AutoInland Vehicle Insurance Claim Challenge

This is a notebook by Samson Tontoye.

Table of Contents

Introduction

This notebook looks into using various python-based machine learning and data science libraries in an attempt to develop a predictive machine learning model that determines if a customer will submit a vehicle insurance claim within next three months from their first transaction.

What we'll end up with

Since we already have a dataset, we'll approach the problem with the following machine learning modelling framework.

To work through these topics, we'll use pandas, Matplotlib and NumPy for data anaylsis, as well as, Scikit-Learn for machine learning and modelling tasks.

We'll work through each step and by the end of the notebook, we'll have a handful of models, all which can predict whether or not a customer will claim insurance within the first three months using precision, recall and F1 score as the evaluation metric.

1. Problem Definition

This notebook is an insurance claim classification machine learning project with an imbalanced class.In this case, the problem we will be exploring is binary classification (a sample can only be one of two things).

This is because we are going to be using a number of different features (pieces of information) to predict if a customer will submit a vehicle insurance claim within next three months from their first transaction.

2. Data

The dataset came from Zindi in a formatted way Zindi

The data describes ~12,000 policies sold by AutoInland for car insurance. Information on the car type, make, customer age and start of policy are in the data.

3. Evaluation

The evaluation metric is something to define at the start of a project.

Since machine learning is very experimental:

```
    Since we are working with a highly imbalanced dataset, we will use the
precision, recall and F1 score as an appropriate evaluation metric. If we can get a
score of say 0.9 or over across this three evaluation metric at predicting whether
or not a customer will submit a vehicle insurance claim within next three months
from their first transaction during the proof of concept, we'll pursue this
project.
```

4. Features

Features are different parts of the data. We're going to visualize the relationships between the different features of the data and how it can lead to a customer that will submit an insurance claim.

One of the most common ways to understand the features is to look at the **data dictionary**. For this project, the data dictionary is in the **Variable Definitions** csv file.

In [367…

```python
# Data Manipulation
import numpy as np
import pandas as pd
import scipy

# Visualizations
import matplotlib.pyplot as plt
import seaborn as sns
import missingno
import plotly.express as px
import plotly.figure_factory as ff

# Feature Selection and Encoding
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# Machine Learning
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
import catboost as cb
import xgboost as xgb
import lightgbm as lgb

# For the imbalanced dataset
from imblearn.over_sampling import SMOTE

# Model Evaluations
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, roc_a
from sklearn.metrics import plot_roc_curve

# Manage warnings
```

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

#For Datetime
import datetime as dt

from tqdm.notebook import tqdm_notebook
from collections import Counter

# Plot the figure inline
%matplotlib inline
```

2. Read Data

In [367…

```
# Load the data into a pandas dataframe
train = pd.read_csv('AutoInland-Vehicle-insurance-claim-challenge/Train.csv',
                    low_memory=False,
                    parse_dates=['Policy Start Date', 'Policy End Date', 'First Transactio
test = pd.read_csv('AutoInland-Vehicle-insurance-claim-challenge/Test.csv',
                   low_memory=False,
                    parse_dates=['Policy Start Date', 'Policy End Date', 'First Transactio
ss = pd.read_csv('AutoInland-Vehicle-insurance-claim-challenge/SampleSubmission.csv')
var_def = pd.read_csv('AutoInland-Vehicle-insurance-claim-challenge/VariableDefinitions.cs
state_names = pd.read_csv('AutoInland-Vehicle-insurance-claim-challenge/NigerianStateNames
```

3. Exploratory Data Analysis

Cleaning: To clean our data, we'll need to work with:

- **Missing values**: Either omit elements from a dataset that contain missing values or impute them (fill them in).
- **Special values**: Numeric variables are endowed with several formalized special values including ±Inf, NA and NaN. Calculations involving special values often result in special values, and need to be handled/cleaned.
- **Outliers**: They should be detected, but not necessarily removed. Their inclusion in the analysis is a statistical decision.
- **Obvious inconsistencies**: A person's age cannot be negative, an under-aged person cannot possess a drivers license. Find the inconsistencies and plan for them.

When exploring our dataset and its features, we have many options available to us. We can explore each feature individually, or compare pairs of features, finding the correlation between. Let's start with some simple Univariate (one feature) analysis.

Features can be of multiple types:

- Nominal: is for mutual exclusive, but not ordered, categories.
- Ordinal: is one where the order matters but not the difference between values.
- Interval: is a measurement where the difference between two values is meaningful.
- Ratio: has all the properties of an interval variable, and also has a clear definition of 0.0.

There are multiple ways of manipulating each feature type, but for simplicity, we'll define only two feature types:

- Numerical: any feature that contains numeric values.
- Categorical: any feature that contains categories, or text.

Since EDA has no real set methodology, the following is a short check list to to walk through:

1. From the dataframe features, what features are the highest indicator that the customer will claim insurance in the first 3 months?
2. What's missing from the data and how do you deal with it?
3. Does gender play a role in a customer claiming insurance in the first 3 months?
4. Does policy start date play a role in a customer claiming insurance in the first 3 months?
5. Does policy end date/First transaction date play a role in a customer claiming insurance in the first 3 months?

In [367…

```
# Top 5 rows of the train dataframe
train.head()
```

Out[367…

	ID	Policy Start Date	Policy End Date	Gender	Age	First Transaction Date	No_Pol	Car_Category	Subject_Car_Colour	Subject_Car_M
0	ID_0040R73	2010-05-14	2011-05-13	Male	30	2010-05-14	1	Saloon	Black	TOY(
1	ID_0046BNK	2010-11-29	2011-11-28	Female	79	2010-11-29	1	JEEP	Grey	TOY(
2	ID_005QMC3	2010-03-21	2011-03-20	Male	43	2010-03-21	1	Saloon	Red	TOY(
3	ID_0079OHW	2010-08-21	2011-08-20	Male	2	2010-08-21	1	NaN	NaN	N
4	ID_00BRP63	2010-08-29	2010-12-31	Entity	20	2010-08-29	3	NaN	NaN	N

In [367…

```
# Top 5 rows of the test dataframe
test.head()
```

Out[367…

	ID	Policy Start Date	Policy End Date	Gender	Age	First Transaction Date	No_Pol	Car_Category	Subject_Car_Colour	Subject_Car_M
0	ID_01QM0NU	2010-10-23	2011-10-22	Female	46	2010-10-23	1	NaN	NaN	I
1	ID_024NJLZ	2010-10-14	2011-10-13	Male	32	2010-10-14	1	NaN	NaN	I
2	ID_02NOVWQ	2010-08-29	2011-08-28	Female	45	2010-08-29	2	Saloon	Black	Hc
3	ID_02VSP68	2010-06-13	2011-06-12	Female	58	2010-06-13	1	Saloon	NaN	TOY
4	ID_02YB37K	2010-07-01	2011-06-30	NaN	120	2010-07-01	1	Saloon	Red	Hyu

In [367…

```
# Check the variable definition
var_def
```

Out[367…

ID	Unique ID for the customer

	ID	Unique ID for the customer
0	Policy Start Date	Date policy started
1	Policy End Date	Date policy ended
2	Gender	Gender of the customer
3	Age	Age of the customer
4	ProductName	Name of Insurance policy
5	First Transaction Date	First date payment was made
6	No_Pol	Number of policies the customer has
7	Car_Category	Type of car
8	Subject_Car_Colour	Car colour
9	Subject_Car_Make	Car make
10	LGA_Name	City where policy was purchased
11	State	State where policy was purchased
12	No_of_claims_3_mon_period	Wether the customer claimed within a 3 month p...

In [367...

```
# Check the state names
state_names
```

Out[367...

	LGA	State
0	Abadam	Borno State
1	Abaji	Federal Capital Territory
2	Abak	Akwa Ibom State
3	Abakaliki	Ebonyi State
4	Aba-North	Abia State
...	...	...
870	Warri	Warri-South-West
871	Warri-Central	Warri-South
872	Wuse-11	Abuja-Municipal-Area-Council
873	Yaba	Lagos-Mainland
874	NaN	Unknown

875 rows × 2 columns

In [367...

```
# Let's see how many positive (1) and negative (0) samples we have in our dataframe
print('Length of the train dataset:', len(train))
print('Length of the test dataset:', len(test))
print('Total no of customers that will not claim insurance in the first 3 months:', len(tr
print('Total no of customers that will claim insurance in the first 3 months:', len(train
```

```
Length of the train dataset: 12079
Length of the test dataset: 1202
Total no of customers that will not claim insurance in the first 3 months: 10624
Total no of customers that will claim insurance in the first 3 months: 1455
```

Since these two values(Customers that will claim insurance within the first 3 months and customers that will not claim insurance in the first 3 months) are not close, our target column can be considered **imbalanced**. An **imbalanced** target column, meaning some classes have far more samples, can be harder to model than a balanced set. From our target column, if the customer will claim insurance in the first 3 months, it is denoted with 1, if the customer will not claim insurance in the first 3 months, it is denoted as 0.

In [367…

```
# To see the value in percentages
train['target'].value_counts(normalize=True)
```

Out[367…

```
0    0.879543
1    0.120457
Name: target, dtype: float64
```

The proportion of customer that will not claim insurance in the first 3 months to those that will claim insurance in the first 3 months is 7.3:1. 1 in 7 customers will claim insurance in the first 3 months.

In [367…

```
# Check the shape of the train and test dataset
print(f'The shape of the train dataset is: {train.shape}\nThe shape of the test dataset is
```

```
The shape of the train dataset is: (12079, 14)
The shape of the test dataset is: (1202, 13)
```

In [368…

```
# Check the information for the training dataset
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12079 entries, 0 to 12078
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   ID                      12079 non-null  object
 1   Policy Start Date       12079 non-null  datetime64[ns]
 2   Policy End Date         12079 non-null  datetime64[ns]
 3   Gender                  11720 non-null  object
 4   Age                     12079 non-null  int64
 5   First Transaction Date  12079 non-null  datetime64[ns]
 6   No_Pol                  12079 non-null  int64
 7   Car_Category            8341 non-null   object
 8   Subject_Car_Colour      5117 non-null   object
 9   Subject_Car_Make        9603 non-null   object
 10  LGA_Name                5603 non-null   object
 11  State                   5591 non-null   object
 12  ProductName             12079 non-null  object
 13  target                  12079 non-null  int64
dtypes: datetime64[ns](3), int64(3), object(8)
memory usage: 1.3+ MB
```

In [368…

```
# Check for missing values in the training dataset
train.isna().sum()
```

Out[368…

```
ID                           0
Policy Start Date            0
Policy End Date              0
Gender                     359
Age                          0
First Transaction Date       0
No_Pol                       0
Car_Category              3738
Subject_Car_Colour        6962
Subject_Car_Make          2476
LGA_Name                  6476
```

```
State                    6488
ProductName                 0
target                      0
dtype: int64
```

In [368…

```
# Check the information for the test dataset
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1202 entries, 0 to 1201
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype         
---  ------                  --------------  -----         
 0   ID                      1202 non-null   object        
 1   Policy Start Date       1202 non-null   datetime64[ns]
 2   Policy End Date         1202 non-null   datetime64[ns]
 3   Gender                  1161 non-null   object        
 4   Age                     1202 non-null   int64         
 5   First Transaction Date  1202 non-null   datetime64[ns]
 6   No_Pol                  1202 non-null   int64         
 7   Car_Category            830 non-null    object        
 8   Subject_Car_Colour      505 non-null    object        
 9   Subject_Car_Make        954 non-null    object        
 10  LGA_Name                546 non-null    object        
 11  State                   546 non-null    object        
 12  ProductName             1202 non-null   object        
dtypes: datetime64[ns](3), int64(2), object(8)
memory usage: 122.2+ KB
```

From the pandas test dataframe above,

- Age and No_Pol are int datatype.
- Policy Start Date, Policy End Date and First Transaction Date are datetime datatype.
- The rest of the columns are object datatype.

In [368…

```
# The .describe() function will demonstrate the count, mean, std dev, min, max, etc values
# Numerical features present in the  train dataset.
train.describe()
```

Out[368…

	Age	No_Pol	target
count	12079.000000	12079.000000	12079.000000
mean	42.234539	1.307227	0.120457
std	97.492565	0.733085	0.325509
min	-6099.000000	1.000000	0.000000
25%	35.000000	1.000000	0.000000
50%	41.000000	1.000000	0.000000
75%	50.000000	1.000000	0.000000
max	320.000000	10.000000	1.000000

In [368…

```
# The .describe() function will demonstrate the count, mean, std dev, min, max, etc values
# Numerical features present in the test dataset
test.describe()
```

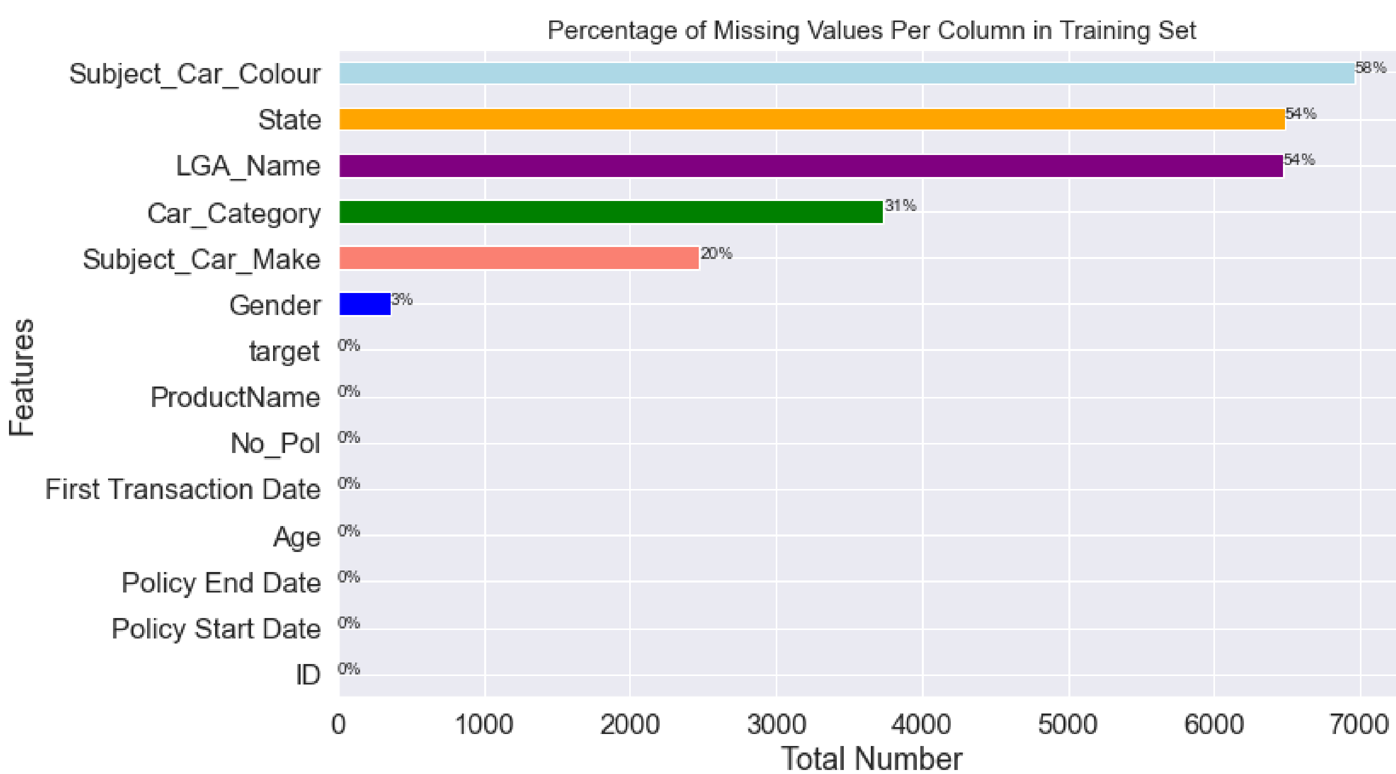
Out[368…

	Age	No_Pol

	Age	No_Pol
count	1202.000000	1202.000000
mean	43.792845	1.257903
std	19.986245	0.613510
min	-26.000000	1.000000
25%	35.000000	1.000000
50%	41.000000	1.000000
75%	50.000000	1.000000
max	120.000000	7.000000

In [368…

```
# Visualizing the missing values in the training dataset
ax = train.isna().sum().sort_values().plot(kind = 'barh', figsize = (10, 7),
                                           color=['indigo', 'yellow', 'brown', 'pink',
                                                  'cyan', 'gray', 'olive', 'orangered',
                                                  'blue', 'salmon', 'green', 'purple',
                                                  'orange','lightblue', 'red', 'violet'])

# Add some attributes
plt.title('Percentage of Missing Values Per Column in Training Set', fontdict={'size':15})
plt.xlabel('Total Number')
plt.ylabel('Features')

for p in ax.patches:
    percentage ='{:,.0f}%'.format((p.get_width()/train.shape[0])*100)
    width, height =p.get_width(),p.get_height()
    x=p.get_x()+width+0.02
    y=p.get_y()+height/2
    ax.annotate(percentage,(x,y));
```

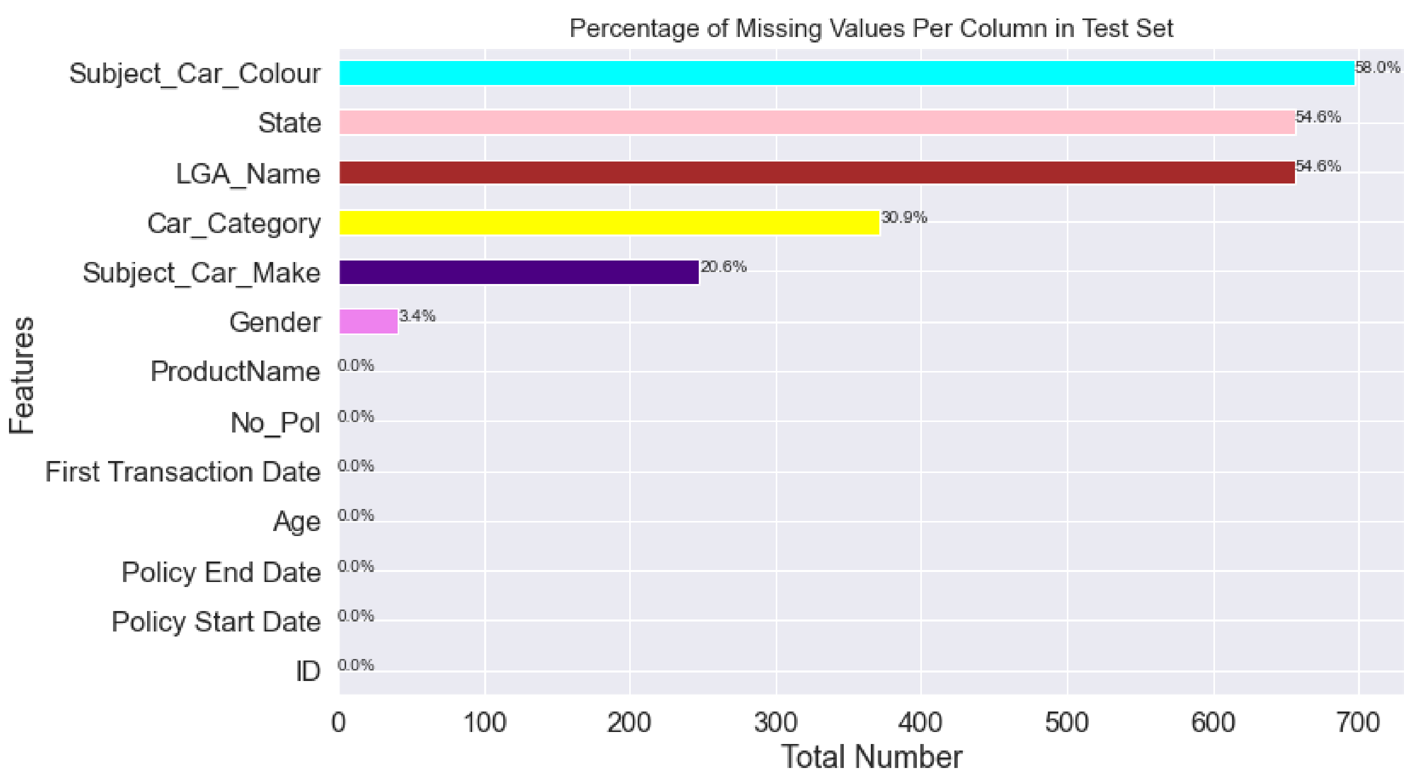


In [368…

```
# Check for missing values in the test dataset
test.isna().sum()
```

Out[368…
```
ID                        0
Policy Start Date         0
Policy End Date           0
Gender                   41
Age                       0
First Transaction Date    0
No_Pol                    0
Car_Category            372
Subject_Car_Colour      697
Subject_Car_Make        248
LGA_Name                656
State                   656
ProductName               0
dtype: int64
```

In [368…
```
# Visualizing the missing values in the test dataset
ax = test.isna().sum().sort_values().plot(kind = 'barh', figsize = (10, 7),
                                          color=['blue', 'salmon', 'green', 'purple',
                                                 'orange','lightblue', 'r
                                                 'indigo', 'yellow', 'bro
                                                 'cyan', 'gray', 'olive',


# Add some attributes
plt.title('Percentage of Missing Values Per Column in Test Set', fontdict={'size':15})
plt.xlabel('Total Number')
plt.ylabel('Features')

for p in ax.patches:
    percentage ='{:,.1f}%'.format((p.get_width()/test.shape[0])*100)
    width, height =p.get_width(),p.get_height()
    x=p.get_x()+width+0.02
    y=p.get_y()+height/2
    ax.annotate(percentage,(x,y))
```



In [368…
```
train.columns
```

Out[368…
```
Index(['ID', 'Policy Start Date', 'Policy End Date', 'Gender', 'Age',
       'First Transaction Date', 'No_Pol', 'Car_Category',
```

```
       'Subject_Car_Colour', 'Subject_Car_Make', 'LGA_Name', 'State',
       'ProductName', 'target'],
      dtype='object')
```

Investigate numeric variables- Age, No_Pol

Histograms for each, their effect on the target.

Potentially graph their effects

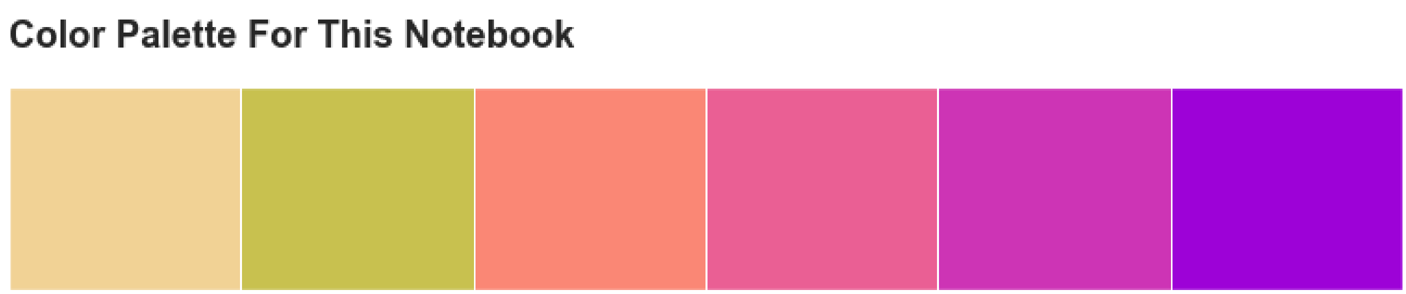
3.1 Create Custom Color Palette

In [368…

```
# Create the different shades of colors for the color pallete
colors = ["#f1d295", "#c8c14f", "#fa8775", "#ea5f94", "#cd34b5", "#9d02d7"]
palette = sns.color_palette(palette = colors)

sns.palplot(palette, size = 2)
plt.text(-0.5, -0.7, 'Color Palette For This Notebook', size = 20, weight = 'bold')
```

Out[368…

```
Text(-0.5, -0.7, 'Color Palette For This Notebook')
```



3.2 Numeric Variables

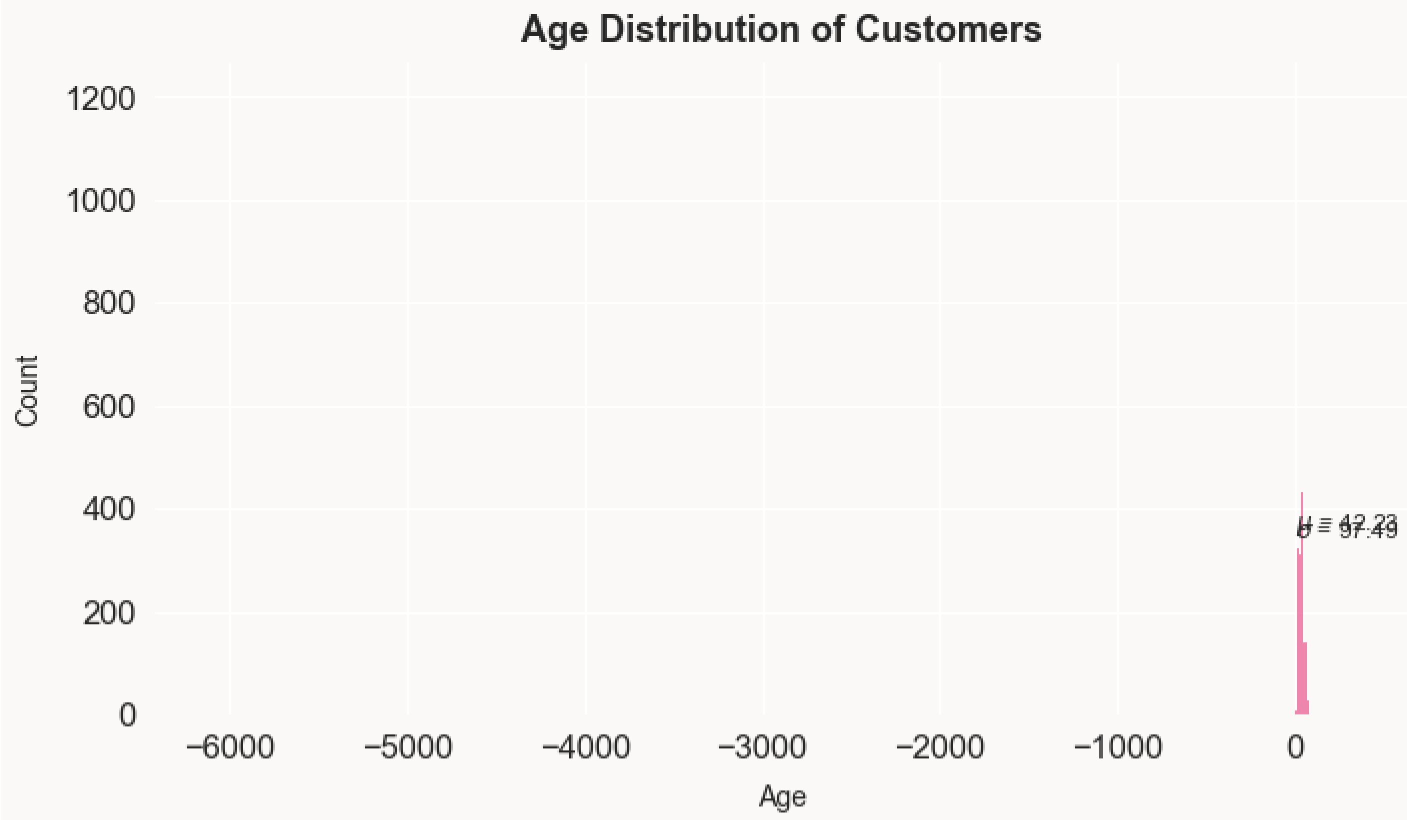
Age Distribution of Customers.

In [369…

```
# Plot the histogram for the age distribution of customers
fig, ax = plt.subplots(figsize = (10,6))
fig.patch.set_facecolor('#faf9f7')
ax.set_facecolor('#faf9f7')

sns.histplot(
    train['Age'],
    kde = False,
    color = "#ea5f94"
)

for i in ['top', 'left', 'bottom', 'right']:
    ax.spines[i].set_visible(False)

plt.text(5, 360, r'$\mu$ = '+str(round(train['Age'].mean(), 2)), fontsize = 12)
plt.text(5, 343, r'$\sigma$ = '+str(round(train['Age'].std(), 2)), fontsize = 12)
plt.title('Age Distribution of Customers', fontsize = 18, fontweight = 'bold', pad = 10)
plt.xlabel('Age', fontsize = 14, labelpad = 10)
plt.ylabel('Count', fontsize = 14, labelpad = 10)
```

Out[369…

```
Text(0, 0.5, 'Count')
```



Check for Outliers in Age

In [369…

```
# Plot a boxplot of the age of the customers showing the spread and IQR before removing ou
sns.boxplot(y = 'Age', data = train ,palette='Accent')
```

Out[369…

<AxesSubplot:ylabel='Age'>

In [369…

```
# Visualize the age distribution of the customers before removing outliers
Age = list(train['Age'].values)
hist_data=[Age]

group_labels=['Age']
colour=['Red']

fig = ff.create_distplot(hist_data, group_labels, show_hist=True, colors=colour)
fig.show()
print('The shape before removing the Age outliers :', train.shape)
```

```
The shape before removing the Age outliers : (12079, 14)
```

In [369…

```python
# Visualize the age distribution of the customers after removing outliers
Age=list(train['Age'].values)
hist_data=[Age]

group_lables=['Age']
colour=['Red']

fig=ff.create_distplot(hist_data,group_lables,show_hist=True,colors=colour)
fig.show()

train.drop(train[train['Age'] < 0].index, inplace = True)
train.drop(train[train['Age'] < 18].index, inplace = True)
train.drop(train[train['Age'] > 150].index, inplace = True)
print("The shape after removing the Age outliers : ",train.shape)
```

```
The shape after removing the Age outliers :  (11511, 14)
```

In [369…

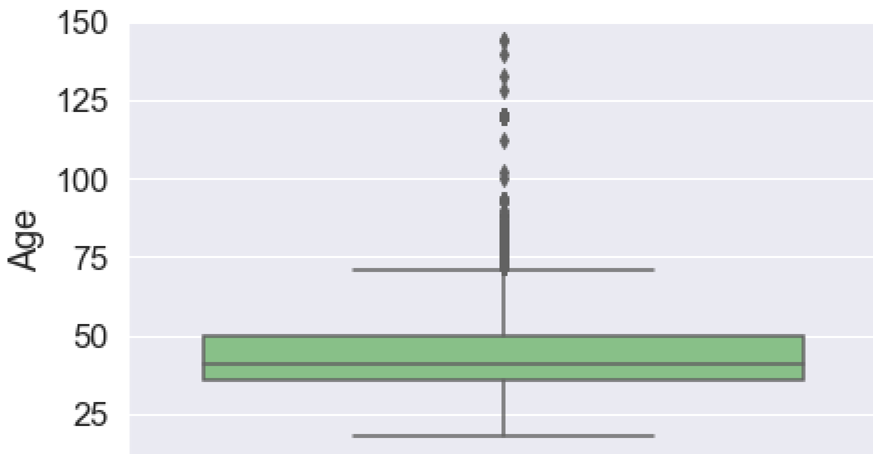
```python
# Visualize the spread of the age distribution of the customers after removing outliers
train_Age = train.groupby('Age', as_index=False)['target'].sum()
fig = px.histogram(train_Age,
                   x = "Age",
                   y = "target",
                   barmode = "group",
                   nbins = 10,
                   opacity = 0.75,
                   range_x = [0,85],
                   color_discrete_sequence=px.colors.qualitative.Light24)

fig.update_layout(height = 500,
                  width = 700,
                  title_text ='Age Distribution of Customers',
                  title_font_size= 20,
                  title_y = 0.97,
                  title_x = 0.48,
                  yaxis_title = 'Count')

fig.show()
```

In [369…

```
# Plot a boxplot of the age of the customers showing the spread and IQR after removing ou
sns.boxplot(y = 'Age', data = train ,palette='Accent')
```

Out[369…

```
<AxesSubplot:ylabel='Age'>
```



Observation:

After removing the outliers, the range is now between 18 and 144 which is where most of the age samples are distributed.

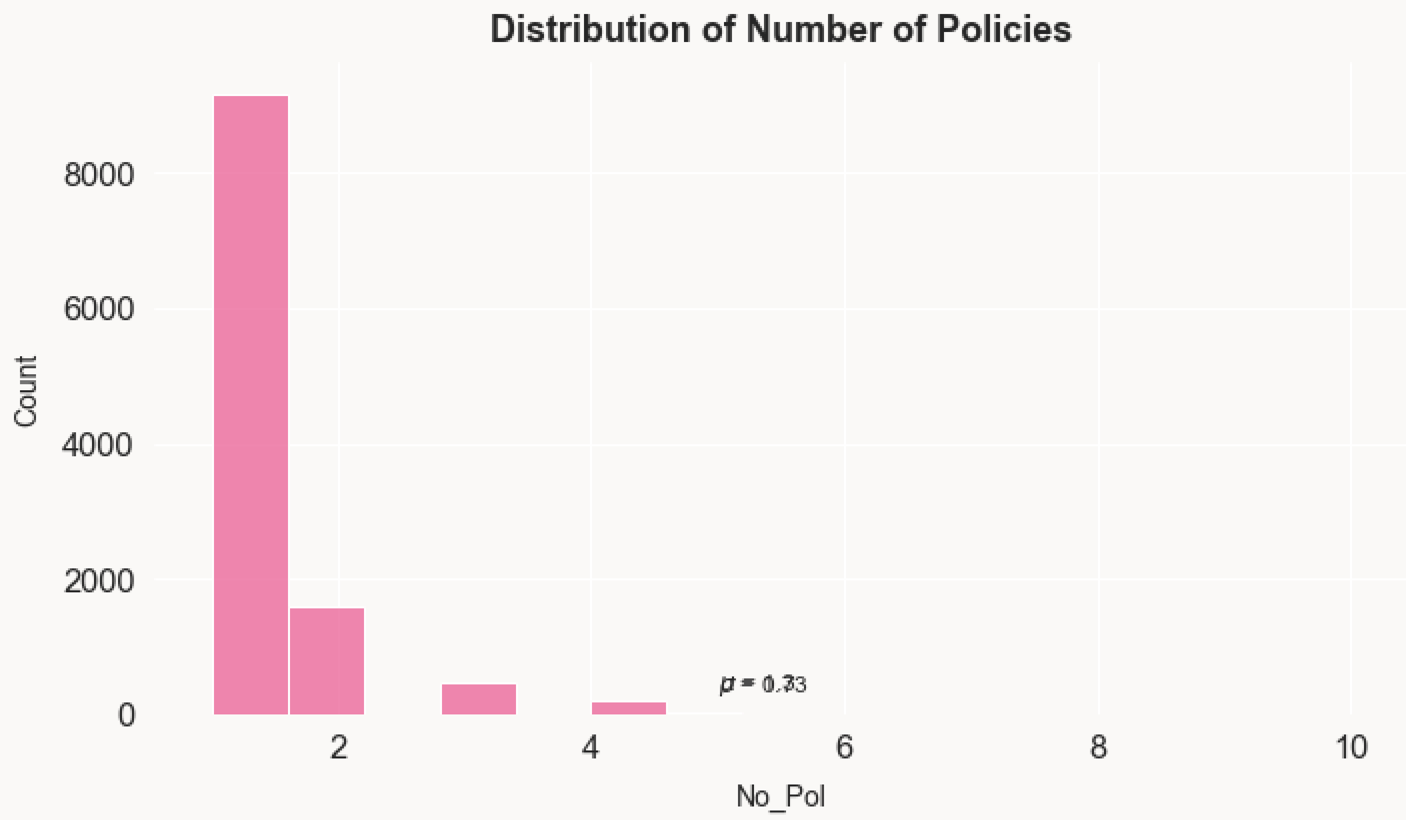
Frequency of Number of Policies.

In [369…

```
fig, ax = plt.subplots(figsize = (10,6))
fig.patch.set_facecolor('#faf9f7')
ax.set_facecolor('#faf9f7')

sns.histplot(
    train['No_Pol'],
    kde = False,
    color = "#ea5f94"
)

for i in ['top', 'left', 'bottom', 'right']:
    ax.spines[i].set_visible(False)

plt.text(5, 360, r'$\mu$ = '+str(round(train['No_Pol'].mean(), 2)), fontsize = 12)
plt.text(5, 343, r'$\sigma$ = '+str(round(train['No_Pol'].std(), 2)), fontsize = 12)
plt.title('Distribution of Number of Policies', fontsize = 18, fontweight = 'bold', pad =
plt.xlabel('No_Pol', fontsize = 14, labelpad = 10)
plt.ylabel('Count', fontsize = 14, labelpad = 10)
```

Out[369…

```
Text(0, 0.5, 'Count')
```



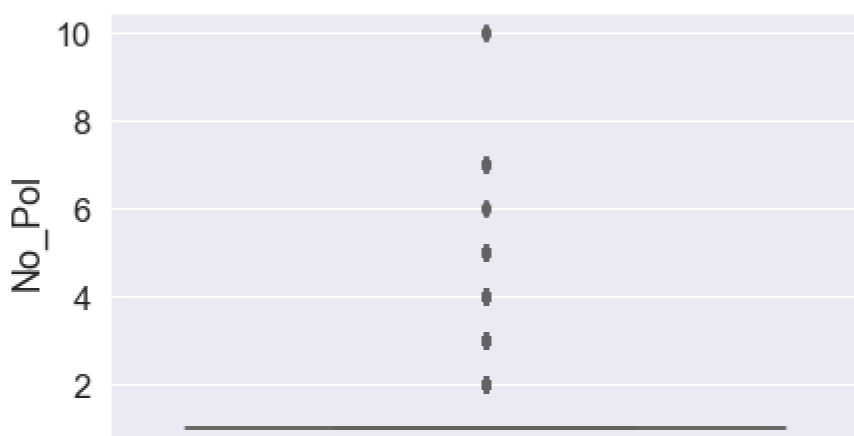
Check Outliers in Number of Policies

In [369…

```
# Plot a boxplot of the number of policies of the customers showing the spread and IQR wi
sns.boxplot(y = 'No_Pol', data = train ,palette='Accent')
```

Out[369…

```
<AxesSubplot:ylabel='No_Pol'>
```



In [369…

```
# Display the Number of Policies Outliers
No_Pol = list(train['No_Pol'].values)
hist_data=[No_Pol]

group_labels=['No_Pol']
colour=['Red']

fig = ff.create_distplot(hist_data, group_labels, show_hist=True, colors=colour)
fig.show()
print('Display the Number of Policies Outliers :', train.shape)
```

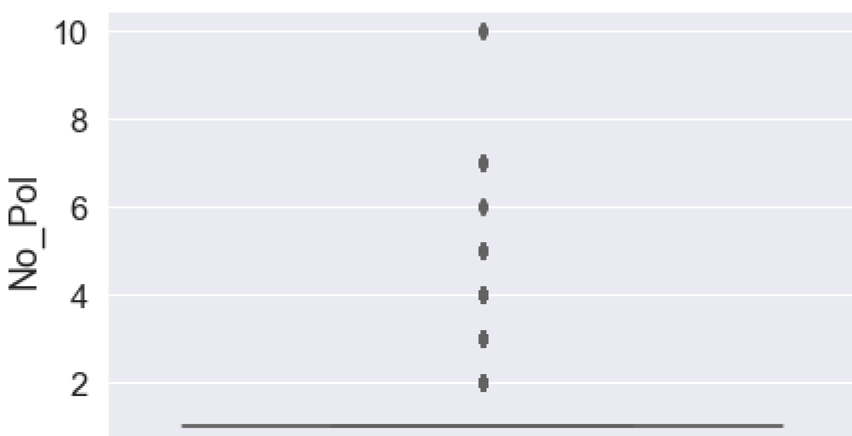
```
Display the Number of Policies Outliers : (11511, 14)
```

In [370…

```
# Plot a boxplot of the number of policies of the customers showing the spread and IQR aft
sns.boxplot(y = 'No_Pol', data = train ,palette='Accent')
```

Out[370…

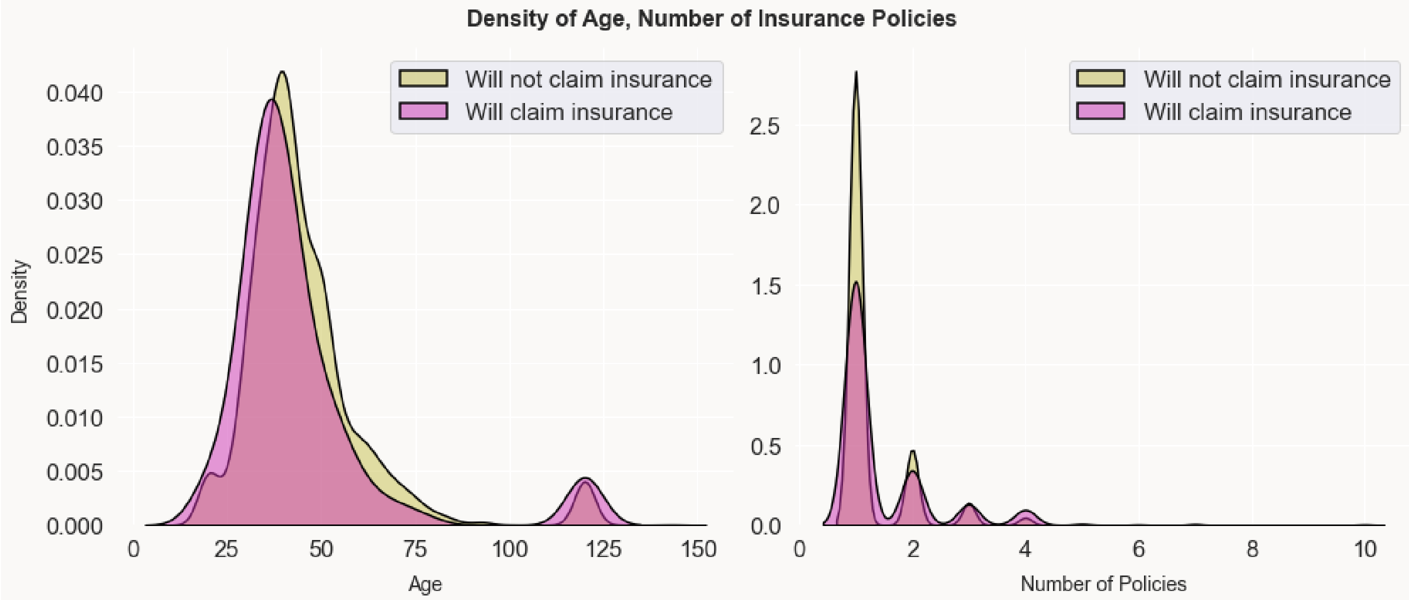
```
<AxesSubplot:ylabel='No_Pol'>
```



In [370…

```
# Plot the density plot of the Age and Number of Policies of Insurance
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (14, 6))
fig.patch.set_facecolor('#faf9f7')

for i in (ax1, ax2):
    i.set_facecolor('#faf9f7')

sns.kdeplot(
    train['Age'][train['target'] == 0],
    ax = ax1,
    color = '#c8c14f',
    shade = True,
```

```
    alpha = 0.5,
    linewidth = 1.5,
    ec = 'black'
)

sns.kdeplot(
    train['Age'][train['target'] == 1],
    ax = ax1,
    color = '#cd34b5',
    shade = True,
    alpha = 0.5,
    linewidth = 1.5,
    ec = 'black'
)

ax1.legend(['Will not claim insurance', 'Will claim insurance'], loc = 'upper right')
ax1.set_xlabel('Age', fontsize = 14, labelpad = 10)
ax1.set_ylabel('Density', fontsize = 14, labelpad = 10)

sns.kdeplot(
    train['No_Pol'][train['target'] == 0],
    ax = ax2,
    color = '#c8c14f',
    shade = True,
    alpha = 0.5,
    linewidth = 1.5,
    ec = 'black'
)

sns.kdeplot(
    train['No_Pol'][train['target'] == 1],
    ax = ax2,
    color = '#cd34b5',
    shade = True,
    alpha = 0.5,
    linewidth = 1.5,
    ec = 'black'
)

ax2.legend(['Will not claim insurance', 'Will claim insurance'], loc='upper right')
ax2.set_xlabel('Number of Policies', fontsize = 14, labelpad = 10)
ax2.set_ylabel('')

plt.suptitle('Density of Age, Number of Insurance Policies', fontsize = 16, fontweight = '

for i in (ax1, ax2):
    for j in ['top', 'left', 'bottom', 'right']:
        i.spines[j].set_visible(False)

fig.tight_layout()
```



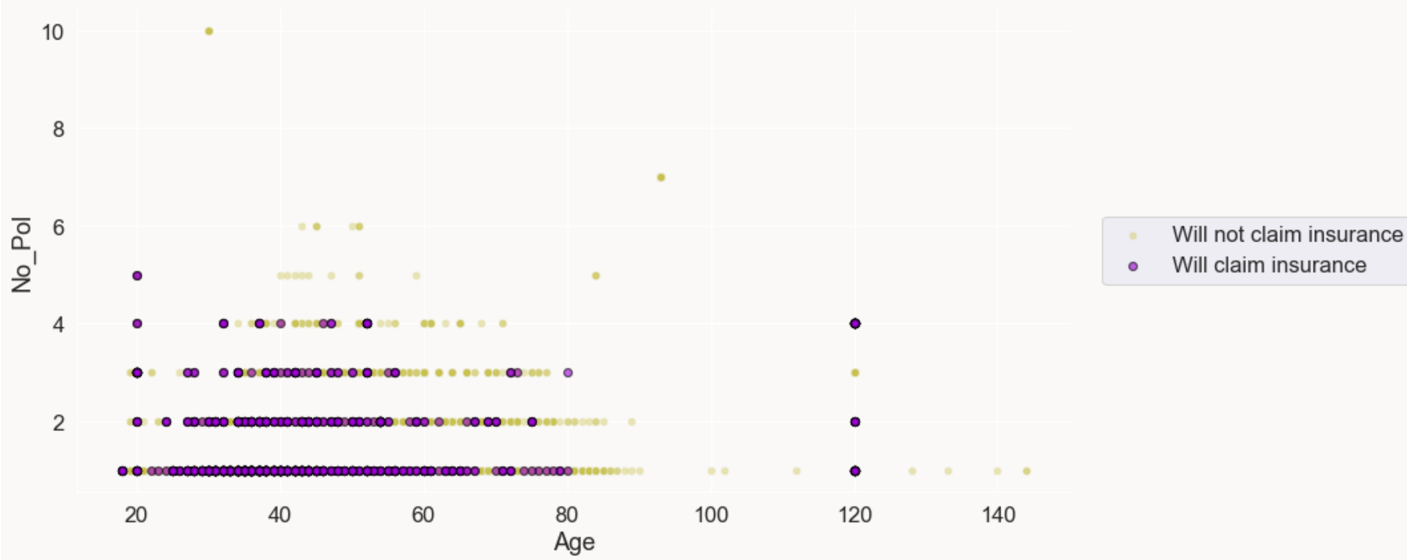
Scatter plots of numerical variables colored by insurance.

In [370…

```
# Assign the target column to a variable
will_claim_insurance = train[train['target'] == 1]
will_not_claim_insurance = train[train['target'] == 0]
```

In [370…

```
# Plot a scatter plot of the number of insurance policies against age
fig, ax = plt.subplots(1, 1, figsize=(15, 6))
fig.patch.set_facecolor('#faf9f7')
for j in range(0, 1):
    ax.set_facecolor('#faf9f7')

## Age vs Number of Policies
sns.scatterplot(
    data = will_not_claim_insurance, x = 'Age', y = 'No_Pol', color = '#c8c14f',
    alpha = 0.4, ax = ax
)
sns.scatterplot(
    data = will_claim_insurance, x = 'Age', y = 'No_Pol', color = "#9d02d7",
    ax = ax, edgecolor = 'black', linewidth = 1.2, alpha = 0.6
)

sns.despine()

for i in range(0, 1):
    ax.legend(['Will not claim insurance', 'Will claim insurance'], loc='center left', bbo


fig.tight_layout()
```



3.3 Categorical Variables

Let's first investigate the target variable.

In [370…

```
# Check the proportion of classes of the target column
train['target'].value_counts(normalize = True)
```

Out[370…

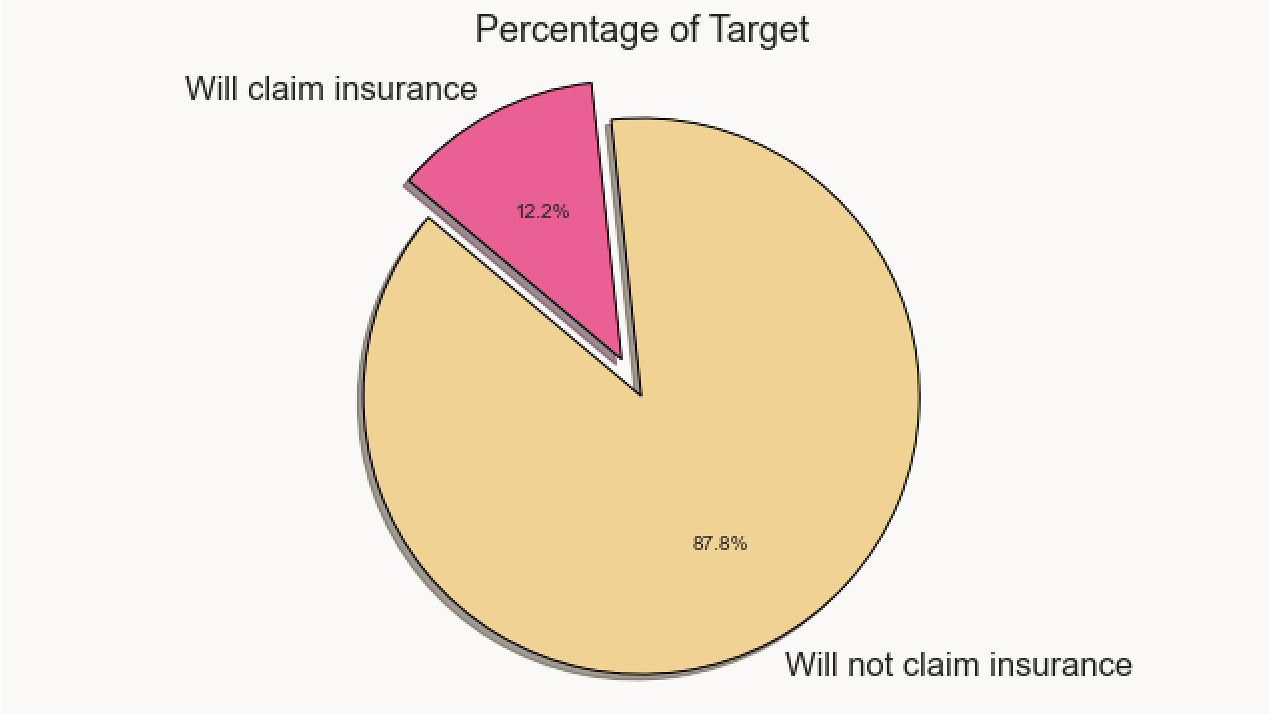
```
0    0.878464
1    0.121536
Name: target, dtype: float64
```

In [370…

```
# Plot a pie chart of the target column showing the proportion
fig, ax = plt.subplots(figsize = (10, 6))
fig.patch.set_facecolor('#faf9f7')
ax.set_facecolor('#faf9f7')

labels = ['Will not claim insurance', 'Will claim insurance']
colors = ['#f1d295', '#ea5f94']
sizes = train['target'].value_counts()

plt.pie(sizes, explode = [0, 0.15], labels = labels, colors = colors,
            autopct = '%1.1f%%', shadow = True, startangle = 140,
            wedgeprops = {'ec': 'black'}, textprops = {'fontweight': 'medium'}
)

plt.axis('equal')
plt.title('Percentage of Target')
```

Out[370…

```
Text(0.5, 1.0, 'Percentage of Target')
```



Gender

In [370…

```
# Check the number of genders
train.Gender.value_counts()
```

Out[370…

```
Male            7322
Female          3193
Joint Gender     215
Entity           201
NOT STATED       146
NO GENDER         62
SEX               35
Name: Gender, dtype: int64
```

In [370…

```
# Replace gender that is not male or female with other
mapper = {'Entity':'Other', 'Joint Gender':'Other', 'NOT STATED':'Other', 'NO GENDER': 'Ot
train.Gender = train.Gender.replace(mapper)

# Confirm mappings
train.Gender.value_counts()
```

Out[370…

```
Male      7322
Female    3193
Other      659
Name: Gender, dtype: int64
```

In [371…

```
# Replace the other with nan
train['Gender'] = train['Gender'].replace({'Other': np.nan})
train['Gender']
```

Out[371…

```
0          Male
1        Female
2          Male
4           NaN
5          Male
          ...
12074    Female
12075    Female
12076      Male
12077       NaN
```

```
12078    Female
Name: Gender, Length: 11511, dtype: object
```

In [371…

```
# Drop cases where either variable is missing
data = train[['Gender', 'target']].dropna()
pd.crosstab(data.Gender, data.target)
```

Out[371…

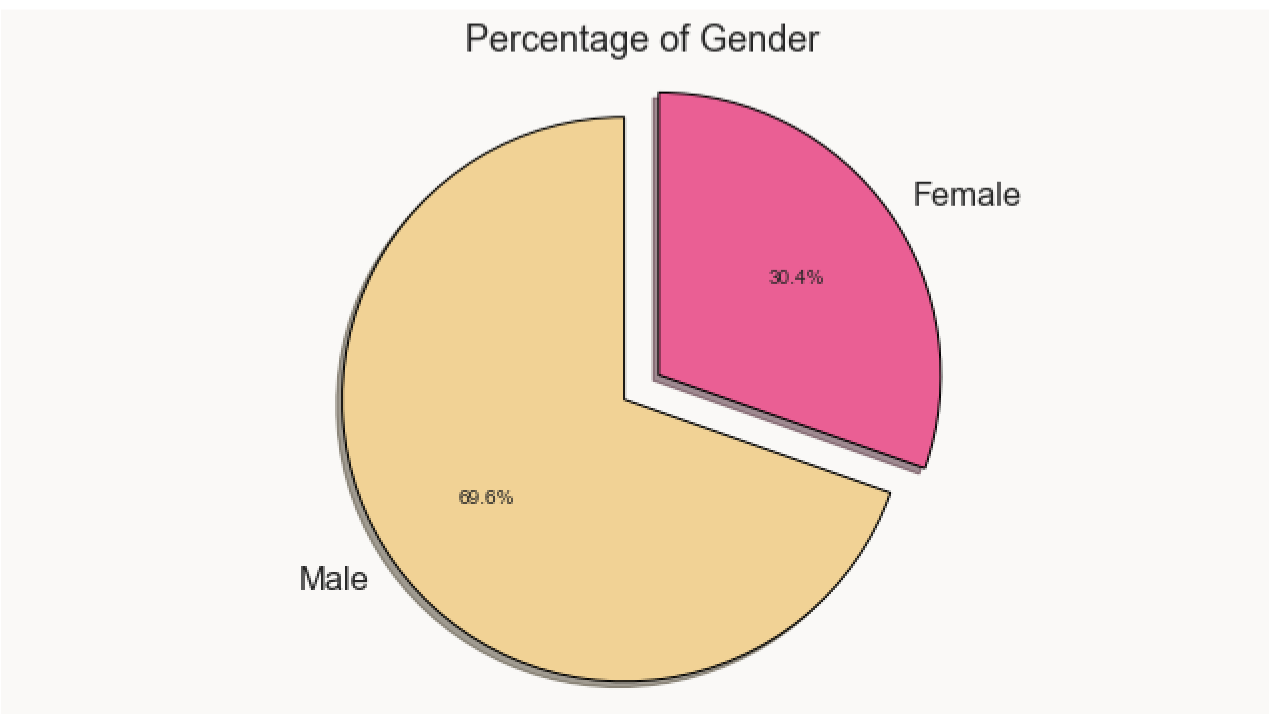
target	0	1
Gender		
Female	2842	351
Male	6432	890

In [371…

```
# Plot a pie chart of the gender column showing the proportion of male to female
fig, ax = plt.subplots(figsize = (10, 6))
fig.patch.set_facecolor('#faf9f7')
ax.set_facecolor('#faf9f7')

labels = ['Male', 'Female']
colors = ['#f1d295', '#ea5f94']
sizes = train['Gender'].value_counts()

plt.pie(sizes, explode = [0, 0.15], labels = labels, colors = colors,
            autopct = '%1.1f%%', shadow = True, startangle = 90,
            wedgeprops = {'ec': 'black'}, textprops = {'fontweight': 'medium'}
)

plt.axis('equal')
plt.title('Percentage of Gender')
```

Out[371…

```
Text(0.5, 1.0, 'Percentage of Gender')
```



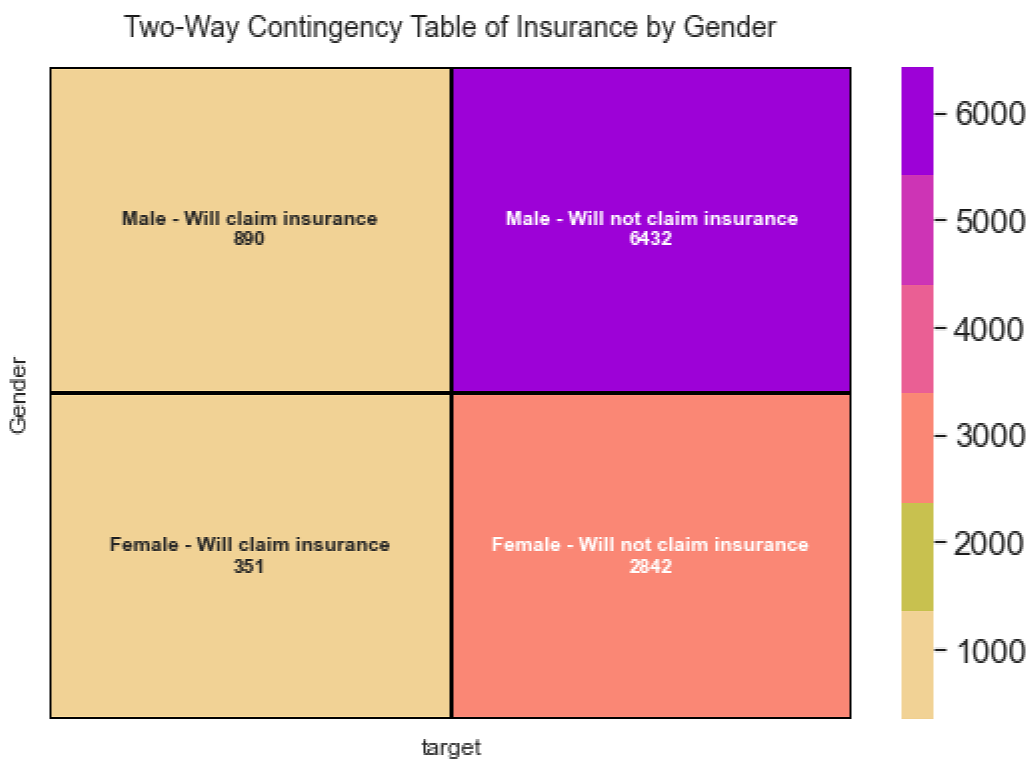
In [371…

```
# plot a two way contingency table of insurance by gender
plt.subplots(figsize=(8,6))

insurance_matrix = np.array([[890, 6432], [351, 2842]])
labels = np.array([['Male - Will claim insurance', 'Male - Will not claim insurance'],
```

```
                          ['Female - Will claim insurance', 'Female - Will not claim insurance']]
formatted = (np.asarray(["{0}\n{1:.0f}".format(text, data) for text, data in zip(labels.fl


sns.heatmap(
    insurance_matrix,
    annot = formatted,
    fmt = '',
    cmap = palette,
    xticklabels = False,
    yticklabels = False,
    linecolor = 'black',
    linewidth = 1,
    annot_kws = {'fontweight': 'semibold'}
)
plt.title('Two-Way Contingency Table of Insurance by Gender', pad = 15, fontsize = 14)
plt.ylabel('Gender', fontsize = 12, labelpad = 10)
plt.xlabel('target', fontsize = 12, labelpad = 10)
```

Out[371…

```
Text(0.5, 50.0, 'target')
```



Observation:

Since there are 3193 women, 2842 women will not claim insurance and 348 of them will claim insurance, we might infer, based on this one variable if the customer is a woman, there's a 11% chance the female customer will claim insurance.

As for males, there are 7322 males, 6432 men will not claim insurance and 890 of them will claim insurance. So we might predict, if the customer is male, there is a 12.2% chance he will claim insurance.

Averaging these two values, we can assume, based on no other parameters, if there's a person, there's a 11.6% chance they will claim insurance.

Car Category

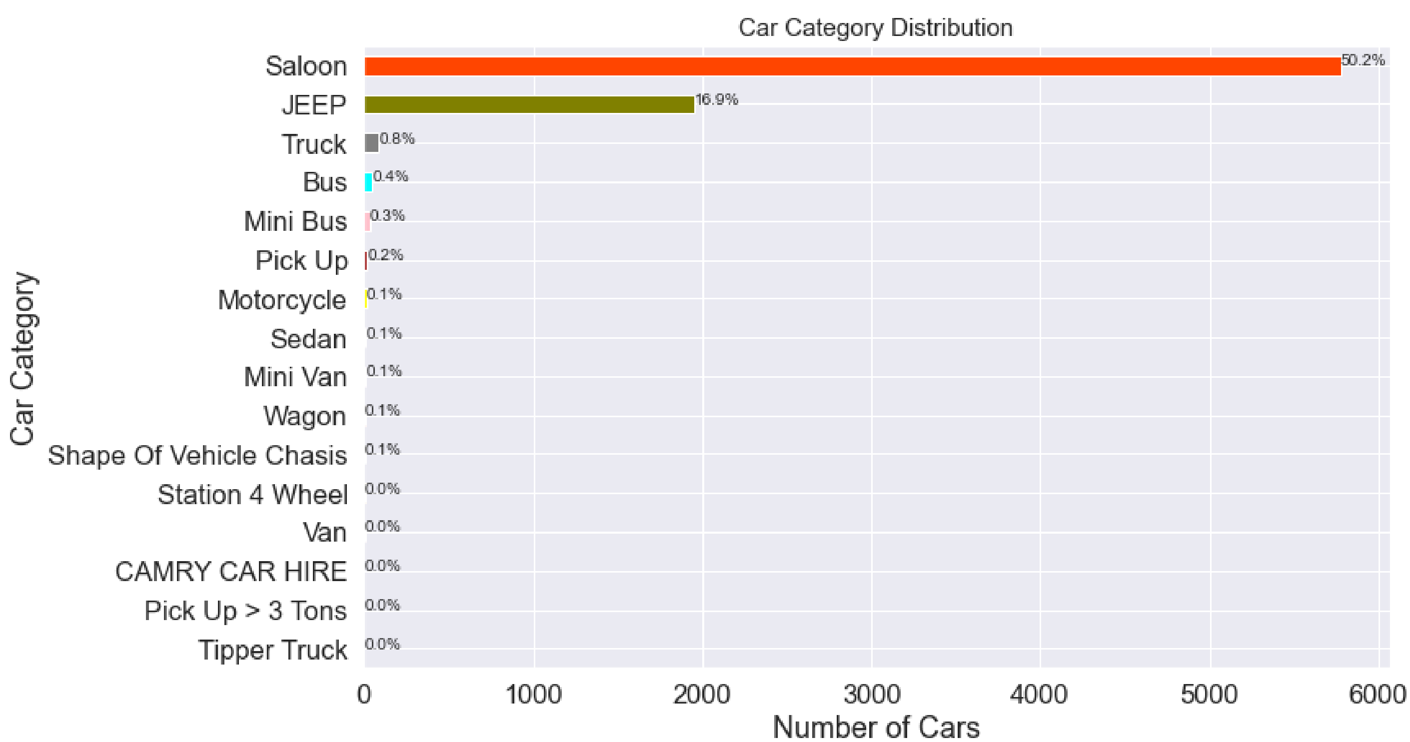
In [371…

```
# Check the proportion of each car in the car category
train.Car_Category.value_counts(normalize=True)
```

```
Out[371…  Saloon                       0.723378
          JEEP                         0.244425
          Truck                        0.011150
          Bus                          0.006264
          Mini Bus                     0.004260
          Pick Up                      0.002756
          Motorcycle                   0.001629
          Mini Van                     0.001503
          Sedan                        0.001503
          Wagon                        0.001002
          Shape Of Vehicle Chasis      0.000752
          Station 4 Wheel              0.000626
          Van                          0.000376
          Tipper Truck                 0.000125
          Pick Up > 3 Tons             0.000125
          CAMRY CAR HIRE               0.000125
          Name: Car_Category, dtype: float64
```

In [371…

```
# Car_category Distribution before Joining
ax = train.Car_Category.value_counts().sort_values().plot(kind='barh',
                                                            figsize=(10, 7),
                                                            color=['blue', 'salmon', 'gree
                                                                   'orange','lightblue', 'r
                                                                   'indigo', 'yellow', 'bro
                                                                   'cyan', 'gray', 'olive',


# Add some attributes
plt.title('Car Category Distribution', fontdict={'size': 15})
plt.xlabel('Number of Cars')
plt.ylabel('Car Category')

for p in ax.patches:
    percentage = '{:,.1f}%'.format((p.get_width()/train.shape[0]) * 100)
    width, height = p.get_width(), p.get_height()
    x = p.get_x() + width + 0.02
    y = p.get_y() + height/2
    ax.annotate(percentage, (x, y));
```



In [371…

```
# Replace car category that is not saloon or jeep with others
mapper = {'Truck':'Others', 'Mini Bus':'Others', 'Bus':'Others', 'Pick Up': 'Others', 'Sec
```

```
            'Mini Van':'Others', 'Wagon': 'Others', 'Shape Of Vehicle Chasis': 'Others', 'St
            'Pick Up > 3 Tons': 'Others', 'CAMRY CAR HIRE': 'Others', 'Tipper Truck': 'Other
train.Car_Category = train.Car_Category.replace(mapper)

# Confirm mappings
train.Car_Category.value_counts()
```

Out[371…

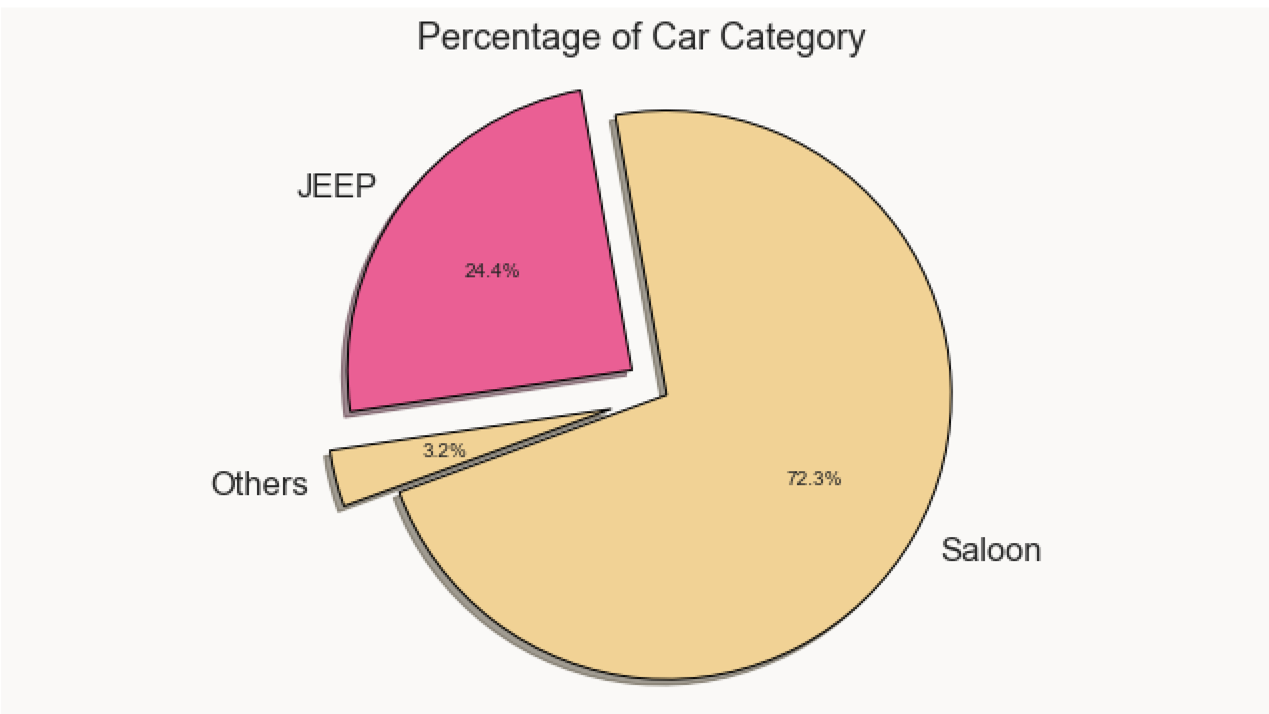
```
Saloon     5774
JEEP       1951
Others      257
Name: Car_Category, dtype: int64
```

In [371…

```
# Plot a pie chart of the car category column showing the proportion after joining
fig, ax = plt.subplots(figsize = (10, 6))
fig.patch.set_facecolor('#faf9f7')
ax.set_facecolor('#faf9f7')

labels = ['Saloon', 'JEEP', 'Others']
colors = ['#f1d295', '#ea5f94']
sizes = train['Car_Category'].value_counts()

plt.pie(sizes, explode = [0, 0.15, 0.2], labels = labels, colors = colors,
           autopct = '%1.1f%%', shadow = True, startangle = 200,
           wedgeprops = {'ec': 'black'}, textprops = {'fontweight': 'medium'}
)

plt.axis('equal')
plt.title('Percentage of Car Category')
```

Out[371…

```
Text(0.5, 1.0, 'Percentage of Car Category')
```



Observation:

72.3% of the vehicles insured by AutoInland insurance are saloon cars, 24.4% of vehicles insured are jeeps and 3.2% of vehicles insured have been grouped as others.

In [371…

```
# Compare the target column to car_category
pd.crosstab(train.Car_Category, train.target)
```

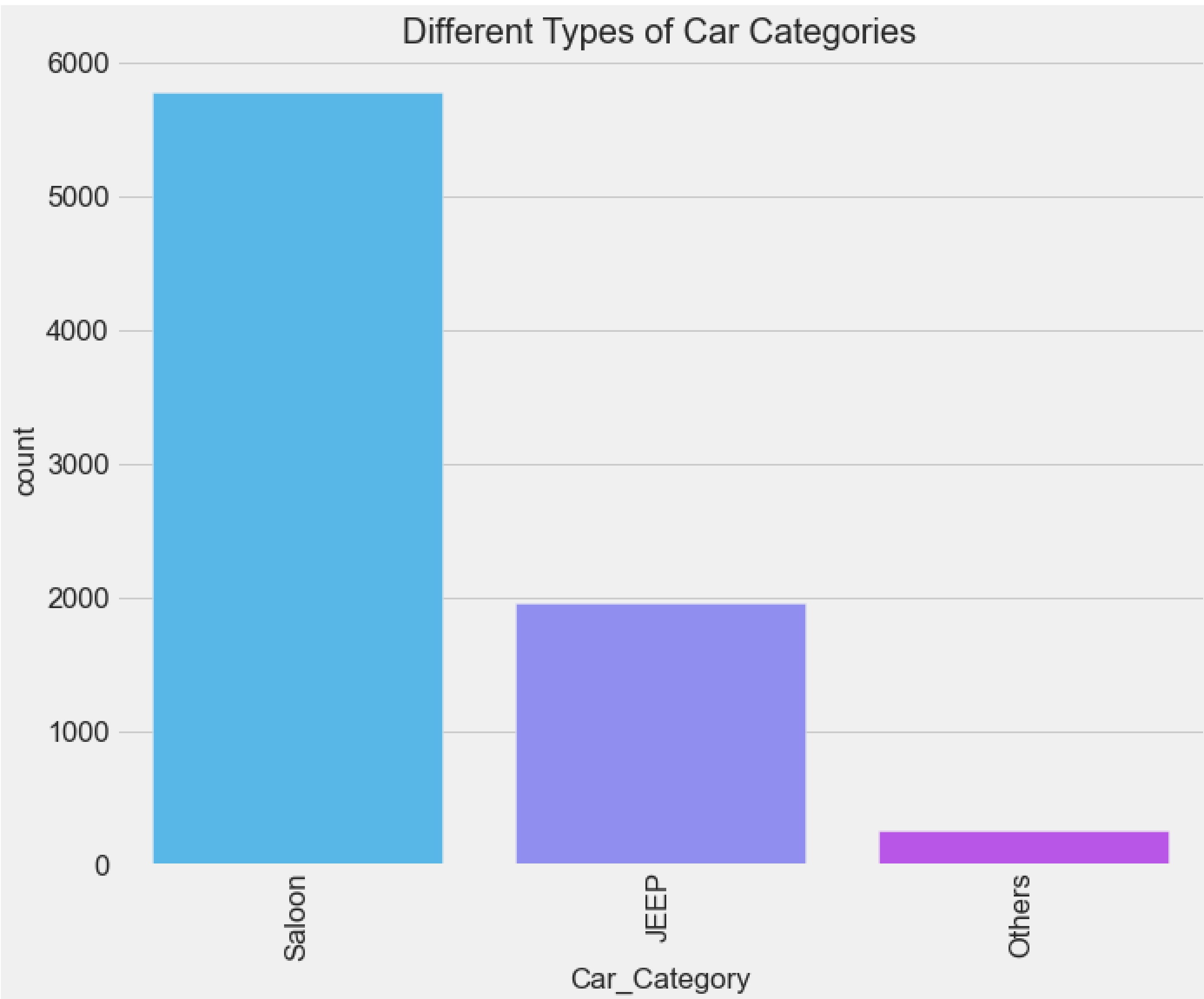
Out[371…

target	0	1
Car_Category		
JEEP	1714	237
Others	202	55
Saloon	4978	796

Frequency of Car Category

In [371…

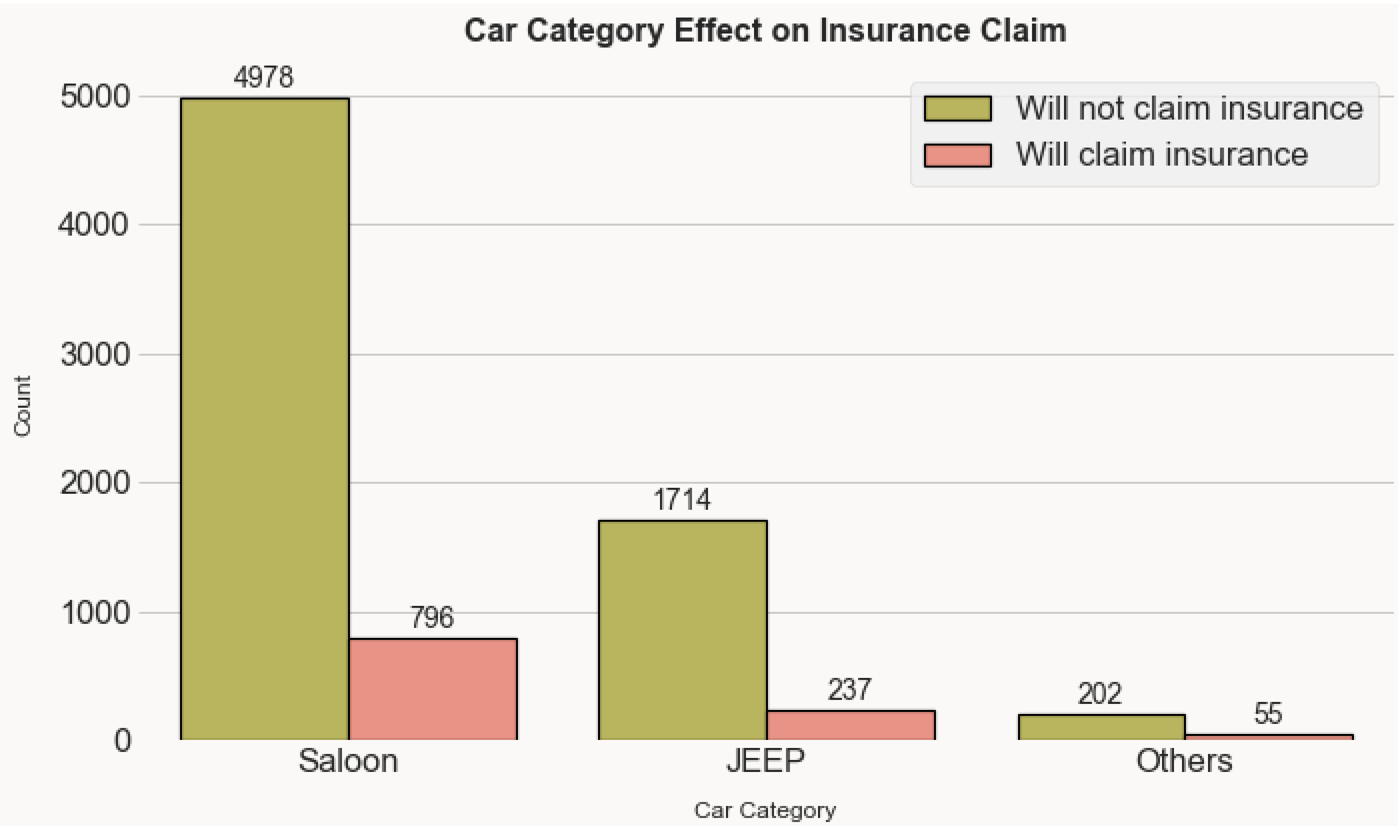
```
# let's check the car categories

plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (10, 8)

sns.countplot(train['Car_Category'], palette = 'cool')
plt.title('Different Types of Car Categories', fontsize = 20)
plt.xticks(rotation = 90)
plt.show()
```



In [372…

```
# Barchart of the different car categories and check the number that claimed insurance and
# claim insurance after joining.
fig, ax = plt.subplots(figsize=(10,6))
fig.patch.set_facecolor('#faf9f7')
```

```
ax.set_facecolor('#faf9f7')

bar_pal = ["#c8c14f", "#fa8775"]

s = sns.countplot(
    data = train, x = 'Car_Category', hue = 'target', palette = bar_pal,
    linewidth = 1.2, ec = 'black'
)

for i in ['top', 'right', 'bottom', 'left']:
    ax.spines[i].set_visible(False)

plt.legend(['Will not claim insurance', 'Will claim insurance'])
plt.title("Car Category Effect on Insurance Claim", size = 16, weight = 'bold', pad = 12)
plt.xlabel('Car Category', size = 12, labelpad = 12)
plt.ylabel('Count', size = 12, labelpad = 12)

for i in s.patches:
    s.annotate(format(i.get_height(), '.0f'),  (i.get_x() + i.get_width() / 2., i.get_heig

fig.tight_layout()
```



Observation

Out of the 5774 saloon car owners, 4978 saloon car owners will not claim insurance within the first three months from their first transaction while 796 will claim insurance within the first three months of their first transaction. Of the 1951 jeep owners, 1714 will not claim insurance while 237 jeep owners will claim insurance within the first three months of their first transaction.Of the others vehicle category, 202 will not claim insurance while 55 will claim insurance within the first three months of their first transaction.

Subject Car Make

In [372…

```
# Check the proportion of subject car make
train.Subject_Car_Make.value_counts(normalize=True)
```

Out[372…

```
TOYOTA          0.520572
```

```
Honda         0.108807
Lexus         0.063844
Mercedes      0.054567
Hyundai       0.045509
                ...
Tata          0.000109
CHANGAN       0.000109
Raston        0.000109
ZOYTE         0.000109
Rols Royce    0.000109
Name: Subject_Car_Make, Length: 68, dtype: float64
```

In [372…

```
# Compare the target column to subject car make
pd.crosstab(train.Subject_Car_Make, train.target)
```

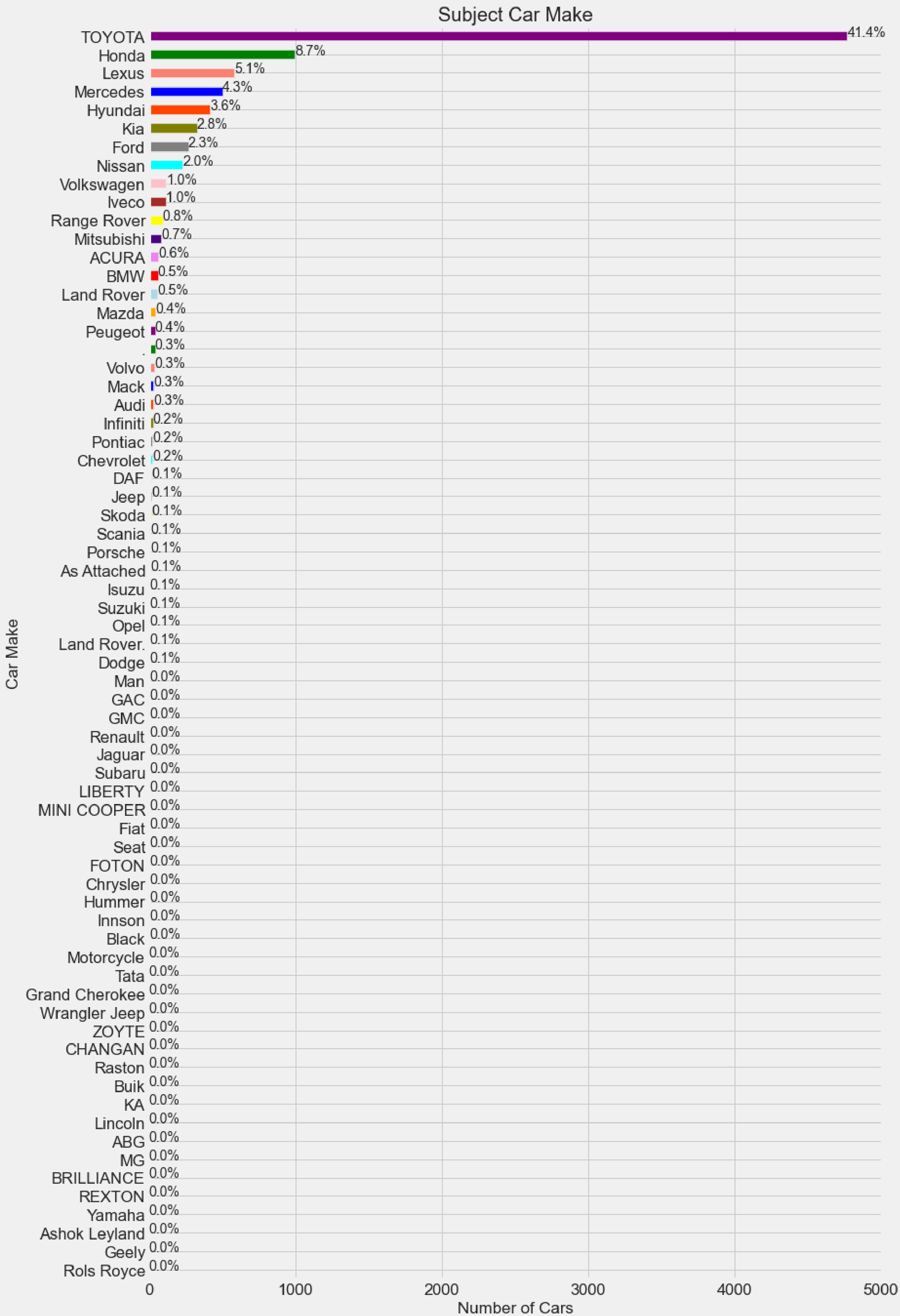
Out[372…

target	0	1
Subject_Car_Make		
.	27	13
ABG	1	0
ACURA	57	7
As Attached	5	5
Ashok Leyland	1	0
...	...	...
Volkswagen	112	5
Volvo	28	8
Wrangler Jeep	1	0
Yamaha	1	0
ZOYTE	1	0

68 rows × 2 columns

In [372…

```
# Subject car make Distribution before joining
ax = train.Subject_Car_Make.value_counts().sort_values().plot(kind='barh',
                                                              figsize=(12, 22),
                                                              color=['blue', 'salmon', 'gree
                                                                     'orange','lightblue', 'r
                                                                     'indigo', 'yellow', 'bro
                                                                     'cyan', 'gray', 'olive',


# Add some attributes
plt.title('Subject Car Make', fontdict={'size': 20})
plt.xlabel('Number of Cars')
plt.ylabel('Car Make')

for p in ax.patches:
    percentage = '{:,.1f}%'.format((p.get_width()/train.shape[0]) * 100)
    width, height = p.get_width(), p.get_height()
    x = p.get_x() + width + 0.02
    y = p.get_y() + height/2
    ax.annotate(percentage, (x, y));
```

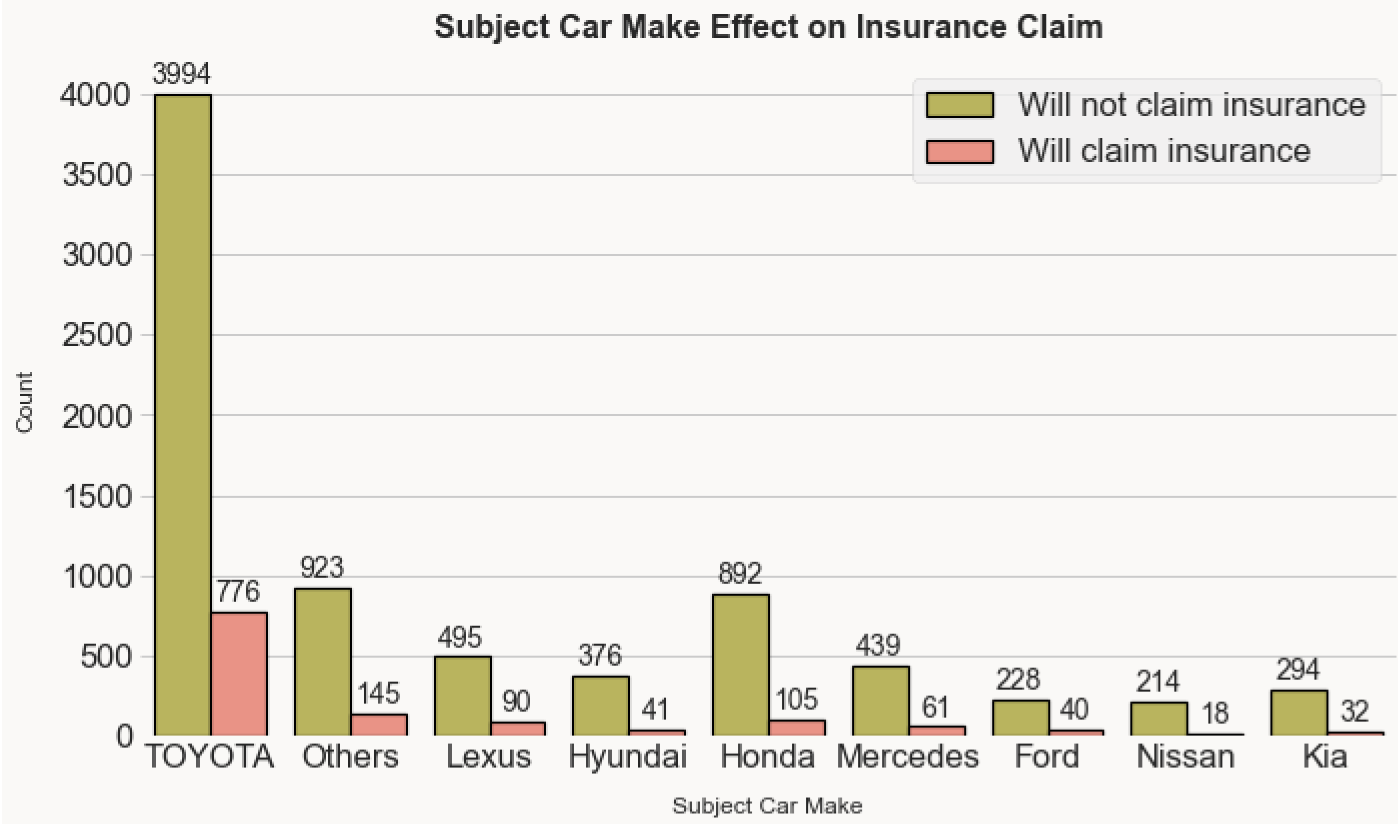


Subject Car Make
TOYOTA
Honda
Lexus
Mercedes
Hyundai
Kia
Ford
Nissan
Volkswagen
Iveco
Range Rover
Mitsubishi
ACURA
BMW
Land Rover
Mazda
Peugeot
Volvo
Mack
Audi
Infiniti
Pontiac
Chevrolet
DAF
Jeep
Skoda
Scania
Porsche
As Attached
Isuzu
Suzuki
Opel
Land Rover.
Dodge
Man
GAC
GMC
Renault
Jaguar
Subaru
LIBERTY
MINI COOPER
Fiat
Seat
FOTON
Chrysler
Hummer
Innson
Black
Motorcycle
Tata
Grand Cherokee
Wrangler Jeep
ZOYTE
CHANGAN
Raston
Buik
KA
Lincoln
ABG
MG
BRILLIANCE
REXTON
Yamaha
Ashok Leyland
Geely
Rols Royce
41.4%
8.7%
5.1%
4.3%
3.6%
2.8%
2.3%
2.0%
1.0%
1.0%
0.8%
0.7%
0.6%
0.5%
0.5%
0.4%
0.4%
0.3%
0.3%
0.3%
0.3%
0.2%
0.2%
0.2%
0.1%
0.0%
Car Make
0
1000
2000
3000
4000
5000
Number of Cars

In [372…
```
# Replace subject car make that are less than 200 with others
mapper = {'Range Rover':'Others', 'Mitsubishi':'Others', 'ACURA':'Others', 'BMW': 'Others'
          'Peugeot':'Others', '.': 'Others', 'Volvo': 'Others', 'Mack': 'Others', 'Audi':
          'Pontiac': 'Others', 'Chevrolet': 'Others', 'DAF': 'Others', 'Skoda': 'Others',
          'Porsche': 'Others', 'As Attached': 'Others', 'Scania':'Others', 'Suzuki':'Other
          'Isuzu': 'Others', 'Dodge':"Others", 'Renault': 'Others', 'Land Rover.':'Others'
          'GAC': 'Others', 'Man':"Others", 'LIBERTY': 'Others', 'MINI COOPER': 'Others', '
          'Hummer': 'Others', 'Chrysler': 'Others', 'Fiat': 'Others', 'Grand Cherokee': '(
          'Seat': 'Others', 'Black': 'Others', 'FOTON':'Others', 'Datsun': 'Others', 'Gee
          'REXTON': 'Others', 'ZOYTE': 'Others', 'CHANGAN':'Others', 'BRILLIANCE': 'Others
          'COMMANDER':'Others', 'Jincheng':'Others', 'Caddillac':'Others', 'Buik':'Others'
          'Howo': 'Others', 'Lincoln':'Others', 'Tata':'Others', 'Ashok Leyland':'Others',
          'Volkswagen':'Others', 'Iveco': 'Others'}
train.Subject_Car_Make = train.Subject_Car_Make.replace(mapper)

# Confirm mappings
train.Subject_Car_Make.value_counts(normalize=True)
```

Out[372…
```
TOYOTA      0.520572
Others      0.116556
Honda       0.108807
Lexus       0.063844
Mercedes    0.054567
Hyundai     0.045509
Kia         0.035578
Ford        0.029248
Nissan      0.025319
Name: Subject_Car_Make, dtype: float64
```

In [372…
```
# Plot a barchart of the different car make and check the number that claimed
# insurance and those that did not claim insurance after joining.
fig, ax = plt.subplots(figsize=(10,6))
fig.patch.set_facecolor('#faf9f7')
ax.set_facecolor('#faf9f7')

bar_pal = ["#c8c14f", "#fa8775"]

s = sns.countplot(
    data = train, x = 'Subject_Car_Make', hue = 'target', palette = bar_pal,
    linewidth = 1.2, ec = 'black'
)

for i in ['top', 'right', 'bottom', 'left']:
    ax.spines[i].set_visible(False)

plt.legend(['Will not claim insurance', 'Will claim insurance'])
plt.title("Subject Car Make Effect on Insurance Claim", size = 16, weight = 'bold', pad =
plt.xlabel('Subject Car Make', size = 12, labelpad = 12)
plt.ylabel('Count', size = 12, labelpad = 12)

for i in s.patches:
    s.annotate(format(i.get_height(), '.0f'),  (i.get_x() + i.get_width() / 2., i.get_heig
               va = 'center', xytext = (0, 9), textcoords = 'offset points')

fig.tight_layout()
```



Observation

Out of the 4770 toyota car owners, 3994 toyota car owners will not claim insurance within the first three months from their first transaction while 776 will claim insurance within the first three months of their first transaction. For the others vehicle category, there are a total of 1068 car owners of differenct car types, out of the 1068 others car make, 923 will not claim insurance while 145 will claim insurance within the first three months of their first transaction.Of the 585 lexus owners, 495 will not claim insurance while 90 lexus owners will claim insurance within the first three months of their first transaction. Out of the 417 hyundai car owners, 376 will not claim insurance while 41 hyundai car owners will claim insurance. There are a total of 997 honda car owners, out of the 997 honda owners, 892 will not claim insurance while 105 will claim insurance. Out of the 500 mercedes benz owners, 439 will not claim insurance within the first three months of their first transaction while 61 will claim insurance within the first three months of their first trasaction. Out of the 268 ford owners, 228 will not claim insurance within the first three months of their first transaction while 40 will claim insurance within the first three months of their first transaction. Of the 232 nissan car owners, 214 will not claim insurance within the first three months of their first transaction while 18 will claim insurance within the first three months of their first transaction. There are a total of 326 kia car owners, 294 will not claim insurance within the first three months while 32 will claim insurance within the first three months.

STATE

In [372…

```
# Check the proportion of state
train.State.value_counts(normalize=True)
```

Out[372…

```
Lagos               0.567426
Benue               0.114120
Eti-Osa             0.040157
Abuja-Municipal     0.035487
Ibeju-Lekki         0.022600
                      ...
Ogba-Ndoni          0.000187
Essien-Udim         0.000187
Ughelli-North       0.000187
```

```
Asari-Toru          0.000187
Ovia-SouthWest      0.000187
Name: State, Length: 111, dtype: float64
```

In [373…

```
# Compare the target column to State
pd.crosstab(train.State, train.target)
```

Out[373…

target	0	1
State		
ABULE-EGBA	5	1
AJAO-ESTATE	2	0
Aba-North	3	0
Aba-South	1	0
Abia	2	0
...	...	...
Ughelli-North	1	0
Umuahia-South	2	0
Warri-Central	23	4
Warri-North	3	0
Warri-South	2	0

111 rows × 2 columns

In [373…

```
# State Distribution before joining
ax = train.State.value_counts().sort_values().plot(kind='barh',
                                                            figsize=(12, 22),
                                                            color=['blue', 'salmon', 'gree
                                                                   'orange','lightblue', 'r
                                                                   'indigo', 'yellow', 'bro
                                                                   'cyan', 'gray', 'olive',


# Add some attributes
plt.title('States', fontdict={'size': 20})
plt.xlabel('Number of Cars')
plt.ylabel('Car Make')

for p in ax.patches:
    percentage = '{:,.1f}%'.format((p.get_width()/train.shape[0]) * 100)
    width, height = p.get_width(), p.get_height()
    x = p.get_x() + width + 0.02
    y = p.get_y() + height/2
    ax.annotate(percentage,(x, y));
```


States
Car Make
Number of Cars
Lagos
Benue
Eti-Osa
Abuja-Municipal
Ibeju-Lekki
Ogun
Oshodi-Isolo
Obia-Akpor
Delta
Abuja
Edo
Rivers
Amuwo-Odofin
Ibadan-West
Port-Harcourt
ENUGU-EAST
Ifako-Ijaye
Imo
Warri-Central
Oyo
Kaduna-North
Osun
Anambra
Akwa-Ibom
Niger-State
Ife-Central
Ondo
Owerri-Municipal
Jos-North
Kaduna
Kaduna-South
Jos-South
Anambra-East
Enugu-North
Ijebu-Ode
Ogbmosho-South
Ibadan-East
Lagelu-North
ABULE-EGBA
Ajeromi-Ifelodun
Etsako-West
Awka-South
Kano-Municipal
Nnewi-North
Ibadan-North
Ogun-Waterside
Nsit-Ubium
Aniocha-South
Ado-Ota
Obafemi-Owode
Cross-River
Udi-Agwu
Esan-West
Aba-North
Ife-North
Onitsha-North
Kwara
Warri-North
Ilesha-West
Orile-Iganmu
Ekiti-East
Owerri-West
Esan-Central
Warri-South
Ilorin-East
Ondo-West
Central-Abuja
Ibarapa-Central
Ilorin-West
Aboh-Mbaise
Umuahia-South
Abia
AJAO-ESTATE
Kogi
Ile-Oluji
Calabar-Municipality
Ebonyi
Gombe
Ado-Ekiti
Idemili-North
Oyo-East
N-A
Bayelsa
Nasarawa
ENUGU-SOUTH
Nnewi-South
Ajegunle-State
Kebbi
Asari-Toru
Kano
Isoko-south
Ogba-Ndoni
Essien-Udim
Ughelli-North
Ethiope-East
Ndokwa-East
Onitsha-South
Ijebu-East
Ilesha-East
Aba-South
Akoko-West
Oshimili-North
Bauchi
QuaAn-Pan
Ekiti-West
Awka-North
Oyo-West
Isoko-North
Ijebu-North
Owerri-North
Ovia-SouthWest
26.4%
5.3%
1.9%
1.7%
1.1%
0.8%
0.8%
0.7%
0.7%
0.6%
0.5%
0.5%
0.4%
0.4%
0.3%
0.3%
0.3%
0.2%
0.2%
0.2%
0.2%
0.2%
0.1%
0.0%
0
500
1000
1500
2000
2500
3000

In [373…

```python
# Replace local governments that are in each states with the actual states
mapper = {'Abuja-Municipal':'Abuja', 'Eti-Osa':'Lagos', 'Ibeju-Lekki':'Lagos', 'Obia-Akpor
          'Ibadan-West':'Oyo', 'Port-Harcourt': 'Rivers', 'Ifako-Ijaye': 'Lagos', 'ENUGU-E
          'Ife-Central': 'Osun', 'Jos-North': 'Plateau', 'Owerri-Municipal': 'Imo', 'Ijebu
          'Kaduna-South': 'Kaduna', 'Anambra-East': 'Anambra', 'Enugu-North':'Enugu', 'Lag
          'Ibadan-East': 'Oyo', 'Nnewi-North':"Anambra", 'Ibadan-North': 'Oyo', 'Etsako-We
          'Kano-Municipal': 'Kano', 'Awka-South':"Anambra", 'Obafemi-Owode': 'Ogun', 'Iles
          'Aniocha-South': 'Delta', 'Onitsha-North': 'Anambra', 'Ado-Ota': 'Ogun', 'Ogun-W
          'Warri-North': 'Delta', 'Ilorin-West': 'Kwara', 'Udi-Agwu':'Enugu', 'AJAO-ESTATE
          'Ondo-West': 'Ondo', 'Ilorin-East': 'Kwara', 'Idemili-North':'Anambra', 'Ekiti-E
          'Aboh-Mbaise':'Imo', 'Ado-Ekiti':'Ekiti', 'Central-Abuja':'Abuja', 'Ibarapa-Cent
          'Calabar-Municipality': 'Cross-River', 'Umuahia-South':'Abia', 'QuaAn-Pan':'Plat
          'Ekiti-West':'Ekiti', 'Ughelli-North': 'Delta', 'Isoko-North': 'Delta', 'Asari-T
          'Oyo-West': 'Oyo', 'Oshimili-North':"Delta", 'Ngor-Okpala': 'Imo', 'Ilesha-East'
          'Owerri-North': 'Imo', 'Ajegunle-State':"Lagos", 'Isoko-south': 'Delta', 'Akoko-
          'Ijebu-North': 'Ogun', 'N-A': 'Kano', 'Ovia-SouthWest': 'Edo', 'Akwa Ibom': 'Akw
train.State = train.State.replace(mapper)

# Confirm mappings
train.State.value_counts()
```

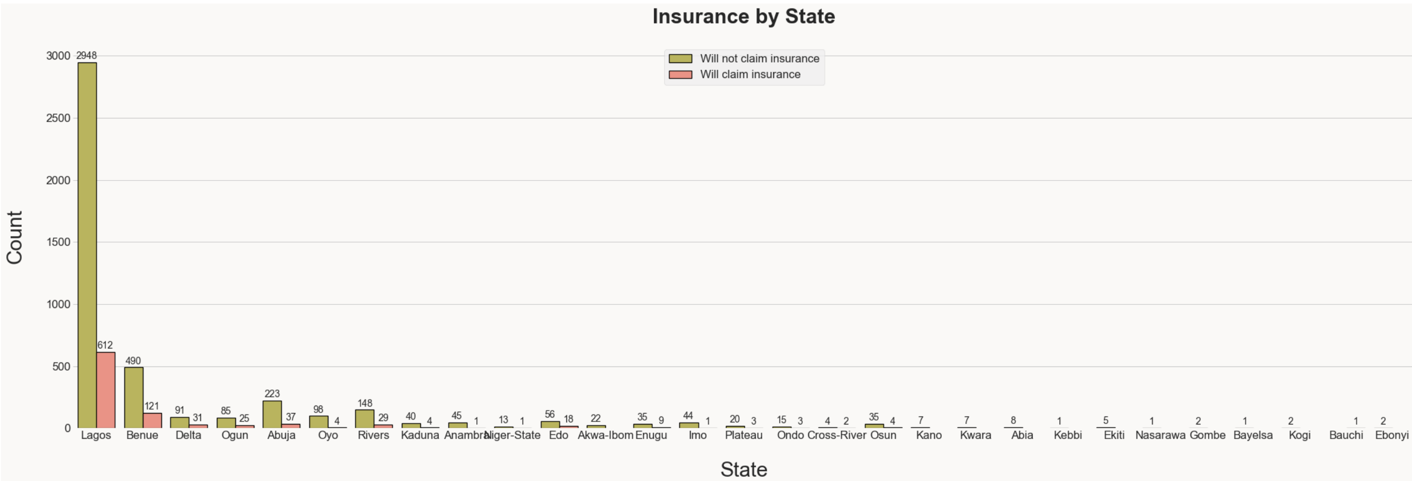
Out[373…

```
Lagos          3560
Benue           611
Abuja           260
Rivers          177
Delta           122
Ogun            110
Oyo             102
Edo              74
Anambra          46
Imo              45
Kaduna           44
Enugu            44
Osun             39
Plateau          23
Akwa-Ibom        22
Ondo             18
Niger-State      14
Abia              8
Kano              7
Kwara             7
Cross-River       6
Ekiti             5
Ebonyi            2
Kogi              2
Gombe             2
Kebbi             1
Bauchi            1
Nasarawa          1
Bayelsa           1
Name: State, dtype: int64
```

In [373…

```python
# Plot a barchart of the different states and check the number that claimed
# insurance and those that did not claim insurance after joining.
fig, ax = plt.subplots(figsize=(29,10))
fig.patch.set_facecolor('#faf9f7')
ax.set_facecolor('#faf9f7')

bar_pal = ["#c8c14f", "#fa8775"]

s = sns.countplot(
    data = train, x = 'State', hue = 'target', palette = bar_pal,
    linewidth = 1.2, ec = 'black'
)
```

```
for i in ['top', 'right', 'bottom', 'left']:
    ax.spines[i].set_visible(False)

plt.legend(['Will not claim insurance', 'Will claim insurance'], loc = 'upper center')
plt.title("Insurance by State", size = 30, weight = 'bold', pad = 30)
plt.xlabel('State', size = 30, labelpad = 30)
plt.ylabel('Count', size = 30, labelpad = 30)

for i in s.patches:
    s.annotate(format(i.get_height(), '.0f'),  (i.get_x() + i.get_width() / 2., i.get_heig
               va = 'center', xytext = (0, 9), textcoords = 'offset points')

fig.tight_layout()
```



LGA NAMES

In [373…

```
train.LGA_Name.value_counts()
```

Out[373…

```
Victoria Island          1098
Ikeja                     391
Surulere                  279
Lagos Mainland            216
Eti-Osa                   215
                         ... 
Oyo West                    1
Isoko south                 1
Ajegunle,Lagos  State       1
Isoko North                 1
Kuje                        1
Name: LGA_Name, Length: 256, dtype: int64
```

In [373…

```
# Compare the target column to State
pd.crosstab(train.LGA_Name, train.target)
```

Out[373…

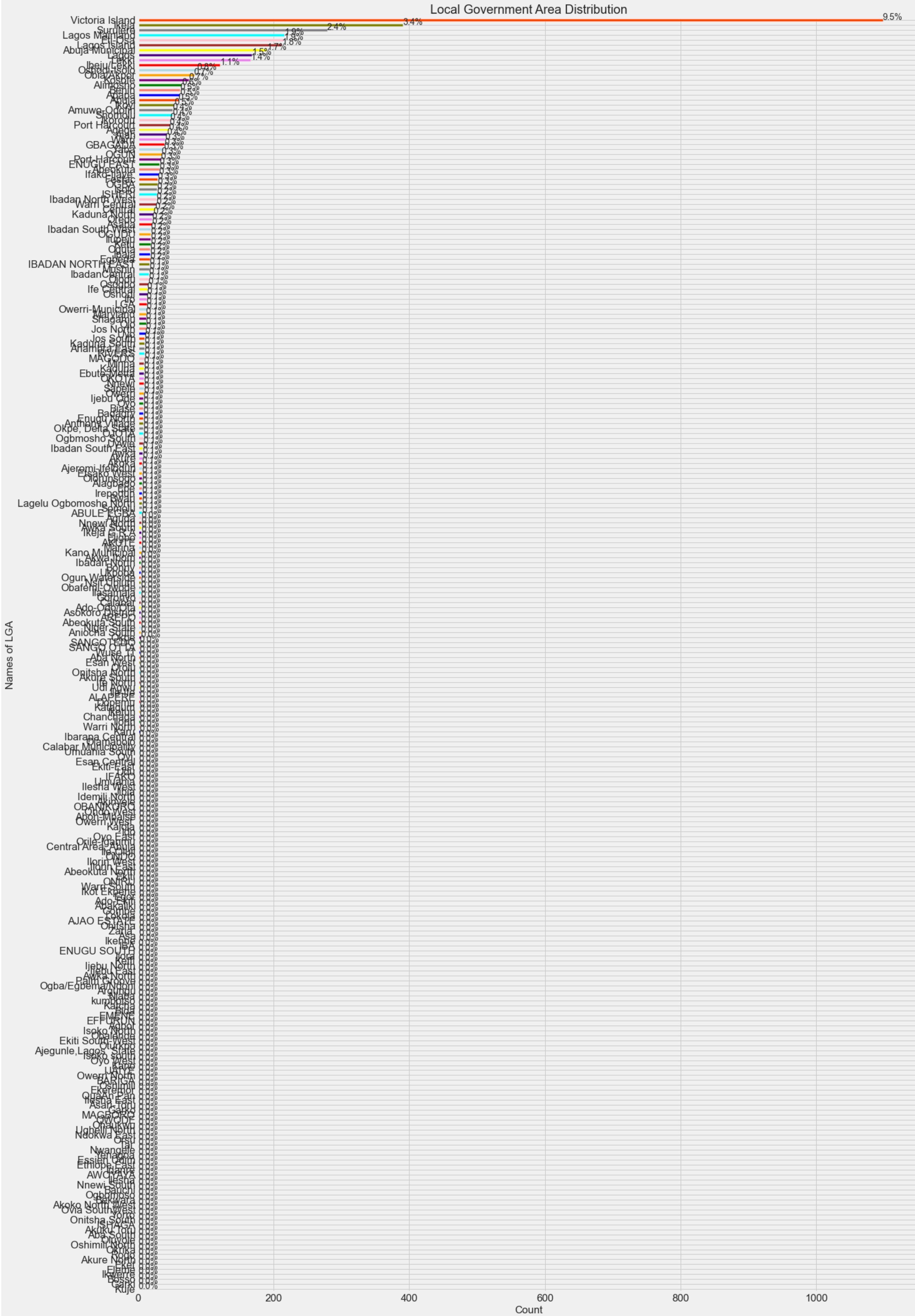
target	0	1
LGA_Name		
IFAKO	1	1
ABULE EGBA	5	1
AJAO ESTATE	2	0
AKUTE	2	3
ALAPERE	3	0
...	...	...

target	0	1
LGA_Name		
Yaba	34	4
Yenagoa	1	0
Yorro	1	0
Zaria	2	0
kumbotso	1	0

256 rows × 2 columns

In [373…

```
# LGA Names Distribution
ax = train.LGA_Name.value_counts().sort_values().plot(kind='barh',
                                                       figsize=(20, 35),
                                                       color=['blue', 'salmon', 'gree
                                                              'orange','lightblue', 'r
                                                              'indigo', 'yellow', 'bro
                                                              'cyan', 'gray', 'olive',

# Add some attributes
plt.title('Local Government Area Distribution', fontdict={'size': 20})
plt.xlabel('Count')
plt.ylabel('Names of LGA')

for p in ax.patches:
    percentage = '{:,.1f}%'.format((p.get_width()/train.shape[0]) * 100)
    width, height = p.get_width(), p.get_height()
    x = p.get_x() + width + 0.02
    y = p.get_y() + height/2
    ax.annotate(percentage,(x, y));
```



Product Name

In [373…

```
train.ProductName.value_counts()
```

Out[373…

```
Car Classic         6297
CarSafe             3524
Customized Motor     498
Car Plus             469
CVTP                 412
CarFlex              176
Muuve                 92
Motor Cycle           38
Car Vintage            5
Name: ProductName, dtype: int64
```

In [374…

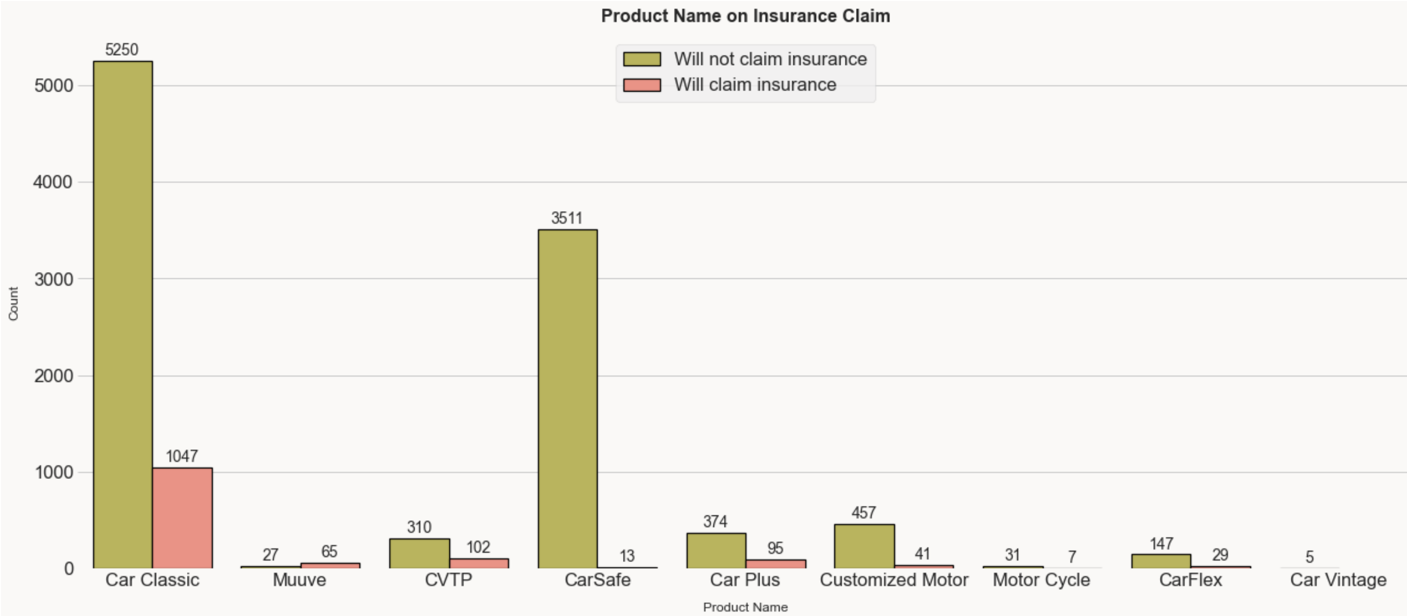
```
# Compare the target column to State
pd.crosstab(train.ProductName, train.target)
```

Out[374…

target	0	1
ProductName		
CVTP	310	102
Car Classic	5250	1047
Car Plus	374	95
Car Vintage	5	0
CarFlex	147	29
CarSafe	3511	13
Customized Motor	457	41
Motor Cycle	31	7
Muuve	27	65

In [374…

```
# Plot a barchart of the different product name and check the number that claimed
# insurance and those that did not claim insurance.
fig, ax = plt.subplots(figsize=(18,8))
fig.patch.set_facecolor('#faf9f7')
ax.set_facecolor('#faf9f7')

bar_pal = ["#c8c14f", "#fa8775"]

s = sns.countplot(
    data = train, x = 'ProductName', hue = 'target', palette = bar_pal,
    linewidth = 1.2, ec = 'black'
)

for i in ['top', 'right', 'bottom', 'left']:
    ax.spines[i].set_visible(False)

plt.legend(['Will not claim insurance', 'Will claim insurance'])
plt.title("Product Name on Insurance Claim", size = 16, weight = 'bold', pad = 12)
plt.xlabel('Product Name', size = 12, labelpad = 12)
plt.ylabel('Count', size = 12, labelpad = 12)

for i in s.patches:
    s.annotate(format(i.get_height(), '.0f'),  (i.get_x() + i.get_width() / 2., i.get_heig
               va = 'center', xytext = (0, 9), textcoords = 'offset points')

fig.tight_layout()
```

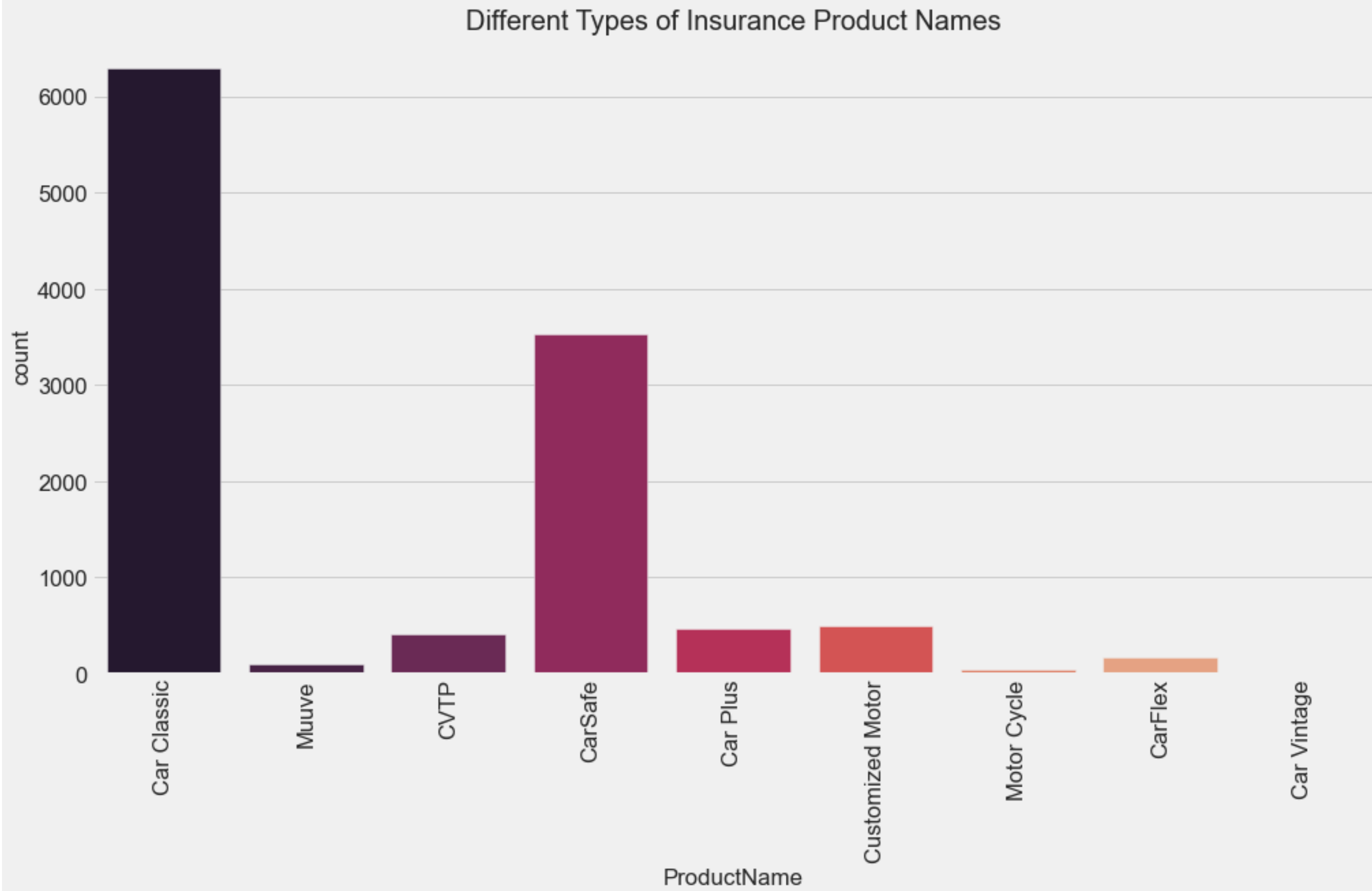


Observation

Out of the 6297 customers that regstered for Car Classic insurance policy, 5250 Car Classic Customers will not claim insurance within the first three months from their first transaction while 1047 will claim insurance within the first three months of their first transaction. For the Muuve insurance policy, there are a total of 92 customers that registered for the Muuve insurance, out of the 92 Muuve customers, 27 will not claim insurance while 65 will claim insurance within the first three months of their first transaction.Of the 412 customers that registered for CVTP insurance policy, 310 will not claim insurance while 102 customers will claim insurance within the first three months of their first transaction. Out of the 3524 customers that registered for CarSafe insurance policy, 3511 will not claim insurance while 13 CarSafe insurance policy holders will claim insurance. There are a total of 469 CarPlus insurance policy holders, out of the 469 CarPlus insurance policy holders, 374 will not claim insurance while 95 will claim insurance. Out of the 489 Customized motor insurance policy holders, 457 customized motor insurance policy holders will not claim insurance within the first three months of their first transaction while 41 will claim insurance within the first three months of their first trasaction. Out of the 28 motorcycle insurance policy holders, 31 will not claim insurance within the first three months of their first transaction while 7 will claim insurance within the first three months of their first transaction. Of the 176 CarFlex insurance policy holders, 147 will not claim insurance within the first three months of their first transaction while 29 will claim insurance within the first three months of their first transaction. There are a total of 5 Car Vintage insurance policy holders, all 5 of them will not claim insurance within the first three months.

In [374…

```
# let's visualize the different product names

plt.style.use('fivethirtyeight')
plt.rcParams['figure.figsize'] = (15, 8)

sns.countplot(train['ProductName'], palette = 'rocket')
plt.title('Different Types of Insurance Product Names', fontsize = 20)
plt.xticks(rotation = 90)
plt.show()
```



In [374…

```
# Let's see how many positive (1) and negative (0) samples we have in our dataframe
print('Length of the training dataset:', len(train))
print('Total no of customers that will not claim insurance in the first 3 months:', len(tr
print('Total no of customers that will claim insurance in the first 3 months:', len(train
```

```
Length of the training dataset: 11511
Total no of customers that will not claim insurance in the first 3 months: 10112
Total no of customers that will claim insurance in the first 3 months: 1399
```

In [374…

```
# Check the shape of the training dataset
print(f'The shape of the training dataset is: {train.shape}')
```

```
The shape of the training dataset is: (11511, 14)
```

In [374…

```
# Check the proportion of the classes in the target column
train['target'].value_counts(normalize=True)
```

Out[374…

```
0    0.878464
1    0.121536
Name: target, dtype: float64
```

In [374…

```
# Plot the target value counts with a bar graph
train.target.value_counts().plot(kind='bar', title = 'Target Class', color=['red', 'blue']
plt.xlabel('0=Will not claim insurance in 3 months, 1=Will claim insurance in 3 months')
plt.ylabel('Frequency')
plt.xticks(rotation=0);
```

In [374…

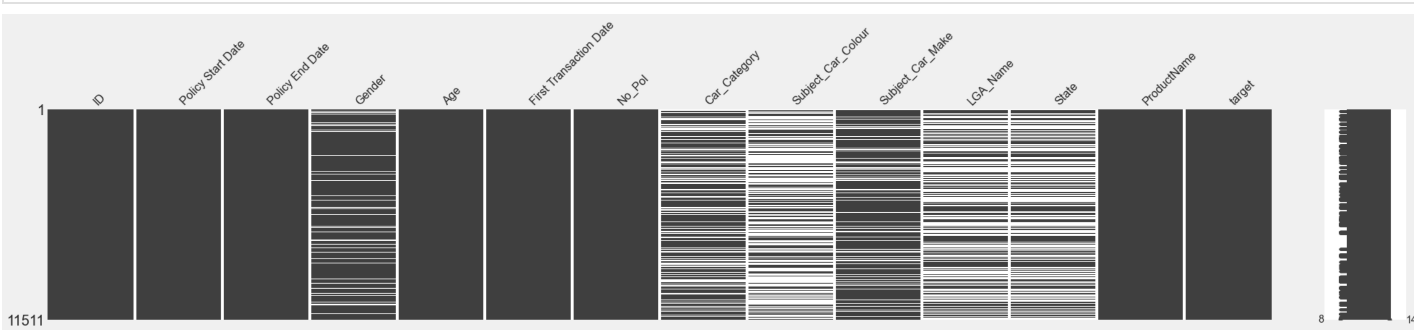
```
# Check if there are any missing values
train.isna().sum()
```

Out[374…

```
ID                        0
Policy Start Date         0
Policy End Date           0
Gender                  996
Age                       0
First Transaction Date    0
No_Pol                    0
Car_Category           3529
Subject_Car_Colour     6568
Subject_Car_Make       2348
LGA_Name               6145
State                  6157
ProductName               0
target                    0
dtype: int64
```

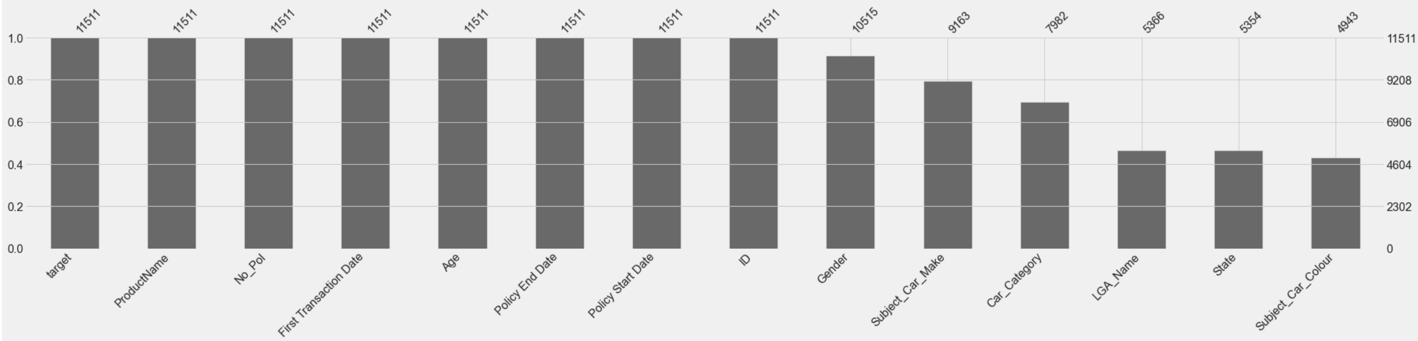
In [374…

```
# Visualizing the missing values in the dataset
missingno.matrix(train, figsize=(30, 5));
```



In [375…

```
# Visualize the missing values in descending order
missingno.bar(train, sort='descending', figsize=(30, 5));
```



In [375…

```
# check dtype of "Policy_Start_Date, Policy_End_Date, First_Transaction_Date"
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11511 entries, 0 to 12078
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   ID                      11511 non-null  object
 1   Policy Start Date       11511 non-null  datetime64[ns]
 2   Policy End Date         11511 non-null  datetime64[ns]
 3   Gender                  10515 non-null  object
 4   Age                     11511 non-null  int64
 5   First Transaction Date  11511 non-null  datetime64[ns]
 6   No_Pol                  11511 non-null  int64
 7   Car_Category            7982 non-null   object
 8   Subject_Car_Colour      4943 non-null   object
 9   Subject_Car_Make        9163 non-null   object
 10  LGA_Name                5366 non-null   object
 11  State                   5354 non-null   object
 12  ProductName             11511 non-null  object
 13  target                  11511 non-null  int64
dtypes: datetime64[ns](3), int64(3), object(8)
memory usage: 1.6+ MB
```

We've turned the `PolicyStartDate`, `PolicyEndDate`, `FirstTransactionDate` column from object datatype to datetime64 datatype.

In [375…

```
train.head().T
```

Out[375…

	0	1	2	4	5
ID	ID_0040R73	ID_0046BNK	ID_005QMC3	ID_00BRP63	ID_00D3EF6
Policy Start Date	2010-05-14 00:00:00	2010-11-29 00:00:00	2010-03-21 00:00:00	2010-08-29 00:00:00	2010-10-21 00:00:00
Policy End Date	2011-05-13 00:00:00	2011-11-28 00:00:00	2011-03-20 00:00:00	2010-12-31 00:00:00	2011-10-20 00:00:00
Gender	Male	Female	Male	NaN	Male
Age	30	79	43	20	37
First Transaction Date	2010-05-14 00:00:00	2010-11-29 00:00:00	2010-03-21 00:00:00	2010-08-29 00:00:00	2010-10-21 00:00:00
No_Pol	1	1	1	3	2
Car_Category	Saloon	JEEP	Saloon	NaN	NaN
Subject_Car_Colour	Black	Grey	Red	NaN	NaN
Subject_Car_Make	TOYOTA	TOYOTA	TOYOTA	NaN	NaN

	0	1	2	4	5
LGA_Name	NaN	NaN	NaN	Lagos	NaN
State	NaN	NaN	NaN	Lagos	NaN
ProductName	Car Classic	Car Classic	Car Classic	Muuve	Car Classic
target	0	1	0	1	0

4. Feature Engineering

Engineering: There are multiple techniques for feature engineering:

- **Decompose**: Converting 2014-09-20T20:45:40Z into categorical attributes like hour_of_the_day, part_of_day, etc.

Imputation: We can impute missing values in a number of different ways:

- **Hot-Deck**: The technique then finds the first missing value and uses the cell value immediately prior to the data that are missing to impute the missing value.
- **Cold-Deck**: Selects donors from another dataset to complete missing data.
- **Mean-substitution**: Another imputation technique involves replacing any missing value with the mean of that variable for all other cases, which has the benefit of not changing the sample mean for that variable.
- **Regression**: A regression model is estimated to predict observed values of a variable based on other variables, and that model is then used to impute values in cases where that variable is missing.

In [375…

```
train['Policy Start Date']
```

Out[375…

```
0        2010-05-14
1        2010-11-29
2        2010-03-21
4        2010-08-29
5        2010-10-21
            ...
12074    2010-05-25
12075    2010-10-03
12076    2010-10-10
12077    2010-02-27
12078    2010-07-01
Name: Policy Start Date, Length: 11511, dtype: datetime64[ns]
```

In [375…

```
train['Policy End Date']
```

Out[375…

```
0        2011-05-13
1        2011-11-28
2        2011-03-20
4        2010-12-31
5        2011-10-20
            ...
12074    2011-05-24
12075    2011-10-02
12076    2011-10-08
12077    2011-02-26
12078    2011-06-30
Name: Policy End Date, Length: 11511, dtype: datetime64[ns]
```

In [375…

```
train['First Transaction Date']
```

Out[375…
```
0        2010-05-14
1        2010-11-29
2        2010-03-21
4        2010-08-29
5        2010-10-21
            ...
12074    2010-05-25
12075    2010-10-03
12076    2010-10-10
12077    2010-02-27
12078    2010-07-01
Name: First Transaction Date, Length: 11511, dtype: datetime64[ns]
```

Sort DataFrame by Policy Start Date, Policy End Date and First Transaction Date

In [375…
```
# Sort DataFrame in date order
train.sort_values(by=['Policy Start Date'], inplace=True, ascending=True)
train['Policy Start Date'].head()
```

Out[375…
```
8010     2001-12-11
10526    2002-03-25
10234    2003-04-13
12066    2003-12-21
8124     2005-08-05
Name: Policy Start Date, dtype: datetime64[ns]
```

In [375…
```
# Sort DataFrame in date order
train.sort_values(by=['Policy End Date'], inplace=True, ascending=True)
train['Policy End Date'].head()
```

Out[375…
```
11738    2010-12-31
6996     2010-12-31
6050     2010-12-31
4778     2010-12-31
11110    2010-12-31
Name: Policy End Date, dtype: datetime64[ns]
```

In [375…
```
# Sort DataFrame in date order
train.sort_values(by=['First Transaction Date'], inplace=True, ascending=True)
train['First Transaction Date'].head()
```

Out[375…
```
8010     2001-12-11
10526    2002-03-25
10234    2003-04-13
12066    2003-12-21
8124     2005-08-05
Name: First Transaction Date, dtype: datetime64[ns]
```

In [375…
```
train.head()
```

Out[375…

	ID	Policy Start Date	Policy End Date	Gender	Age	First Transaction Date	No_Pol	Car_Category	Subject_Car_Colour	Subject_Ca
8010	ID_O51ZQ1B	2001-12-11	2011-12-10	Female	37	2001-12-11	1	Saloon	Black	
10526	ID_VJ1FAVO	2002-03-25	2011-03-24	Male	37	2002-03-25	1	Saloon	Black	

	ID	Policy Start Date	Policy End Date	Gender	Age	First Transaction Date	No_Pol	Car_Category	Subject_Car_Colour	Subject_Ca
10234	ID_ULWS8VL	2003-04-13	2011-04-12	Male	41	2003-04-13	2	Saloon	Black	
12066	ID_ZYKGSP7	2003-12-21	2034-05-20	Male	48	2003-12-21	2	Saloon	NaN	
8124	ID_OEWBKGF	2005-08-05	2011-09-29	Female	44	2005-08-05	1	NaN	NaN	

Add datetime parameters for PolicyStartDate, PolicyEndDate, FirstTransactionDate column

Why?

So we can enrich our dataset with as much information as possible.

Because we imported the data using `train_csv()` and we asked pandas to parse the dates using parse_dates=[`'PolicyStartDate'` , `'PolicyEndDate'` , `'FirstTransactionDate'`]), we can now access the different datetime attributes of the date column.

In [376…

```
# Add datetime for Policy Start Date
train['PolicyStartYear'] = train['Policy Start Date'].dt.year
train['PolicyStartMonth'] = train['Policy Start Date'].dt.month
train['PolicyStartDay'] = train['Policy Start Date'].dt.day
train['PolicyStartDayofweek'] = train['Policy Start Date'].dt.dayofweek
train['PolicyStartDayofyear'] = train['Policy Start Date'].dt.dayofyear

# Drop original PolicyStartDate
train.drop("Policy Start Date", axis=1, inplace=True)
```

In [376…

```
# Add datetime for Policy End Date
train['PolicyEndYear'] = train['Policy End Date'].dt.year
train['PolicyEndMonth'] = train['Policy End Date'].dt.month
train['PolicyEndDay'] = train['Policy End Date'].dt.day
train['PolicyEndDayofweek'] = train['Policy End Date'].dt.dayofweek
train['PolicyEndDayofyear'] = train['Policy End Date'].dt.dayofyear

# Drop original PolicyEndDate
train.drop("Policy End Date", axis=1, inplace=True)
```

In [376…

```
# Add datetime for FirstTransactionDate
train['FirstTransactionYear'] = train['First Transaction Date'].dt.year
train['FirstTransactionMonth'] = train['First Transaction Date'].dt.month
train['FirstTransactionDay'] = train['First Transaction Date'].dt.day
train['FirstTransactionDayofweek'] = train['First Transaction Date'].dt.dayofweek
train['FirstTransactionDayofyear'] = train['First Transaction Date'].dt.dayofyear

# Drop original FirstTransactionDate
train.drop("First Transaction Date", axis=1, inplace=True)
```

In [376…

```
train.reset_index(drop=True)
```

Out[376…

	ID	Gender	Age	No_Pol	Car_Category	Subject_Car_Colour	Subject_Car_Make	LGA_Name	State

	ID	Gender	Age	No_Pol	Car_Category	Subject_Car_Colour	Subject_Car_Make	LGA_Name	State
0	ID_O51ZQ1B	Female	37	1	Saloon	Black	Honda	NaN	NaN
1	ID_VJ1FAVO	Male	37	1	Saloon	Black	TOYOTA	Ekiti	Benue
2	ID_ULWS8VL	Male	41	2	Saloon	Black	TOYOTA	Ikeja	Lagos
3	ID_ZYKGSP7	Male	48	2	Saloon	NaN	Others	NaN	NaN
4	ID_OEWBKGF	Female	44	1	NaN	NaN	Others	Ajah	Lagos
...	...	...	...	...	...	...	...	...	...
11506	ID_SAFB882	Male	48	1	NaN	NaN	NaN	NaN	NaN
11507	ID_S6CWED4	Male	37	1	NaN	NaN	NaN	NaN	NaN
11508	ID_ZMXI8LN	Male	36	1	NaN	NaN	NaN	NaN	NaN
11509	ID_85P2ABI	Male	66	1	Saloon	NaN	TOYOTA	NaN	NaN
11510	ID_MLGO8DZ	Male	51	4	Saloon	Black	Honda	Victoria Island	Lagos

11511 rows × 26 columns

In [376…

```
train.head().T
```

Out[376…

	8010	10526	10234	12066	8124
ID	ID_O51ZQ1B	ID_VJ1FAVO	ID_ULWS8VL	ID_ZYKGSP7	ID_OEWBKGF
Gender	Female	Male	Male	Male	Female
Age	37	37	41	48	44
No_Pol	1	1	2	2	1
Car_Category	Saloon	Saloon	Saloon	Saloon	NaN
Subject_Car_Colour	Black	Black	Black	NaN	NaN
Subject_Car_Make	Honda	TOYOTA	TOYOTA	Others	Others
LGA_Name	NaN	Ekiti	Ikeja	NaN	Ajah
State	NaN	Benue	Lagos	NaN	Lagos
ProductName	Car Vintage	Car Classic	Car Vintage	Car Vintage	CVTP
target	0	0	0	0	0
PolicyStartYear	2001	2002	2003	2003	2005
PolicyStartMonth	12	3	4	12	8
PolicyStartDay	11	25	13	21	5
PolicyStartDayofweek	1	0	6	6	4
PolicyStartDayofyear	345	84	103	355	217
PolicyEndYear	2011	2011	2011	2034	2011
PolicyEndMonth	12	3	4	5	9
PolicyEndDay	10	24	12	20	29
PolicyEndDayofweek	5	3	1	5	3

	8010	10526	10234	12066	8124
PolicyEndDayofyear	344	83	102	140	272
FirstTransactionYear	2001	2002	2003	2003	2005
FirstTransactionMonth	12	3	4	12	8
FirstTransactionDay	11	25	13	21	5
FirstTransactionDayofweek	1	0	6	6	4
FirstTransactionDayofyear	345	84	103	355	217

In [376…

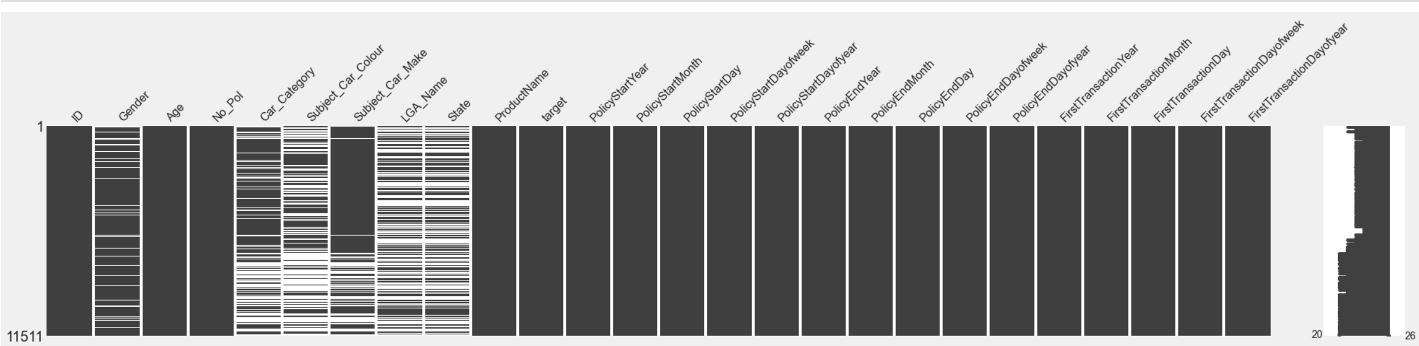
```
# Check for missing values
train.isna().sum()
```

Out[376…

```
ID                               0
Gender                         996
Age                              0
No_Pol                           0
Car_Category                  3529
Subject_Car_Colour            6568
Subject_Car_Make              2348
LGA_Name                      6145
State                         6157
ProductName                      0
target                           0
PolicyStartYear                  0
PolicyStartMonth                 0
PolicyStartDay                   0
PolicyStartDayofweek             0
PolicyStartDayofyear             0
PolicyEndYear                    0
PolicyEndMonth                   0
PolicyEndDay                     0
PolicyEndDayofweek               0
PolicyEndDayofyear               0
FirstTransactionYear             0
FirstTransactionMonth            0
FirstTransactionDay              0
FirstTransactionDayofweek        0
FirstTransactionDayofyear        0
dtype: int64
```

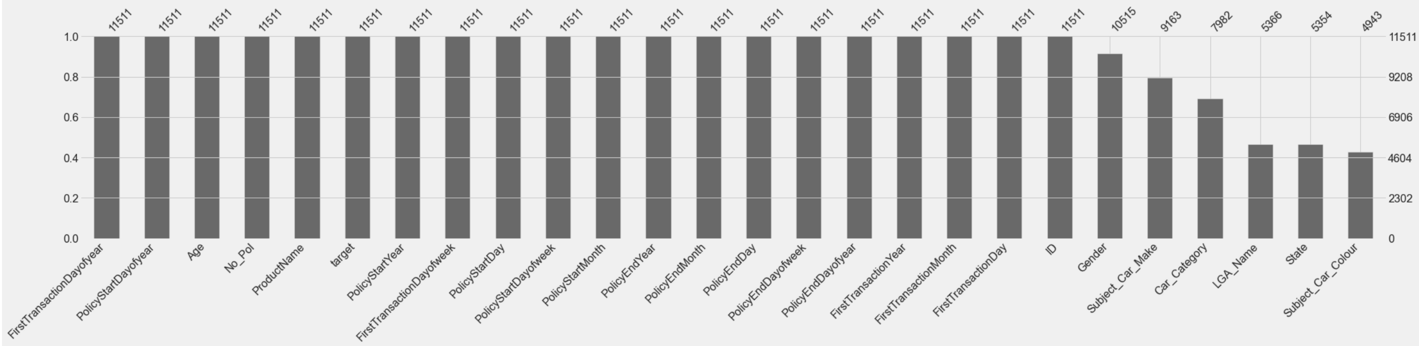
In [376…

```
# Visualizing the missing values in the dataset
missingno.matrix(train, figsize=(30, 5));
```



In [376…

```
# Visualize the missing values in the features in descending order
missingno.bar(train
              , sort='descending', figsize=(30, 5));
```



In [376…

```
# Check for missing categories and different datatypes
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11511 entries, 8010 to 7479
Data columns (total 26 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   ID                         11511 non-null  object
 1   Gender                     10515 non-null  object
 2   Age                        11511 non-null  int64 
 3   No_Pol                     11511 non-null  int64 
 4   Car_Category               7982 non-null   object
 5   Subject_Car_Colour         4943 non-null   object
 6   Subject_Car_Make           9163 non-null   object
 7   LGA_Name                   5366 non-null   object
 8   State                      5354 non-null   object
 9   ProductName                11511 non-null  object
 10  target                     11511 non-null  int64 
 11  PolicyStartYear            11511 non-null  int64 
 12  PolicyStartMonth           11511 non-null  int64 
 13  PolicyStartDay             11511 non-null  int64 
 14  PolicyStartDayofweek       11511 non-null  int64 
 15  PolicyStartDayofyear       11511 non-null  int64 
 16  PolicyEndYear              11511 non-null  int64 
 17  PolicyEndMonth             11511 non-null  int64 
 18  PolicyEndDay               11511 non-null  int64 
 19  PolicyEndDayofweek         11511 non-null  int64 
 20  PolicyEndDayofyear         11511 non-null  int64 
 21  FirstTransactionYear       11511 non-null  int64 
 22  FirstTransactionMonth      11511 non-null  int64 
 23  FirstTransactionDay        11511 non-null  int64 
 24  FirstTransactionDayofweek  11511 non-null  int64 
 25  FirstTransactionDayofyear  11511 non-null  int64 
dtypes: int64(18), object(8)
memory usage: 2.4+ MB
```

Convert the strings into categories

In [376…

```
# Find the columns which contains strings
for label, content in train.drop(['ID', 'ProductName'], axis=1).items():
    if pd.api.types.is_string_dtype(content):
        print(label)
```

```
Gender
Car_Category
Subject_Car_Colour
Subject_Car_Make
LGA_Name
State
```

In [377…

```python
# This will turn all strings values into categories
for label, content in train.items():
    if pd.api.types.is_string_dtype(content):
        train[label]=content.astype('category').cat.as_ordered()
```

In [377…

```python
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11511 entries, 8010 to 7479
Data columns (total 26 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          11511 non-null  category
 1   Gender                      10515 non-null  category
 2   Age                         11511 non-null  int64
 3   No_Pol                      11511 non-null  int64
 4   Car_Category                7982 non-null   category
 5   Subject_Car_Colour          4943 non-null   category
 6   Subject_Car_Make            9163 non-null   category
 7   LGA_Name                    5366 non-null   category
 8   State                       5354 non-null   category
 9   ProductName                 11511 non-null  category
 10  target                      11511 non-null  int64
 11  PolicyStartYear             11511 non-null  int64
 12  PolicyStartMonth            11511 non-null  int64
 13  PolicyStartDay              11511 non-null  int64
 14  PolicyStartDayofweek        11511 non-null  int64
 15  PolicyStartDayofyear        11511 non-null  int64
 16  PolicyEndYear               11511 non-null  int64
 17  PolicyEndMonth              11511 non-null  int64
 18  PolicyEndDay                11511 non-null  int64
 19  PolicyEndDayofweek          11511 non-null  int64
 20  PolicyEndDayofyear          11511 non-null  int64
 21  FirstTransactionYear        11511 non-null  int64
 22  FirstTransactionMonth       11511 non-null  int64
 23  FirstTransactionDay         11511 non-null  int64
 24  FirstTransactionDayofweek   11511 non-null  int64
 25  FirstTransactionDayofyear   11511 non-null  int64
dtypes: category(8), int64(18)
memory usage: 2.1 MB
```

All of our data is categorical and thus we can now turn the categories into integers, however it's still missing values....

In [377…

```python
# Check the proportion of missing values
train.isnull().sum()/len(train)
```

Out[377…

```
ID                             0.000000
Gender                         0.086526
Age                            0.000000
No_Pol                         0.000000
Car_Category                   0.306576
Subject_Car_Colour             0.570585
Subject_Car_Make               0.203979
LGA_Name                       0.533837
State                          0.534880
ProductName                    0.000000
target                         0.000000
PolicyStartYear                0.000000
PolicyStartMonth               0.000000
PolicyStartDay                 0.000000
PolicyStartDayofweek           0.000000
PolicyStartDayofyear           0.000000
```

```
PolicyEndYear                 0.000000
PolicyEndMonth                0.000000
PolicyEndDay                  0.000000
PolicyEndDayofweek            0.000000
PolicyEndDayofyear            0.000000
FirstTransactionYear          0.000000
FirstTransactionMonth         0.000000
FirstTransactionDay           0.000000
FirstTransactionDayofweek     0.000000
FirstTransactionDayofyear     0.000000
dtype: float64
```

Filling and turning categorical variables into numbers

In [377…

```
for label, content in train.items():
    if pd.api.types.is_categorical_dtype(content):
        print(label)
```

```
ID
Gender
Car_Category
Subject_Car_Colour
Subject_Car_Make
LGA_Name
State
ProductName
```

In [377…

```
# Check for which categorical columns have null(missing) values
for label, content in train.items():
    if pd.api.types.is_categorical_dtype(content):
        if pd.isnull(content).sum():
            print(label)
```

```
Gender
Car_Category
Subject_Car_Colour
Subject_Car_Make
LGA_Name
State
```

In [377…

```
# Turn categorical variables into numbers
for label, content in train.items():
    # Check columns which are not numeric
    if not pd.api.types.is_numeric_dtype(content):

            # Add binary column to indicate whether sample had missing value
         train[label + '_is_missing'] = pd.isnull(content)

            # Turn categories into numbers and add +1 because pandas encodes missing categ
         train[label] = pd.Categorical(content).codes + 1
```

In [377…

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11511 entries, 8010 to 7479
Data columns (total 34 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ID                        11511 non-null  int16
 1   Gender                    11511 non-null  int8
 2   Age                       11511 non-null  int64
 3   No_Pol                    11511 non-null  int64
```

```
 4   Car_Category                   11511 non-null  int8
 5   Subject_Car_Colour             11511 non-null  int8
 6   Subject_Car_Make               11511 non-null  int8
 7   LGA_Name                       11511 non-null  int16
 8   State                          11511 non-null  int8
 9   ProductName                    11511 non-null  int8
 10  target                         11511 non-null  int64
 11  PolicyStartYear                11511 non-null  int64
 12  PolicyStartMonth               11511 non-null  int64
 13  PolicyStartDay                 11511 non-null  int64
 14  PolicyStartDayofweek           11511 non-null  int64
 15  PolicyStartDayofyear           11511 non-null  int64
 16  PolicyEndYear                  11511 non-null  int64
 17  PolicyEndMonth                 11511 non-null  int64
 18  PolicyEndDay                   11511 non-null  int64
 19  PolicyEndDayofweek             11511 non-null  int64
 20  PolicyEndDayofyear             11511 non-null  int64
 21  FirstTransactionYear           11511 non-null  int64
 22  FirstTransactionMonth          11511 non-null  int64
 23  FirstTransactionDay            11511 non-null  int64
 24  FirstTransactionDayofweek      11511 non-null  int64
 25  FirstTransactionDayofyear      11511 non-null  int64
 26  ID_is_missing                  11511 non-null  bool
 27  Gender_is_missing              11511 non-null  bool
 28  Car_Category_is_missing        11511 non-null  bool
 29  Subject_Car_Colour_is_missing  11511 non-null  bool
 30  Subject_Car_Make_is_missing    11511 non-null  bool
 31  LGA_Name_is_missing            11511 non-null  bool
 32  State_is_missing               11511 non-null  bool
 33  ProductName_is_missing         11511 non-null  bool
dtypes: bool(8), int16(2), int64(18), int8(6)
memory usage: 1.9 MB
```

In [377…

```python
train.isna().sum()
```

Out[377…

```
ID                               0
Gender                           0
Age                              0
No_Pol                           0
Car_Category                     0
Subject_Car_Colour               0
Subject_Car_Make                 0
LGA_Name                         0
State                            0
ProductName                      0
target                           0
PolicyStartYear                  0
PolicyStartMonth                 0
PolicyStartDay                   0
PolicyStartDayofweek             0
PolicyStartDayofyear             0
PolicyEndYear                    0
PolicyEndMonth                   0
PolicyEndDay                     0
PolicyEndDayofweek               0
PolicyEndDayofyear               0
FirstTransactionYear             0
FirstTransactionMonth            0
FirstTransactionDay              0
FirstTransactionDayofweek        0
FirstTransactionDayofyear        0
ID_is_missing                    0
Gender_is_missing                0
Car_Category_is_missing          0
Subject_Car_Colour_is_missing    0
```

```
Subject_Car_Make_is_missing      0
LGA_Name_is_missing              0
State_is_missing                 0
ProductName_is_missing           0
dtype: int64
```

In [377…

```
train.head().T
```

Out[377…

	8010	10526	10234	12066	8124
ID	7623	10025	9745	11499	7732
Gender	1	2	2	2	1
Age	37	37	41	48	44
No_Pol	1	1	2	2	1
Car_Category	3	3	3	3	0
Subject_Car_Colour	5	5	5	0	0
Subject_Car_Make	2	9	9	8	8
LGA_Name	0	74	115	0	22
State	0	7	21	0	21
ProductName	4	2	4	4	1
target	0	0	0	0	0
PolicyStartYear	2001	2002	2003	2003	2005
PolicyStartMonth	12	3	4	12	8
PolicyStartDay	11	25	13	21	5
PolicyStartDayofweek	1	0	6	6	4
PolicyStartDayofyear	345	84	103	355	217
PolicyEndYear	2011	2011	2011	2034	2011
PolicyEndMonth	12	3	4	5	9
PolicyEndDay	10	24	12	20	29
PolicyEndDayofweek	5	3	1	5	3
PolicyEndDayofyear	344	83	102	140	272
FirstTransactionYear	2001	2002	2003	2003	2005
FirstTransactionMonth	12	3	4	12	8
FirstTransactionDay	11	25	13	21	5
FirstTransactionDayofweek	1	0	6	6	4
FirstTransactionDayofyear	345	84	103	355	217
ID_is_missing	False	False	False	False	False
Gender_is_missing	False	False	False	False	False
Car_Category_is_missing	False	False	False	False	True
Subject_Car_Colour_is_missing	False	False	False	True	True
Subject_Car_Make_is_missing	False	False	False	False	False

	8010	10526	10234	12066	8124
LGA_Name_is_missing	True	False	False	True	False
State_is_missing	True	False	False	True	False
ProductName_is_missing	False	False	False	False	False

Now all of our data is numeric and there are no missing values, we should be able to build a machine learning model!

In [377…

```
len(train)
```

Out[377…

```
11511
```

In [378…

```
print('Rows containing 0 =', len(train[train['target']==0]))
print('Rows containing 1 =', len(train[train['target']==1]))
```

```
Rows containing 0 = 10112
Rows containing 1 = 1399
```

In [378…

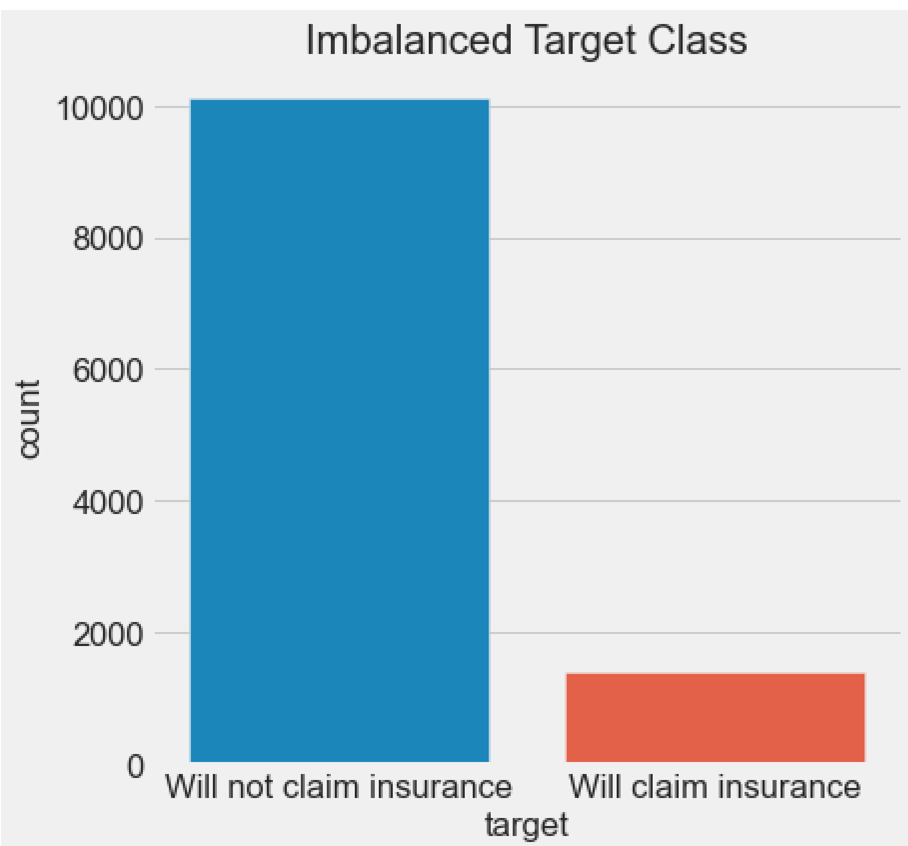
```
train['target'].value_counts()
```

Out[378…

```
0    10112
1     1399
Name: target, dtype: int64
```

In [378…

```
# Visualize the target variable with a bar graph
plt.figure(figsize=(6, 6))
g = sns.countplot('target', data = train)
plt.title("Imbalanced Target Class")
g.set_xticklabels(['Will not claim insurance', 'Will claim insurance'])
plt.show();
```



In [378...
```
# Drop the _is_missing column
train.drop(['ID_is_missing', 'Gender_is_missing', 'Car_Category_is_missing', 'Subject_Car_
        'Subject_Car_Make_is_missing', 'LGA_Name_is_missing', 'State_is_missing', 'Product
train.head()
```

Out[378...

	ID	Gender	Age	No_Pol	Car_Category	Subject_Car_Colour	Subject_Car_Make	LGA_Name	State	Produ
8010	7623	1	37	1	3	5	2	0	0	
10526	10025	2	37	1	3	5	9	74	7	
10234	9745	2	41	2	3	5	9	115	21	
12066	11499	2	48	2	3	0	8	0	0	
8124	7732	1	44	1	0	0	8	22	21	

5 rows × 26 columns

Splitting data into train/validation/test sets

In [378...
```
# Split the training dataset into X and y
X = train.drop('target', axis = 1)
y = train['target']
```

In [378...
```
X.shape, y.shape
```

Out[378...
```
((11511, 25), (11511,))
```

In [378...
```
y.value_counts()
```

Out[378...
```
0    10112
1     1399
Name: target, dtype: int64
```

5. Modelling

What is Data Imbalance?

Data imbalance usually reflects an unequal distribution of classes within a dataset. As with the data set we're working with, The proportion of customers who will claim a car insurance in the first 3 months and customers who will not claim a car insurance in the first 3 months is about 8.13 : 1. If we train our binary classification model without fixing this problem, the model will be completely biased towards the customers who will not claim a car insurance in the first 3 months. Since all of our data is numeric and there are no missing values and we have a highly imbalanced class, we'll attempt to balance the dataset by OverSampling and Undersampling the majority and minority class.

Applying Oversampling technique for the training dataset(Random Oversampling Minority class)

With my training data created, I'll upsample the minority class using the SMOTE algorithm (Synthetic Minority Oversampling Technique). At a high level, SMOTE creates synthetic observations of the minority class by:

- Finding the k-nearest-neighbors for minority class observations (finding similar observations)

- Randomly choosing one of the k-nearest-neighbors and using it to create a similar, but randomly tweaked, new observation.

After upsampling to a class ratio of 1.0, I should have a balanced dataset. There’s no need (and often it’s not smart) to balance the classes, but it magnifies the issue caused by incorrectly timed oversampling.

In [378…

```
# summarize class distribution
print("Before Oversampling: ", Counter(y))

# over = RandomOverSampler(sampling_strategy='minority')
sm = SMOTE(sampling_strategy='minority', random_state=42)

X_sm, y_sm = sm.fit_resample(X, y)

Counter(y_sm)
print(f"After Oversampling: {Counter(y_sm)}")
```

```
Before Oversampling:  Counter({0: 10112, 1: 1399})
After Oversampling: Counter({0: 10112, 1: 10112})
```

In [378…

```
X_sm.shape, y_sm.shape
```

Out[378…

```
((20224, 25), (20224,))
```

In [378…

```
# split the new oversampled data
X_train, X_test, y_train, y_test = train_test_split(X_sm, y_sm, test_size = 0.2, random_st
```

In [379…

```
len(X_train), len(y_test)
```

Out[379…

```
(16179, 4045)
```

We're going to be using 6 models to evaluate the sampled dataset:

- Logistic Regression
- RandomForestClassifier
- KNeighborClassifier
- LGBMClassifier
- CatBoostClassifier
- XGBClassifier

All of the algorithms in the Scikit-Learn library use the same functions, for training a model, model.fit(X_train, y_train) and for scoring a model model.score(X_test, y_test). score() returns the ratio of correct predictions (1.0 = 100% correct).

Metrics:

- Precision is the total number of customers the model correctly identified as customers that will claim insurance out of all the people PREDICTED to claim insurance

- Recall is the total number of customers the model correctly identified as customers that will claim insurance out of all the people who ACTUALLY claimed insurance.

- Accuracy is the total number of correct predictions divided by the total number of predictions.

- It is not possible to achieve both a high precision and a high recall value- we must determine which is more important for us in our model.

- F1 gives us the harmonic mean of precision and recall (Aim for a high F1 value to indicate a good precision and a good recall value).

- ROC (Receiver Operating Characteristic) Curve is a plot betwen the True Positive Rate on the y-axis and the False Positive Rate on the x-axis. A plot with the graph closer to the left and top axes is indicative of a better model.

- AUC (Area Under Curve) values range from 0 to 1 with higher scores indicating a better model. The diagonal line on ROC curves usually represents a random model with an AUC of 0.5. (Would definitely want our model's AUC to be higher than 0.5, since that would signify it is better than random chance.

- PRC (Precision-Recall Curves) plot values of precision scores on the y-axis and recall on the x-axis. A plot with the graph closer to the top and right axes is indicative of a better model. As with ROC curves, we should aim for a high AUC.

5.1 Logistic Regression

In [379…

```
# Logistic Regression
np.random.seed(42)

# Instantiate the model
log = LogisticRegression()

# Fit the model on the train data
log.fit(X_train, y_train)

# Score the model on the test data
log.score(X_test, y_test)
```

Out[379…

```
0.6702101359703337
```

In [379…

```
# Make predictions on the model
log_pred = log.predict(X_test)
log_pred[:10]
```

Out[379…

```
array([1, 1, 0, 0, 1, 0, 0, 0, 0, 1], dtype=int64)
```

In [379…

```
y_test[:10]
```

Out[379…

```
4981     0
5421     0
16026    1
8057     0
119      0
18553    1
9814     0
9787     1
2699     0
19323    1
Name: target, dtype: int64
```

In [379…

```
print(classification_report(y_test, log_pred));
```

```
              precision    recall  f1-score   support

           0       0.67      0.67      0.67      2005
           1       0.67      0.67      0.67      2040

    accuracy                           0.67      4045
   macro avg       0.67      0.67      0.67      4045
weighted avg       0.67      0.67      0.67      4045
```

In [379…

```python
print('Precision Score: ', round(precision_score(y_test, log_pred), 2))
print('Recall Score: ', round(recall_score(y_test, log_pred), 2))
print('F1 Score: ', round(f1_score(y_test, log_pred), 2))
print('Accuracy Score: ', round(accuracy_score(y_test, log_pred), 2))
print('ROC AUC: ', round(roc_auc_score(y_test, log_pred), 2))
```

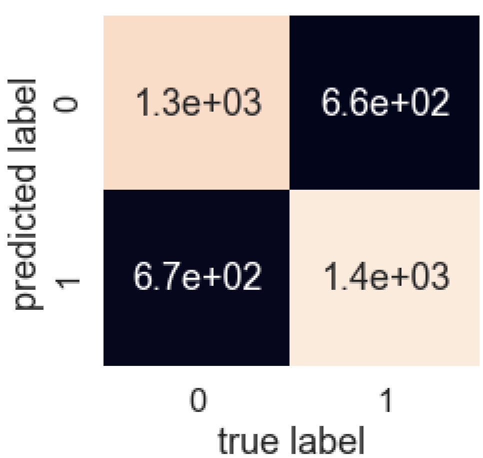
```
Precision Score:  0.67
Recall Score:  0.67
F1 Score:  0.67
Accuracy Score:  0.67
ROC AUC:  0.67
```

5.1.1 Confusion Matrix of LogisticRegression Model

In [379…

```python
sns.set(font_scale=1.5)

def plot_conf_mat(y_test, log_pred):
    """
    Plots a confusion matrix using Seaborn's heatmap().
    """
    fig, ax = plt.subplots(figsize=(3, 3))
    ax = sns.heatmap(confusion_matrix(y_test, log_pred),
                     annot=True,
                     cbar=False)
    plt.xlabel("true label")
    plt.ylabel("predicted label")

plot_conf_mat(y_test, log_pred)
print(confusion_matrix(y_test, log_pred))
```

```
[[1343  662]
 [ 672 1368]]
```

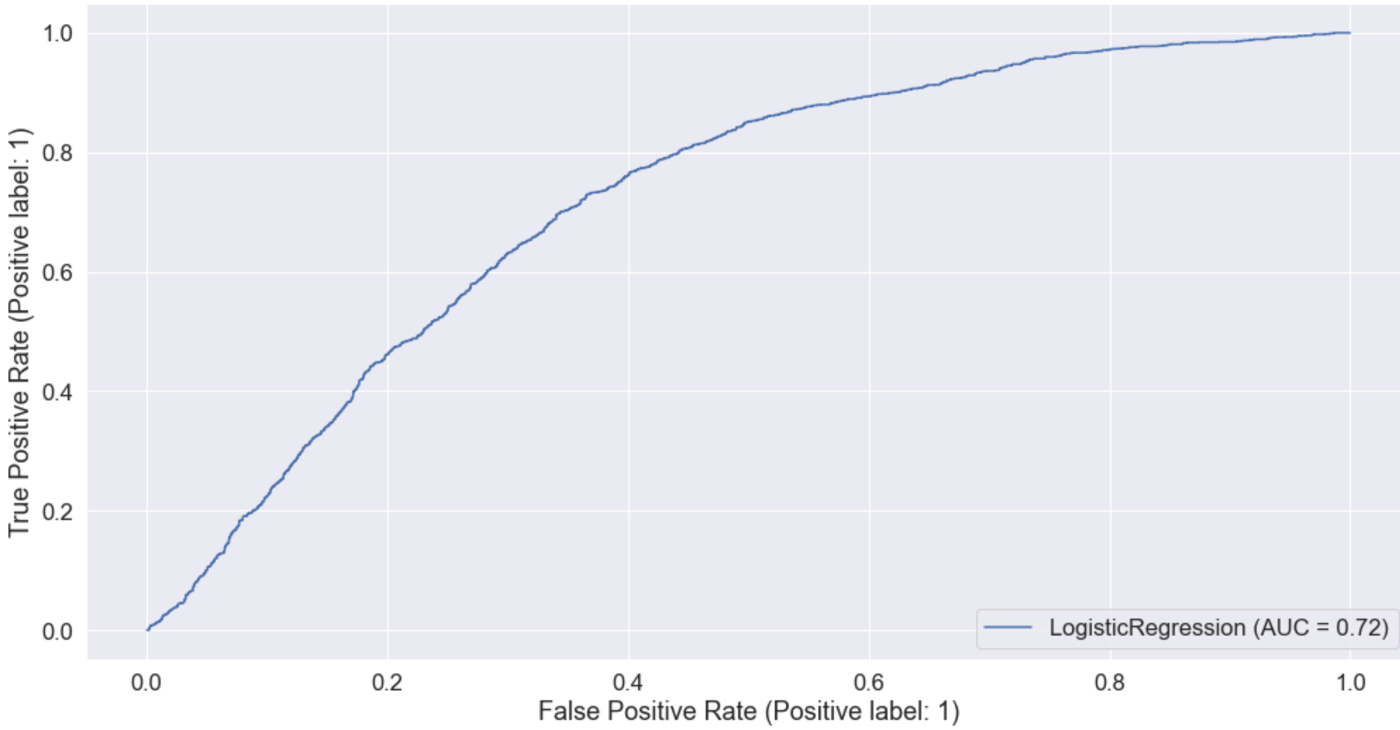


You can see the model gets confused (predicts the wrong label). In essence, there are 662 occasaions where the model predicted 0 when it should've been 1 (false negative) and 672 occasions where the model predicted 1 instead of 0 (false positive).

5.1.2 ROC Curve and AUC Scores for the Logistic Regression model

In [379...

```
# Plot ROC curve and calculate AUC metric
plot_roc_curve(log, X_test, y_test);
```



5.2 Random Forest

In [379...

```
# Random Forest
np.random.seed(42)

# Instantiate the model
rf = RandomForestClassifier()

# # Fit the model on the training data
rf.fit(X_train, y_train)

# Score the model on the test data
rf.score(X_test, y_test)
```

Out[379...

```
0.892459826946848
```

In [379...

```
# Make predictions on the model
rf_pred = rf.predict(X_test)
rf_pred[:10]
```

Out[379...

```
array([0, 1, 1, 0, 1, 1, 0, 0, 0, 1], dtype=int64)
```

In [380...

```
y_test[:10]
```

Out[380...

```
4981     0
5421     0
16026    1
8057     0
119      0
18553    1
```

```
9814     0
9787     1
2699     0
19323    1
Name: target, dtype: int64
```

In [380…

```python
print(classification_report(y_test, rf_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      2005
           1       0.89      0.90      0.89      2040

    accuracy                           0.89      4045
   macro avg       0.89      0.89      0.89      4045
weighted avg       0.89      0.89      0.89      4045
```

In [380…

```python
print('Precision Score:', round(precision_score(y_test, rf_pred), 2))
print('Recall Score:', round(recall_score(y_test, rf_pred), 2))
print('F1 Score:', round(f1_score(y_test, rf_pred), 2))
print('Accuracy Score:', round(accuracy_score(y_test, rf_pred), 2))
print('ROC AUC: ', round(roc_auc_score(y_test, rf_pred), 2))
```

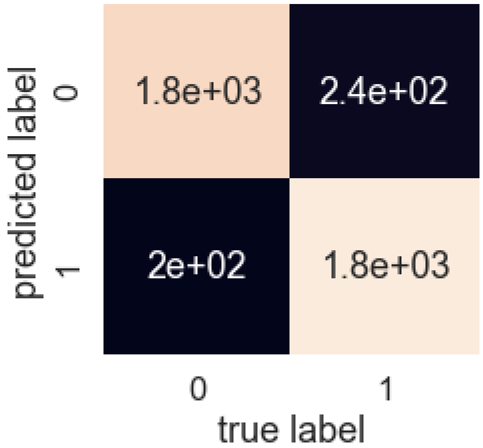
```
Precision Score: 0.89
Recall Score: 0.9
F1 Score: 0.89
Accuracy Score: 0.89
ROC AUC:  0.89
```

5.2.1 Confusion Matrix of RandomForest Model

In [380…

```python
sns.set(font_scale=1.5)

def plot_conf_mat(y_test, rf_pred):
    """
    Plots a confusion matrix using Seaborn's heatmap().
    """
    fig, ax = plt.subplots(figsize=(3, 3))
    ax = sns.heatmap(confusion_matrix(y_test, rf_pred),
                     annot=True,
                     cbar=False)
    plt.xlabel("true label")
    plt.ylabel("predicted label")

plot_conf_mat(y_test, rf_pred)
print(confusion_matrix(y_test, rf_pred))
```

```
[[1766  239]
 [ 196 1844]]
```

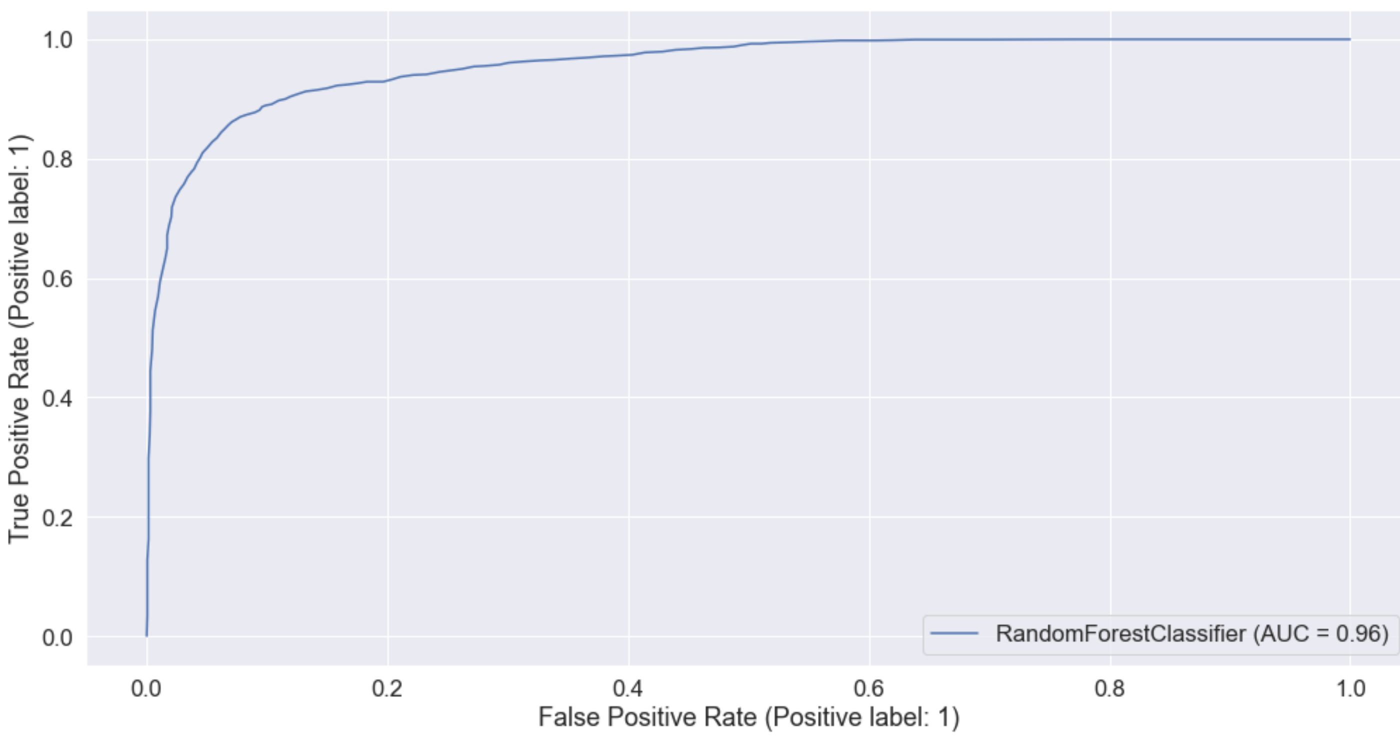


You can see the model gets confused (predicts the wrong label). In essence, there are 239 occasaions where the model predicted 0 when it should've been 1 (false negative) and 196 occasions where the model predicted 1 instead of 0 (false positive).

5.2.2 ROC Curve and AUC Scores for the RandomForestClassifier Model

In [380…

```
# Plot ROC curve and calculate AUC metric
plot_roc_curve(rf, X_test, y_test);
```



This is great, the model does far better than guessing which would be a line going from the bottom left corner to the top right corner, AUC = 0.96. But a perfect model would achieve an AUC score of 1.0.

5.3 KNeighborsClassifier

In [380…

```
np.random.seed(42)

# Instantiate the model
knn = KNeighborsClassifier()

# Fit the model to the training data
knn.fit(X_train, y_train)
```

```
# Score the model on the test data
knn.score(X_test, y_test)
```

Out[380…

```
0.8012360939431397
```

In [380…

```
# Make predictions on the model
knn_pred = knn.predict(X_test)
knn_pred[:10]
```

Out[380…

```
array([0, 1, 1, 0, 1, 1, 1, 1, 0, 1], dtype=int64)
```

In [380…

```
y_test[:10]
```

Out[380…

```
4981     0
5421     0
16026    1
8057     0
119      0
18553    1
9814     0
9787     1
2699     0
19323    1
Name: target, dtype: int64
```

In [380…

```
print(classification_report(y_test, knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.66      0.77      2005
           1       0.74      0.94      0.83      2040

    accuracy                           0.80      4045
   macro avg       0.83      0.80      0.80      4045
weighted avg       0.83      0.80      0.80      4045
```

In [380…

```
print('Precision Score:', round(precision_score(y_test, knn_pred), 2))
print('Recall Score:', round(recall_score(y_test, knn_pred), 2))
print('F1 Score:', round(f1_score(y_test, knn_pred), 2))
print('Accuracy Score:', round(accuracy_score(y_test, knn_pred), 2))
print('ROC AUC: ', round(roc_auc_score(y_test, knn_pred), 2))
```

```
Precision Score: 0.74
Recall Score: 0.94
F1 Score: 0.83
Accuracy Score: 0.8
ROC AUC:  0.8
```

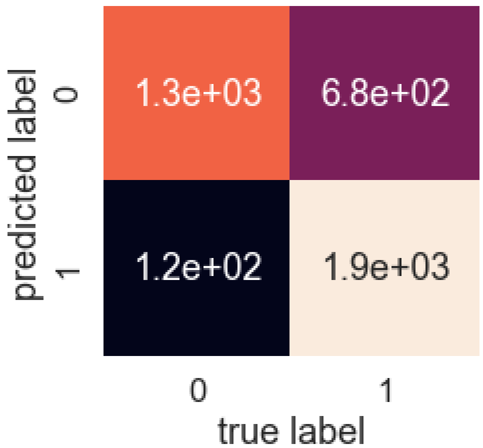
5.3.1 Confusion Matrix of KNeighborsClassifier Model

In [381…

```
sns.set(font_scale=1.5)

def plot_conf_mat(y_test, knn_pred):
    """
    Plots a confusion matrix using Seaborn's heatmap().
    """
    fig, ax = plt.subplots(figsize=(3, 3))
```

```
    ax = sns.heatmap(confusion_matrix(y_test, knn_pred),
                     annot=True,
                     cbar=False)
    plt.xlabel("true label")
    plt.ylabel("predicted label")

plot_conf_mat(y_test, knn_pred)
print(confusion_matrix(y_test, knn_pred))
```

```
[[1323  682]
 [ 122 1918]]
```

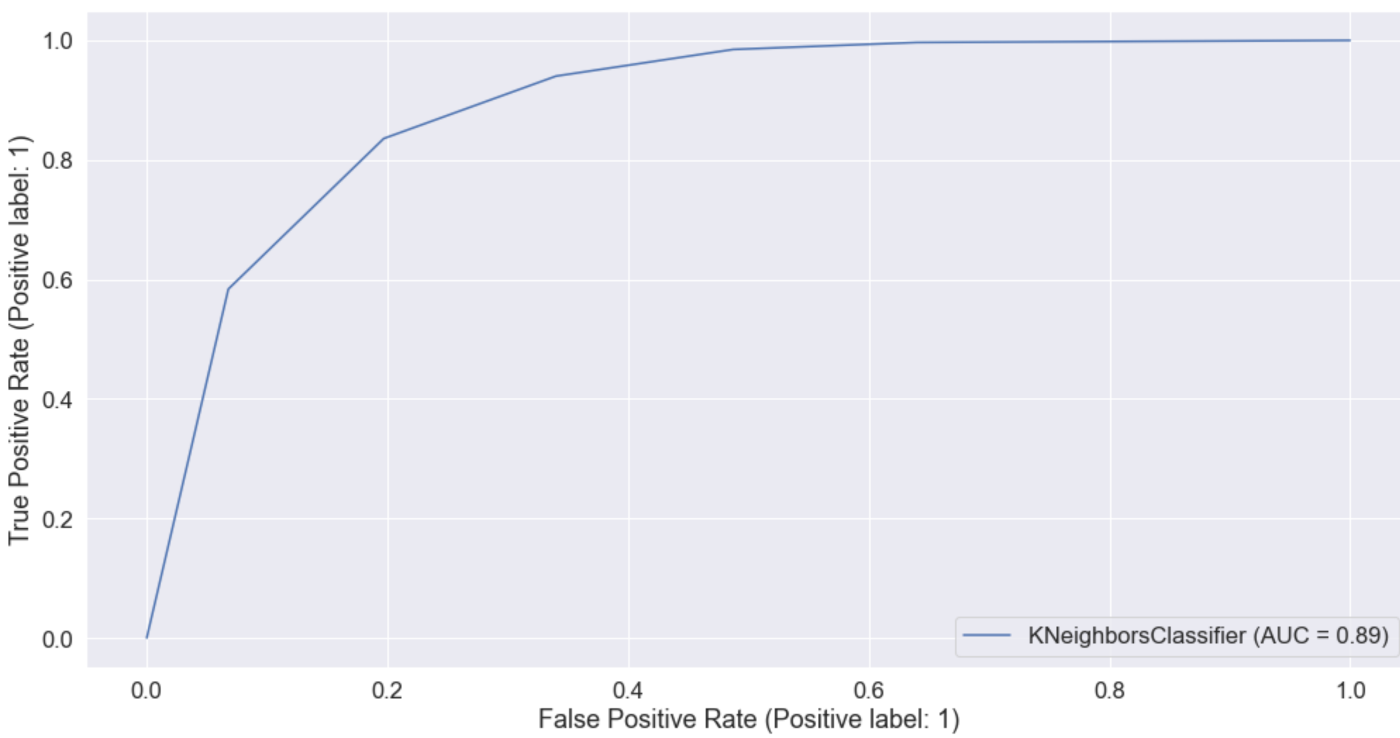


You can see the model gets confused (predicts the wrong label). In essence, there are 682 occasaions where the model predicted 0 when it should've been 1 (false negative) and 122 occasions where the model predicted 1 instead of 0 (false positive).

5.3.2 ROC Curve and AUC Scores for the KNeighborsClassifier Model

In [381…

```
# Plot ROC curve and calculate AUC metric
plot_roc_curve(knn, X_test, y_test);
```



This is great, the model does far better than guessing which would be a line going from the bottom left corner to the top right corner, AUC = 0.89. But a perfect model would achieve an AUC score of 1.0.

5.4 LightGBM Model

In [381…

```
np.random.seed(42)

# Instantiate the model
lgbm = LGBMClassifier()

# Fit the model
lgbm.fit(X_train, y_train)

# Score the model on the test data
lgbm.score(X_test, y_test)
```

Out[381…

```
0.8956736711990111
```

In [381…

```
# Make predictions on the model
lgbm_pred = lgbm.predict(X_test)
lgbm_pred[:10]
```

Out[381…

```
array([0, 0, 1, 0, 1, 1, 0, 0, 0, 1], dtype=int64)
```

In [381…

```
y_test[:10]
```

Out[381…

```
4981     0
5421     0
16026    1
8057     0
119      0
18553    1
9814     0
9787     1
2699     0
19323    1
Name: target, dtype: int64
```

In [381…

```
print(classification_report(y_test, lgbm_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.91      0.90      2005
           1       0.91      0.88      0.89      2040

    accuracy                           0.90      4045
   macro avg       0.90      0.90      0.90      4045
weighted avg       0.90      0.90      0.90      4045
```

In [381…

```
print('Precision Score:', round(precision_score(y_test, lgbm_pred), 2))
print('Recall Score:', round(recall_score(y_test, lgbm_pred), 2))
print('F1 Score:', round(f1_score(y_test, lgbm_pred), 2))
print('Accuracy Score:', round(accuracy_score(y_test, lgbm_pred), 2))
print('ROC AUC: ', round(roc_auc_score(y_test, lgbm_pred), 2))
```

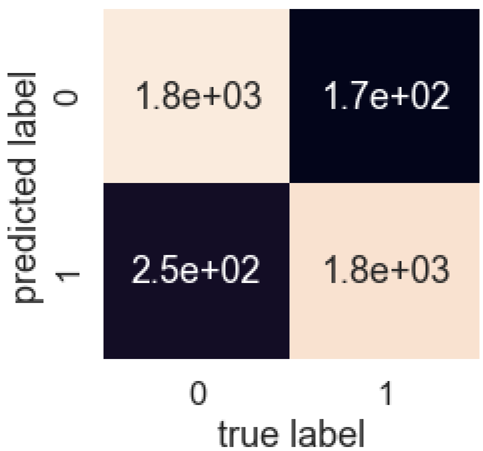
```
Precision Score: 0.91
Recall Score: 0.88
F1 Score: 0.89
Accuracy Score: 0.9
ROC AUC:  0.9
```

5.4.1 Confusion Matrix of LightGBMClassifier Model

In [381…

```
sns.set(font_scale=1.5)

def plot_conf_mat(y_test, lgbm_pred):
    """
    Plots a confusion matrix using Seaborn's heatmap().
    """
    fig, ax = plt.subplots(figsize=(3, 3))
    ax = sns.heatmap(confusion_matrix(y_test, lgbm_pred),
                     annot=True,
                     cbar=False)
    plt.xlabel("true label")
    plt.ylabel("predicted label")

plot_conf_mat(y_test, lgbm_pred)
print(confusion_matrix(y_test, lgbm_pred))
```

```
[[1833  172]
 [ 250 1790]]
```

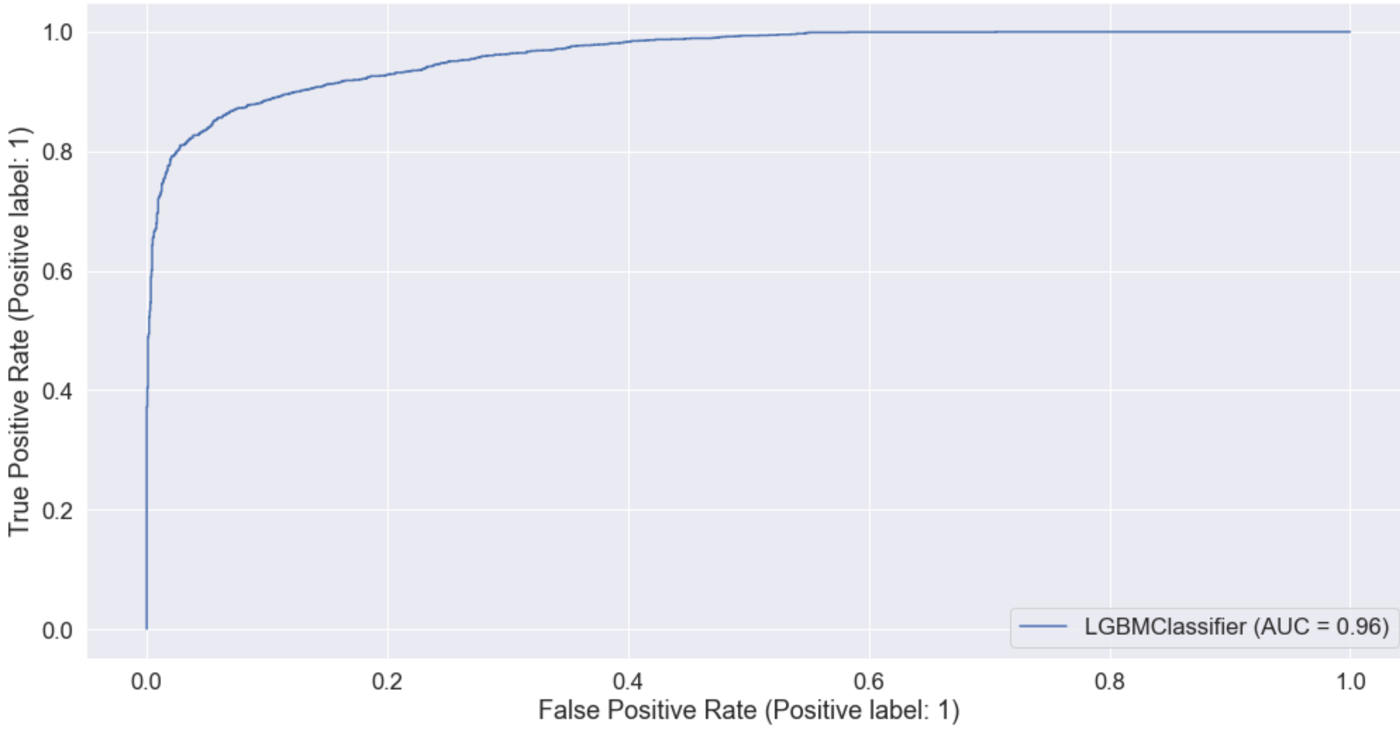


You can see the model gets confused (predicts the wrong label). In essence, there are 172 occasaions where the model predicted 0 when it should've been 1 (false negative) and 250 occasions where the model predicted 1 instead of 0 (false positive).

5.4.2 ROC Curve and AUC Scores for the LightGBMClassifier Model

In [381…

```
# Plot ROC curve and calculate AUC metric
plot_roc_curve(lgbm, X_test, y_test);
```



This is great, the model does far better than guessing which would be a line going from the bottom left corner to the top right corner, AUC = 0.96. But a perfect model would achieve an AUC score of 1.0.

5.5 CatBoost Model

In [381…

```
np. random.seed(42)

# Instantiate the model
cat = CatBoostClassifier()

# Fit the model to the training data
cat.fit(X_train, y_train)

# Score the model on the test data
cat.score(X_test, y_test)
```

```
Learning rate set to 0.033819
0:	learn: 0.6712682	total: 53.6ms	remaining: 53.6s
1:	learn: 0.6572553	total: 74.7ms	remaining: 37.3s
2:	learn: 0.6424698	total: 95.7ms	remaining: 31.8s
3:	learn: 0.6251396	total: 116ms	remaining: 29s
4:	learn: 0.6116585	total: 135ms	remaining: 26.9s
5:	learn: 0.5981645	total: 155ms	remaining: 25.7s
6:	learn: 0.5883858	total: 174ms	remaining: 24.7s
7:	learn: 0.5801929	total: 200ms	remaining: 24.8s
8:	learn: 0.5699848	total: 224ms	remaining: 24.7s
9:	learn: 0.5610843	total: 254ms	remaining: 25.1s
10:	learn: 0.5542218	total: 286ms	remaining: 25.7s
11:	learn: 0.5493848	total: 310ms	remaining: 25.5s
12:	learn: 0.5421030	total: 334ms	remaining: 25.4s
13:	learn: 0.5352862	total: 357ms	remaining: 25.2s
14:	learn: 0.5306646	total: 381ms	remaining: 25s
15:	learn: 0.5264965	total: 406ms	remaining: 25s
16:	learn: 0.5198893	total: 429ms	remaining: 24.8s
17:	learn: 0.5144392	total: 452ms	remaining: 24.6s
18:	learn: 0.5085763	total: 474ms	remaining: 24.5s
19:	learn: 0.5030830	total: 497ms	remaining: 24.3s
20:	learn: 0.4976880	total: 524ms	remaining: 24.4s
21:	learn: 0.4938460	total: 548ms	remaining: 24.4s
```

```
22:	learn: 0.4892602	total: 571ms	remaining: 24.3s
23:	learn: 0.4843191	total: 601ms	remaining: 24.4s
24:	learn: 0.4806110	total: 627ms	remaining: 24.4s
25:	learn: 0.4749912	total: 650ms	remaining: 24.4s
26:	learn: 0.4719118	total: 673ms	remaining: 24.3s
27:	learn: 0.4680723	total: 695ms	remaining: 24.1s
28:	learn: 0.4650074	total: 717ms	remaining: 24s
29:	learn: 0.4626284	total: 737ms	remaining: 23.8s
30:	learn: 0.4600966	total: 758ms	remaining: 23.7s
31:	learn: 0.4579340	total: 777ms	remaining: 23.5s
32:	learn: 0.4558001	total: 797ms	remaining: 23.4s
33:	learn: 0.4524839	total: 819ms	remaining: 23.3s
34:	learn: 0.4494158	total: 843ms	remaining: 23.2s
35:	learn: 0.4460671	total: 866ms	remaining: 23.2s
36:	learn: 0.4430541	total: 891ms	remaining: 23.2s
37:	learn: 0.4401941	total: 916ms	remaining: 23.2s
38:	learn: 0.4382272	total: 939ms	remaining: 23.1s
39:	learn: 0.4366878	total: 975ms	remaining: 23.4s
40:	learn: 0.4345521	total: 1s	remaining: 23.4s
41:	learn: 0.4323829	total: 1.03s	remaining: 23.5s
42:	learn: 0.4304313	total: 1.06s	remaining: 23.5s
43:	learn: 0.4283602	total: 1.08s	remaining: 23.5s
44:	learn: 0.4262053	total: 1.11s	remaining: 23.5s
45:	learn: 0.4248846	total: 1.13s	remaining: 23.4s
46:	learn: 0.4224761	total: 1.15s	remaining: 23.3s
47:	learn: 0.4206120	total: 1.17s	remaining: 23.2s
48:	learn: 0.4195968	total: 1.19s	remaining: 23.1s
49:	learn: 0.4173155	total: 1.21s	remaining: 23.1s
50:	learn: 0.4143647	total: 1.24s	remaining: 23s
51:	learn: 0.4129885	total: 1.26s	remaining: 23s
52:	learn: 0.4117976	total: 1.28s	remaining: 22.9s
53:	learn: 0.4102620	total: 1.3s	remaining: 22.8s
54:	learn: 0.4084982	total: 1.32s	remaining: 22.7s
55:	learn: 0.4075365	total: 1.34s	remaining: 22.6s
56:	learn: 0.4061186	total: 1.36s	remaining: 22.5s
57:	learn: 0.4042866	total: 1.38s	remaining: 22.4s
58:	learn: 0.4029119	total: 1.39s	remaining: 22.2s
59:	learn: 0.4017834	total: 1.41s	remaining: 22.1s
60:	learn: 0.4009041	total: 1.42s	remaining: 21.9s
61:	learn: 0.3998416	total: 1.44s	remaining: 21.8s
62:	learn: 0.3989350	total: 1.46s	remaining: 21.7s
63:	learn: 0.3979156	total: 1.48s	remaining: 21.6s
64:	learn: 0.3961941	total: 1.49s	remaining: 21.5s
65:	learn: 0.3943945	total: 1.52s	remaining: 21.5s
66:	learn: 0.3931992	total: 1.53s	remaining: 21.3s
67:	learn: 0.3924789	total: 1.55s	remaining: 21.2s
68:	learn: 0.3915640	total: 1.57s	remaining: 21.1s
69:	learn: 0.3901291	total: 1.58s	remaining: 21s
70:	learn: 0.3894822	total: 1.6s	remaining: 20.9s
71:	learn: 0.3887619	total: 1.61s	remaining: 20.8s
72:	learn: 0.3878339	total: 1.63s	remaining: 20.7s
73:	learn: 0.3866301	total: 1.65s	remaining: 20.6s
74:	learn: 0.3854762	total: 1.67s	remaining: 20.6s
75:	learn: 0.3839835	total: 1.69s	remaining: 20.5s
76:	learn: 0.3831677	total: 1.71s	remaining: 20.4s
77:	learn: 0.3823591	total: 1.72s	remaining: 20.3s
78:	learn: 0.3812355	total: 1.74s	remaining: 20.2s
79:	learn: 0.3804802	total: 1.75s	remaining: 20.1s
80:	learn: 0.3794473	total: 1.77s	remaining: 20.1s
81:	learn: 0.3781248	total: 1.78s	remaining: 20s
82:	learn: 0.3767945	total: 1.8s	remaining: 19.9s
83:	learn: 0.3761664	total: 1.81s	remaining: 19.8s
84:	learn: 0.3750156	total: 1.83s	remaining: 19.7s
85:	learn: 0.3745030	total: 1.85s	remaining: 19.7s
86:	learn: 0.3736831	total: 1.87s	remaining: 19.6s
87:	learn: 0.3726815	total: 1.89s	remaining: 19.6s
```

```
88:	learn: 0.3715521	total: 1.91s	remaining: 19.5s
89:	learn: 0.3704860	total: 1.93s	remaining: 19.5s
90:	learn: 0.3694714	total: 1.95s	remaining: 19.4s
91:	learn: 0.3690793	total: 1.97s	remaining: 19.4s
92:	learn: 0.3682972	total: 1.99s	remaining: 19.4s
93:	learn: 0.3676623	total: 2.02s	remaining: 19.5s
94:	learn: 0.3667227	total: 2.04s	remaining: 19.4s
95:	learn: 0.3657908	total: 2.06s	remaining: 19.4s
96:	learn: 0.3653725	total: 2.08s	remaining: 19.4s
97:	learn: 0.3649755	total: 2.1s	remaining: 19.4s
98:	learn: 0.3644728	total: 2.13s	remaining: 19.4s
99:	learn: 0.3638868	total: 2.15s	remaining: 19.4s
100:	learn: 0.3626555	total: 2.17s	remaining: 19.4s
101:	learn: 0.3619257	total: 2.19s	remaining: 19.3s
102:	learn: 0.3606202	total: 2.22s	remaining: 19.3s
103:	learn: 0.3596390	total: 2.23s	remaining: 19.3s
104:	learn: 0.3589833	total: 2.25s	remaining: 19.2s
105:	learn: 0.3584979	total: 2.27s	remaining: 19.2s
106:	learn: 0.3579996	total: 2.29s	remaining: 19.1s
107:	learn: 0.3565079	total: 2.31s	remaining: 19.1s
108:	learn: 0.3556456	total: 2.34s	remaining: 19.1s
109:	learn: 0.3548126	total: 2.36s	remaining: 19.1s
110:	learn: 0.3542609	total: 2.37s	remaining: 19s
111:	learn: 0.3537148	total: 2.39s	remaining: 19s
112:	learn: 0.3532923	total: 2.41s	remaining: 18.9s
113:	learn: 0.3526028	total: 2.42s	remaining: 18.9s
114:	learn: 0.3519070	total: 2.44s	remaining: 18.8s
115:	learn: 0.3510361	total: 2.46s	remaining: 18.7s
116:	learn: 0.3505613	total: 2.47s	remaining: 18.7s
117:	learn: 0.3499602	total: 2.49s	remaining: 18.6s
118:	learn: 0.3493980	total: 2.51s	remaining: 18.6s
119:	learn: 0.3488312	total: 2.53s	remaining: 18.6s
120:	learn: 0.3482120	total: 2.55s	remaining: 18.5s
121:	learn: 0.3475245	total: 2.56s	remaining: 18.4s
122:	learn: 0.3469174	total: 2.58s	remaining: 18.4s
123:	learn: 0.3459984	total: 2.59s	remaining: 18.3s
124:	learn: 0.3453751	total: 2.61s	remaining: 18.3s
125:	learn: 0.3448275	total: 2.62s	remaining: 18.2s
126:	learn: 0.3445346	total: 2.64s	remaining: 18.1s
127:	learn: 0.3440405	total: 2.65s	remaining: 18.1s
128:	learn: 0.3435968	total: 2.67s	remaining: 18s
129:	learn: 0.3428384	total: 2.69s	remaining: 18s
130:	learn: 0.3423867	total: 2.71s	remaining: 18s
131:	learn: 0.3417751	total: 2.72s	remaining: 17.9s
132:	learn: 0.3413310	total: 2.74s	remaining: 17.9s
133:	learn: 0.3407564	total: 2.76s	remaining: 17.8s
134:	learn: 0.3404133	total: 2.78s	remaining: 17.8s
135:	learn: 0.3397137	total: 2.79s	remaining: 17.8s
136:	learn: 0.3391678	total: 2.81s	remaining: 17.7s
137:	learn: 0.3387259	total: 2.83s	remaining: 17.7s
138:	learn: 0.3381940	total: 2.85s	remaining: 17.7s
139:	learn: 0.3373923	total: 2.87s	remaining: 17.6s
140:	learn: 0.3367444	total: 2.89s	remaining: 17.6s
141:	learn: 0.3364043	total: 2.91s	remaining: 17.6s
142:	learn: 0.3356413	total: 2.93s	remaining: 17.6s
143:	learn: 0.3350042	total: 2.95s	remaining: 17.5s
144:	learn: 0.3343983	total: 2.97s	remaining: 17.5s
145:	learn: 0.3340653	total: 2.99s	remaining: 17.5s
146:	learn: 0.3334036	total: 3.01s	remaining: 17.5s
147:	learn: 0.3329964	total: 3.03s	remaining: 17.5s
148:	learn: 0.3323536	total: 3.05s	remaining: 17.4s
149:	learn: 0.3317905	total: 3.07s	remaining: 17.4s
150:	learn: 0.3307952	total: 3.1s	remaining: 17.4s
151:	learn: 0.3301358	total: 3.12s	remaining: 17.4s
152:	learn: 0.3296541	total: 3.13s	remaining: 17.4s
153:	learn: 0.3290319	total: 3.15s	remaining: 17.3s
```

```
154:	learn: 0.3285908	total: 3.17s	remaining: 17.3s
155:	learn: 0.3279407	total: 3.18s	remaining: 17.2s
156:	learn: 0.3274039	total: 3.2s	remaining: 17.2s
157:	learn: 0.3269668	total: 3.22s	remaining: 17.1s
158:	learn: 0.3264922	total: 3.23s	remaining: 17.1s
159:	learn: 0.3254779	total: 3.25s	remaining: 17.1s
160:	learn: 0.3251240	total: 3.26s	remaining: 17s
161:	learn: 0.3247704	total: 3.28s	remaining: 17s
162:	learn: 0.3240534	total: 3.3s	remaining: 16.9s
163:	learn: 0.3234484	total: 3.32s	remaining: 16.9s
164:	learn: 0.3224481	total: 3.33s	remaining: 16.9s
165:	learn: 0.3220021	total: 3.35s	remaining: 16.9s
166:	learn: 0.3215427	total: 3.37s	remaining: 16.8s
167:	learn: 0.3210668	total: 3.39s	remaining: 16.8s
168:	learn: 0.3207147	total: 3.4s	remaining: 16.7s
169:	learn: 0.3201675	total: 3.42s	remaining: 16.7s
170:	learn: 0.3196579	total: 3.44s	remaining: 16.7s
171:	learn: 0.3190103	total: 3.45s	remaining: 16.6s
172:	learn: 0.3186680	total: 3.47s	remaining: 16.6s
173:	learn: 0.3182310	total: 3.48s	remaining: 16.5s
174:	learn: 0.3179345	total: 3.5s	remaining: 16.5s
175:	learn: 0.3175129	total: 3.52s	remaining: 16.5s
176:	learn: 0.3170810	total: 3.54s	remaining: 16.5s
177:	learn: 0.3167255	total: 3.56s	remaining: 16.4s
178:	learn: 0.3164670	total: 3.57s	remaining: 16.4s
179:	learn: 0.3159627	total: 3.59s	remaining: 16.3s
180:	learn: 0.3154177	total: 3.6s	remaining: 16.3s
181:	learn: 0.3149466	total: 3.62s	remaining: 16.3s
182:	learn: 0.3141825	total: 3.63s	remaining: 16.2s
183:	learn: 0.3135832	total: 3.65s	remaining: 16.2s
184:	learn: 0.3129995	total: 3.66s	remaining: 16.1s
185:	learn: 0.3126704	total: 3.68s	remaining: 16.1s
186:	learn: 0.3121896	total: 3.69s	remaining: 16.1s
187:	learn: 0.3117773	total: 3.71s	remaining: 16s
188:	learn: 0.3113839	total: 3.73s	remaining: 16s
189:	learn: 0.3108032	total: 3.75s	remaining: 16s
190:	learn: 0.3105311	total: 3.76s	remaining: 15.9s
191:	learn: 0.3103329	total: 3.78s	remaining: 15.9s
192:	learn: 0.3098035	total: 3.79s	remaining: 15.9s
193:	learn: 0.3094054	total: 3.81s	remaining: 15.8s
194:	learn: 0.3093055	total: 3.82s	remaining: 15.8s
195:	learn: 0.3091356	total: 3.84s	remaining: 15.7s
196:	learn: 0.3087594	total: 3.86s	remaining: 15.7s
197:	learn: 0.3078204	total: 3.87s	remaining: 15.7s
198:	learn: 0.3076020	total: 3.89s	remaining: 15.6s
199:	learn: 0.3072299	total: 3.9s	remaining: 15.6s
200:	learn: 0.3066929	total: 3.92s	remaining: 15.6s
201:	learn: 0.3061531	total: 3.94s	remaining: 15.6s
202:	learn: 0.3058140	total: 3.96s	remaining: 15.5s
203:	learn: 0.3056290	total: 3.98s	remaining: 15.5s
204:	learn: 0.3053307	total: 4s	remaining: 15.5s
205:	learn: 0.3046262	total: 4.02s	remaining: 15.5s
206:	learn: 0.3043929	total: 4.04s	remaining: 15.5s
207:	learn: 0.3041252	total: 4.05s	remaining: 15.4s
208:	learn: 0.3038569	total: 4.07s	remaining: 15.4s
209:	learn: 0.3036478	total: 4.08s	remaining: 15.4s
210:	learn: 0.3032488	total: 4.1s	remaining: 15.3s
211:	learn: 0.3027748	total: 4.12s	remaining: 15.3s
212:	learn: 0.3023895	total: 4.14s	remaining: 15.3s
213:	learn: 0.3020091	total: 4.15s	remaining: 15.3s
214:	learn: 0.3012542	total: 4.17s	remaining: 15.2s
215:	learn: 0.3010046	total: 4.19s	remaining: 15.2s
216:	learn: 0.3004466	total: 4.2s	remaining: 15.2s
217:	learn: 0.3000877	total: 4.22s	remaining: 15.1s
218:	learn: 0.2994684	total: 4.24s	remaining: 15.1s
219:	learn: 0.2992437	total: 4.25s	remaining: 15.1s
```

```
220:	learn: 0.2990175	total: 4.27s	remaining: 15s
221:	learn: 0.2987653	total: 4.28s	remaining: 15s
222:	learn: 0.2982857	total: 4.3s	remaining: 15s
223:	learn: 0.2977328	total: 4.32s	remaining: 15s
224:	learn: 0.2974501	total: 4.34s	remaining: 14.9s
225:	learn: 0.2970175	total: 4.35s	remaining: 14.9s
226:	learn: 0.2967879	total: 4.37s	remaining: 14.9s
227:	learn: 0.2965931	total: 4.39s	remaining: 14.9s
228:	learn: 0.2961444	total: 4.41s	remaining: 14.8s
229:	learn: 0.2958300	total: 4.42s	remaining: 14.8s
230:	learn: 0.2954847	total: 4.44s	remaining: 14.8s
231:	learn: 0.2951541	total: 4.45s	remaining: 14.7s
232:	learn: 0.2946887	total: 4.47s	remaining: 14.7s
233:	learn: 0.2944265	total: 4.49s	remaining: 14.7s
234:	learn: 0.2940002	total: 4.5s	remaining: 14.7s
235:	learn: 0.2935705	total: 4.53s	remaining: 14.6s
236:	learn: 0.2932859	total: 4.54s	remaining: 14.6s
237:	learn: 0.2927227	total: 4.56s	remaining: 14.6s
238:	learn: 0.2925477	total: 4.58s	remaining: 14.6s
239:	learn: 0.2923151	total: 4.59s	remaining: 14.5s
240:	learn: 0.2918112	total: 4.61s	remaining: 14.5s
241:	learn: 0.2914736	total: 4.62s	remaining: 14.5s
242:	learn: 0.2912538	total: 4.64s	remaining: 14.4s
243:	learn: 0.2907315	total: 4.65s	remaining: 14.4s
244:	learn: 0.2903213	total: 4.67s	remaining: 14.4s
245:	learn: 0.2900680	total: 4.68s	remaining: 14.4s
246:	learn: 0.2897323	total: 4.7s	remaining: 14.3s
247:	learn: 0.2894900	total: 4.72s	remaining: 14.3s
248:	learn: 0.2889866	total: 4.74s	remaining: 14.3s
249:	learn: 0.2886983	total: 4.75s	remaining: 14.3s
250:	learn: 0.2883234	total: 4.77s	remaining: 14.2s
251:	learn: 0.2879427	total: 4.78s	remaining: 14.2s
252:	learn: 0.2878361	total: 4.8s	remaining: 14.2s
253:	learn: 0.2873397	total: 4.82s	remaining: 14.1s
254:	learn: 0.2870274	total: 4.83s	remaining: 14.1s
255:	learn: 0.2867832	total: 4.85s	remaining: 14.1s
256:	learn: 0.2861946	total: 4.87s	remaining: 14.1s
257:	learn: 0.2858757	total: 4.89s	remaining: 14.1s
258:	learn: 0.2856474	total: 4.91s	remaining: 14s
259:	learn: 0.2854630	total: 4.93s	remaining: 14s
260:	learn: 0.2852898	total: 4.95s	remaining: 14s
261:	learn: 0.2851623	total: 4.97s	remaining: 14s
262:	learn: 0.2848833	total: 4.99s	remaining: 14s
263:	learn: 0.2846202	total: 5.01s	remaining: 14s
264:	learn: 0.2844816	total: 5.03s	remaining: 14s
265:	learn: 0.2842308	total: 5.05s	remaining: 13.9s
266:	learn: 0.2836850	total: 5.08s	remaining: 13.9s
267:	learn: 0.2832295	total: 5.1s	remaining: 13.9s
268:	learn: 0.2830334	total: 5.13s	remaining: 13.9s
269:	learn: 0.2827210	total: 5.16s	remaining: 13.9s
270:	learn: 0.2825026	total: 5.18s	remaining: 13.9s
271:	learn: 0.2821759	total: 5.2s	remaining: 13.9s
272:	learn: 0.2819006	total: 5.23s	remaining: 13.9s
273:	learn: 0.2816415	total: 5.25s	remaining: 13.9s
274:	learn: 0.2813704	total: 5.26s	remaining: 13.9s
275:	learn: 0.2809740	total: 5.28s	remaining: 13.9s
276:	learn: 0.2807785	total: 5.3s	remaining: 13.8s
277:	learn: 0.2805628	total: 5.32s	remaining: 13.8s
278:	learn: 0.2803178	total: 5.34s	remaining: 13.8s
279:	learn: 0.2800480	total: 5.36s	remaining: 13.8s
280:	learn: 0.2798526	total: 5.39s	remaining: 13.8s
281:	learn: 0.2794822	total: 5.4s	remaining: 13.8s
282:	learn: 0.2792129	total: 5.42s	remaining: 13.7s
283:	learn: 0.2789232	total: 5.45s	remaining: 13.7s
284:	learn: 0.2786405	total: 5.47s	remaining: 13.7s
285:	learn: 0.2784351	total: 5.49s	remaining: 13.7s
```

```
286:	learn: 0.2781036	total: 5.5s	remaining: 13.7s
287:	learn: 0.2779303	total: 5.52s	remaining: 13.6s
288:	learn: 0.2776343	total: 5.54s	remaining: 13.6s
289:	learn: 0.2773688	total: 5.55s	remaining: 13.6s
290:	learn: 0.2770885	total: 5.57s	remaining: 13.6s
291:	learn: 0.2768113	total: 5.59s	remaining: 13.6s
292:	learn: 0.2764446	total: 5.61s	remaining: 13.5s
293:	learn: 0.2761856	total: 5.62s	remaining: 13.5s
294:	learn: 0.2755186	total: 5.64s	remaining: 13.5s
295:	learn: 0.2752966	total: 5.65s	remaining: 13.4s
296:	learn: 0.2749683	total: 5.67s	remaining: 13.4s
297:	learn: 0.2746787	total: 5.69s	remaining: 13.4s
298:	learn: 0.2743144	total: 5.71s	remaining: 13.4s
299:	learn: 0.2742161	total: 5.73s	remaining: 13.4s
300:	learn: 0.2739640	total: 5.75s	remaining: 13.3s
301:	learn: 0.2737479	total: 5.76s	remaining: 13.3s
302:	learn: 0.2733166	total: 5.79s	remaining: 13.3s
303:	learn: 0.2731112	total: 5.81s	remaining: 13.3s
304:	learn: 0.2728891	total: 5.83s	remaining: 13.3s
305:	learn: 0.2725994	total: 5.84s	remaining: 13.3s
306:	learn: 0.2723851	total: 5.86s	remaining: 13.2s
307:	learn: 0.2719409	total: 5.88s	remaining: 13.2s
308:	learn: 0.2717026	total: 5.9s	remaining: 13.2s
309:	learn: 0.2714600	total: 5.92s	remaining: 13.2s
310:	learn: 0.2712507	total: 5.94s	remaining: 13.2s
311:	learn: 0.2709506	total: 5.96s	remaining: 13.1s
312:	learn: 0.2704911	total: 5.99s	remaining: 13.1s
313:	learn: 0.2703236	total: 6.01s	remaining: 13.1s
314:	learn: 0.2700347	total: 6.03s	remaining: 13.1s
315:	learn: 0.2698594	total: 6.05s	remaining: 13.1s
316:	learn: 0.2696118	total: 6.07s	remaining: 13.1s
317:	learn: 0.2690417	total: 6.09s	remaining: 13.1s
318:	learn: 0.2687016	total: 6.11s	remaining: 13s
319:	learn: 0.2684239	total: 6.12s	remaining: 13s
320:	learn: 0.2681468	total: 6.14s	remaining: 13s
321:	learn: 0.2679534	total: 6.16s	remaining: 13s
322:	learn: 0.2676510	total: 6.17s	remaining: 12.9s
323:	learn: 0.2674140	total: 6.19s	remaining: 12.9s
324:	learn: 0.2671640	total: 6.21s	remaining: 12.9s
325:	learn: 0.2667131	total: 6.23s	remaining: 12.9s
326:	learn: 0.2664364	total: 6.25s	remaining: 12.9s
327:	learn: 0.2661154	total: 6.26s	remaining: 12.8s
328:	learn: 0.2659024	total: 6.28s	remaining: 12.8s
329:	learn: 0.2656896	total: 6.29s	remaining: 12.8s
330:	learn: 0.2655026	total: 6.31s	remaining: 12.8s
331:	learn: 0.2652732	total: 6.32s	remaining: 12.7s
332:	learn: 0.2650162	total: 6.34s	remaining: 12.7s
333:	learn: 0.2646062	total: 6.36s	remaining: 12.7s
334:	learn: 0.2641961	total: 6.37s	remaining: 12.6s
335:	learn: 0.2639380	total: 6.38s	remaining: 12.6s
336:	learn: 0.2636339	total: 6.4s	remaining: 12.6s
337:	learn: 0.2634648	total: 6.42s	remaining: 12.6s
338:	learn: 0.2631299	total: 6.44s	remaining: 12.6s
339:	learn: 0.2626969	total: 6.46s	remaining: 12.5s
340:	learn: 0.2625376	total: 6.47s	remaining: 12.5s
341:	learn: 0.2624153	total: 6.49s	remaining: 12.5s
342:	learn: 0.2621792	total: 6.5s	remaining: 12.5s
343:	learn: 0.2620601	total: 6.52s	remaining: 12.4s
344:	learn: 0.2618517	total: 6.54s	remaining: 12.4s
345:	learn: 0.2615932	total: 6.56s	remaining: 12.4s
346:	learn: 0.2614305	total: 6.58s	remaining: 12.4s
347:	learn: 0.2611970	total: 6.59s	remaining: 12.4s
348:	learn: 0.2608583	total: 6.61s	remaining: 12.3s
349:	learn: 0.2606898	total: 6.63s	remaining: 12.3s
350:	learn: 0.2604461	total: 6.65s	remaining: 12.3s
351:	learn: 0.2602405	total: 6.67s	remaining: 12.3s
```

```
352:	learn: 0.2599980	total: 6.69s	remaining: 12.3s
353:	learn: 0.2598291	total: 6.71s	remaining: 12.2s
354:	learn: 0.2595411	total: 6.73s	remaining: 12.2s
355:	learn: 0.2593528	total: 6.74s	remaining: 12.2s
356:	learn: 0.2592517	total: 6.76s	remaining: 12.2s
357:	learn: 0.2589458	total: 6.77s	remaining: 12.1s
358:	learn: 0.2586411	total: 6.8s	remaining: 12.1s
359:	learn: 0.2582643	total: 6.82s	remaining: 12.1s
360:	learn: 0.2579747	total: 6.83s	remaining: 12.1s
361:	learn: 0.2577136	total: 6.84s	remaining: 12.1s
362:	learn: 0.2576067	total: 6.86s	remaining: 12s
363:	learn: 0.2573350	total: 6.88s	remaining: 12s
364:	learn: 0.2570846	total: 6.89s	remaining: 12s
365:	learn: 0.2569065	total: 6.91s	remaining: 12s
366:	learn: 0.2565767	total: 6.93s	remaining: 11.9s
367:	learn: 0.2562966	total: 6.95s	remaining: 11.9s
368:	learn: 0.2562203	total: 6.96s	remaining: 11.9s
369:	learn: 0.2559913	total: 6.98s	remaining: 11.9s
370:	learn: 0.2556800	total: 7s	remaining: 11.9s
371:	learn: 0.2555352	total: 7.02s	remaining: 11.9s
372:	learn: 0.2553861	total: 7.04s	remaining: 11.8s
373:	learn: 0.2550712	total: 7.06s	remaining: 11.8s
374:	learn: 0.2547942	total: 7.08s	remaining: 11.8s
375:	learn: 0.2545758	total: 7.1s	remaining: 11.8s
376:	learn: 0.2545241	total: 7.12s	remaining: 11.8s
377:	learn: 0.2543207	total: 7.14s	remaining: 11.8s
378:	learn: 0.2540558	total: 7.16s	remaining: 11.7s
379:	learn: 0.2538763	total: 7.18s	remaining: 11.7s
380:	learn: 0.2536396	total: 7.2s	remaining: 11.7s
381:	learn: 0.2534406	total: 7.22s	remaining: 11.7s
382:	learn: 0.2532216	total: 7.24s	remaining: 11.7s
383:	learn: 0.2527608	total: 7.25s	remaining: 11.6s
384:	learn: 0.2524770	total: 7.27s	remaining: 11.6s
385:	learn: 0.2522338	total: 7.29s	remaining: 11.6s
386:	learn: 0.2521224	total: 7.31s	remaining: 11.6s
387:	learn: 0.2518998	total: 7.33s	remaining: 11.6s
388:	learn: 0.2516730	total: 7.35s	remaining: 11.5s
389:	learn: 0.2514511	total: 7.37s	remaining: 11.5s
390:	learn: 0.2511580	total: 7.38s	remaining: 11.5s
391:	learn: 0.2509498	total: 7.4s	remaining: 11.5s
392:	learn: 0.2508010	total: 7.41s	remaining: 11.4s
393:	learn: 0.2504680	total: 7.43s	remaining: 11.4s
394:	learn: 0.2502431	total: 7.45s	remaining: 11.4s
395:	learn: 0.2498886	total: 7.47s	remaining: 11.4s
396:	learn: 0.2497697	total: 7.48s	remaining: 11.4s
397:	learn: 0.2495204	total: 7.5s	remaining: 11.3s
398:	learn: 0.2492942	total: 7.51s	remaining: 11.3s
399:	learn: 0.2490411	total: 7.53s	remaining: 11.3s
400:	learn: 0.2489174	total: 7.55s	remaining: 11.3s
401:	learn: 0.2487167	total: 7.57s	remaining: 11.3s
402:	learn: 0.2483743	total: 7.59s	remaining: 11.2s
403:	learn: 0.2483126	total: 7.61s	remaining: 11.2s
404:	learn: 0.2481405	total: 7.63s	remaining: 11.2s
405:	learn: 0.2479741	total: 7.65s	remaining: 11.2s
406:	learn: 0.2478526	total: 7.67s	remaining: 11.2s
407:	learn: 0.2474642	total: 7.69s	remaining: 11.2s
408:	learn: 0.2472735	total: 7.71s	remaining: 11.1s
409:	learn: 0.2470251	total: 7.73s	remaining: 11.1s
410:	learn: 0.2469665	total: 7.74s	remaining: 11.1s
411:	learn: 0.2467012	total: 7.76s	remaining: 11.1s
412:	learn: 0.2465114	total: 7.79s	remaining: 11.1s
413:	learn: 0.2462933	total: 7.81s	remaining: 11.1s
414:	learn: 0.2460553	total: 7.82s	remaining: 11s
415:	learn: 0.2458673	total: 7.84s	remaining: 11s
416:	learn: 0.2458307	total: 7.86s	remaining: 11s
417:	learn: 0.2458050	total: 7.88s	remaining: 11s
```

```
418:	learn: 0.2455402	total: 7.9s	remaining: 11s
419:	learn: 0.2454385	total: 7.92s	remaining: 10.9s
420:	learn: 0.2454138	total: 7.94s	remaining: 10.9s
421:	learn: 0.2452091	total: 7.96s	remaining: 10.9s
422:	learn: 0.2450936	total: 7.97s	remaining: 10.9s
423:	learn: 0.2449103	total: 7.99s	remaining: 10.9s
424:	learn: 0.2447284	total: 8.01s	remaining: 10.8s
425:	learn: 0.2444740	total: 8.03s	remaining: 10.8s
426:	learn: 0.2442504	total: 8.04s	remaining: 10.8s
427:	learn: 0.2440234	total: 8.07s	remaining: 10.8s
428:	learn: 0.2438293	total: 8.09s	remaining: 10.8s
429:	learn: 0.2436087	total: 8.11s	remaining: 10.7s
430:	learn: 0.2433979	total: 8.12s	remaining: 10.7s
431:	learn: 0.2432203	total: 8.14s	remaining: 10.7s
432:	learn: 0.2430367	total: 8.15s	remaining: 10.7s
433:	learn: 0.2427694	total: 8.17s	remaining: 10.7s
434:	learn: 0.2425663	total: 8.18s	remaining: 10.6s
435:	learn: 0.2424114	total: 8.2s	remaining: 10.6s
436:	learn: 0.2421857	total: 8.22s	remaining: 10.6s
437:	learn: 0.2419853	total: 8.24s	remaining: 10.6s
438:	learn: 0.2418100	total: 8.26s	remaining: 10.6s
439:	learn: 0.2416578	total: 8.28s	remaining: 10.5s
440:	learn: 0.2414939	total: 8.3s	remaining: 10.5s
441:	learn: 0.2412960	total: 8.31s	remaining: 10.5s
442:	learn: 0.2410711	total: 8.33s	remaining: 10.5s
443:	learn: 0.2407957	total: 8.35s	remaining: 10.5s
444:	learn: 0.2406237	total: 8.37s	remaining: 10.4s
445:	learn: 0.2404531	total: 8.39s	remaining: 10.4s
446:	learn: 0.2404296	total: 8.4s	remaining: 10.4s
447:	learn: 0.2402700	total: 8.42s	remaining: 10.4s
448:	learn: 0.2401724	total: 8.44s	remaining: 10.4s
449:	learn: 0.2399540	total: 8.45s	remaining: 10.3s
450:	learn: 0.2397468	total: 8.46s	remaining: 10.3s
451:	learn: 0.2395874	total: 8.48s	remaining: 10.3s
452:	learn: 0.2394757	total: 8.5s	remaining: 10.3s
453:	learn: 0.2391526	total: 8.52s	remaining: 10.2s
454:	learn: 0.2389486	total: 8.54s	remaining: 10.2s
455:	learn: 0.2387521	total: 8.55s	remaining: 10.2s
456:	learn: 0.2385657	total: 8.57s	remaining: 10.2s
457:	learn: 0.2384094	total: 8.59s	remaining: 10.2s
458:	learn: 0.2381323	total: 8.61s	remaining: 10.1s
459:	learn: 0.2379596	total: 8.62s	remaining: 10.1s
460:	learn: 0.2377768	total: 8.64s	remaining: 10.1s
461:	learn: 0.2375886	total: 8.65s	remaining: 10.1s
462:	learn: 0.2375651	total: 8.67s	remaining: 10.1s
463:	learn: 0.2373433	total: 8.69s	remaining: 10s
464:	learn: 0.2371125	total: 8.7s	remaining: 10s
465:	learn: 0.2369499	total: 8.72s	remaining: 9.99s
466:	learn: 0.2367310	total: 8.74s	remaining: 9.97s
467:	learn: 0.2367095	total: 8.75s	remaining: 9.95s
468:	learn: 0.2365106	total: 8.77s	remaining: 9.93s
469:	learn: 0.2364890	total: 8.78s	remaining: 9.91s
470:	learn: 0.2362694	total: 8.8s	remaining: 9.89s
471:	learn: 0.2360595	total: 8.82s	remaining: 9.87s
472:	learn: 0.2359237	total: 8.84s	remaining: 9.85s
473:	learn: 0.2357589	total: 8.86s	remaining: 9.83s
474:	learn: 0.2355460	total: 8.88s	remaining: 9.82s
475:	learn: 0.2353242	total: 8.9s	remaining: 9.8s
476:	learn: 0.2351715	total: 8.93s	remaining: 9.79s
477:	learn: 0.2349737	total: 8.95s	remaining: 9.77s
478:	learn: 0.2347948	total: 8.97s	remaining: 9.75s
479:	learn: 0.2345506	total: 8.99s	remaining: 9.74s
480:	learn: 0.2343316	total: 9.01s	remaining: 9.72s
481:	learn: 0.2341916	total: 9.03s	remaining: 9.71s
482:	learn: 0.2339970	total: 9.06s	remaining: 9.7s
483:	learn: 0.2339665	total: 9.08s	remaining: 9.68s
```

```
484:	learn: 0.2338320	total: 9.09s	remaining: 9.66s
485:	learn: 0.2336872	total: 9.11s	remaining: 9.64s
486:	learn: 0.2335232	total: 9.13s	remaining: 9.62s
487:	learn: 0.2333792	total: 9.15s	remaining: 9.6s
488:	learn: 0.2332022	total: 9.17s	remaining: 9.58s
489:	learn: 0.2331332	total: 9.19s	remaining: 9.56s
490:	learn: 0.2329681	total: 9.2s	remaining: 9.54s
491:	learn: 0.2328321	total: 9.22s	remaining: 9.52s
492:	learn: 0.2326682	total: 9.23s	remaining: 9.5s
493:	learn: 0.2324758	total: 9.25s	remaining: 9.47s
494:	learn: 0.2323193	total: 9.27s	remaining: 9.45s
495:	learn: 0.2321425	total: 9.29s	remaining: 9.44s
496:	learn: 0.2319933	total: 9.31s	remaining: 9.42s
497:	learn: 0.2318277	total: 9.33s	remaining: 9.4s
498:	learn: 0.2315322	total: 9.35s	remaining: 9.38s
499:	learn: 0.2313906	total: 9.36s	remaining: 9.36s
500:	learn: 0.2312012	total: 9.39s	remaining: 9.35s
501:	learn: 0.2311733	total: 9.4s	remaining: 9.33s
502:	learn: 0.2310452	total: 9.42s	remaining: 9.31s
503:	learn: 0.2308628	total: 9.44s	remaining: 9.29s
504:	learn: 0.2306536	total: 9.46s	remaining: 9.27s
505:	learn: 0.2304534	total: 9.48s	remaining: 9.25s
506:	learn: 0.2302427	total: 9.5s	remaining: 9.23s
507:	learn: 0.2300662	total: 9.52s	remaining: 9.22s
508:	learn: 0.2298873	total: 9.53s	remaining: 9.2s
509:	learn: 0.2297143	total: 9.55s	remaining: 9.18s
510:	learn: 0.2295651	total: 9.57s	remaining: 9.16s
511:	learn: 0.2294264	total: 9.6s	remaining: 9.14s
512:	learn: 0.2292272	total: 9.61s	remaining: 9.13s
513:	learn: 0.2292060	total: 9.63s	remaining: 9.11s
514:	learn: 0.2290781	total: 9.65s	remaining: 9.09s
515:	learn: 0.2289366	total: 9.66s	remaining: 9.06s
516:	learn: 0.2288137	total: 9.68s	remaining: 9.04s
517:	learn: 0.2286338	total: 9.7s	remaining: 9.03s
518:	learn: 0.2284921	total: 9.72s	remaining: 9.01s
519:	learn: 0.2284662	total: 9.73s	remaining: 8.99s
520:	learn: 0.2282772	total: 9.75s	remaining: 8.96s
521:	learn: 0.2281742	total: 9.77s	remaining: 8.94s
522:	learn: 0.2279901	total: 9.78s	remaining: 8.92s
523:	learn: 0.2278396	total: 9.8s	remaining: 8.9s
524:	learn: 0.2276992	total: 9.81s	remaining: 8.88s
525:	learn: 0.2275615	total: 9.83s	remaining: 8.86s
526:	learn: 0.2273314	total: 9.84s	remaining: 8.84s
527:	learn: 0.2271808	total: 9.86s	remaining: 8.81s
528:	learn: 0.2270430	total: 9.88s	remaining: 8.79s
529:	learn: 0.2268014	total: 9.89s	remaining: 8.78s
530:	learn: 0.2265209	total: 9.92s	remaining: 8.76s
531:	learn: 0.2263794	total: 9.94s	remaining: 8.74s
532:	learn: 0.2262018	total: 9.96s	remaining: 8.72s
533:	learn: 0.2258534	total: 9.98s	remaining: 8.71s
534:	learn: 0.2256831	total: 9.99s	remaining: 8.69s
535:	learn: 0.2254942	total: 10s	remaining: 8.67s
536:	learn: 0.2253375	total: 10s	remaining: 8.65s
537:	learn: 0.2252043	total: 10s	remaining: 8.63s
538:	learn: 0.2250680	total: 10.1s	remaining: 8.61s
539:	learn: 0.2247975	total: 10.1s	remaining: 8.59s
540:	learn: 0.2246667	total: 10.1s	remaining: 8.56s
541:	learn: 0.2245075	total: 10.1s	remaining: 8.54s
542:	learn: 0.2243353	total: 10.1s	remaining: 8.53s
543:	learn: 0.2241648	total: 10.2s	remaining: 8.51s
544:	learn: 0.2238417	total: 10.2s	remaining: 8.49s
545:	learn: 0.2236453	total: 10.2s	remaining: 8.47s
546:	learn: 0.2236283	total: 10.2s	remaining: 8.45s
547:	learn: 0.2235056	total: 10.2s	remaining: 8.43s
548:	learn: 0.2233695	total: 10.2s	remaining: 8.41s
549:	learn: 0.2232381	total: 10.3s	remaining: 8.4s
```

```
550:	learn: 0.2231184	total: 10.3s	remaining: 8.38s
551:	learn: 0.2229823	total: 10.3s	remaining: 8.36s
552:	learn: 0.2227052	total: 10.3s	remaining: 8.34s
553:	learn: 0.2225477	total: 10.3s	remaining: 8.32s
554:	learn: 0.2223481	total: 10.4s	remaining: 8.3s
555:	learn: 0.2222126	total: 10.4s	remaining: 8.29s
556:	learn: 0.2220556	total: 10.4s	remaining: 8.27s
557:	learn: 0.2218929	total: 10.4s	remaining: 8.25s
558:	learn: 0.2215851	total: 10.4s	remaining: 8.23s
559:	learn: 0.2214369	total: 10.5s	remaining: 8.21s
560:	learn: 0.2212971	total: 10.5s	remaining: 8.19s
561:	learn: 0.2210794	total: 10.5s	remaining: 8.17s
562:	learn: 0.2208963	total: 10.5s	remaining: 8.15s
563:	learn: 0.2207252	total: 10.5s	remaining: 8.13s
564:	learn: 0.2207048	total: 10.5s	remaining: 8.11s
565:	learn: 0.2205761	total: 10.6s	remaining: 8.09s
566:	learn: 0.2204351	total: 10.6s	remaining: 8.07s
567:	learn: 0.2203082	total: 10.6s	remaining: 8.05s
568:	learn: 0.2201264	total: 10.6s	remaining: 8.04s
569:	learn: 0.2199468	total: 10.6s	remaining: 8.02s
570:	learn: 0.2198374	total: 10.7s	remaining: 8s
571:	learn: 0.2196894	total: 10.7s	remaining: 7.98s
572:	learn: 0.2194157	total: 10.7s	remaining: 7.97s
573:	learn: 0.2192484	total: 10.7s	remaining: 7.95s
574:	learn: 0.2190807	total: 10.7s	remaining: 7.94s
575:	learn: 0.2190469	total: 10.8s	remaining: 7.92s
576:	learn: 0.2189359	total: 10.8s	remaining: 7.9s
577:	learn: 0.2187015	total: 10.8s	remaining: 7.88s
578:	learn: 0.2185836	total: 10.8s	remaining: 7.87s
579:	learn: 0.2184044	total: 10.8s	remaining: 7.85s
580:	learn: 0.2182398	total: 10.9s	remaining: 7.84s
581:	learn: 0.2181441	total: 10.9s	remaining: 7.82s
582:	learn: 0.2179946	total: 10.9s	remaining: 7.8s
583:	learn: 0.2179340	total: 10.9s	remaining: 7.78s
584:	learn: 0.2178000	total: 10.9s	remaining: 7.76s
585:	learn: 0.2175684	total: 11s	remaining: 7.74s
586:	learn: 0.2174174	total: 11s	remaining: 7.72s
587:	learn: 0.2173047	total: 11s	remaining: 7.7s
588:	learn: 0.2170198	total: 11s	remaining: 7.68s
589:	learn: 0.2168254	total: 11s	remaining: 7.67s
590:	learn: 0.2166062	total: 11.1s	remaining: 7.65s
591:	learn: 0.2164051	total: 11.1s	remaining: 7.63s
592:	learn: 0.2162341	total: 11.1s	remaining: 7.61s
593:	learn: 0.2160758	total: 11.1s	remaining: 7.59s
594:	learn: 0.2158660	total: 11.1s	remaining: 7.57s
595:	learn: 0.2157443	total: 11.1s	remaining: 7.55s
596:	learn: 0.2155716	total: 11.2s	remaining: 7.53s
597:	learn: 0.2154150	total: 11.2s	remaining: 7.51s
598:	learn: 0.2151640	total: 11.2s	remaining: 7.49s
599:	learn: 0.2150041	total: 11.2s	remaining: 7.47s
600:	learn: 0.2148074	total: 11.2s	remaining: 7.45s
601:	learn: 0.2145814	total: 11.2s	remaining: 7.43s
602:	learn: 0.2144455	total: 11.3s	remaining: 7.41s
603:	learn: 0.2142903	total: 11.3s	remaining: 7.4s
604:	learn: 0.2141458	total: 11.3s	remaining: 7.38s
605:	learn: 0.2140034	total: 11.3s	remaining: 7.36s
606:	learn: 0.2138105	total: 11.3s	remaining: 7.34s
607:	learn: 0.2136666	total: 11.4s	remaining: 7.32s
608:	learn: 0.2135007	total: 11.4s	remaining: 7.3s
609:	learn: 0.2133251	total: 11.4s	remaining: 7.28s
610:	learn: 0.2131523	total: 11.4s	remaining: 7.26s
611:	learn: 0.2130682	total: 11.4s	remaining: 7.24s
612:	learn: 0.2129124	total: 11.4s	remaining: 7.22s
613:	learn: 0.2127505	total: 11.5s	remaining: 7.2s
614:	learn: 0.2126343	total: 11.5s	remaining: 7.18s
615:	learn: 0.2125188	total: 11.5s	remaining: 7.16s
```

```
616:	learn: 0.2123933	total: 11.5s	remaining: 7.14s
617:	learn: 0.2123108	total: 11.5s	remaining: 7.13s
618:	learn: 0.2121836	total: 11.6s	remaining: 7.11s
619:	learn: 0.2120632	total: 11.6s	remaining: 7.09s
620:	learn: 0.2118664	total: 11.6s	remaining: 7.07s
621:	learn: 0.2116975	total: 11.6s	remaining: 7.05s
622:	learn: 0.2115561	total: 11.6s	remaining: 7.04s
623:	learn: 0.2114153	total: 11.6s	remaining: 7.02s
624:	learn: 0.2112205	total: 11.7s	remaining: 7s
625:	learn: 0.2110420	total: 11.7s	remaining: 6.98s
626:	learn: 0.2108773	total: 11.7s	remaining: 6.96s
627:	learn: 0.2107448	total: 11.7s	remaining: 6.95s
628:	learn: 0.2105610	total: 11.7s	remaining: 6.93s
629:	learn: 0.2104120	total: 11.8s	remaining: 6.91s
630:	learn: 0.2102504	total: 11.8s	remaining: 6.91s
631:	learn: 0.2101537	total: 11.8s	remaining: 6.89s
632:	learn: 0.2100129	total: 11.9s	remaining: 6.88s
633:	learn: 0.2098724	total: 11.9s	remaining: 6.86s
634:	learn: 0.2096583	total: 11.9s	remaining: 6.84s
635:	learn: 0.2095164	total: 11.9s	remaining: 6.83s
636:	learn: 0.2093475	total: 12s	remaining: 6.81s
637:	learn: 0.2092558	total: 12s	remaining: 6.79s
638:	learn: 0.2090646	total: 12s	remaining: 6.78s
639:	learn: 0.2089087	total: 12s	remaining: 6.76s
640:	learn: 0.2086999	total: 12s	remaining: 6.75s
641:	learn: 0.2085543	total: 12.1s	remaining: 6.73s
642:	learn: 0.2083834	total: 12.1s	remaining: 6.72s
643:	learn: 0.2082809	total: 12.1s	remaining: 6.7s
644:	learn: 0.2080249	total: 12.1s	remaining: 6.68s
645:	learn: 0.2079106	total: 12.2s	remaining: 6.66s
646:	learn: 0.2078009	total: 12.2s	remaining: 6.64s
647:	learn: 0.2076663	total: 12.2s	remaining: 6.63s
648:	learn: 0.2075302	total: 12.2s	remaining: 6.61s
649:	learn: 0.2073523	total: 12.2s	remaining: 6.59s
650:	learn: 0.2072515	total: 12.3s	remaining: 6.58s
651:	learn: 0.2071518	total: 12.3s	remaining: 6.56s
652:	learn: 0.2069922	total: 12.3s	remaining: 6.54s
653:	learn: 0.2068608	total: 12.3s	remaining: 6.53s
654:	learn: 0.2067450	total: 12.4s	remaining: 6.51s
655:	learn: 0.2066167	total: 12.4s	remaining: 6.49s
656:	learn: 0.2064567	total: 12.4s	remaining: 6.47s
657:	learn: 0.2063038	total: 12.4s	remaining: 6.45s
658:	learn: 0.2061555	total: 12.4s	remaining: 6.43s
659:	learn: 0.2059175	total: 12.4s	remaining: 6.41s
660:	learn: 0.2058012	total: 12.5s	remaining: 6.39s
661:	learn: 0.2056832	total: 12.5s	remaining: 6.37s
662:	learn: 0.2055634	total: 12.5s	remaining: 6.35s
663:	learn: 0.2054128	total: 12.5s	remaining: 6.33s
664:	learn: 0.2052362	total: 12.5s	remaining: 6.31s
665:	learn: 0.2051397	total: 12.5s	remaining: 6.29s
666:	learn: 0.2049839	total: 12.6s	remaining: 6.27s
667:	learn: 0.2048110	total: 12.6s	remaining: 6.25s
668:	learn: 0.2046835	total: 12.6s	remaining: 6.23s
669:	learn: 0.2045773	total: 12.6s	remaining: 6.21s
670:	learn: 0.2044687	total: 12.6s	remaining: 6.19s
671:	learn: 0.2043181	total: 12.6s	remaining: 6.17s
672:	learn: 0.2041802	total: 12.7s	remaining: 6.15s
673:	learn: 0.2040448	total: 12.7s	remaining: 6.13s
674:	learn: 0.2039226	total: 12.7s	remaining: 6.11s
675:	learn: 0.2038039	total: 12.7s	remaining: 6.09s
676:	learn: 0.2037120	total: 12.7s	remaining: 6.07s
677:	learn: 0.2035912	total: 12.7s	remaining: 6.05s
678:	learn: 0.2034977	total: 12.8s	remaining: 6.03s
679:	learn: 0.2033854	total: 12.8s	remaining: 6.01s
680:	learn: 0.2032374	total: 12.8s	remaining: 5.99s
681:	learn: 0.2030712	total: 12.8s	remaining: 5.97s
```

```
682:	learn: 0.2029717	total: 12.8s	remaining: 5.95s
683:	learn: 0.2028374	total: 12.8s	remaining: 5.93s
684:	learn: 0.2026134	total: 12.9s	remaining: 5.91s
685:	learn: 0.2024452	total: 12.9s	remaining: 5.89s
686:	learn: 0.2022764	total: 12.9s	remaining: 5.87s
687:	learn: 0.2021411	total: 12.9s	remaining: 5.85s
688:	learn: 0.2019984	total: 12.9s	remaining: 5.83s
689:	learn: 0.2018939	total: 12.9s	remaining: 5.82s
690:	learn: 0.2017059	total: 13s	remaining: 5.8s
691:	learn: 0.2015830	total: 13s	remaining: 5.78s
692:	learn: 0.2013938	total: 13s	remaining: 5.76s
693:	learn: 0.2012149	total: 13s	remaining: 5.74s
694:	learn: 0.2011203	total: 13s	remaining: 5.72s
695:	learn: 0.2009910	total: 13.1s	remaining: 5.71s
696:	learn: 0.2008490	total: 13.1s	remaining: 5.69s
697:	learn: 0.2007209	total: 13.1s	remaining: 5.67s
698:	learn: 0.2006336	total: 13.1s	remaining: 5.65s
699:	learn: 0.2005111	total: 13.1s	remaining: 5.63s
700:	learn: 0.2004100	total: 13.2s	remaining: 5.62s
701:	learn: 0.2002490	total: 13.2s	remaining: 5.6s
702:	learn: 0.2001431	total: 13.2s	remaining: 5.58s
703:	learn: 0.1999525	total: 13.2s	remaining: 5.56s
704:	learn: 0.1998387	total: 13.3s	remaining: 5.55s
705:	learn: 0.1997083	total: 13.3s	remaining: 5.53s
706:	learn: 0.1995924	total: 13.3s	remaining: 5.51s
707:	learn: 0.1994780	total: 13.3s	remaining: 5.5s
708:	learn: 0.1993468	total: 13.3s	remaining: 5.47s
709:	learn: 0.1992362	total: 13.4s	remaining: 5.46s
710:	learn: 0.1991526	total: 13.4s	remaining: 5.43s
711:	learn: 0.1990002	total: 13.4s	remaining: 5.42s
712:	learn: 0.1988888	total: 13.4s	remaining: 5.4s
713:	learn: 0.1987543	total: 13.4s	remaining: 5.38s
714:	learn: 0.1986022	total: 13.4s	remaining: 5.36s
715:	learn: 0.1984771	total: 13.5s	remaining: 5.34s
716:	learn: 0.1983355	total: 13.5s	remaining: 5.32s
717:	learn: 0.1982115	total: 13.5s	remaining: 5.3s
718:	learn: 0.1981065	total: 13.5s	remaining: 5.28s
719:	learn: 0.1979598	total: 13.5s	remaining: 5.26s
720:	learn: 0.1977618	total: 13.5s	remaining: 5.24s
721:	learn: 0.1976580	total: 13.6s	remaining: 5.22s
722:	learn: 0.1975277	total: 13.6s	remaining: 5.2s
723:	learn: 0.1974255	total: 13.6s	remaining: 5.18s
724:	learn: 0.1972680	total: 13.6s	remaining: 5.16s
725:	learn: 0.1970943	total: 13.6s	remaining: 5.14s
726:	learn: 0.1969891	total: 13.6s	remaining: 5.12s
727:	learn: 0.1968977	total: 13.7s	remaining: 5.1s
728:	learn: 0.1967691	total: 13.7s	remaining: 5.08s
729:	learn: 0.1966304	total: 13.7s	remaining: 5.06s
730:	learn: 0.1964994	total: 13.7s	remaining: 5.04s
731:	learn: 0.1963686	total: 13.7s	remaining: 5.03s
732:	learn: 0.1962648	total: 13.7s	remaining: 5.01s
733:	learn: 0.1961706	total: 13.8s	remaining: 4.99s
734:	learn: 0.1960423	total: 13.8s	remaining: 4.97s
735:	learn: 0.1959417	total: 13.8s	remaining: 4.95s
736:	learn: 0.1958212	total: 13.8s	remaining: 4.93s
737:	learn: 0.1957181	total: 13.8s	remaining: 4.91s
738:	learn: 0.1955991	total: 13.8s	remaining: 4.89s
739:	learn: 0.1955082	total: 13.9s	remaining: 4.87s
740:	learn: 0.1954082	total: 13.9s	remaining: 4.85s
741:	learn: 0.1953059	total: 13.9s	remaining: 4.83s
742:	learn: 0.1951702	total: 13.9s	remaining: 4.81s
743:	learn: 0.1950954	total: 13.9s	remaining: 4.79s
744:	learn: 0.1950236	total: 13.9s	remaining: 4.77s
745:	learn: 0.1949324	total: 14s	remaining: 4.75s
746:	learn: 0.1948106	total: 14s	remaining: 4.73s
747:	learn: 0.1947091	total: 14s	remaining: 4.71s
```

```
748:	learn: 0.1946039	total: 14s	remaining: 4.69s
749:	learn: 0.1944443	total: 14s	remaining: 4.67s
750:	learn: 0.1942851	total: 14s	remaining: 4.65s
751:	learn: 0.1941161	total: 14.1s	remaining: 4.63s
752:	learn: 0.1939942	total: 14.1s	remaining: 4.62s
753:	learn: 0.1939104	total: 14.1s	remaining: 4.59s
754:	learn: 0.1938426	total: 14.1s	remaining: 4.58s
755:	learn: 0.1937614	total: 14.1s	remaining: 4.56s
756:	learn: 0.1936363	total: 14.1s	remaining: 4.54s
757:	learn: 0.1935263	total: 14.2s	remaining: 4.52s
758:	learn: 0.1934098	total: 14.2s	remaining: 4.5s
759:	learn: 0.1933373	total: 14.2s	remaining: 4.48s
760:	learn: 0.1931812	total: 14.2s	remaining: 4.46s
761:	learn: 0.1930720	total: 14.2s	remaining: 4.44s
762:	learn: 0.1929635	total: 14.3s	remaining: 4.43s
763:	learn: 0.1928835	total: 14.3s	remaining: 4.41s
764:	learn: 0.1928067	total: 14.3s	remaining: 4.39s
765:	learn: 0.1926471	total: 14.3s	remaining: 4.37s
766:	learn: 0.1925152	total: 14.3s	remaining: 4.35s
767:	learn: 0.1923927	total: 14.3s	remaining: 4.33s
768:	learn: 0.1922823	total: 14.4s	remaining: 4.31s
769:	learn: 0.1921154	total: 14.4s	remaining: 4.29s
770:	learn: 0.1919949	total: 14.4s	remaining: 4.28s
771:	learn: 0.1918926	total: 14.4s	remaining: 4.25s
772:	learn: 0.1918211	total: 14.4s	remaining: 4.24s
773:	learn: 0.1917181	total: 14.4s	remaining: 4.22s
774:	learn: 0.1916591	total: 14.5s	remaining: 4.2s
775:	learn: 0.1915373	total: 14.5s	remaining: 4.18s
776:	learn: 0.1914393	total: 14.5s	remaining: 4.16s
777:	learn: 0.1913243	total: 14.5s	remaining: 4.14s
778:	learn: 0.1912583	total: 14.5s	remaining: 4.12s
779:	learn: 0.1912260	total: 14.5s	remaining: 4.1s
780:	learn: 0.1911181	total: 14.6s	remaining: 4.08s
781:	learn: 0.1909865	total: 14.6s	remaining: 4.06s
782:	learn: 0.1908632	total: 14.6s	remaining: 4.04s
783:	learn: 0.1907360	total: 14.6s	remaining: 4.02s
784:	learn: 0.1906424	total: 14.6s	remaining: 4s
785:	learn: 0.1905113	total: 14.6s	remaining: 3.98s
786:	learn: 0.1904467	total: 14.7s	remaining: 3.97s
787:	learn: 0.1903448	total: 14.7s	remaining: 3.95s
788:	learn: 0.1902477	total: 14.7s	remaining: 3.93s
789:	learn: 0.1901782	total: 14.7s	remaining: 3.91s
790:	learn: 0.1900766	total: 14.7s	remaining: 3.89s
791:	learn: 0.1899848	total: 14.7s	remaining: 3.87s
792:	learn: 0.1898770	total: 14.8s	remaining: 3.85s
793:	learn: 0.1897526	total: 14.8s	remaining: 3.83s
794:	learn: 0.1896376	total: 14.8s	remaining: 3.81s
795:	learn: 0.1894897	total: 14.8s	remaining: 3.79s
796:	learn: 0.1893867	total: 14.8s	remaining: 3.78s
797:	learn: 0.1892671	total: 14.8s	remaining: 3.76s
798:	learn: 0.1891485	total: 14.9s	remaining: 3.74s
799:	learn: 0.1890272	total: 14.9s	remaining: 3.72s
800:	learn: 0.1889476	total: 14.9s	remaining: 3.7s
801:	learn: 0.1888926	total: 14.9s	remaining: 3.68s
802:	learn: 0.1887466	total: 14.9s	remaining: 3.66s
803:	learn: 0.1886532	total: 15s	remaining: 3.64s
804:	learn: 0.1885487	total: 15s	remaining: 3.63s
805:	learn: 0.1884400	total: 15s	remaining: 3.61s
806:	learn: 0.1883277	total: 15s	remaining: 3.59s
807:	learn: 0.1882113	total: 15s	remaining: 3.57s
808:	learn: 0.1880428	total: 15s	remaining: 3.55s
809:	learn: 0.1879659	total: 15.1s	remaining: 3.53s
810:	learn: 0.1878673	total: 15.1s	remaining: 3.52s
811:	learn: 0.1877195	total: 15.1s	remaining: 3.5s
812:	learn: 0.1876925	total: 15.1s	remaining: 3.48s
813:	learn: 0.1875827	total: 15.1s	remaining: 3.46s
```

```
814:	learn: 0.1874015	total: 15.2s	remaining: 3.44s
815:	learn: 0.1872823	total: 15.2s	remaining: 3.42s
816:	learn: 0.1871389	total: 15.2s	remaining: 3.41s
817:	learn: 0.1870282	total: 15.2s	remaining: 3.39s
818:	learn: 0.1869611	total: 15.3s	remaining: 3.37s
819:	learn: 0.1868554	total: 15.3s	remaining: 3.35s
820:	learn: 0.1867649	total: 15.3s	remaining: 3.33s
821:	learn: 0.1866653	total: 15.3s	remaining: 3.31s
822:	learn: 0.1865257	total: 15.3s	remaining: 3.3s
823:	learn: 0.1864398	total: 15.3s	remaining: 3.28s
824:	learn: 0.1863280	total: 15.4s	remaining: 3.26s
825:	learn: 0.1861762	total: 15.4s	remaining: 3.24s
826:	learn: 0.1860560	total: 15.4s	remaining: 3.22s
827:	learn: 0.1859555	total: 15.4s	remaining: 3.2s
828:	learn: 0.1858086	total: 15.4s	remaining: 3.18s
829:	learn: 0.1857233	total: 15.5s	remaining: 3.17s
830:	learn: 0.1856096	total: 15.5s	remaining: 3.15s
831:	learn: 0.1855295	total: 15.5s	remaining: 3.13s
832:	learn: 0.1854529	total: 15.5s	remaining: 3.11s
833:	learn: 0.1853420	total: 15.5s	remaining: 3.09s
834:	learn: 0.1852633	total: 15.5s	remaining: 3.07s
835:	learn: 0.1851655	total: 15.6s	remaining: 3.05s
836:	learn: 0.1850523	total: 15.6s	remaining: 3.03s
837:	learn: 0.1849468	total: 15.6s	remaining: 3.01s
838:	learn: 0.1847993	total: 15.6s	remaining: 2.99s
839:	learn: 0.1846862	total: 15.6s	remaining: 2.98s
840:	learn: 0.1845468	total: 15.6s	remaining: 2.96s
841:	learn: 0.1844317	total: 15.7s	remaining: 2.94s
842:	learn: 0.1843395	total: 15.7s	remaining: 2.92s
843:	learn: 0.1842406	total: 15.7s	remaining: 2.9s
844:	learn: 0.1841082	total: 15.7s	remaining: 2.88s
845:	learn: 0.1840060	total: 15.7s	remaining: 2.86s
846:	learn: 0.1839127	total: 15.7s	remaining: 2.84s
847:	learn: 0.1838730	total: 15.8s	remaining: 2.82s
848:	learn: 0.1837767	total: 15.8s	remaining: 2.81s
849:	learn: 0.1836744	total: 15.8s	remaining: 2.79s
850:	learn: 0.1835636	total: 15.8s	remaining: 2.77s
851:	learn: 0.1834628	total: 15.8s	remaining: 2.75s
852:	learn: 0.1833582	total: 15.8s	remaining: 2.73s
853:	learn: 0.1832431	total: 15.9s	remaining: 2.71s
854:	learn: 0.1831434	total: 15.9s	remaining: 2.69s
855:	learn: 0.1829965	total: 15.9s	remaining: 2.67s
856:	learn: 0.1829111	total: 15.9s	remaining: 2.65s
857:	learn: 0.1827899	total: 15.9s	remaining: 2.63s
858:	learn: 0.1827423	total: 15.9s	remaining: 2.62s
859:	learn: 0.1826493	total: 16s	remaining: 2.6s
860:	learn: 0.1825302	total: 16s	remaining: 2.58s
861:	learn: 0.1823801	total: 16s	remaining: 2.56s
862:	learn: 0.1823050	total: 16s	remaining: 2.54s
863:	learn: 0.1822156	total: 16s	remaining: 2.52s
864:	learn: 0.1820914	total: 16s	remaining: 2.5s
865:	learn: 0.1819844	total: 16.1s	remaining: 2.48s
866:	learn: 0.1818595	total: 16.1s	remaining: 2.46s
867:	learn: 0.1817566	total: 16.1s	remaining: 2.45s
868:	learn: 0.1816263	total: 16.1s	remaining: 2.43s
869:	learn: 0.1815108	total: 16.1s	remaining: 2.41s
870:	learn: 0.1813803	total: 16.1s	remaining: 2.39s
871:	learn: 0.1812841	total: 16.2s	remaining: 2.37s
872:	learn: 0.1812156	total: 16.2s	remaining: 2.35s
873:	learn: 0.1810773	total: 16.2s	remaining: 2.34s
874:	learn: 0.1809326	total: 16.2s	remaining: 2.32s
875:	learn: 0.1807977	total: 16.2s	remaining: 2.3s
876:	learn: 0.1806914	total: 16.3s	remaining: 2.28s
877:	learn: 0.1805569	total: 16.3s	remaining: 2.26s
878:	learn: 0.1804521	total: 16.3s	remaining: 2.24s
879:	learn: 0.1803312	total: 16.3s	remaining: 2.22s
```

```
880:	learn: 0.1802638	total: 16.3s	remaining: 2.2s
881:	learn: 0.1801537	total: 16.3s	remaining: 2.19s
882:	learn: 0.1800751	total: 16.4s	remaining: 2.17s
883:	learn: 0.1799586	total: 16.4s	remaining: 2.15s
884:	learn: 0.1798514	total: 16.4s	remaining: 2.13s
885:	learn: 0.1798187	total: 16.4s	remaining: 2.11s
886:	learn: 0.1796709	total: 16.4s	remaining: 2.09s
887:	learn: 0.1795876	total: 16.4s	remaining: 2.07s
888:	learn: 0.1795159	total: 16.5s	remaining: 2.06s
889:	learn: 0.1794336	total: 16.5s	remaining: 2.04s
890:	learn: 0.1793410	total: 16.5s	remaining: 2.02s
891:	learn: 0.1792623	total: 16.5s	remaining: 2s
892:	learn: 0.1791633	total: 16.5s	remaining: 1.98s
893:	learn: 0.1790979	total: 16.5s	remaining: 1.96s
894:	learn: 0.1789995	total: 16.6s	remaining: 1.94s
895:	learn: 0.1789065	total: 16.6s	remaining: 1.92s
896:	learn: 0.1788343	total: 16.6s	remaining: 1.91s
897:	learn: 0.1787206	total: 16.6s	remaining: 1.89s
898:	learn: 0.1785994	total: 16.6s	remaining: 1.87s
899:	learn: 0.1784772	total: 16.6s	remaining: 1.85s
900:	learn: 0.1783438	total: 16.7s	remaining: 1.83s
901:	learn: 0.1782642	total: 16.7s	remaining: 1.81s
902:	learn: 0.1781333	total: 16.7s	remaining: 1.79s
903:	learn: 0.1780521	total: 16.7s	remaining: 1.77s
904:	learn: 0.1779421	total: 16.7s	remaining: 1.75s
905:	learn: 0.1778208	total: 16.7s	remaining: 1.74s
906:	learn: 0.1776993	total: 16.8s	remaining: 1.72s
907:	learn: 0.1776187	total: 16.8s	remaining: 1.7s
908:	learn: 0.1774760	total: 16.8s	remaining: 1.68s
909:	learn: 0.1773354	total: 16.8s	remaining: 1.66s
910:	learn: 0.1772730	total: 16.8s	remaining: 1.64s
911:	learn: 0.1771922	total: 16.9s	remaining: 1.63s
912:	learn: 0.1771217	total: 16.9s	remaining: 1.61s
913:	learn: 0.1770239	total: 16.9s	remaining: 1.59s
914:	learn: 0.1769451	total: 16.9s	remaining: 1.57s
915:	learn: 0.1769022	total: 16.9s	remaining: 1.55s
916:	learn: 0.1768400	total: 16.9s	remaining: 1.53s
917:	learn: 0.1767630	total: 16.9s	remaining: 1.51s
918:	learn: 0.1766463	total: 17s	remaining: 1.5s
919:	learn: 0.1765362	total: 17s	remaining: 1.48s
920:	learn: 0.1763987	total: 17s	remaining: 1.46s
921:	learn: 0.1762748	total: 17s	remaining: 1.44s
922:	learn: 0.1761895	total: 17s	remaining: 1.42s
923:	learn: 0.1760816	total: 17.1s	remaining: 1.4s
924:	learn: 0.1760474	total: 17.1s	remaining: 1.38s
925:	learn: 0.1759466	total: 17.1s	remaining: 1.36s
926:	learn: 0.1758475	total: 17.1s	remaining: 1.35s
927:	learn: 0.1757882	total: 17.1s	remaining: 1.33s
928:	learn: 0.1756939	total: 17.1s	remaining: 1.31s
929:	learn: 0.1756296	total: 17.2s	remaining: 1.29s
930:	learn: 0.1755531	total: 17.2s	remaining: 1.27s
931:	learn: 0.1754644	total: 17.2s	remaining: 1.25s
932:	learn: 0.1753724	total: 17.2s	remaining: 1.24s
933:	learn: 0.1752900	total: 17.2s	remaining: 1.22s
934:	learn: 0.1751712	total: 17.3s	remaining: 1.2s
935:	learn: 0.1750910	total: 17.3s	remaining: 1.18s
936:	learn: 0.1750628	total: 17.3s	remaining: 1.16s
937:	learn: 0.1750309	total: 17.3s	remaining: 1.14s
938:	learn: 0.1749298	total: 17.3s	remaining: 1.13s
939:	learn: 0.1748403	total: 17.3s	remaining: 1.11s
940:	learn: 0.1747669	total: 17.4s	remaining: 1.09s
941:	learn: 0.1746853	total: 17.4s	remaining: 1.07s
942:	learn: 0.1745755	total: 17.4s	remaining: 1.05s
943:	learn: 0.1744726	total: 17.4s	remaining: 1.03s
944:	learn: 0.1743726	total: 17.4s	remaining: 1.01s
945:	learn: 0.1743043	total: 17.4s	remaining: 996ms
```

```
946:	learn: 0.1742363	total: 17.5s	remaining: 977ms
947:	learn: 0.1741959	total: 17.5s	remaining: 959ms
948:	learn: 0.1741002	total: 17.5s	remaining: 940ms
949:	learn: 0.1740739	total: 17.5s	remaining: 922ms
950:	learn: 0.1739405	total: 17.5s	remaining: 903ms
951:	learn: 0.1738429	total: 17.5s	remaining: 885ms
952:	learn: 0.1737440	total: 17.6s	remaining: 866ms
953:	learn: 0.1736278	total: 17.6s	remaining: 848ms
954:	learn: 0.1734900	total: 17.6s	remaining: 829ms
955:	learn: 0.1734046	total: 17.6s	remaining: 811ms
956:	learn: 0.1733172	total: 17.6s	remaining: 792ms
957:	learn: 0.1732051	total: 17.6s	remaining: 774ms
958:	learn: 0.1730937	total: 17.7s	remaining: 755ms
959:	learn: 0.1730207	total: 17.7s	remaining: 737ms
960:	learn: 0.1729372	total: 17.7s	remaining: 718ms
961:	learn: 0.1728308	total: 17.7s	remaining: 700ms
962:	learn: 0.1727234	total: 17.7s	remaining: 681ms
963:	learn: 0.1726516	total: 17.7s	remaining: 663ms
964:	learn: 0.1725212	total: 17.8s	remaining: 644ms
965:	learn: 0.1724377	total: 17.8s	remaining: 626ms
966:	learn: 0.1723482	total: 17.8s	remaining: 608ms
967:	learn: 0.1722480	total: 17.8s	remaining: 589ms
968:	learn: 0.1721605	total: 17.8s	remaining: 571ms
969:	learn: 0.1720340	total: 17.9s	remaining: 552ms
970:	learn: 0.1719318	total: 17.9s	remaining: 534ms
971:	learn: 0.1718580	total: 17.9s	remaining: 515ms
972:	learn: 0.1717541	total: 17.9s	remaining: 497ms
973:	learn: 0.1716414	total: 17.9s	remaining: 478ms
974:	learn: 0.1715517	total: 17.9s	remaining: 460ms
975:	learn: 0.1714704	total: 17.9s	remaining: 441ms
976:	learn: 0.1713987	total: 18s	remaining: 423ms
977:	learn: 0.1712982	total: 18s	remaining: 405ms
978:	learn: 0.1712051	total: 18s	remaining: 386ms
979:	learn: 0.1710958	total: 18s	remaining: 368ms
980:	learn: 0.1709796	total: 18s	remaining: 349ms
981:	learn: 0.1709007	total: 18.1s	remaining: 331ms
982:	learn: 0.1708171	total: 18.1s	remaining: 312ms
983:	learn: 0.1707177	total: 18.1s	remaining: 294ms
984:	learn: 0.1706525	total: 18.1s	remaining: 276ms
985:	learn: 0.1705481	total: 18.1s	remaining: 257ms
986:	learn: 0.1704410	total: 18.1s	remaining: 239ms
987:	learn: 0.1703672	total: 18.2s	remaining: 221ms
988:	learn: 0.1703173	total: 18.2s	remaining: 202ms
989:	learn: 0.1702322	total: 18.2s	remaining: 184ms
990:	learn: 0.1701582	total: 18.2s	remaining: 165ms
991:	learn: 0.1700870	total: 18.2s	remaining: 147ms
992:	learn: 0.1699972	total: 18.3s	remaining: 129ms
993:	learn: 0.1699775	total: 18.3s	remaining: 110ms
994:	learn: 0.1699310	total: 18.3s	remaining: 91.9ms
995:	learn: 0.1698776	total: 18.3s	remaining: 73.5ms
996:	learn: 0.1698303	total: 18.3s	remaining: 55.1ms
997:	learn: 0.1697276	total: 18.3s	remaining: 36.7ms
998:	learn: 0.1696425	total: 18.3s	remaining: 18.4ms
999:	learn: 0.1695313	total: 18.4s	remaining: 0us
```

Out[381…

```
0.9033374536464771
```

In [382…

```python
# Make predictions on the model
cat_pred = cat.predict(X_test)
cat_pred[:10]
```

Out[382…

```
array([0, 0, 1, 0, 1, 1, 0, 0, 0, 1], dtype=int64)
```

In [382…

```python
y_test[:10]
```

Out[382…
```
4981     0
5421     0
16026    1
8057     0
119      0
18553    1
9814     0
9787     1
2699     0
19323    1
Name: target, dtype: int64
```

In [382…
```
print(classification_report(y_test, cat_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90      2005
           1       0.92      0.88      0.90      2040

    accuracy                           0.90      4045
   macro avg       0.90      0.90      0.90      4045
weighted avg       0.90      0.90      0.90      4045
```

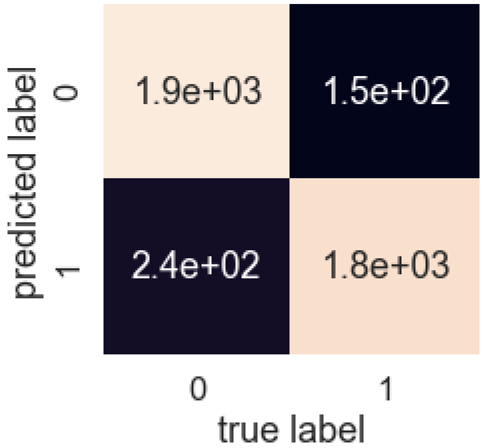
In [382…
```
print('Precision Score:', round(precision_score(y_test, cat_pred), 2))
print('Recall Score:', round(recall_score(y_test, cat_pred), 2))
print('F1 Score:', round(f1_score(y_test, cat_pred), 2))
print('Accuracy Score:', round(accuracy_score(y_test, cat_pred), 2))
print('ROC AUC: ', round(roc_auc_score(y_test, cat_pred), 2))
```

```
Precision Score: 0.92
Recall Score: 0.88
F1 Score: 0.9
Accuracy Score: 0.9
ROC AUC:  0.9
```

5.5.1 Confusion Matrix of CatBoostClassifier Model

In [382…
```
sns.set(font_scale=1.5)

def plot_conf_mat(y_test, cat_pred):
    """
    Plots a confusion matrix using Seaborn's heatmap().
    """
    fig, ax = plt.subplots(figsize=(3, 3))
    ax = sns.heatmap(confusion_matrix(y_test, cat_pred),
                     annot=True,
                     cbar=False)
    plt.xlabel("true label")
    plt.ylabel("predicted label")

plot_conf_mat(y_test, cat_pred)
print(confusion_matrix(y_test, cat_pred))
```

```
[[1851  154]
 [ 237 1803]]
```

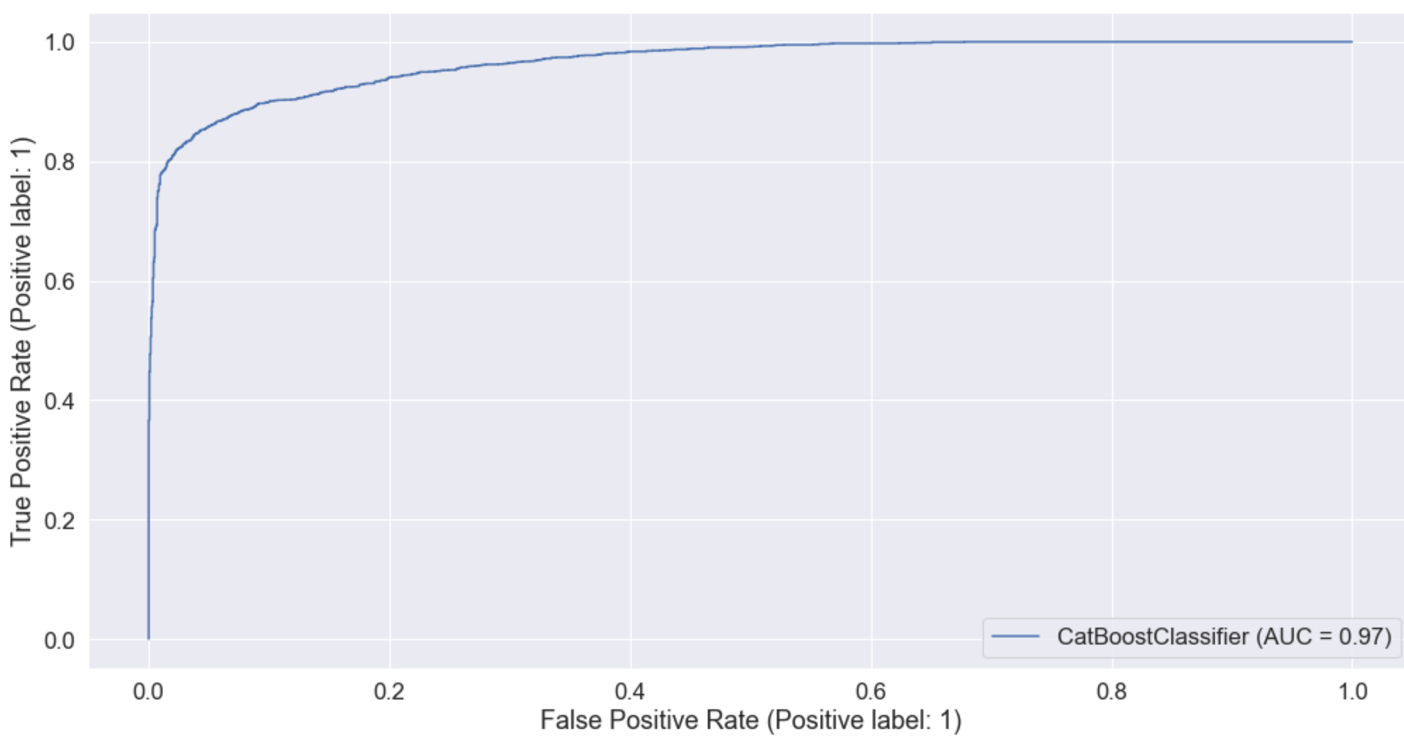


You can see the model gets confused (predicts the wrong label). In essence, there are 154 occasaions where the model predicted 0 when it should've been 1 (false negative) and 237 occasions where the model predicted 1 instead of 0 (false positive).

5.5.2 ROC Curve and AUC Scores for the CatBoostClassifier Model

In [382…

```
# Plot ROC curve and calculate AUC metric
plot_roc_curve(cat, X_test, y_test);
```



This is great, the model does far better than guessing which would be a line going from the bottom left corner to the top right corner, AUC = 0.97. But a perfect model would achieve an AUC score of 1.0.

5.6 XGBoost

In [382…

```
np. random.seed(42)

# Instantiate the model
xg = XGBClassifier()

# Fit the model to the training data
xg.fit(X_train, y_train)
```

```
# Score the model on the test data
xg.score(X_test, y_test)
```

```
[09:51:49] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learn
er.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objecti
ve 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if
you'd like to restore the old behavior.
```

Out[382…

```
0.907292954264524
```

In [382…

```
# Make predictions on the model
xg_pred = xg.predict(X_test)
xg_pred[:10]
```

Out[382…

```
array([0, 1, 1, 0, 1, 1, 0, 0, 0, 1], dtype=int64)
```

In [382…

```
y_test[:10]
```

Out[382…

```
4981     0
5421     0
16026    1
8057     0
119      0
18553    1
9814     0
9787     1
2699     0
19323    1
Name: target, dtype: int64
```

In [382…

```
print(classification_report(y_test, xg_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.92      0.91      2005
           1       0.92      0.89      0.91      2040

    accuracy                           0.91      4045
   macro avg       0.91      0.91      0.91      4045
weighted avg       0.91      0.91      0.91      4045
```

In [383…

```
print('Precision Score:', round(precision_score(y_test, xg_pred), 2))
print('Recall Score:', round(recall_score(y_test, xg_pred), 2))
print('F1 Score:', round(f1_score(y_test, xg_pred), 2))
print('Accuracy Score:', round(accuracy_score(y_test, xg_pred), 2))
print('ROC AUC: ', round(roc_auc_score(y_test, xg_pred), 2))
```

```
Precision Score: 0.92
Recall Score: 0.89
F1 Score: 0.91
Accuracy Score: 0.91
ROC AUC:  0.91
```

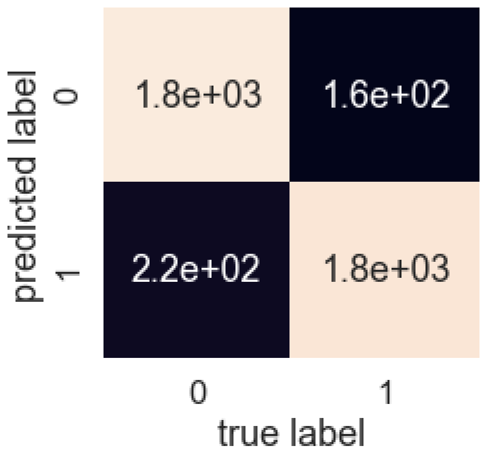
5.6.1 Confusion Matrix of XGBClassifier Model

In [383…

```
sns.set(font_scale=1.5)

def plot_conf_mat(y_test, xg_pred):
```

```
    """
    Plots a confusion matrix using Seaborn's heatmap().
    """
    fig, ax = plt.subplots(figsize=(3, 3))
    ax = sns.heatmap(confusion_matrix(y_test, xg_pred),
                     annot=True,
                     cbar=False)
    plt.xlabel("true label")
    plt.ylabel("predicted label")

plot_conf_mat(y_test, xg_pred)
print(confusion_matrix(y_test, xg_pred))
```

```
[[1847  158]
 [ 217 1823]]
```

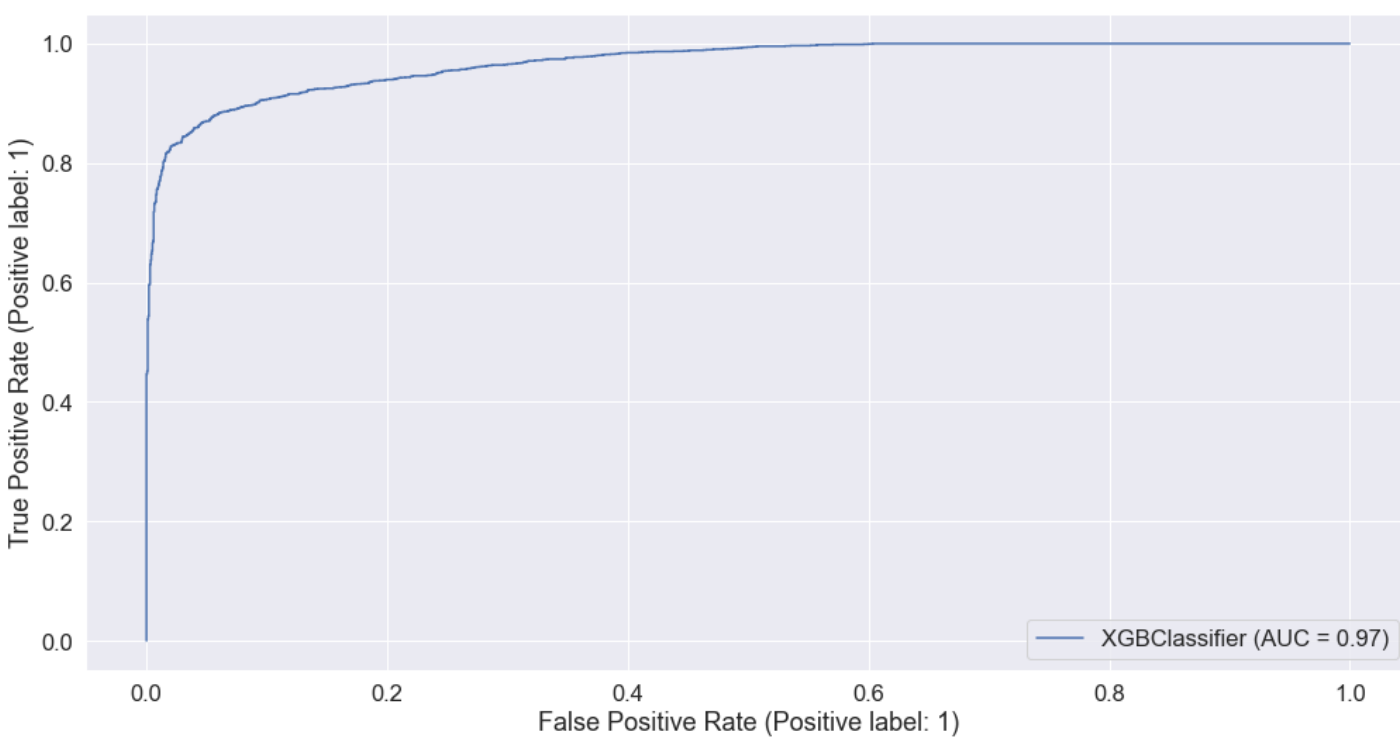


You can see the model gets confused (predicts the wrong label). In essence, there are 158 occasaions where the model predicted 0 when it should've been 1 (false negative) and 217 occasions where the model predicted 1 instead of 0 (false positive).

5.6.2 ROC Curve and AUC Scores for the XGBClassifier Model

In [383…

```
# Plot ROC curve and calculate AUC metric
plot_roc_curve(xg, X_test, y_test);
```



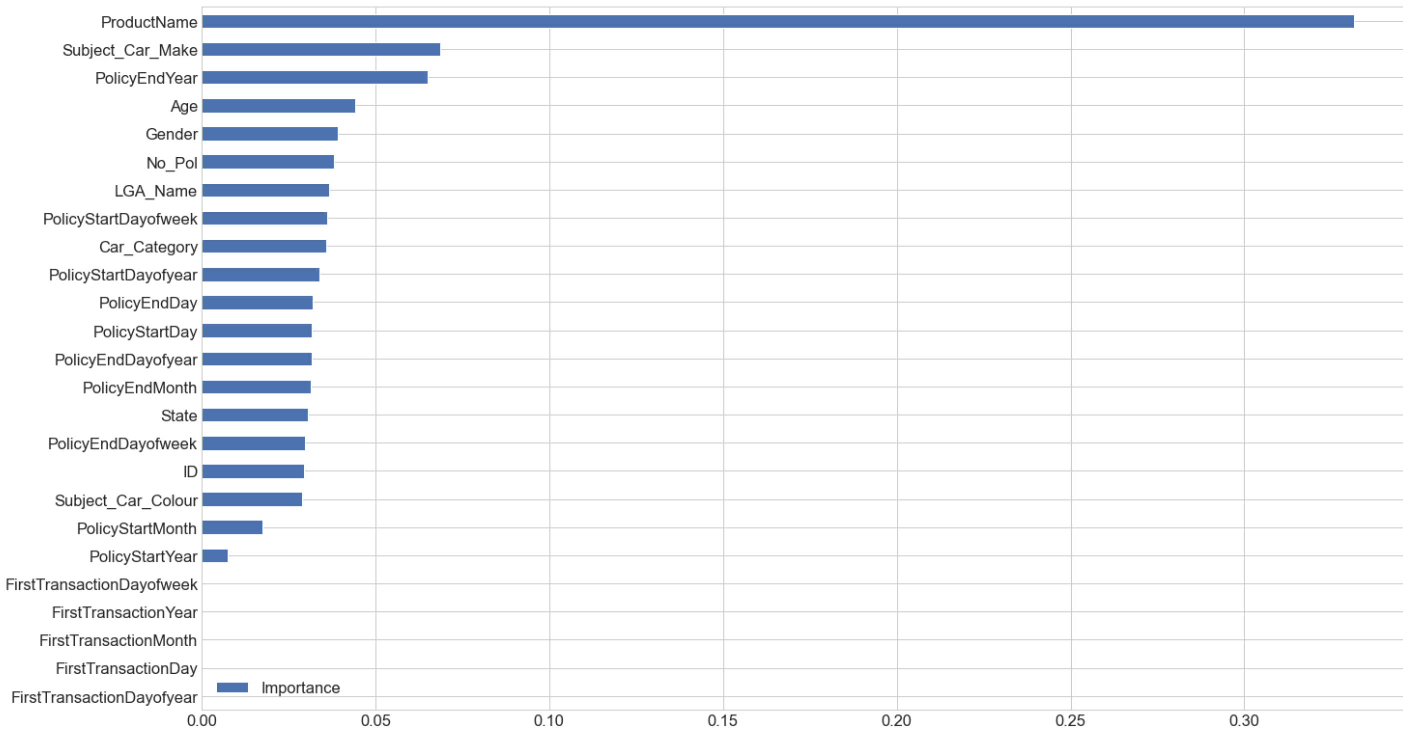
This is great, the model does far better than guessing which would be a line going from the bottom left corner to the top right corner, AUC = 0.97. But a perfect model would achieve an AUC score of 1.0.

Feature Importance

In [383…

```
# Using XGBoost to gain an insight on Feature Importance
clf = XGBClassifier()
clf.fit(train.drop('target', axis=1), train['target'])

plt.style.use('seaborn-whitegrid')
importance = clf.feature_importances_
importance = pd.DataFrame(importance, index=train.drop('target', axis=1).columns, columns=
importance.sort_values(by='Importance', ascending=True).plot(kind='barh', figsize=(20, len
```

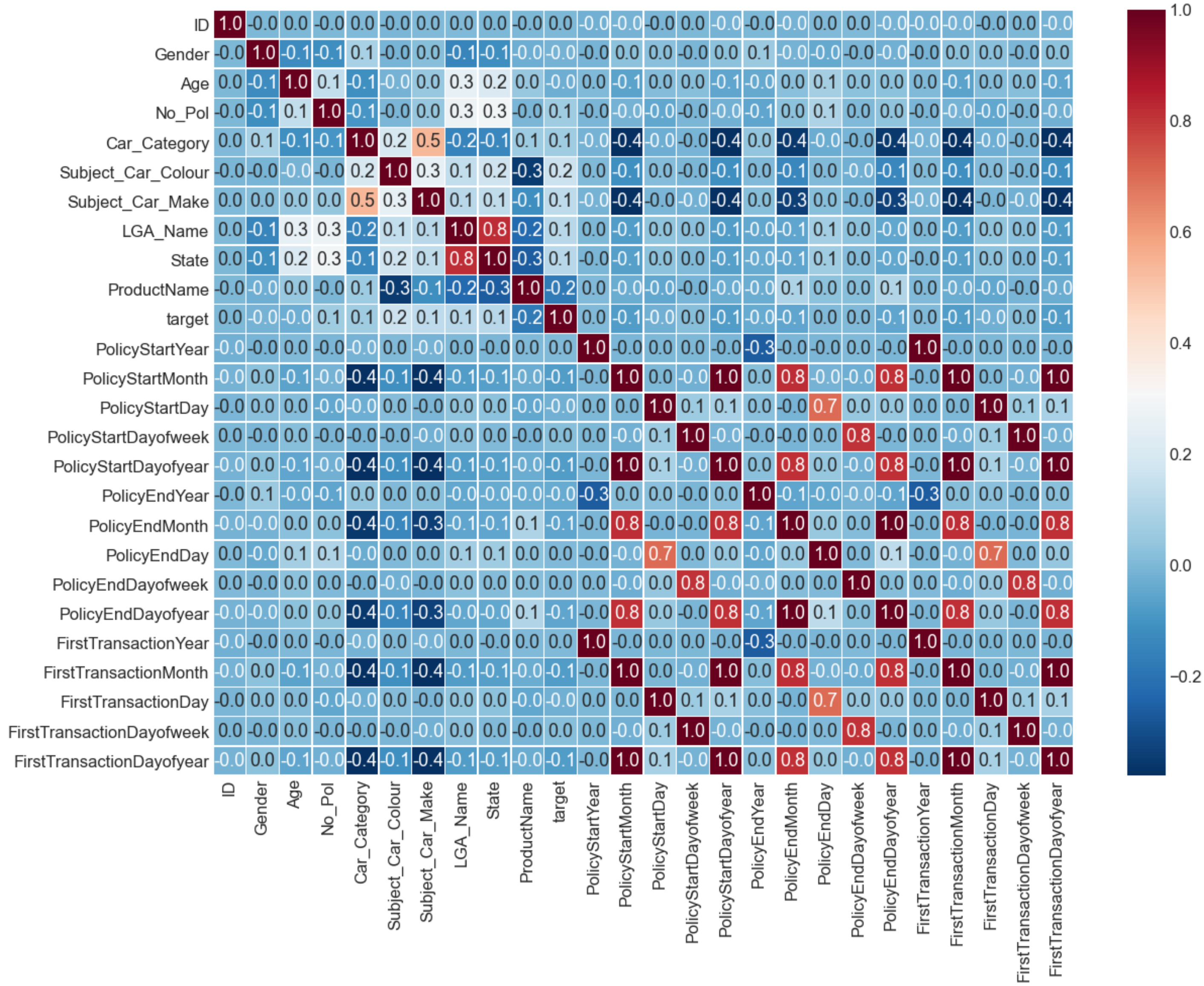
```
[09:53:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```



Correlation between independent variables

In [383…

```
# Find the correlation between the independent variables
corr_matrix = train.corr()
plt.figure(figsize=(17, 13))
sns.heatmap(corr_matrix,
            annot=True,
            linewidths=0.5,
            fmt='.1f',
            cmap="RdBu_r");
```



Make predictions on test data

Now we've got a trained model, it's time to make predictions on the test data.

So what we're doing is trying to use the patterns our model has learned in the training data to predict whether a customer will claim insurance or not with characteristics it's never seen before but are assumed to be similar to that of those in the training data.

In [383…

```
test = pd.read_csv('AutoInland-Vehicle-insurance-claim-challenge/Test.csv',
                    low_memory=False,
                     parse_dates=['Policy Start Date', 'Policy End Date', 'First Transacti
test.head()
```

Out[383…

	ID	Policy Start Date	Policy End Date	Gender	Age	First Transaction Date	No_Pol	Car_Category	Subject_Car_Colour	Subject_Car_M
0	ID_01QM0NU	2010-10-23	2011-10-22	Female	46	2010-10-23	1	NaN	NaN	I
1	ID_024NJLZ	2010-10-14	2011-10-13	Male	32	2010-10-14	1	NaN	NaN	I
2	ID_02NOVWQ	2010-08-29	2011-08-28	Female	45	2010-08-29	2	Saloon	Black	Hc

	ID	Policy Start Date	Policy End Date	Gender	Age	First Transaction Date	No_Pol	Car_Category	Subject_Car_Colour	Subject_Car_M
3	ID_02VSP68	2010-06-13	2011-06-12	Female	58	2010-06-13	1	Saloon	NaN	TOY
4	ID_02YB37K	2010-07-01	2011-06-30	NaN	120	2010-07-01	1	Saloon	Red	Hyu

Preprocessing the data(getting the test data in the same format as our training data)

Our model has been trained on data formatted in the same way as the training data. This means in order to make predictions on the test data, we need to take the same steps we used to preprocess the training data to preprocess the test data. Remember: Whatever you do to the training data, you have to do to the test data. Let's create a function for doing so (by copying the preprocessing steps we used above).

In [383…

```
def preprocess_data(test):
    """
    performs transformations on test df and returns transformed test df
    """
    # Add datetime for Policy Start Date
    test['PolicyStartYear'] = test['Policy Start Date'].dt.year
    test['PolicyStartMonth'] = test['Policy Start Date'].dt.month
    test['PolicyStartDay'] = test['Policy Start Date'].dt.day
    test['PolicyStartDayofweek'] = test['Policy Start Date'].dt.dayofweek
    test['PolicyStartDayofyear'] = test['Policy Start Date'].dt.dayofyear

    # Drop original PolicyStartDate
    test.drop("Policy Start Date", axis=1, inplace=True)

    # Add datetime for Policy End Date
    test['PolicyEndYear'] = test['Policy End Date'].dt.year
    test['PolicyEndMonth'] = test['Policy End Date'].dt.month
    test['PolicyEndDay'] = test['Policy End Date'].dt.day
    test['PolicyEndDayofweek'] = test['Policy End Date'].dt.dayofweek
    test['PolicyEndDayofyear'] = test['Policy End Date'].dt.dayofyear

    # Drop original PolicyEndDate
    test.drop("Policy End Date", axis=1, inplace=True)

    # Add datetime for FirstTransactionDate
    test['FirstTransactionYear'] = test['First Transaction Date'].dt.year
    test['FirstTransactionMonth'] = test['First Transaction Date'].dt.month
    test['FirstTransactionDay'] = test['First Transaction Date'].dt.day
    test['FirstTransactionDayofweek'] = test['First Transaction Date'].dt.dayofweek
    test['FirstTransactionDayofyear'] = test['First Transaction Date'].dt.dayofyear

    # Drop original FirstTransactionDate
    test.drop("First Transaction Date", axis=1, inplace=True)

    test.reset_index(drop=True)

    # Find the columns which contains strings
    for label, content in test.drop(['ID', 'ProductName'], axis=1).items():
        if pd.api.types.is_string_dtype(content):
            print(label)

    # This will turn all strings values into categories
    for label, content in test.drop(['ID'], axis=1).items():
        if pd.api.types.is_string_dtype(content):
```

```
            test[label]=content.astype('category').cat.as_ordered()

    for label, content in test.items():
        if pd.api.types.is_categorical_dtype(content):
            print(label)

    # Check for which categorical columns have null(missing) values
    for label, content in test.items():
        if pd.api.types.is_categorical_dtype(content):
            if pd.isnull(content).sum():
                print(label)

    # Turn categorical variables into numbers
    for label, content in test.drop(['ID'], axis=1).items():

        # Check columns which are not numeric
        if not pd.api.types.is_numeric_dtype(content):

            # Add binary column to indicate whether sample had missing value
             test[label + '_is_missing'] = pd.isnull(content)

            # Turn categories into numbers and add +1 because pandas encodes missing categ
             test[label] = pd.Categorical(content).codes + 1

    return(test)
```

In [383…

```
# Process the test data
test = preprocess_data(test)
test.head()
```

```
Gender
Car_Category
Subject_Car_Colour
Subject_Car_Make
LGA_Name
State
Gender
Car_Category
Subject_Car_Colour
Subject_Car_Make
LGA_Name
State
ProductName
Gender
Car_Category
Subject_Car_Colour
Subject_Car_Make
LGA_Name
State
```

Out[383…

	ID	Gender	Age	No_Pol	Car_Category	Subject_Car_Colour	Subject_Car_Make	LGA_Name	State	Pro
0	ID_01QM0NU	2	46	1	0	0	10	10	4	
1	ID_024NJLZ	4	32	1	0	0	0	70	8	
2	ID_02NOVWQ	2	45	2	7	4	12	113	3	
3	ID_02VSP68	2	58	1	7	0	35	0	0	
4	ID_02YB37K	0	120	1	7	13	13	110	30	

5 rows × 32 columns

In [383…

```python
# Drop the _is_missing column
test.drop(['Gender_is_missing', 'Car_Category_is_missing', 'Subject_Car_Colour_is_missing'
        'Subject_Car_Make_is_missing', 'LGA_Name_is_missing', 'State_is_missing', 'Product
test.head()
```

Out[383…

	ID	Gender	Age	No_Pol	Car_Category	Subject_Car_Colour	Subject_Car_Make	LGA_Name	State	Pro
0	ID_01QM0NU	2	46	1	0	0	10	10	4	
1	ID_024NJLZ	4	32	1	0	0	0	70	8	
2	ID_02NOVWQ	2	45	2	7	4	12	113	3	
3	ID_02VSP68	2	58	1	7	0	35	0	0	
4	ID_02YB37K	0	120	1	7	13	13	110	30	

5 rows × 25 columns

In [384…

```python
test.columns
```

Out[384…

```
Index(['ID', 'Gender', 'Age', 'No_Pol', 'Car_Category', 'Subject_Car_Colour',
       'Subject_Car_Make', 'LGA_Name', 'State', 'ProductName',
       'PolicyStartYear', 'PolicyStartMonth', 'PolicyStartDay',
       'PolicyStartDayofweek', 'PolicyStartDayofyear', 'PolicyEndYear',
       'PolicyEndMonth', 'PolicyEndDay', 'PolicyEndDayofweek',
       'PolicyEndDayofyear', 'FirstTransactionYear', 'FirstTransactionMonth',
       'FirstTransactionDay', 'FirstTransactionDayofweek',
       'FirstTransactionDayofyear'],
      dtype='object')
```

In [384…

```python
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1202 entries, 0 to 1201
Data columns (total 25 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   ID                         1202 non-null   object
 1   Gender                     1202 non-null   int8  
 2   Age                        1202 non-null   int64 
 3   No_Pol                     1202 non-null   int64 
 4   Car_Category               1202 non-null   int8  
 5   Subject_Car_Colour         1202 non-null   int8  
 6   Subject_Car_Make           1202 non-null   int8  
 7   LGA_Name                   1202 non-null   int8  
 8   State                      1202 non-null   int8  
 9   ProductName                1202 non-null   int8  
 10  PolicyStartYear            1202 non-null   int64 
 11  PolicyStartMonth           1202 non-null   int64 
 12  PolicyStartDay             1202 non-null   int64 
 13  PolicyStartDayofweek       1202 non-null   int64 
 14  PolicyStartDayofyear       1202 non-null   int64 
 15  PolicyEndYear              1202 non-null   int64 
 16  PolicyEndMonth             1202 non-null   int64 
 17  PolicyEndDay               1202 non-null   int64 
 18  PolicyEndDayofweek         1202 non-null   int64 
 19  PolicyEndDayofyear         1202 non-null   int64 
 20  FirstTransactionYear       1202 non-null   int64 
 21  FirstTransactionMonth      1202 non-null   int64 
 22  FirstTransactionDay        1202 non-null   int64 
 23  FirstTransactionDayofweek  1202 non-null   int64 
 24  FirstTransactionDayofyear  1202 non-null   int64 
```

```
dtypes: int64(17), int8(7), object(1)
memory usage: 177.4+ KB
```

In [384…

```
# This will turn ID strings values into categories
for label, content in test.items():
    if pd.api.types.is_string_dtype(content):
        test['ID']=content.astype('category').cat.as_ordered()
```

In [384…

```
# Turn ID categorical variables into integer
for label, content in test.items():
    # Turn categories into numbers and add +1 because pandas encodes missing categories as
        test['ID'] = pd.Categorical(content).codes + 1
```

In [384…

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1202 entries, 0 to 1201
Data columns (total 25 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   ID                         1202 non-null   int16
 1   Gender                     1202 non-null   int8
 2   Age                        1202 non-null   int64
 3   No_Pol                     1202 non-null   int64
 4   Car_Category               1202 non-null   int8
 5   Subject_Car_Colour         1202 non-null   int8
 6   Subject_Car_Make           1202 non-null   int8
 7   LGA_Name                   1202 non-null   int8
 8   State                      1202 non-null   int8
 9   ProductName                1202 non-null   int8
 10  PolicyStartYear            1202 non-null   int64
 11  PolicyStartMonth           1202 non-null   int64
 12  PolicyStartDay             1202 non-null   int64
 13  PolicyStartDayofweek       1202 non-null   int64
 14  PolicyStartDayofyear       1202 non-null   int64
 15  PolicyEndYear              1202 non-null   int64
 16  PolicyEndMonth             1202 non-null   int64
 17  PolicyEndDay               1202 non-null   int64
 18  PolicyEndDayofweek         1202 non-null   int64
 19  PolicyEndDayofyear         1202 non-null   int64
 20  FirstTransactionYear       1202 non-null   int64
 21  FirstTransactionMonth      1202 non-null   int64
 22  FirstTransactionDay        1202 non-null   int64
 23  FirstTransactionDayofweek  1202 non-null   int64
 24  FirstTransactionDayofyear  1202 non-null   int64
dtypes: int16(1), int64(17), int8(7)
memory usage: 170.3 KB
```

In [384…

```
# Make predictions on the test dataset using the best model
test_preds = xg.predict(test)
test_preds[:10]
```

Out[384…

```
array([1, 0, 1, 0, 0, 0, 0, 1, 0, 0], dtype=int64)
```

In [384…

```
y_test[:10]
```

Out[384…

```
4981     0
5421     0
16026    1
8057     0
```

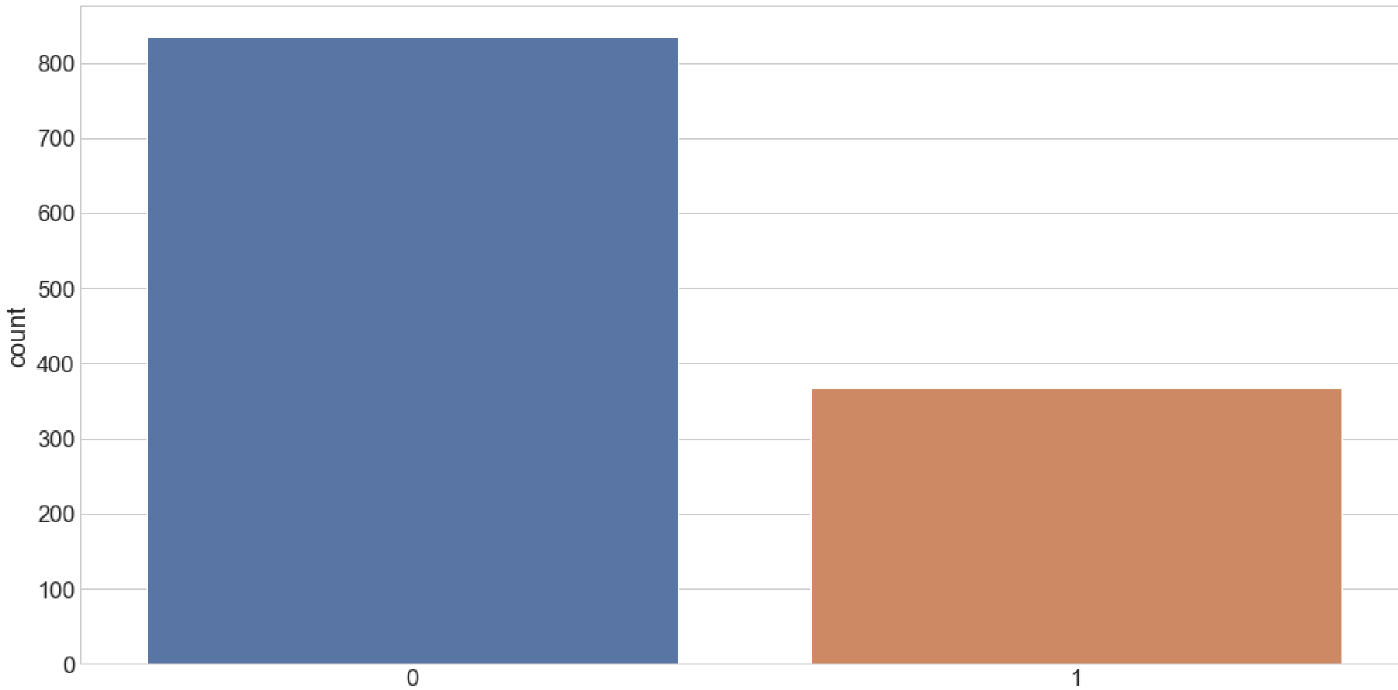
```
119      0
18553    1
9814     0
9787     1
2699     0
19323    1
Name: target, dtype: int64
```

In [384…

```
sns.countplot(test_preds)
```

Out[384…

```
<AxesSubplot:ylabel='count'>
```



In [384…

```
set(test.ID==ss.ID)
```

Out[384…

```
{False}
```

In [384…

```
test.ID
```

Out[384…

```
0       258
1       249
2       211
3       142
4       158
       ...
1197    294
1198     13
1199    181
1200     44
1201     69
Name: ID, Length: 1202, dtype: int16
```

In [385…

```
ss.head()
```

Out[385…

	ID	target
0	ID_01QM0NU	0
1	ID_024NJLZ	0

	ID	target
2	ID_02NOVWQ	0
3	ID_02VSP68	0
4	ID_02YB37K	0

In [385…

```
sub_file = ss.copy()
```

In [385…

```
sub_file.target = test_preds
```

In [385…

```
sub_file.head()
```

Out[385…

	ID	target
0	ID_01QM0NU	1
1	ID_024NJLZ	0
2	ID_02NOVWQ	1
3	ID_02VSP68	0
4	ID_02YB37K	0

In [385…

```
sub_file.to_csv('base_model_pred_model.csv', index=False)
```

6. Conclusion

Given its high scores across the board, XGBoost classifier performed slightly better than all the other machine learning models. With a high precision, recall and F1 score. This XGBoost model should be quite reliable at predicting which customers will claim insurance within the first 3 months.

Thanks for viewing this Notebook.

In [ ]:

```

```