Introduction

This dataset is a classification dataset and it is for Vehicle Insurance of a company. It is built based on customers information if the customer will be applying for Vehicle insurance or not.

Problem Statement:

Building a model to predict whether a customer would be interested in Vehicle Insurance is extremely helpful for the company.

Importance

Doing this can then help the company planning its communication strategy to reach out to those customers and optimize its business model and revenue. And this will make the company profit increase.

Import Libraries

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,
ConfusionMatrixDisplay
import seaborn as sns
```

Datasets

The dataset for this project have been gathered by the Insurance company .
Two csv files have been compiled, they contain the folowing:

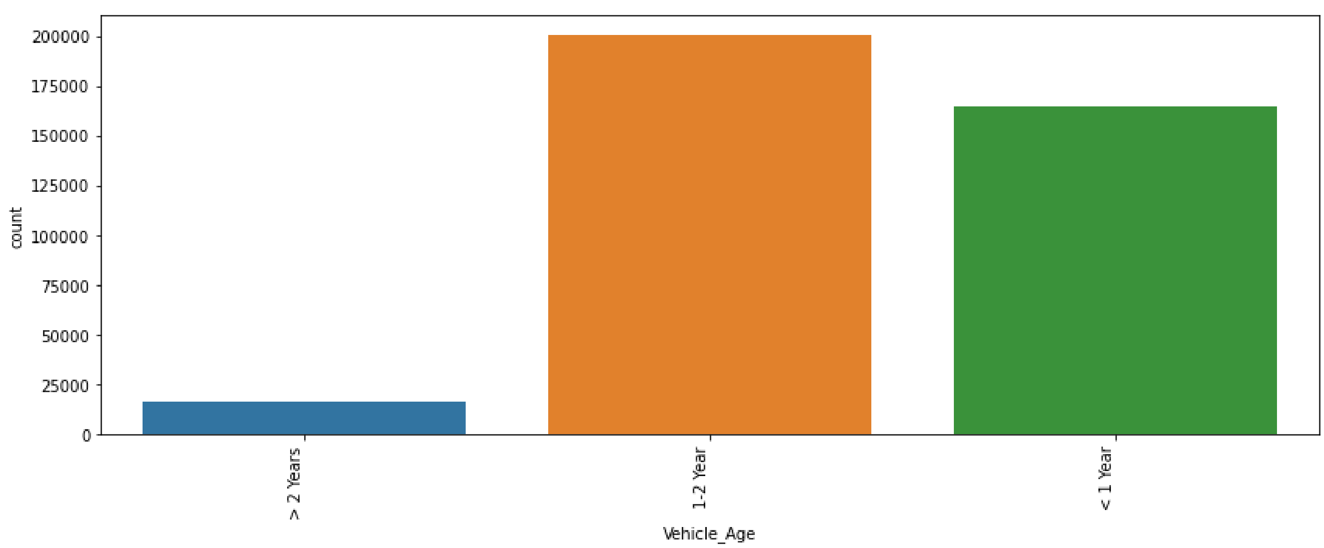
	id	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage
count	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000
mean	190555.000000	38.822584	0.997869	26.388807	0.458210	30564.389581	112.034295	154.347397
std	110016.836208	15.511611	0.046110	13.229888	0.498251	17213.155057	54.203995	83.671304
min	1.000000	20.000000	0.000000	0.000000	0.000000	2630.000000	1.000000	10.000000
25%	95278.000000	25.000000	1.000000	15.000000	0.000000	24405.000000	29.000000	82.000000
50%	190555.000000	36.000000	1.000000	28.000000	0.000000	31669.000000	133.000000	154.000000
75%	285832.000000	49.000000	1.000000	35.000000	1.000000	39400.000000	152.000000	227.000000
max	381109.000000	85.000000	1.000000	52.000000	1.000000	540165.000000	163.000000	299.000000

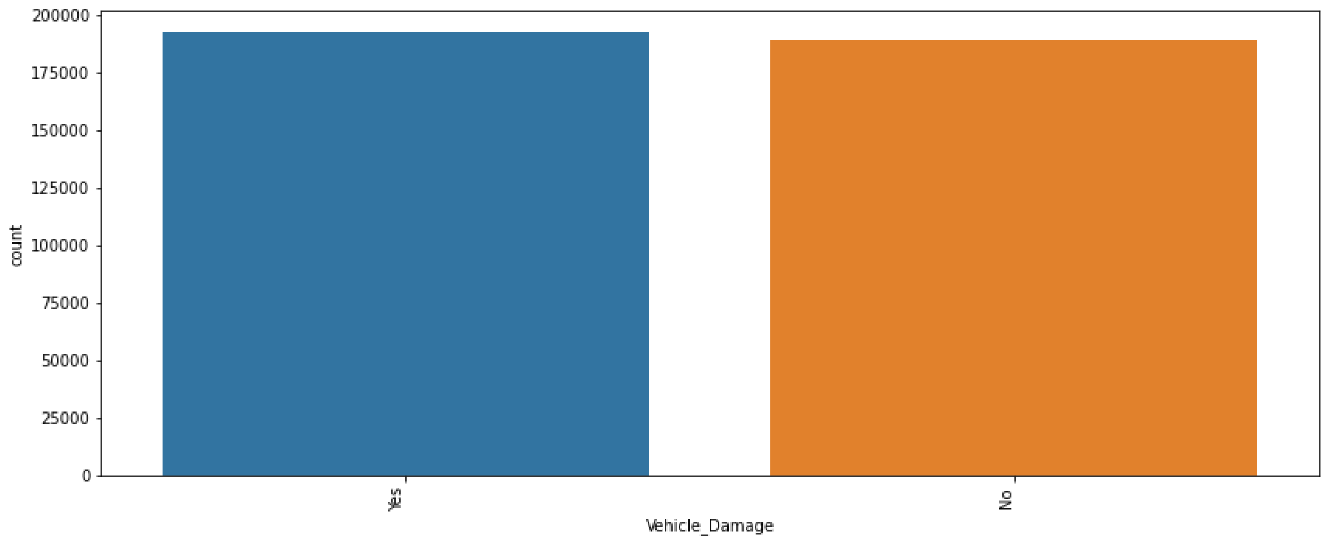
While the train set contain an extra column called Response, to show customers response.

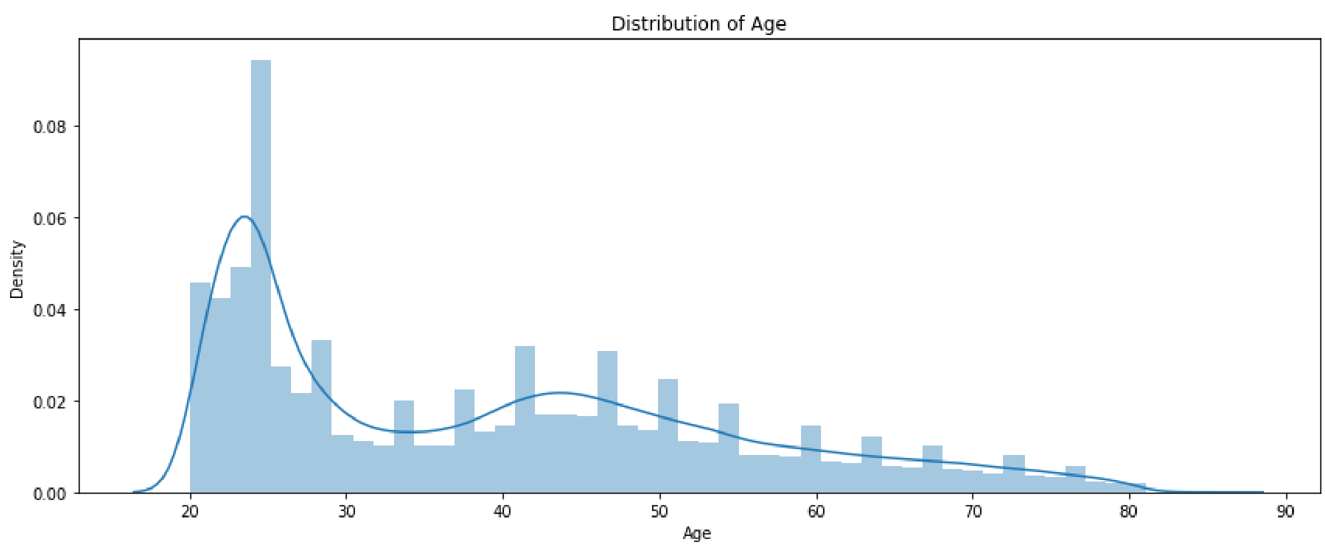
Response
381109.000000
0.122563
0.327936
0.000000
0.000000
0.000000
0.000000
1.000000

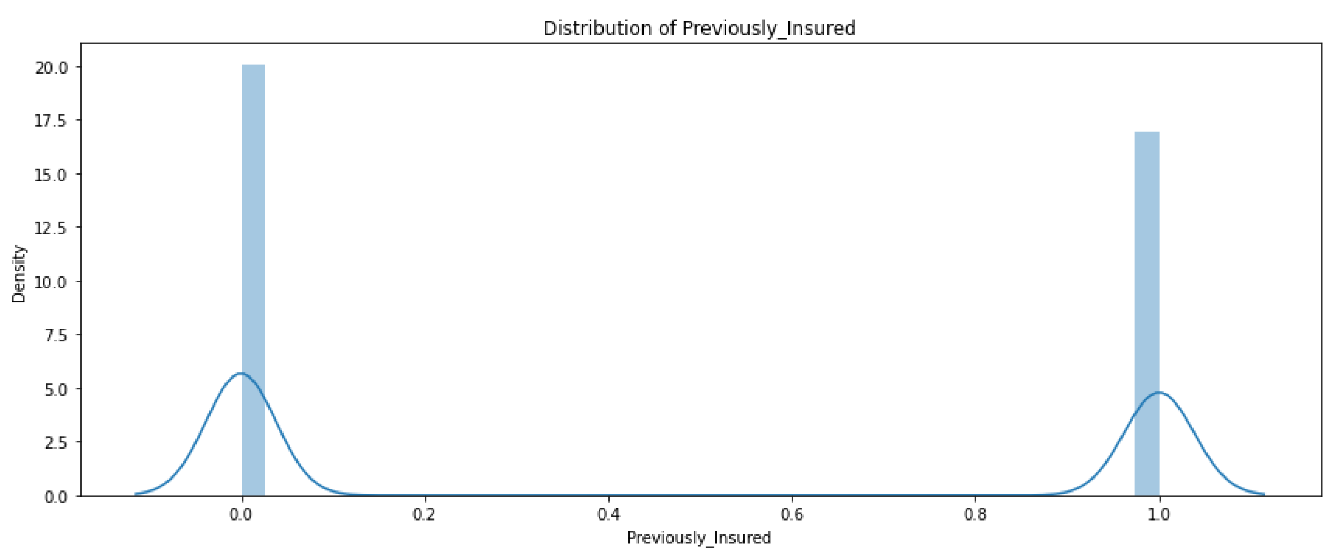
EDA (Exploratory Data Analysis)

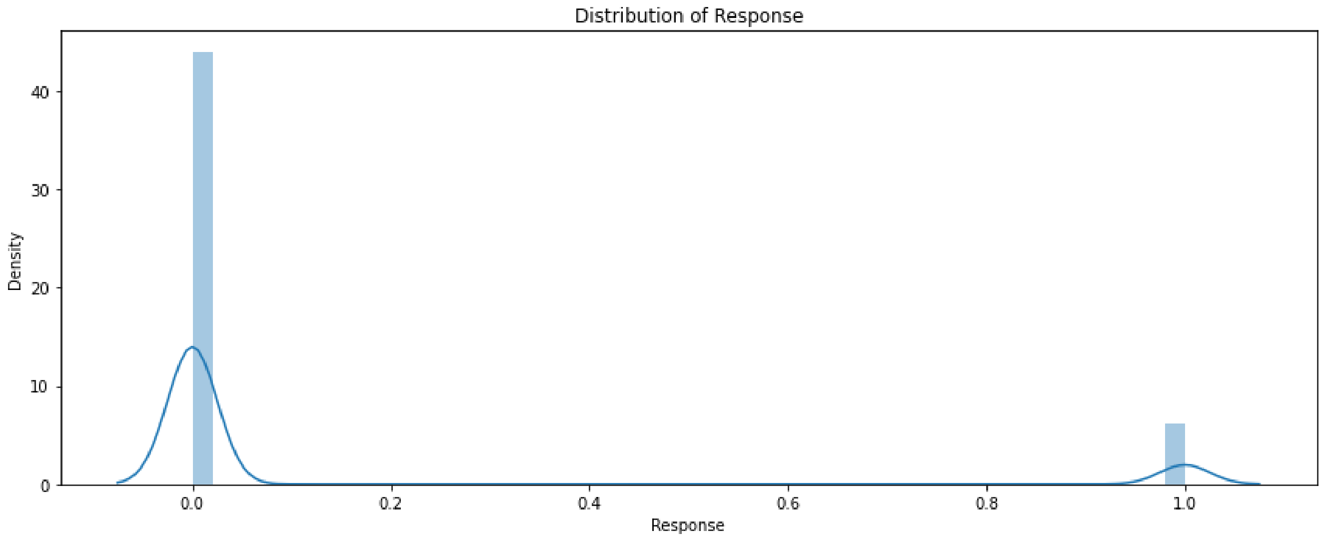
The following plots shows some data analysis derived from the dataset

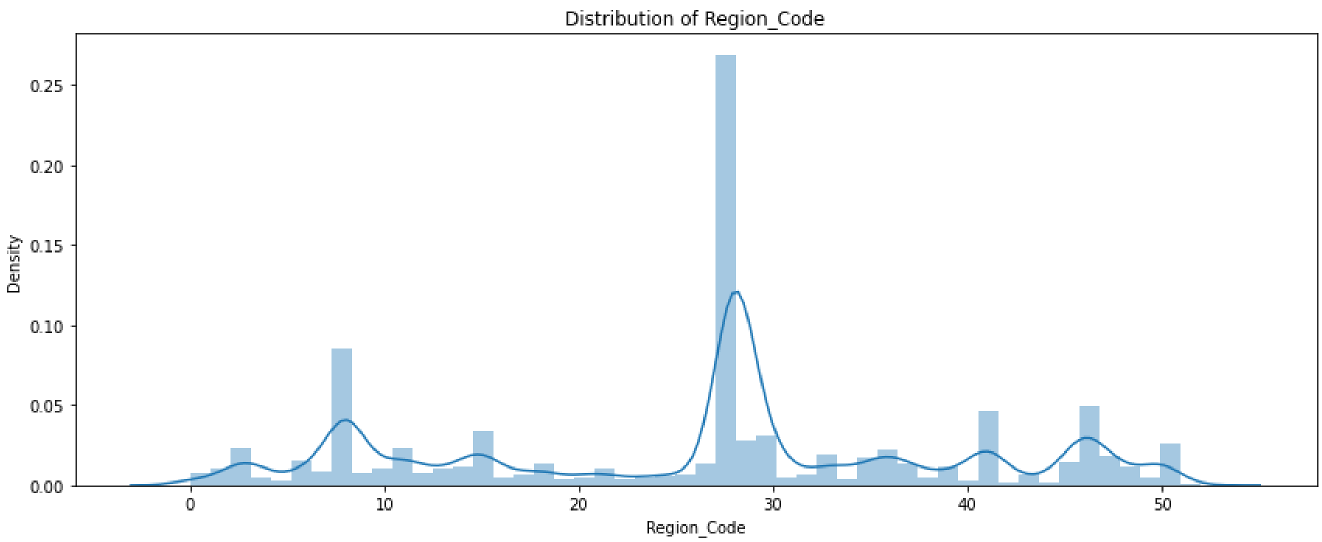



count
Vehicle_Damage
Yes
No


Distribution of Age
Density
Age

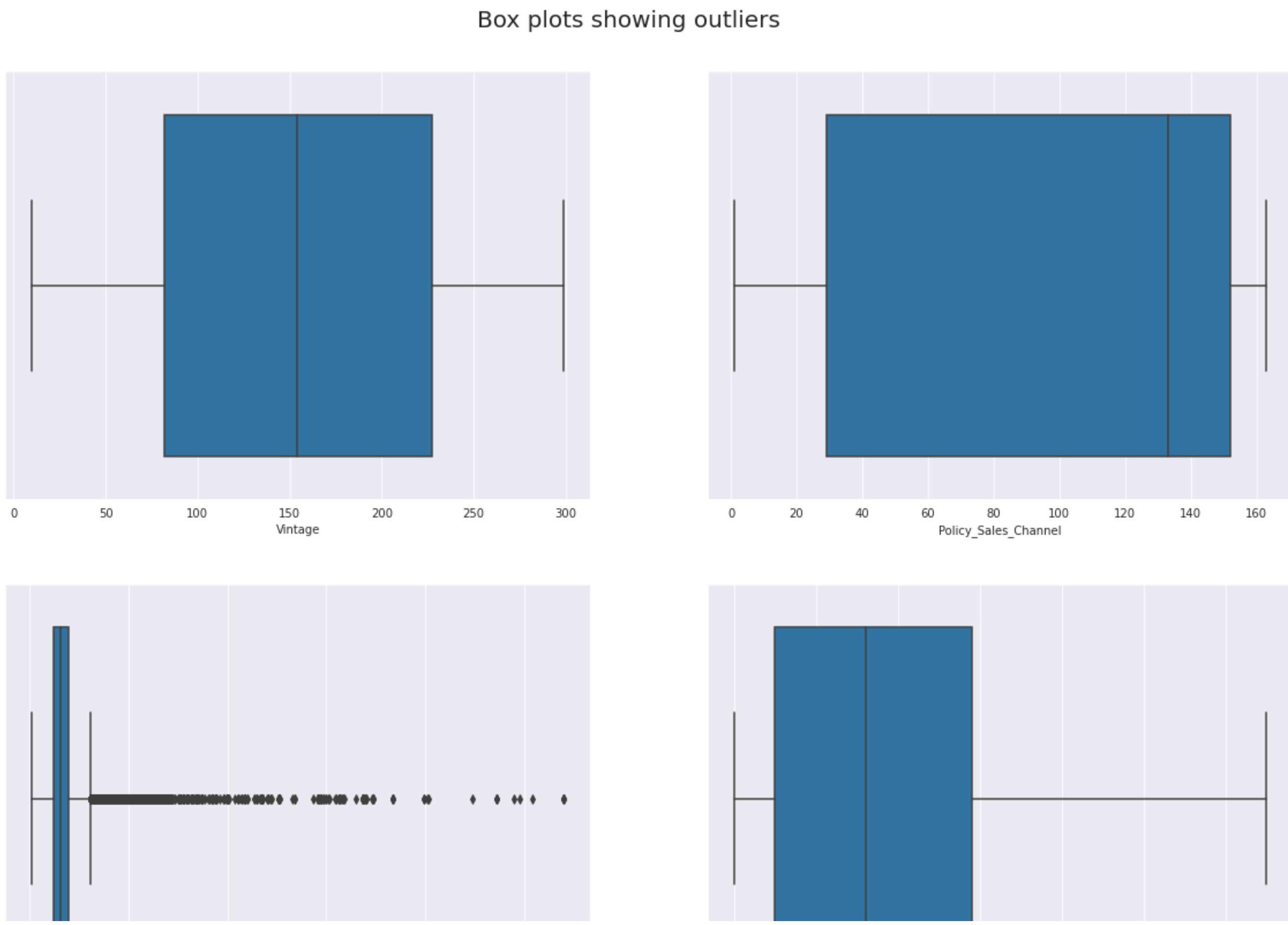

Distribution of Previously_Insured
Density
Previously_Insured

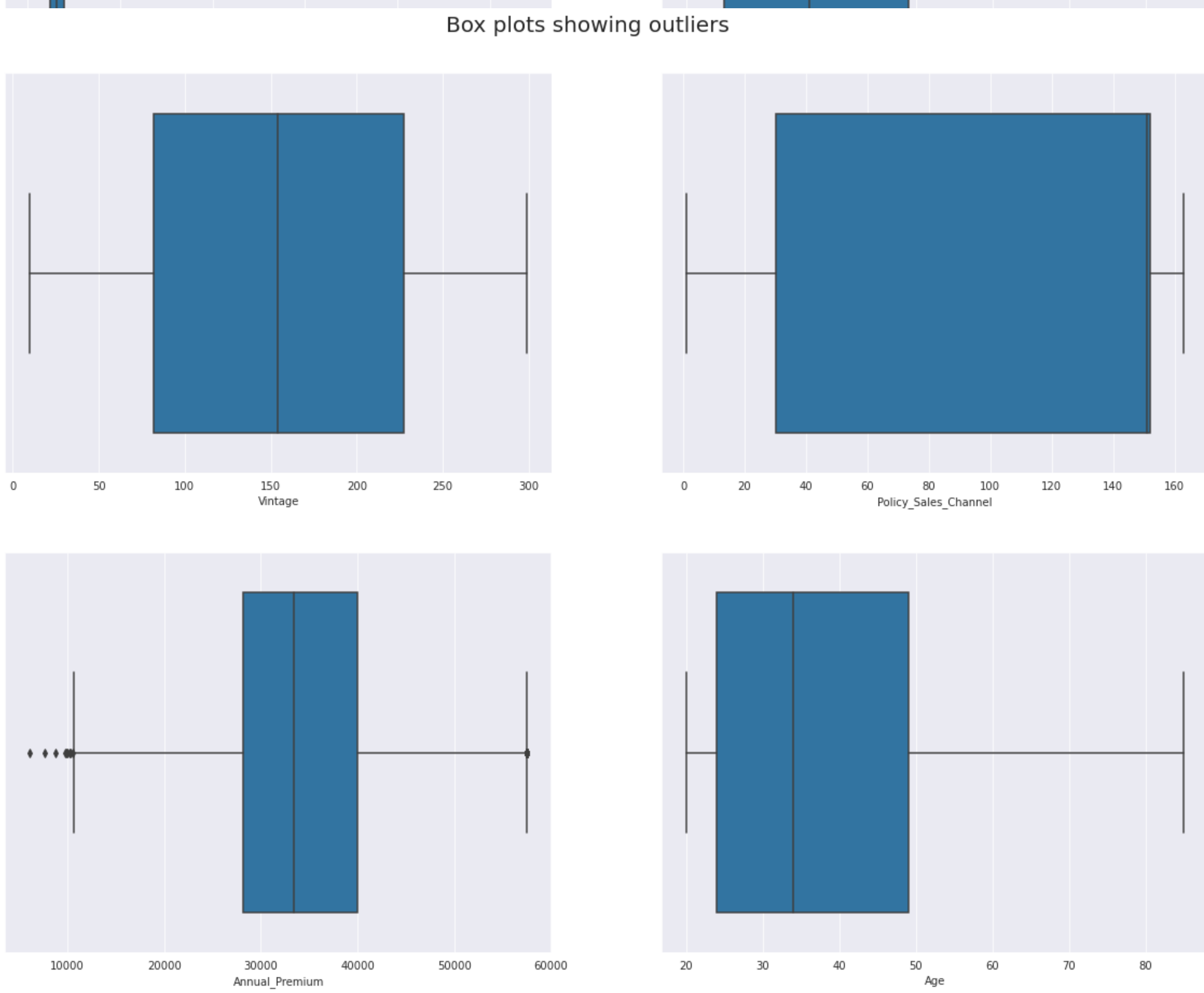




Outliers

In the dataset, only the Annual_Premium column have outliers, while others are balanced and good. The plot below shows how bad it is.
To remove this outlier, a range of selection was chosen from the quantile ratio of the column datapoint, and this range was chosen based on the column description, the code and output is below.


Box plots showing outliers
Vintage
Policy_Sales_Channel


Box plots showing outliers
Vintage
Policy_Sales_Channel
Annual_Premium
Age

In [24]:
```
# Checking it's description again.
health_train['Annual_Premium'].describe()
```
Out[24]:
```
count    381109.000000
mean      30564.389581
std       17213.155057
min        2630.000000
25%       24405.000000
50%       31669.000000
75%       39400.000000
max      540165.000000
Name: Annual_Premium, dtype: float64
```

Again is skewed and has lots of ouliers, needs to be fixed.

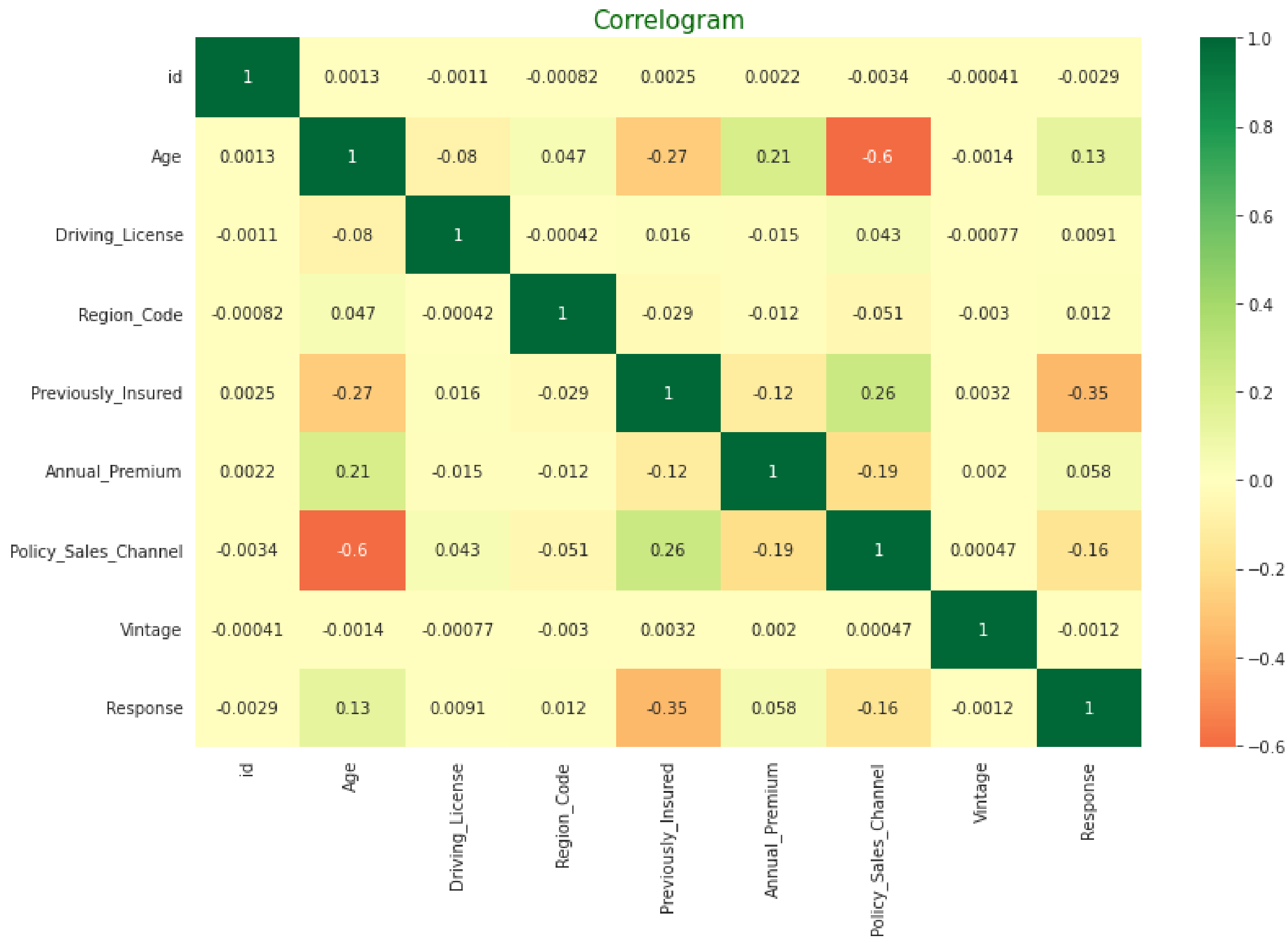
Removing the outliers in Annual_Premium by using the quantile ratio.

In [25]:
```
# Selecting the quantile in the range of 4% to 96%.

min_threshold_1, max_threshold_1 = health_train['Annual_Premium'].quantile([0.04, 0.96])
min_threshold_1, max_threshold_1
```
Out[25]:
```
(2630.0, 57564.67999999999)
```

And after removing the outlier, this is the result. The column is now good to work with, since there are less outliers.

Correlation

This is the correlation plot for the columns in the dataset.
And here, it is seen that the there is only a little correlation between the dataset's columns.



Data Pre-prpocessing

The following images shows how the preprocessing took place. And this happened by the removal of some columns that are not useful to the dataset. And also the creation of a validation set, to help validate the model. Also the use of pipelines was included in the preprocessing.

Data Preprocessing

Here we can see that, Id, policy_sales_channel and Previously Insured aren't correlating with the response .

So we can remove them.

In [33]:
```
# Removing them.

health_train_1 = health_train_1.drop(['Previously_Insured', 'Policy_Sales_Channel'], axis=1)
```

In [34]:
```
# Removing them from the test set also

health_test = health_test.drop(['Previously_Insured', 'Policy_Sales_Channel'], axis=1)
```

In [35]:
```
health_label = health_train_1['Response']
health_train_2 = health_train_1.drop('Response', axis=1)
```

Everything seems good. We can proceed to the next phase.

In [36]:
```
# Creating a validation dataset

from sklearn.model_selection import train_test_split
health_train_3, health_validation  = train_test_split(health_train_2, test_size=0.2, random_state=10)
health_train_label, health_validation_label = train_test_split(health_label, test_size=0.2, random_state=10)
```

Checking the shapes of the train and validation set.

In [37]:
```
health_train_3.shape
```

Out[37]:
```
(240789, 9)
```

In [38]:
```
health_train_label.shape
```

Out[38]:
```
(240789,)
```

In [39]:
```
health_validation.shape
```

Out[39]:
```
(60198, 9)
```

```
In [62]: # Creating a pipeline.
         # One for numerical Operations and the other for categorical operations.
         # StandardScaler is used to make the datapoints have a general range. This makes model algorithm works better.
         # OneHotEncoding has to do with a general way of encoding the categories in the best way for the model.
         # And that is to make each category a column in the dataset, the give each an encoding of
         # 0 and 1. 0 when it isn't of that category and 1 if it is.


         num_pipeline = Pipeline([
             ("Standard scaler", StandardScaler())
         ])

         cat_pipeline = Pipeline([
             ("cat_encoder", OneHotEncoder(sparse=False)),
         ])

In [44]: # Combining the two pipelines so they can work on a dataset generally.

         full_pipeline = ColumnTransformer([
             ("num", num_pipeline, train_data_num),
             ("objects", OneHotEncoder(), train_data_object),
         ])

In [45]: # Fitting the train set to the pipeline.

         health_train_prepared = full_pipeline.fit_transform(health_train_3)

In [46]: # Transforming the validation set also.

         health_validation_prepared = full_pipeline.transform(health_validation)

In [47]: # Then transforming the test set.

         health_test_prepared = full_pipeline.transform(health_test)
```

Modelling

Modeling

The following classification models were adopted in training the train set :
LinearSVC
RandomForesrClassifier
XgboostClassifier
LGBMClassifier
GradientBoostingClassifier

Two parameters were used to measure the performance of each model. And they are : “Accuracy” and “ROC AUC Score”.
Accuracy was found to have a maximum of 88% for most of the models.
ROC AUC Score was found to be a bit over 50 %, and this is because the dataset is not balanced. In the response we have more zeros than ones.

LinearSVC

In [48]:
```python
from sklearn.svm import LinearSVC

lin_clf = LinearSVC()
lin_clf.fit(health_train_prepared, health_train_label)
```

Out[48]: LinearSVC()

In [49]:
```python
health_pred = lin_clf.predict(health_validation_prepared)
```

In [50]:
```python
print(accuracy_score(health_validation_label, health_pred))
```

```
0.8809428884680555
```

RandomForestClassifier

In [51]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score

forest_clf = RandomForestClassifier(random_state = 10)
forest_clf.fit(health_train_prepared, health_train_label)
health_pred = forest_clf.predict(health_validation_prepared)
print(accuracy_score(health_validation_label, health_pred))
print(roc_auc_score(health_validation_label, health_pred))
```

```
0.8749626233429683
0.5226708026807165
```

SGDClassifier

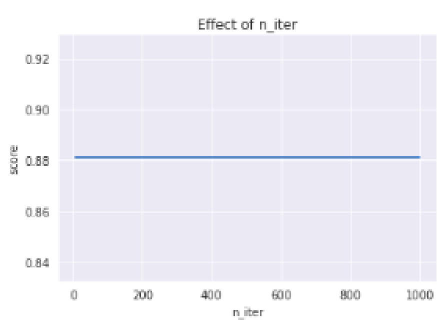
In [52]:
```python
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(health_train_prepared, health_train_label)
health_pred = sgd_clf.predict(health_validation_prepared)
print(accuracy_score(health_validation_label, health_pred))
print(roc_auc_score(health_validation_label, health_pred))
```

In [52]:
```python
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(health_train_prepared, health_train_label)
health_pred = sgd_clf.predict(health_validation_prepared)
print(accuracy_score(health_validation_label, health_pred))
print(roc_auc_score(health_validation_label, health_pred))
```


```
0.8809428884680555
0.5
```

In [64]:
```python
n_iters = [5, 10, 20, 50, 100, 1000]
scores = []
for n_iter in n_iters:
    model = SGDClassifier(loss="hinge", penalty="l2", max_iter=n_iter)
    model.fit(health_train_prepared, health_train_label)
    scores.append(model.score(health_validation_prepared, health_validation_label))

plt.title("Effect of n_iter")
plt.xlabel("n_iter")
plt.ylabel("score")
plt.plot(n_iters, scores)
```

Out[64]: [<matplotlib.lines.Line2D at 0x7f912c588e80>]



DecisionTreeClassifier

In [53]:
```
from sklearn.tree import DecisionTreeClassifier as dtc

tree_clf = dtc(max_depth=10)
tree_clf.fit(health_train_prepared, health_train_label)
health_pred = tree_clf.predict(health_validation_prepared)
print(accuracy_score(health_validation_label, health_pred))
print(roc_auc_score(health_validation_label, health_pred))
```

```
0.8793149274062261
0.5031788431099942
```

XGBoostClassifier

In [54]:
```
import xgboost

from xgboost.sklearn import XGBClassifier
xgboost_clf = XGBClassifier(random_state=42, eval_metric='mlogloss')
xgboost_clf.fit(health_train_prepared, health_train_label)
health_pred = xgboost_clf.predict(health_validation_prepared)
print(accuracy_score(health_validation_label, health_pred))
print(roc_auc_score(health_validation_label, health_pred))
```

```
0.8806272633642314
0.5047080552069582
```

LGBMClassifier

In [55]:
```
from lightgbm import LGBMClassifier

lgb_clf = LGBMClassifier()
lgb_clf.fit(health_train_prepared, health_train_label)
health_pred = lgb_clf.predict(health_validation_prepared)
print(accuracy_score(health_validation_label, health_pred))
print(roc_auc_score(health_validation_label, health_pred))
```

```
0.8810259477059038
0.5010125142295954
```

LGBMClassifier

In [55]:
```python
from lightgbm import LGBMClassifier

lgb_clf = LGBMClassifier()
lgb_clf.fit(health_train_prepared, health_train_label)
health_pred = lgb_clf.predict(health_validation_prepared)
print(accuracy_score(health_validation_label, health_pred))
print(roc_auc_score(health_validation_label, health_pred))
```

```
0.8810259477059038
0.5010125142295954
```

GradientBoostingClassifier

In [56]:
```python
from sklearn.ensemble import GradientBoostingClassifier

gbc_clf = GradientBoostingClassifier()
gbc_clf.fit(health_train_prepared, health_train_label)
health_pred = gbc_clf.predict(health_validation_prepared)
print(accuracy_score(health_validation_label, health_pred))
print(roc_auc_score(health_validation_label, health_pred))
```

```
0.8810591714010432
0.5007900281266099
```

So with the result we have now, we can choose any of the classifier model as our main Model.

And looking at it XGBClasifier has the highest accuracy.

In [57]:
```python
test_pred = xgboost_clf.predict(health_test_prepared)
```

In [58]:
```python
test_pred[:100]
```

Out[58]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

It is observed that the model will not perform any better than it already did.

Conclusion

XgboostClassifier model was used as the final model. Then it was used to predict the test set, and the prediction was stored in a dataframe which can be converted to a csv file for submission and grading.

And looking at it XGBClasifier has the highest accuracy.

In [57]:
```
test_pred = xgboost_clf.predict(health_test_prepared)
```

In [58]:
```
test_pred[:100]
```

Out[58]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [59]:
```
# Now creating a dataframe that have the response prediction and the id. Our Submission.

test_response = pd.DataFrame()
test_response['id'] = health_test.id
test_response['Predicted Response'] = test_pred
```

In [60]:
```
test_response
```

Out[60]:

	id	Predicted Response
0	381110	0
1	381111	0
2	381112	0
3	381113	0
4	381114	0
...	...	...
127032	508142	0
127033	508143	0
127034	508144	0
127035	508145	0
127036	508146	0

127037 rows × 2 columns

In [60]:
```
test_response
```

Out[60]:

	id	Predicted Response
0	381110	0
1	381111	0
2	381112	0
3	381113	0
4	381114	0
...	...	...
127032	508142	0
127033	508143	0
127034	508144	0
127035	508145	0
127036	508146	0

127037 rows × 2 columns

This submission dataset can be exported, then submitted.

In [61]:
```
# The following code will do that

# test_response.to_csv(r'Path where you want to store the exported CSV file\File Name.csv', index = False)
```

Results

We clearly see that, there aren't strong correlations between the columns of the dataset. If the information given can be correlating to the response of the customer we will have a better result. And that if the outlier is not removed, it will make the model perform badly.

Limitations

The limitation the model used has is the fact that the dataset is not balanced. This brought an hindrance to the performance of the models used. If the dataset can be balanced then bot the ROC AUC Score and Accuracy will be better.

References

1. About scikit-learn models -- https://scikit-learn.org/stable/supervised_learning.html

2. Outliers Removal -- https://machinelearningmastery.com/how-to-use-statistics-to-identify-outliers-in-data/

3. Roc Auc score -- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html