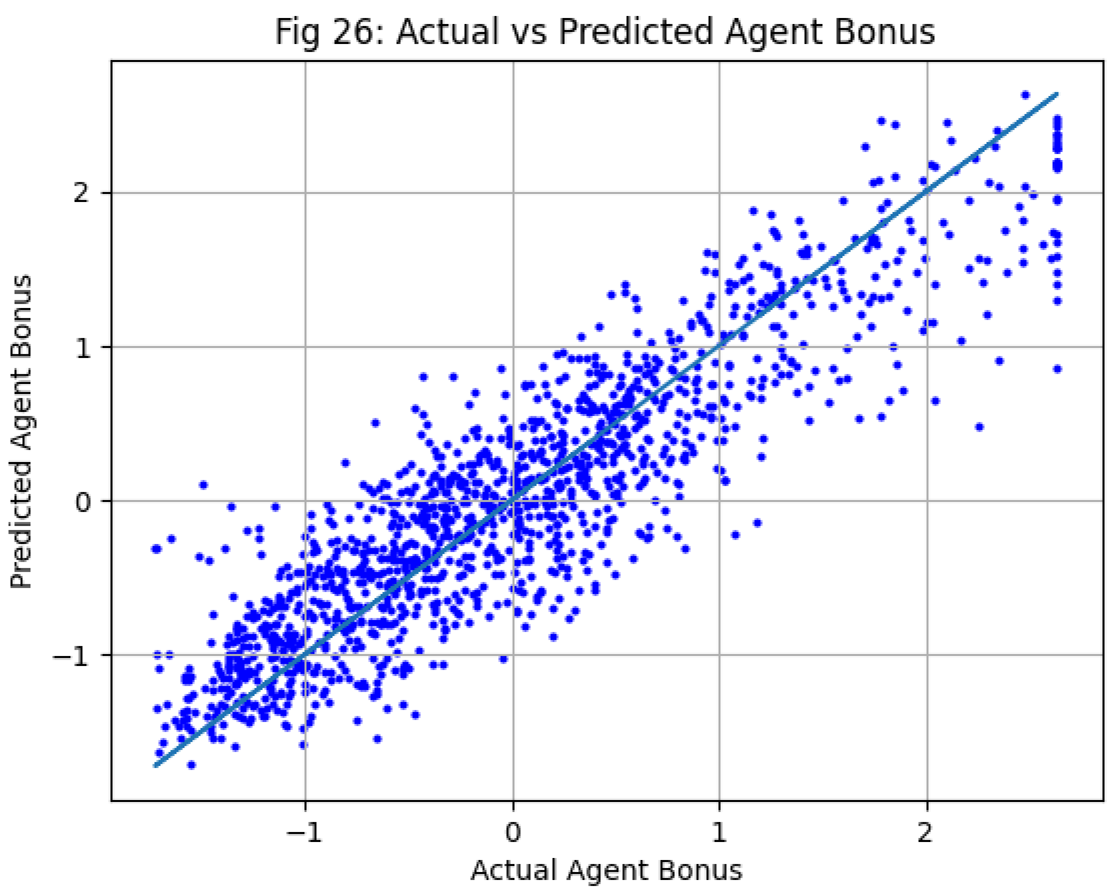
```
resultsDf_test = pd.DataFrame({'MSE': [MSE_test], 'RMSE': [RMSE_test], 'R-squared':
resultsDf_test
```

Out[288…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
XGB Regression - Test set	0.190459	0.436416	0.809541	0.804787	0.340289	1.594048

In [289…

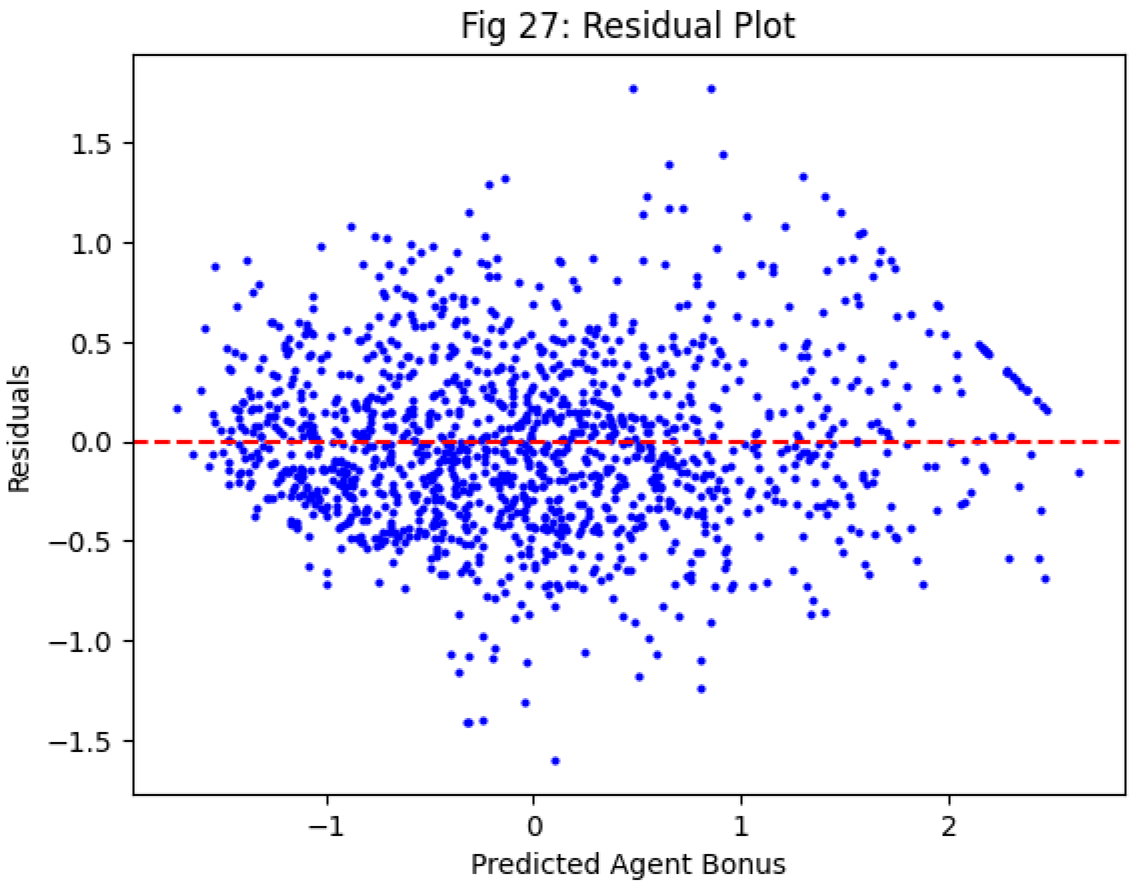
```
# Actual vs Predicted Plot

plt.plot(y_test_scaled, y_test_scaled)
plt.scatter(y_test_scaled, y_pred_test, s=4, c="blue")
plt.xlabel('Actual Agent Bonus')
plt.ylabel('Predicted Agent Bonus')
plt.title('Fig 26: Actual vs Predicted Agent Bonus')
plt.grid()
plt.show()
```



In [290…

```
# Residual Plot

y_pred_test=y_pred_test.reshape(1356,1)

residuals = y_test_scaled - y_pred_test
plt.scatter(y_pred_test, residuals, s=4, c="blue")
plt.xlabel('Predicted Agent Bonus')
plt.ylabel('Residuals')
```

```python
plt.title('Fig 27: Residual Plot')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



In [291…

```python
resultsDf = pd.concat([resultsDf, resultsDf_train, resultsDf_test])
resultsDf
```

Out[291...

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Linear Regression - Train set	0.198220	0.445220	0.801780	0.799690	0.351211	1.926033
Linear Regression - Test set	0.210326	0.458613	0.789674	0.784424	0.363047	2.427080
Lasso Regression - Train set	0.225945	0.475336	0.774055	0.771673	0.375781	1.753539
Lasso Regression - Test set	0.237153	0.486984	0.762847	0.756927	0.387169	1.769022
Ridge Regression - Train set	0.198224	0.445224	0.801776	0.799686	0.351218	1.925817
Ridge Regression - Test set	0.210318	0.458604	0.789682	0.784432	0.363065	2.425757
Random Forest Regression - Train set	0.019520	0.139714	0.980480	0.980274	0.105904	0.615818
Random Forest Regression - Test set	0.159737	0.399672	0.840263	0.836275	0.304728	1.327642
XGB Regression - Train set	0.010667	0.103281	0.989333	0.989221	0.073909	0.424946
XGB Regression - Test set	0.190459	0.436416	0.809541	0.804787	0.340289	1.594048

Model Interpretation:

- Predicted Agent Bonus increases when Actual Agent Bonus increase.
- Residuals are scattered randomly around zero.
- Lower value of MSE/RMSE, MAE/MAPE of regression model (test) indicates that it can predict the value of a response variable in absolute terms.
- Higher value of R-squared / Adj. R-squared of regression model (test) indicates that the predictor variables can explain the variation in the response variable.

AdaBoost Regression Model

In [292...

```
ABReg = AdaBoostRegressor(random_state=1)
ABReg.fit(X_train_scaled,y_train_scaled)
```

Out[292...

```
AdaBoostRegressor
AdaBoostRegressor(random_state=1)
```

In [293...

```
# Predictions on the Train and Test dataset
```

```
y_pred_train = ABReg.predict(X_train_scaled)
y_pred_test = ABReg.predict(X_test_scaled)
```

In [294…

```
# Calculate performance metrics on the Train dataset

MSE_train = mean_squared_error(y_train_scaled,y_pred_train)
RMSE_train = np.sqrt(mean_squared_error(y_train_scaled,y_pred_train))
R_squared_train = r2_score(y_train_scaled, y_pred_train)
Adj_R_squared_train = 1 - (1-ABReg.score(X_train_scaled, y_train_scaled))*(len(y_tr
MAE_train = mean_absolute_error(y_train_scaled,y_pred_train)
MAPE_train = mean_absolute_percentage_error(y_train_scaled,y_pred_train)
```

In [295…

```
resultsDf_train = pd.DataFrame({'MSE': [MSE_train], 'RMSE': [RMSE_train], 'R-square
resultsDf_train
```

Out[295…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
AdaBoost Regression - Train set	0.218844	0.467808	0.781156	0.778849	0.39196	2.588765

In [296…

```
# Calculate performance metrics on the Test dataset

MSE_test = mean_squared_error(y_test_scaled,y_pred_test)
RMSE_test = np.sqrt(mean_squared_error(y_test_scaled,y_pred_test))
R_squared_test = r2_score(y_test_scaled, y_pred_test)
Adj_R_squared_test = 1 - (1-ABReg.score(X_test_scaled, y_test_scaled))*(len(y_test_
MAE_test = mean_absolute_error(y_test_scaled,y_pred_test)
MAPE_test = mean_absolute_percentage_error(y_test_scaled,y_pred_test)
```

In [297…

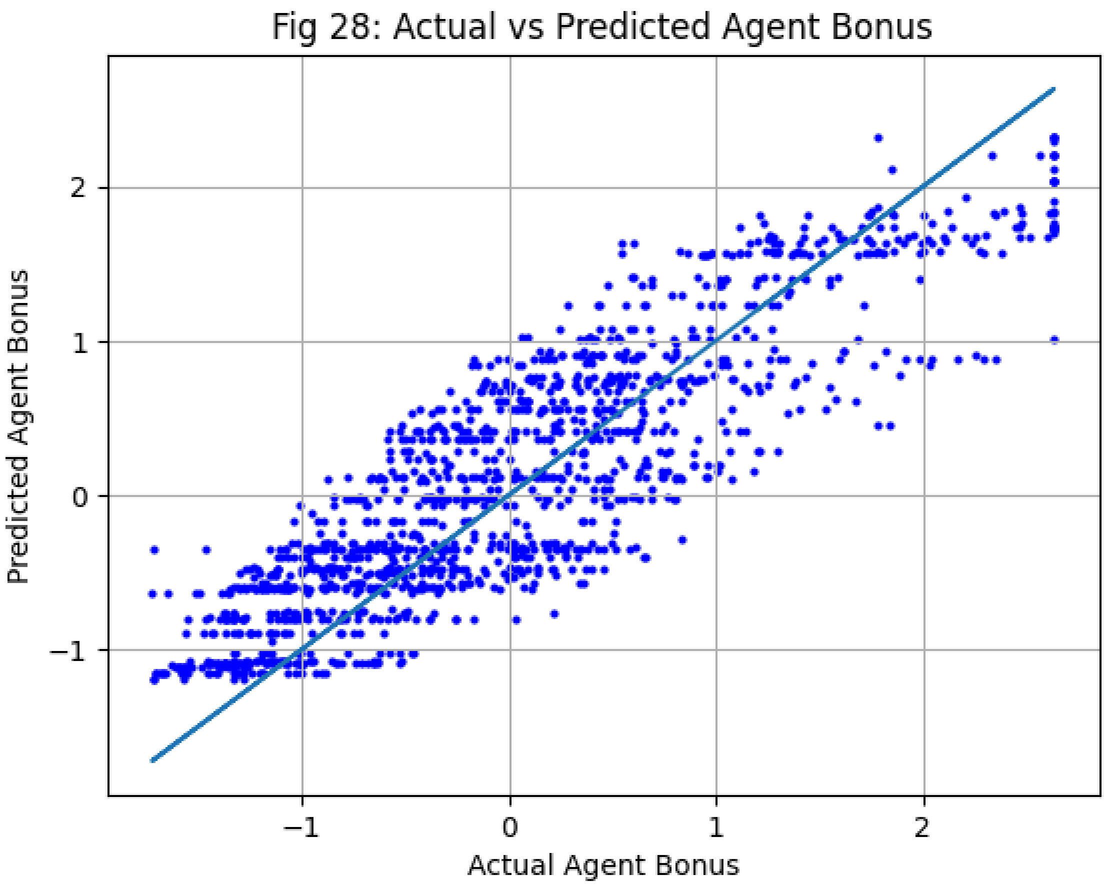
```
resultsDf_test = pd.DataFrame({'MSE': [MSE_test], 'RMSE': [RMSE_test], 'R-squared':
resultsDf_test
```

Out[297…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
AdaBoost Regression - Test set	0.241848	0.49178	0.758152	0.752115	0.409625	3.26109

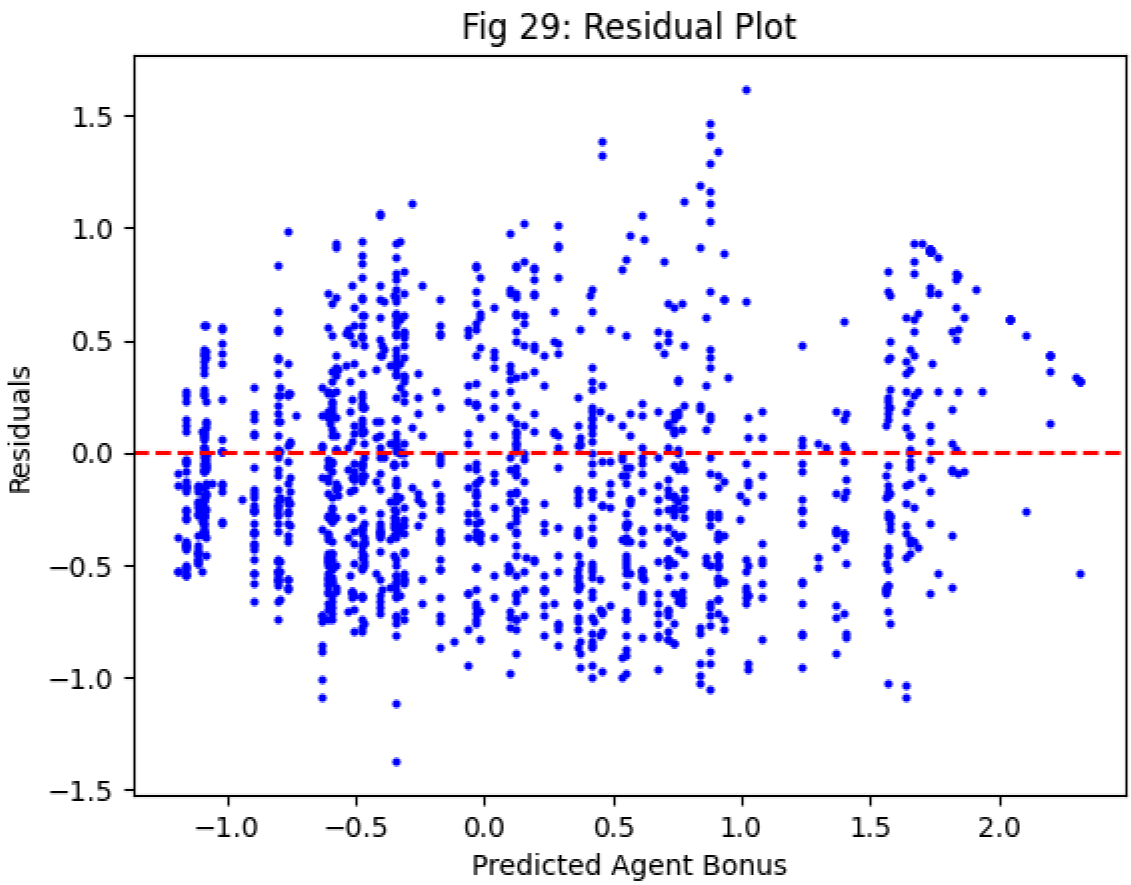
In [298…

```
# Actual vs Predicted Plot

plt.plot(y_test_scaled, y_test_scaled)
plt.scatter(y_test_scaled, y_pred_test, s=4, c="blue")
plt.xlabel('Actual Agent Bonus')
plt.ylabel('Predicted Agent Bonus')
plt.title('Fig 28: Actual vs Predicted Agent Bonus')
plt.grid()
plt.show()
```



In [299…

```
# Residual Plot

y_pred_test=y_pred_test.reshape(1356,1)

residuals = y_test_scaled - y_pred_test
plt.scatter(y_pred_test, residuals, s=4, c="blue")
plt.xlabel('Predicted Agent Bonus')
plt.ylabel('Residuals')
plt.title('Fig 29: Residual Plot')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



In [300…

```
resultsDf = pd.concat([resultsDf, resultsDf_train, resultsDf_test])
resultsDf
```

Out[300…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Linear Regression - Train set	0.198220	0.445220	0.801780	0.799690	0.351211	1.926033
Linear Regression - Test set	0.210326	0.458613	0.789674	0.784424	0.363047	2.427080
Lasso Regression - Train set	0.225945	0.475336	0.774055	0.771673	0.375781	1.753539
Lasso Regression - Test set	0.237153	0.486984	0.762847	0.756927	0.387169	1.769022
Ridge Regression - Train set	0.198224	0.445224	0.801776	0.799686	0.351218	1.925817
Ridge Regression - Test set	0.210318	0.458604	0.789682	0.784432	0.363065	2.425757
Random Forest Regression - Train set	0.019520	0.139714	0.980480	0.980274	0.105904	0.615818
Random Forest Regression - Test set	0.159737	0.399672	0.840263	0.836275	0.304728	1.327642
XGB Regression - Train set	0.010667	0.103281	0.989333	0.989221	0.073909	0.424946
XGB Regression - Test set	0.190459	0.436416	0.809541	0.804787	0.340289	1.594048
AdaBoost Regression - Train set	0.218844	0.467808	0.781156	0.778849	0.391960	2.588765
AdaBoost Regression - Test set	0.241848	0.491780	0.758152	0.752115	0.409625	3.261090

Model Interpretation:

- Predicted Agent Bonus increases when Actual Agent Bonus increase.
- Residuals are scattered randomly around zero.
- Lower value of MSE/RMSE, MAE/MAPE of regression model (test) indicates that it can predict the value of a response variable in absolute terms.
- Higher value of R-squared / Adj. R-squared of regression model (test) indicates that the predictor variables can explain the variation in the response variable.

SVR Regression Model

In [301…

```
SVRReg = SVR(kernel='linear')
SVRReg.fit(X_train_scaled,y_train_scaled)
```

Out[301...

```
SVR
SVR(kernel='linear')
```

In [302...

```
# Model coefficients

coefficients = SVRReg.coef_[0]
print('Model coefficients:', coefficients)
```

```
Model coefficients: [ 0.14740195  0.15486157  0.01164195 -0.00114738  0.14111031  0.
00967051
  0.07898906  0.59089069 -0.00098151  0.01791316 -0.00110302 -0.00807866
 -0.10438481 -0.14092821 -0.19891942 -0.01758532 -0.07198249 -0.02533681
 -0.03487155 -0.00700113  0.00723265 -0.10486991 -0.11581868 -0.05294478
 -0.016162   -0.01399456  0.00538911  0.01624268  0.01098859  0.01631664
  0.02344233  0.00421129 -0.01067535]
```

In [303...

```
# Model intercept

intercept = SVRReg.intercept_[0]
print('Model intercept:', intercept)
```

```
Model intercept: -0.03542130352198503
```

In [304...

```
# Predictions on the Train and Test dataset

y_pred_train = SVRReg.predict(X_train_scaled)
y_pred_test = SVRReg.predict(X_test_scaled)
```

In [305...

```
# Calculate performance metrics on the Train dataset

MSE_train = mean_squared_error(y_train_scaled,y_pred_train)
RMSE_train = np.sqrt(mean_squared_error(y_train_scaled,y_pred_train))
R_squared_train = r2_score(y_train_scaled, y_pred_train)
Adj_R_squared_train = 1 - (1-SVRReg.score(X_train_scaled, y_train_scaled))*(len(y_t
MAE_train = mean_absolute_error(y_train_scaled,y_pred_train)
MAPE_train = mean_absolute_percentage_error(y_train_scaled,y_pred_train)
```

In [306...

```
resultsDf_train = pd.DataFrame({'MSE': [MSE_train], 'RMSE': [RMSE_train], 'R-square
resultsDf_train
```

Out[306...

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
SVR Regression - Train set	0.200825	0.448135	0.799175	0.797058	0.348234	1.891723

In [307...

```
# Calculate performance metrics on the Test dataset

MSE_test = mean_squared_error(y_test_scaled,y_pred_test)
RMSE_test = np.sqrt(mean_squared_error(y_test_scaled,y_pred_test))
R_squared_test = r2_score(y_test_scaled, y_pred_test)
Adj_R_squared_test = 1 - (1-SVRReg.score(X_test_scaled, y_test_scaled))*(len(y_test
```

```
MAE_test = mean_absolute_error(y_test_scaled,y_pred_test)
MAPE_test = mean_absolute_percentage_error(y_test_scaled,y_pred_test)
```

In [308…

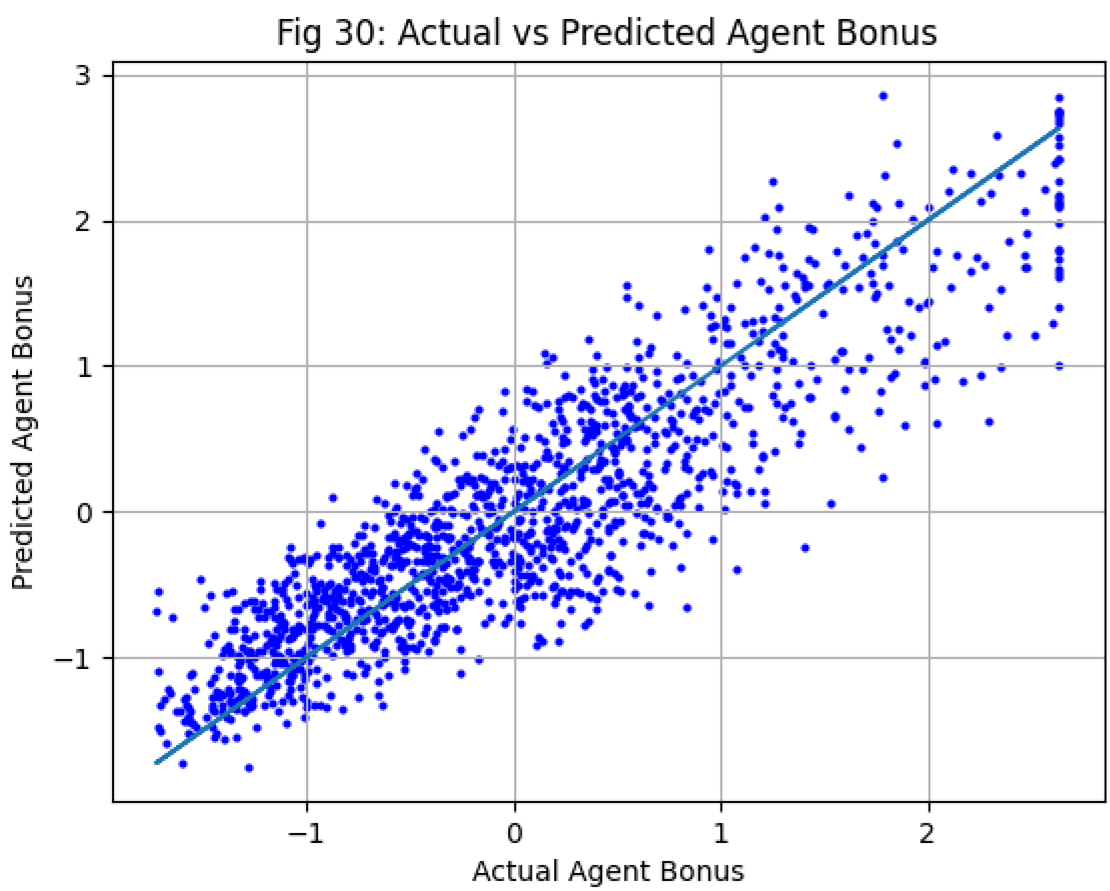
```
resultsDf_test = pd.DataFrame({'MSE': [MSE_test], 'RMSE': [RMSE_test], 'R-squared':
resultsDf_test
```

Out[308…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
SVR Regression - Test set	0.211215	0.459581	0.788785	0.783513	0.359833	2.5285

In [309…

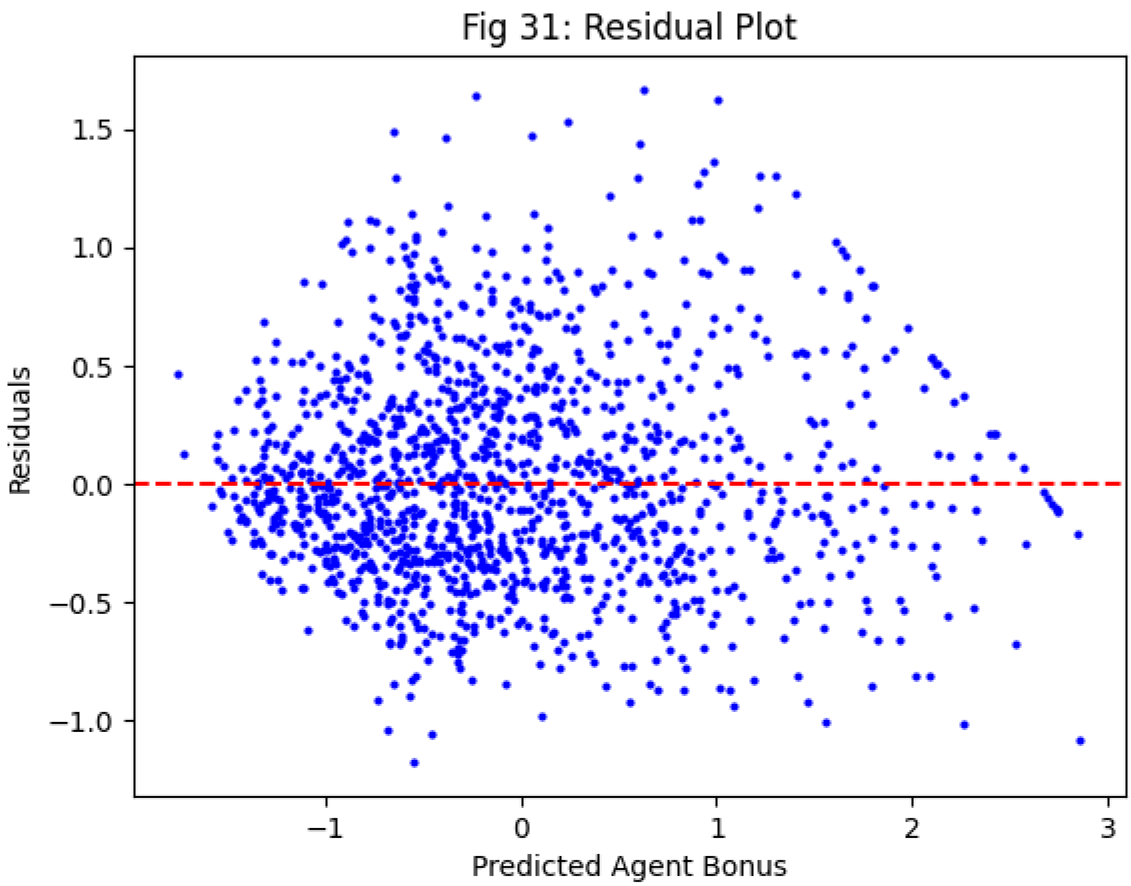
```
# Actual vs Predicted Plot

plt.plot(y_test_scaled, y_test_scaled)
plt.scatter(y_test_scaled, y_pred_test, s=4, c="blue")
plt.xlabel('Actual Agent Bonus')
plt.ylabel('Predicted Agent Bonus')
plt.title('Fig 30: Actual vs Predicted Agent Bonus')
plt.grid()
plt.show()
```



In [310…

```
# Residual Plot

y_pred_test=y_pred_test.reshape(1356,1)

residuals = y_test_scaled - y_pred_test
```

```
plt.scatter(y_pred_test, residuals, s=4, c="blue")
plt.xlabel('Predicted Agent Bonus')
plt.ylabel('Residuals')
plt.title('Fig 31: Residual Plot')
plt.axhline(y=0, color='r', linestyle='--')
plt.show()
```



Model Interpretation:

- Predicted Agent Bonus increases when Actual Agent Bonus increase.
- Residuals are scattered randomly around zero.
- Lower value of MSE/RMSE, MAE/MAPE of regression model (test) indicates that it can predict the value of a response variable in absolute terms.
- Higher value of R-squared / Adj. R-squared of regression model (test) indicates that the predictor variables can explain the variation in the response variable.

Model Comparison and Final Model Selection

In [311…

```
resultsDf = pd.concat([resultsDf, resultsDf_train, resultsDf_test])
resultsDf
```

Out[311…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Linear Regression - Train set	0.198220	0.445220	0.801780	0.799690	0.351211	1.926033
Linear Regression - Test set	0.210326	0.458613	0.789674	0.784424	0.363047	2.427080
Lasso Regression - Train set	0.225945	0.475336	0.774055	0.771673	0.375781	1.753539
Lasso Regression - Test set	0.237153	0.486984	0.762847	0.756927	0.387169	1.769022
Ridge Regression - Train set	0.198224	0.445224	0.801776	0.799686	0.351218	1.925817
Ridge Regression - Test set	0.210318	0.458604	0.789682	0.784432	0.363065	2.425757
Random Forest Regression - Train set	0.019520	0.139714	0.980480	0.980274	0.105904	0.615818
Random Forest Regression - Test set	0.159737	0.399672	0.840263	0.836275	0.304728	1.327642
XGB Regression - Train set	0.010667	0.103281	0.989333	0.989221	0.073909	0.424946
XGB Regression - Test set	0.190459	0.436416	0.809541	0.804787	0.340289	1.594048
AdaBoost Regression - Train set	0.218844	0.467808	0.781156	0.778849	0.391960	2.588765
AdaBoost Regression - Test set	0.241848	0.491780	0.758152	0.752115	0.409625	3.261090
SVR Regression - Train set	0.200825	0.448135	0.799175	0.797058	0.348234	1.891723
SVR Regression - Test set	0.211215	0.459581	0.788785	0.783513	0.359833	2.528500

Final Model: Random Forest Regression. It has highest Adj. R-squared and R-squared values for the Test set. MSE/RMSE and MAE/MAPE values are lowest for Test set as well.

Model Performance Improvement - Random Forest Regression Model

In [312…

```
param_grid = {
    'n_estimators': [10, 50, 100],  # Number of trees in the forest
    'max_depth': [5, 7, 9],     # Maximum depth of the trees
    'min_samples_split': [2, 5, 10], # Minimum number of samples required to split
    'min_samples_leaf': [5, 6, 7],  # Minimum number of samples required at each le
```

```
}

rf_classifier = RandomForestRegressor()

grid_search = GridSearchCV(
    estimator=rf_classifier,
    param_grid=param_grid,
    cv=5,
    scoring='recall',
    n_jobs=-1
)

grid_search.fit(X_train_scaled, y_train_scaled)

print("Best parameters:", grid_search.best_params_)
```

```
Best parameters: {'max_depth': 5, 'min_samples_leaf': 5, 'min_samples_split': 2, 'n_
estimators': 10}
```

In [313…

```
# Random Forest Regression creation - Tuned

RFReg_best = grid_search.best_estimator_
```

In [314…

```
params_used = RFReg_best.get_params()

# Print the parameters

print("Parameters used in the Random Forest Regressor:\n")
for param_name, param_value in params_used.items():
    print(f"{param_name}: {param_value}")
```

```
Parameters used in the Random Forest Regressor:

bootstrap: True
ccp_alpha: 0.0
criterion: squared_error
max_depth: 5
max_features: 1.0
max_leaf_nodes: None
max_samples: None
min_impurity_decrease: 0.0
min_samples_leaf: 5
min_samples_split: 2
min_weight_fraction_leaf: 0.0
n_estimators: 10
n_jobs: None
oob_score: False
random_state: None
verbose: 0
warm_start: False
```

In [315…

```
# Predictions on the Train and Test dataset

y_pred_train = RFReg_best.predict(X_train_scaled)
y_pred_test = RFReg_best.predict(X_test_scaled)
```

In [316…

```
# Calculate performance metrics on the Train dataset

MSE_train = mean_squared_error(y_train_scaled,y_pred_train)
RMSE_train = np.sqrt(mean_squared_error(y_train_scaled,y_pred_train))
R_squared_train = r2_score(y_train_scaled, y_pred_train)
Adj_R_squared_train = 1 - (1-RFReg_best.score(X_train_scaled, y_train_scaled))*(len
MAE_train = mean_absolute_error(y_train_scaled,y_pred_train)
MAPE_train = mean_absolute_percentage_error(y_train_scaled,y_pred_train)
```

In [317…

```
resultsDf_train_tuned = pd.DataFrame({'MSE': [MSE_train], 'RMSE': [RMSE_train], 'R-
resultsDf_train_tuned
```

Out[317…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Random Forest Regression (Tuned) - Train set	0.163039	0.403781	0.836961	0.835242	0.312387	1.79359

In [318…

```
# Calculate performance metrics on the Test dataset

MSE_test = mean_squared_error(y_test_scaled,y_pred_test)
RMSE_test = np.sqrt(mean_squared_error(y_test_scaled,y_pred_test))
R_squared_test = r2_score(y_test_scaled, y_pred_test)
Adj_R_squared_test = 1 - (1-RFReg_best.score(X_test_scaled, y_test_scaled))*(len(y_
MAE_test = mean_absolute_error(y_test_scaled,y_pred_test)
MAPE_test = mean_absolute_percentage_error(y_test_scaled,y_pred_test)
```

In [319…

```
resultsDf_test_tuned = pd.DataFrame({'MSE': [MSE_test], 'RMSE': [RMSE_test], 'R-squ
resultsDf_test_tuned
```

Out[319…

	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Random Forest Regression (Tuned) - Test set	0.187532	0.43305	0.812468	0.807787	0.336253	1.990554

In [320…

```
resultsDf_tuned = pd.concat([resultsDf_train_tuned, resultsDf_test_tuned])
resultsDf_tuned
```

Out[320…

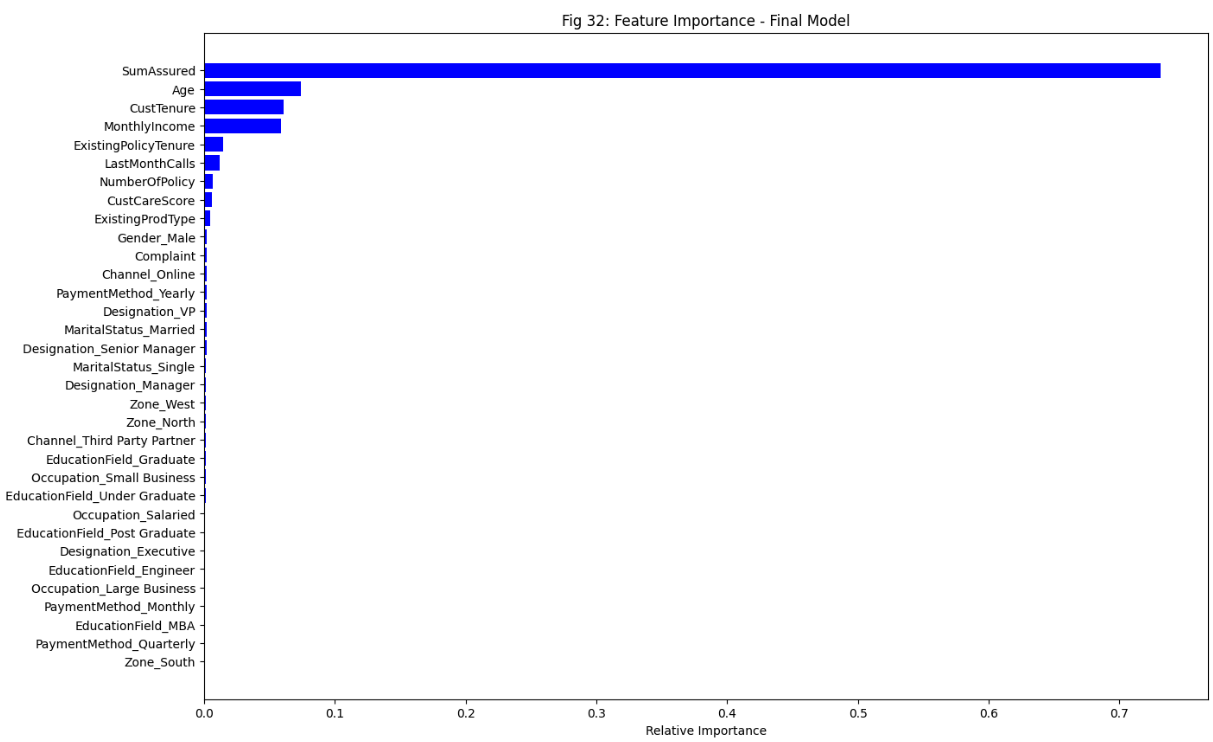
	MSE	RMSE	R-squared	Adj. R-squared	MAE	MAPE
Random Forest Regression (Tuned) - Train set	0.163039	0.403781	0.836961	0.835242	0.312387	1.793590
Random Forest Regression (Tuned) - Test set	0.187532	0.433050	0.812468	0.807787	0.336253	1.990554

Random Forest Regression remains final model. It has highest Adj. R-squared and R-squared values for the Test set. MSE/RMSE and MAE/MAPE values are lowest for Test set as well.

Feature Importance based on Final Model

In [321…

```
# Feature importance based on Final Model

feature_names = X_train_scaled.columns
importances = RFReg.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(15, 10))
plt.title("Fig 32: Feature Importance - Final Model")
plt.barh(range(len(indices)), importances[indices], color="blue", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



Observations and Insights:

- SumAssured, Age, CustTenure, MonthlyIncome, ExistingPolicyTenure, LastMonthCalls, NumberOfPolicy and CustCareScore are the most important factors for life insurance company.

Actionable Insights

- **SumAssured:** Max of sum assured in all existing policies of customer is the most important factor for the life insurance company.

- **Age:** Age of customer is the most important factor for the life insurance company.
- **CustTenure:** Tenure of customer in organization is the most important factor for the life insurance company.
- **MonthlyIncome:** Gross monthly income of customer is the most important factor for the life insurance company.
- **ExistingPolicyTenure:** Max tenure in all existing policies of customer is the most important factor for the life insurance company.
- **LastMonthCalls:** Total calls attempted by company to a customer for cross sell is the most important factor for the life insurance company.
- **NumberOfPolicy:** Total number of existing policy of a customer is the most important factor for the life insurance company.
- **CustCareScore:** Customer satisfaction score given by customer in previous service call is the most important factor for the life insurance company.

Business Recommendations

- Life insurance company can launch advertisement campaigns (print / social media) on policies with high sum assured which in turn can increase agent bonus.
- Life insurance company can target customers with higher age for high sum assured policies which in turn can increase agent bonus.
- Life insurance company can target customers with higher tenure in organization for more policies which in turn can increase agent bonus.
- Life insurance company can target customers with higher gross monthly income which can increase number of policies sold to the customers.
- Life insurance company can target customers with higher max tenure in all existing policies which can increase number of policies sold to the customers.
- Life insurance company can increase number of calls for cross sell which can increase number of policies sold to the customers.
- Life insurance company can target customers for higher number of policies which in turn can increase agent bonus.
- Life insurance company can improve customer satisfaction score which can increase number of policies sold to the customers.
- Life insurance company can design engagement activities for their high performing agents which are having high average bonus.
- Life insurance company can design upskill programs for their low performing agents which are having low average bonus.