Exploratory Analysis of Auto Insurance Claims

Problem Statement:

This project's goal is to examine car insurance claims and conduct exploratory data analysis.

A number of factors might influence the monthly premium payments of a vehicle insurance policy. When it comes to insurance providers, 68 percent of complaints are about claims handling. Claim handling was the biggest area of concern among consumers. We will acquire insights and conclusions to our inquiries using an investigation by using vehicle insurance claims data.

It would be useful to both customers and insurance agencies, if we could acquire solid insights from this data. If we could figure out which demographics, ages, and other factors are most likely to file a claim. We are capable of providing better services and resolving 68% of concerns.

Raw Data Structure:

To undertake exploratory data analysis, we will use a data set collected from www.data.world. This dataset has a total of 10,000 rows and 19 columns.

We will only use a few of the 19 columns, despite the fact that we can use all of the rows.

The raw data that we obtained are 19 fields:

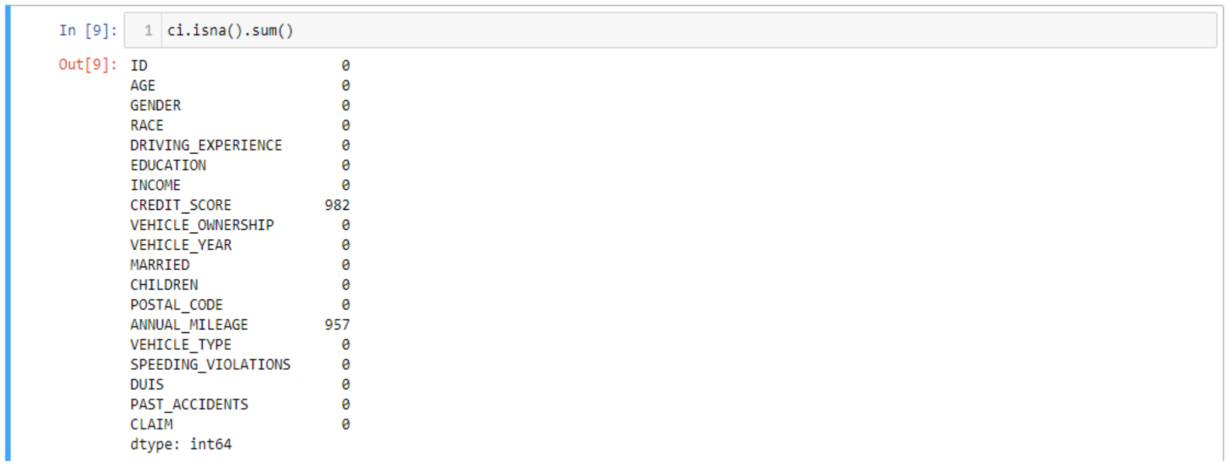
1) ID
2) AGE
3) GENDER
4) RACE
5) DRIVING_EXPERIENCE
6) EDUCATION
7) INCOME
8) CREDIT_SCORE
9) VEHICLE_OWNERSHIP
10) VEHICLE_YEAR
11) MARRIED
12) CHILDREN
13) POSTAL_CODE
14) ANNUAL_MILEAGE
15) VEHICLE_TYPE
16) SPEEDING_VIOLATIONS
17) DUIS
18) PAST_ACCIDENTS
19) CLAIM

Methods:

Data Cleaning and Organization:

We used the pandas package to load the raw CSV data into a dataframe after importing it into the jupyter notebook.

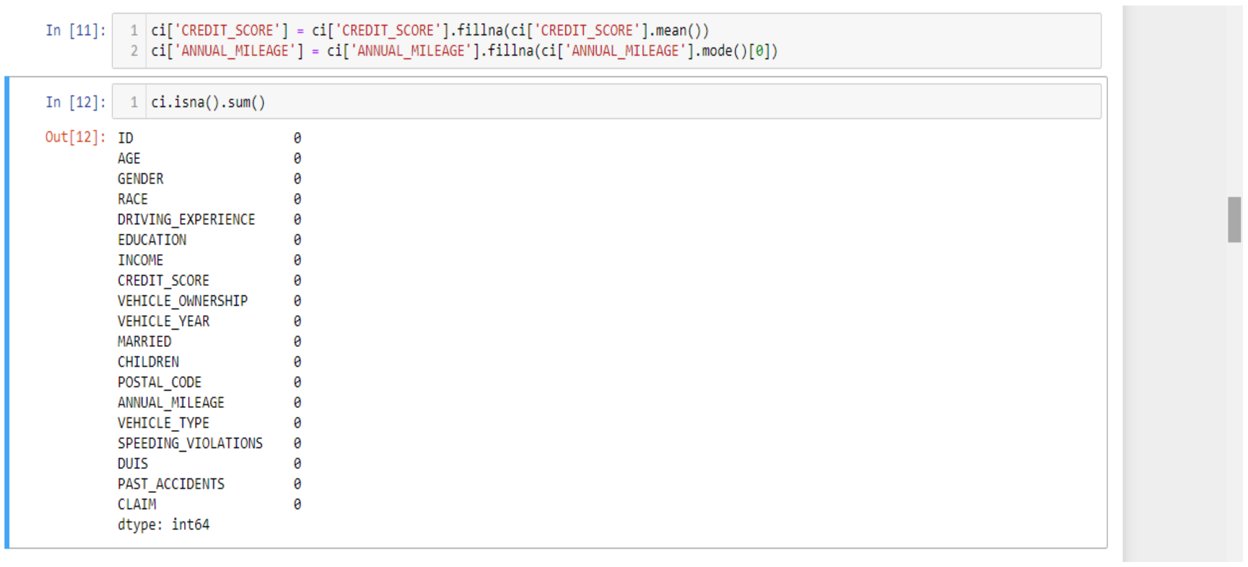
We filtered away Null values and cleansed the data by checking for them.



```
In [9]:  ci.isna().sum()

Out[9]: ID                      0
        AGE                     0
        GENDER                  0
        RACE                    0
        DRIVING_EXPERIENCE      0
        EDUCATION               0
        INCOME                  0
        CREDIT_SCORE          982
        VEHICLE_OWNERSHIP       0
        VEHICLE_YEAR            0
        MARRIED                 0
        CHILDREN                0
        POSTAL_CODE             0
        ANNUAL_MILEAGE        957
        VEHICLE_TYPE            0
        SPEEDING_VIOLATIONS     0
        DUIS                    0
        PAST_ACCIDENTS          0
        CLAIM                   0
        dtype: int64
```

There are 982 missing fields in the CREDIT_SCORE column and 957 missing fields in the ANNUAL_MILEAGE column.

Later, we replaced the Null values with Mean and Mode. For, CREDIT_SCORE we used mean() and for ANNUAL_MILEAGE we used mode() to replace the null value fields.



```
In [11]:  ci['CREDIT_SCORE'] = ci['CREDIT_SCORE'].fillna(ci['CREDIT_SCORE'].mean())
          ci['ANNUAL_MILEAGE'] = ci['ANNUAL_MILEAGE'].fillna(ci['ANNUAL_MILEAGE'].mode()[0])

In [12]:  ci.isna().sum()

Out[12]: ID                     0
         AGE                    0
         GENDER                 0
         RACE                   0
         DRIVING_EXPERIENCE     0
         EDUCATION              0
         INCOME                 0
         CREDIT_SCORE           0
         VEHICLE_OWNERSHIP      0
         VEHICLE_YEAR           0
         MARRIED                0
         CHILDREN               0
         POSTAL_CODE            0
         ANNUAL_MILEAGE         0
         VEHICLE_TYPE           0
         SPEEDING_VIOLATIONS    0
         DUIS                   0
         PAST_ACCIDENTS         0
         CLAIM                  0
         dtype: int64
```

Data Visualization:

Matplotlib is a popular data visualization library for Python and Numpy that runs on all platforms. It is an open-source replacement for MATLAB.

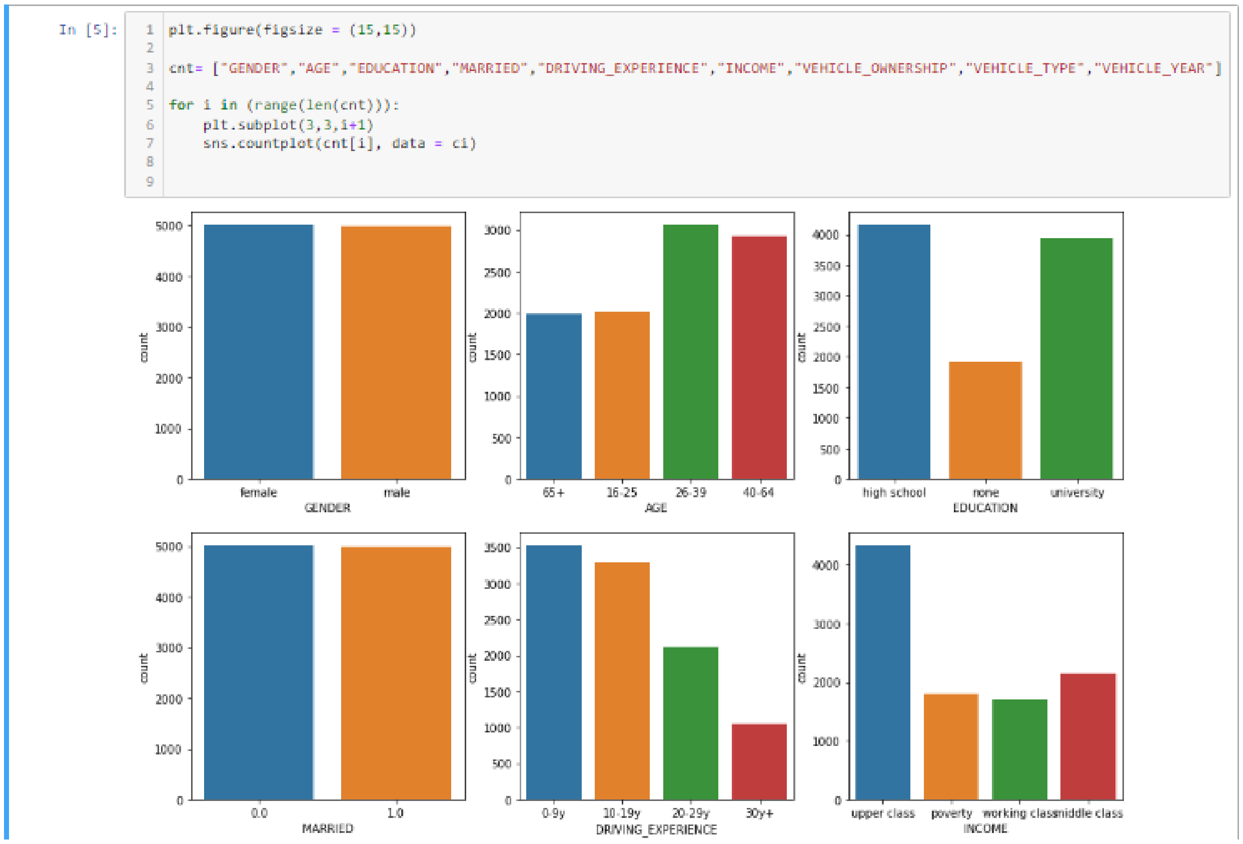
We used the Python libraries and imported matplotlib and seaborn libraries to generate visuals that permit us to start sharing our observations in a pictorial format that anybody can understand and benefit from.

- We created charts that shows the user details such as:

  Gender, Age, Education, Married, Driving experience, and Income.

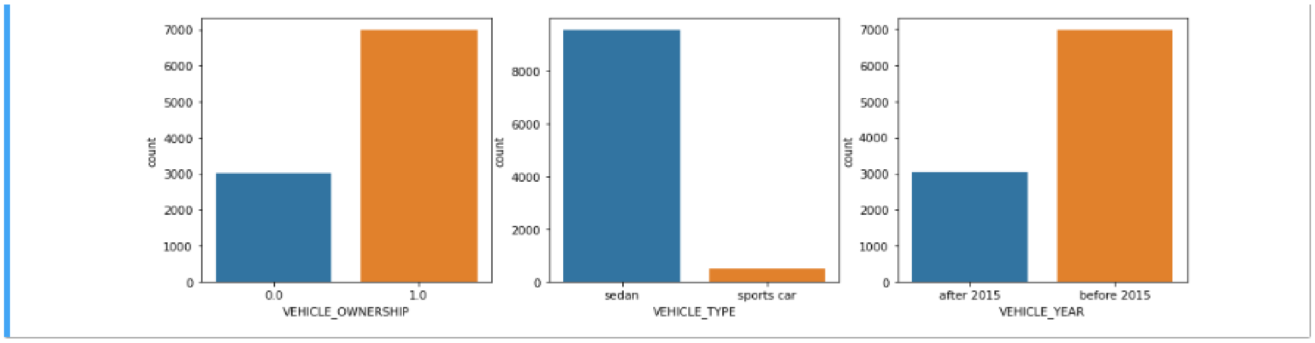
- Also the vehicle details such as:
  Vehicle_Ownership, Vehicle_Type and Vehicle_Year.

  Here we used **countplot** to generate graphs for individual columns.

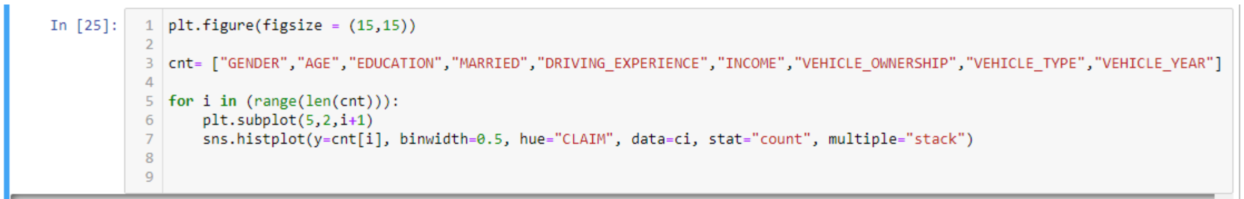


In [5]:

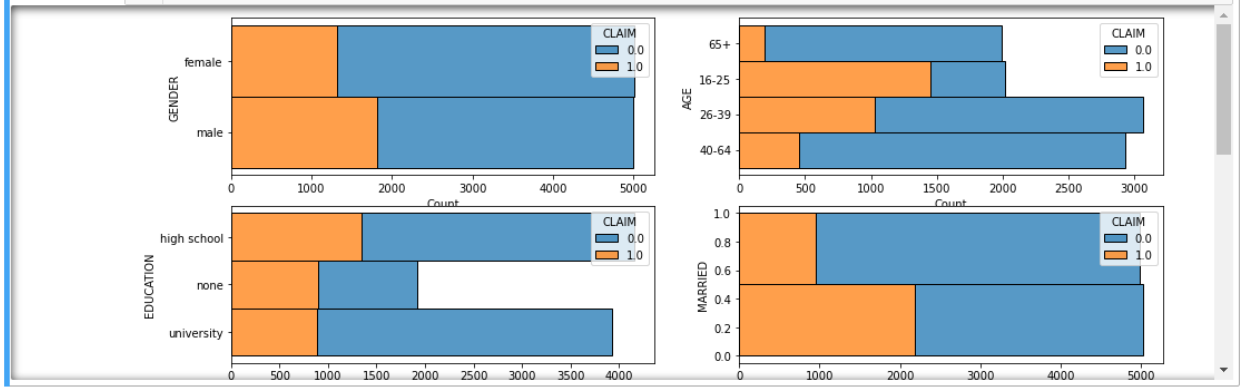
```
plt.figure(figsize = (15,15))

cnt= ["GENDER","AGE","EDUCATION","MARRIED","DRIVING_EXPERIENCE","INCOME","VEHICLE_OWNERSHIP","VEHICLE_TYPE","VEHICLE_YEAR"]

for i in (range(len(cnt))):
    plt.subplot(3,3,i+1)
    sns.countplot(cnt[i], data = ci)


```

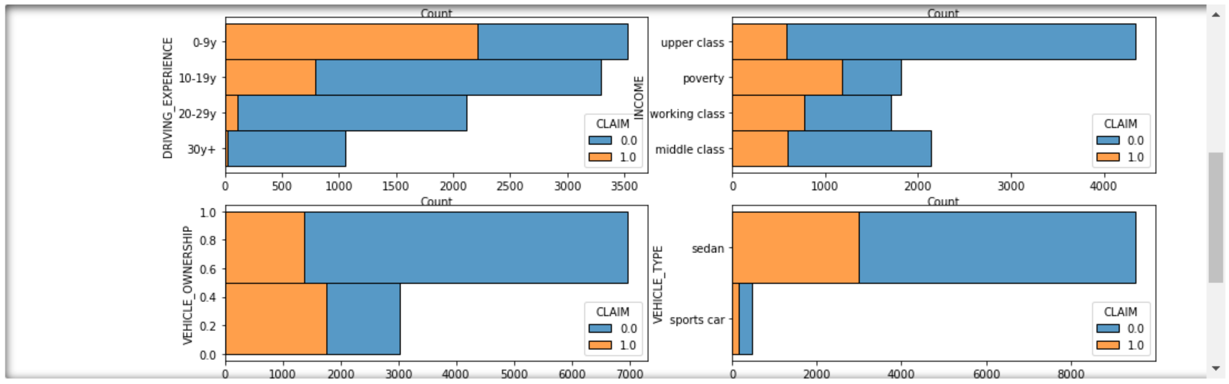


Later, we created **histplot** for each column with respect to "CLAIM" column as hue.



```
In [25]:
plt.figure(figsize = (15,15))

cnt= ["GENDER","AGE","EDUCATION","MARRIED","DRIVING_EXPERIENCE","INCOME","VEHICLE_OWNERSHIP","VEHICLE_TYPE","VEHICLE_YEAR"]

for i in (range(len(cnt))):
    plt.subplot(5,2,i+1)
    sns.histplot(y=cnt[i], binwidth=0.5, hue="CLAIM", data=ci, stat="count", multiple="stack")
```

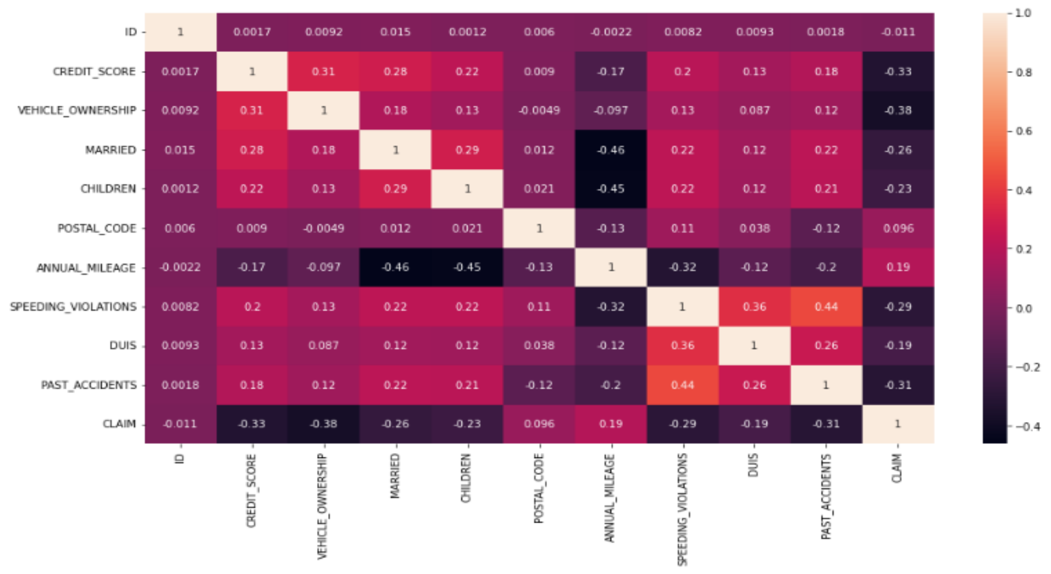




Plotting with the correlation matrix:

A correlation matrix is a table that incorporates correlation coefficients for various variables. The relationship between the two variables is represented from every cell in the matrix.

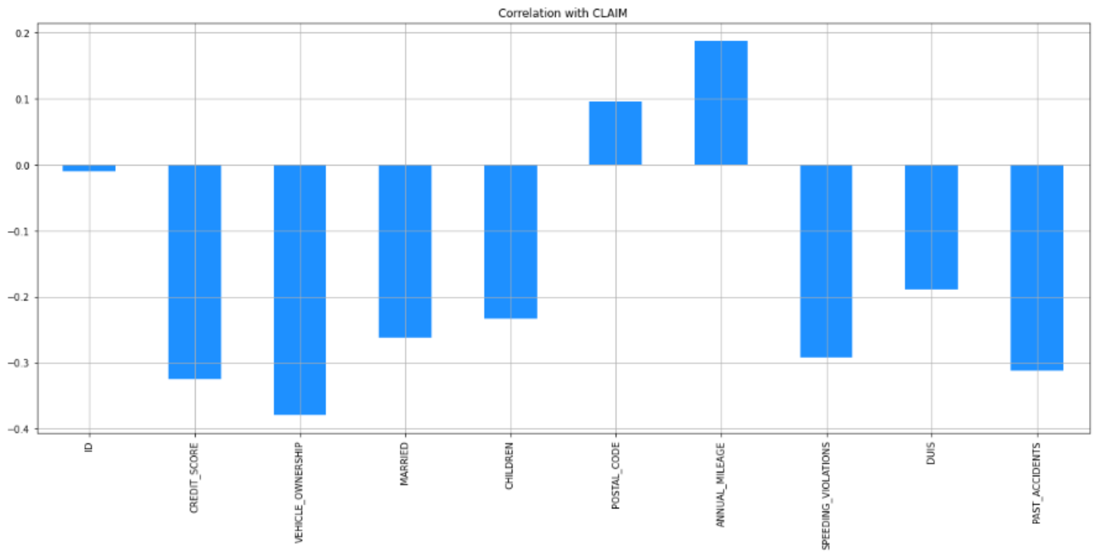
```
In [7]:  plt.figure(figsize = (16,8))
         sns.heatmap(ci.corr(), annot = True)
Out[7]:  <AxesSubplot:>
```



	ID	CREDIT_SCORE	VEHICLE_OWNERSHIP	MARRIED	CHILDREN	POSTAL_CODE	ANNUAL_MILEAGE	SPEEDING_VIOLATIONS	DUIS	PAST_ACCIDENTS	CLAIM
ID	1	0.0017	0.0092	0.015	0.0012	0.006	-0.0022	0.0082	0.0093	0.0018	-0.011
CREDIT_SCORE	0.0017	1	0.31	0.28	0.22	0.009	-0.17	0.2	0.13	0.18	-0.33
VEHICLE_OWNERSHIP	0.0092	0.31	1	0.18	0.13	-0.0049	-0.097	0.13	0.087	0.12	-0.38
MARRIED	0.015	0.28	0.18	1	0.29	0.012	-0.46	0.22	0.12	0.22	-0.26
CHILDREN	0.0012	0.22	0.13	0.29	1	0.021	-0.45	0.22	0.12	0.21	-0.23
POSTAL_CODE	0.006	0.009	-0.0049	0.012	0.021	1	-0.13	0.11	0.038	-0.12	0.096
ANNUAL_MILEAGE	-0.0022	-0.17	-0.097	-0.46	-0.45	-0.13	1	-0.32	-0.12	-0.2	0.19
SPEEDING_VIOLATIONS	0.0082	0.2	0.13	0.22	0.22	0.11	-0.32	1	0.36	0.44	-0.29
DUIS	0.0093	0.13	0.087	0.12	0.12	0.038	-0.12	0.36	1	0.26	-0.19
PAST_ACCIDENTS	0.0018	0.18	0.12	0.22	0.21	-0.12	-0.2	0.44	0.26	1	-0.31
CLAIM	-0.011	-0.33	-0.38	-0.26	-0.23	0.096	0.19	-0.29	-0.19	-0.31	1

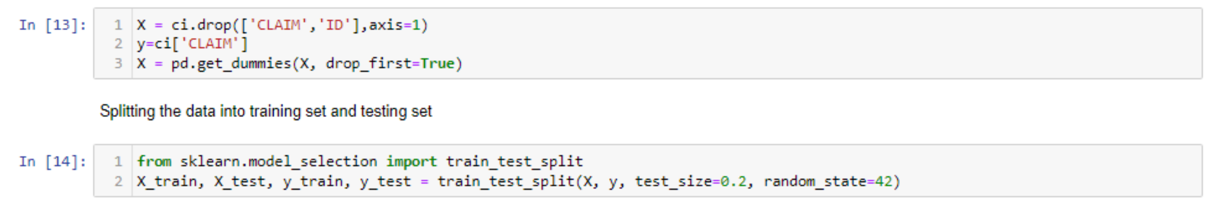
Plotting the correlation with respect to "CLAIM" field:

```
In [8]:  ci.drop('CLAIM', axis=1).corrwith(ci.CLAIM).plot(kind='bar', grid=True,
         figsize=(20, 8), title="Correlation with CLAIM",color="#1E90FF");
```



ML Modeling:

The data has been divided into two sets, the training set and the testing set, using an 80-20 split, which means we will use 80% of the data to train our models and 20% to test them.



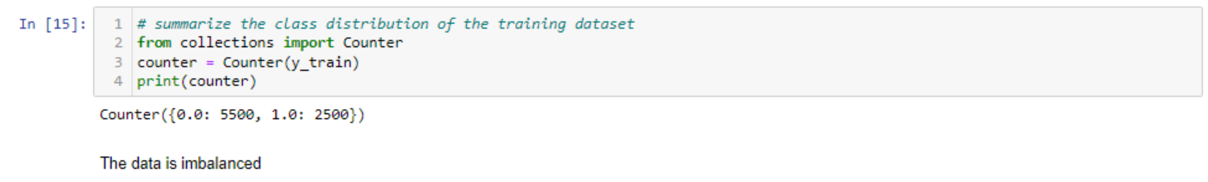
In [13]:
```python
X = ci.drop(['CLAIM','ID'],axis=1)
y=ci['CLAIM']
X = pd.get_dummies(X, drop_first=True)
```

Splitting the data into training set and testing set

In [14]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Handling Data Imbalance:

In the train sets the target class has an uneven distribution of observations i.e., one class label has high number of observations and the other has a very low number of observations as displayed below:

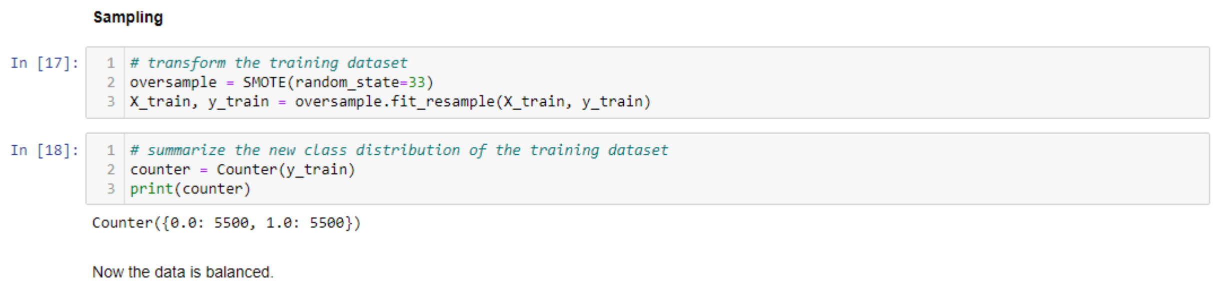


In [15]:
```python
# summarize the class distribution of the training dataset
from collections import Counter
counter = Counter(y_train)
print(counter)
```

```
Counter({0.0: 5500, 1.0: 2500})
```

The data is imbalanced

So, we used **oversampling** the minority class to make the data balanced.



Sampling

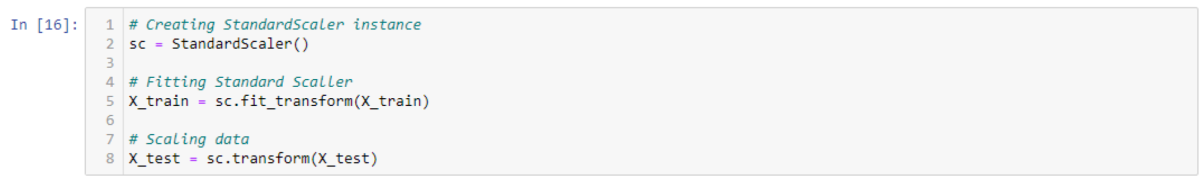
In [17]:
```python
# transform the training dataset
oversample = SMOTE(random_state=33)
X_train, y_train = oversample.fit_resample(X_train, y_train)
```

In [18]:
```python
# summarize the new class distribution of the training dataset
counter = Counter(y_train)
print(counter)
```

```
Counter({0.0: 5500, 1.0: 5500})
```

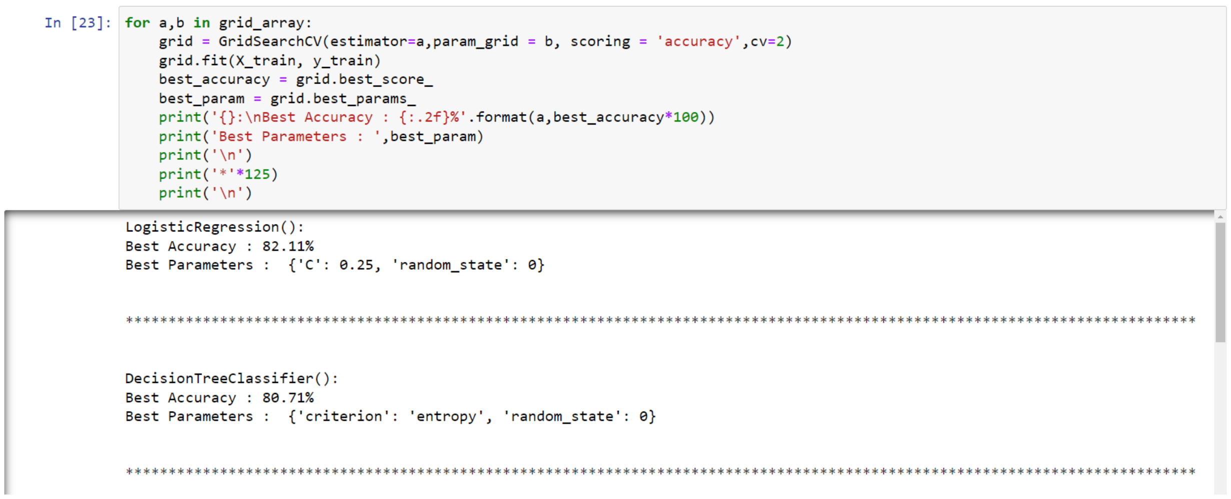
Now the data is balanced.

Finally, we scaled the data using the feature scaling.



In [16]:
```python
# Creating StandardScaler instance
sc = StandardScaler()

# Fitting Standard Scaller
X_train = sc.fit_transform(X_train)

# Scaling data
X_test = sc.transform(X_test)
```

To find the right parameter values we used **HyperParameter Tuning** and obtained the best fit parameter as shown below:



```
In [23]: for a,b in grid_array:
             grid = GridSearchCV(estimator=a,param_grid = b, scoring = 'accuracy',cv=2)
             grid.fit(X_train, y_train)
             best_accuracy = grid.best_score_
             best_param = grid.best_params_
             print('{}:\nBest Accuracy : {:.2f}%'.format(a,best_accuracy*100))
             print('Best Parameters : ',best_param)
             print('\n')
             print('*'*125)
             print('\n')
```

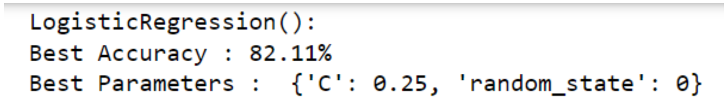
```
LogisticRegression():
Best Accuracy : 82.11%
Best Parameters :  {'C': 0.25, 'random_state': 0}


*****************************************************************************************************************************


DecisionTreeClassifier():
Best Accuracy : 80.71%
Best Parameters :  {'criterion': 'entropy', 'random_state': 0}


*****************************************************************************************************************************
```

Results:

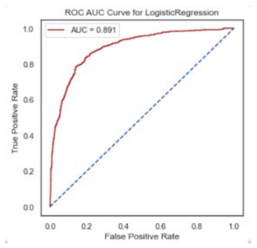
After tuning the parameters, we obtained the best fit parameters and plotted the ROC AUC curve for each of the models that we used as displayed below:

1) Logistic Regression

(LogisticRegression(),[{'C':[0.25,0.5,0.75,1],'random_state':[0]}])

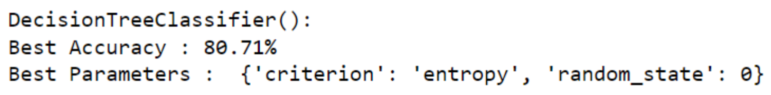


```
LogisticRegression():
Best Accuracy : 82.11%
Best Parameters :  {'C': 0.25, 'random_state': 0}
```

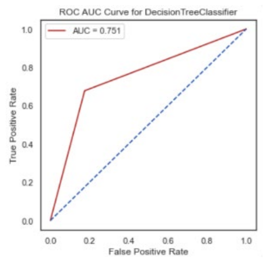


2) Decision Tree Classifier

(DecisionTreeClassifier(),[{'criterion':['gini','entropy'],'random_state':[0]}]),

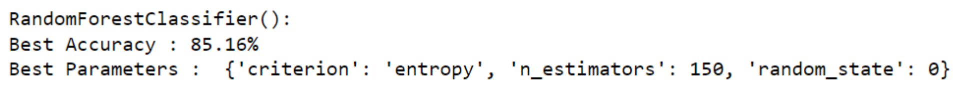


```
DecisionTreeClassifier():
Best Accuracy : 80.71%
Best Parameters :  {'criterion': 'entropy', 'random_state': 0}
```

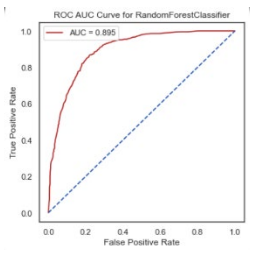


3) Random Forest Classifier

(RandomForestClassifier(),[{'n_estimators':[100,150,200],'criterion':['gini','entropy'],'random_state':[0]}])

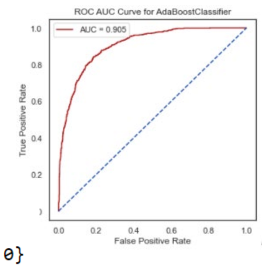


```
RandomForestClassifier():
Best Accuracy : 85.16%
Best Parameters :  {'criterion': 'entropy', 'n_estimators': 150, 'random_state': 0}
```



4) Ada Boost Classifier

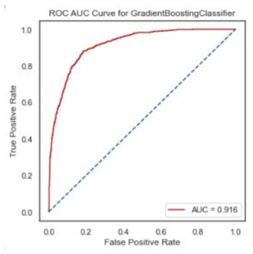
(AdaBoostClassifier(),{'n_estimators':[100,150,200],'learning_rate':[0.1, 0.5, 0.8, 1],'algorithm':['SAMME', 'SAMME.R'], 'random_state':[0]}])



```
AdaBoostClassifier():
Best Accuracy : 83.81%
Best Parameters :  {'algorithm': 'SAMME', 'learning_rate': 0.5, 'n_estimators': 200, 'random_state': 0}
```

5) Gradient Boosting Classifier

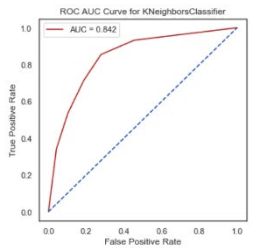
(GradientBoostingClassifier(),[{'n_estimators':[100,150,200],'criterion':['friedman_mse','mse'],'loss':['deviance','exponential'],'learning_rate':[0.1, 0.5, 0.8, 1],'random_state':[0]}])



```
GradientBoostingClassifier():
Best Accuracy : 83.52%
Best Parameters :  {'criterion': 'friedman_mse', 'learning_rate': 0.1, 'loss': 'deviance', 'n_estimators': 100, 'random_stat
e': 0}
```

6) K Neighbors Classifier

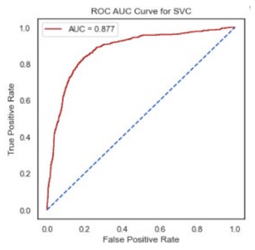
(KNeighborsClassifier(),[{'n_neighbors':[5,7, 8, 10], 'metric': ['euclidean', 'manhattan', 'chebyshev', 'minkowski']}])



```
KNeighborsClassifier():
Best Accuracy : 83.48%
Best Parameters :  {'metric': 'manhattan', 'n_neighbors': 5}
```

7) SVC

(SVC(),[{'C':[0.25,0.5,0.75,1],'kernel':['linear', 'rbf'],'random_state':[0]}])]



```
SVC():
Best Accuracy : 83.32%
Best Parameters :  {'C': 1, 'kernel': 'rbf', 'random_state': 0}
```

Quality of the data:

Data quality is critical because it is the core concept of the project. As a result, the quality of the data corresponds to the quality of the output. We replaced anomalies in the entire dataset, such as null values, to maintain quality.

Accuracy:

Accuracy has been one of our key performance metrics because it is used to determine which model is significantly better than others, as well as to identify any hidden relationships and patterns related to the training data provided between variables in a dataset.

In our approach we used seven classifiers to perform predictions using our dataset

The highest score achieved was with Gradient Boosting Classifier with an accuracy of 84% and an AUC score of 0.91.

Consistency:

Data in the dataset should always be consistent by each measurement of variables or columns. And consistency is critical because we are cleaning, organizing, and filtering raw data, which should not result in the loss of any of the datasets contents, thus preserving their originality.

The value obtained from the data earlier and later any operations should not differ, i.e. there should be no data loss after performing any operations upon that dataset.

The data should always be constant in order to be used in various ways without changing or removing its existing form.

<u>Tools:</u>

The libraries we used were as follows:

PANDAS:

- Pandas is a Python library for dealing with large data collections. It has a wide range of applications.
- Pandas is a data cleaning, exploration, and manipulation tool.
- We got raw data in the form of comma separated values (CSV), and pandas was our first pick because our dataset was not too large (less than 10 GB). On the same screen, we could see the processing and output of any operations performed on the data.

MATPLOTLIB:

- It's a two-dimensional plotting package written in Python.
- For data sets, we can generate maps, bar plots, histograms, and pie charts, among other graphics.
- It is one of the most sophisticated visualization packages available in Python.
- Our main goal was to get relevant information out of the data.
- Matplotlib was helpful.

SCIKIT-LEARN:

- One of the most useful libraries in Python is Scikit-learn. It includes many statistical modelling and machine learning algorithms such as classification, regression, and clustering, among others.
- We used a total of 7 algorithms, they are:
    - Logistic Regression
    - Decision Tree Classifier
    - Random Forest Classifier
    - AdaBoost Classifier
    - Gradient Boosting Classifier

- K-Neighbor Classifier
- Support Vector Classifier

Lessons Learned:

- We started by removing null values from two of the columns, "CREDIT SCORE" and "ANNUAL MILEAGE," and replacing them with the mean and mode, respectively.
- We then created graphs to show how the categories in the categorical data were distributed, as well as graphs for individual columns and columns in relation to the "CLAIM" column.
- The label was "CLAIM," and the remainder of the columns were features. We then looked for data imbalance and discovered that the training set was uneven, therefore we oversampled the minority class to balance the data.
- Applied Feature Scaling to rescale the features such that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one.
- Performed HyperParameter Tuning to find the best fit parameters for the ML models we used in order to achieve the best accuracy.
- Finally, for each of the ML models, we showed the ROC to characterize the trade-off between the true positive rate and the false positive rate.
- The following table displays the outcomes:

Models	**Accuracy**	**AUC**
Logistic Regression	0.82	0.89
Decision Tree Classifier	0.77	0.75
Random Forest Classifier	0.82	0.89
Ada Boost Classifier	0.83	0.90
Gradient Boosting Classifier	0.84	0.91
KNeighbors Classifier	0.78	0.84
SVC	0.82	0.87