In [ ]:
```
from google.colab import files
files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving ss.csv to ss.csv
Saving test.csv to test.csv
Saving train.csv to train.csv
```

Imports

In [5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-whitegrid')

import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import accuracy_score, f1_score

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression

from lightgbm import LGBMClassifier

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

pd.set_option('display.max_colwidth', -1)

import warnings
warnings.simplefilter('ignore')

import plotly.offline as pyo
import plotly.figure_factory as ff
from plotly import tools

import plotly.graph_objs as go
```

In [ ]:
```
#train = pd.read_csv('C:\\Users\\Anuvrat Shukla\\Desktop\\competitions\\Analytics\\Individual\\IMT Hyd\\train.cs
#train.head()
```

In [ ]:
```
train = pd.read_csv('train.csv')
train.head()
```

Out[6]:

	Custmer_Id	Gender	Age	DL	City_Code	Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Sales_Channel	Customer_Associat
0	1	Male	44	1	28	0	> 2 Years	Yes	40454	26	
1	2	Male	76	1	3	0	1-2 Year	No	33536	26	
2	3	Male	47	1	28	0	> 2 Years	Yes	38294	26	
3	4	Male	21	1	11	1	< 1 Year	No	28619	152	
4	5	Female	29	1	41	1	< 1 Year	No	27496	152	

Hypothesis Generation

1. Does insurance respone is gender bisased?
2. Does old/middle age people subscribed more or young people subscribed more? **Middle aged people (30-50)**
3. Having a DL led to get vehicle insurance? 4.Does Vehicle age had any relation on getting vehicle insurance? **1-2 years and >2 years**
4. Does people with vehicle damage, subscribed to vehicle insurance? **Yes**
5. People who are paying high annual preimium also took vehicle insurance or not? **Yes people paying high premium took the Vehicle insurance**
6. What were the top few sales channel where people were most likely to take insurance?
7. Did an old customer also took the insurance? **New Customers as well as Vintage customers subscribed toinsurance**

In [ ]:
```
ID_COL, TARGET_COL = 'id', 'Response'
```

In [ ]:
```
print(f'\nTrain contains {train.shape[0]} samples and {train.shape[1]} variables')
#print(f'\nTest contains {test.shape[0]} samples and {test.shape[1]} variables')

features = [c for c in train.columns if c not in [ID_COL, TARGET_COL]]
print(f'\nThe dataset contains {len(features)} features')
```

```
Train contains 381109 samples and 12 variables

The dataset contains 11 features
```

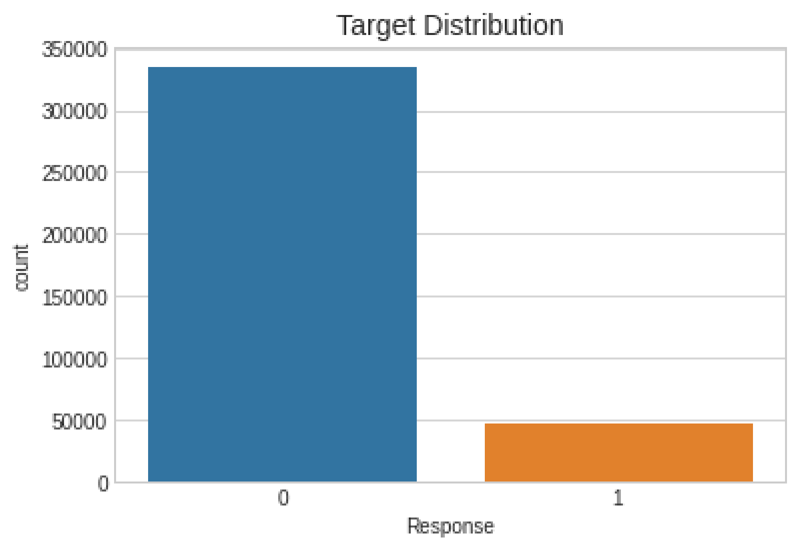
3. Target Distribution

This is a binary classification problem. Lets have a look at the number of positive and negative examples that we have, or in our problem statement terms: '*Number of People who did subscribe for a term deposit and the number of people who did not*'

In [ ]:
```
train[TARGET_COL].value_counts(normalize=True)
```

Out[11]:
```
0    0.877437
1    0.122563
Name: Response, dtype: float64
```

In [ ]:
```
_ = sns.countplot(train[TARGET_COL])
_ = plt.title("Target Distribution", fontsize=14)
```



Variable Types

In [ ]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 381109 entries, 0 to 381108
Data columns (total 12 columns):
 #   Column                Non-Null Count   Dtype 
---  ------                --------------   ----- 
 0   Custmer_Id            381109 non-null  int64 
 1   Gender                381109 non-null  object
 2   Age                   381109 non-null  int64 
 3   DL                    381109 non-null  int64 
 4   City_Code             381109 non-null  int64 
 5   Insured               381109 non-null  int64 
 6   Vehicle_Age           381109 non-null  object
 7   Vehicle_Damage        381109 non-null  object
 8   Annual_Premium        381109 non-null  int64 
 9   Sales_Channel         381109 non-null  int64 
 10  Customer_Association  381109 non-null  int64 
 11  Response              381109 non-null  int64 
dtypes: int64(9), object(3)
memory usage: 34.9+ MB
```

Null Values

In [ ]:
```
null_values_per_variable = 100 * (train.isnull().sum()/train.shape[0]).round(3)#.reset_index()
null_values_per_variable.sort_values(ascending=False)
```

Out[10]:
```
Response                0.0
Customer_Association    0.0
Sales_Channel           0.0
Annual_Premium          0.0
Vehicle_Damage          0.0
Vehicle_Age             0.0
Insured                 0.0
City_Code               0.0
DL                      0.0
Age                     0.0
Gender                  0.0
Custmer_Id              0.0
dtype: float64
```

In [ ]:
```
train.nunique()
```

Out[11]:
```
Custmer_Id              381109
Gender                  2
Age                     66
DL                      2
City_Code               53
Insured                 2
Vehicle_Age             3
Vehicle_Damage          2
Annual_Premium          48838
Sales_Channel           155
Customer_Association    290
Response                2
dtype: int64
```

Features

In [8]:
```
cat_cols = ['Gender',
 'Vehicle_Age',
 'Vehicle_Damage']
```

In [9]:
```
num_cols = [c for c in features if c not in cat_cols]
num_cols
```

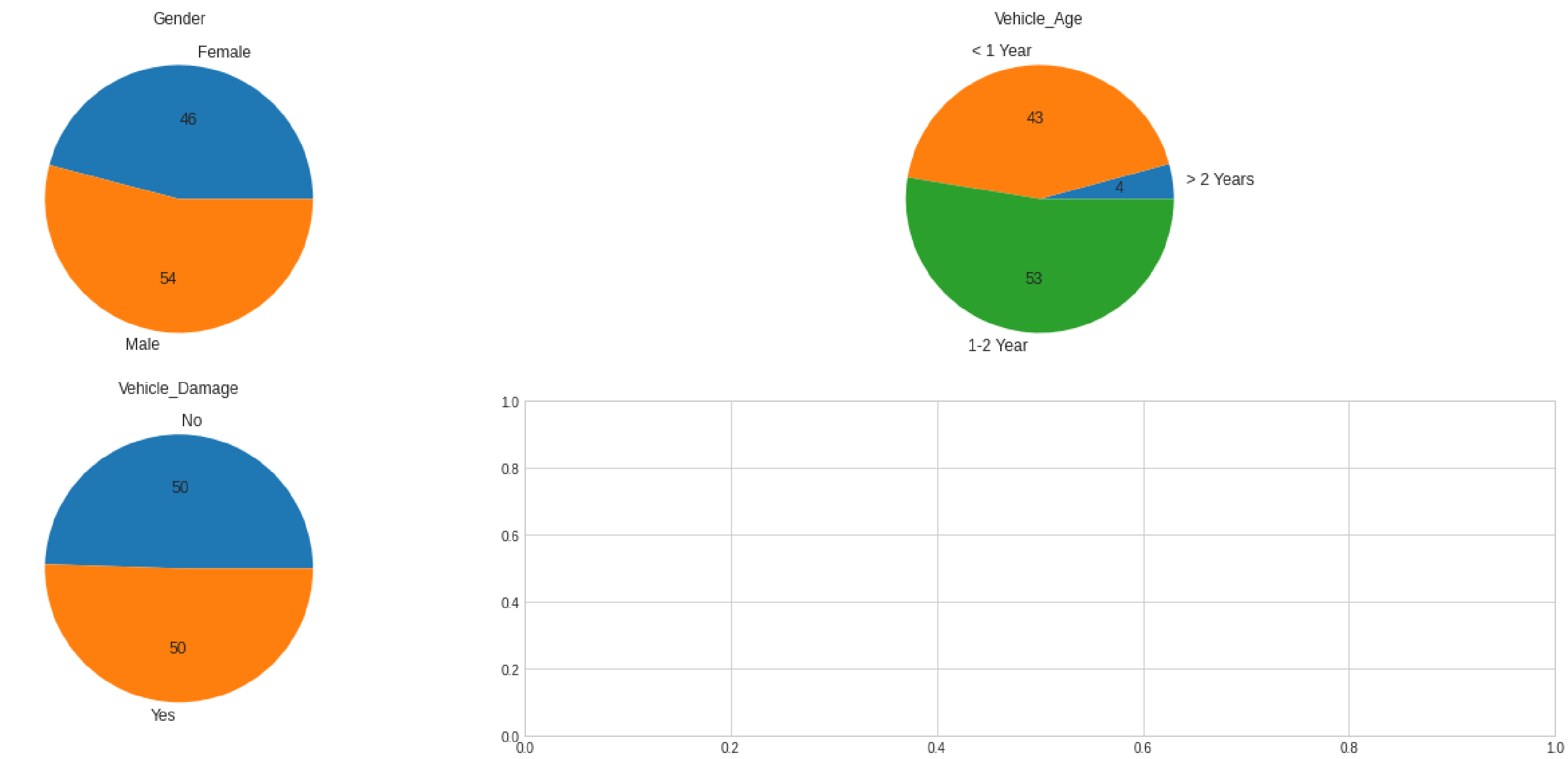
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-d14ec6e6ba05> in <module>()
----> 1 num_cols = [c for c in features if c not in cat_cols]
      2 num_cols

NameError: name 'features' is not defined
```

Categorical Features

Univariate Analysis

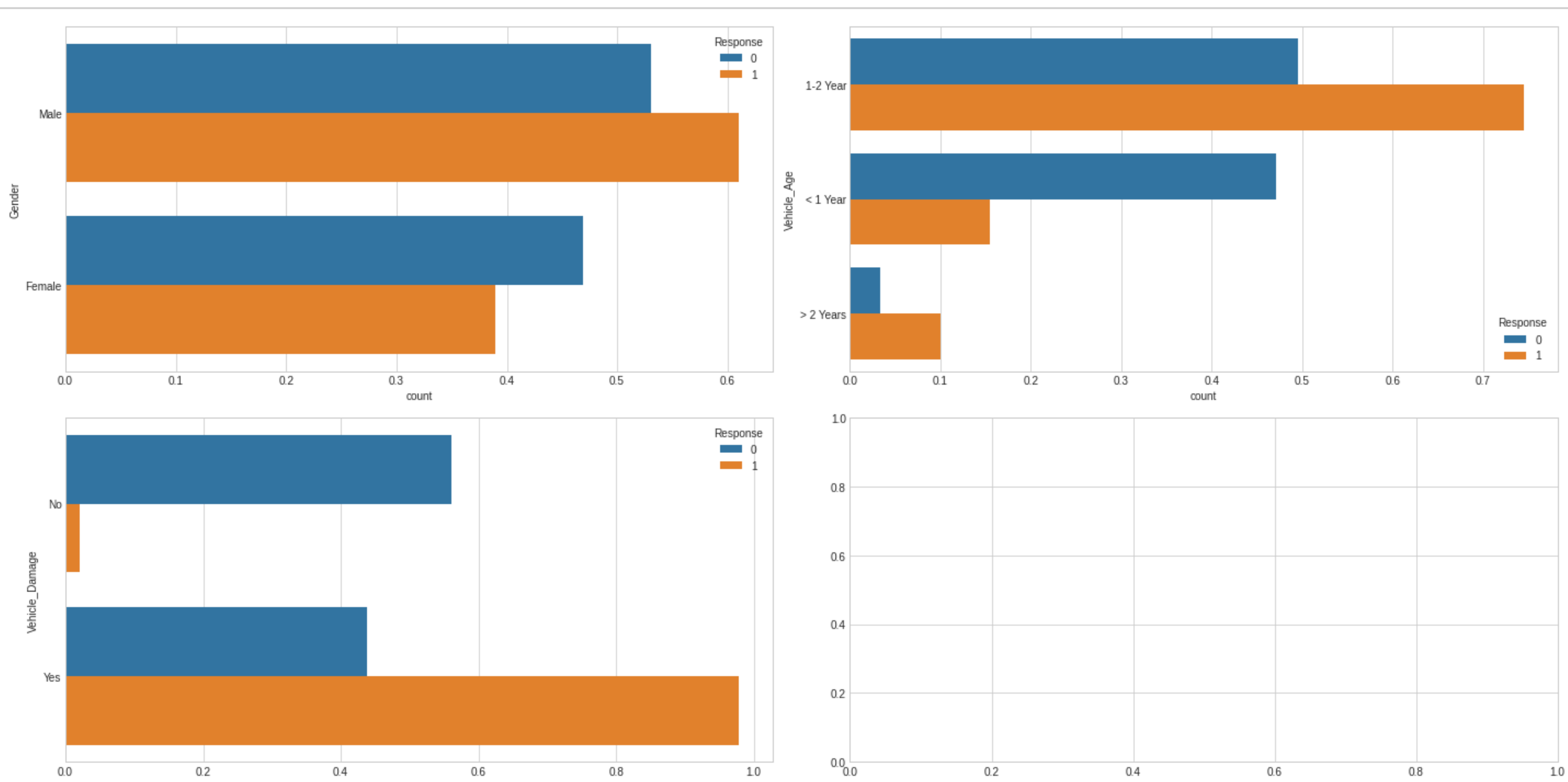
Pie Charts can be useful in seeing the proportion of samples, that fall into each category of a categorical variable. For each of the categorical variables we will make a pie chart.

In [ ]:
```
fig, axes = plt.subplots(2, 2, figsize=(20, 8))
axes = [ax for axes_row in axes for ax in axes_row]

for i, c in enumerate(train[cat_cols]):
    _ = train[c].value_counts()[::-1].plot(kind = 'pie', ax=axes[i], title=c, autopct='%.0f', fontsize=12)
    _ = axes[i].set_ylabel('')

_ = plt.tight_layout()
```



Bivariate Analysis Relationships with Target

In [ ]:
```
fig, axes = plt.subplots(2, 2, figsize=(20, 10))
axes = [ax for axes_row in axes for ax in axes_row]

for i, c in enumerate(train[cat_cols]):
    fltr = train[TARGET_COL] == 0
    vc_a = train[fltr][c].value_counts(normalize=True).reset_index().rename({'index' : c, c: 'count'}, axis=1)

    vc_b = train[~fltr][c].value_counts(normalize=True).reset_index().rename({'index' : c, c: 'count'}, axis=1)

    vc_a[TARGET_COL] = 0
    vc_b[TARGET_COL] = 1

    df = pd.concat([vc_a, vc_b]).reset_index(drop = True)

    _ = sns.barplot(y = c, x = 'count', data =df , hue=TARGET_COL, ax=axes[i])

_ = plt.tight_layout()
```



Observations

1. Among male customers, majority took vehicle insurance and among females, majority did not took insurance
2. Customers with Vehicle age of 1-2 years and more than 2 years tend to take insurances
3. Customers with damaged vehicles took more vehicle insurances and opposite case for customers without vehicle damage.

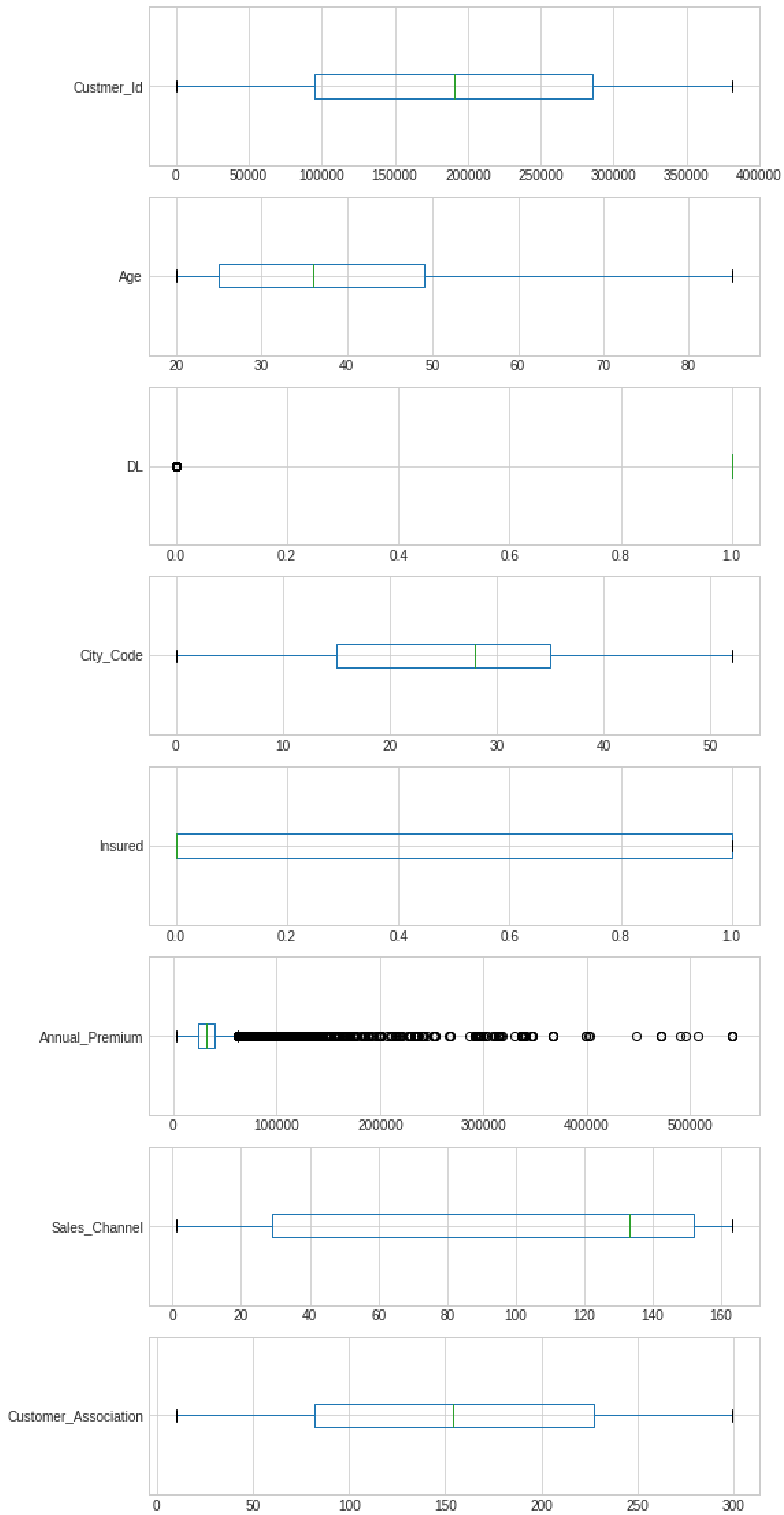
Numerical Features

1. Univariate Analysis - Boxplots

Boxplot can be used to see the spread of the numerical variables, and identify outliers

[https://en.wikipedia.org/wiki/Box_plot (https://en.wikipedia.org/wiki/Box_plot)](https://en.wikipedia.org/wiki/Box_plot)

In [ ]:
```
fig, axes = plt.subplots(8, 1, figsize=(8, 20))
for i, c in enumerate(num_cols):
  _ = train[[c]].boxplot(ax=axes[i], vert=False)
```

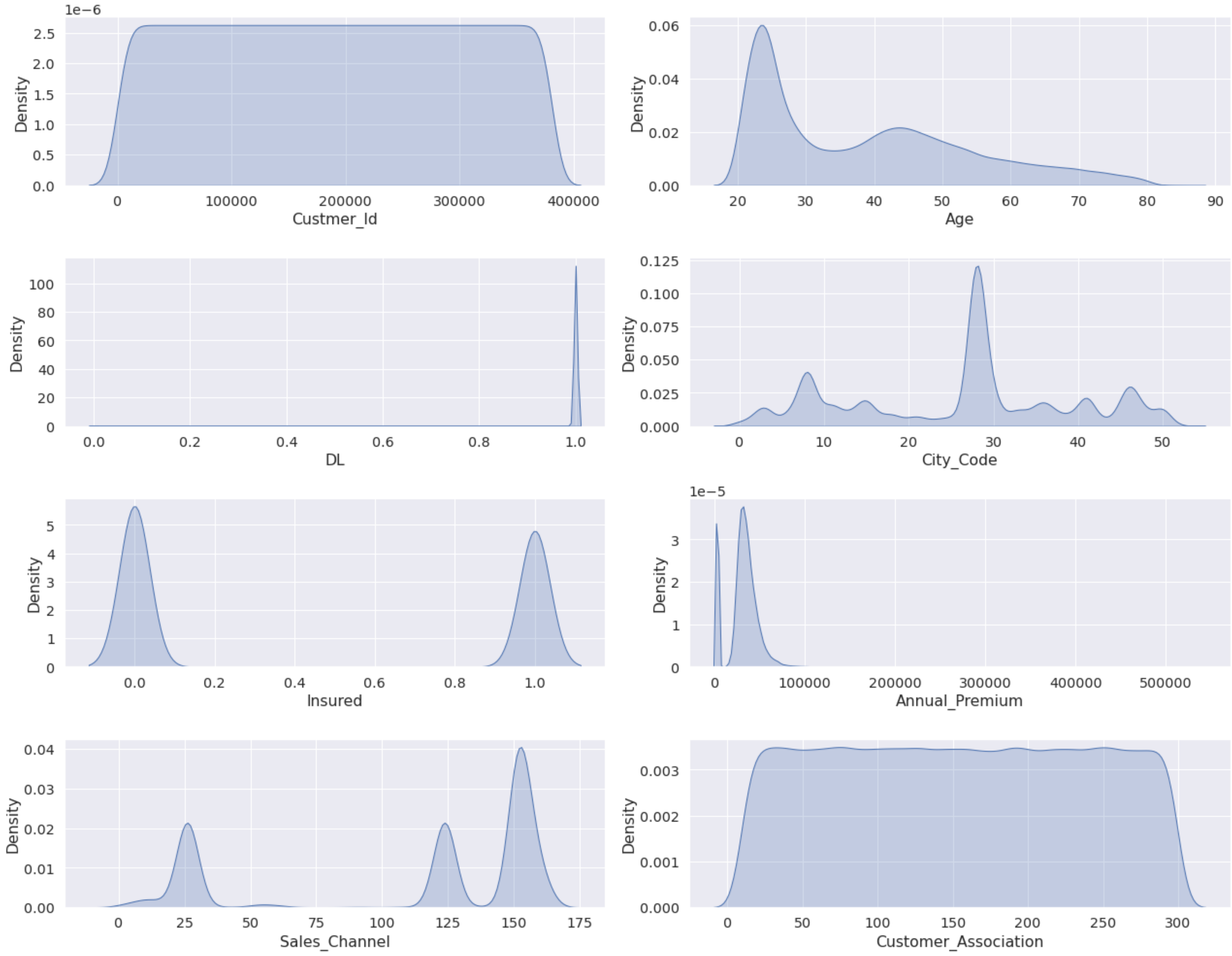


2. Univariate Analysis - Density Plots

A kernel density estimate (KDE) plot is a method for visualizing the distribution of observations in a dataset, analagous to a histogram. KDE represents the data using a continuous probability density curve in one or more dimensions.

[https://seaborn.pydata.org/generated/seaborn.kdeplot.html (https://seaborn.pydata.org/generated/seaborn.kdeplot.html)](https://seaborn.pydata.org/generated/seaborn.kdeplot.html)

In [ ]:
```
sns.set(font_scale=1.3)
fig, axes = plt.subplots(4, 2, figsize=(18, 14))
axes = [ax for axes_row in axes for ax in axes_row]
for i, c in enumerate(num_cols):
  plot = sns.kdeplot(data=train, x=c, ax=axes[i], fill=True)
plt.tight_layout()
```

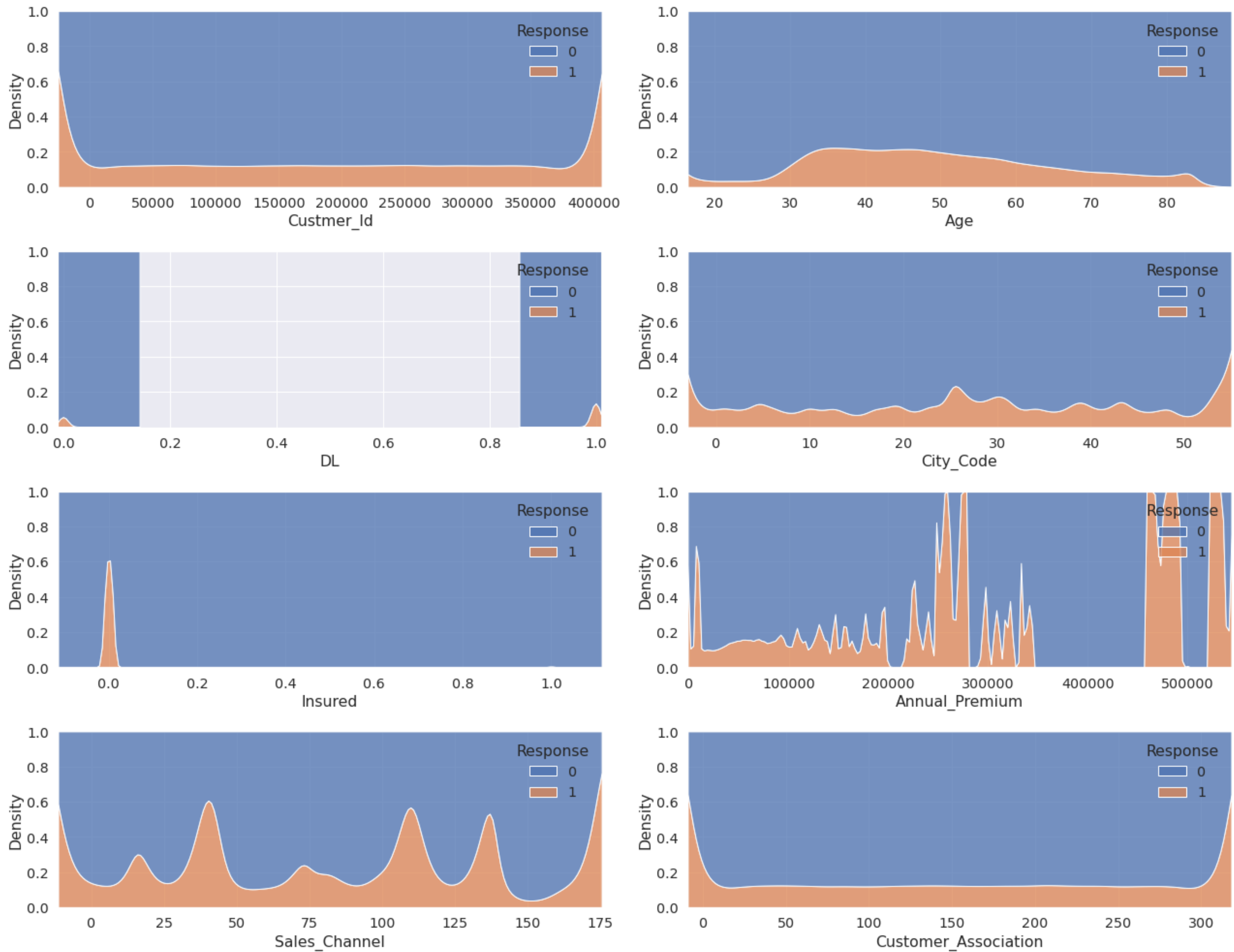


Observations

1. Database mostly had target customers with young age and middle age
2. Database had customers mostly who have DL

Bivariate Analysis KDE plots - Relationships with Target Variable.

In [ ]:
```
sns.set(font_scale=1.3)
fig, axes = plt.subplots(4, 2, figsize=(18, 14))
axes = [ax for axes_row in axes for ax in axes_row]
for i, c in enumerate(num_cols):
  plot = sns.kdeplot(data=train, x=c, hue=TARGET_COL, multiple='fill', ax=axes[i])
plt.tight_layout()
```



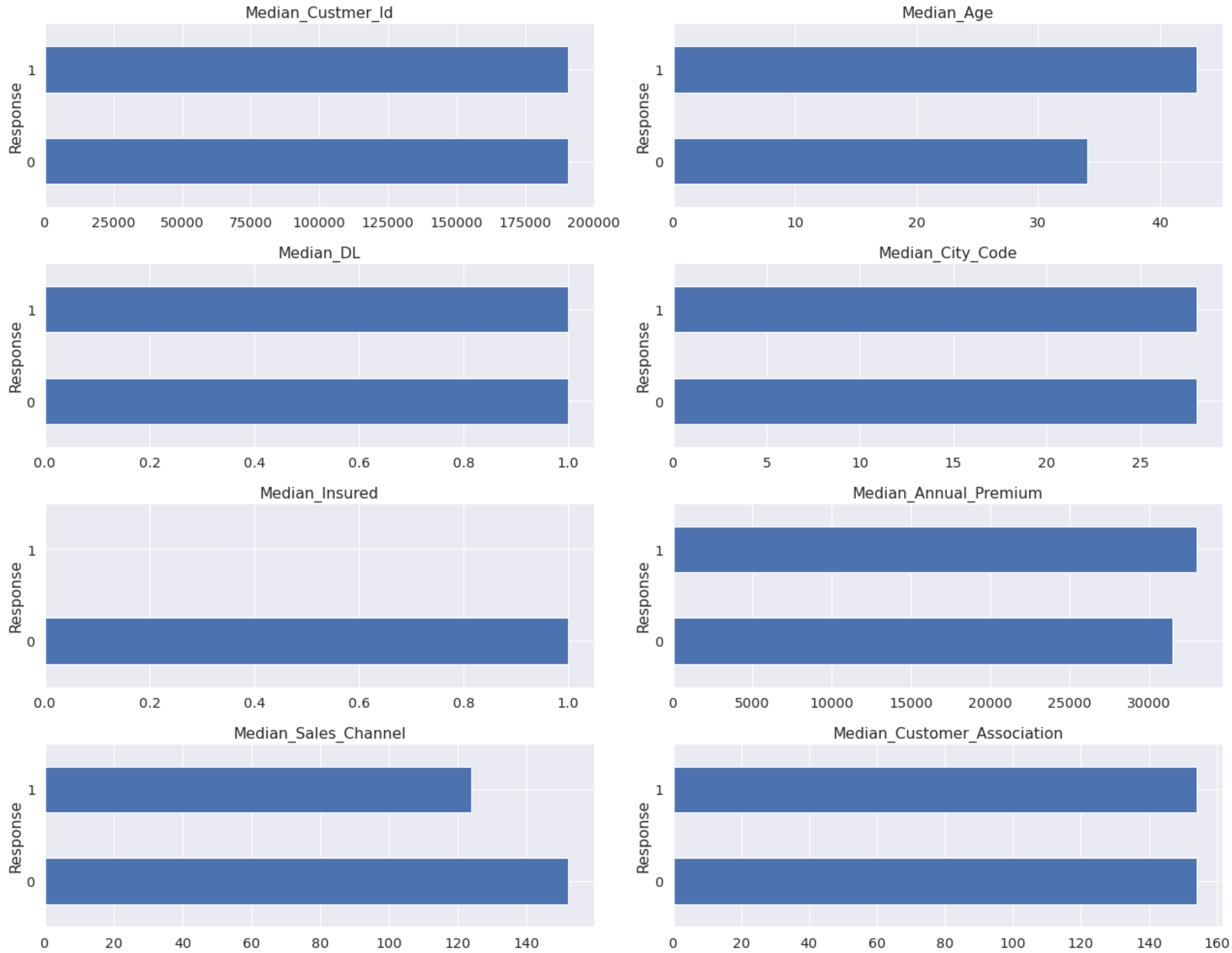
Observations

1. Customers in the range 30-50 responded more for Vehicle Insurance
2. Customers with DL reponded more than customers without DL
3. Customers who were were not insured previously tend to take Vehicle Insurance
4. Mostly Veteran Customers and new customers took the vehicle insurance

Median Effect

We are choosing median since median is not affected by outliers

In [ ]:
```
sns.set(font_scale=1.3)
fig, axes = plt.subplots(4, 2, figsize=(18, 14))
axes = [ax for axes_row in axes for ax in axes_row]
for i, c in enumerate(num_cols):
  plot = train.groupby(TARGET_COL)[c].median().plot(kind = 'barh', title=f'Median_{c}', ax=axes[i])
plt.tight_layout()
```

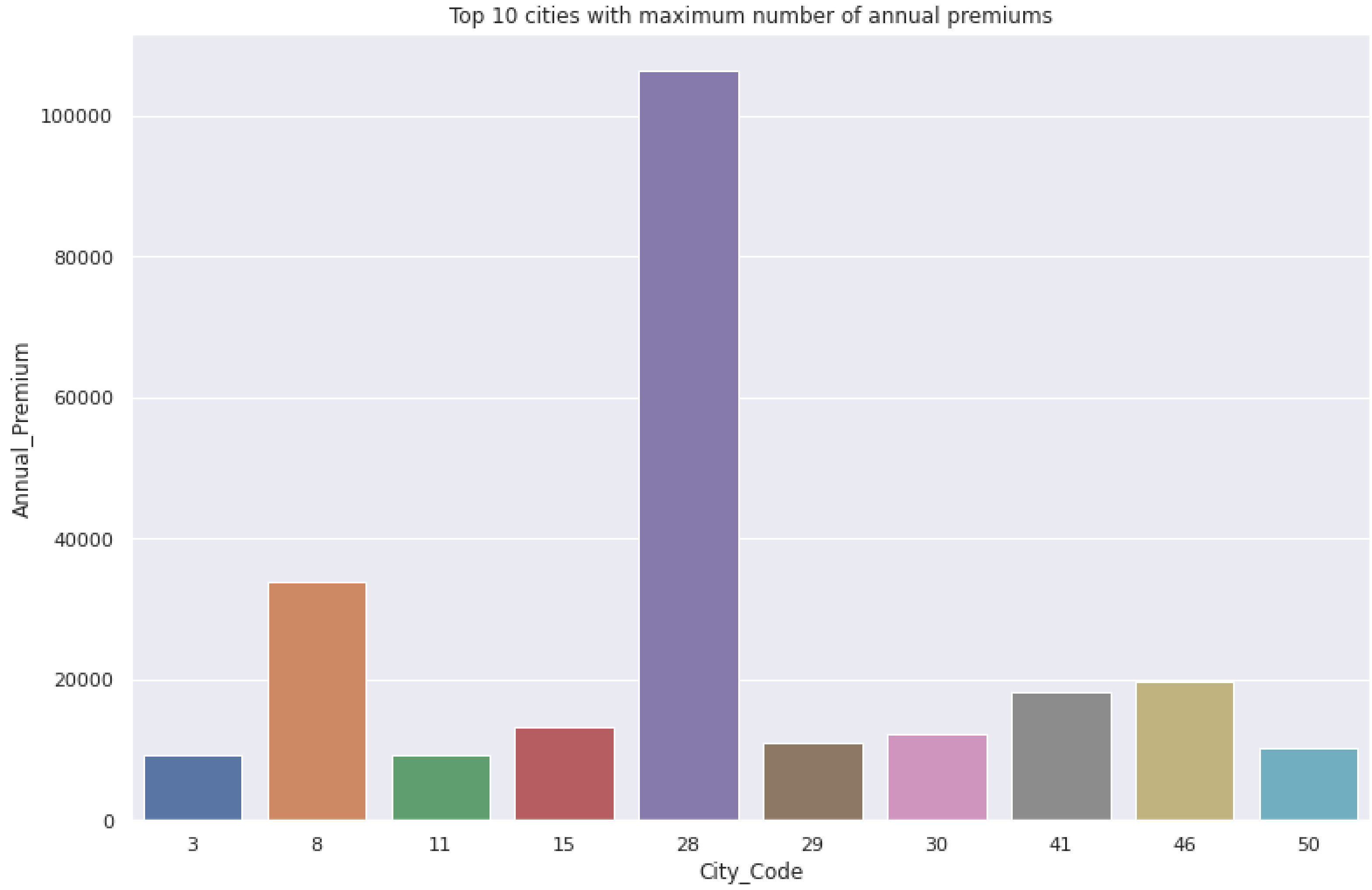


Obsevations

1. Customers of 40+ age tend to take vehicle insurances more
2. Median annual premium paid was around 30,000

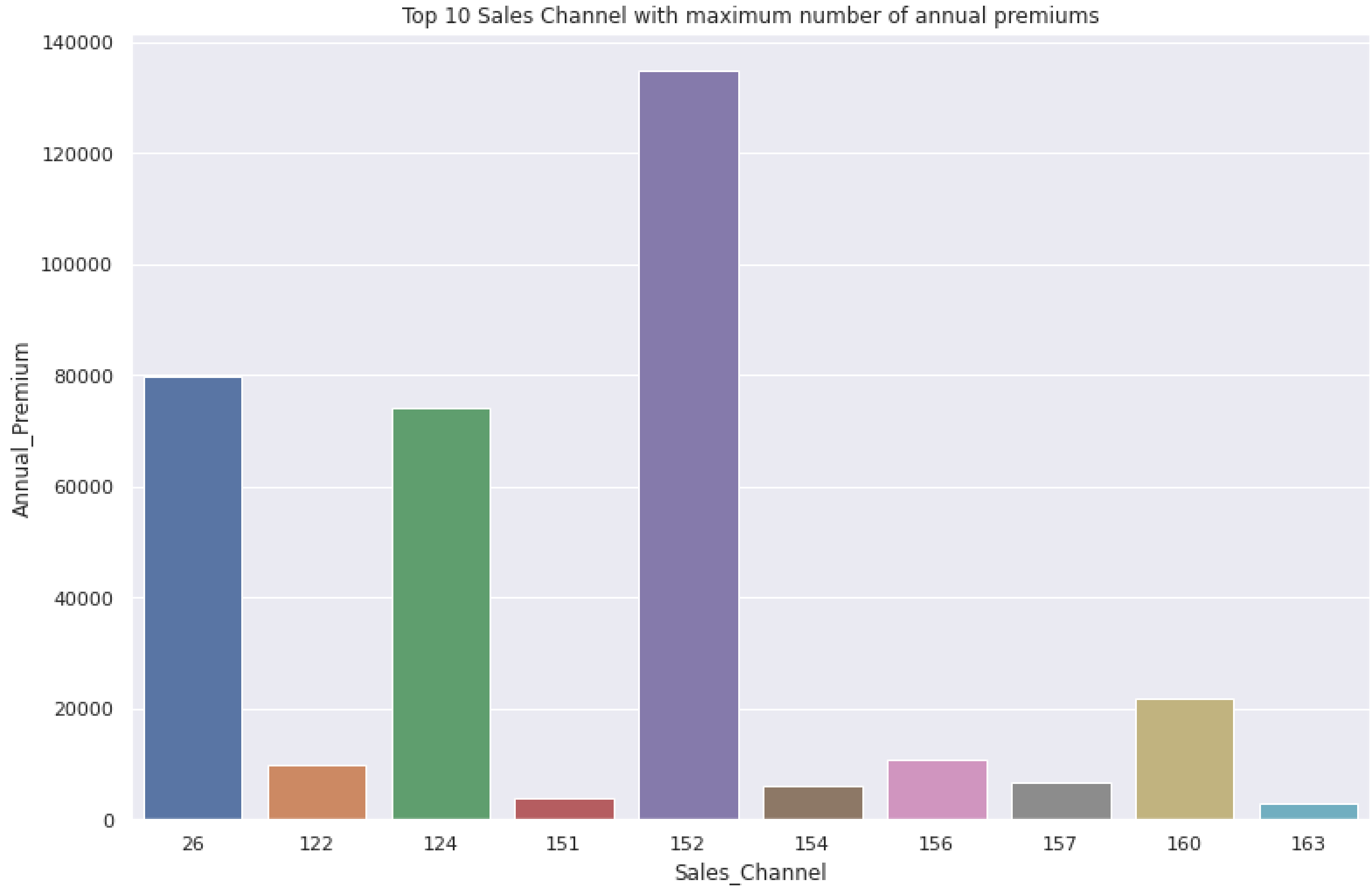
City generating maximum annual premiums

In [ ]:
```
sns.set(rc={'figure.figsize':(12.7, 8.27)})

top_20_channels = train['City_Code'].value_counts()[:10].reset_index()
top_20_channels.columns = ['City_Code', 'Annual_Premium']

_ = sns.barplot(data = top_20_channels, y = 'Annual_Premium', x = 'City_Code')
_ = plt.title("Top 10 cities with maximum number of annual premiums")
```



#Sales Channel generating maximum annual premiums

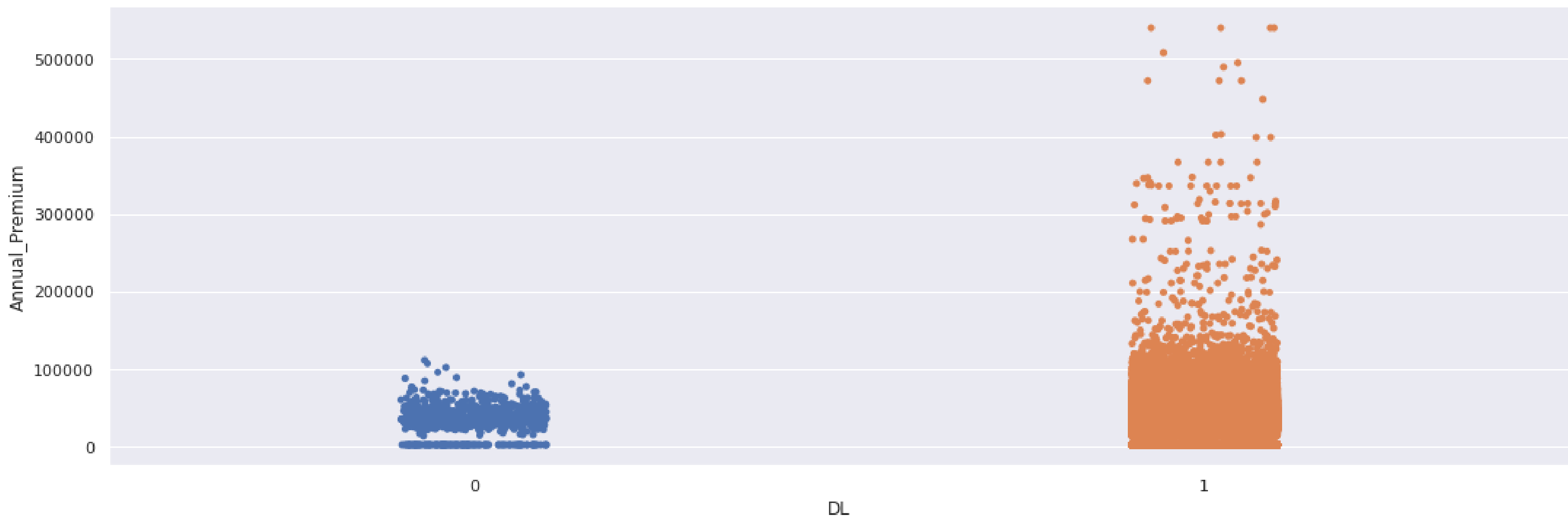
In [ ]:
```
sns.set(rc={'figure.figsize':(12.7, 8.27)})

top_20_channels = train['Sales_Channel'].value_counts()[:10].reset_index()
top_20_channels.columns = ['Sales_Channel', 'Annual_Premium']

_ = sns.barplot(data = top_20_channels, y = 'Annual_Premium', x = 'Sales_Channel')
_ = plt.title("Top 10 Sales Channel with maximum number of annual premiums")
```



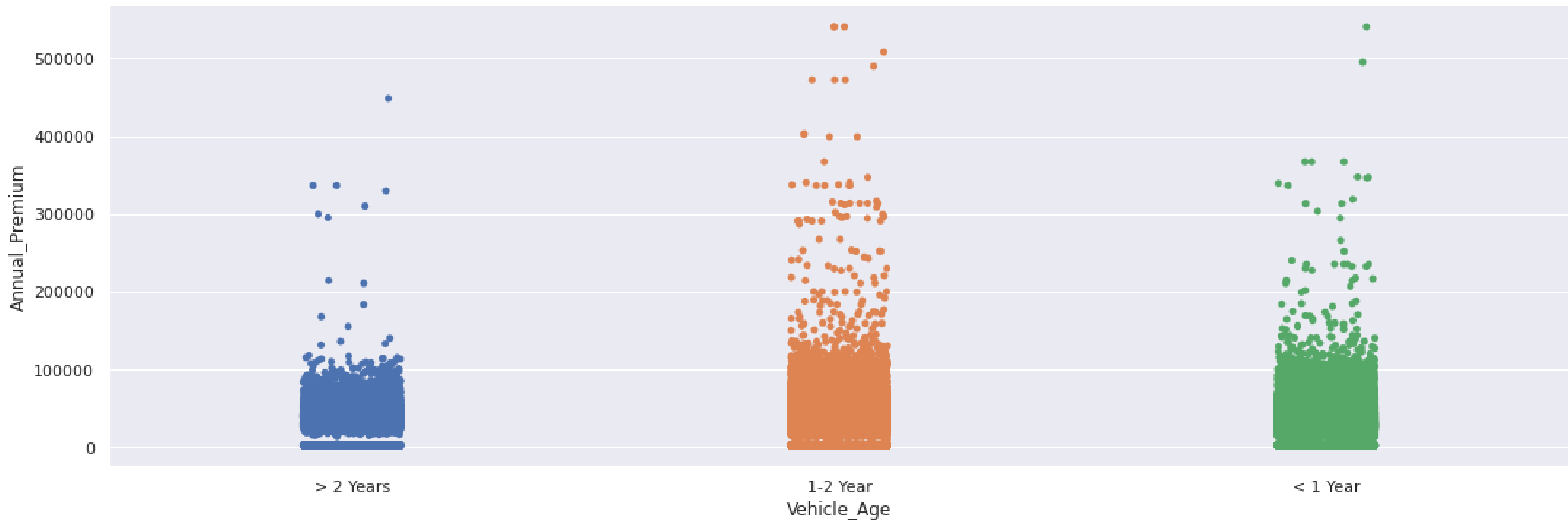
In [ ]:
```
train.nunique()
```

Out[24]:
```
Custmer_Id              381109
Gender                  2
Age                     66
DL                      2
City_Code               53
Insured                 2
Vehicle_Age             3
Vehicle_Damage          2
Annual_Premium          48838
Sales_Channel           155
Customer_Association    290
Response                2
dtype: int64
```

In [ ]:
```
_ = sns.catplot(x="DL", y="Annual_Premium", data=train, height=5, aspect=24/8)
```



In [ ]:
```
_ = sns.catplot(x="Vehicle_Age", y="Annual_Premium", data=train, height=5, aspect=24/8)
```



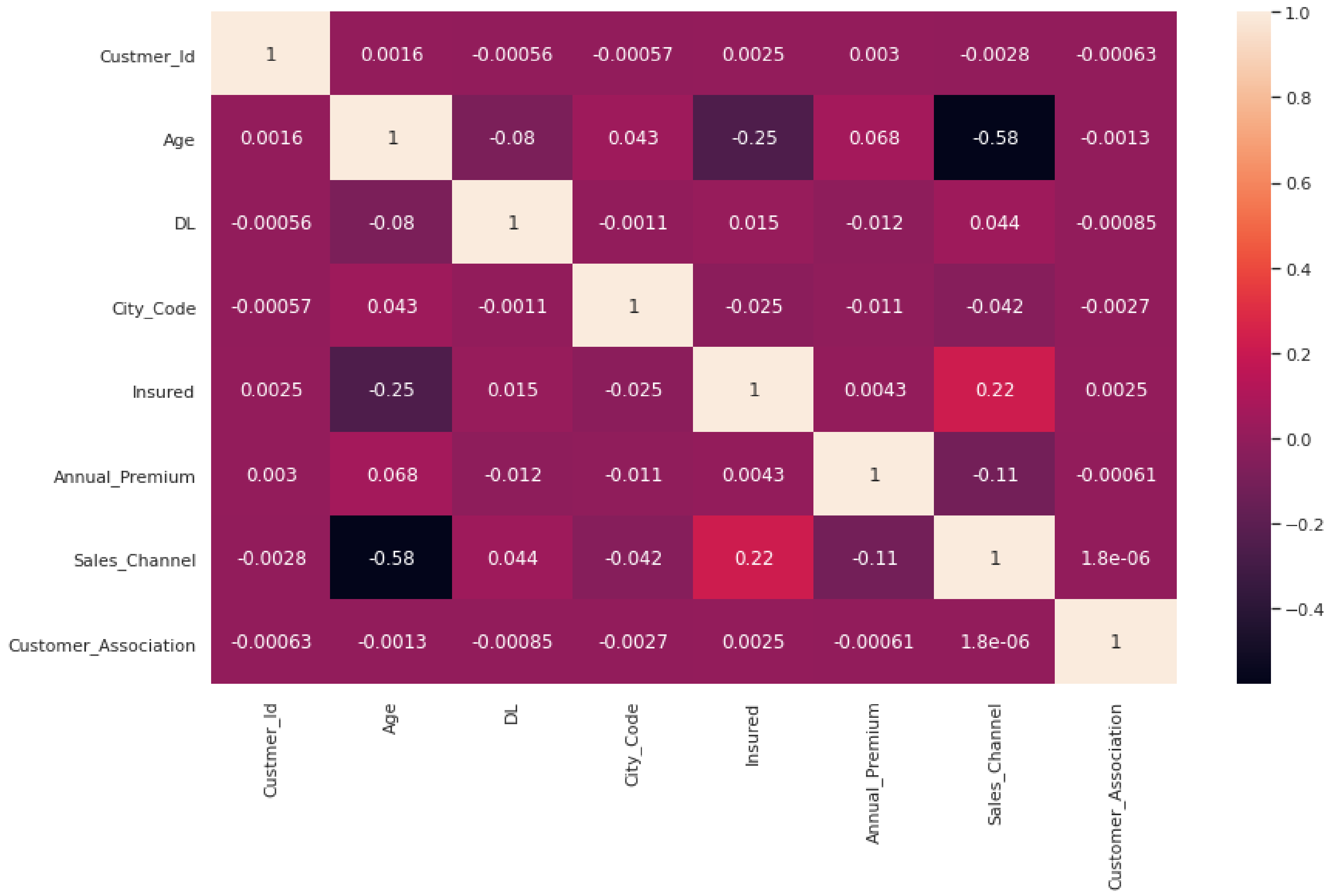
Observations

1. Customers with DL paid Higher Annual premiums than customers without DL
2. Customers with Vehicle age 1-2 years paid higher premiums

Correlations

In [ ]:
```
plt.figure(figsize=(14, 8))
_ = sns.heatmap(train[num_cols].corr(), annot=True)
```

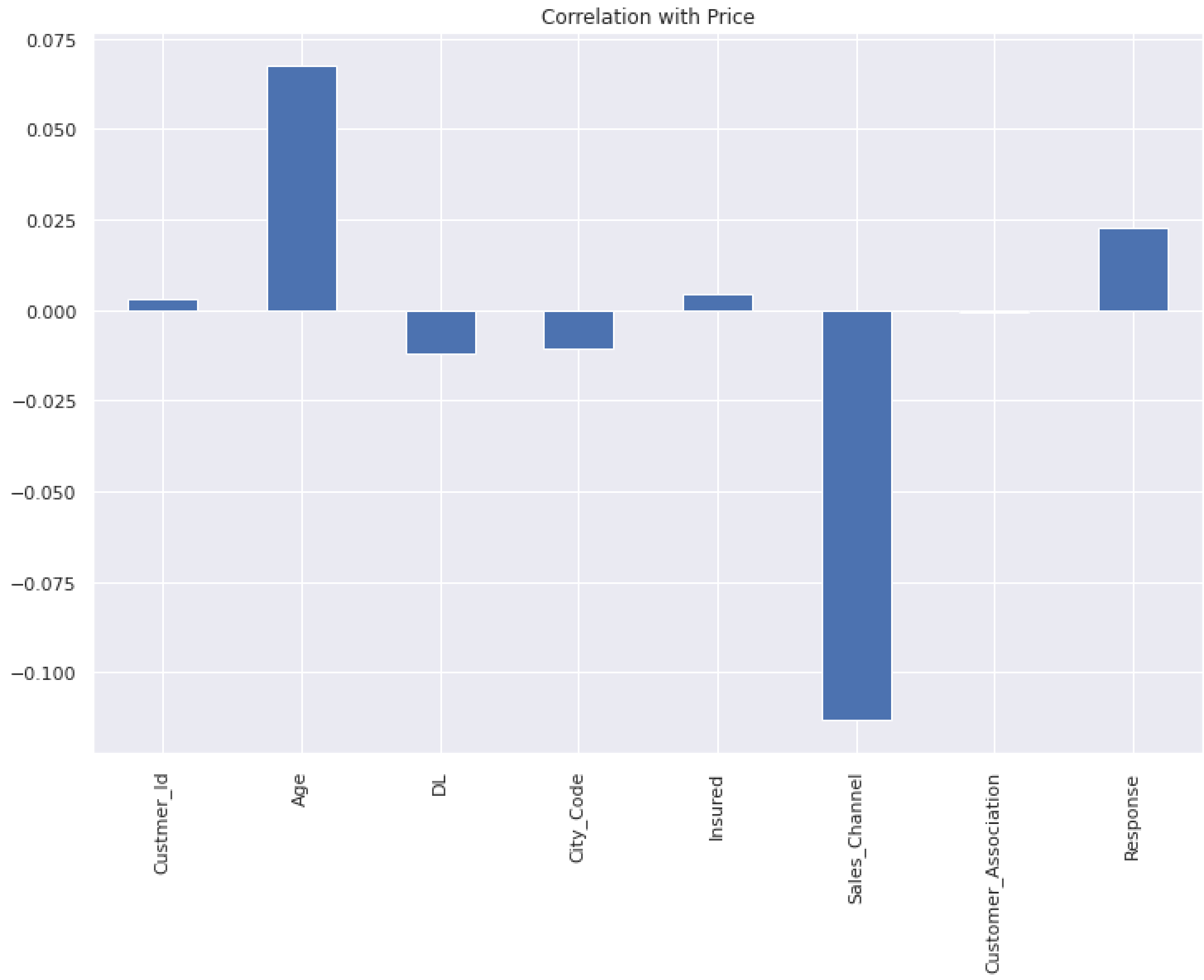
Out[27]: <Figure size 1008x576 with 0 Axes>



Factors affecting Annual Premium

In [ ]:
```
train.drop('Annual_Premium', axis=1).corrwith(train.Annual_Premium).plot(kind='bar', grid=True, figsize=(12, 8),
                                                    title="Correlation with Price")
```

Out[28]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f8b9173d320>`



Observations

1. As the customers age increases, they tend to go for higher subscriptions

Model

In [10]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

In [31]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-whitegrid')

from sklearn.preprocessing import LabelEncoder

import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.metrics import accuracy_score, f1_score

from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score


from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

from lightgbm import LGBMClassifier
#from catboost import CatBoostClassifier
from xgboost import XGBClassifier

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

pd.set_option('display.max_colwidth', -1)

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

Data

In [11]:
```
train.head()
```

Out[11]:

	Custmer_Id	Gender	Age	DL	City_Code	Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Sales_Channel	Customer_Associat
0	1	Male	44	1	28	0	> 2 Years	Yes	40454	26	2
1	2	Male	76	1	3	0	1-2 Year	No	33536	26	18
2	3	Male	47	1	28	0	> 2 Years	Yes	38294	26	2
3	4	Male	21	1	11	1	< 1 Year	No	28619	152	20
4	5	Female	29	1	41	1	< 1 Year	No	27496	152	3

In [12]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 310176 entries, 0 to 310175
Data columns (total 12 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Custmer_Id            310176 non-null  int64  
 1   Gender                310176 non-null  object 
 2   Age                   310176 non-null  int64  
 3   DL                    310176 non-null  int64  
 4   City_Code             310176 non-null  int64  
 5   Insured               310176 non-null  int64  
 6   Vehicle_Age           310176 non-null  object 
 7   Vehicle_Damage        310176 non-null  object 
 8   Annual_Premium        310176 non-null  int64  
 9   Sales_Channel         310176 non-null  int64  
 10  Customer_Association  310175 non-null  float64
 11  Response              310175 non-null  float64
dtypes: float64(2), int64(7), object(3)
memory usage: 28.4+ MB
```

Target and Features

In [13]:
```
ID_COL, TARGET_COL = 'Custmer_Id', 'Response'
features = [c for c in train.columns if c not in [ID_COL, TARGET_COL]]
```

In [14]:
```
cat_cols = ['Gender',
 'Vehicle_Damage',
 'Vehicle_Age']

num_cols = [c for c in features if c not in cat_cols]
```

Label Encoding

In [15]:
```
le=LabelEncoder()
train['Gender']=le.fit_transform(train['Gender'])
train['Vehicle_Damage']=le.fit_transform(train['Vehicle_Damage'])
train['Vehicle_Age']=train['Vehicle_Age'].map({'< 1 Year':0,'1-2 Year':1,'> 2 Years':2})
```

In [16]:
```
train.head()
```

Out[16]:

	Custmer_Id	Gender	Age	DL	City_Code	Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Sales_Channel	Customer_Associat
0	1	1	44	1	28	0	2	1	40454	26	2
1	2	1	76	1	3	0	1	0	33536	26	18
2	3	1	47	1	28	0	2	1	38294	26	2
3	4	1	21	1	11	1	0	0	28619	152	20
4	5	0	29	1	41	1	0	0	27496	152	3

Split the train set into train and validation sets.

We will use 80-20 split with 80% of the rows belonging to training data. Stratified Sampling is necessary, since the dataset is highly imbalanced. Stratified sampling ensures that the minority class is distributed proportionally among the two classes.

In [19]:
```
features = [c for c in train.columns if c not in [ID_COL, TARGET_COL]]
```

In [21]:
```
train.isnull().sum()
```

Out[21]:
```
Custmer_Id              0
Gender                  0
Age                     0
DL                      0
City_Code               0
Insured                 0
Vehicle_Age             0
Vehicle_Damage          0
Annual_Premium          0
Sales_Channel           0
Customer_Association    1
Response                1
dtype: int64
```

Since only 1 record had null, thus deleting it.

In [22]:
```
train = train.dropna()
```

In [23]:
```
trn, val = train_test_split(train, test_size=0.2, random_state = 1, stratify = train[TARGET_COL])

###### Input to our model will be the features
X_trn, X_val = trn[features], val[features]

###### Output of our model will be the TARGET_COL
y_trn, y_val = trn[TARGET_COL], val[TARGET_COL]

##### Features for the test data that we will be predicting
X_test = test[features]
```

In [25]:
```
train.shape,trn.shape, val.shape
```

Out[25]:
```
((310175, 12), (248140, 12), (62035, 12))
```

SMOTE Over Sampling

In [26]:
```
from imblearn.over_sampling import SMOTE
smote=SMOTE()
```

In [27]:
```
#trn
X_trn_smote, y_trn_smote= smote.fit_sample(X_trn,y_trn)
#val
X_val_smote, y_val_smote= smote.fit_sample(X_val,y_val)
```

In [28]:
```
from collections import Counter
print('#trn\nBefore SMOTE:' , Counter(y_trn))
print('After SMOTE:' , Counter(y_trn_smote))

print('\n#val\nBefore SMOTE:' , Counter(y_val))
print('After SMOTE:' , Counter(y_val_smote))
```

```
#trn
Before SMOTE: Counter({0.0: 217640, 1.0: 30500})
After SMOTE: Counter({1.0: 217640, 0.0: 217640})

#val
Before SMOTE: Counter({0.0: 54410, 1.0: 7625})
After SMOTE: Counter({0.0: 54410, 1.0: 54410})
```

Decision Tree

In [29]:
```
clf1 = DecisionTreeClassifier(random_state = 1)
_ = clf1.fit(X_trn_smote, y_trn_smote)

#prediction
preds_val = clf1.predict(X_val_smote)
preds_val_proba = clf1.predict_proba(X_val_smote)
```

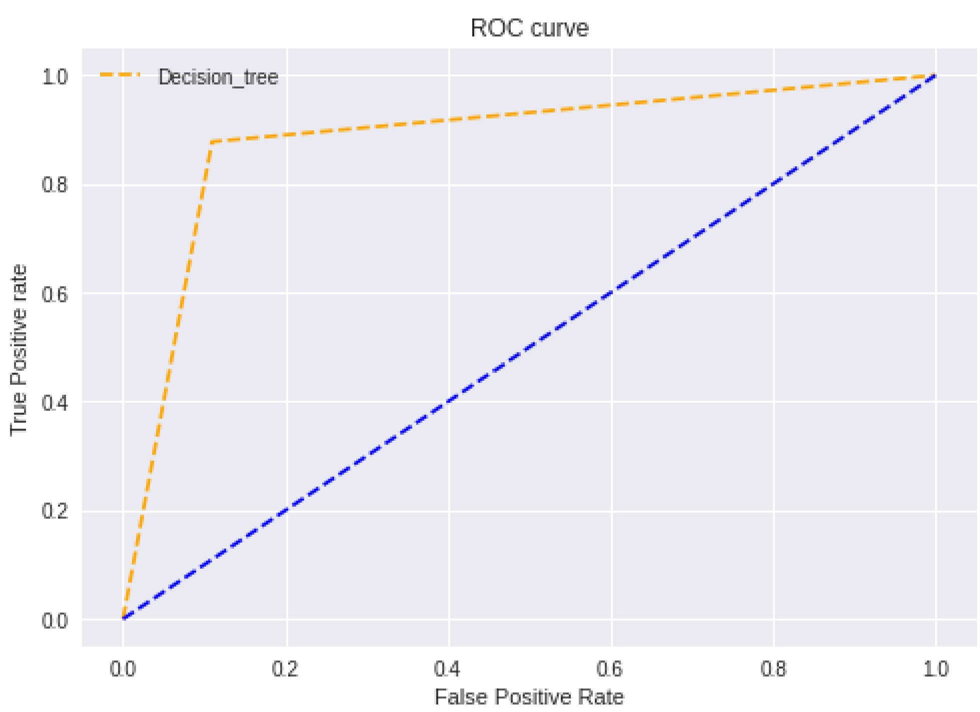
In [32]:
```
# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_val_smote, preds_val_proba[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_val_smote))]
p_fpr, p_tpr, _ = roc_curve(y_val_smote, random_probs, pos_label=1)

# auc scores
auc_score1 = roc_auc_score(y_val_smote, preds_val_proba[:,1])

print(auc_score1)
```

```
0.8840448025418856
```

Plot and Results

In [33]:
```
print('F1 Score\n',f1_score(y_val_smote, preds_val))

print(metrics.classification_report(y_val_smote, preds_val))

plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Decision_tree')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')

# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```

```
F1 Score
 0.8833071303061131
              precision    recall  f1-score   support

         0.0       0.88      0.89      0.88     54410
         1.0       0.89      0.88      0.88     54410

    accuracy                           0.88    108820
   macro avg       0.88      0.88      0.88    108820
weighted avg       0.88      0.88      0.88    108820
```



Decision Tree- Hyperparameter tuning with Randomized Search CV

In [36]:
```
from sklearn.model_selection import RandomizedSearchCV

#Params to tune
param={
    'max_depth': [4, 6, 8, 10, 12],
    'criterion': ['gini', 'entropy'],
    'min_samples_split': [2, 10, 20, 30, 40],
    'max_features': [0.2, 0.4, 0.6, 0.8, 1],
    'max_leaf_nodes': [8, 16, 32, 64, 128],
    'class_weight': [{0: 1, 1: 1}, {0: 1, 1: 2}, {0: 1, 1: 3}, {0: 1, 1: 4}, {0: 1, 1: 5}]
}
```

In [37]:
```
#tuning
clf2 = RandomizedSearchCV(DecisionTreeClassifier(),
                          param,
                          n_iter=20,
                          scoring='f1',
                          random_state=1)
```

In [38]:
```
#Param results
search=clf2.fit(train[features],train[TARGET_COL ])
search.best_params_
```

Out[38]:
```
{'class_weight': {0: 1, 1: 3},
 'criterion': 'gini',
 'max_depth': 8,
 'max_features': 0.4,
 'max_leaf_nodes': 128,
 'min_samples_split': 20}
```

In [39]:
```
#Saving result
optimal_params={'class_weight': {0: 1, 1: 3},
 'criterion': 'gini',
 'max_depth': 8,
 'max_features': 0.4,
 'max_leaf_nodes': 128,
 'min_samples_split': 20}
```

In [40]:
```
#Validation Score
clf2=DecisionTreeClassifier(random_state=1, **optimal_params)
_=clf2.fit(X_trn_smote,y_trn_smote)
```

In [41]:
```
#prediction
preds_val = clf2.predict(X_val_smote)
preds_val_proba = clf2.predict_proba(X_val_smote)
```

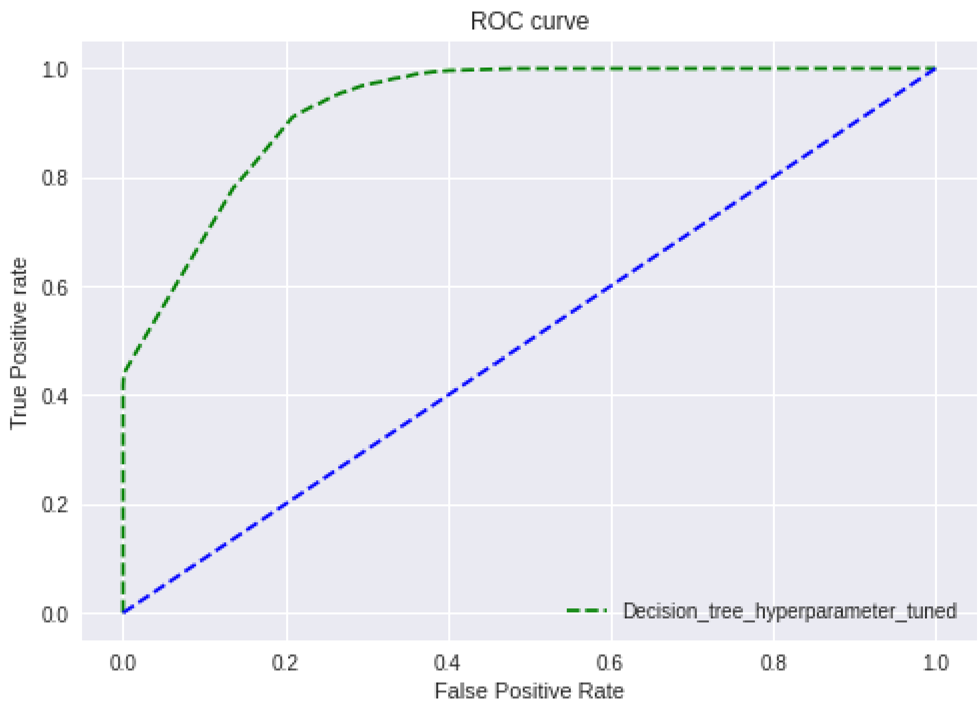
In [44]:
```
# roc curve for models
fpr2, tpr2, thresh2 = roc_curve(y_val_smote, preds_val_proba[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_val_smote))]
#p_fpr, p_tpr, _ = roc_curve(y_val_smote, random_probs, pos_label=1)

# auc scores
auc_score2 = roc_auc_score(y_val_smote, preds_val_proba[:,1])

print(auc_score2)
```

```
0.9286139721550937
```

Plot and Results

In [43]:
```
print('F1 Score\n',f1_score(y_val_smote, preds_val))

print(metrics.classification_report(y_val_smote, preds_val))

plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='Decision_tree_hyperparameter_tuned')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')

# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```

```
F1 Score
 0.8473594872852452
              precision    recall  f1-score   support

         0.0       0.97      0.66      0.79     54410
         1.0       0.75      0.98      0.85     54410

    accuracy                           0.82    108820
   macro avg       0.86      0.82      0.82    108820
weighted avg       0.86      0.82      0.82    108820
```



Random Forest Classifier

In [45]:
```
from sklearn.ensemble import RandomForestClassifier

clf3 = RandomForestClassifier(criterion = 'entropy', random_state = 42)
clf3.fit(X_trn_smote,y_trn_smote)
```

Out[45]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

In [46]:
```
#prediction
preds_val = clf3.predict(X_val_smote)
preds_val_proba = clf3.predict_proba(X_val_smote)
```

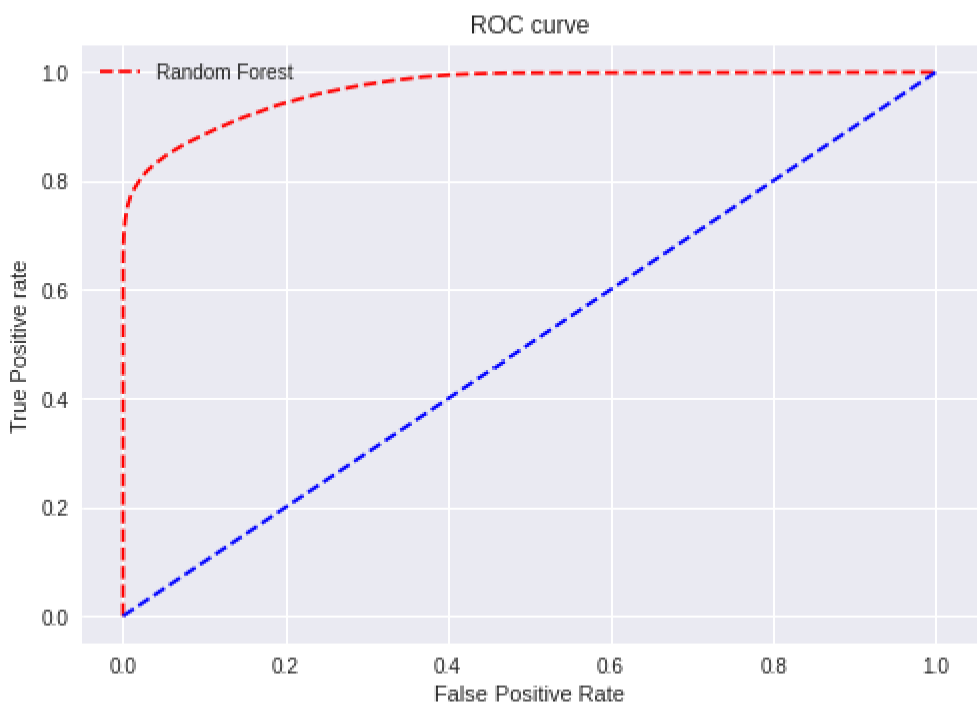
In [47]:
```
# roc curve for models
fpr3, tpr3, thresh3 = roc_curve(y_val_smote, preds_val_proba[:,1], pos_label=1)

# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_val_smote))]
p_fpr, p_tpr, _ = roc_curve(y_val_smote, random_probs, pos_label=1)

# auc scores
auc_score3 = roc_auc_score(y_val_smote, preds_val_proba[:,1])

print(auc_score3)
```

```
0.9693651195911862
```

PLot and Results

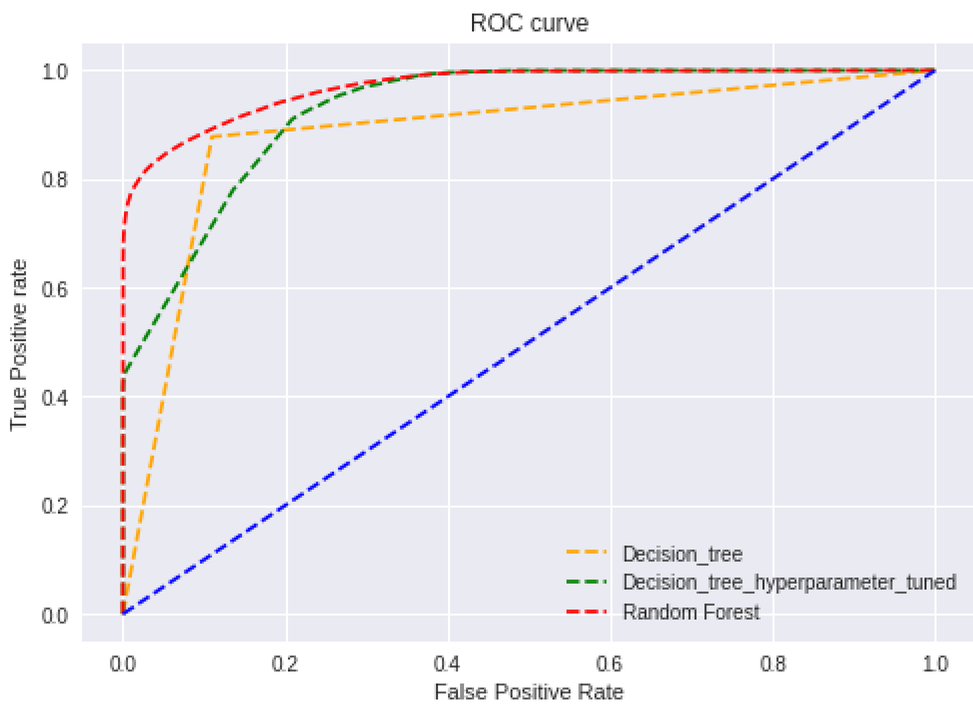
In [48]:
```
print('F1 Score\n',f1_score(y_val_smote, preds_val))

print(metrics.classification_report(y_val_smote, preds_val))

plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr3, tpr3, linestyle='--',color='red', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')

# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```

```
F1 Score
 0.8922862035646292
              precision    recall  f1-score   support

         0.0       0.87      0.94      0.90     54410
         1.0       0.93      0.86      0.89     54410

    accuracy                           0.90    108820
   macro avg       0.90      0.90      0.90    108820
weighted avg       0.90      0.90      0.90    108820
```



Baseline ML models Summary

In [49]:
```
plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Decision_tree')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='Decision_tree_hyperparameter_tuned')
plt.plot(fpr3, tpr3, linestyle='--',color='red', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')

# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```



Validation Strategy.

Problems with simple train_test_split validation

- We are not using complete 100 % of the dataset for training. More data implies more performance, if the data quality is good.
- We are not using complete 100 % of the dataset for validation. Our validation now is biased towards the validation set we have got through train_test_split. What if the test set is different from the validation set ?

In [50]:
```
target=train[TARGET_COL]
```

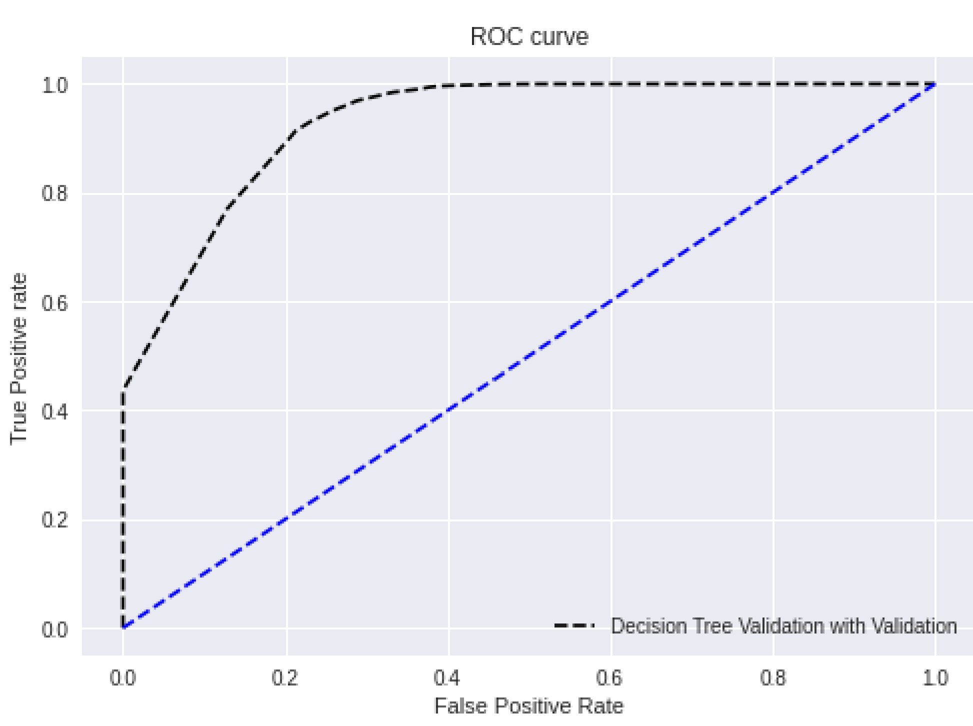
In [69]:
```
def run_clf_kfold(clf, train, features):

  N_SPLITS = 5


  oofs = np.zeros(len(train))        #train prediction
  #preds = np.zeros((len(test)))      #test prediction

  folds = StratifiedKFold(n_splits = N_SPLITS)

  for fold_, (trn_idx, val_idx) in enumerate(folds.split(train, train[TARGET_COL])):
    print(f'\n------------- Fold {fold_ + 1} -------------')

    ############# Get train, validation and test sets along with targets ################

    ### Training Set
    X_trn, y_trn = train[features].iloc[trn_idx], target.iloc[trn_idx]

    ### Validation Set
    X_val, y_val = train[features].iloc[val_idx], target.iloc[val_idx]



    ############# SMOTE ##################################
    smote=SMOTE()

    #trn
    X_trn_smote, y_trn_smote= smote.fit_sample(X_trn,y_trn)
    #val
    X_val_smote, y_val_smote= smote.fit_sample(X_val,y_val)
    ####################################################



    ############# Fitting and Predicting ################

    _ = clf.fit(X_trn_smote, y_trn_smote)

    ### Instead of directly predicting the classes we will obtain the probability of positive class.
    preds_val = clf.predict_proba(X_val_smote)[:, 1]    #oofs prediction

    fold_score = f1_score(y_val_smote, preds_val.round())   #fold score
    print(f'\nF1 score for validation set is {fold_score}')



    oofs = preds_val

    print(oofs.shape)


  oofs_score = f1_score(y_val_smote, oofs.round())    #combined OOFS score
  rocauc_score= roc_auc_score(y_val_smote, oofs.round())

  print(f'\n\nF1 score for oofs is {oofs_score}')

  print(metrics.classification_report(y_val_smote, oofs.round()))

  # roc curve for model
  fpr, tpr, thresh = roc_curve(y_val_smote, oofs, pos_label=1)

  # roc curve for tpr = fpr
  random_probs = [0 for i in range(len(y_val_smote))]
  p_fpr, p_tpr, _ = roc_curve(y_val_smote, random_probs, pos_label=1)

  # auc scores
  auc_score = roc_auc_score(y_val_smote, oofs.round())
  print(f'\nRoc_auc score for oofs is {auc_score}\n')

  plt.style.use('seaborn')

  # plot roc curves
  plt.plot(fpr, tpr, linestyle='--',color='black', label='Light GBM  with Validation')
  plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')

  # title
  plt.title('ROC curve')
  # x label
  plt.xlabel('False Positive Rate')
  # y label
  plt.ylabel('True Positive rate')

  plt.legend(loc='best')
  plt.savefig('ROC',dpi=300)
  plt.show();

  return oofs
```

KFold on Decision Tree

In [61]:
```
dt_params={'class_weight': {0: 1, 1: 3},
 'criterion': 'gini',
 'max_depth': 8,
 'max_features': 0.4,
 'max_leaf_nodes': 128,
 'min_samples_split': 20}
```

In [62]:
```
#DT_Validation score With tuning
clf = DecisionTreeClassifier(**dt_params)
dt_oofs = run_clf_kfold(clf, train, features)
```

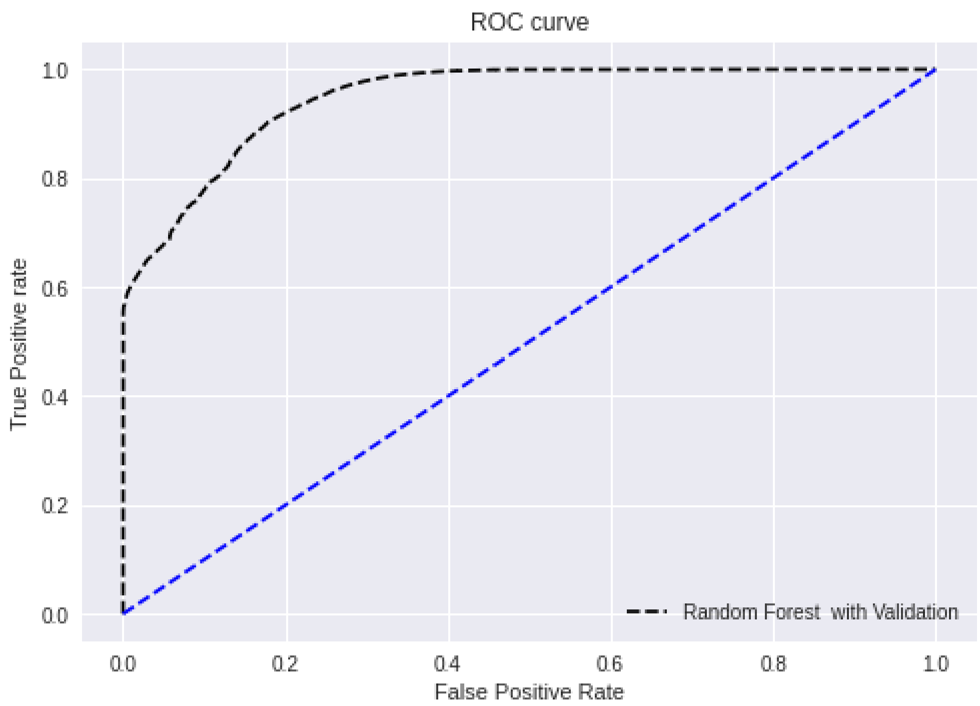
```
------------- Fold 1 -------------

F1 score for validation set is 0.8482206546722676
(108820,)

------------- Fold 2 -------------

F1 score for validation set is 0.8525780103375662
(108820,)

------------- Fold 3 -------------

F1 score for validation set is 0.8498081418075354
(108820,)

------------- Fold 4 -------------

F1 score for validation set is 0.8543408073926025
(108820,)

------------- Fold 5 -------------

F1 score for validation set is 0.8516597510373445
(108820,)


F1 score for oofs is 0.8516597510373445
              precision    recall  f1-score   support

         0.0       0.97      0.68      0.80     54410
         1.0       0.75      0.98      0.85     54410

    accuracy                           0.83    108820
   macro avg       0.86      0.83      0.83    108820
weighted avg       0.86      0.83      0.83    108820


Roc_auc score for oofs is 0.8291674324572689
```



Random Forest Validation

In [67]:
```
#RMF_Validation score With tuning
clf = RandomForestClassifier(**dt_params)
rmf_oofs = run_clf_kfold(clf, train, features)
```

```
------------- Fold 1 -------------

F1 score for validation set is 0.8496355207354346
(108820,)

------------- Fold 2 -------------

F1 score for validation set is 0.8487119807487474
(108820,)

------------- Fold 3 -------------

F1 score for validation set is 0.8465769161165446
(108820,)

------------- Fold 4 -------------

F1 score for validation set is 0.8499502047139538
(108820,)

------------- Fold 5 -------------

F1 score for validation set is 0.8506057804016407
(108820,)


F1 score for oofs is 0.8506057804016407
              precision    recall  f1-score   support

         0.0       0.98      0.66      0.79     54410
         1.0       0.75      0.99      0.85     54410

    accuracy                           0.83    108820
   macro avg       0.86      0.83      0.82    108820
weighted avg       0.86      0.83      0.82    108820


Roc_auc score for oofs is 0.8262911229553391
```

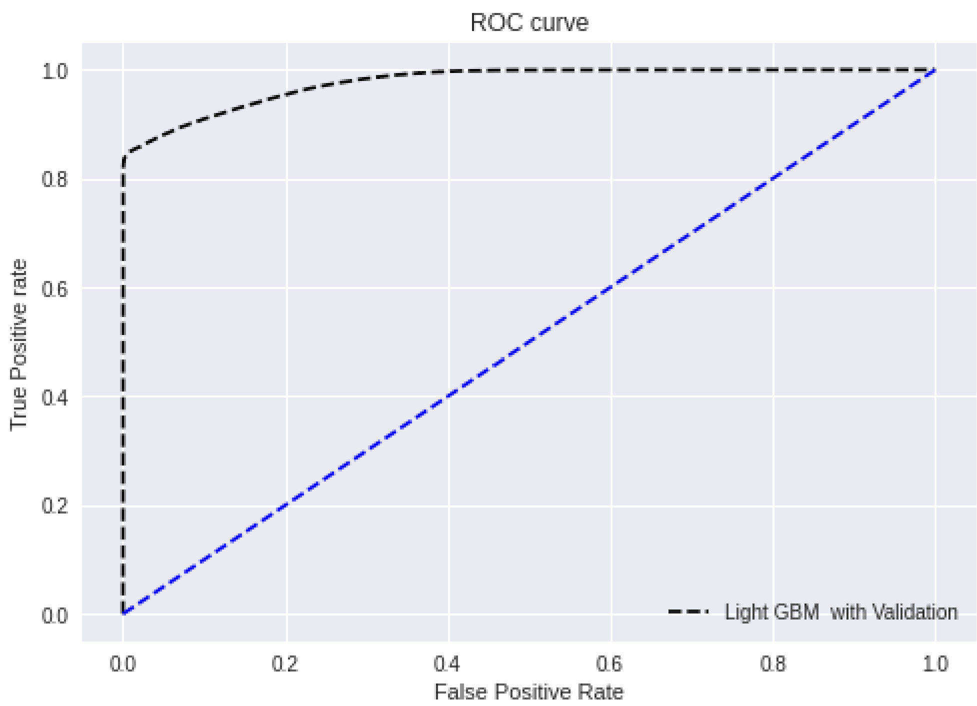


Gradient Boosting

LightGBM

In [70]:
```
clf = LGBMClassifier()
lgb_oofs= run_clf_kfold(clf, train, features)
```

```
------------- Fold 1 -------------

F1 score for validation set is 0.9149297964339503
(108820,)

------------- Fold 2 -------------

F1 score for validation set is 0.9136500934937909
(108820,)

------------- Fold 3 -------------

F1 score for validation set is 0.9150548470870964
(108820,)

------------- Fold 4 -------------

F1 score for validation set is 0.9156654522905202
(108820,)

------------- Fold 5 -------------

F1 score for validation set is 0.9134050359642584
(108820,)


F1 score for oofs is 0.9134050359642584
              precision    recall  f1-score   support

         0.0       0.88      0.97      0.92     54410
         1.0       0.97      0.87      0.91     54410

    accuracy                           0.92    108820
   macro avg       0.92      0.92      0.92    108820
weighted avg       0.92      0.92      0.92    108820


Roc_auc score for oofs is 0.917800036757949
```



Type *Markdown* and LaTeX: α^2

In [74]:
```
pip install catboost
```

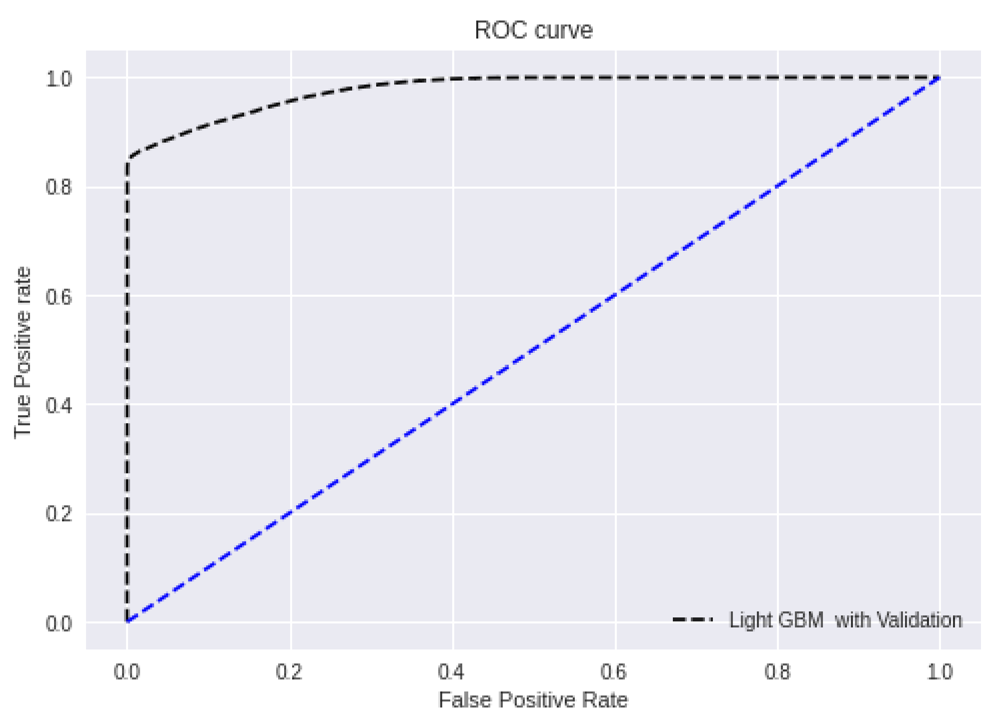
```
Collecting catboost
  Downloading https://files.pythonhosted.org/packages/20/37/bc4e0ddc30c07a96482abf1de7ed1ca54e59bba2026a33bca6d2ef286e5b/catboost-0.24.4-cp36-none-manylinux1_x86_64.whl (https://files.pythonhosted.org/packages/20/37/bc4e0ddc30c07a96482abf1de7ed1ca54e59bba2026a33bca6d2ef286e5b/catboost-0.24.4-cp36-none-manylinux1_x86_64.whl) (65.7MB)
     |████████████████████████████████| 65.8MB 47kB/s
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from catboost) (1.15.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from catboost) (1.4.1)
Requirement already satisfied: pandas>=0.24.0 in /usr/local/lib/python3.6/dist-packages (from catboost) (1.1.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (from catboost) (3.2.2)
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from catboost) (4.4.1)
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-packages (from catboost) (0.10.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.6/dist-packages (from catboost) (1.19.4)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.24.0->catboost) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.6/dist-packages (from pandas>=0.24.0->catboost) (2.8.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib->catboost) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->catboost) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->catboost) (1.3.1)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages (from plotly->catboost) (1.3.3)
Installing collected packages: catboost
Successfully installed catboost-0.24.4
```

In [75]:
```
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
```

XGBoost

In [77]:
```
clf = XGBClassifier(n_estimators = 1000,
                    max_depth = 6,
                    learning_rate = 0.05,
                    colsample_bytree = 0.5,
                    random_state=1452,
                    )

fit_params = {'verbose': 200, 'early_stopping_rounds': 200}
xgb_oofs= run_clf_kfold(clf, train, features)
```

```
------------- Fold 1 -------------

F1 score for validation set is 0.9173192742106174
(108820,)

------------- Fold 2 -------------

F1 score for validation set is 0.9208252238224991
(108820,)

------------- Fold 3 -------------

F1 score for validation set is 0.9204990527421397
(108820,)

------------- Fold 4 -------------

F1 score for validation set is 0.9210153632785413
(108820,)

------------- Fold 5 -------------

F1 score for validation set is 0.9180807659879635
(108820,)


F1 score for oofs is 0.9180807659879635
              precision    recall  f1-score   support

         0.0       0.88      0.97      0.93     54410
         1.0       0.97      0.87      0.92     54410

    accuracy                           0.92    108820
   macro avg       0.93      0.92      0.92    108820
weighted avg       0.93      0.92      0.92    108820


Roc_auc score for oofs is 0.9223212644734424
```



In [ ]: