Introduction

Objective: Premium paid by the customer is the major revenue source for insurance companies. Default in premium payments results in significant revenue losses and hence insurance companies would like to know upfront which type of customers would default premium payments. The objective of this project is to predict the probability that a customer will default the premium payment, so that the insurance agent can proactively reach out to the policy holder to follow up for the payment of premium.

Goal:

Dataset

The dataset contains the following information about 79854 policy holders:

1. id: Unique customer ID
2. perc_premium_paid_by_cash_credit (later "Perc_premium_paid"): What % of the premium was paid by cash payments?
3. age_in_days (later "Age"): age of the customer in days
4. Income: Income of the customer
5. Marital Status: Married/Unmarried, Married (1), unmarried (0)
6. Veh_owned: Number of vehicles owned (1-3)
7. Count_3-6_months_late (later "3-6_late"): # of times premium was paid 3-6 months late
8. Count_6-12_months_late (later "6-12_late"): # of times premium was paid 6-12 months late
9. Count_more_than_12_months_late (later "12_more_late"): # of times premium was paid more than 12 months late
10. Risk_score:Risk score of customer
11. No_of_dep: Number of dependents in the family on the customer (1-4)
12. Accomodation: Owned (1), Rented (0)
13. no_of_premiums_paid (later "Premiums_paid"): # of premiums paid till date
14. sourcing_channel (later "Source"): channel through which customer was sourced
15. residence_area_type (later "Res_type"): Residence type of the customer
16. premium (later "Premium"): Amount of premium
17. renewal (later "Renewal"): Y variable - 0 indicates that customer has not renewed the premium and 1 indicates that customer has renewed the premium

Import libraries

In [372…

```
#Import necessary libraries
#For calculations:
import numpy as np
import pandas as pd
```

```
#For visualizations
import seaborn as sns
# sns.set()
import matplotlib.pyplot as plt
%matplotlib inline

#For metrics
import sklearn.metrics as metrics

#Disable warnings
import warnings
warnings.filterwarnings('ignore')

#Data presentation:
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", 200)

from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import StandardScaler

from sklearn import metrics
from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_auc_scor

import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.tools.tools import add_constant

from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from imblearn.over_sampling import SMOTE
```

Import data

In [373…

```
#Save to "data", then make a copy and save as "df"
data = pd.read_excel('premium.xlsx', sheet_name='premium')
df = data.copy()
```

Data overview

In [374…

```
#View top five rows
df.head(10)
```

Out[374…

	id	perc_premium_paid_by_cash_credit	age_in_days	age_in_years (Added)	Income	Count_3-6_months_late	12_mo
0	75339	0.007	18991	51	90262600	0	
1	27647	0.000	24828	67	53821900	0	

	id	perc_premium_paid_by_cash_credit	age_in_days	age_in_years (Added)	Income	Count_3-6_months_late	12_mo
2	71799	0.164	16070	43	46803140	0	
3	802	0.469	16072	44	32175090	1	
4	4645	0.042	20086	54	25051240	0	
5	51193	0.072	18256	49	21075130	0	
6	34371	0.064	18993	51	20986030	0	
7	75652	0.440	23365	63	17471210	0	
8	69392	0.080	22272	60	16874010	0	
9	29578	0.036	15707	43	12847560	0	

- Seeing that there are large count numbers in the variables of 3-6, 6-12 months, it would follow that these figures are reflective of a long term trend beyond the timeframe of a single policy
- Fix "Marital Status" to eliminate space
- Consider renaming some variables to shorten and simplify names
- Note that age is in days instead of years

In [375…

```
#View data information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 79853 entries, 0 to 79852
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   id                                79853 non-null  int64  
 1   perc_premium_paid_by_cash_credit  79853 non-null  float64
 2   age_in_days                       79853 non-null  int64  
 3   age_in_years (Added)              79853 non-null  int64  
 4   Income                            79853 non-null  int64  
 5   Count_3-6_months_late             79853 non-null  int64  
 6   Count_6-12_months_late            79853 non-null  int64  
 7   Count_more_than_12_months_late    79853 non-null  int64  
 8   Marital Status                    79853 non-null  int64  
 9   Veh_Owned                         79853 non-null  int64  
 10  No_of_dep                         79853 non-null  int64  
 11  Accomodation                      79853 non-null  int64  
 12  risk_score                        79853 non-null  float64
 13  no_of_premiums_paid               79853 non-null  int64  
 14  sourcing_channel                  79853 non-null  object 
 15  residence_area_type               79853 non-null  object 
 16  premium                           79853 non-null  int64  
 17  renewal                           79853 non-null  int64  
dtypes: float64(2), int64(14), object(2)
memory usage: 11.0+ MB
```

- There are no null values
- Three data types: Int, float, and object. This last one will need to be converted to category

In [376…

```
#View shape of the dataset
df.shape
print('There are {} rows and {} columns in the dataset.'.format(df.shape[0], df.shape[1
```

```
There are 79853 rows and 18 columns in the dataset.
```

In [377…

```
#View data statistics
df.describe().T
```

Out[377…

	count	mean	std	min	25%	50'
id	79853.0	39927.000000	23051.719860	1.0	19964.000	39927.0C
perc_premium_paid_by_cash_credit	79853.0	0.314288	0.334915	0.0	0.034	0.16
age_in_days	79853.0	18846.696906	5208.719136	7670.0	14974.000	18625.0C
age_in_years (Added)	79853.0	50.832793	14.060513	20.0	40.000	50.0C
Income	79853.0	208847.171177	496582.597257	24030.0	108010.000	166560.0C
Count_3-6_months_late	79853.0	0.248369	0.691102	0.0	0.000	0.0C
Count_6-12_months_late	79853.0	0.078093	0.436251	0.0	0.000	0.0C
Count_more_than_12_months_late	79853.0	0.059935	0.311840	0.0	0.000	0.0C
Marital Status	79853.0	0.498679	0.500001	0.0	0.000	0.0C
Veh_Owned	79853.0	1.998009	0.817248	1.0	1.000	2.0C
No_of_dep	79853.0	2.503012	1.115901	1.0	2.000	3.0C
Accomodation	79853.0	0.501296	0.500001	0.0	0.000	1.0C
risk_score	79853.0	99.067243	0.725892	91.9	98.830	99.1E
no_of_premiums_paid	79853.0	10.863887	5.170687	2.0	7.000	10.0C
premium	79853.0	10924.507533	9401.676542	1200.0	5400.000	7500.0C
renewal	79853.0	0.937410	0.242226	0.0	1.000	1.0C

In [378…

```
df.sourcing_channel.value_counts()
```

Out[378…

```
A    43134
B    16512
C    12039
D     7559
E      609
Name: sourcing_channel, dtype: int64
```

In [379…

```
df.residence_area_type.value_counts()
```

Out[379…

```
Urban    48183
Rural    31670
Name: residence_area_type, dtype: int64
```

In [380…

```python
#See which customers have paid 0% of their premiums
df.loc[df['perc_premium_paid_by_cash_credit']==0]
```

Out[380…

	id	perc_premium_paid_by_cash_credit	age_in_days	age_in_years (Added)	Income	Count_3-6_months_late	1
1	27647	0.0	24828	67	53821900	0	
15	9290	0.0	21915	59	7500140	0	
17	38778	0.0	17167	46	7124830	0	
19	2405	0.0	19357	52	7038040	4	
27	44324	0.0	17899	49	4500070	0	
...	...	...	...	...	...	...	
79811	43490	0.0	9861	26	24080	0	
79819	12327	0.0	26288	71	24050	0	
79828	50642	0.0	8766	23	24050	0	
79831	8894	0.0	22278	60	24040	0	
79834	64790	0.0	11320	30	24040	0	

5723 rows × 18 columns

- 5,723 customers paid 0% of their premiums

In [381…

```python
#See which customers have paid 100% of their premiums

df.loc[df['perc_premium_paid_by_cash_credit']==1]
```

Out[381…

	id	perc_premium_paid_by_cash_credit	age_in_days	age_in_years (Added)	Income	Count_3-6_months_late	12
30	4455	1.0	19718	53	3990130	0	
75	96	1.0	16439	45	2490050	0	
87	51318	1.0	21909	59	2340030	0	
104	24711	1.0	15706	42	2190130	0	
128	58627	1.0	17898	48	1998070	1	
...	...	...	...	...	...	...	
79835	7459	1.0	10956	29	24040	0	
79839	78691	1.0	9505	26	24040	0	
79847	55899	1.0	12418	33	24030	0	
79849	23749	1.0	9870	27	24030	0	
79851	44224	1.0	8773	24	24030	0	

5004 rows × 18 columns

- 5,004 customers paid 100% of their premiums

In [382...

```
df.describe(include='object')
```

Out[382...

	sourcing_channel	residence_area_type
count	79853	79853
unique	5	2
top	A	Urban
freq	43134	48183

In [383...

```
#Number of unique values within variable
df.nunique()
```

Out[383...

```
id                                  79853
perc_premium_paid_by_cash_credit     1001
age_in_days                           833
age_in_years (Added)                   82
Income                              24165
Count_3-6_months_late                  14
Count_6-12_months_late                 17
Count_more_than_12_months_late         10
Marital Status                          2
Veh_Owned                               3
No_of_dep                               4
Accomodation                            2
risk_score                            673
no_of_premiums_paid                    57
sourcing_channel                        5
residence_area_type                     2
premium                                30
renewal                                 2
dtype: int64
```

In [384...

```
#Check for null values
df.isnull().sum()
```

Out[384...

```
id                                  0
perc_premium_paid_by_cash_credit    0
age_in_days                         0
age_in_years (Added)                0
Income                              0
Count_3-6_months_late               0
Count_6-12_months_late              0
Count_more_than_12_months_late      0
Marital Status                      0
Veh_Owned                           0
No_of_dep                           0
Accomodation                        0
```

```
risk_score                    0
no_of_premiums_paid           0
sourcing_channel              0
residence_area_type           0
premium                       0
renewal                       0
dtype: int64
```

- There are no Null values

In [385…

```
#Check for duplicates
df.duplicated().sum()
```

Out[385…

```
0
```

- There are no duplicated values

Age converted into years

In [386…

```
#Convert "age_in_days" to float, convert into years, and round
df.age_in_days = df.age_in_days.astype('float')

df['age_in_days'] = df['age_in_days']/365
df['age_in_days'] = df['age_in_days'].round()
```

Variable names simplified

In [387…

```
# Change names of variables to simplify
df.rename(columns={'perc_premium_paid_by_cash_credit': 'Perc_premium_paid', 'age_in_day
                   'Marital Status' : 'Marital_status', 'Count_3-6_months_late': '3-6_l
                   'Count_more_than_12_months_late': '12_more_late', 'risk_score': 'Ris
                   'sourcing_channel': 'Source', 'residence_area_type': 'Res_type', 'pr
```

In [388…

```
#View counts of premium renewals and non-renewals
df.Renewal.value_counts()
```

Out[388…

```
1    74855
0     4998
Name: Renewal, dtype: int64
```

In [389…

```
#Find percentage of renewals
df.Renewal.sum()/len(df.Renewal)
```

Out[389…

```
0.937409990858202
```

In [390…

```
#Find rows where age is equal to or greater than 100
df.loc[df['Age']>=100]
```

Out[390…

id	Perc_premium_paid	Age	age_in_years (Added)	Income	3-6_late	6-12_late	12_more_late	Marital_s

	id	Perc_premium_paid	Age	age_in_years (Added)	Income	3-6_late	6-12_late	12_more_late	Marital_s
61785	32428	0.110	102.0	101	102580	0	0	0	
63162	74848	0.010	102.0	101	99060	0	0	0	
67316	49071	0.003	101.0	100	86570	2	0	0	
76434	46150	0.026	101.0	100	50050	0	0	0	
76654	1923	1.000	103.0	102	48130	0	0	0	

EDA / Univariate

In [391…

```
# While doing uni-variate analysis of numerical variables we want to study their centra
# Let us write a function that will help us create boxplot and histogram for any input
# variable.
# This function takes the numerical column as the input and returns the boxplots
# and histograms for the variable.
# Let us see if this help us write faster and cleaner code.
def histogram_boxplot(feature, figsize=(12,7), bins = None):
    """ Boxplot and histogram combined
    feature: 1-d feature array
    figsize: size of fig (default (9,8))
    bins: number of bins (default None / auto)
    """
    sns.set(font_scale=2) # setting the font scale  of the seaborn
    f2, (ax_box2, ax_hist2) = plt.subplots(nrows = 2, # Number of rows of the subplot g
                                           sharex = True, # x-axis will be shared among
                                           gridspec_kw = {"height_ratios": (.25, .75)},
                                           figsize = figsize
                                           ) # creating the 2 subplots
    sns.boxplot(feature, ax=ax_box2, showmeans=True, color='red') # boxplot will be cre
    sns.distplot(feature, kde=False, ax=ax_hist2, bins=bins) if bins else sns.distplot(
    ax_hist2.axvline(np.mean(feature), color='g', linestyle='--') # Add mean to the his
    ax_hist2.axvline(np.median(feature), color='black', linestyle='-') # Add median to
```

In [392…

```
# Function to create barplots that indicate percentage for each category.

def perc_on_bar(plot, feature):
    '''
    plot
    feature: categorical feature
    the function won't work if a column is passed in hue parameter
    '''
    total = len(feature) # length of the column
    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height()/total) # percentage of each
        x = p.get_x() + p.get_width() / 2 - 0.15 # width of the plot
        y = p.get_y() + p.get_height()           # hieght of the plot
        ax.annotate(percentage, (x, y), size = 12, ha='center') # annotate the percanta
    plt.show() # show the plot
```

Percentage of premium paid by cash

In [393…

```
histogram_boxplot(df['Perc_premium_paid'])
```

- The data is right skewed
- This shows that a good percentage of people paid close to 0% of the premium was paid by cash
- Who are those that are paying such a low amount? What are the conditions?
- The median amount is just under 20%, with a mean just over 30%
- About 6000 customers paid close to 100% of their premiums

In [394…

```
#Percentage of premium paid counted
df['Perc_premium_paid'].value_counts(normalize=True)
```

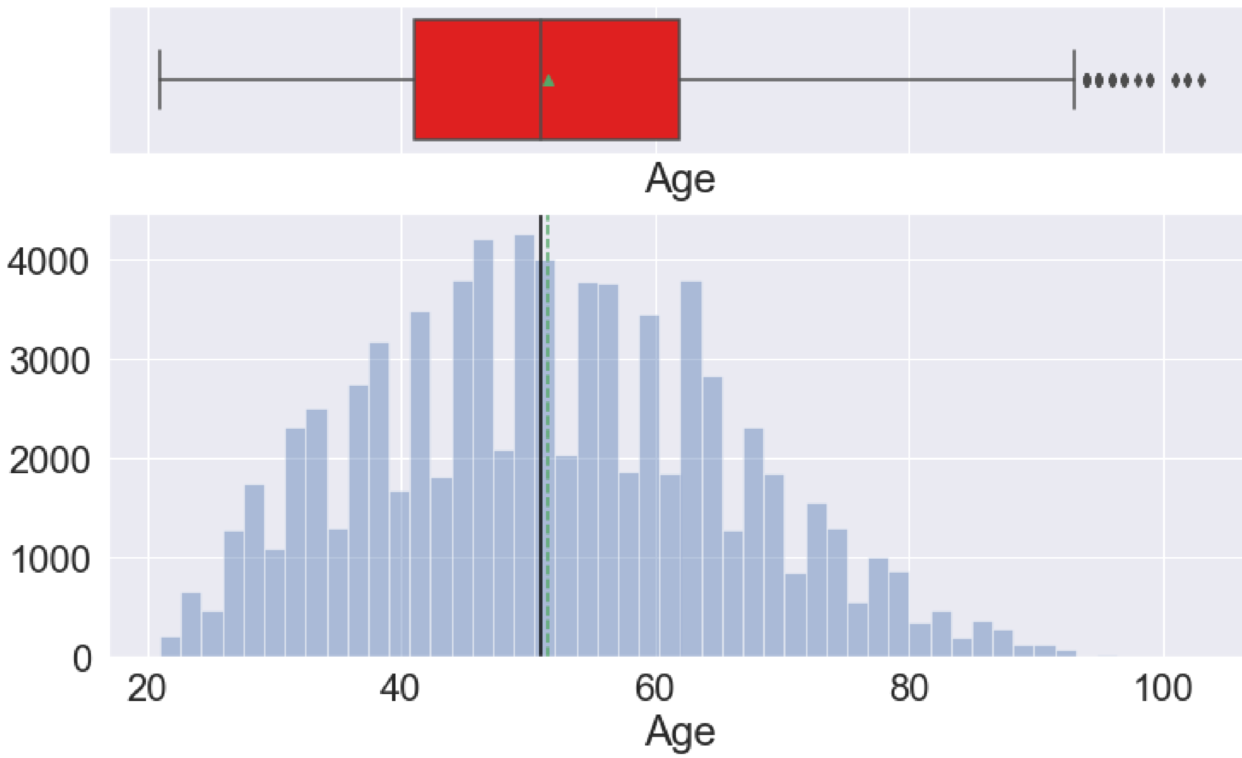
Out[394…

```
0.000    0.071669
1.000    0.062665
0.001    0.008741
0.002    0.007664
0.003    0.006900
           ...
0.724    0.000238
0.869    0.000213
0.759    0.000213
0.851    0.000200
0.742    0.000188
Name: Perc_premium_paid, Length: 1001, dtype: float64
```

Age in years (after conversion from days)

In [395…

```
histogram_boxplot(df['Age'])
```

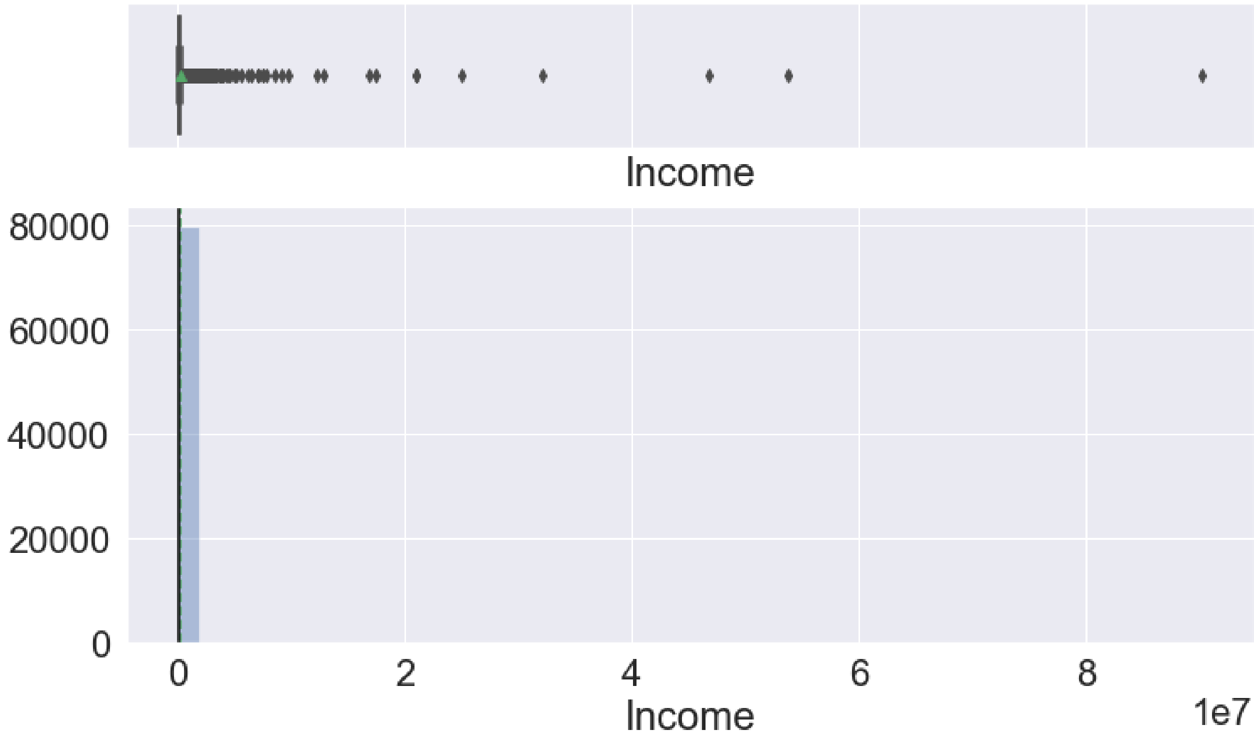


- Normal distribution
- Median and mean age is around 51 years

Income

In [396…

```
histogram_boxplot(df['Income'])
```

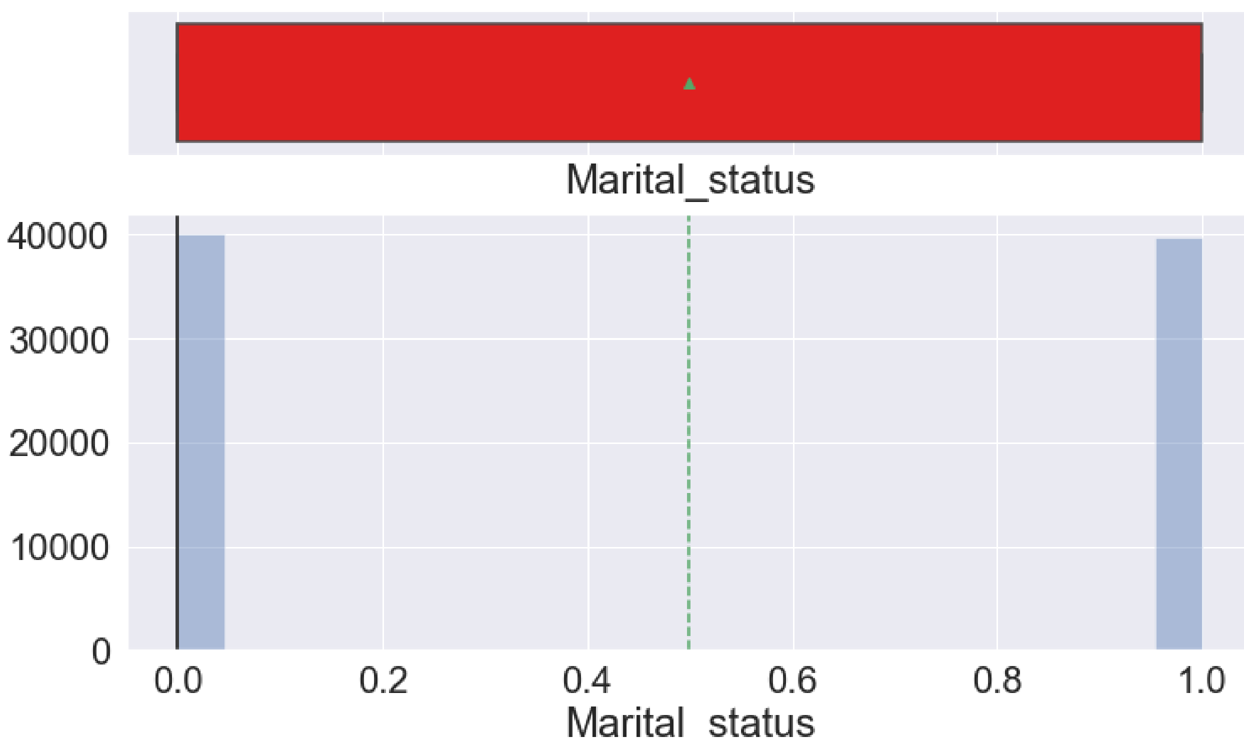


- There are many outliers to the right of the box (which is not visible) that then spread out to fewer customers with very large incomes
- The mean/median are not visible

Marital status (0: Unmarried, 1: Married)

In [397…

```
histogram_boxplot(df['Marital_status'])
```

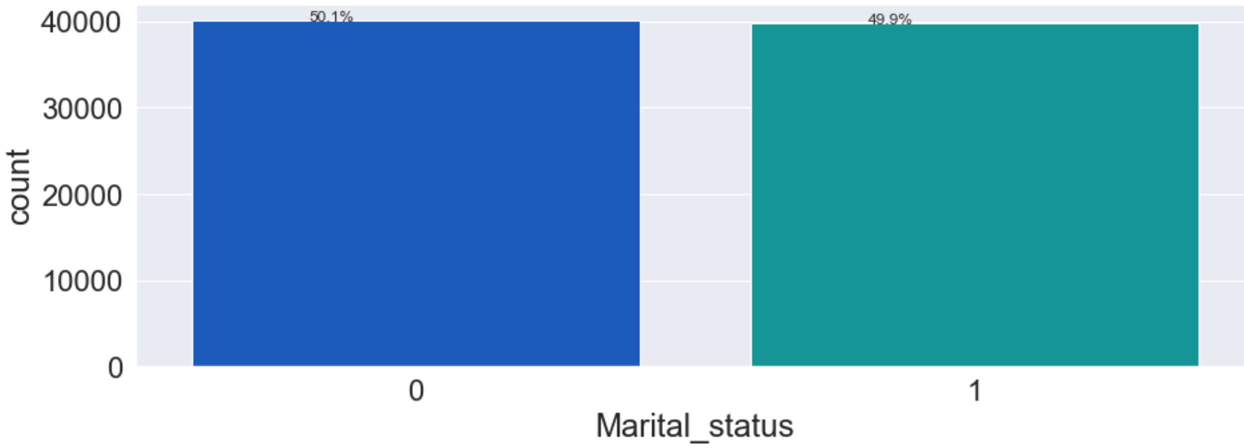


- There is an even distribution of married and unmarried customers in the dataset

Marital Status

In [398…

```
plt.figure(figsize=(15,5))
ax = sns.countplot(df["Marital_status"],palette='winter')
perc_on_bar(ax,df["Marital_status"])
```

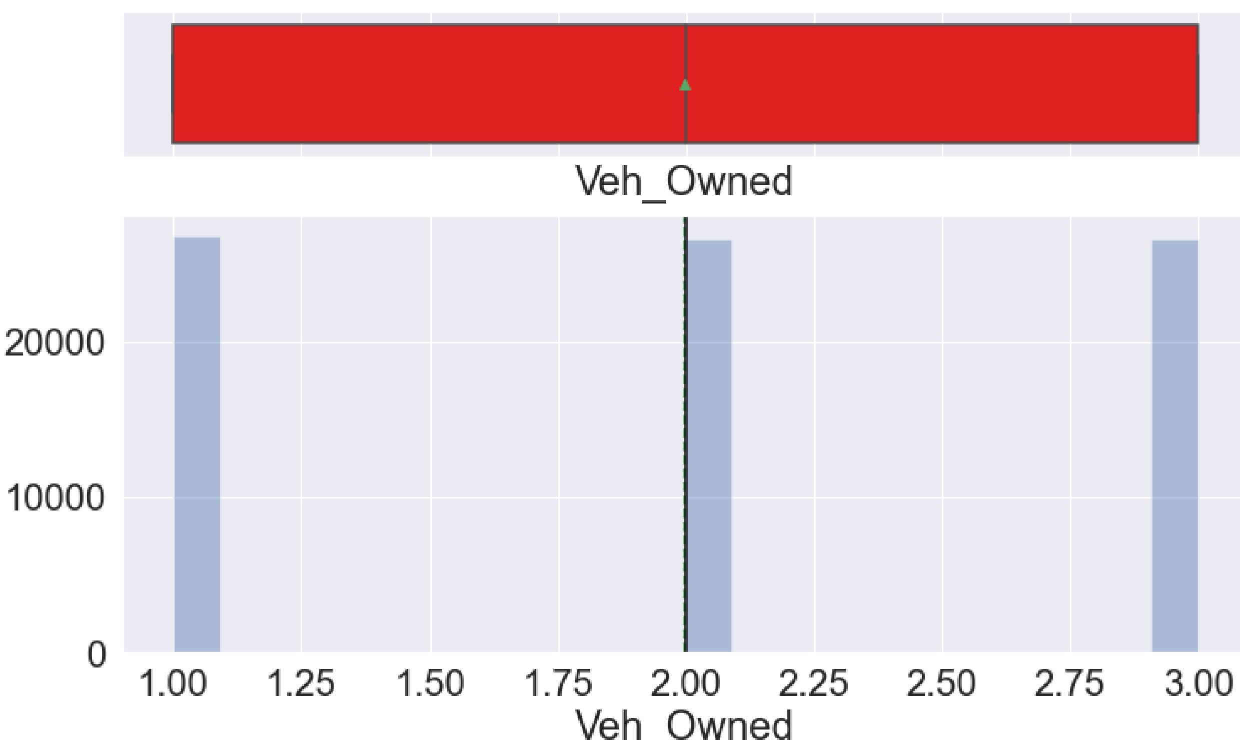


- 50.1% of the customers are married
- 49.9% of the customers are not married

Vehicles owned

In [399…

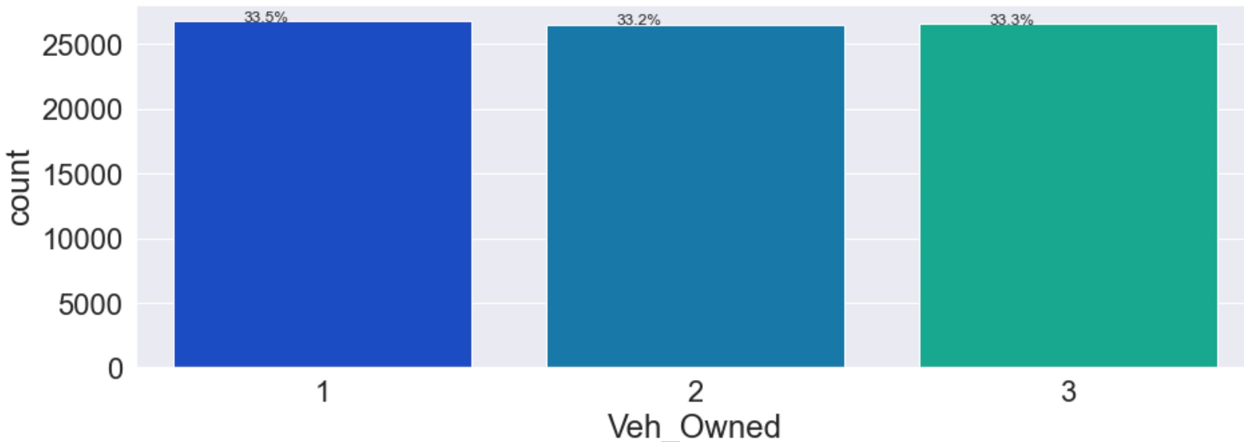
```
histogram_boxplot(df['Veh_Owned'])
```



- There is an even distribution of number of vehicles owned
- Average number of cars owned is 2

In [400…

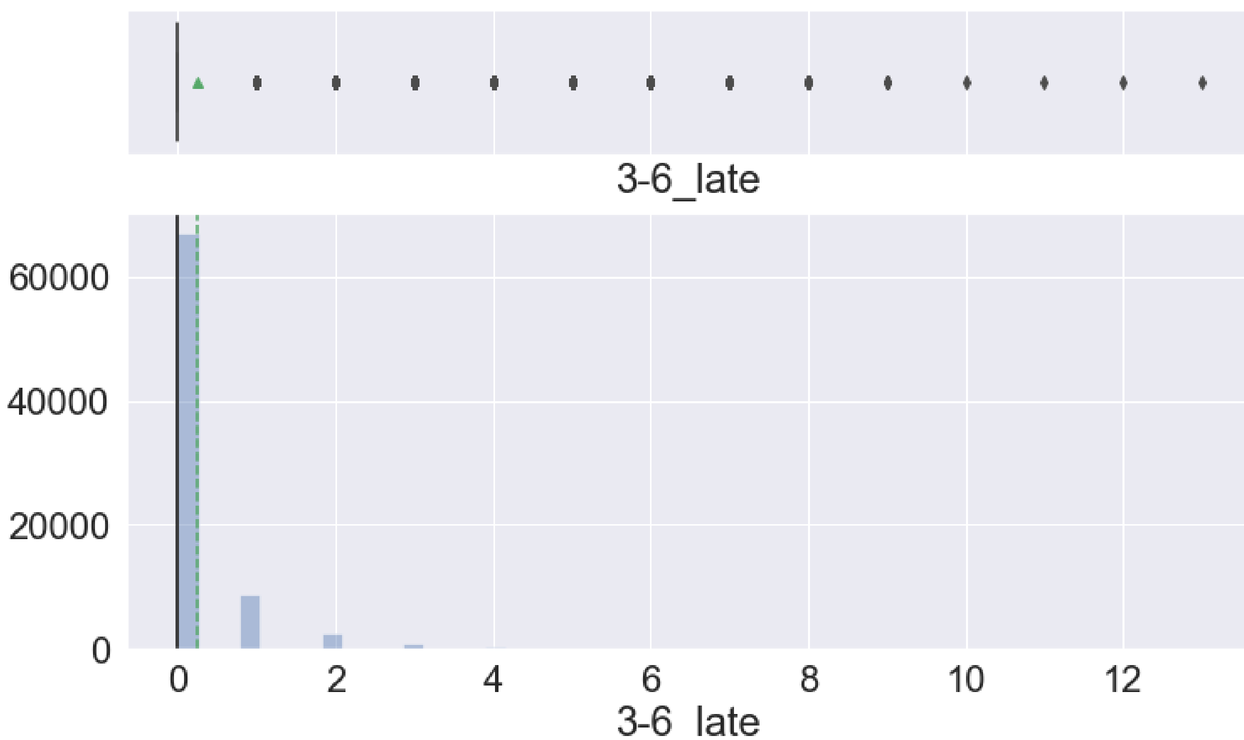
```
plt.figure(figsize=(15,5))
ax = sns.countplot(df["Veh_Owned"],palette='winter')
perc_on_bar(ax,df["Veh_Owned"])
```



Times premium was paid 3-6 months late

In [401…

```
histogram_boxplot(df['3-6_late'])
```



- 84% of the customers were not late more than 2 months
- 11% of the customers were late at least once

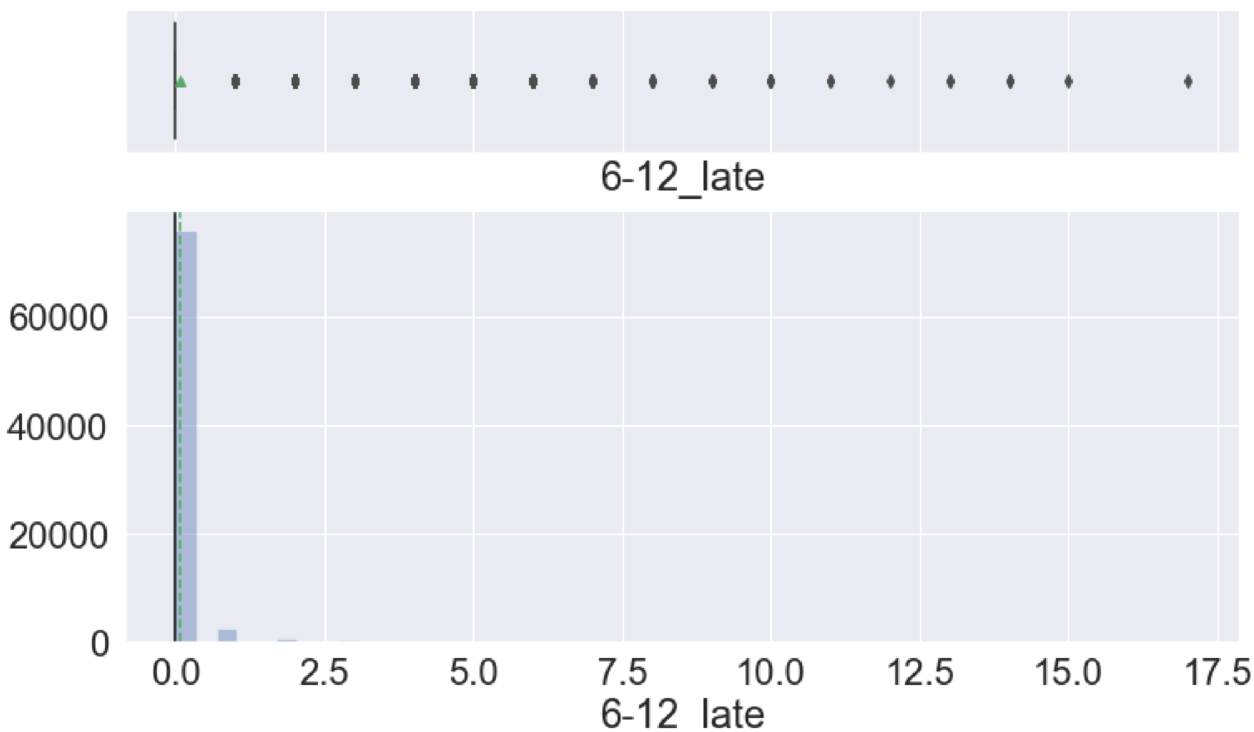
In [402…

```
df['3-6_late'].value_counts(normalize=True)
```

Out[402…
```
0     0.837764
1     0.110528
2     0.031545
3     0.011947
4     0.004684
5     0.002104
6     0.000852
7     0.000288
8     0.000188
9     0.000050
13    0.000013
12    0.000013
11    0.000013
10    0.000013
Name: 3-6_late, dtype: float64
```

Times premium was paid 6-12 months late

In [403…
```python
histogram_boxplot(df['6-12_late'])
```



- 95% of the customers were not late more than 5 months

In [404…
```python
df['6-12_late'].value_counts(normalize=True)
```

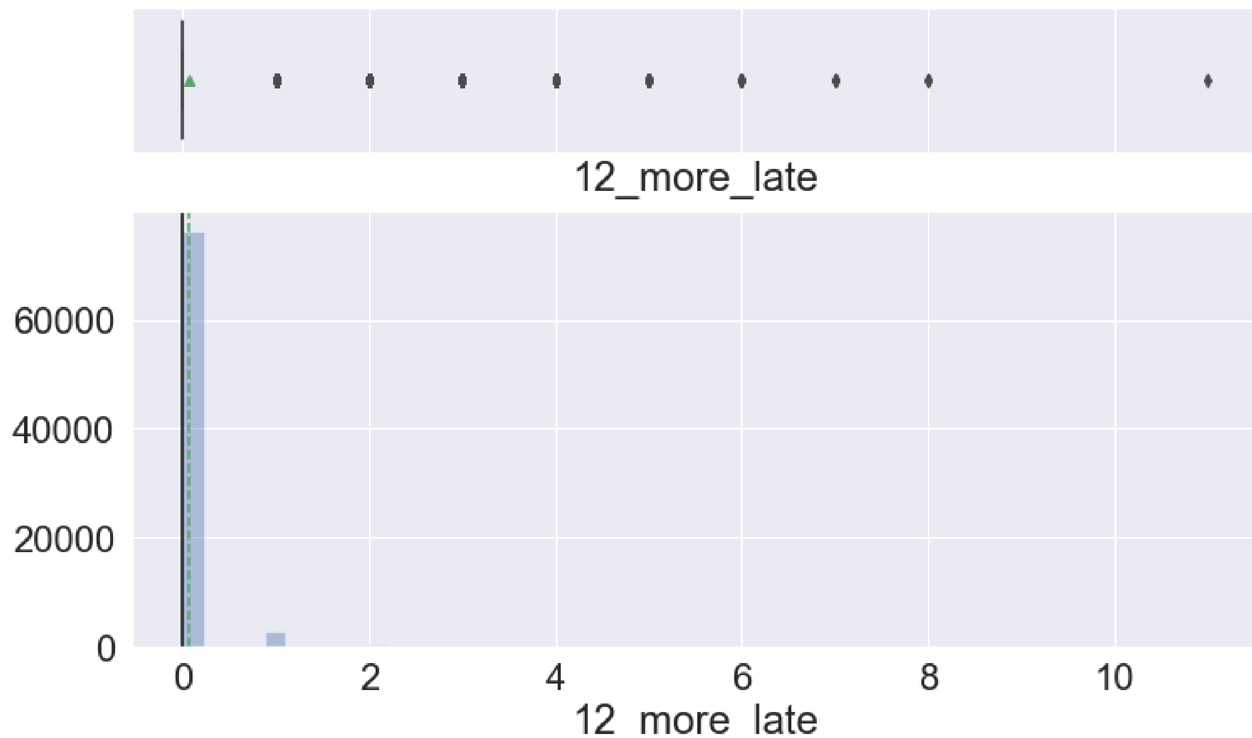
Out[404…
```
0     0.950847
1     0.033562
2     0.008678
3     0.003970
4     0.001628
5     0.000576
6     0.000326
7     0.000138
```

```
8      0.000063
10     0.000050
9      0.000050
14     0.000025
11     0.000025
13     0.000025
12     0.000013
15     0.000013
17     0.000013
Name: 6-12_late, dtype: float64
```

Times premium was paid more than 12 months late

In [405…

```
histogram_boxplot(df['12_more_late'])
```



- 95% of the customers were not late more than 12 months

In [406…

```
df['12_more_late'].value_counts(normalize=True)
```

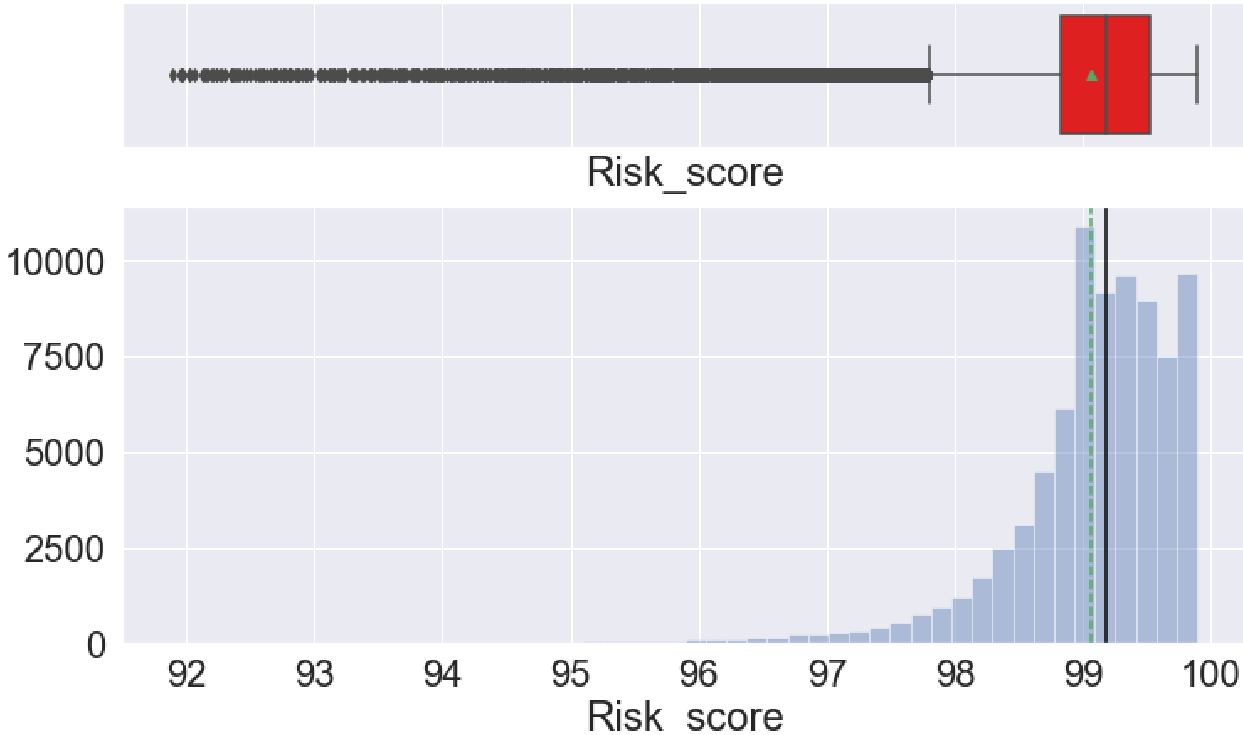
Out[406…

```
0      0.953439
1      0.037519
2      0.006236
3      0.001891
4      0.000601
5      0.000163
6      0.000075
7      0.000038
8      0.000025
11     0.000013
Name: 12_more_late, dtype: float64
```

Risk Score

In [407…

```
histogram_boxplot(df['Risk_score'])
```

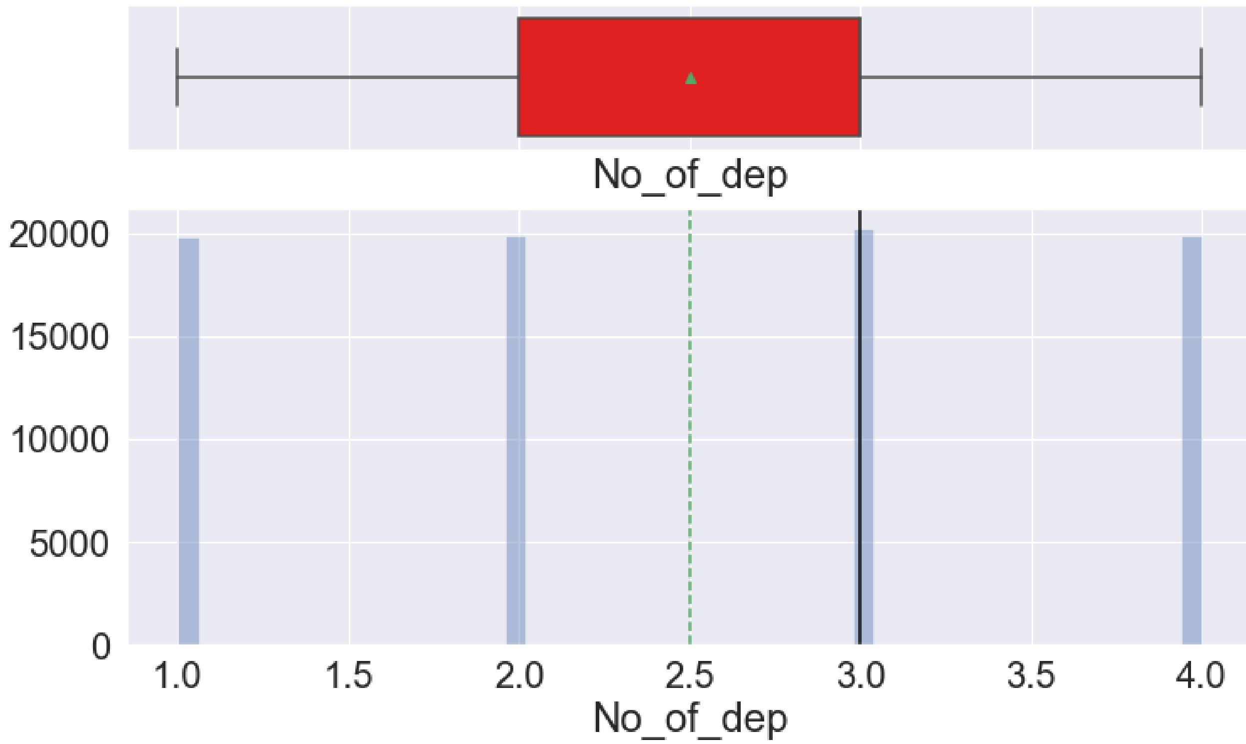


- The risk score has a general range from 91.9 to 99.89
- The average and median scores are just over 99
- There are many outliers to the right of the chart, but they are not able to alter the distribution too much (the mean and median are still relatively close) as the counterbalance is brought about by the number of people with scores above 99.

Number of dependents

In [408…

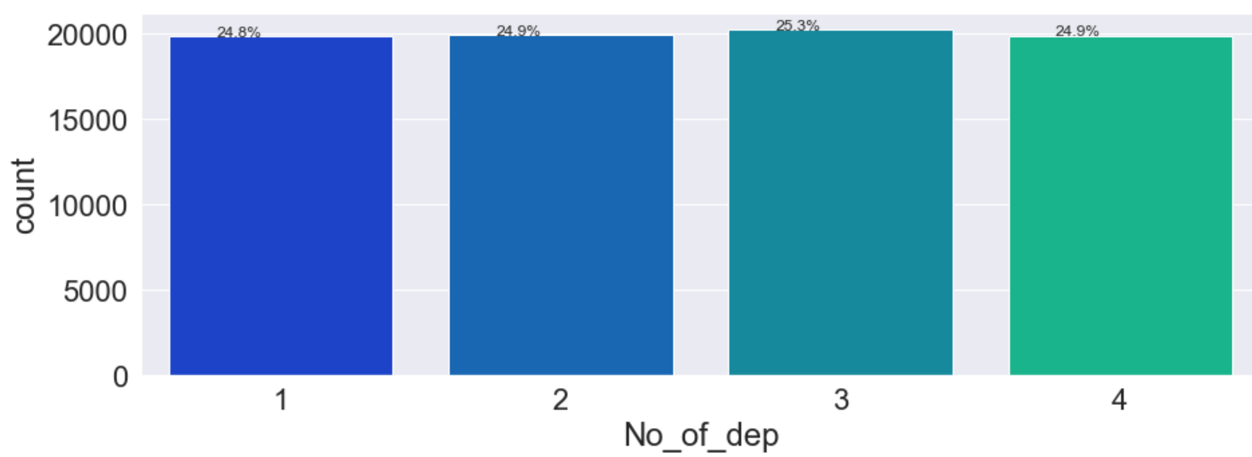
```
histogram_boxplot(df['No_of_dep'])
```



- This somewhat of a uniform distribution
- The average number of dependents is 2.5

In [409…

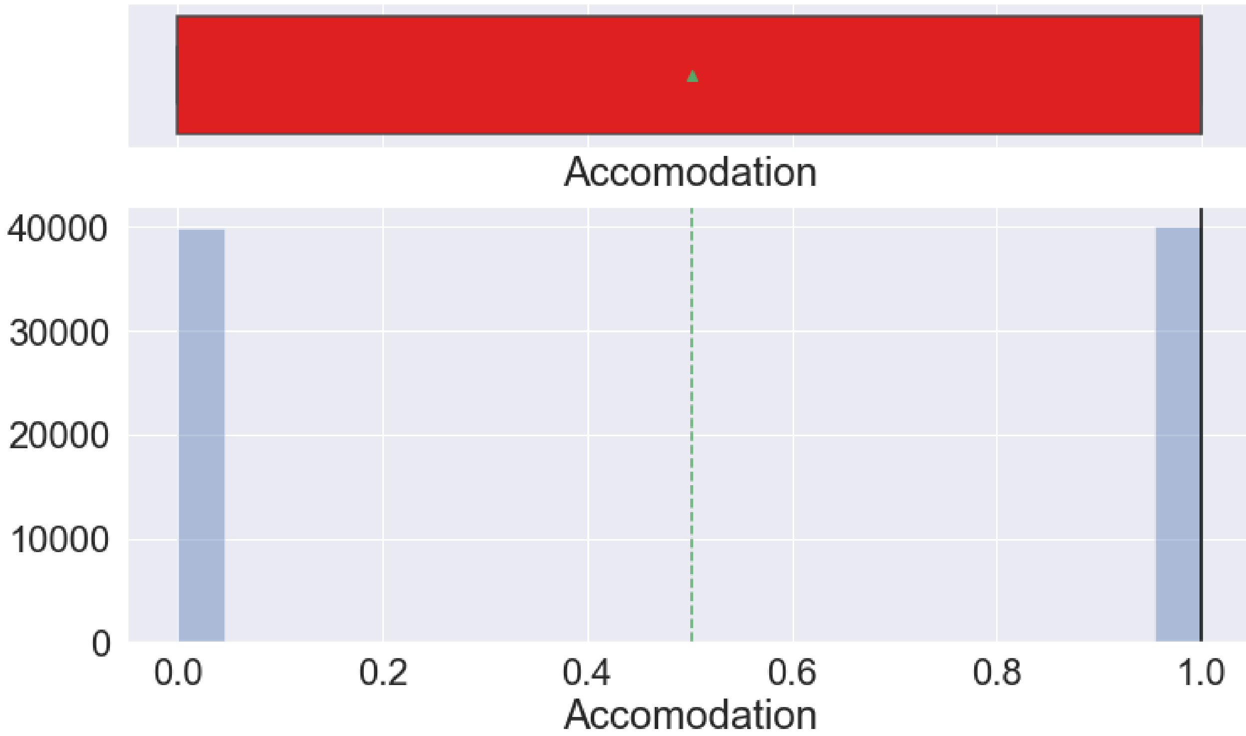
```
plt.figure(figsize=(15,5))
ax = sns.countplot(df["No_of_dep"],palette='winter')
perc_on_bar(ax,df["No_of_dep"])
```



Accomodation (0: Rented, 1: Owned)

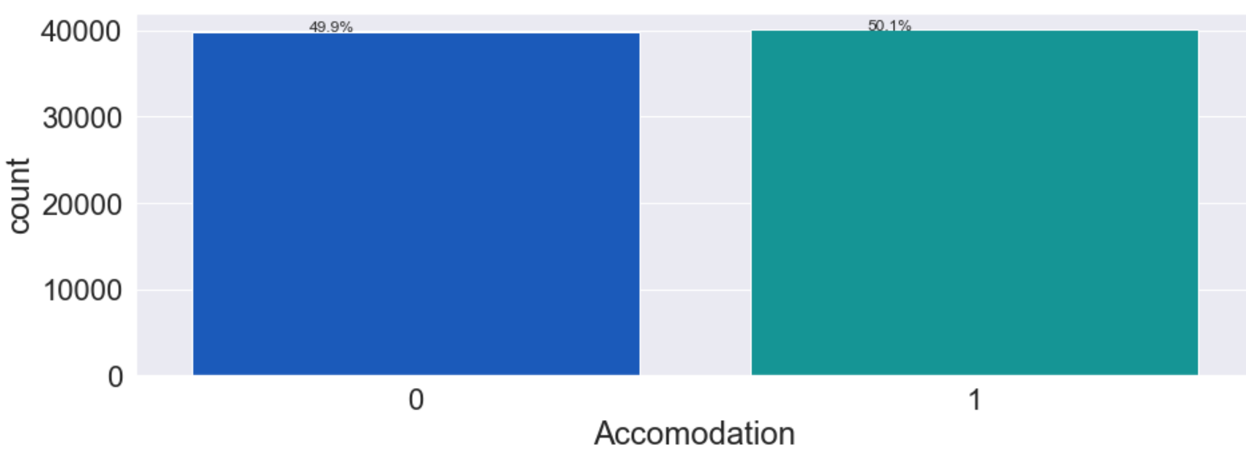
In [410…

```
histogram_boxplot(df['Accomodation'])
```



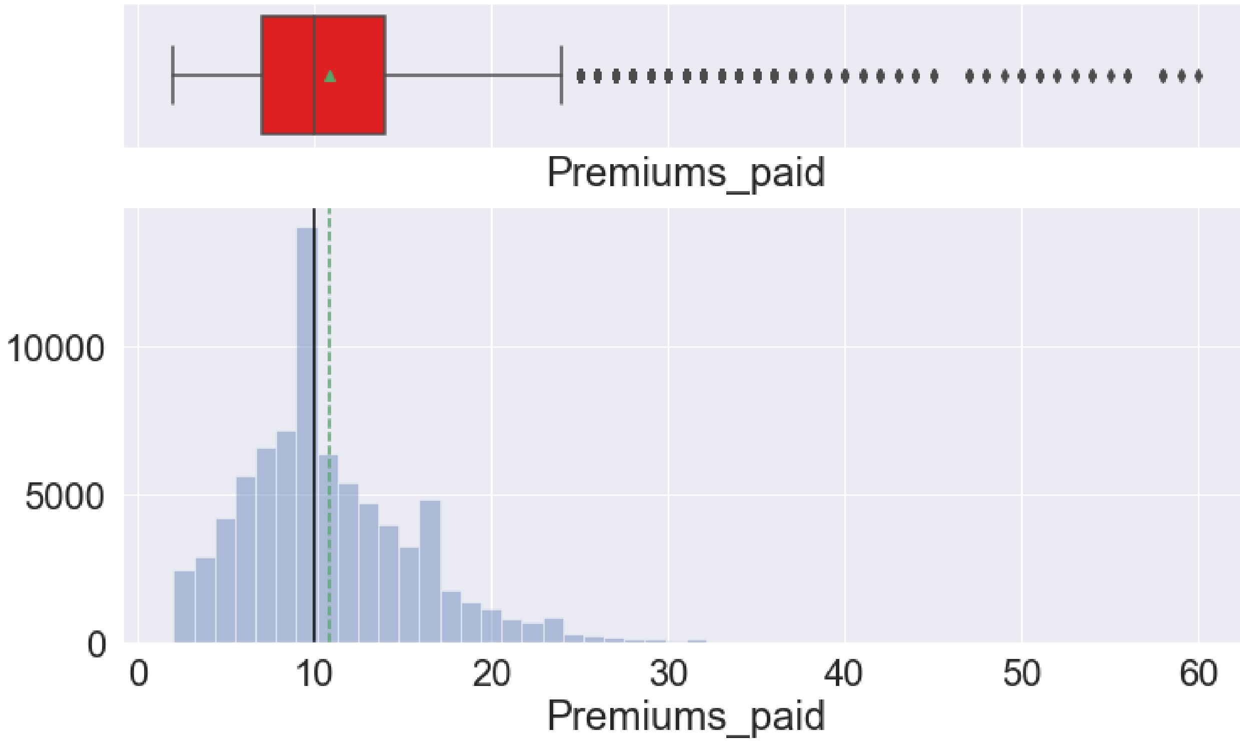
- This is another uniformly distributed chart with almost half of the customers owning and renting

In [411…
```
plt.figure(figsize=(15,5))
ax = sns.countplot(df["Accomodation"],palette='winter')
perc_on_bar(ax,df["Accomodation"])
```



Number of premiums paid

In [412…
```
histogram_boxplot(df['Premiums_paid'])
```

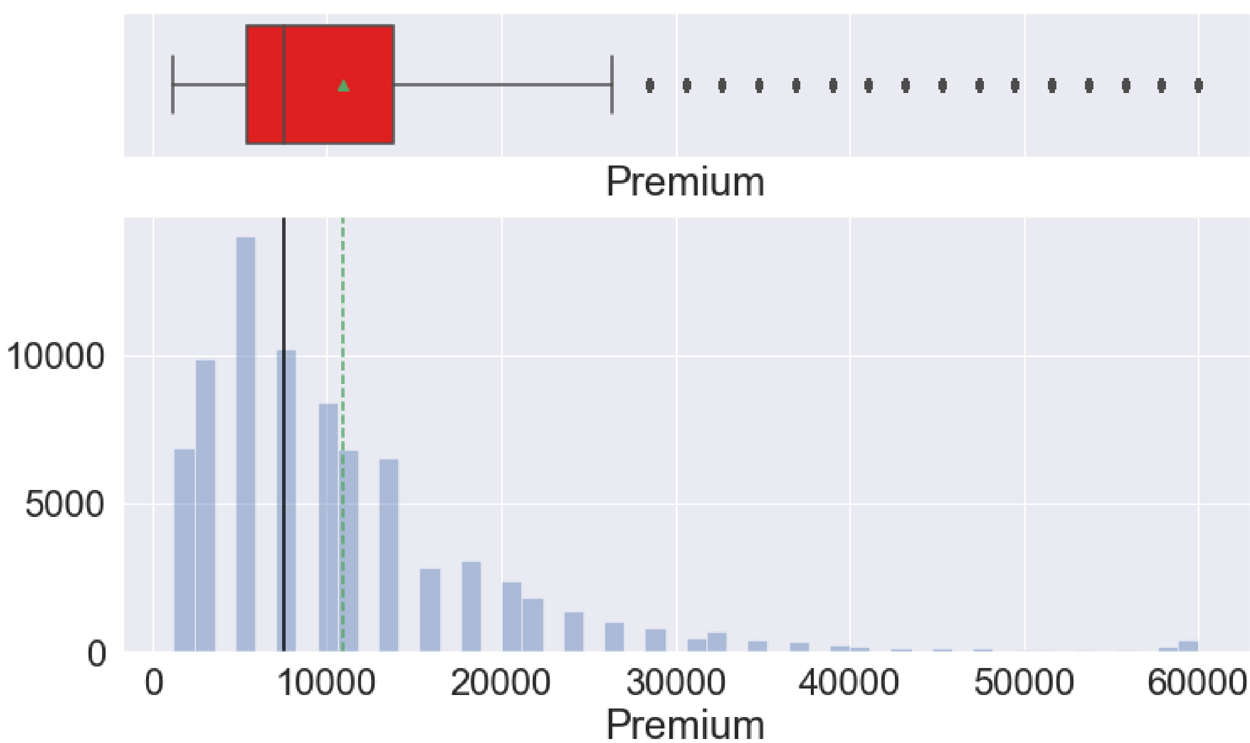


- Almost a normal distribution, if not for the outliers to the right
- The median number of premiums paid to date is 10, with the mean just a little higher at around 11.

Premium amounts

In [413…

```
histogram_boxplot(df['Premium'])
```

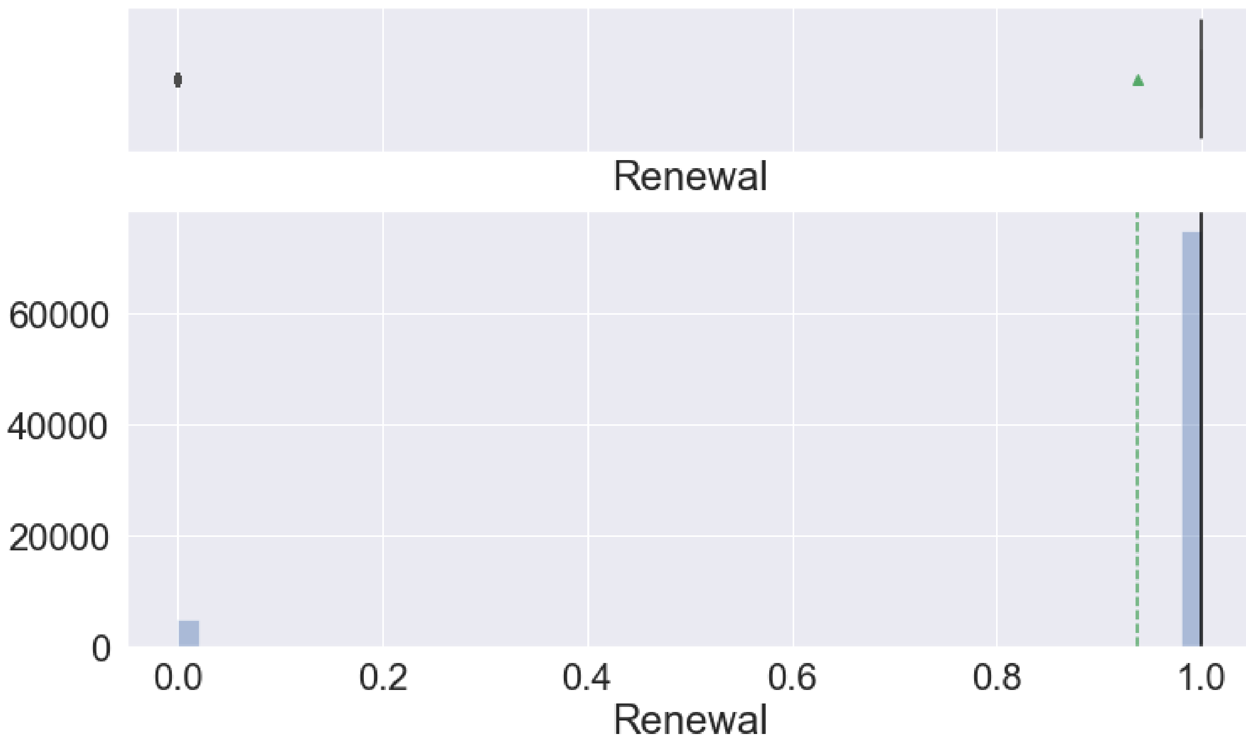


- There is a slight right skew to the chart, brought about by the outliers to the right
- The median amount is around $7,000 and the mean is around \$11,000
- There are premium amounts at $60,000

Renewals (0: Non-renewal, 1: Renewal)

In [414...

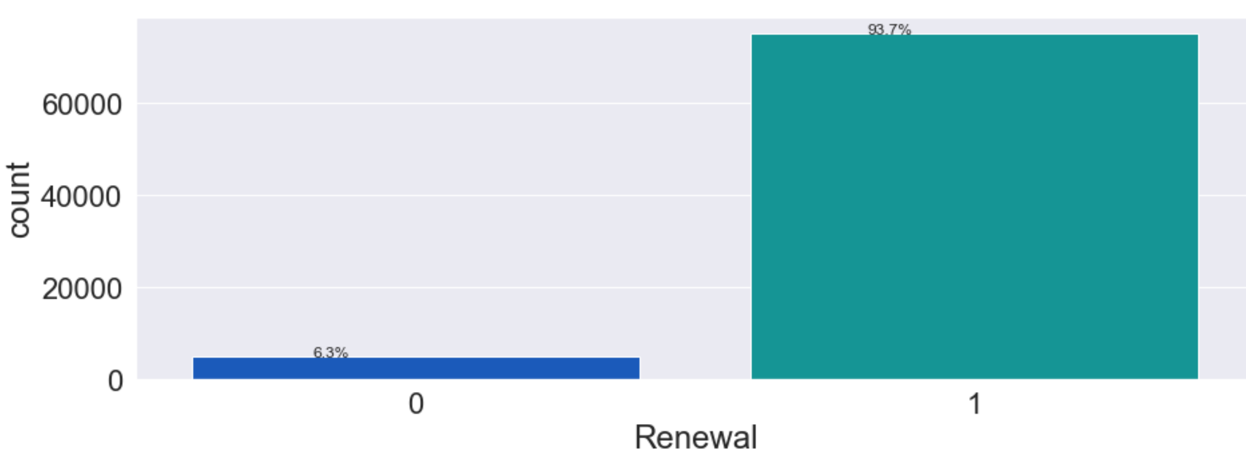
```
histogram_boxplot(df['Renewal'])
```



- 93% of the customers renewed their premium
- 6% of the customers did not renew

In [415...

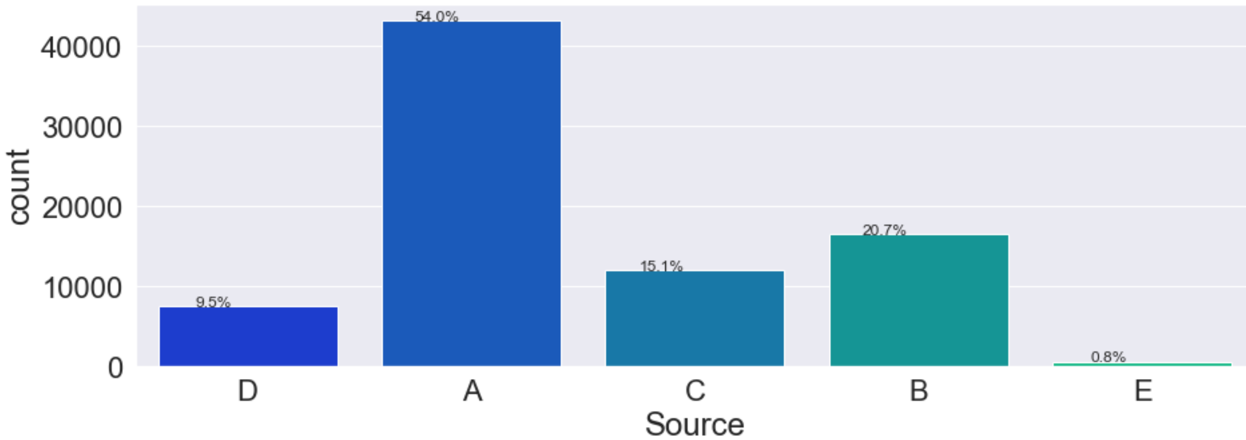
```
plt.figure(figsize=(15,5))
ax = sns.countplot(df["Renewal"],palette='winter')
perc_on_bar(ax,df["Renewal"]);
```



Sourcing channel

In [416…

```
plt.figure(figsize=(15,5))
ax = sns.countplot(df["Source"],palette='winter')
perc_on_bar(ax,df["Source"])
```

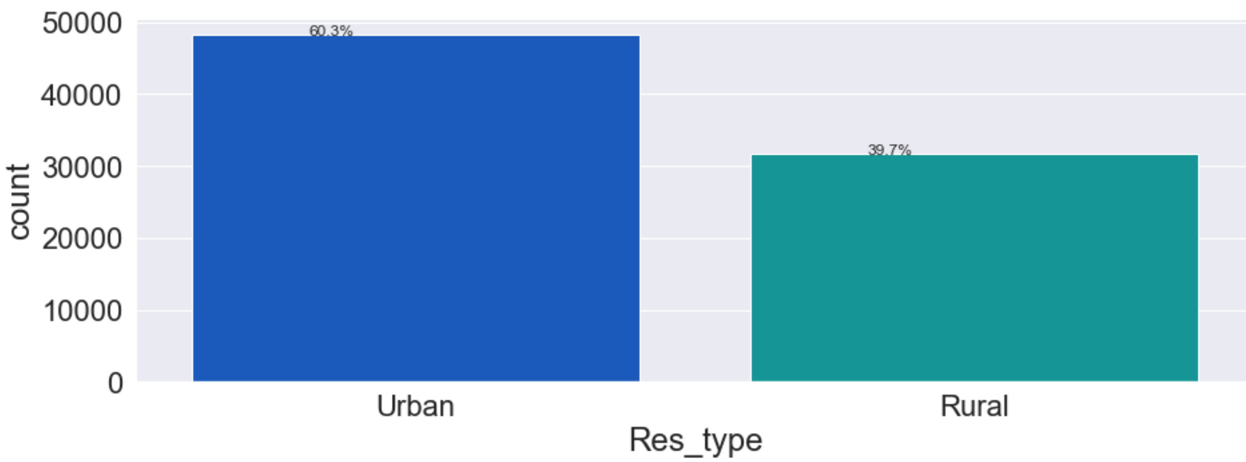


- 54% of the customers were sourced through channel A
- 20.7% were sourced through channel B
- 0.8% were sourced through channel E

Residence type

In [417…

```
plt.figure(figsize=(15,5))
ax = sns.countplot(df["Res_type"],palette='winter')
perc_on_bar(ax,df["Res_type"])
```



- 60.3% of the customers live in Urban areas
- 39.7% live in Rural areas

EDA / Bivariate (X vs. Y)

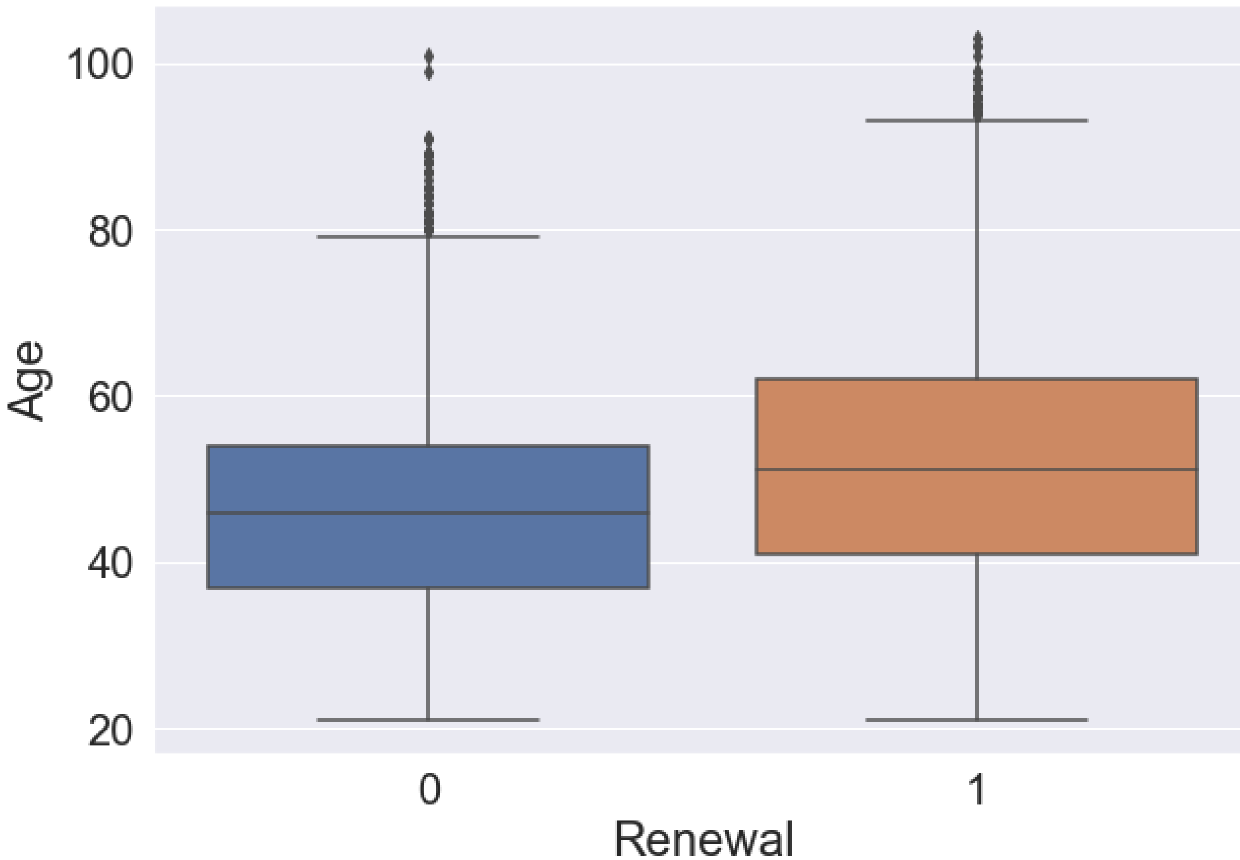
Age vs. Renewal

In [418…

```
plt.figure(figsize=(10,7))
sns.boxplot(df['Renewal'], df['Age'])
```

Out[418…

```
<AxesSubplot:xlabel='Renewal', ylabel='Age'>
```



- The ages of the customers that renewed are just a little older than non-renewers

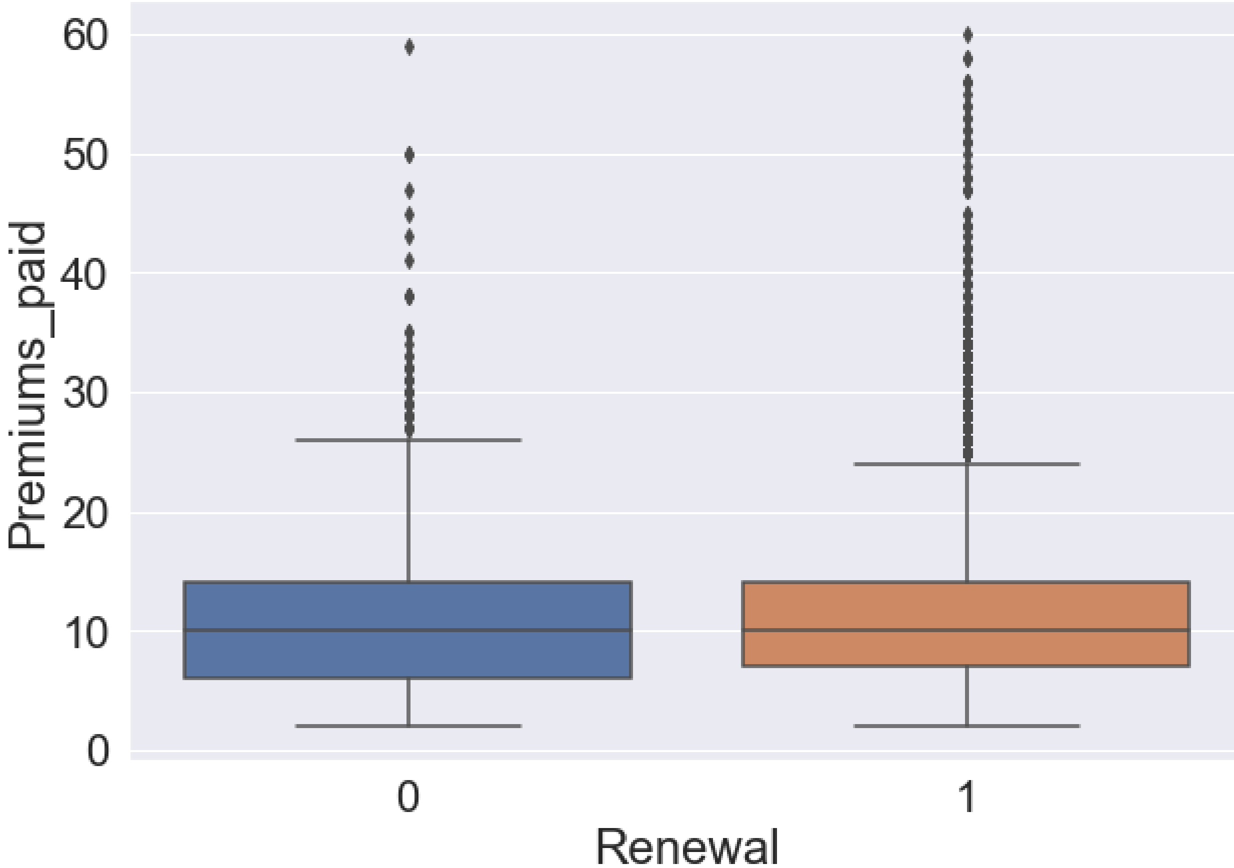
Number of premiums paid vs. Renewal

In [419…

```
plt.figure(figsize=(10,7))
sns.boxplot(df['Renewal'], df['Premiums_paid'])
```

Out[419…

```
<AxesSubplot:xlabel='Renewal', ylabel='Premiums_paid'>
```



- There does not appear to be much of a difference between the customers renewal tendencies based on the number of premiums they paid till date

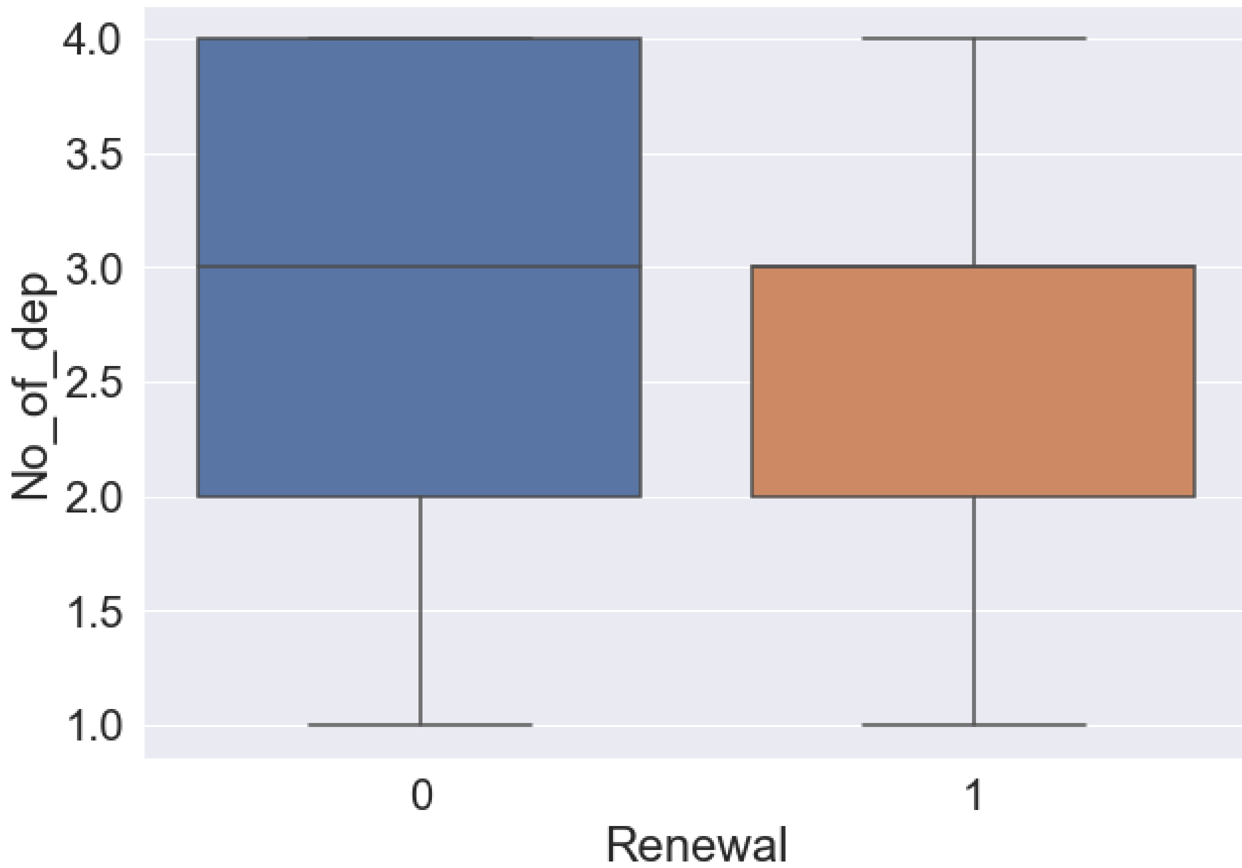
Number of dependents vs. Renewal

In [420…

```
plt.figure(figsize=(10,7))
sns.boxplot(df['Renewal'], df['No_of_dep'])
```

Out[420…

```
<AxesSubplot:xlabel='Renewal', ylabel='No_of_dep'>
```



- There are more customers that did not renew that also had more dependents

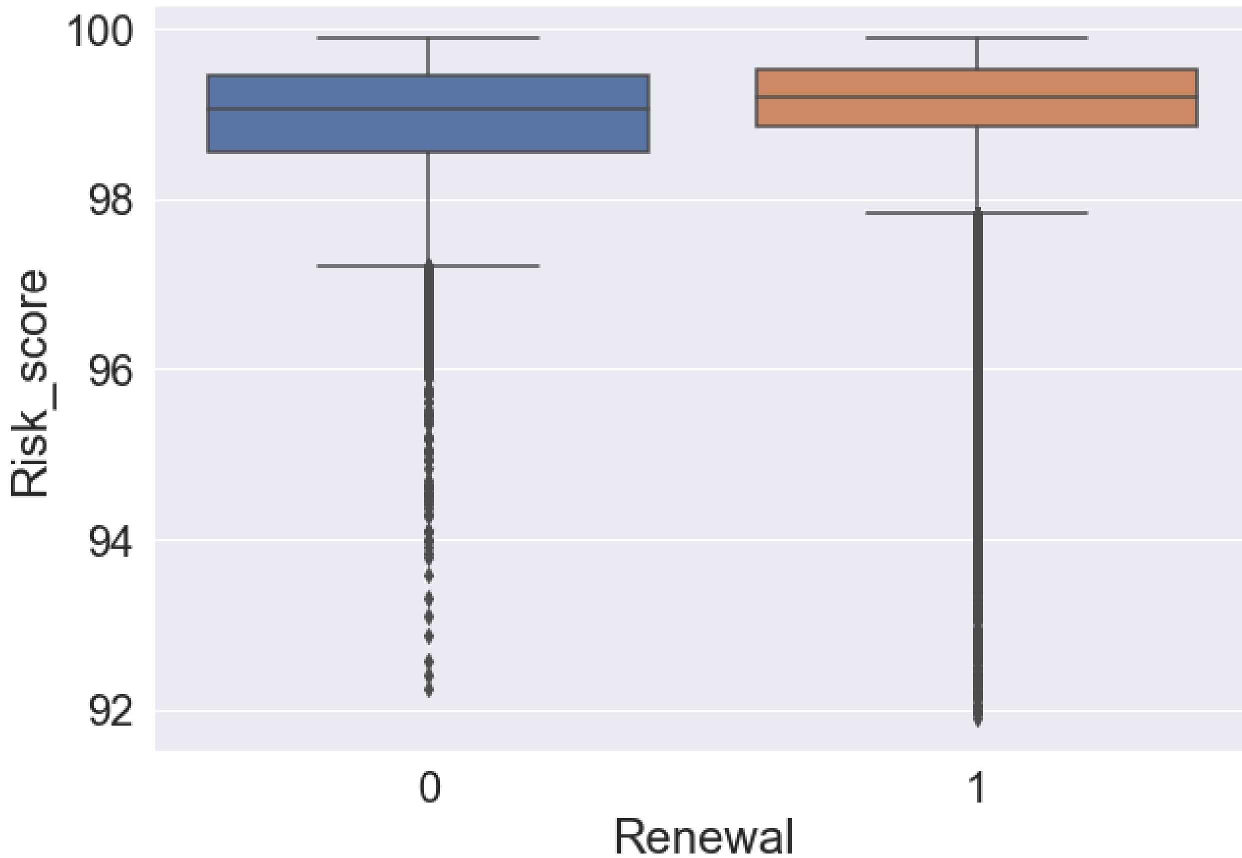
Risk score vs. Renewal

In [421…

```
plt.figure(figsize=(10,7))
sns.boxplot(df['Renewal'], df['Risk_score'])
```

Out[421…

```
<AxesSubplot:xlabel='Renewal', ylabel='Risk_score'>
```



- The risk scores for those that renewed tend to sit a little higher than those that did not renew

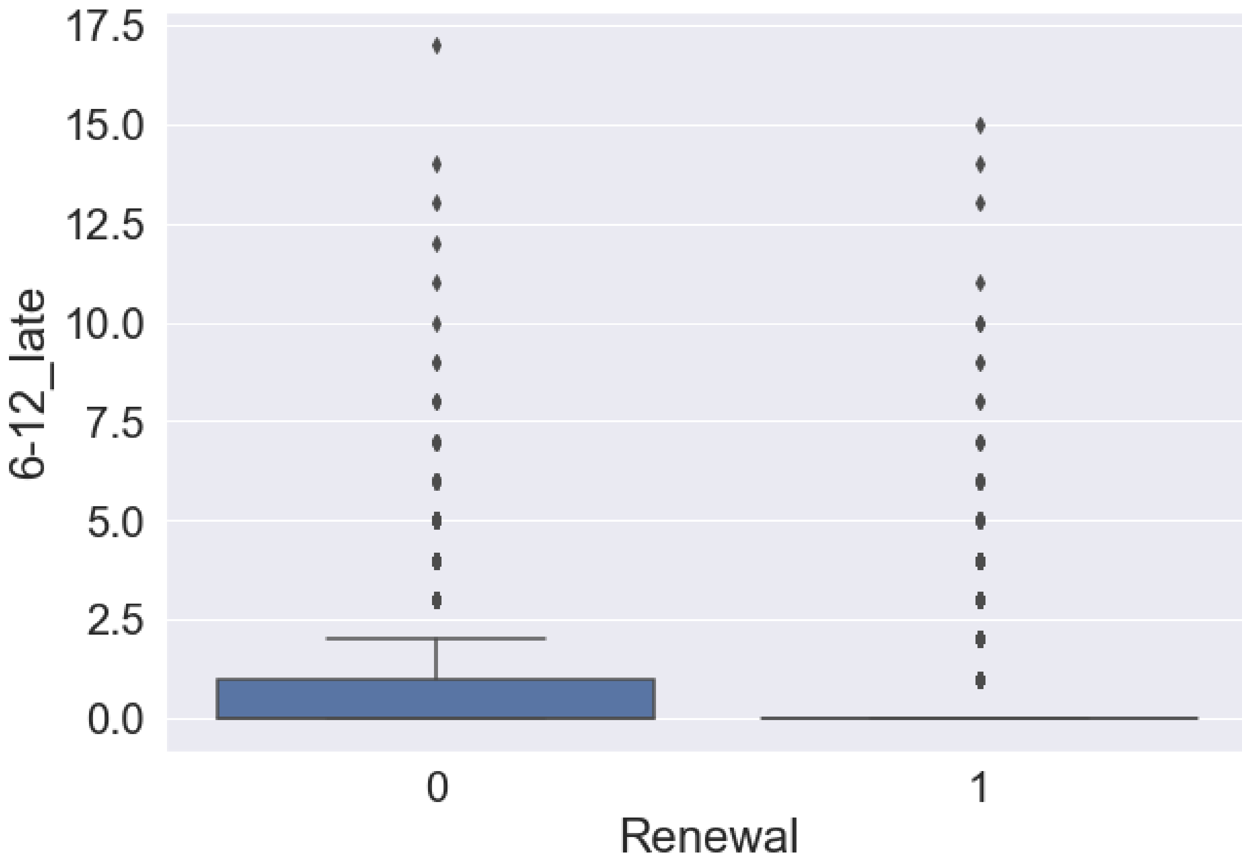
Count 6-12 months late vs. Renewal

In [422…

```
plt.figure(figsize=(10,7))
sns.boxplot(df['Renewal'], df['6-12_late'])
```

Out[422…

```
<AxesSubplot:xlabel='Renewal', ylabel='6-12_late'>
```



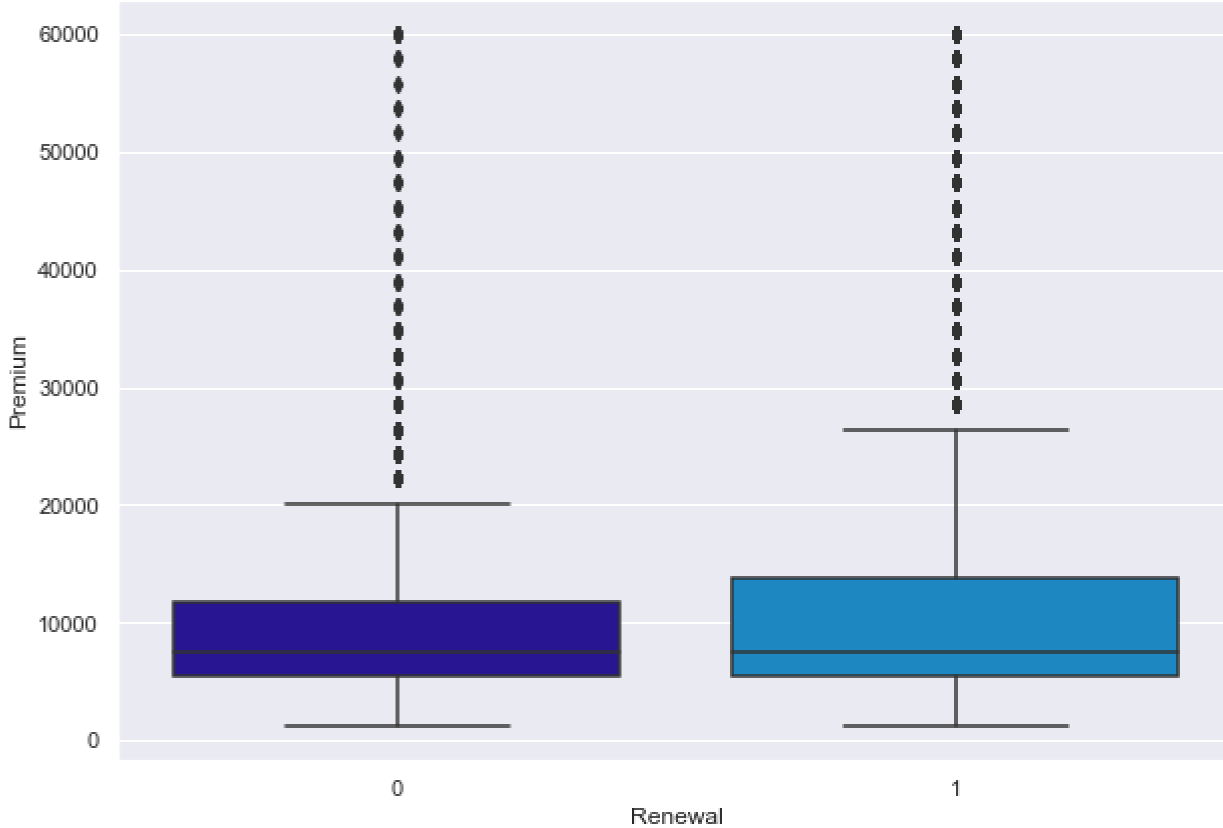
- Those that did renew had more people not pay 6-12 months late

In [511…

```
plt.figure(figsize=(10,7))
sns.boxplot(df['Renewal'], df['Premium'])
```

Out[511…

```
<AxesSubplot:xlabel='Renewal', ylabel='Premium'>
```



- Those with higher premiums were slightly more likely to not default

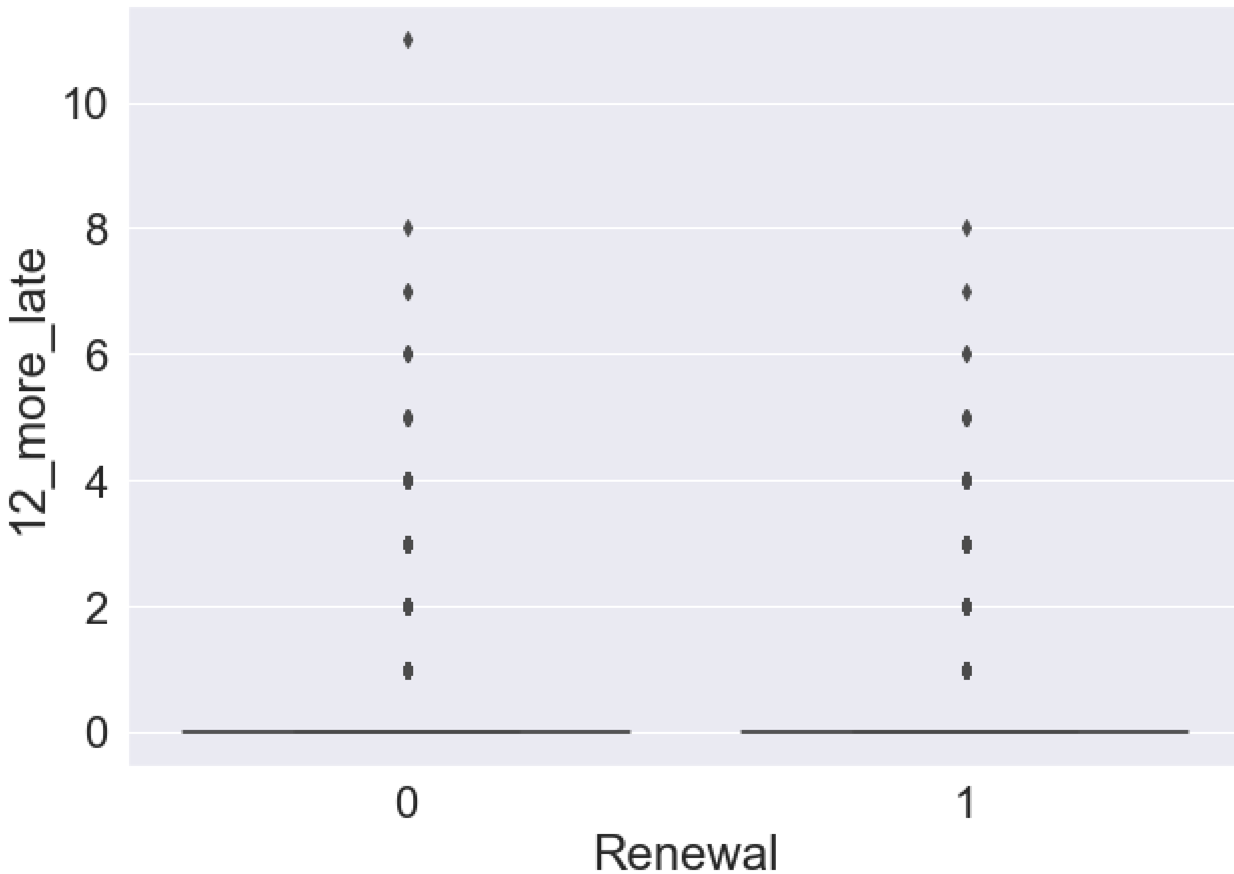
Count more than 12 months late vs. Renewal

In [423…

```
plt.figure(figsize=(10,7))
sns.boxplot(df['Renewal'], df['12_more_late'])
```

Out[423…

```
<AxesSubplot:xlabel='Renewal', ylabel='12_more_late'>
```



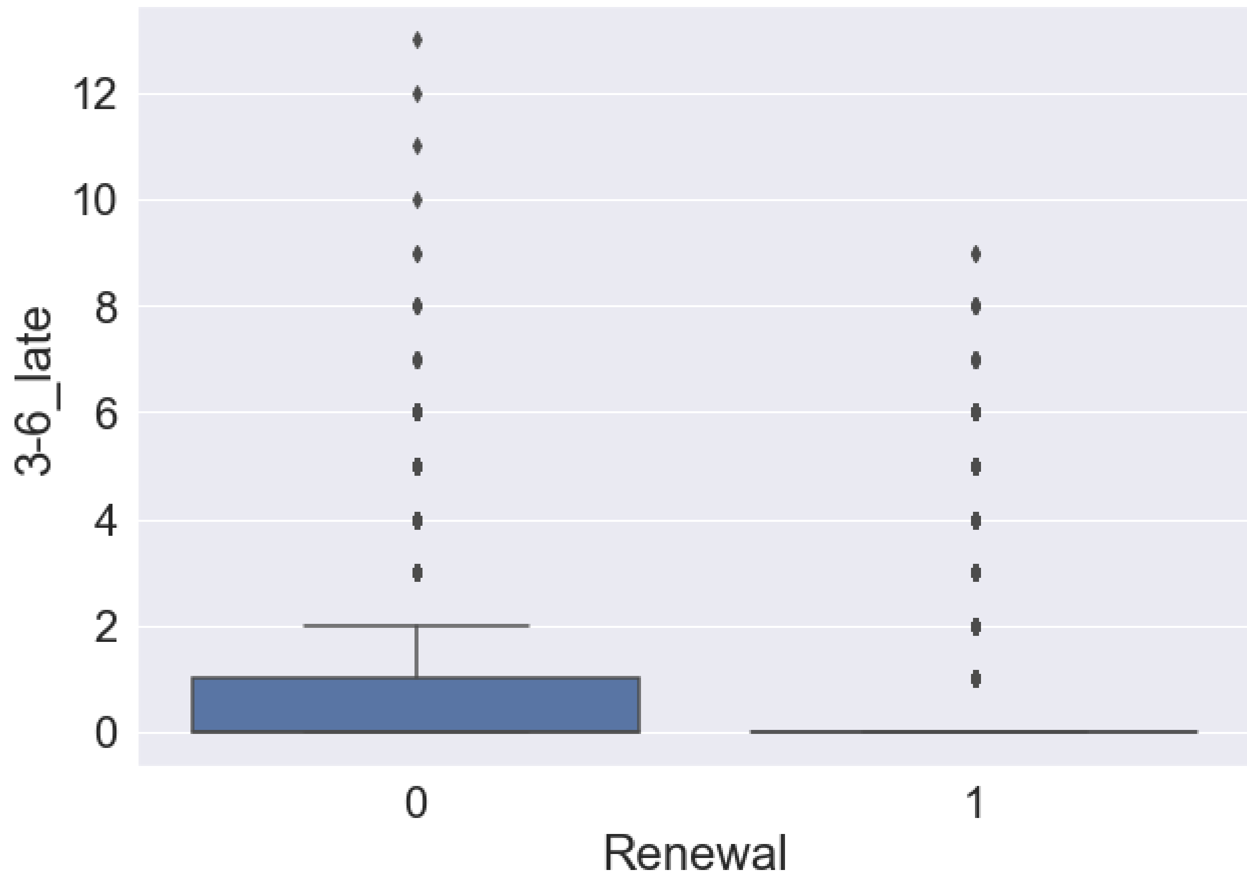
Count 3-6 months late vs. Renewal

In [424…

```
plt.figure(figsize=(10,7))
sns.boxplot(df['Renewal'], df['3-6_late'])
```

Out[424…

```
<AxesSubplot:xlabel='Renewal', ylabel='3-6_late'>
```



- Those that did renew had more people not pay 3-6 months late

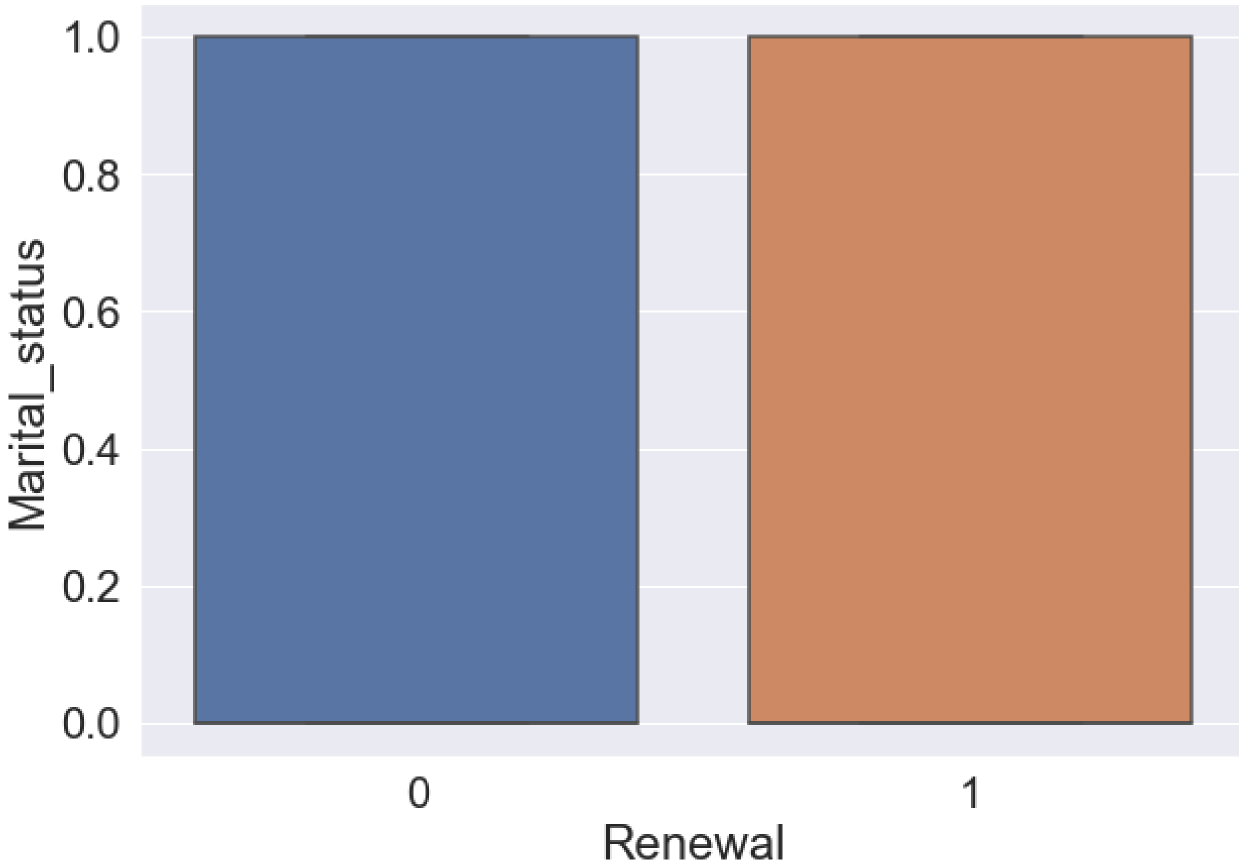
Marital status vs. Renewal

In [425…

```
plt.figure(figsize=(10,7))
sns.boxplot(df['Renewal'], df['Marital_status'])
```

Out[425…

```
<AxesSubplot:xlabel='Renewal', ylabel='Marital_status'>
```



- There are an equal number of married and unmarried customers that renewed and did not renew, in the dataset

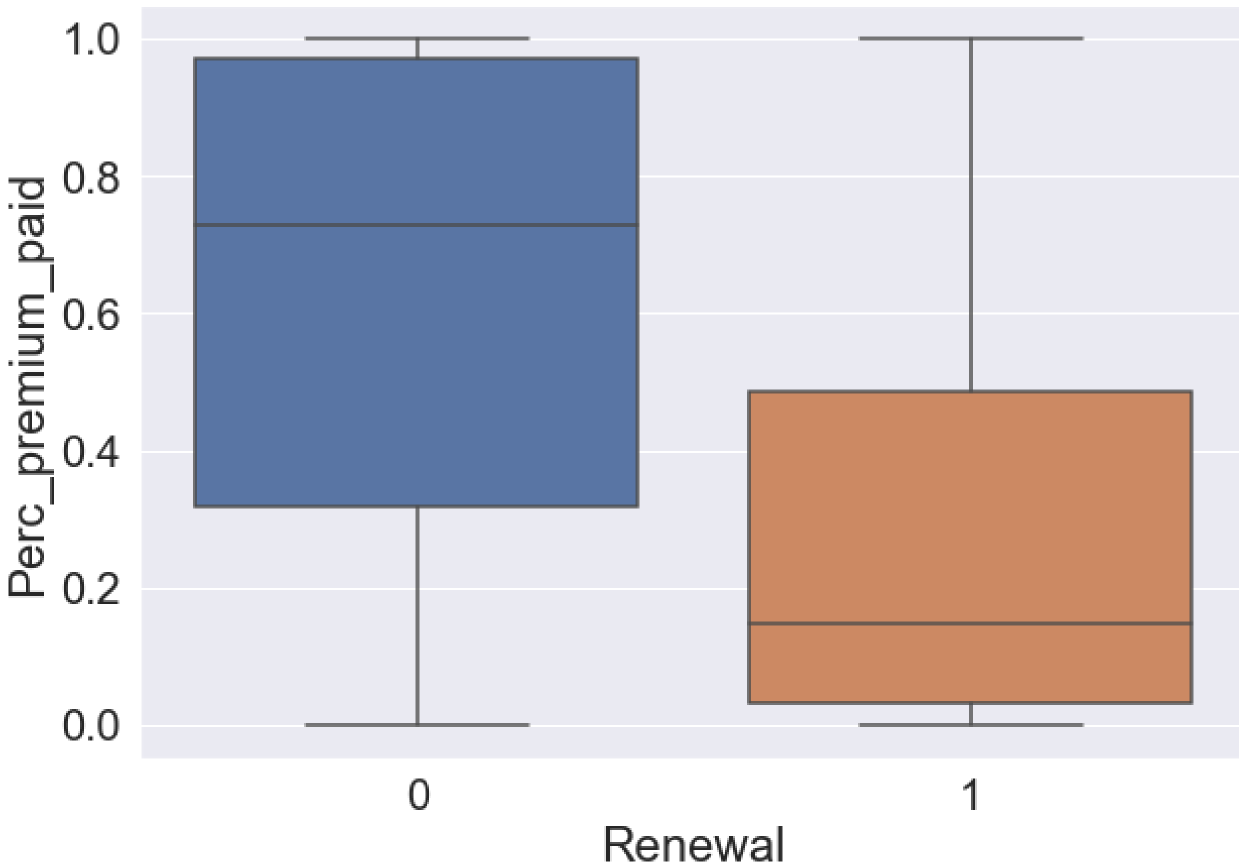
Percentage of premium paid by cash vs. Renewal

In [426…

```
plt.figure(figsize=(10,7))
sns.boxplot(df['Renewal'], df['Perc_premium_paid'])
```

Out[426…

```
<AxesSubplot:xlabel='Renewal', ylabel='Perc_premium_paid'>
```

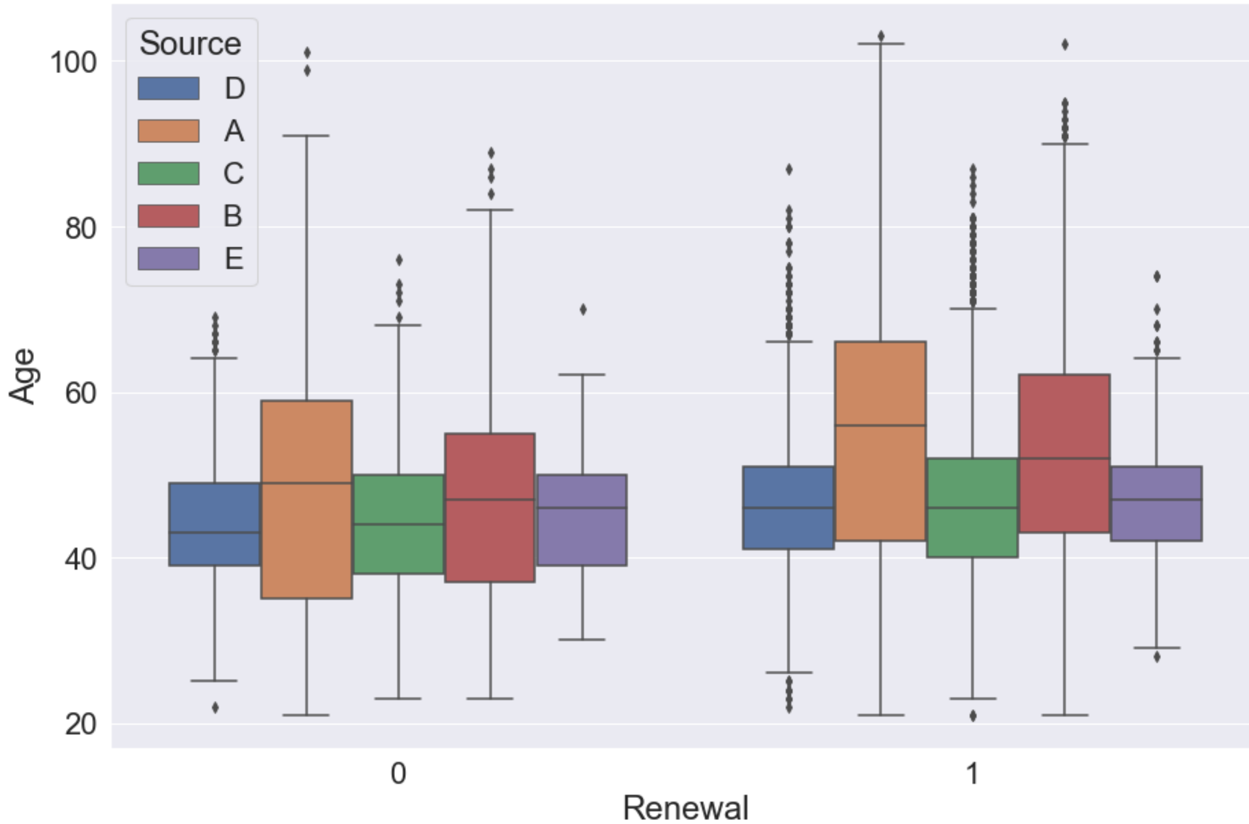


- There are more customers that did not renew AND paid a higher percentage of their premium by cash

Renewal vs. Age vs. Source

In [427…

```
plt.figure(figsize=(15,10))
sns.boxplot(df['Renewal'],df['Age'],hue=df['Source'])
plt.show()
```

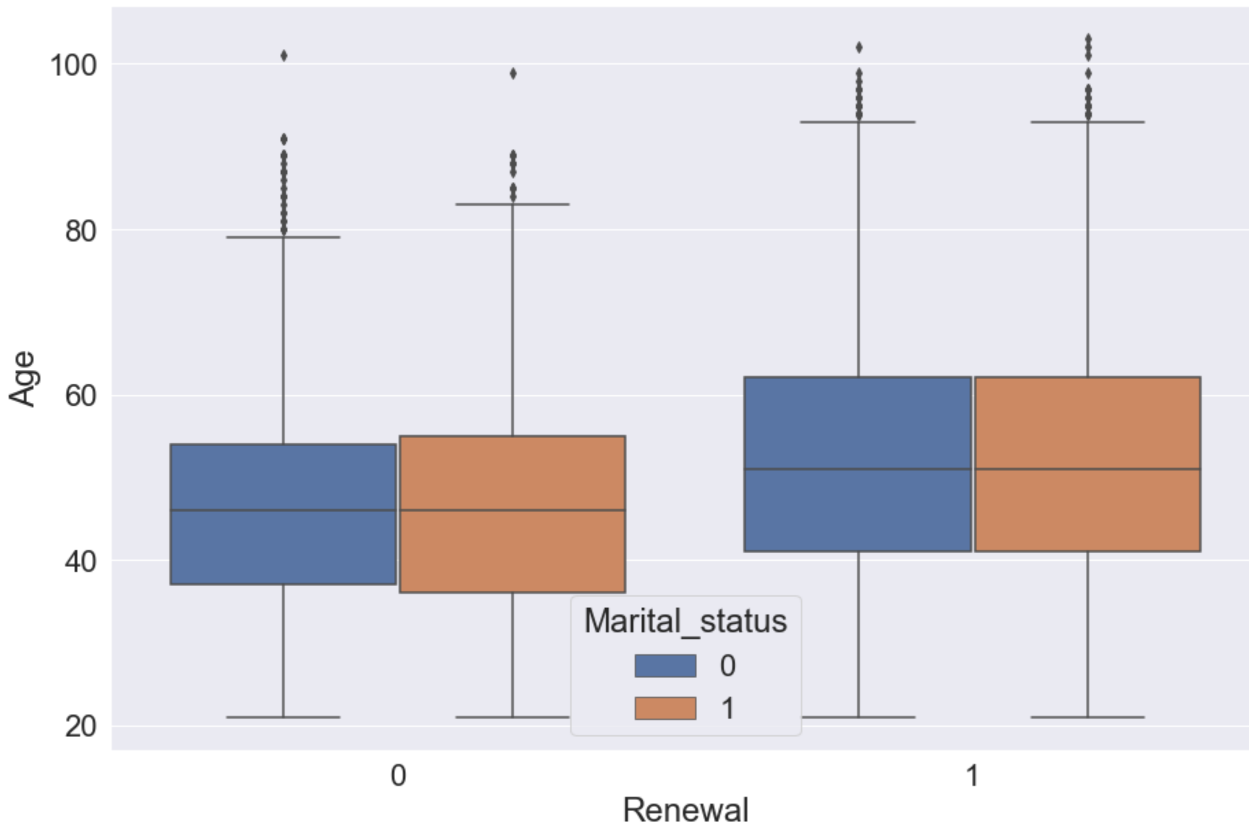


In [ ]:

Renewal vs. Age vs. Marital status

In [428…

```
plt.figure(figsize=(15,10))
sns.boxplot(df['Renewal'],df['Age'],hue=df['Marital_status'])
plt.show()
```

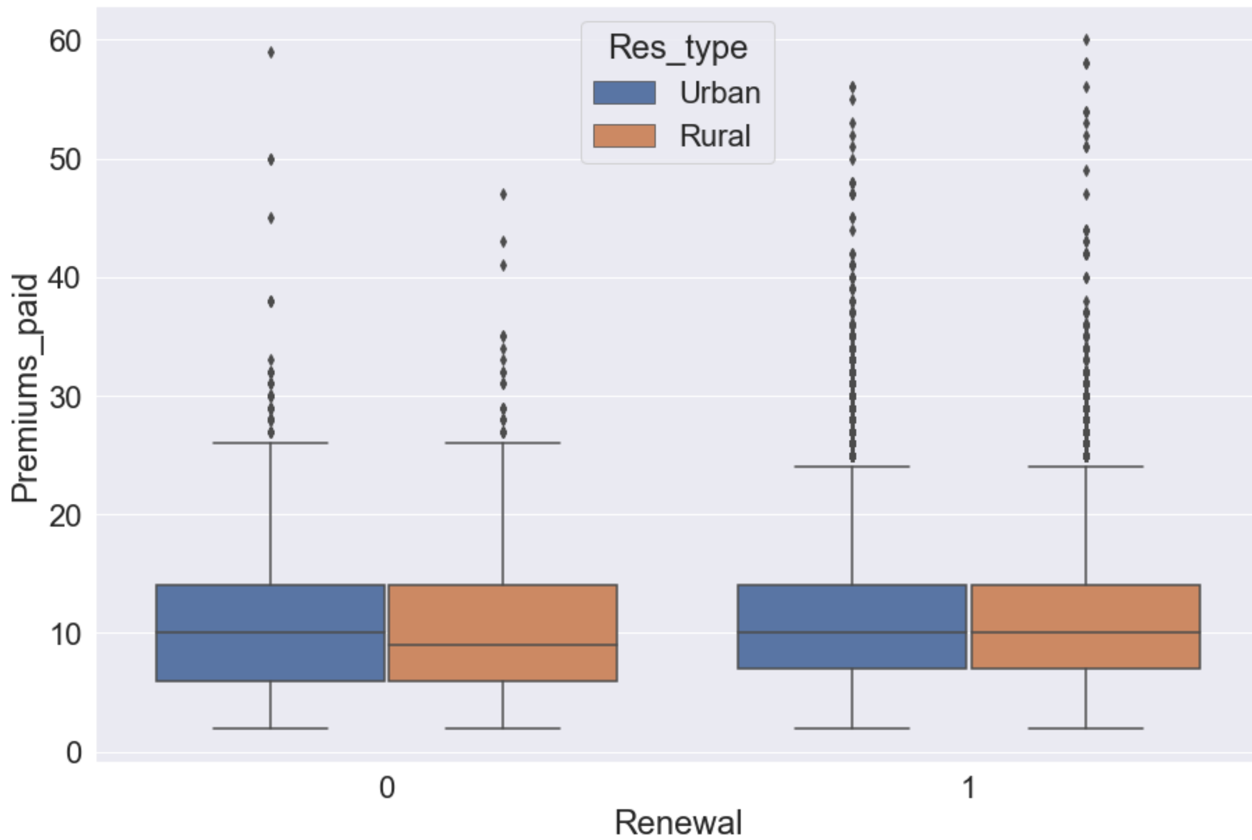


In [ ]:

Renewal vs. Premiums paid vs. Residential type

In [429…

```
plt.figure(figsize=(15,10))
sns.boxplot(df['Renewal'],df['Premiums_paid'],hue=df['Res_type'])
plt.show()
```



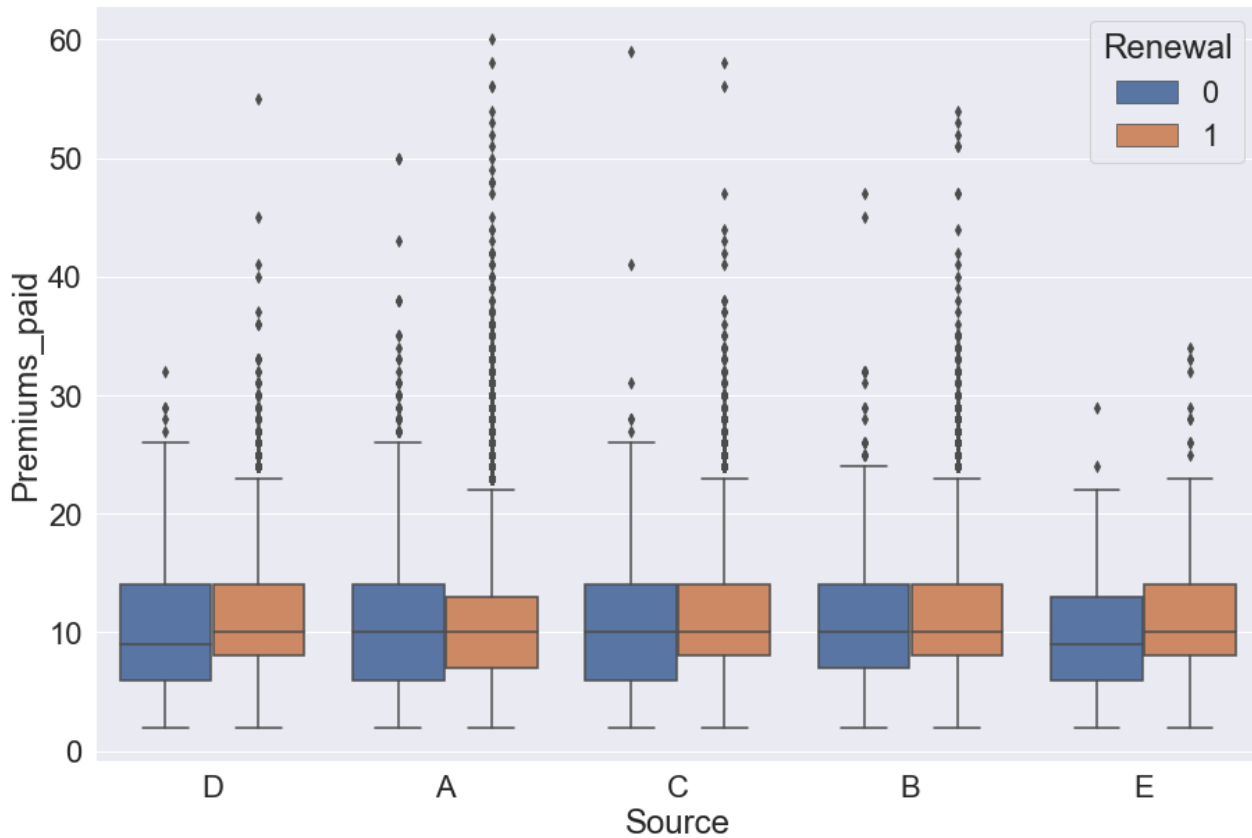
In [ ]:

```

```

Source vs. Premiums paid vs. Renewal

In [430…

```
plt.figure(figsize=(15,10))
sns.boxplot(df['Source'],df['Premiums_paid'],hue=df['Renewal'])
plt.show()
```

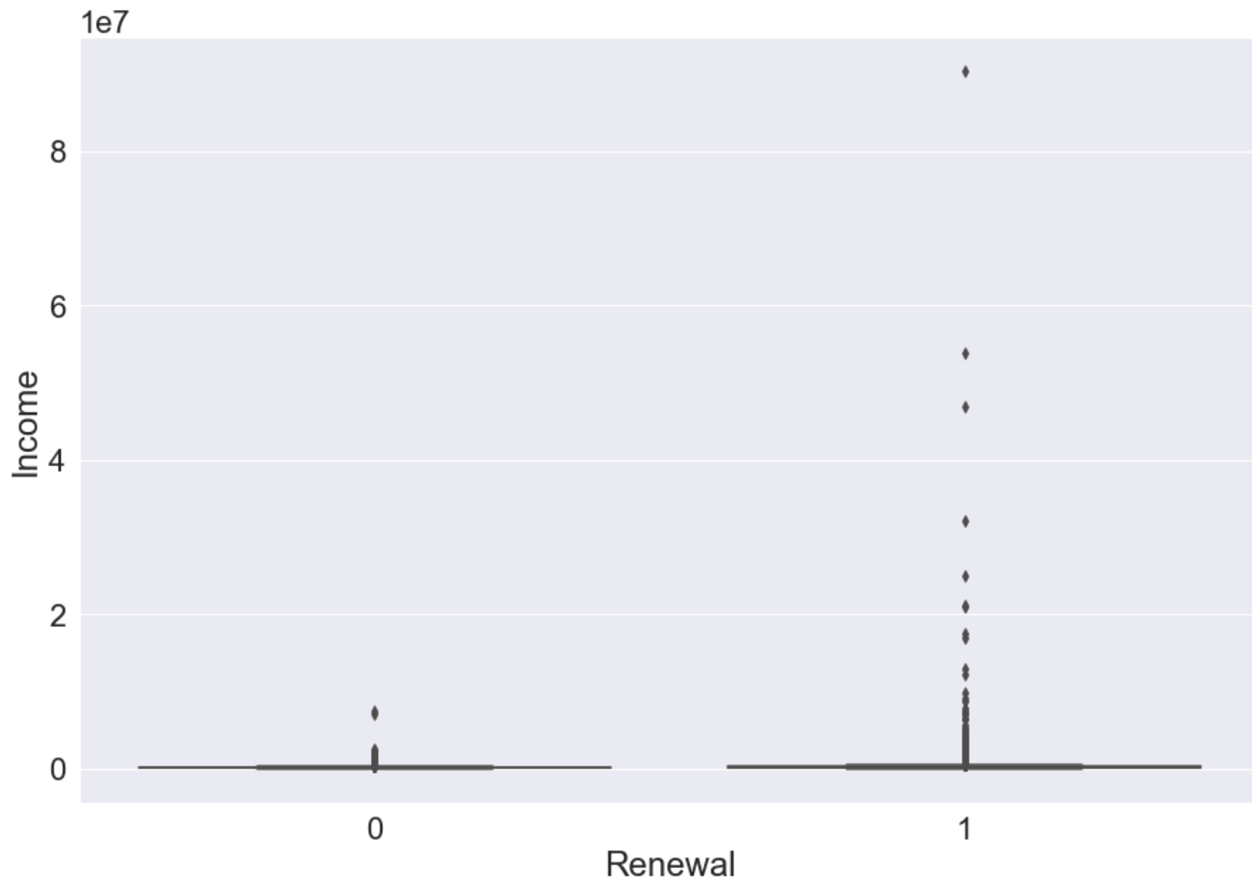


In [ ]:

Renewal vs. Income

In [431…

```
plt.figure(figsize=(15,10))
sns.boxplot(df['Renewal'],df['Income'])
plt.show()
```



In [ ]:

In [432…

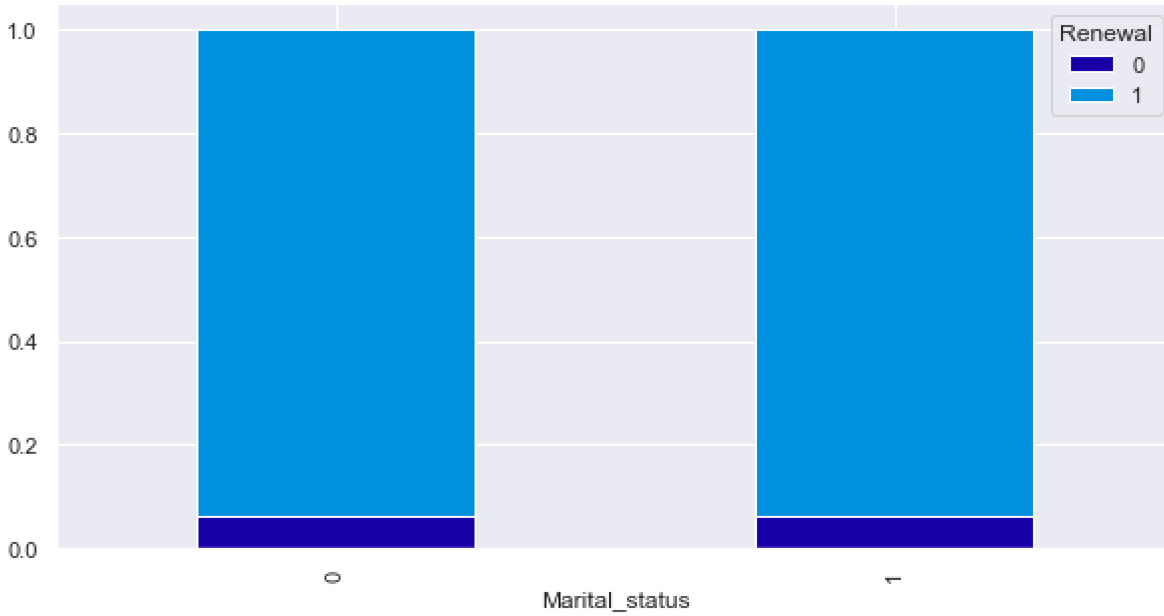
```
## Function to plot stacked bar chart
def stacked_plot(x):
    sns.set(palette='nipy_spectral')
    tab1 = pd.crosstab(x,df['Renewal'],margins=True)
    print(tab1)
    print('-'*120)
    tab = pd.crosstab(x,df['Renewal'],normalize='index')
    tab.plot(kind='bar',stacked=True,figsize=(10,5))
    #plt.legend(loc='lower left', frameon=False)
    #plt.legend(loc="upper left", bbox_to_anchor=(0,1))
    plt.show()
```

Marital status vs. Renewal

In [433…

```
stacked_plot(df['Marital_status'])
```

```
Renewal            0      1    All
Marital_status
0               2571  37461  40032
1               2427  37394  39821
All             4998  74855  79853
----------------------------------------------------------------------------------------
--------------------------------
```

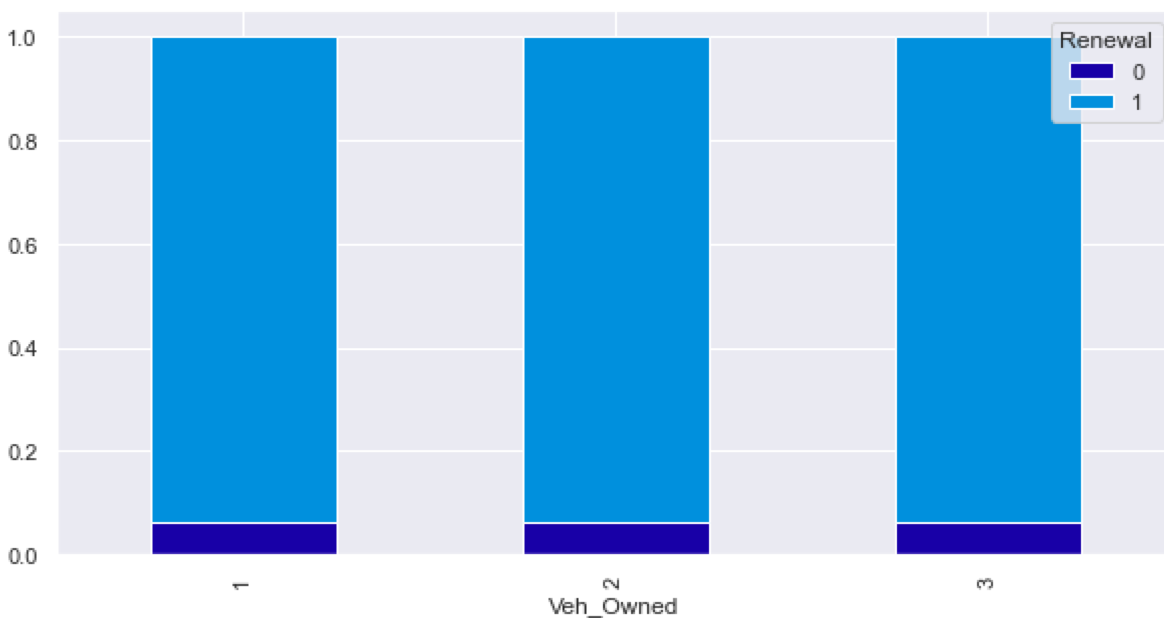


- There is not much difference in the two plots

In [434…

```
stacked_plot(df['Veh_Owned'])
```

```
Renewal       0      1    All
Veh_Owned
1          1668  25078  26746
2          1678  24842  26520
3          1652  24935  26587
All        4998  74855  79853
------------------------------------------------------------------------------------------
--------------------------------
```

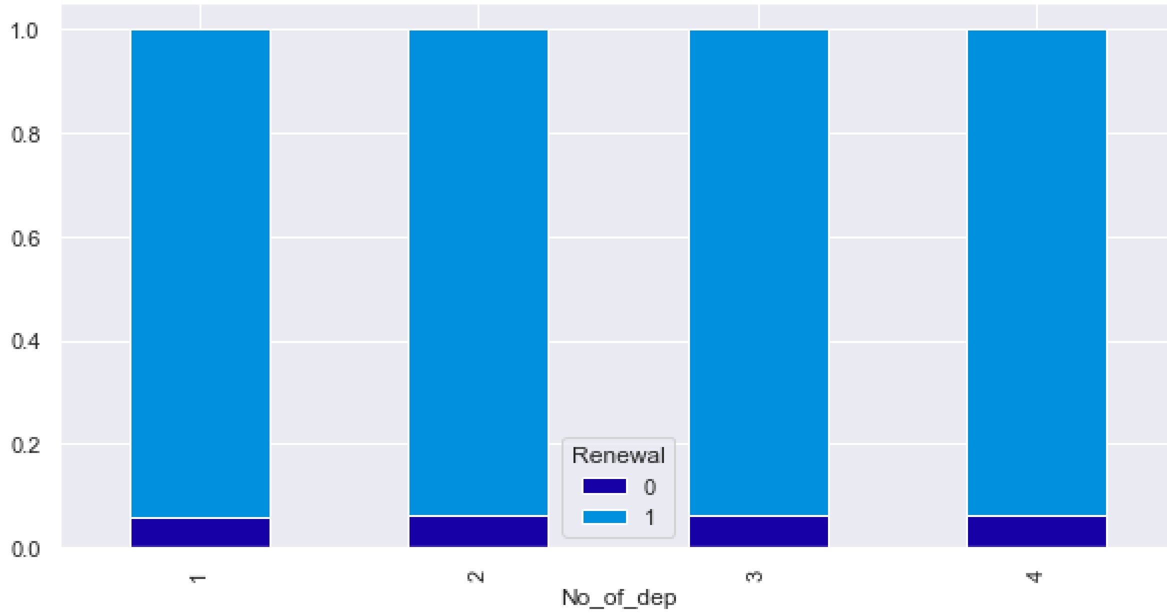


- There is not much difference in the three plots

In [435…

```
stacked_plot(df['No_of_dep'])
```

```
Renewal       0      1    All
No_of_dep
1          1190  18650  19840
2          1258  18644  19902
3          1283  18932  20215
4          1267  18629  19896
All        4998  74855  79853
----------------------------------------------------------------------------------------------------------------------
```

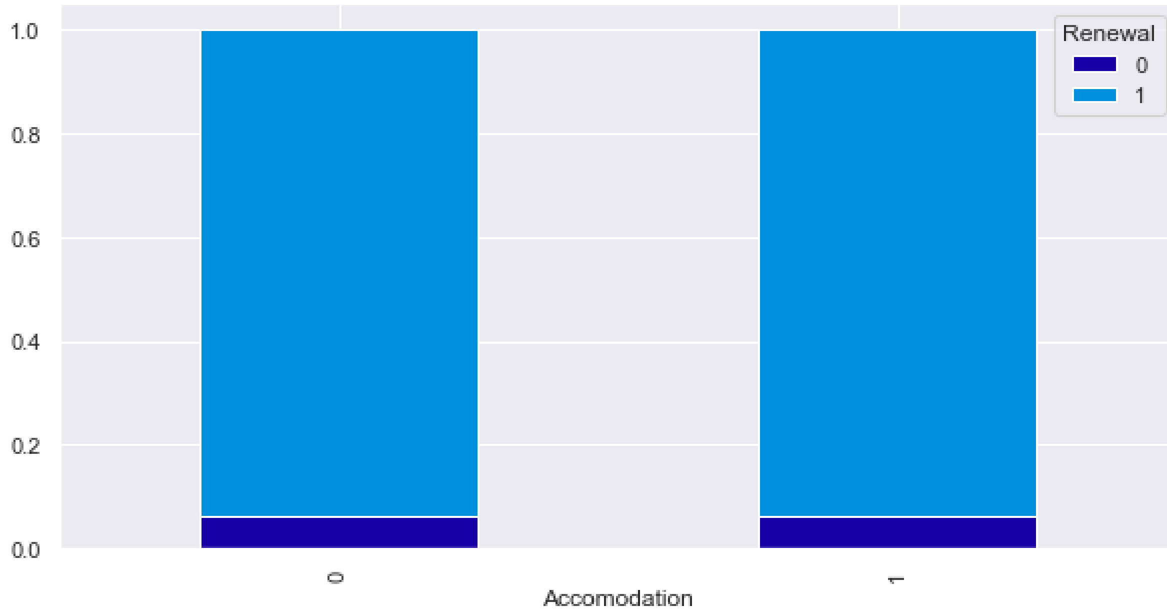


- There is not much difference in the four plots

In [436…

```
stacked_plot(df['Accomodation'])
```

```
Renewal          0      1    All
Accomodation
0             2453  37370  39823
1             2545  37485  40030
All           4998  74855  79853
----------------------------------------------------------------------------------------------------------------------
```

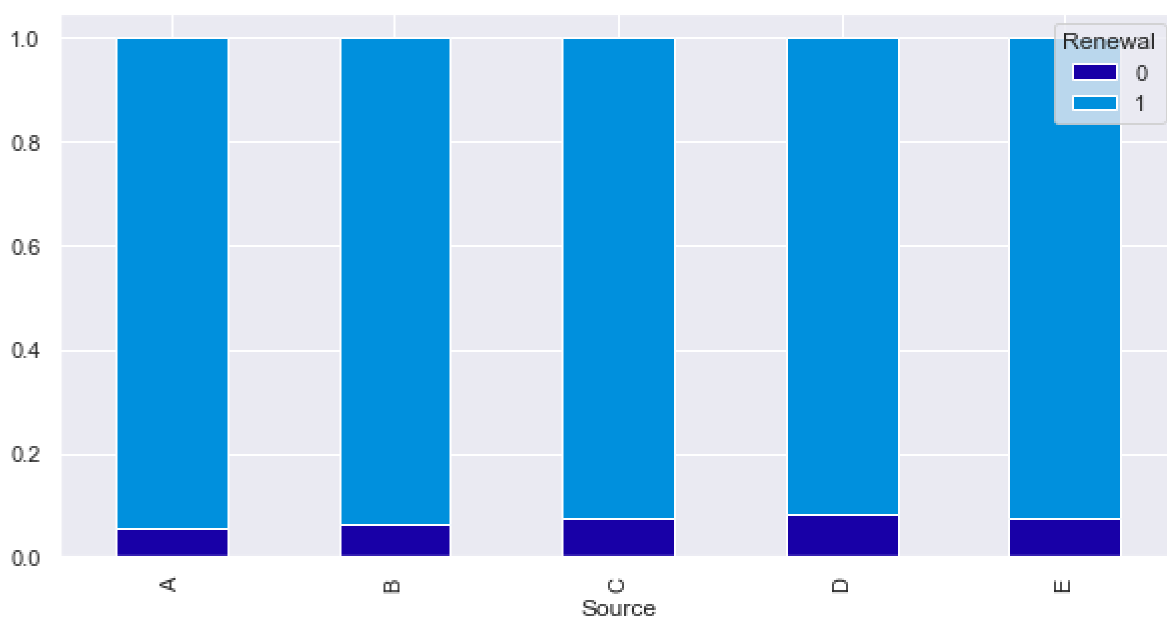


- There is not much difference in the two plots

In [437…

```
stacked_plot(df['Source'])
```

```
Renewal     0      1    All
Source
A        2349  40785  43134
B        1066  15446  16512
C         903  11136  12039
D         634   6925   7559
E          46    563    609
All      4998  74855  79853
------------------------------------------------------------------------------------------
--------------------------------
```

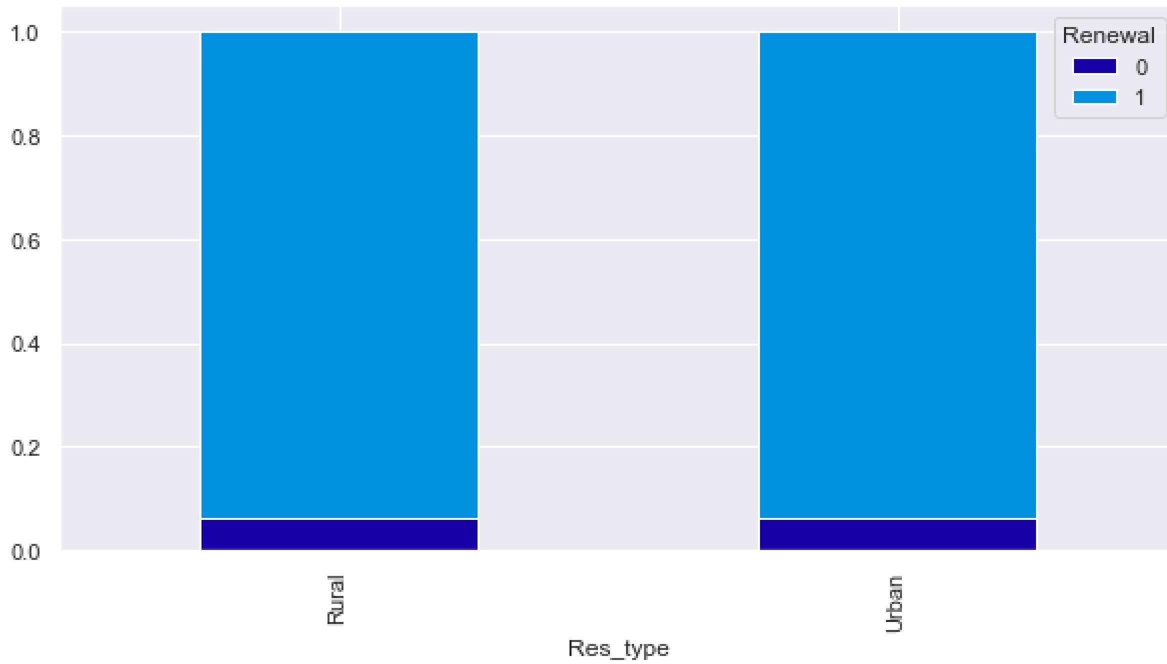


- Source A had a smaller percentage of customers through this source that did not renew.
- The percentages within each source are close globally.

In [438…

```
stacked_plot(df['Res_type'])
```

```
Renewal      0      1    All
Res_type
Rural     1998  29672  31670
Urban     3000  45183  48183
All       4998  74855  79853
----------------------------------------------------------------------------------------------------
--------------------------------
```



- The percentages within each group are similar between both groups.

EDA / Correlation (X vs. X)

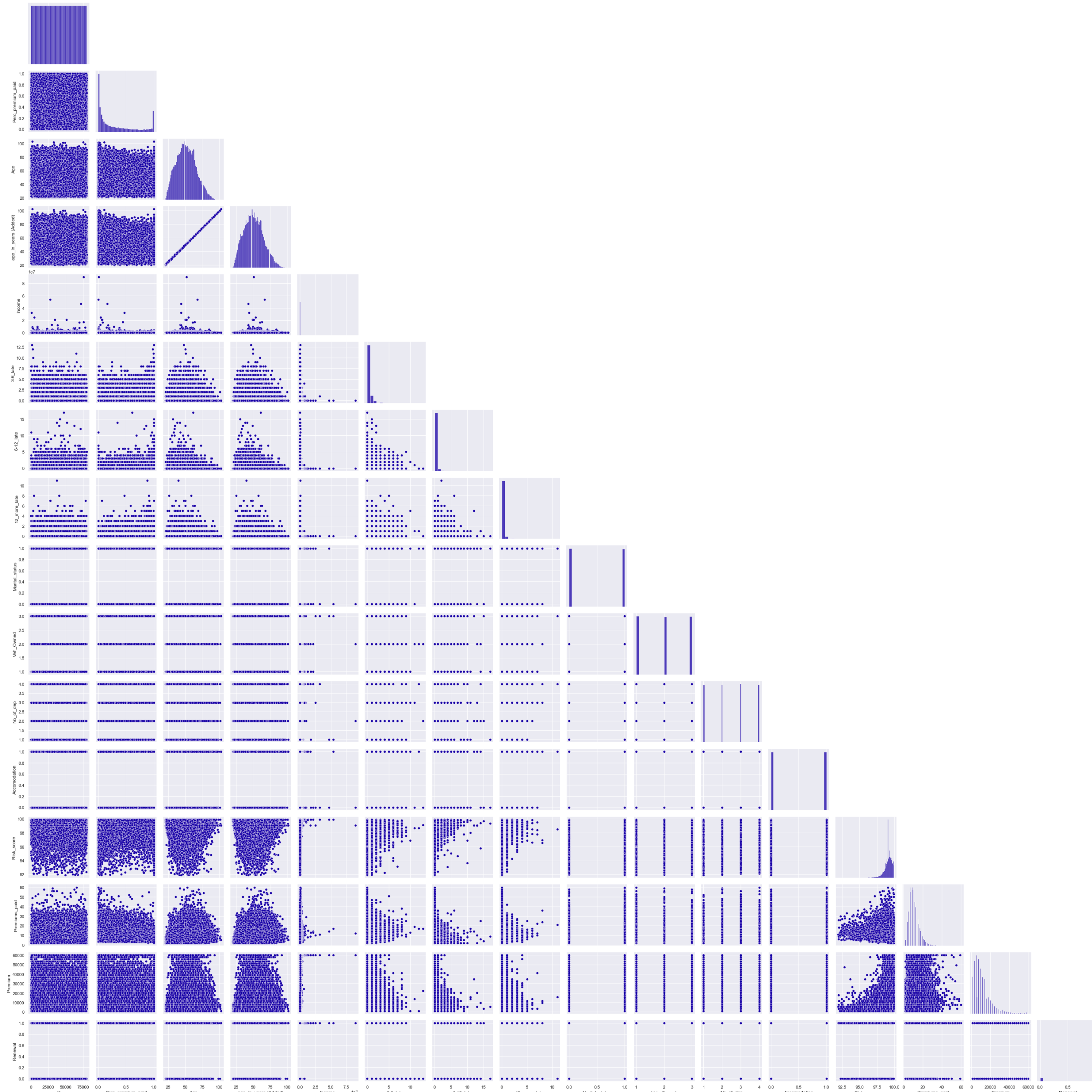
Pairplot

In [439…

```
sns.pairplot(df, corner=True)
```

Out[439…

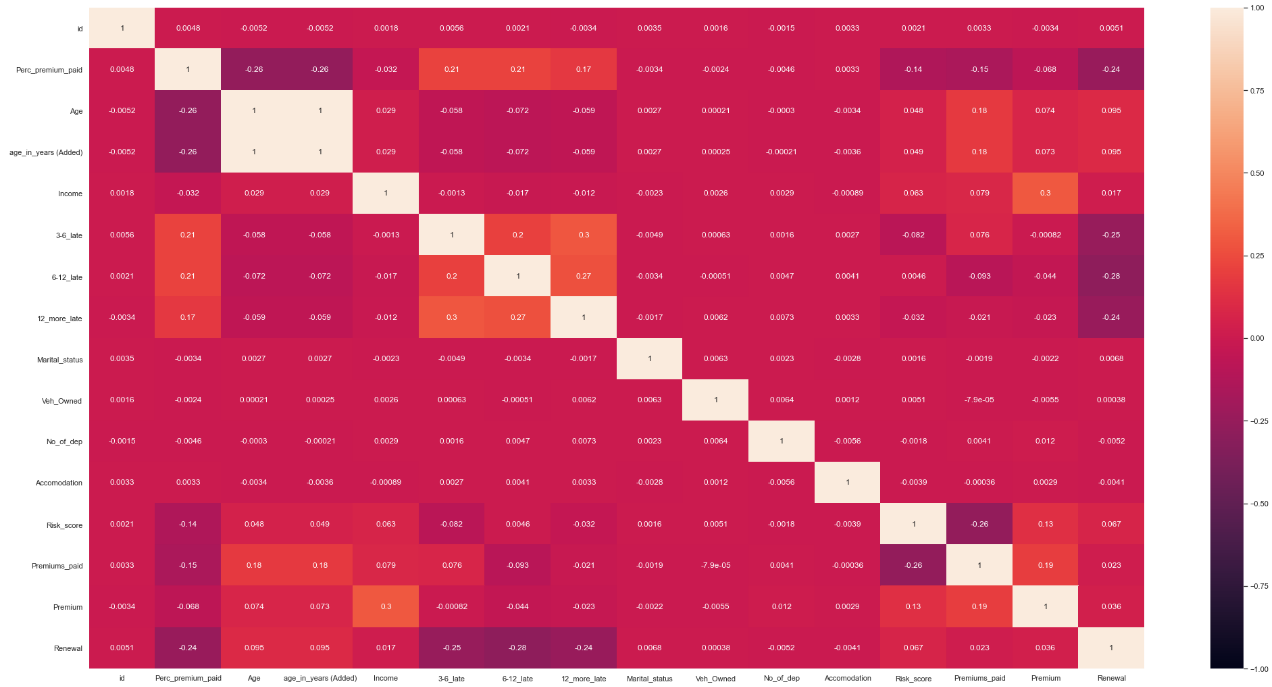
```
<seaborn.axisgrid.PairGrid at 0x21b3a426d00>
```



Heatmap

In [440…

```
plt.figure(figsize=(35,18))
sns.heatmap(df.corr(),annot=True,vmin=-1,vmax=1,fmt='.2g')
plt.show()
```



- There are no strong correlations between variables

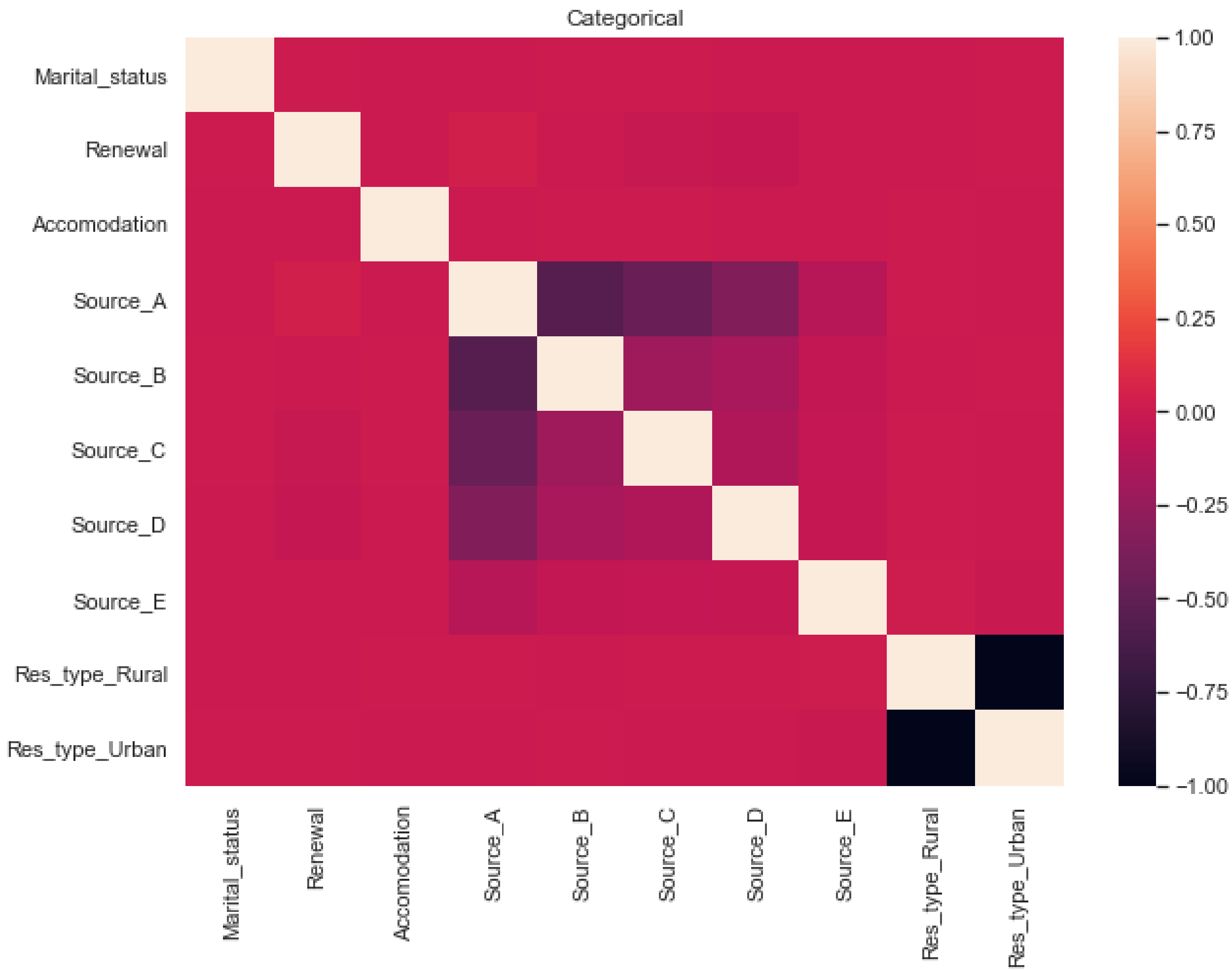
Correlation / Categorical

In [441…

```
# first chose your category columns of interest
data = df[['Marital_status', 'Renewal', 'Accomodation', 'Source', 'Res_type']]

# now change this to dummy variables, one-hot encoded:
DataMatrix = pd.get_dummies(data)

# plot as simply as:
plt.figure(figsize=(10,7))  # for large datasets
plt.title('Categorical')
sns.heatmap(DataMatrix.corr('pearson'), cmap='rocket', center=0)
```

Out[441…

```
<AxesSubplot:title={'center':'Categorical'}>
```



- There are no strong correlations between variables

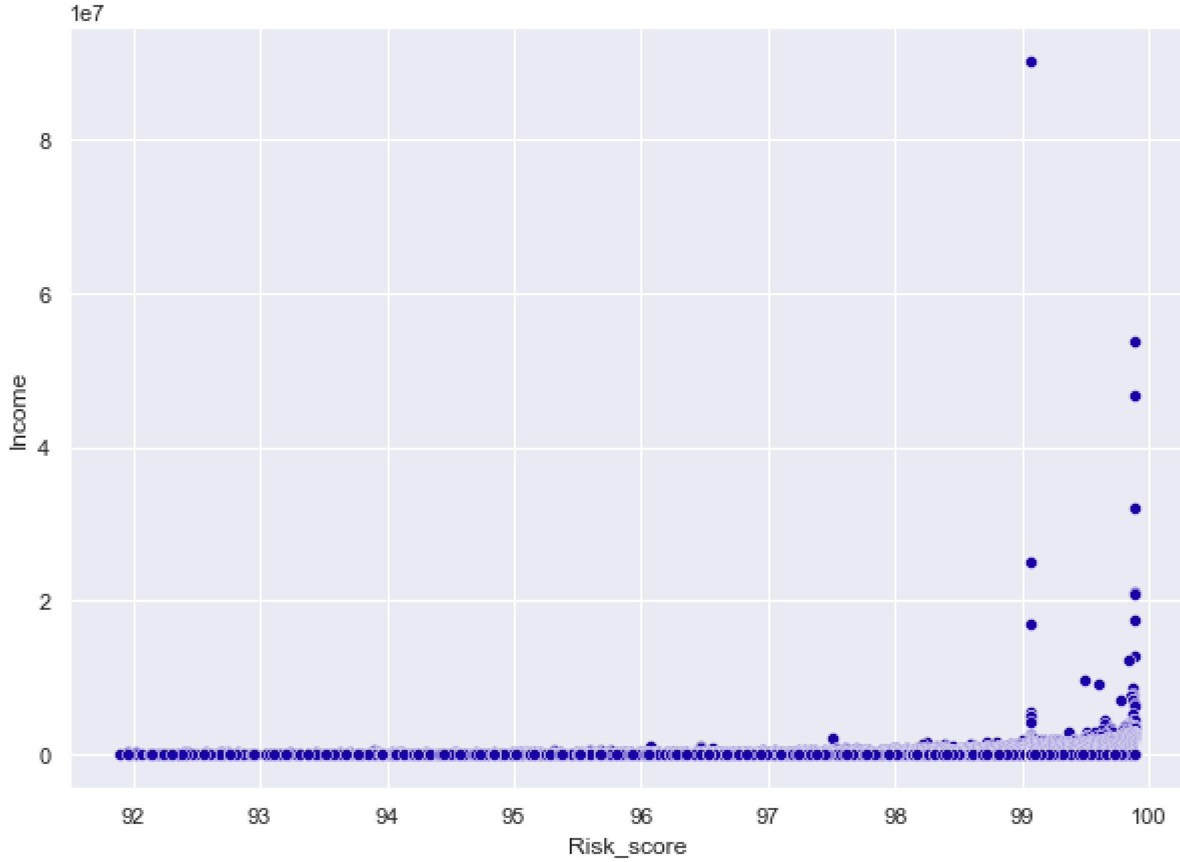
Risk score vs. Income

In [442…

```
plt.figure(figsize=(10,7))
sns.scatterplot(df['Risk_score'], df['Income'])
```

Out[442…

```
<AxesSubplot:xlabel='Risk_score', ylabel='Income'>
```



- There is not much change in the distribution until near 99 where there is an increase in income as the score nears 100

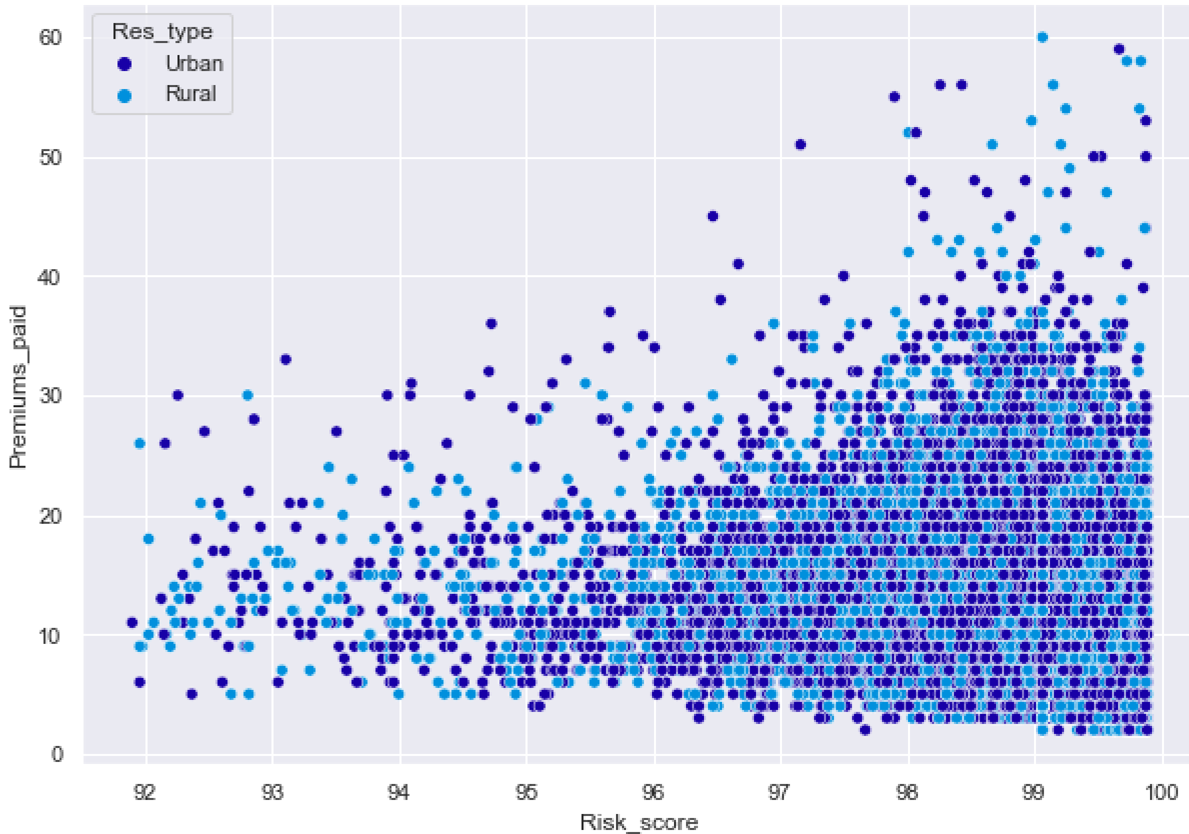
Risk score vs. Premiums paid

In [443…

```
plt.figure(figsize=(10,7))
sns.scatterplot(df['Risk_score'], df['Premiums_paid'], hue=df['Res_type'])
```

Out[443…

```
<AxesSubplot:xlabel='Risk_score', ylabel='Premiums_paid'>
```



- The data hues show no pattern
- There does seem to be a slight correlation where the number of premiums paid is higher as the score increases

Risk score vs. 3-6 late

In [444…

```
plt.figure(figsize=(10,7))
sns.scatterplot(df['Risk_score'], df['3-6_late'], hue=df['Res_type'])
```

Out[444…

```
<AxesSubplot:xlabel='Risk_score', ylabel='3-6_late'>
```

- The data hues show no pattern
- There seems to be a small semblance of correlation wherein as the score increases, so do the counts of late payments

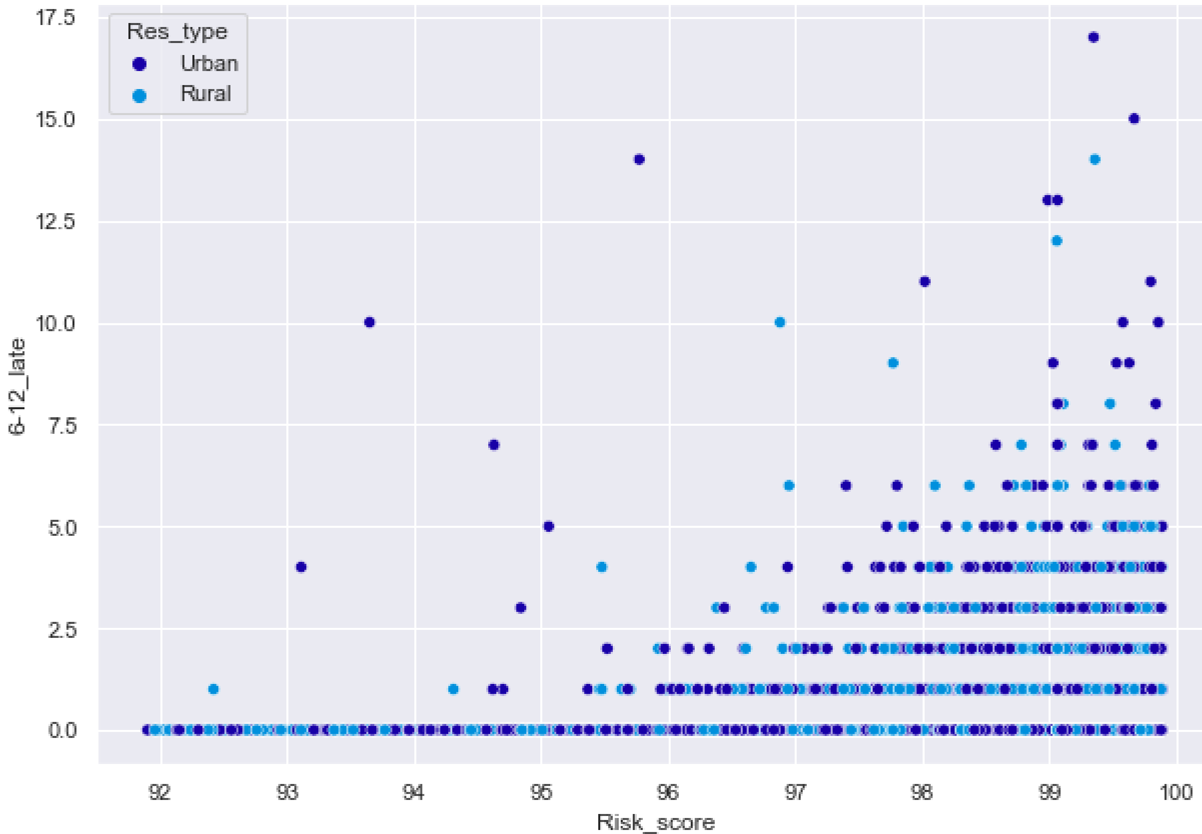
Risk score vs. 6-12 late

In [445…

```
plt.figure(figsize=(10,7))
sns.scatterplot(df['Risk_score'], df['6-12_late'], hue=df['Res_type'])
```

Out[445…

```
<AxesSubplot:xlabel='Risk_score', ylabel='6-12_late'>
```



- The data hues show no pattern
- There does seem to be a slight correlation where the number of premiums paid is higher as the score increases

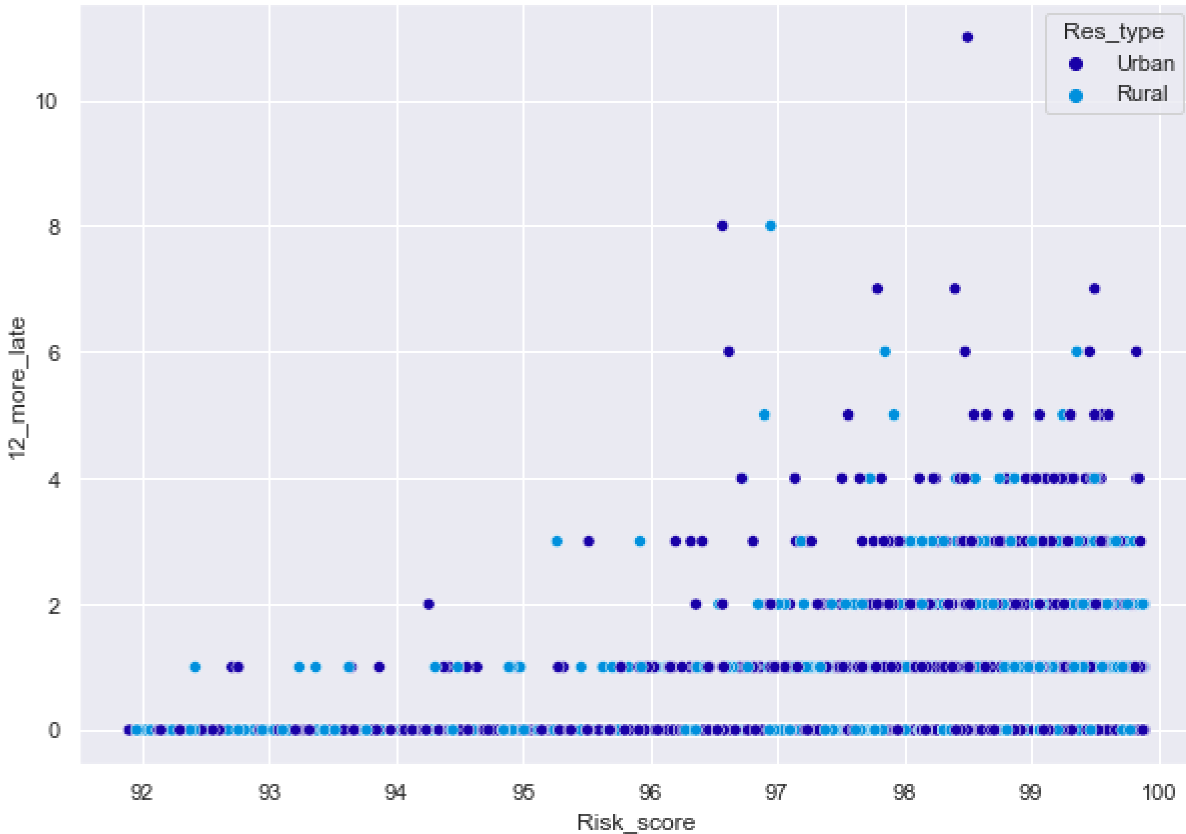
Risk score vs. 12_more late

In [446…

```
plt.figure(figsize=(10,7))
sns.scatterplot(df['Risk_score'], df['12_more_late'], hue=df['Res_type'])
```

Out[446…

```
<AxesSubplot:xlabel='Risk_score', ylabel='12_more_late'>
```



- The data hues show no pattern
- There does seem to be a slight correlation where the number of premiums paid is higher as the score increases

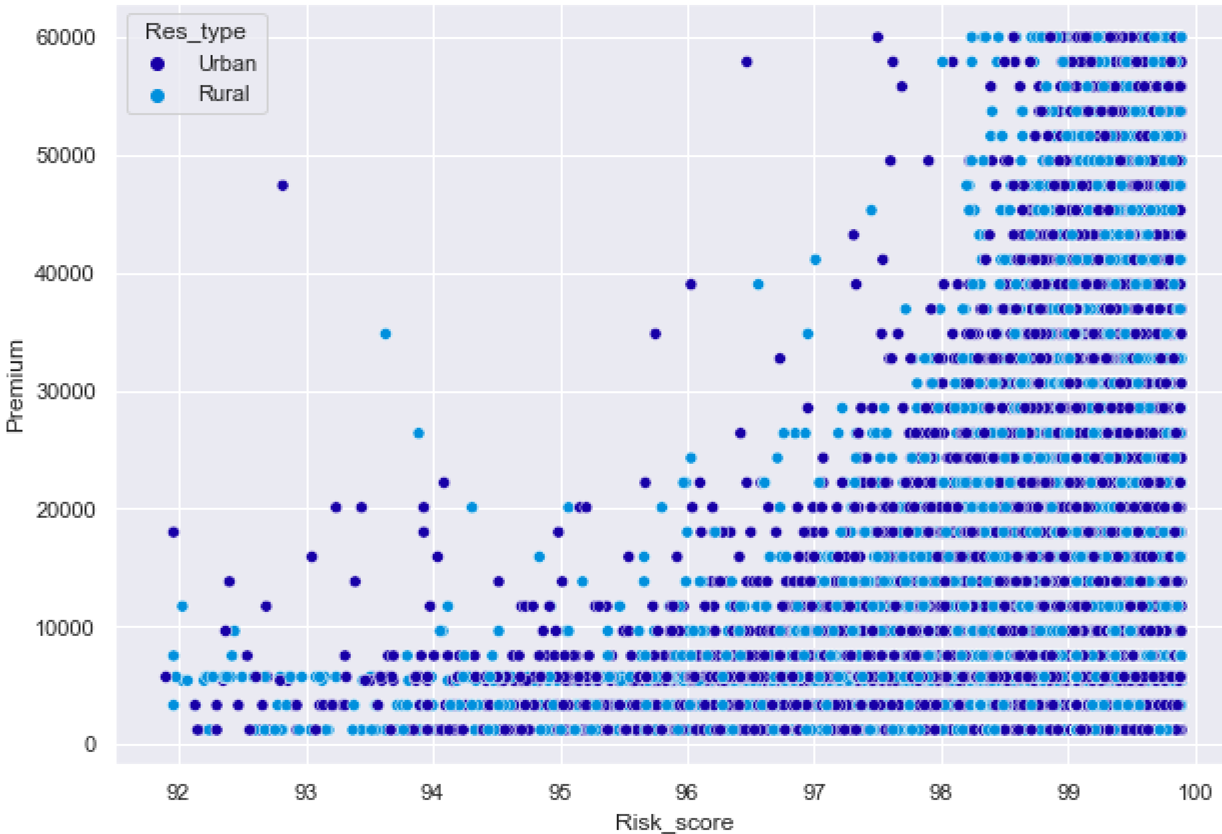
Risk score vs. Premium

In [447…

```
plt.figure(figsize=(10,7))
sns.scatterplot(df['Risk_score'], df['Premium'], hue=df['Res_type'])
```

Out[447…

```
<AxesSubplot:xlabel='Risk_score', ylabel='Premium'>
```



- The data hues show no pattern
- There does seem to be a slight correlation where the number of premiums paid is higher as the score increases

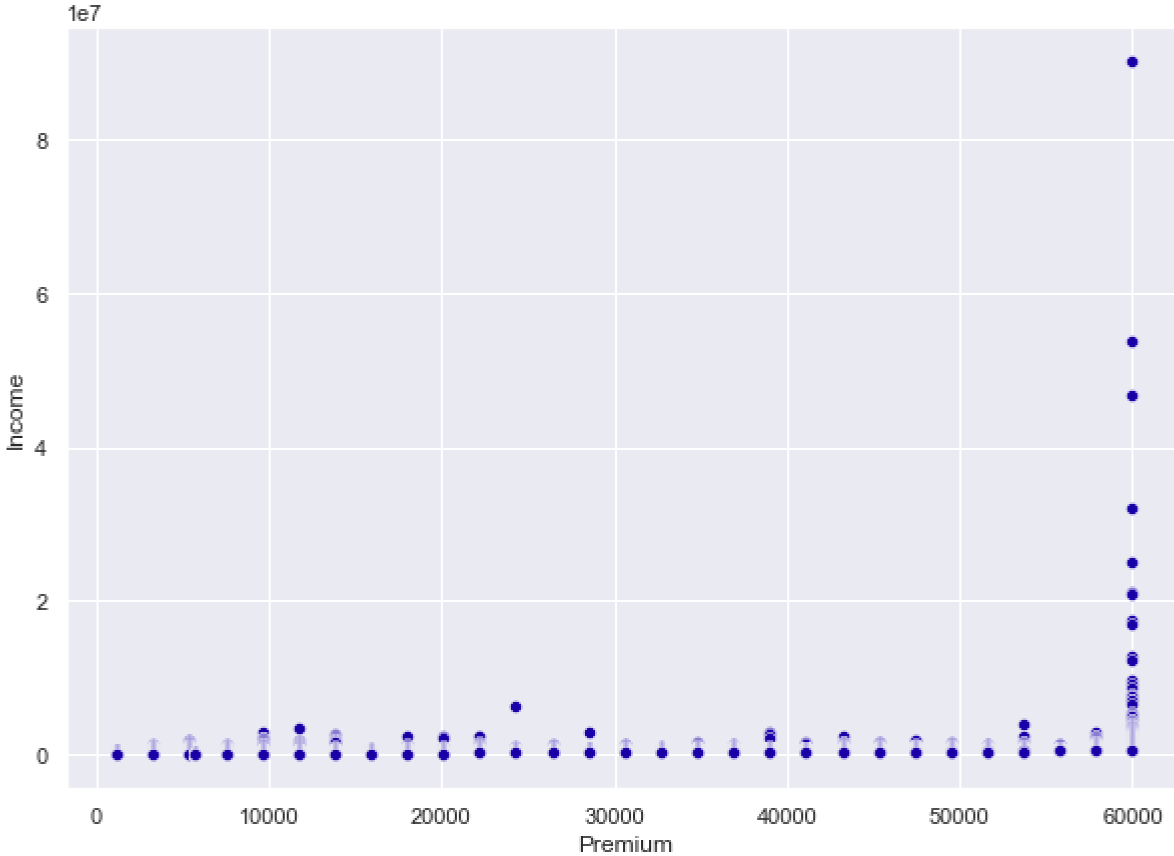
Premium vs. Income

In [448…

```
plt.figure(figsize=(10,7))
sns.scatterplot(df['Premium'], df['Income'])
```

Out[448…

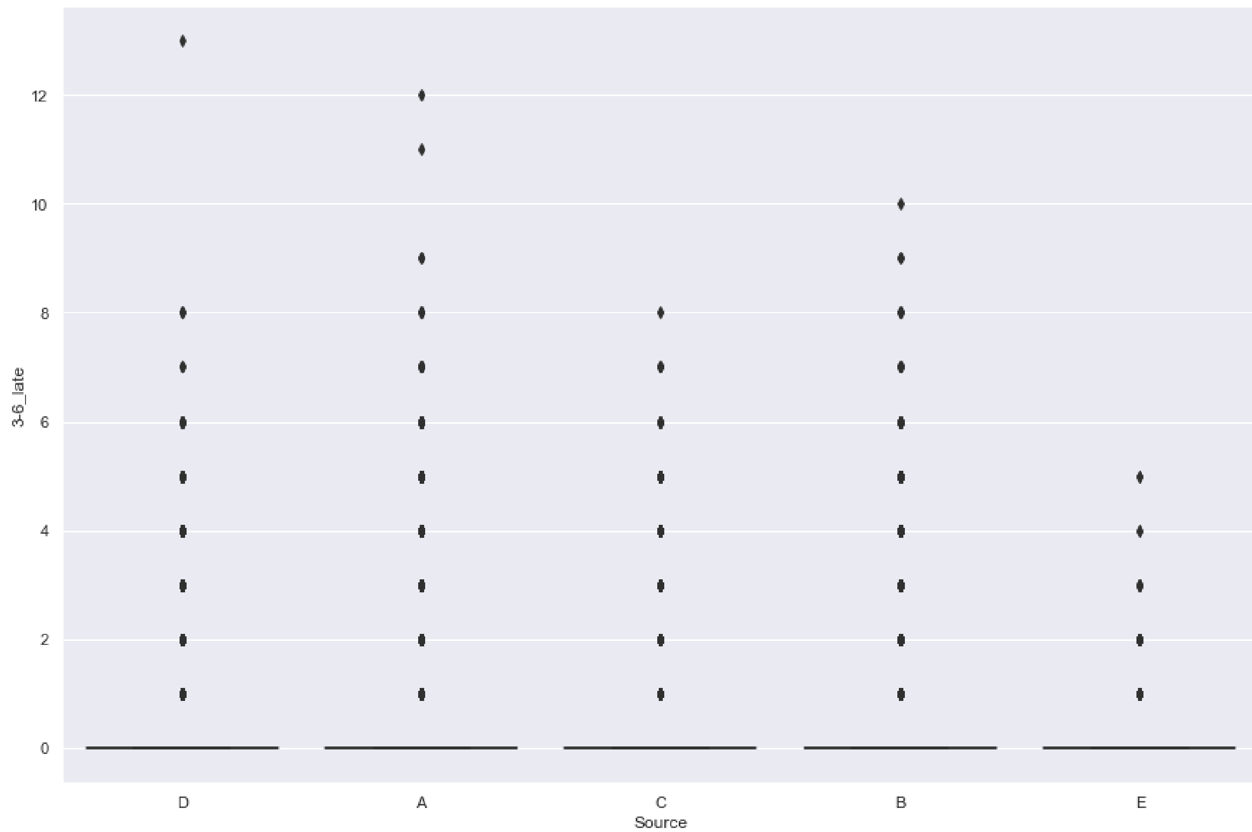
```
<AxesSubplot:xlabel='Premium', ylabel='Income'>
```



Source vs. 3-6 late

In [449...

```
plt.figure(figsize=(15,10))
sns.boxplot(df['Source'],df['3-6_late'])
plt.show()
```



Data Pre-processing

Variable drops

In [450…

```
#The ID variable is just an identifier, and does not contribute to the overall analysis
df.drop('id', axis=1, inplace=True)
```

Missing values

- There are no missing values to impute.

Dtype conversions

In [451…

```
# Dtype conversions
# Convert objects to categories
df.Source = df.Source.astype('category')
df.Res_type = df.Res_type.astype('category')
# df.Marital_status = df.Marital_status.astype('category')
# df.Accomodation = df.Accomodation.astype('category')
# df.Renewal = df.Renewal.astype('category')
df.Age = df.Age.astype('int64')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 79853 entries, 0 to 79852
```

```
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Perc_premium_paid    79853 non-null  float64
 1   Age                  79853 non-null  int64
 2   age_in_years (Added) 79853 non-null  int64
 3   Income               79853 non-null  int64
 4   3-6_late             79853 non-null  int64
 5   6-12_late            79853 non-null  int64
 6   12_more_late         79853 non-null  int64
 7   Marital_status       79853 non-null  int64
 8   Veh_Owned            79853 non-null  int64
 9   No_of_dep            79853 non-null  int64
 10  Accomodation         79853 non-null  int64
 11  Risk_score           79853 non-null  float64
 12  Premiums_paid        79853 non-null  int64
 13  Source               79853 non-null  category
 14  Res_type             79853 non-null  category
 15  Premium              79853 non-null  int64
 16  Renewal              79853 non-null  int64
dtypes: category(2), float64(2), int64(13)
memory usage: 9.3 MB
```

Create new Feature variable

In [452…

```
# Create new column with sum of all late counts
# This could be used for EDA only or in models (e.g., clustering), at which time the re
# be dropped to avoid multicollinearity
df['Total_late'] = df['3-6_late']+df['6-12_late']+df['12_more_late']
```

In [453…

```
# Re-order column names
df=df[['Perc_premium_paid', 'Age', 'Income', '3-6_late', '6-12_late',
       '12_more_late', 'Total_late', 'Marital_status', 'Veh_Owned', 'No_of_dep',
       'Accomodation', 'Risk_score', 'Premiums_paid', 'Source', 'Res_type',
       'Premium', 'Renewal']]
```

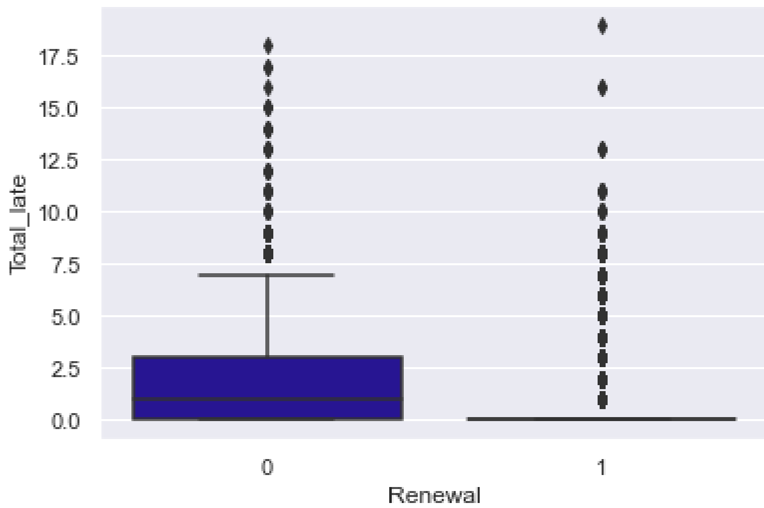
New feature plot against Y

In [454…

```
sns.boxplot(df['Renewal'], df['Total_late'])
```

Out[454…

```
<AxesSubplot:xlabel='Renewal', ylabel='Total_late'>
```



- More customers that did not renew had more times of paying late across the time frame

Round values

In [455…

```
# Round Perc_premium_paid and Risk_score to 2 decimal places
df['Perc_premium_paid'] = df['Perc_premium_paid'].round(2)
df['Risk_score'] = df['Risk_score'].round(2)
```

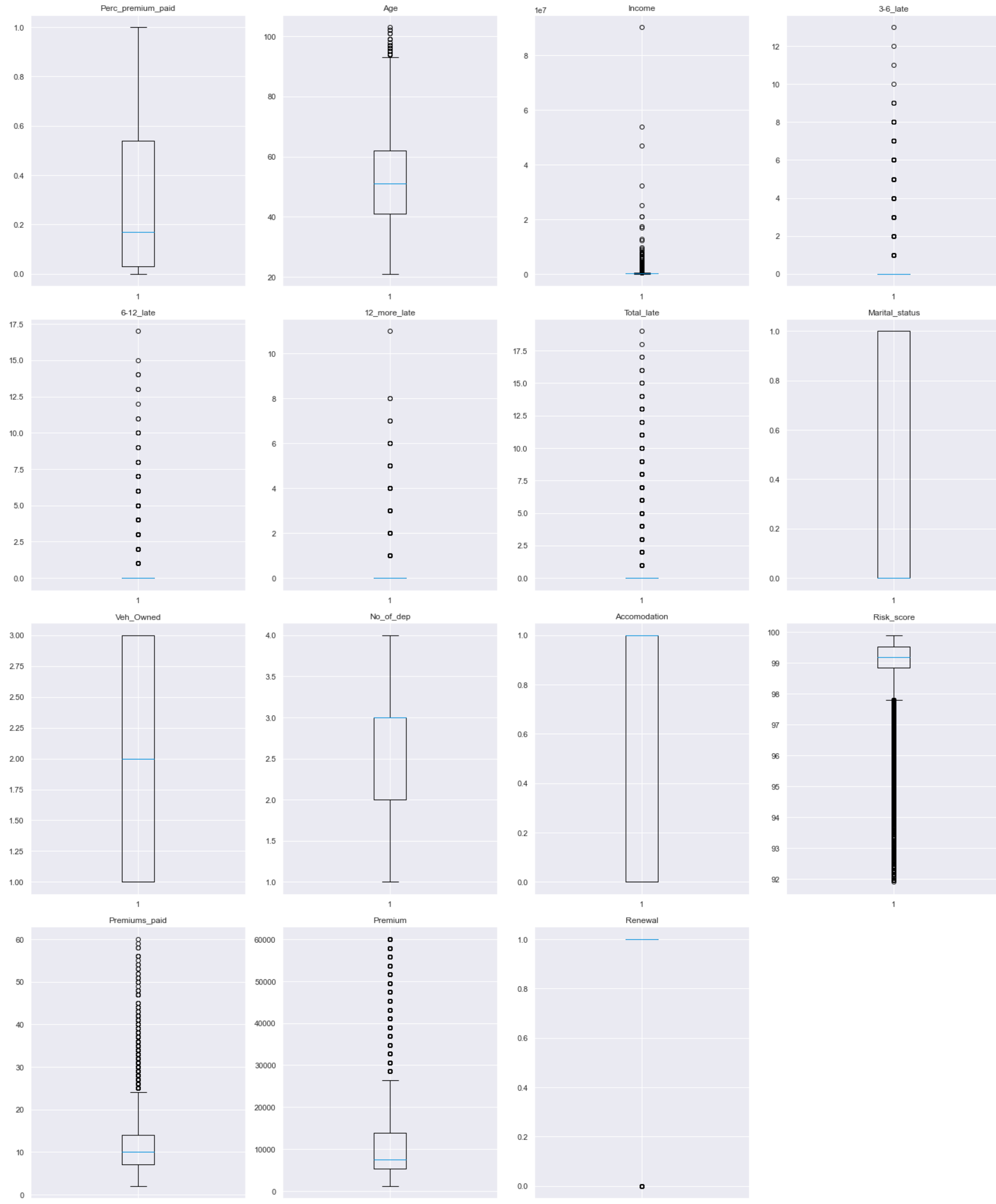
Outlier treatment

In [456…

```
# Make a copy of original datset as we will revert to this one for modeling. df2 will o
# distributions after removing outliers
df2=df.copy()
```

In [457…

```
# Outlier detection using boxplot

numerical_col = df.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(20,30))

for i, variable in enumerate(numerical_col):
                     plt.subplot(5,4,i+1)
                     plt.boxplot(df[variable],whis=1.5)
                     plt.tight_layout()
                     plt.title(variable)
plt.show()
```



- There are outliers within the variables: Age, Income, Risk score, Premiums paid, and Premium

In [458…

```
# Function for treating outliers

def treat_outliers(df2,col):
    '''
    treats outliers in a varaible
    col: str, name of the numerical varaible
    df2: data frame
    col: name of the column
```

```
    '''
    Q1=df2[col].quantile(0.25) # 25th quantile
    Q3=df2[col].quantile(0.75)  # 75th quantile
    IQR=Q3-Q1
    Lower_Whisker = Q1 - 1.5*IQR
    Upper_Whisker = Q3 + 1.5*IQR
    df2[col] = np.clip(df2[col], Lower_Whisker, Upper_Whisker) # all the values samller
                                                                # and all the values above
    return df2

def treat_outliers_all(df2, col_list):
    '''
    treat outlier in all numerical varaibles
    col_list: list of numerical varaibles
    df2: data frame
    '''
    for c in col_list:
        df2 = treat_outliers(df2,c)

    return df2
```

In [459…

```
treat_outliers(df2, 'Income')
treat_outliers(df2, 'Risk_score')
treat_outliers(df2, 'Premiums_paid')
treat_outliers(df2, 'Premium')
treat_outliers(df2, 'Age')
```

Out[459…

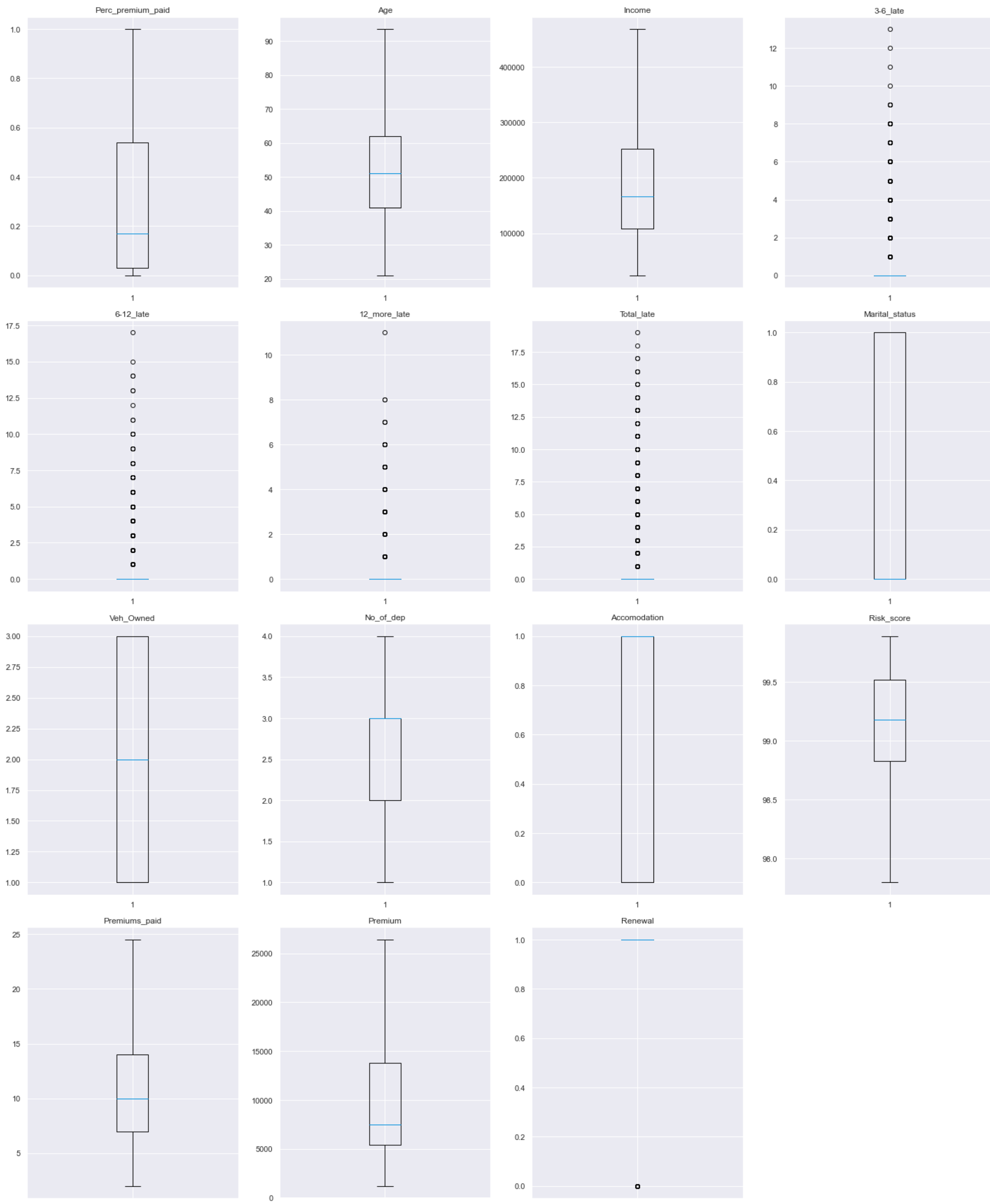
	Perc_premium_paid	Age	Income	3-6_late	6-12_late	12_more_late	Total_late	Marital_status	Veh
0	0.01	52.0	468210.0	0	0	0	0	0	
1	0.00	68.0	468210.0	0	0	0	0	0	
2	0.16	44.0	468210.0	0	0	0	0	1	
3	0.47	44.0	468210.0	1	0	0	1	0	
4	0.04	55.0	468210.0	0	0	0	0	1	
...	...	...	...	...	...	...	...	...	
79848	0.92	30.0	24030.0	0	0	0	0	1	
79849	1.00	27.0	24030.0	0	0	0	0	0	
79850	0.33	26.0	24030.0	0	0	1	1	1	
79851	1.00	24.0	24030.0	0	0	0	0	0	
79852	0.61	22.0	24030.0	0	0	0	0	0	

79853 rows × 17 columns

In [460…

```
# Check for outliers after removal

numerical_col = df2.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(20,30))
```

```
for i, variable in enumerate(numerical_col):
                     plt.subplot(5,4,i+1)
                     plt.boxplot(df2[variable],whis=1.5)
                     plt.tight_layout()
                     plt.title(variable)
plt.show()
```



EDA Income histogram after removal of outliers

In [461…

```
plt.figure(figsize=(12,7))
sns.distplot(df2[df2["Renewal"] == 0]['Income'], color = 'r',label='Renewal=0')
sns.distplot(df2[df2["Renewal"] == 1]['Income'], color = 'b',label='Renewal=1')
plt.legend()
plt.title("Income Distribution");
```

- The income distribution looks much better after outliers were removed. However, there is still a right skew to the distribution, meaning that there are a good number of people who are earning quite a bit.
- As income increases, there are more customers who renewed their insurance than those that did not. However, there is an overall decrease in the number of renewals which coincides with the decrease in the number of people at the high end of the income spectrum.
- Those with lower income tend to not renew as much. This could possibly be due to the cost of the premium that prices the customers out

In [462…

```
df.Income.describe()
```

Out[462…

```
count    7.985300e+04
mean     2.088472e+05
std      4.965826e+05
min      2.403000e+04
25%      1.080100e+05
50%      1.665600e+05
75%      2.520900e+05
max      9.026260e+07
Name: Income, dtype: float64
```

- 75% of the customers make less than $252,090/yr

EDA Age histogram after removal of outliers

In [463...

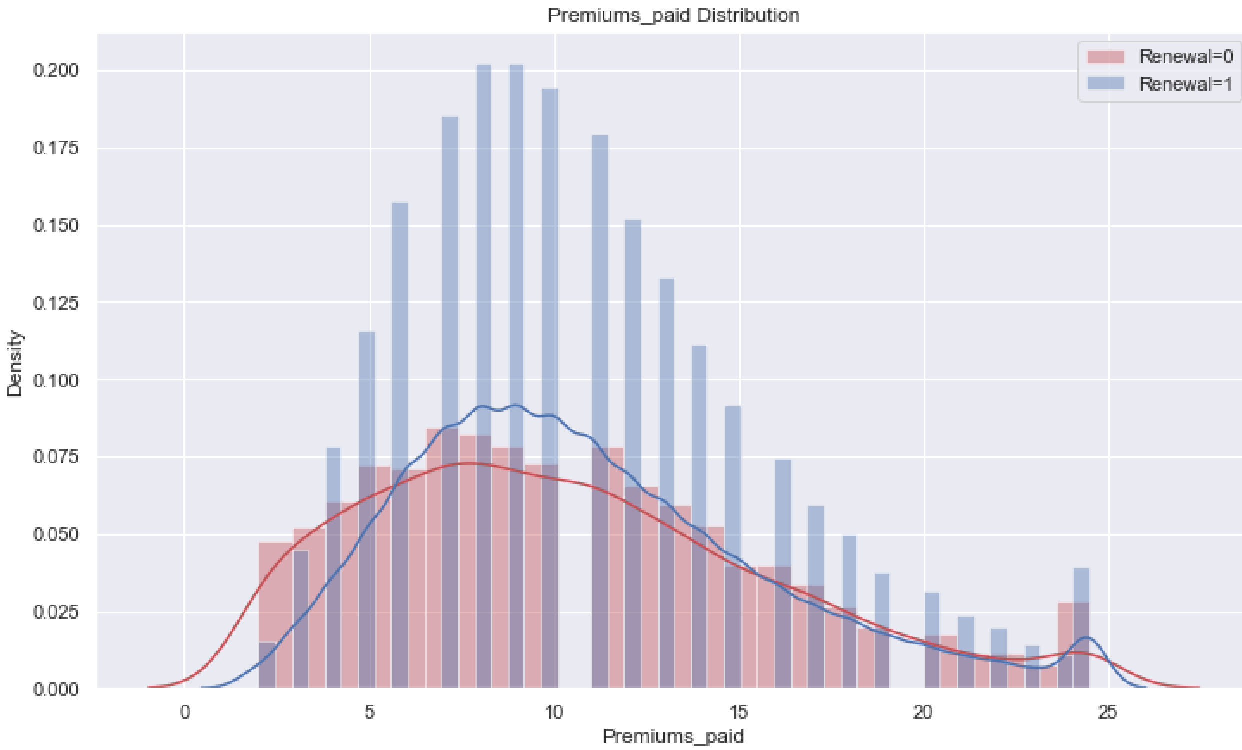
```
plt.figure(figsize=(12,7))
sns.distplot(df2[df2["Renewal"] == 0]['Age'], color = 'r',label='Renewal=0')
sns.distplot(df2[df2["Renewal"] == 1]['Age'], color = 'b',label='Renewal=1')
plt.legend()
plt.title("Age Distribution");
```

- The Age distribution is fairly normal, however we can see that as the age increases, so does the rate of renewals. It would be interesting to see a percentage comparision of renewals to ages.
- There are many non-renewals in the younger age brackets.
- There are peaks almost equidistant from each other (about every 4 years) in renewals, which would indicate a policy renewal cycling (change or modification).

EDA Premiums paid histogram after removal of outliers

In [464...

```
plt.figure(figsize=(12,7))
sns.distplot(df2[df2["Renewal"] == 0]['Premiums_paid'], color = 'r',label='Renewal=0')
sns.distplot(df2[df2["Renewal"] == 1]['Premiums_paid'], color = 'b',label='Renewal=1')
plt.legend()
plt.title("Premiums_paid Distribution");
```

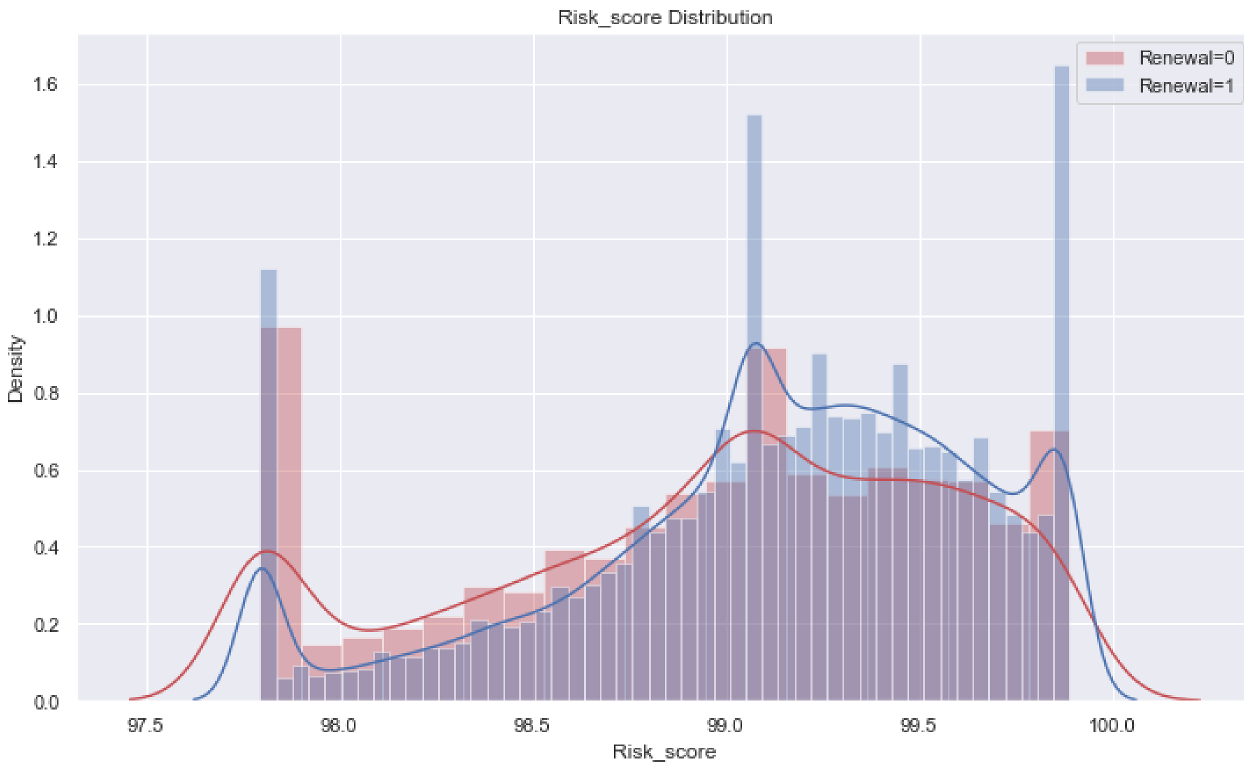


- The premiums-paid distribution looks much more normal, with a slight right skew remaining.
- For those paying their premiums, the renewal comparison is obviously greater.
- It goes without saying that if someone is paying their premiums, they most likely will renew.

EDA Risk score histogram after removal of outliers

In [465…

```
plt.figure(figsize=(12,7))
sns.distplot(df2[df2["Renewal"] == 0]['Risk_score'], color = 'r',label='Renewal=0')
sns.distplot(df2[df2["Renewal"] == 1]['Risk_score'], color = 'b',label='Renewal=1')
plt.legend()
plt.title("Risk_score Distribution");
```

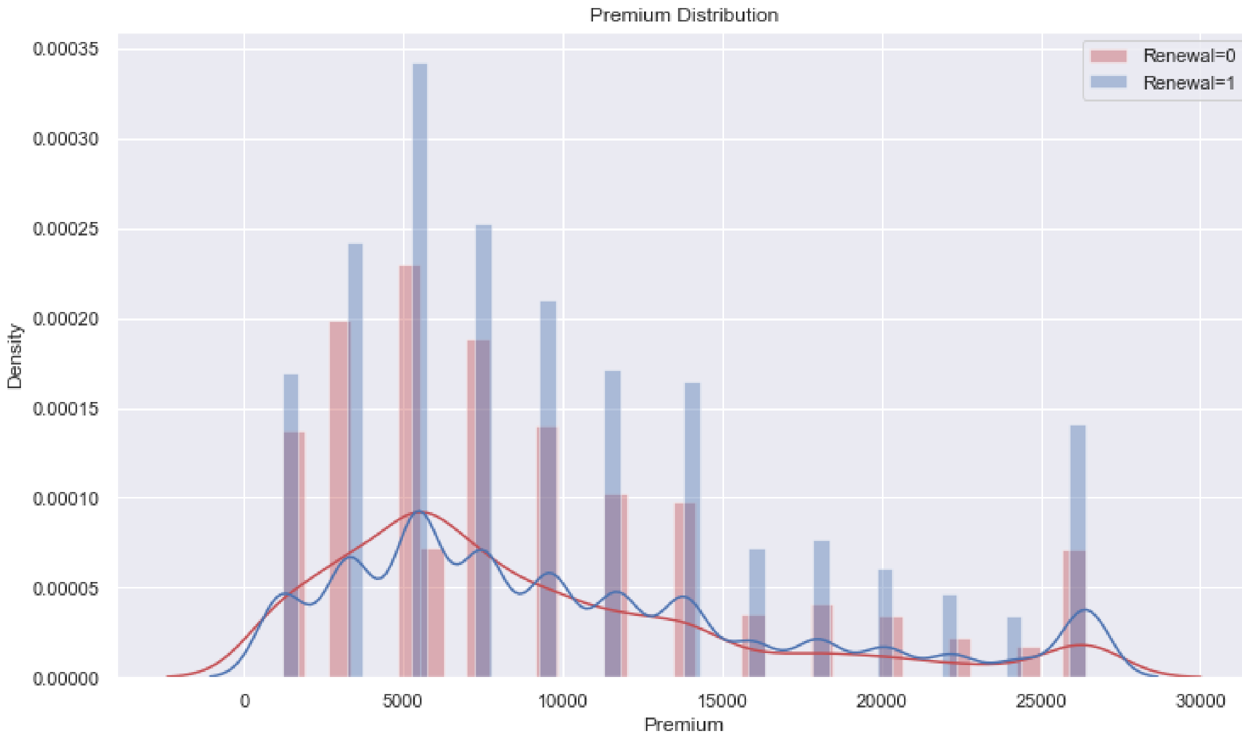


- There are a few odd peaks in the renewal distribution, all which favor the renewals. These could indicate, as mentioned earlier, starts of new policies by people of particular risk scores. As it looks, there are three "moments" in the scores where these peaks occur, almost all around the non-decimal amount.

EDA Premium histogram after removal of outliers

In [466…

```
plt.figure(figsize=(12,7))
sns.distplot(df2[df2["Renewal"] == 0]['Premium'], color = 'r',label='Renewal=0')
sns.distplot(df2[df2["Renewal"] == 1]['Premium'], color = 'b',label='Renewal=1')
plt.legend()
plt.title("Premium Distribution");
```



- Regardless of premium amount, the renewals were more than non-renewals.
- The renewals appear to decrease as the premiums rise, but this could also reflect rising incomes (fewer customers) as well.
- There appear to be many "squiggles" in the renewal distribution line which seem to fall after specific premium-points, and rise right before these points. The non-renewal line appears smoother. It follows that as the renewals are rising, the non-renewal trend line is riding on top of the crest of the blue line (as a "ceiling"). When renewals drop, the red line is riding at the troughs of the blue line (as a "support"). So, when the blue line dips, it "adds" to the red line, propping it in the upward trend. When the blue line rises, it meets the red line, stabilizing it and keeping it from rising further.

EDA after Pre-processing

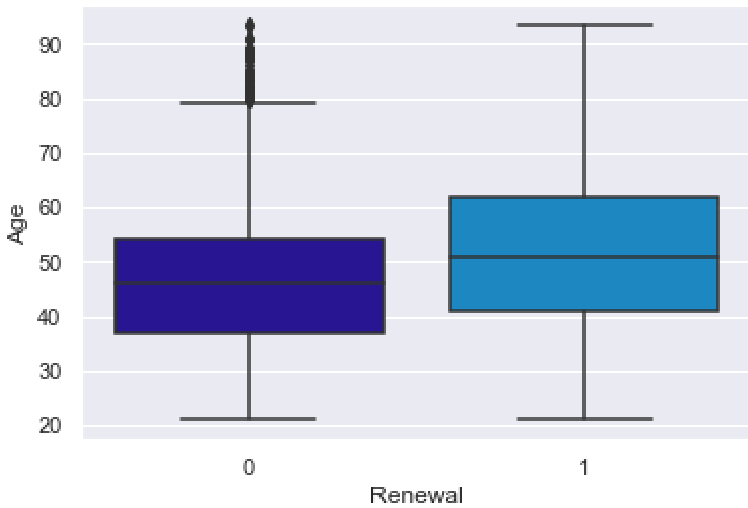
Age vs. Renewal

In [467…

```
#Re-check Renewal vs. Age
sns.boxplot(df2['Renewal'], df2['Age'])
```

Out[467…

```
<AxesSubplot:xlabel='Renewal', ylabel='Age'>
```

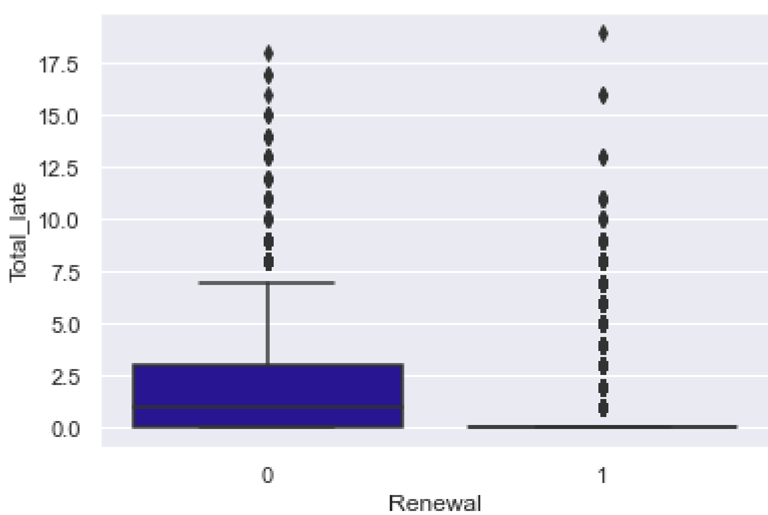


- There was no significant change in the relationship. The ages of those that renewed is slightly higher than those that did not renew. Various factors could be at play here: older clientele could be less willing to change their policies for a number of reasons; younger clientele could be more comfortable comparing and shopping around for different products elsewhere; the younger clientele could be defaulting more as a sign of financial instability, etc.

Total_late vs. Renewal

In [468…
```
sns.boxplot(df2['Renewal'], df2['Total_late'])
```

Out[468…
```
<AxesSubplot:xlabel='Renewal', ylabel='Total_late'>
```



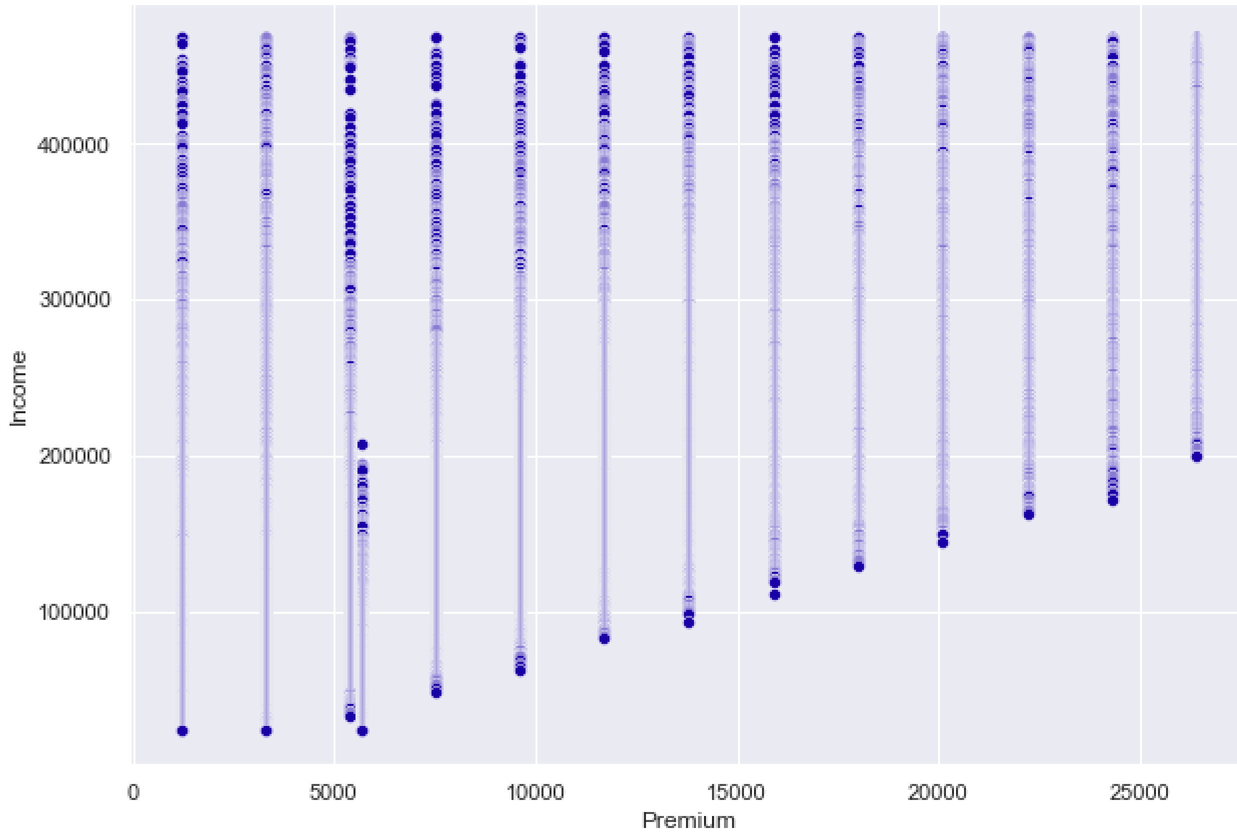
- There was no change from the previous plot

Premium vs. Income

In [469…
```
plt.figure(figsize=(10,7))
sns.scatterplot(df2['Premium'], df2['Income'])
```

```
<AxesSubplot:xlabel='Premium', ylabel='Income'>
```

Out[469...



- As the premiums rise, there is a decrease in the number of customers participating with lower incomes.

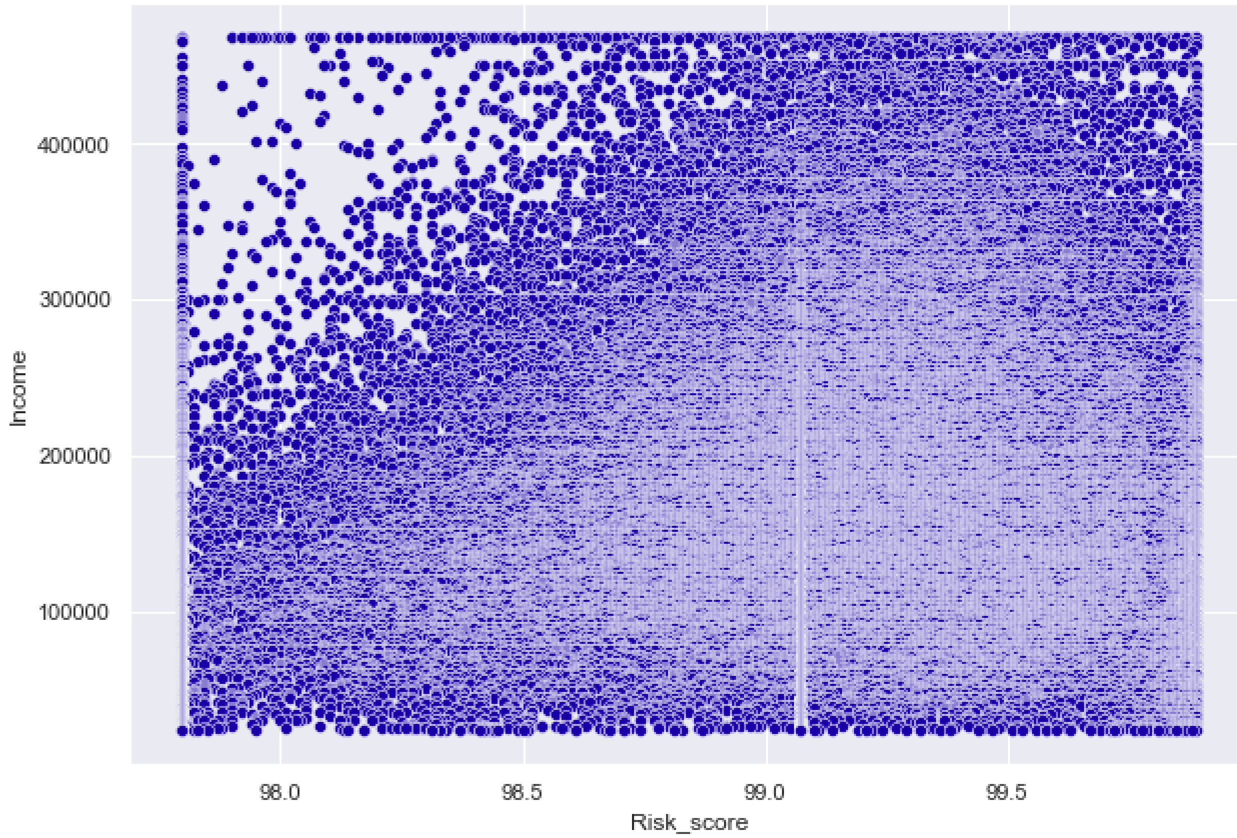
Risk score vs. Income

In [470...

```
plt.figure(figsize=(10,7))
sns.scatterplot(df2['Risk_score'], df2['Income'])
```

Out[470...

```
<AxesSubplot:xlabel='Risk_score', ylabel='Income'>
```



- Not helpful at all

Variable Transformation

Encoding (Get_dummies)

In [471…

```
# Use df (original) dataset to get encoded values for categoricals
df_dummy = pd.get_dummies(df,drop_first=True)
df_dummy.head()
```

Out[471…

	Perc_premium_paid	Age	Income	3-6_late	6-12_late	12_more_late	Total_late	Marital_status	Veh_Owı
0	0.01	52	90262600	0	0	0	0	0	
1	0.00	68	53821900	0	0	0	0	0	
2	0.16	44	46803140	0	0	0	0	1	
3	0.47	44	32175090	1	0	0	1	0	
4	0.04	55	25051240	0	0	0	0	1	

X and y

In [472…

```
X = df_dummy.drop(['Renewal', 'Total_late'], axis=1)
y = df_dummy['Renewal']
```

In [473…

```
# dataframe with numerical column only
num_feature_set = X.copy()
num_feature_set = add_constant(num_feature_set)
num_feature_set = num_feature_set.astype(float)
```

In [474…

```
X.head()
```

Out[474…

	Perc_premium_paid	Age	Income	3-6_late	6-12_late	12_more_late	Marital_status	Veh_Owned	No_of
0	0.01	52	90262600	0	0	0	0	2	
1	0.00	68	53821900	0	0	0	0	3	
2	0.16	44	46803140	0	0	0	1	3	
3	0.47	44	32175090	1	0	0	0	3	
4	0.04	55	25051240	0	0	0	1	3	

Looking forward...

- The dataset is unbalanced, so SMOTE or a similar (oversample, undersample) technique will need to be utilized to balance the class.
- Models (Supervised Classification) to consider using would include Logistic Regression, Random Forest, SVM, KNN, and Gradient Boosting XGBoost.
- We will utilize feature selection to decrease the number of features (dimensionality reducation) to potentially get better results from the models.
- We will also use hyperparameter tuning to help maximize the models' predictions, including GridSearch.
- VIF can be used to recognize multicollinearity in the variables; from this we will be able to remove collinear variables from the analysis by observing p-values.

Using VIF to remove collinearities

In [ ]:

In [475…

```
vif_series1 = pd.Series([variance_inflation_factor(num_feature_set.values,i) for i in r
print('Series before feature selection: \n\n{}\n'.format(vif_series1))
```

```
Series before feature selection:

const                21858.624615
Perc_premium_paid        1.206686
Age                      1.158976
Income                   1.103245
```

```
3-6_late                 1.163646
6-12_late                1.133267
12_more_late             1.159077
Marital_status           1.000225
Veh_Owned                1.000262
No_of_dep                1.000408
Accomodation             1.000160
Risk_score               1.162929
Premiums_paid            1.228464
Premium                  1.200426
Source_B                 1.113158
Source_C                 1.145834
Source_D                 1.115406
Source_E                 1.011750
Res_type_Urban           1.001197
dtype: float64
```

Split X and y train test

In [476…

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=142,
print(X_train.shape, X_test.shape)
```

```
(55897, 18) (23956, 18)
```

Define functions

In [477…

```
##  Function to calculate different metric scores of the model - Accuracy, Recall and P
def get_metrics_score1(model,train,test,train_y,test_y,flag=True):
    '''
    model : classifier to predict values of X

    '''
    # defining an empty list to store train and test results

    score_list=[]

    pred_train = model.predict(train)
    pred_test = model.predict(test)

    pred_train = np.round(pred_train)
    pred_test = np.round(pred_test)

    train_acc = accuracy_score(pred_train,train_y)
    test_acc = accuracy_score(pred_test,test_y)

    train_recall = recall_score(train_y,pred_train)
    test_recall = recall_score(test_y,pred_test)

    train_precision = precision_score(train_y,pred_train)
    test_precision = precision_score(test_y,pred_test)

    score_list.extend((train_acc,test_acc,train_recall,test_recall,train_precision,test

   # If the flag is set to True then only the following print statements will be dispay
```

```python
    if flag == True:
        print("Accuracy on training set : ",accuracy_score(pred_train,train_y))
        print("Accuracy on test set : ",accuracy_score(pred_test,test_y))
        print("Recall on training set : ",recall_score(train_y,pred_train))
        print("Recall on test set : ",recall_score(test_y,pred_test))
        print("Precision on training set : ",precision_score(train_y,pred_train))
        print("Precision on test set : ",precision_score(test_y,pred_test))
    return score_list # returning the list with train and test scores
```

In [478…

```python
##  Function to calculate different metric scores of the model - Accuracy, Recall and P
def get_metrics_score2(model,train,test,train_y,test_y,flag=True):
    '''
    model : classifier to predict values of X

    '''
    # defining an empty list to store train and test results

    score_list=[]

    pred_train = model.predict(train)
    pred_test = model.predict(test)

    train_acc = accuracy_score(pred_train,train_y)
    test_acc = accuracy_score(pred_test,test_y)

    train_recall = recall_score(train_y,pred_train)
    test_recall = recall_score(test_y,pred_test)

    train_precision = precision_score(train_y,pred_train)
    test_precision = precision_score(test_y,pred_test)

    score_list.extend((train_acc,test_acc,train_recall,test_recall,train_precision,test_

    # If the flag is set to True then only the following print statements will be dispa
    if flag == True:
        print("Accuracy on training set : ",accuracy_score(pred_train,train_y))
        print("Accuracy on test set : ",accuracy_score(pred_test,test_y))
        print("Recall on training set : ",recall_score(train_y,pred_train))
        print("Recall on test set : ",recall_score(test_y,pred_test))
        print("Precision on training set : ",precision_score(train_y,pred_train))
        print("Precision on test set : ",precision_score(test_y,pred_test))
        print("ROC-AUC Score on training set:",metrics.roc_auc_score(train_y,pred_train
        print("ROC-AUC Score on test set:",metrics.roc_auc_score(test_y,pred_test))

    return score_list # returning the list with train and test scores
```

In [479…

```python
## Function to create confusion matrix
def make_confusion_matrix(model, y_actual, labels=[1, 0]):
    """
    model: classifier to predict values of X
    y_actual: ground truth

    """
    y_predict = model.predict(X_test)
    cm = metrics.confusion_matrix(y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(
```

```
        cm,
        index=[i for i in ["Actual - No", "Actual - Yes"]],
        columns=[i for i in ["Predicted - No", "Predicted - Yes"]],
    )
    group_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cm.flatten() / np.sum(cm)
    labels = [f"{v1}\n{v2}" for v1, v2 in zip(group_counts, group_percentages)]
    labels = np.asarray(labels).reshape(2, 2)
    plt.figure(figsize=(10, 7))
    sns.heatmap(df_cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Using Logit to remove features with p>0.5

In [480…

```
logit = sm.Logit(y_train, X_train.astype(float))
lg = logit.fit()

# logreg = LogisticRegression(solver='newton-cg',max_iter=1000,verbose=True,n_jobs=-1,r
# lg = logreg.fit(X_train, y_train)

print(lg.summary())

print('')
# Let's check model performances for this model
scores_LR = get_metrics_score1(lg,X_train,X_test,y_train,y_test, flag=True)
```

```
Optimization terminated successfully.
         Current function value: 0.186594
         Iterations 8
                           Logit Regression Results
==============================================================================
Dep. Variable:                Renewal   No. Observations:                55897
Model:                          Logit   Df Residuals:                    55879
Method:                           MLE   Df Model:                           17
Date:                Sun, 29 Aug 2021   Pseudo R-squ.:                  0.2028
Time:                        22:02:33   Log-Likelihood:                -10430.
converged:                       True   LL-Null:                       -13083.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                      coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Perc_premium_paid  -1.9411      0.058    -33.602      0.000      -2.054      -1.828
Age                 0.0154      0.002      9.747      0.000       0.012       0.019
Income           3.361e-07   1.85e-07      1.813      0.070   -2.72e-08    6.99e-07
3-6_late           -0.4372      0.018    -23.687      0.000      -0.473      -0.401
6-12_late          -0.6659      0.029    -22.773      0.000      -0.723      -0.609
12_more_late       -0.5920      0.039    -15.083      0.000      -0.669      -0.515
Marital_status      0.0440      0.038      1.144      0.253      -0.031       0.119
Veh_Owned          -0.0098      0.024     -0.415      0.678      -0.056       0.036
No_of_dep          -0.0457      0.017     -2.649      0.008      -0.079      -0.012
Accomodation       -0.0209      0.038     -0.544      0.587      -0.096       0.054
Risk_score          0.0367      0.001     30.417      0.000       0.034       0.039
Premiums_paid      -0.0344      0.004     -8.876      0.000      -0.042      -0.027
Premium          5.033e-06   3.21e-06      1.566      0.117   -1.27e-06    1.13e-05
Source_B           -0.0356      0.050     -0.708      0.479      -0.134       0.063
Source_C           -0.0894      0.054     -1.649      0.099      -0.196       0.017
```

```
Source_D             -0.1310      0.063     -2.065      0.039      -0.255      -0.007
Source_E             -0.2030      0.196     -1.034      0.301      -0.588       0.182
Res_type_Urban        0.0240      0.039      0.612      0.540      -0.053       0.101
=====================================================================================

Accuracy on training set :  0.9393348480240442
Accuracy on test set :  0.939889797962932
Recall on training set :  0.9935684568113287
Recall on test set :  0.9942556886494189
Precision on training set :  0.944588587498866
Precision on test set :  0.9445408012183256
```

In [481…

```python
# Veh_Owned has the highest p-value
X_train1 = X_train.drop(['Veh_Owned'], axis = 1)
X_test1 = X_test.drop(['Veh_Owned'], axis = 1)

logit1 = sm.Logit(y_train, X_train1.astype(float))
lg1 = logit1.fit()

## Let's check model performances for this model
get_metrics_score1(lg1,X_train1.astype(float),X_test1.astype(float),y_train,y_test,flag
print('')
print(lg1.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.186595
         Iterations 8
Accuracy on training set :  0.9393348480240442
Accuracy on test set :  0.9399315411587911
Recall on training set :  0.9935684568113287
Recall on test set :  0.9942556886494189
Precision on training set :  0.944588587498866
Precision on test set :  0.9445807597935527

                           Logit Regression Results
==============================================================================
Dep. Variable:                Renewal   No. Observations:                55897
Model:                          Logit   Df Residuals:                    55880
Method:                           MLE   Df Model:                           16
Date:                Sun, 29 Aug 2021   Pseudo R-squ.:                  0.2028
Time:                        22:02:33   Log-Likelihood:                -10430.
converged:                       True   LL-Null:                       -13083.
Covariance Type:            nonrobust   LLR p-value:                     0.000
=====================================================================================
                        coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------
Perc_premium_paid    -1.9411      0.058    -33.602      0.000      -2.054      -1.828
Age                   0.0154      0.002      9.744      0.000       0.012       0.019
Income             3.363e-07   1.85e-07      1.815      0.070   -2.69e-08    6.99e-07
3-6_late             -0.4371      0.018    -23.685      0.000      -0.473      -0.401
6-12_late            -0.6658      0.029    -22.772      0.000      -0.723      -0.609
12_more_late         -0.5922      0.039    -15.088      0.000      -0.669      -0.515
Marital_status        0.0440      0.038      1.145      0.252      -0.031       0.119
No_of_dep            -0.0457      0.017     -2.650      0.008      -0.080      -0.012
Accomodation         -0.0208      0.038     -0.541      0.589      -0.096       0.055
Risk_score            0.0365      0.001     32.887      0.000       0.034       0.039
Premiums_paid        -0.0344      0.004     -8.879      0.000      -0.042      -0.027
Premium            5.038e-06   3.21e-06      1.567      0.117   -1.26e-06    1.13e-05
Source_B             -0.0356      0.050     -0.709      0.478      -0.134       0.063
```

```
Source_C              -0.0895      0.054     -1.651      0.099     -0.196      0.017
Source_D              -0.1310      0.063     -2.065      0.039     -0.255     -0.007
Source_E              -0.2032      0.196     -1.035      0.301     -0.588      0.182
Res_type_Urban         0.0241      0.039      0.614      0.539     -0.053      0.101
=====================================================================================
```

In [482…

```python
# Accomodation was the next with the highest p-value
X_train2 = X_train1.drop(['Accomodation'], axis = 1)
X_test2 = X_test1.drop(['Accomodation'], axis = 1)

logit2 = sm.Logit(y_train, X_train2.astype(float))
lg2 = logit2.fit()

## Let's check model performances for this model
get_metrics_score1(lg2,X_train2.astype(float),X_test2.astype(float),y_train,y_test,flag
print('')
print(lg2.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.186598
         Iterations 8
Accuracy on training set :  0.9393885181673435
Accuracy on test set :  0.9400150275505093
Recall on training set :  0.9936447956028857
Recall on test set :  0.9943002181947722
Precision on training set :  0.9445754716981132
Precision on test set :  0.9446230645570691

                           Logit Regression Results
==============================================================================
Dep. Variable:                Renewal   No. Observations:                55897
Model:                          Logit   Df Residuals:                    55881
Method:                           MLE   Df Model:                           15
Date:                Sun, 29 Aug 2021   Pseudo R-squ.:                  0.2028
Time:                        22:02:34   Log-Likelihood:                -10430.
converged:                       True   LL-Null:                       -13083.
Covariance Type:            nonrobust   LLR p-value:                     0.000
=====================================================================================
                        coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------
Perc_premium_paid    -1.9411      0.058    -33.600      0.000     -2.054     -1.828
Age                   0.0154      0.002      9.746      0.000      0.012      0.019
Income             3.363e-07   1.85e-07      1.814      0.070   -2.7e-08      7e-07
3-6_late             -0.4371      0.018    -23.686      0.000     -0.473     -0.401
6-12_late            -0.6658      0.029    -22.774      0.000     -0.723     -0.608
12_more_late         -0.5922      0.039    -15.090      0.000     -0.669     -0.515
Marital_status        0.0441      0.038      1.147      0.251     -0.031      0.119
No_of_dep            -0.0457      0.017     -2.651      0.008     -0.080     -0.012
Risk_score            0.0364      0.001     33.335      0.000      0.034      0.038
Premiums_paid        -0.0344      0.004     -8.878      0.000     -0.042     -0.027
Premium            5.036e-06   3.21e-06      1.567      0.117  -1.26e-06    1.13e-05
Source_B             -0.0357      0.050     -0.711      0.477     -0.134      0.063
Source_C             -0.0894      0.054     -1.650      0.099     -0.196      0.017
Source_D             -0.1309      0.063     -2.063      0.039     -0.255     -0.007
Source_E             -0.2029      0.196     -1.034      0.301     -0.588      0.182
Res_type_Urban        0.0240      0.039      0.613      0.540     -0.053      0.101
=====================================================================================
```

Will not drop Res_type*Urban, Source*(k), or Premium as these are

encoded values and are subsets of features

In [483…

```
# X_train3 = X_train2.drop(['Res_type_Urban'], axis = 1)
# X_test3 = X_test2.drop(['Res_type_Urban'], axis = 1)

# logit3 = sm.Logit(y_train, X_train3.astype(float))
# lg3 = logit3.fit()

# ## Let's check model performances for this model
# get_metrics_score1(lg3,X_train3.astype(float),X_test3.astype(float),y_train,y_test,fl
# print('')
# print(lg3.summary())
```

In [484…

```
# X_train4 = X_train3.drop(['Source_B'], axis = 1)
# X_test4 = X_test3.drop(['Source_B'], axis = 1)

# logit4 = sm.Logit(y_train, X_train4.astype(float))
# lg4 = logit4.fit()

# ## Let's check model performances for this model
# get_metrics_score1(lg4,X_train4.astype(float),X_test4.astype(float),y_train,y_test,fl
# print('')
# print(lg4.summary())
```

In [485…

```
# X_train5 = X_train4.drop(['Source_E'], axis = 1)
# X_test5 = X_test4.drop(['Source_E'], axis = 1)

# logit5 = sm.Logit(y_train, X_train5.astype(float))
# lg5 = logit5.fit()

# ## Let's check model performances for this model
# get_metrics_score1(lg5,X_train5.astype(float),X_test5.astype(float),y_train,y_test,fl
# print('')
# print(lg5.summary())
```

In [486…

```
X_train9 = X_train2.drop(['Marital_status'], axis = 1)
X_test9 = X_test2.drop(['Marital_status'], axis = 1)

logit9 = sm.Logit(y_train, X_train9.astype(float))
lg9 = logit9.fit()

## Let's check model performances for this model
get_metrics_score1(lg9,X_train9.astype(float),X_test9.astype(float),y_train,y_test,flag
print('')
print(lg3.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.186609
         Iterations 8
Accuracy on training set :  0.9392453977852121
Accuracy on test set :  0.939889797962932
Recall on training set :  0.9936257109049964
Recall on test set :  0.9942556886494189
Precision on training set :  0.9444545223669412
Precision on test set :  0.9445408012183256
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                Renewal   No. Observations:                55897
Model:                          Logit   Df Residuals:                    55882
Method:                           MLE   Df Model:                           14
Date:                Sun, 29 Aug 2021   Pseudo R-squ.:                  0.2027
Time:                        22:02:35   Log-Likelihood:                -10431.
converged:                       True   LL-Null:                       -13083.
Covariance Type:            nonrobust   LLR p-value:                     0.000
=====================================================================================
                        coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------
Perc_premium_paid    -1.9408      0.058    -33.596      0.000      -2.054      -1.828
Age                   0.0154      0.002      9.750      0.000       0.012       0.019
Income             3.376e-07   1.86e-07      1.820      0.069    -2.6e-08    7.01e-07
3-6_late             -0.4370      0.018    -23.687      0.000      -0.473      -0.401
6-12_late            -0.6659      0.029    -22.785      0.000      -0.723      -0.609
12_more_late         -0.5925      0.039    -15.101      0.000      -0.669      -0.516
No_of_dep            -0.0456      0.017     -2.646      0.008      -0.079      -0.012
Risk_score            0.0366      0.001     34.022      0.000       0.034       0.039
Premiums_paid        -0.0344      0.004     -8.886      0.000      -0.042      -0.027
Premium            5.001e-06   3.22e-06      1.555      0.120    -1.3e-06    1.13e-05
Source_B             -0.0349      0.050     -0.695      0.487      -0.133       0.064
Source_C             -0.0886      0.054     -1.634      0.102      -0.195       0.018
Source_D             -0.1306      0.063     -2.058      0.040      -0.255      -0.006
Source_E             -0.2017      0.196     -1.028      0.304      -0.586       0.183
Res_type_Urban        0.0242      0.039      0.617      0.537      -0.053       0.101
=====================================================================================
```

In [487…

```
# X_train7 = X_train6.drop(['Source_C'], axis = 1)
# X_test7 = X_test6.drop(['Source_C'], axis = 1)

# logit7 = sm.Logit(y_train, X_train7.astype(float))
# lg7 = logit7.fit()

# ## Let's check model performances for this model
# get_metrics_score1(lg7,X_train7.astype(float),X_test7.astype(float),y_train,y_test,fl
# print('')
# print(lg7.summary())
```

In [488…

```
# X_train8= X_train7.drop(['Premium'], axis = 1)
# X_test8 = X_test7.drop(['Premium'], axis = 1)

# logit8 = sm.Logit(y_train, X_train8.astype(float))
# lg8 = logit8.fit()

# ## Let's check model performances for this model
# get_metrics_score1(lg8,X_train8.astype(float),X_test8.astype(float),y_train,y_test,fl
# print('')
# print(lg8.summary())
```

In [489…

```
# X_train9= X_train8.drop(['Source_D'], axis = 1)
# X_test9 = X_test8.drop(['Source_D'], axis = 1)

# logit9 = sm.Logit(y_train, X_train9.astype(float))
# lg9 = logit9.fit()
```

```
# ## Let's check model performances for this model
# get_metrics_score1(lg9,X_train9.astype(float),X_test9.astype(float),y_train,y_test,fl
# print('')
# print(lg9.summary())
```

Remaining features with X_train9

In [490…

```
X_train9.columns
```

Out[490…

```
Index(['Perc_premium_paid', 'Age', 'Income', '3-6_late', '6-12_late',
       '12_more_late', 'No_of_dep', 'Risk_score', 'Premiums_paid', 'Premium',
       'Source_B', 'Source_C', 'Source_D', 'Source_E', 'Res_type_Urban'],
      dtype='object')
```

In [491…

```
X_test9.columns
```

Out[491…

```
Index(['Perc_premium_paid', 'Age', 'Income', '3-6_late', '6-12_late',
       '12_more_late', 'No_of_dep', 'Risk_score', 'Premiums_paid', 'Premium',
       'Source_B', 'Source_C', 'Source_D', 'Source_E', 'Res_type_Urban'],
      dtype='object')
```

LR AUC-ROC curve

Though this model was not used, it returned a good AUC and overall Recall score, initially

In [492…

```
#Logistic Regression AUC-ROC
logit_roc_auc_train = roc_auc_score(y_train, lg9.predict(X_train9))
fpr, tpr, thresholds = roc_curve(y_train, lg9.predict(X_train9))
plt.figure(figsize=(7,5))
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

In [493...

```
##  Function to calculate different metric scores of the model - Accuracy, Recall and P
def get_metrics_score(model,flag=True):
    '''
    model : classifier to predict values of X

    '''
    # defining an empty list to store train and test results
    score_list=[]

    pred_train = model.predict(X_train9)
    pred_test = model.predict(X_test9)

    train_acc = model.score(X_train9,y_train)
    test_acc = model.score(X_test9,y_test)

    train_recall = metrics.recall_score(y_train,pred_train)
    test_recall = metrics.recall_score(y_test,pred_test)

    train_precision = metrics.precision_score(y_train,pred_train)
    test_precision = metrics.precision_score(y_test,pred_test)

    score_list.extend((train_acc,test_acc,train_recall,test_recall,train_precision,test

    # If the flag is set to True then only the following print statements will be dispa
    if flag == True:
        print("Accuracy on training set : ",model.score(X_train9,y_train))
        print("Accuracy on test set : ",model.score(X_test9,y_test))
        print("Recall on training set : ",metrics.recall_score(y_train,pred_train))
        print("Recall on test set : ",metrics.recall_score(y_test,pred_test))
        print("Precision on training set : ",metrics.precision_score(y_train,pred_train
        print("Precision on test set : ",metrics.precision_score(y_test,pred_test))

    return score_list # returning the list with train and test scores
```

In [494...

```
## Function to create confusion matrix
def make_confusion_matrix(model, y_actual, labels=[1, 0]):
```

```
    """
    model: classifier to predict values of X
    y_actual: ground truth

    """
    y_predict = model.predict(X_test9)
    cm = metrics.confusion_matrix(y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(
        cm,
        index=[i for i in ["Actual - No", "Actual - Yes"]],
        columns=[i for i in ["Predicted - No", "Predicted - Yes"]],
    )
    group_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cm.flatten() / np.sum(cm)
    labels = [f"{v1}\n{v2}" for v1, v2 in zip(group_counts, group_percentages)]
    labels = np.asarray(labels).reshape(2, 2)
    plt.figure(figsize=(10, 7))
    sns.heatmap(df_cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Basic models (RF and XGB)

Oversampling using SMOTE on Train only

In [495…

```
print("Before UpSampling, counts of label '1': {}".format(sum(y_train==1)))
print("Before UpSampling, counts of label '0': {} \n".format(sum(y_train==0)))

sm = SMOTE(sampling_strategy = 1 ,k_neighbors = 5, random_state=1)   #Synthetic Minorit
X_train_res, y_train_res = sm.fit_resample(X_train9, y_train.ravel())


print("After UpSampling, counts of label '1': {}".format(sum(y_train_res==1)))
print("After UpSampling, counts of label '0': {} \n".format(sum(y_train_res==0)))



print('After UpSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After UpSampling, the shape of train_y: {} \n'.format(y_train_res.shape))
```

```
Before UpSampling, counts of label '1': 52398
Before UpSampling, counts of label '0': 3499

After UpSampling, counts of label '1': 52398
After UpSampling, counts of label '0': 52398

After UpSampling, the shape of train_X: (104796, 15)
After UpSampling, the shape of train_y: (104796,)
```

- The datasets are balanced out

Creating Models with SMOTE (Random Forest

and XGB)

Standard Scaler

In [496…

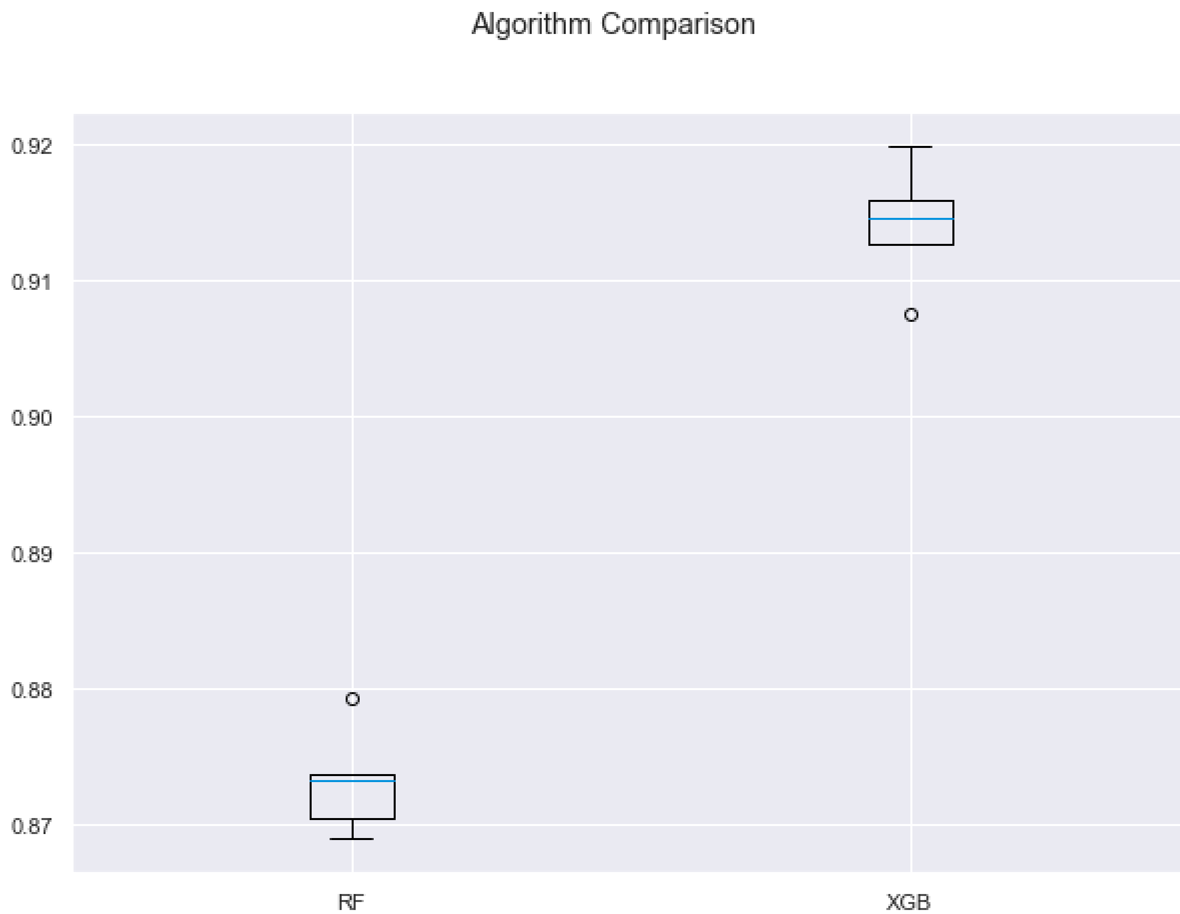
```
models = []  # Empty list to store all the models

# Appending pipelines for each model into the list

models.append(
    (
        "RF",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("random_forest", RandomForestClassifier(random_state=1)),
            ]
        ),
    )
)



models.append(
    (
        "XGB",
        Pipeline(
            steps=[
                ("scaler", StandardScaler()),
                ("xgboost", XGBClassifier(random_state=1,eval_metric='logloss')),
            ]
        ),
    )
)

results = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models

# Loop through all models to get the mean cross validated score
for name, model in models:
    scoring = "recall"
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  # Setting number of splits equal to 5
    cv_result = cross_val_score(
        estimator=model, X=X_train_res, y=y_train_res, scoring=scoring, cv=kfold
    )
    results.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean() * 100))
```

```
RF: 87.31248593151406
XGB: 91.40807332585928
```

In [497…

```
# Plotting boxplots for CV scores of all models defined above
fig = plt.figure(figsize=(10, 7))
```

```
fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results)
ax.set_xticklabels(names)

plt.show()
```



- XGB performed about 4% better than RF

RF Model

In [498…

```
#Fitting the RF model
rf_estimator2 = RandomForestClassifier(random_state=1)
rf_estimator2.fit(X_train_res,y_train_res)

#Calculating different metrics
get_metrics_score(rf_estimator2)

#Creating confusion matrix
# make_confusion_matrix(rf_estimator2,y_test)
```

```
Accuracy on training set :  0.9999821099522336
Accuracy on test set :  0.8473451327433629
Recall on training set :  1.0
Recall on test set :  0.8778109275504297
Precision on training set :  0.9999809156663295
Precision on test set :  0.9557354794919034
```

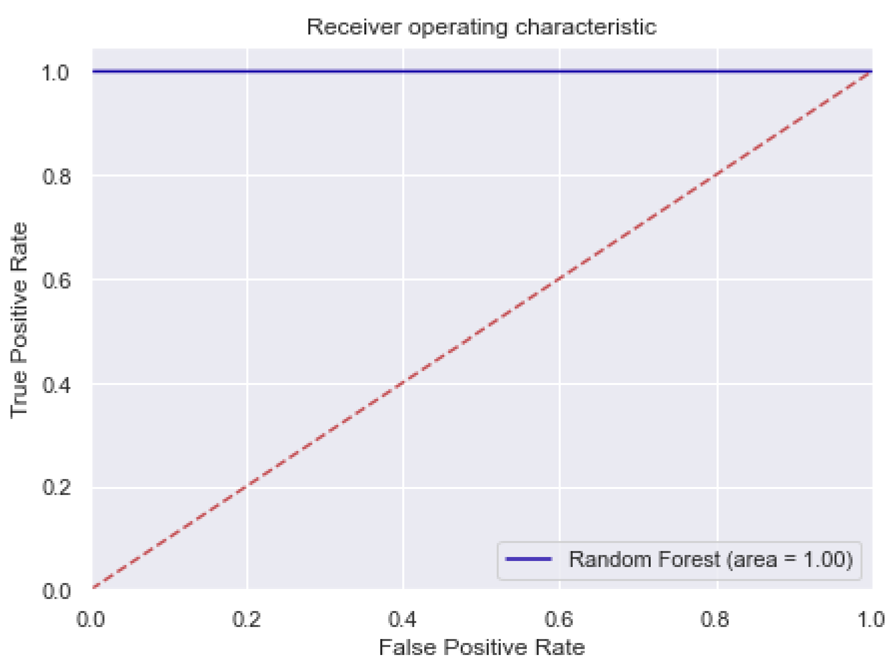
Out[498…

```
[0.9999821099522336,
 0.8473451327433629,
 1.0,
 0.8778109275504297,
 0.9999809156663295,
 0.9557354794919034]
```

- The model is overfitting where Recall train is 1.0 and Recall train is 0.88

In [499…

```python
# ROC-AUC on Random Forest Train
logit_roc_auc_train = roc_auc_score(y_train_res, rf_estimator2.predict(X_train_res))
fpr, tpr, thresholds = roc_curve(y_train_res, rf_estimator2.predict(X_train_res))
plt.figure(figsize=(7,5))
plt.plot(fpr, tpr, label='Random Forest (area = %0.2f)' % logit_roc_auc_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



- Interesting that the Train AUC = 1.0

In [500…

```python
y_train_res.shape
```

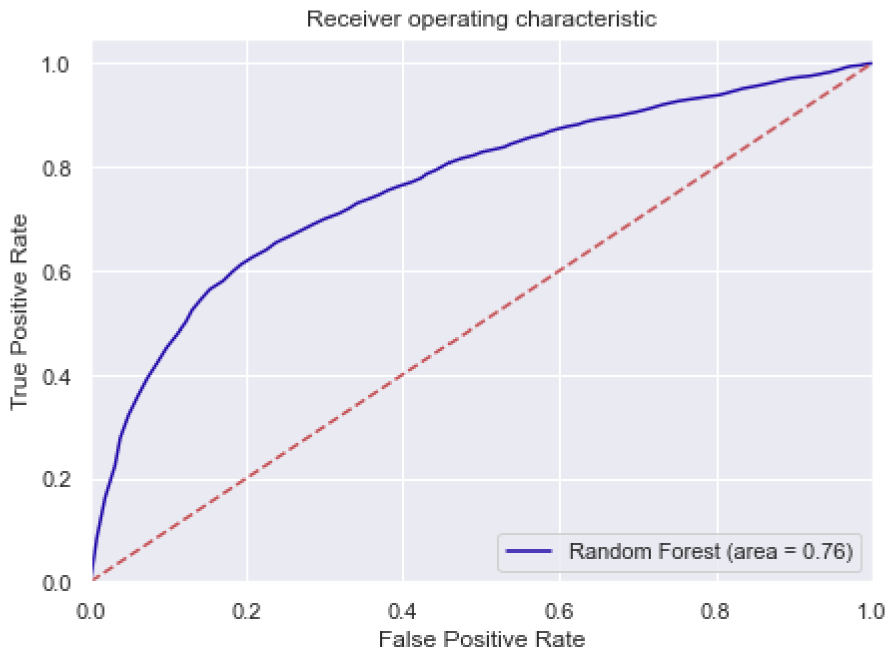
Out[500…

```
(104796,)
```

In [501…

```python
# ROC-AUC on Random Forest Test
logit_roc_auc_train = roc_auc_score(y_test, rf_estimator2.predict_proba(X_test9)[:,1])
fpr, tpr, thresholds = roc_curve(y_test, rf_estimator2.predict_proba(X_test9)[:,1])
plt.figure(figsize=(7,5))
plt.plot(fpr, tpr, label='Random Forest (area = %0.2f)' % logit_roc_auc_train)
```

```
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



- Test AUC = .76

XGBoost Model

In [502…

```
#Fitting the XGB model
xgb_classifier2 = XGBClassifier(random_state=1)
xgb_classifier2.fit(X_train_res,y_train_res)

#Calculating different metrics
get_metrics_score(xgb_classifier2)

#Creating confusion matrix
# make_confusion_matrix(xgb_classifier2,y_test)
```

```
[22:04:33] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lea
rner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metr
ic if you'd like to restore the old behavior.
Accuracy on training set :  0.9026781401506342
Accuracy on test set :  0.8753548171648021
Recall on training set :  0.9303408527043017
Recall on test set :  0.9123213251992697
Precision on training set :  0.9645811072856069
Precision on test set :  0.952708672401767
```

Out[502…

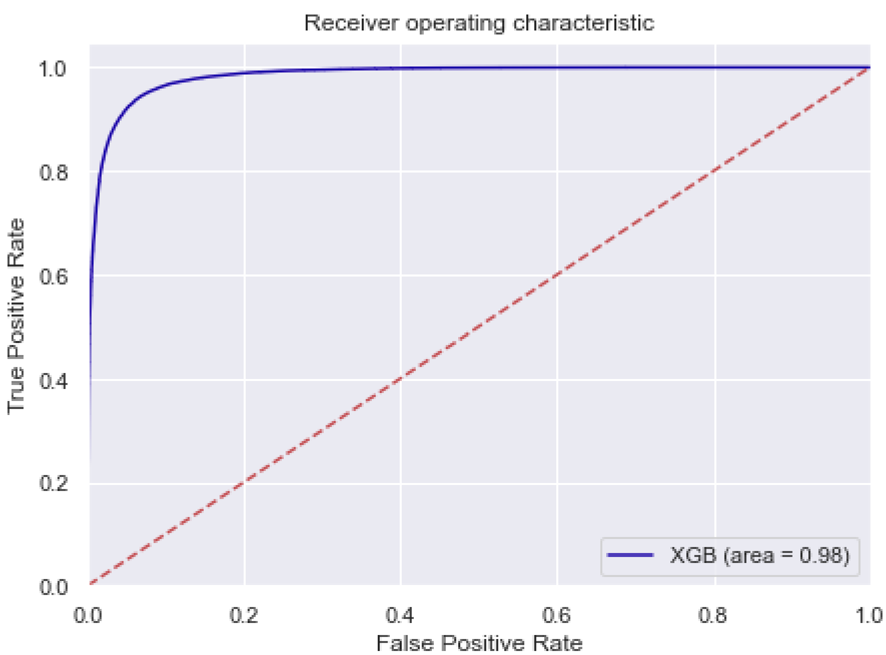
```
[0.9026781401506342,
 0.8753548171648021,
 0.9303408527043017,
```

```
0.9123213251992697,
0.9645811072856069,
0.952708672401767]
```

- Pretty decent recall train/test scores 93% and 91%
- Precision and accuracy are not bad either

In [503…

```python
# ROC-AUC on XGB Train
logit_roc_auc_train = roc_auc_score(y_train_res, xgb_classifier2.predict_proba(X_train_
fpr, tpr, thresholds = roc_curve(y_train_res, xgb_classifier2.predict_proba(X_train_res
plt.figure(figsize=(7,5))
plt.plot(fpr, tpr, label='XGB (area = %0.2f)' % logit_roc_auc_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

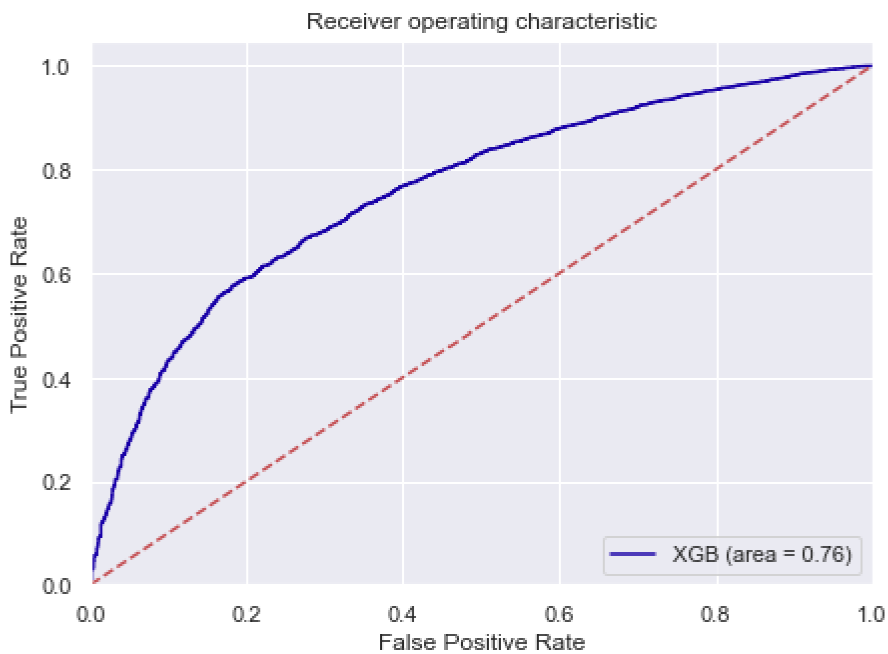


- Train AUC = .98
- Good curve

In [504…

```python
# ROC-AUC on XGB Test
logit_roc_auc_train = roc_auc_score(y_test, xgb_classifier2.predict_proba(X_test9)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, xgb_classifier2.predict_proba(X_test9)[:,1])
plt.figure(figsize=(7,5))
plt.plot(fpr, tpr, label='XGB (area = %0.2f)' % logit_roc_auc_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
```

```
plt.legend(loc="lower right")
plt.show()
```



- Test AUC = .76

RF Hypertuned

In [505…

```
%%time

# Creating pipeline
pipe = make_pipeline(StandardScaler(), RandomForestClassifier(random_state=1))

# Parameter grid to pass in RandomSearchCV
param_grid = {
    "randomforestclassifier__n_estimators": [100,150,250],
    "randomforestclassifier__min_samples_leaf": np.arange(1, 6),
    "randomforestclassifier__max_features": [np.arange(0.3, 0.6, 0.1),'sqrt','log2'],
    "randomforestclassifier__max_samples": np.arange(0.2, 0.6, 0.1),
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=pipe, param_distributions=param_grid, n_jo

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_res,y_train_res)

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,ran
```

```
Best parameters are {'randomforestclassifier__n_estimators': 250, 'randomforestclassifie
r__min_samples_leaf': 1, 'randomforestclassifier__max_samples': 0.5000000000000001, 'ran
domforestclassifier__max_features': 'sqrt'} with CV score=0.8652809472088488:
Wall time: 9min 35s
```

In [506…

```
# Creating new pipeline with best parameters
rf_tuned2 = make_pipeline(
    StandardScaler(),
    RandomForestClassifier(
        n_estimators=250,
        max_features='sqrt',
        random_state=1,
        max_samples=0.5000000000000001,
        min_samples_leaf=1
    ),
)

# Fit the model on training data
rf_tuned2.fit(X_train_res, y_train_res)
```

Out[506…

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('randomforestclassifier',
                 RandomForestClassifier(max_features='sqrt',
                                        max_samples=0.5000000000000001,
                                        n_estimators=250, random_state=1))])
```

In [507…

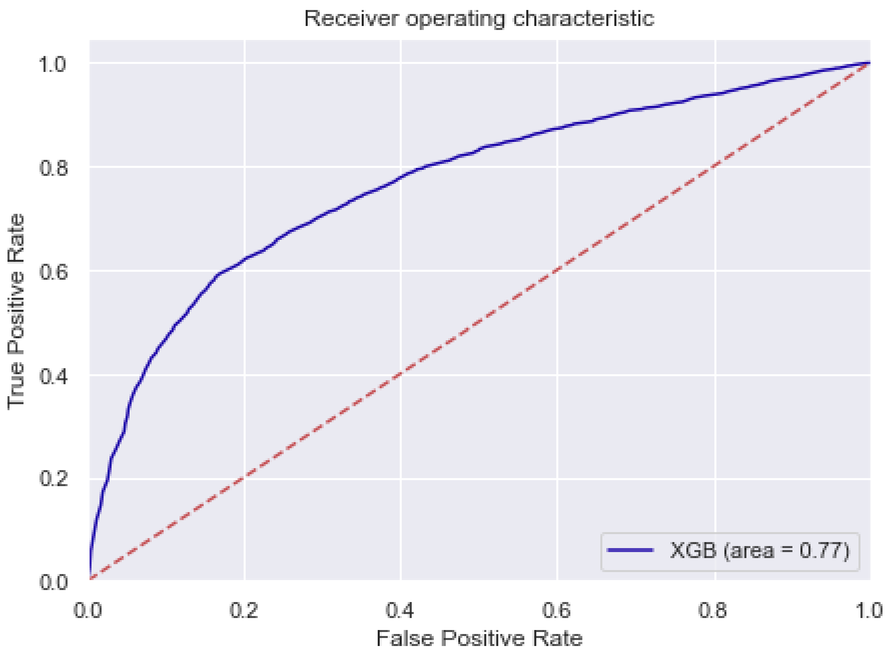
```
# Calculating different metrics
get_metrics_score1(rf_tuned2,X_train_res,X_test9,y_train_res,y_test)

# Creating confusion matrix
make_confusion_matrix(rf_tuned2, y_test)
```

```
Accuracy on training set :  0.9842742089392725
Accuracy on test set :  0.8398313574887294
Recall on training set :  0.9774800564907058
Recall on test set :  0.8682816048448145
Precision on training set :  0.9909453236853306
Precision on test set :  0.9568652468348219
```

- Recall on train is 98%, but 87% on test
- 22,457 positive responses, 1,499 negative
- 620 true negatives and 2958 false negatives
- 620 correctly predicted as Default
- 4% false positives

In [508…

```
# ROC-AUC on RF Test
logit_roc_auc_train = roc_auc_score(y_test, rf_tuned2.predict_proba(X_test9)[:,1])
fpr, tpr, thresholds = roc_curve(y_test, rf_tuned2.predict_proba(X_test9)[:,1])
plt.figure(figsize=(7,5))
plt.plot(fpr, tpr, label='XGB (area = %0.2f)' % logit_roc_auc_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



- Test AUC = .77

Model Chosen: XGBoost Hypertuned

In [509…

```
%%time

#Creating pipeline
pipe=make_pipeline(StandardScaler(),XGBClassifier(random_state=1,eval_metric='logloss')

#Parameter grid to pass in RandomSearchCV
param_grid={'xgbclassifier__n_estimators':np.arange(50,300,50),'xgbclassifier__scale_po
            'xgbclassifier__learning_rate':[0.01,0.1,0.2,0.05], 'xgbclassifier__gamma':
            'xgbclassifier__subsample':[0.7,0.8,0.9,1]}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=pipe, param_distributions=param_grid, n_it

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_res,y_train_res)

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_params_,ran
```

```
Best parameters are {'xgbclassifier__subsample': 0.8, 'xgbclassifier__scale_pos_weight':
10, 'xgbclassifier__n_estimators': 250, 'xgbclassifier__learning_rate': 0.2, 'xgbclassif
ier__gamma': 5} with CV score=0.9962593580472469:
Wall time: 19min 51s
```

In [510…

```
# Creating new pipeline with best parameters
xgb_tuned2 = Pipeline(
    steps=[
        ("scaler", StandardScaler()),
        (
```

```
                "XGB",
                XGBClassifier(
                    random_state=1,
                    n_estimators=250,
                    scale_pos_weight=10,
                    learning_rate=0.02,
                    gamma=5,
                    subsample=0.8,
                    eval_metric='logloss',
                ),
            ),
        ]
)
# Fit the model on training data
xgb_tuned2.fit(X_train_res, y_train_res)
```

Out[510…

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('XGB',
                 XGBClassifier(base_score=0.5, booster='gbtree',
                               colsample_bylevel=1, colsample_bynode=1,
                               colsample_bytree=1, eval_metric='logloss',
                               gamma=5, gpu_id=-1, importance_type='gain',
                               interaction_constraints='', learning_rate=0.02,
                               max_delta_step=0, max_depth=6,
                               min_child_weight=1, missing=nan,
                               monotone_constraints='()', n_estimators=250,
                               n_jobs=8, num_parallel_tree=1, random_state=1,
                               reg_alpha=0, reg_lambda=1, scale_pos_weight=10,
                               subsample=0.8, tree_method='exact',
                               validate_parameters=1, verbosity=None))])
```

In [352…

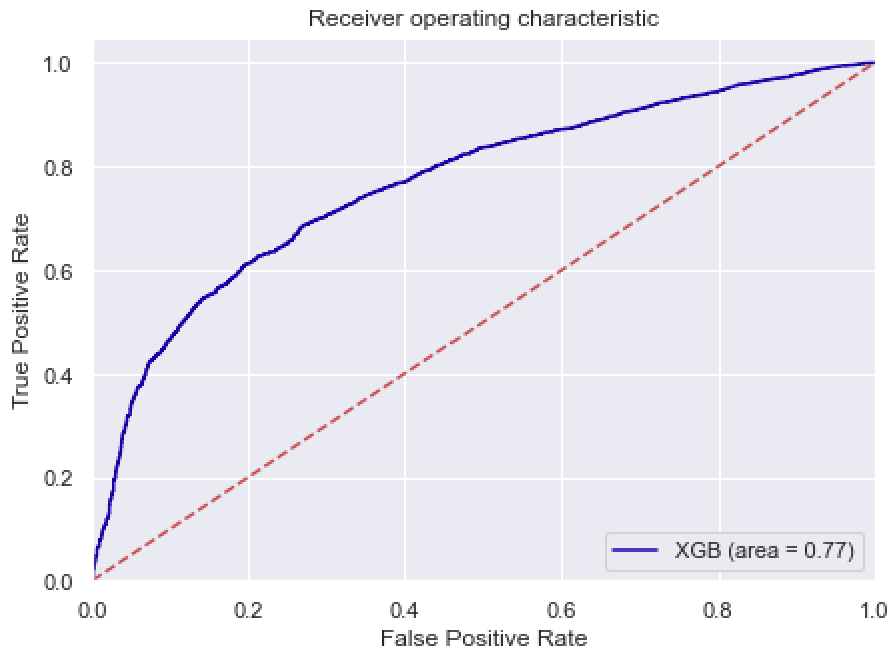
```
# Calculating different metrics
get_metrics_score1(xgb_tuned2,X_train_res,X_test9,y_train_res,y_test)

# Creating confusion matrix
make_confusion_matrix(xgb_tuned2, y_test)
```

```
Accuracy on training set :  0.6573819611435551
Accuracy on test set :  0.9316246451828352
Recall on training set :  0.9908202603152793
Recall on test set :  0.9897582045687313
Precision on training set :  0.5944172839788874
Precision on test set :  0.9404273323460969
```

- The model was able to capture 22,457 (93.74%) Non-Defaulters, but more importantly, only 1499 Defaulters (6.26%), which is the same as the Random Forest model
- However, there were only 91 true negatives and 230 false negatives.
- 6% false positives
- Really high recall train and test scores
- Precision is low on training, but high on test

In [353…

```
# ROC-AUC on XGB Test
logit_roc_auc_train = roc_auc_score(y_test, xgb_tuned2.predict_proba(X_test9)[:,1])
fpr, tpr, thresholds = roc_curve(y_test, xgb_tuned2.predict_proba(X_test9)[:,1])
plt.figure(figsize=(7,5))
plt.plot(fpr, tpr, label='XGB (area = %0.2f)' % logit_roc_auc_train)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```



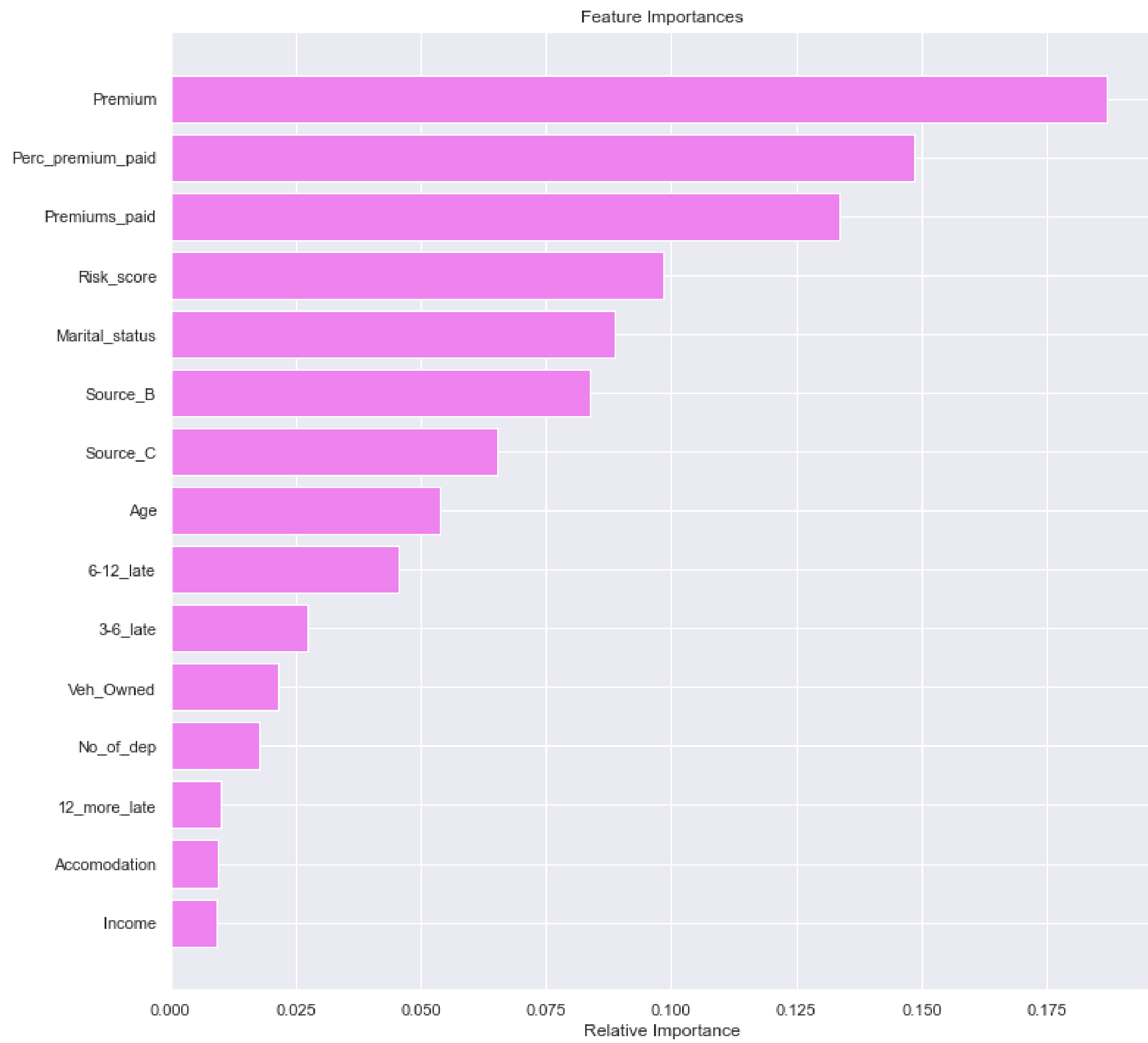
In [354…

```
# The optimal cut off would be where tpr is high and fpr is low
fpr, tpr, thresholds = roc_curve(y_test, xgb_tuned2.predict_proba(X_test9)[:,1])

optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print(optimal_threshold)
```

```
0.9461598
```

In [355…

```
feature_names = X_train.columns
importances = xgb_tuned2[1].feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12, 12))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



- Top 5 most important features: Premium, Percentage of Premium paid, the Number of Premiums paid, Risk score, and Marital Status

Executive Summary

Problem Statement

- Premium paid by the customer is the major revenue source for insurance companies. Default in premium payments results in significant revenue losses and hence, insurance companies would like to know upfront which type of customers would default premium payments. The objective of this project is to predict the probability that a customer will default the premium payment, so that the insurance agent can proactively reach out to the policy holder to follow up for the payment of premium.

Methods, Final Insights, Recommendations

- Two different classification models were created to analyse the data so that we could predict who would default and who would not default on the payment of the insurance premium.
- The two models were the Random Forest and the XGBoost models. The Random Forest model uses and fits a number of decision trees on sub-samples of the dataset, then takes an average to predict. The method helps to reduce variance and thus control over-fitting. The XGBoost model is also a Decision Tree based model, but is different in that it uses a parallel series of weak models to help strengthen the final model by learning from the errors of the weak models. This method is fast and very accurate by avoiding overfitting and also decreasing bias.
- AUC-ROC and Confusion Matrices were used to test the performance of the models. Hyperparameter tuning was also utilized to find the best parameters to use in creating the final models.
- To deal with the imbalanced dataset, SMOTE (Synthetic Minority Oversampling Technique) was used to oversample the minority class to create balance.
- The top 5 important features derived from the model are:
  - The amount of the premium of the policy
  - The percentage of the premium paid by customer
  - The number of premiums paid by the customer
  - The customer’s risk score
  - The customer’s marital status
- Insights and Recommendations:
  - With a high recall score and a high precision score on tests, we can say that the model did a good job of identifying defaulters.
  - The model performance can be improved, or other models could be considered. Using the important features:
  - Customers that show and increase in premium amounts will see a less likelihood of defaulting.
  - Those that, on average, paid a higher percentage of their premiums were least likely to default (see negative coefficient).
  - Married couples, and customers that paid more premiums tend to default less.
  - Discounts or promotional deals could be extended to those responsible customers who have demonstrated consistency in on-time payments. Loyalty rewards could also be presented as incentives for new and continuing customers—those that have made many payments over a certain number of policy renewals. For married couples, adjusted rates could be offered to reflect joint income and credit risk potentials.
  - Efforts could be made to reach out to those who are higher risk or have made many and/or consistent late payments (6-12). Special outreach or educational programs could be provided for these higher risk customers (ways to lower premiums, payment plans for high premiums or different policies that allow a modified payment method, ways to reduce risk score through online classes), and follow up contacts (email and phone).

Approach to final model selection

- After creating the initial models, I ran an algorithm comparison wherein the XGBoost model outperformed the Random Forest by about 4% (91.5% to 87.5%)
- I then compared the AUC-ROC of the train and test datasets where the XGBOost train performed much better than the Random Forest, but both returned the same AUC
- After hypertuning, the AUC's were identical, but the Confusion Matrices were also created to compare the two classes predicitions and actuals. The XGBoost did a better job of generalizing

In [ ]:

```

```