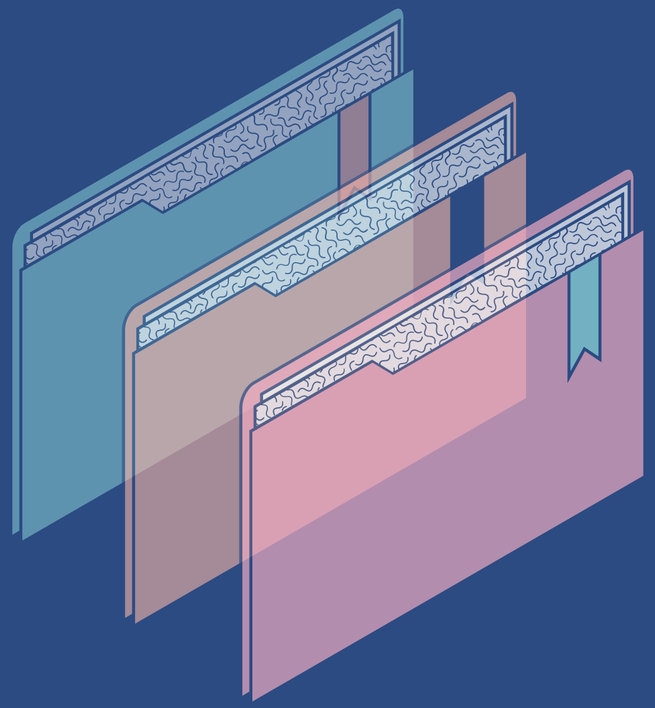
Fraud detection in auto insurance claims

PROBLEM STATEMENT

To predict the fraud in auto insurance claims based on the demographic, policy, claim, and vehicle related features provided in the datasets and also generate the top 20 patterns for fraud on target attribute.



Data Provided

- 1) Demographics Data: These files consist of the demographic data of each customer
- 2) Policy Information: These files consist of the customer's auto insurance policy information, connected to the claim with the insurance company.
- 3) Data of Claim: These files consist of the details about the insurance claim, that the customer applied.
- 4) Data of Vehicle: These files consist of the details about the Vehicle.
- 5) Fraud Data: This Train.csv table contains the Fraud information details, like CustomerID, and ReportedFraud.

Evaluation metrics used

WHY USE THESE AS A METRIC?

1) Just classifying a given customer's details into fraud and Not fraud and blocking a genuine transaction for a long time would bring a bad impact on customer experience
2) Rather I consider providing a probability score for each class and based on that if there is a higher probability of the transaction being fraud we might send that details to the insurance company to have more checks instead of checking every customer details.

- ➤ Log-Loss
- ➤ Confusion Metrics
- ➤ F1-score / F1 statistic

Approach Used For Solution

1 STEP

Data Understanding and Merging

The data provided in different CSV files are understood and merged

2 STEP

Data Cleaning

Raw Data provided is null value treated, duplicate value removed, and so on.

3 STEP

Data Visualization

Univariate, Bivariate, and multivariate analysis (3D,pair-plot, PCA, T-SNE) are done

4 STEP

Best Model Selection, Building and Tunning

Choosing a robust Model based on the evaluation metrics and hyperparameter tunning the best model and performing the training

5 STEP

Predicting on test data

Finally the model is used to predict on the test dataset

Data Cleaning

For Null values treatment in the Demographics and policy claim datasets, I have used Model-based imputation and for some columns, I have implemented mean median and mode-based imputation.

For Demographics Data

Training a simple model for missing value imputation

In [29]:
```
k = list(range(1,30,2))

knn_acc = []

for i in k:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(x_train,y_train)
    knn_acc.append(neigh.score(x_test, y_test))
```

In [30]:
```
print("The optimal value for k is",k[np.argmax(np.array(knn_acc))],"with accuracy score",np.max(np.array(knn_acc)))
```

```
The optimal value for k is 1 with accuracy score 0.7439193884642112
```

In [31]:
```
# Training with the optimal value for k
neigh = KNeighborsClassifier(n_neighbors=k[np.argmax(np.array(knn_acc))])
neigh.fit(x_train,y_train)
```

Out[31]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [32]:
```
joblib.dump(neigh, 'Demographic.joblib')
```

Out[32]:
```
['Demographic.joblib']
```

to predict

In [33]:
```
# changes for the dataframe with null values
df3.drop(["InsuredGender","CustomerID"],axis=1,inplace=True)
```

For Policy Claim Data

In [27]:
```
print("The optimal value for k is",k[np.argmax(np.array(lr_acc))],"with
```

```
The optimal value for k is 0.1 with accuracy score 0.5389009793253536
```

In [28]:
```
# Training KNN with optimal K=1 on entire dataset
neigh = KNeighborsClassifier(n_neighbors=1)
neigh.fit(x,y)
```

Out[28]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [29]:
```
joblib.dump(neigh, 'policyClaim.joblib')
```

Out[29]:
```
['policyClaim.joblib']
```

predict for null values

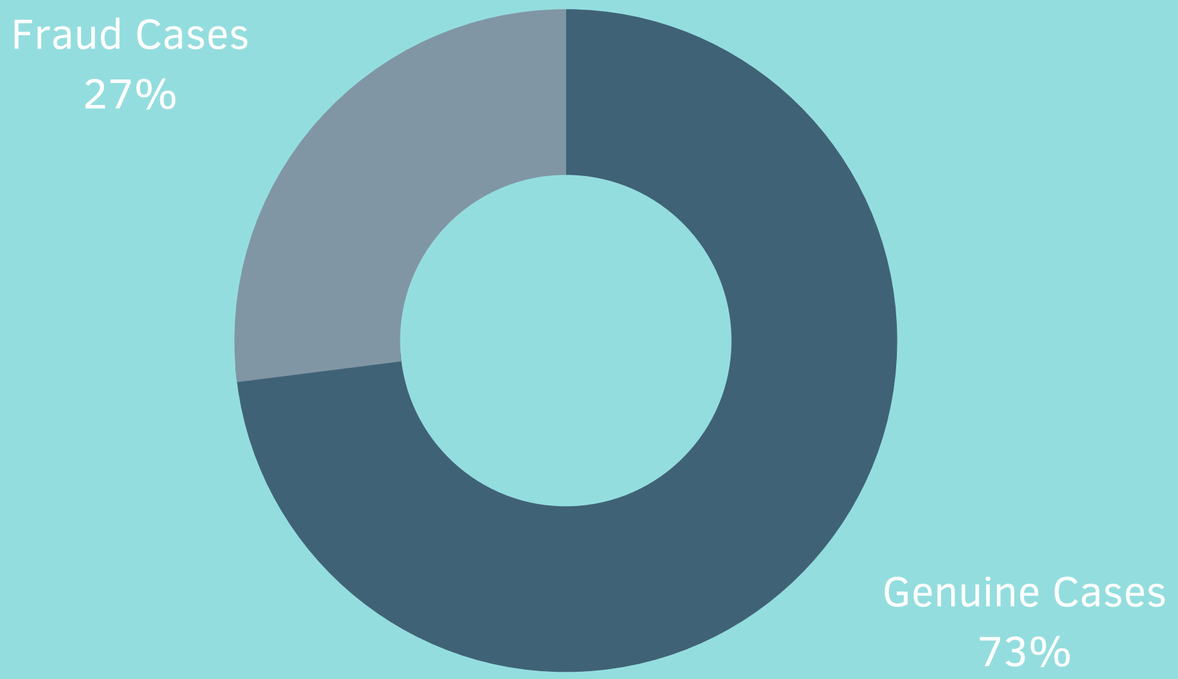
In [30]:
```
pred = neigh.predict(df3)
pred
```

Out[30]:
```
array([1, 1, 0, ..., 1, 1, 1], dtype=int8)
```

Exploratory Data Analysis

1) Percentage of the fraud and Genuine

The dataset contains 73.0 percentage of Genuine cases
The dataset contains 27.0 percent of Fraud cases

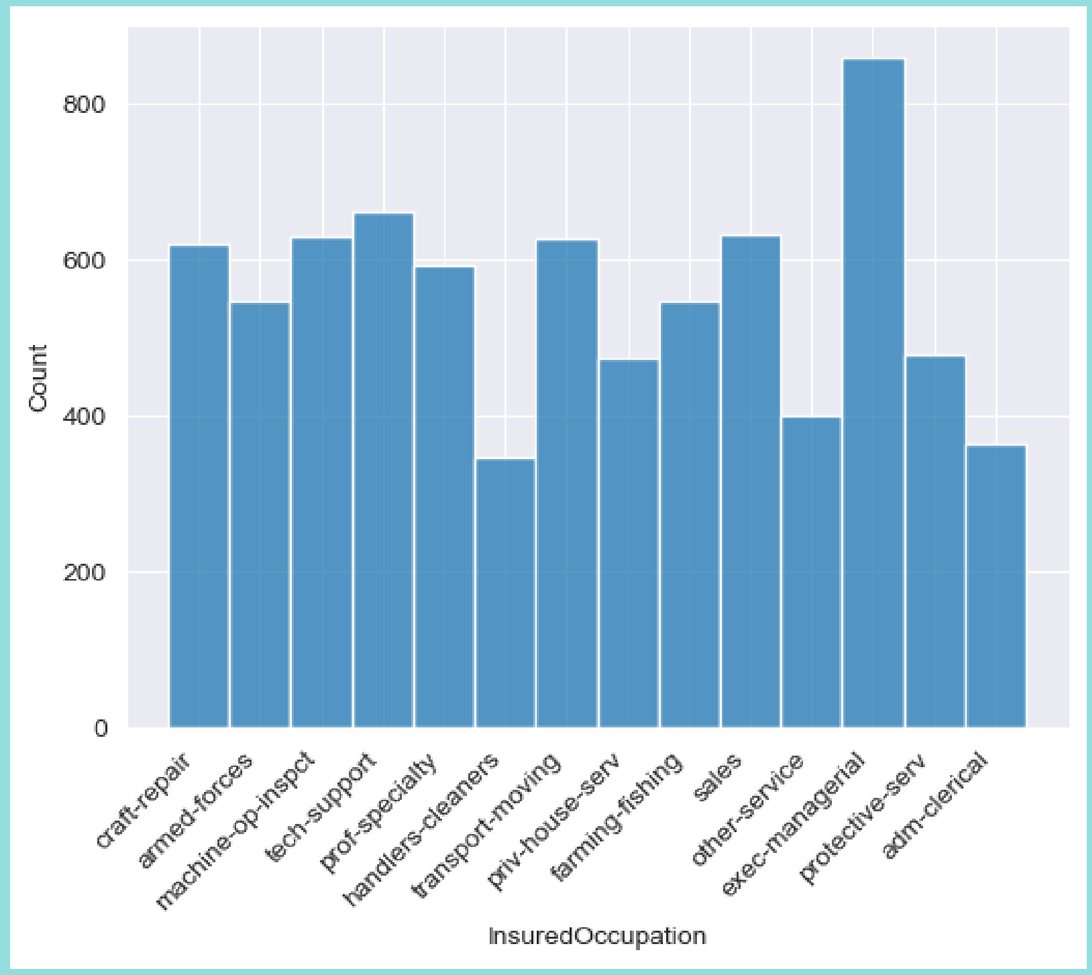


2) Top 10 zipcode from where most fraud has been noted

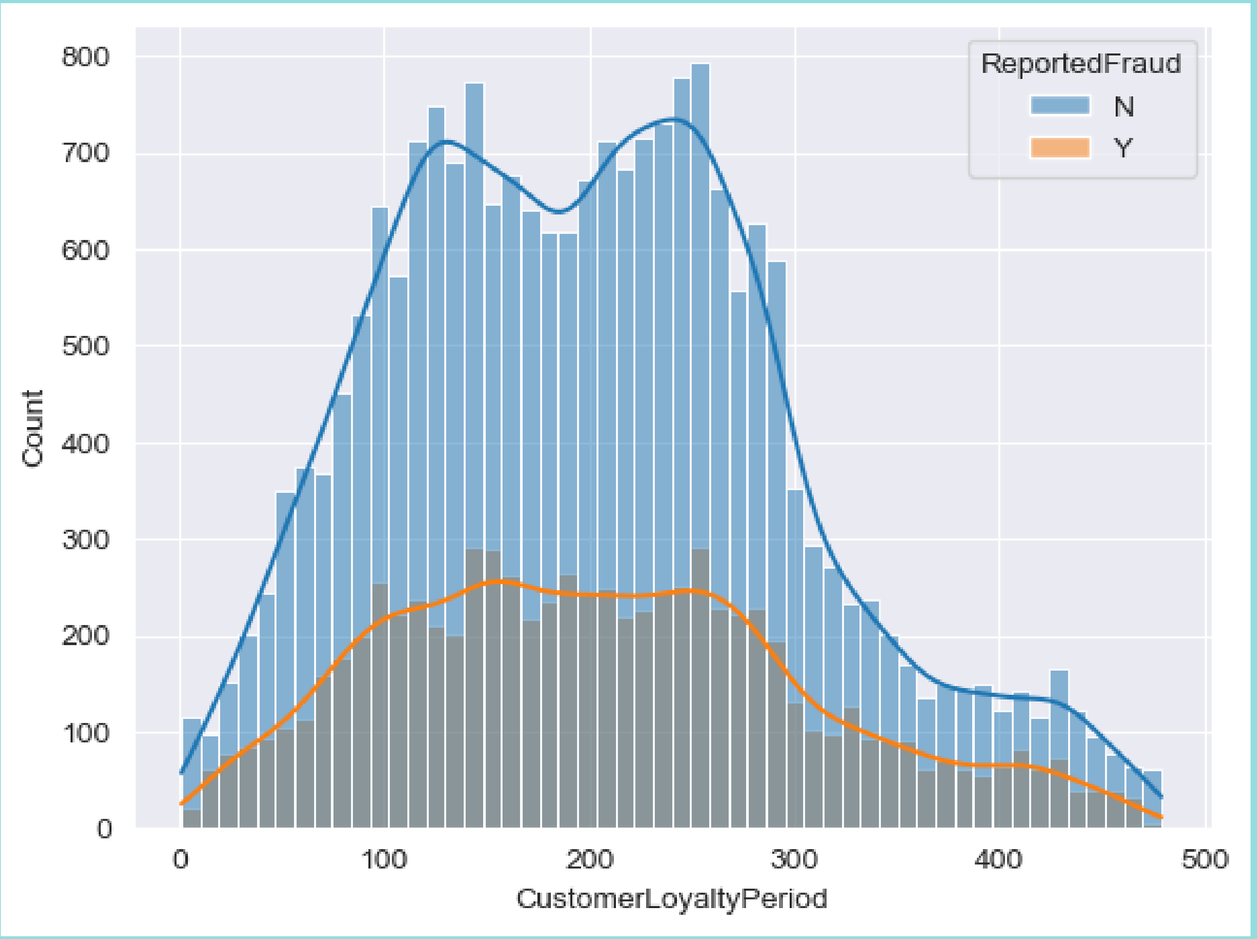
Number of unique zip codes : 995.

476198 68
608331 67
478456 58
606352 55
438529 54
474771 54
469429 54
474801 52
603882 50
613607 50

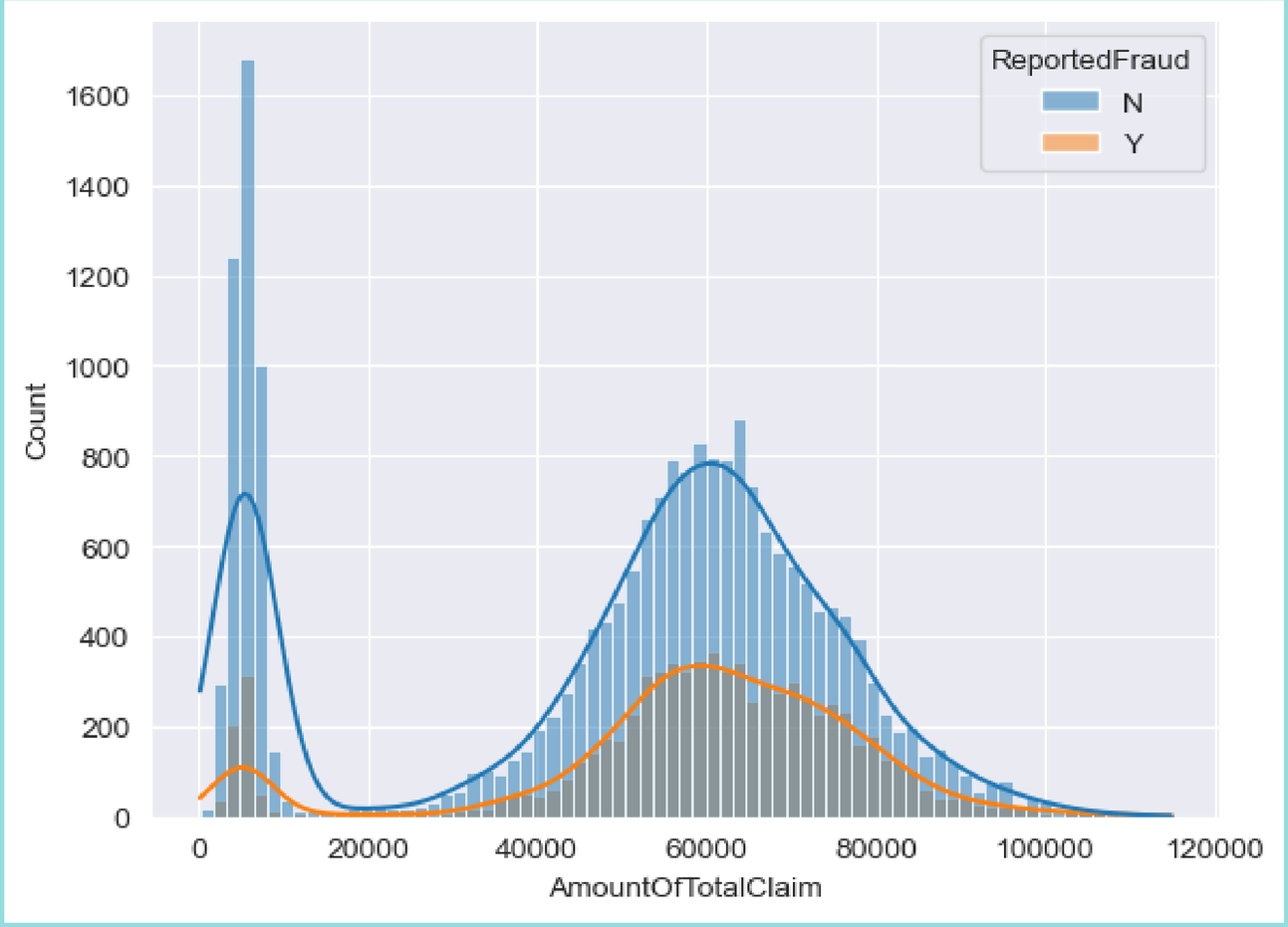
3) Occupations that cases classified as Fraud cases are registered



4) Check for a PDF of customer loyalty period for both Fraud and Not Fraud



5) Check for a PDF of Amountoftotalclaim for both Fraud and Not Fraud



6) Categories Percentage registered as Fraudulent cases in SeverityOfIncident

Over 59.00% of fraudulent cases are from Major Damage
Over 19.72% of fraudulent cases are from Minor Damage

7) Categories Percentage registered as Fraudulent cases in IncidentState

Over 30.61% of Fradulent cases are from State7
Over 24.17% of Fradulent cases are from State5
Over 15.77% of Fradulent cases are from State9
Over 13.04% of Fradulent cases are from State4

8) Categories Percentage registered as Fraudulent cases in PropertyDamage

Over 50.33% of Fradulent cases are from NO

Over 49.67% of Fradulent cases are from YES

9) Categories Percentage registered as Fraudulent cases in PoliceReport

Over 34.75% of Fraudulent cases are from NoReport

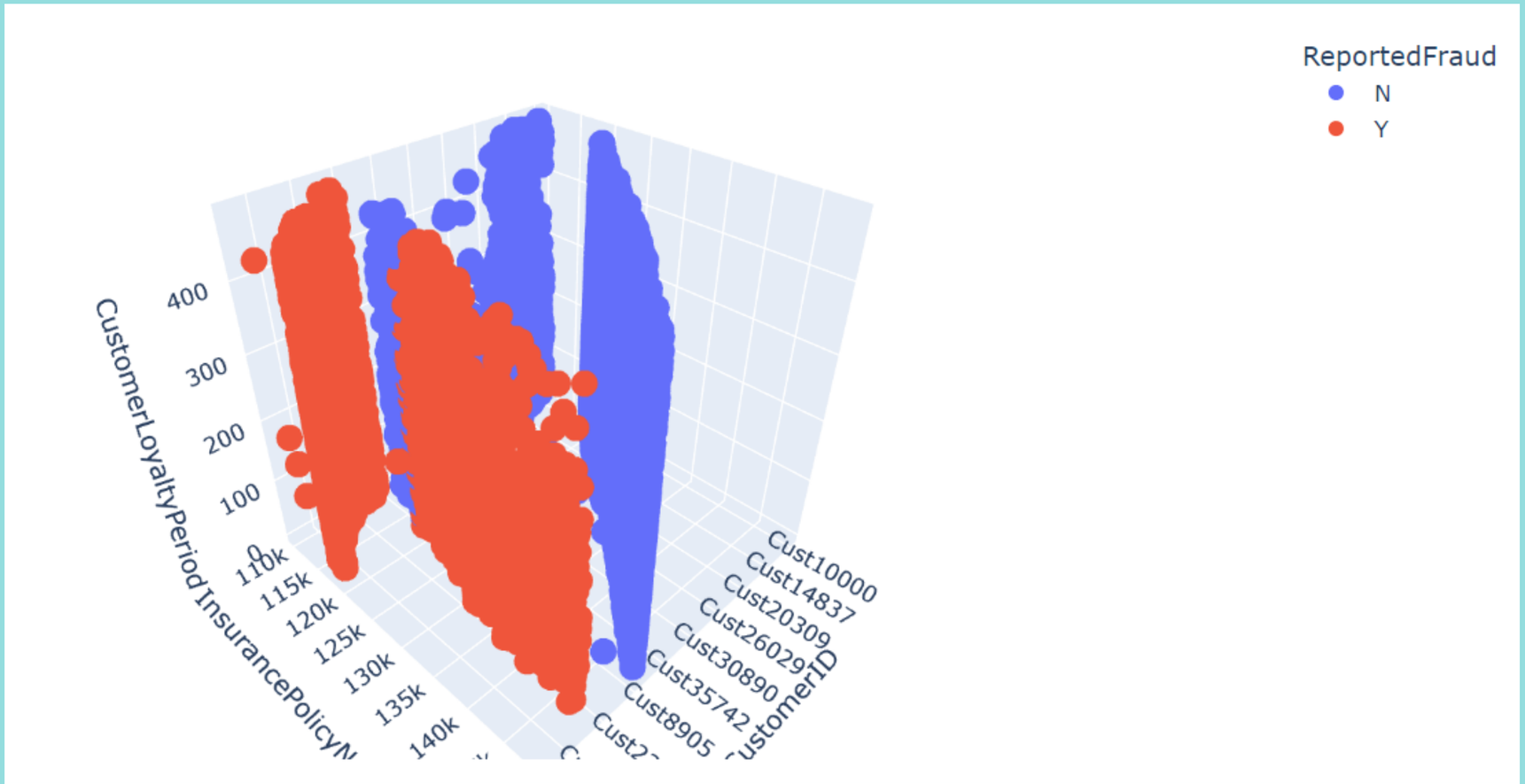
Over 34.27% of Fraudulent cases are from NO

Over 30.98% of Fraudulent cases are from YES

Bi-Variate Analysis

Pair-plot between "Policy_Deductible", "Witnesses", "AmountOfTotalClaim", "AmountOfInjury" and Reported Fraud as HUE.

3d_Scatter plot between "InsurancePolicyNumber", "CustomerLoyaltyPeriod", "CustomerID" and Reported Fraud as HUE.

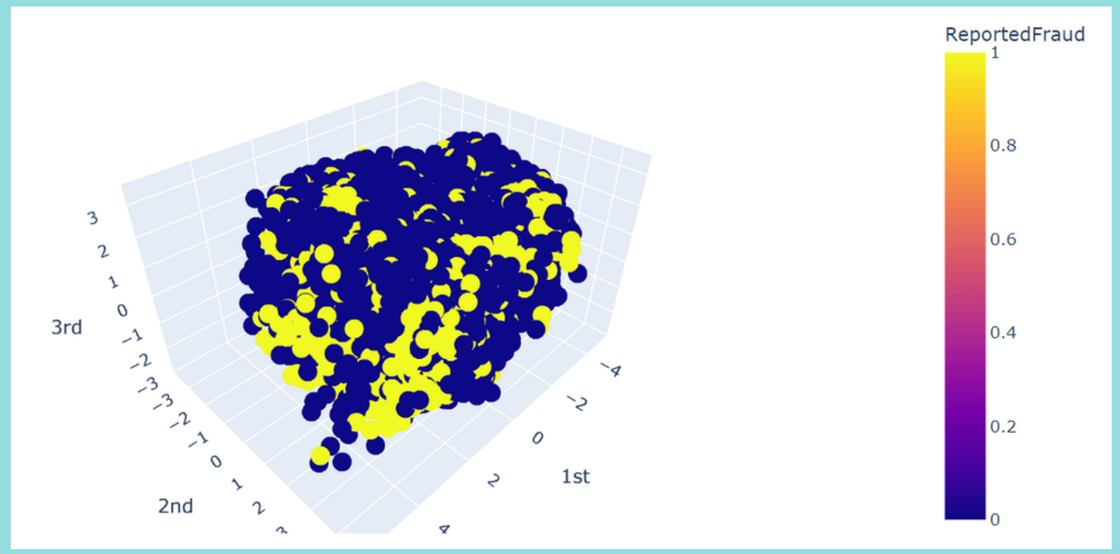


PCA for visualizing and Analysis

So the total variance explained here by the top three reduced features is 7.94 % only

2D-Plot

3D-Plot



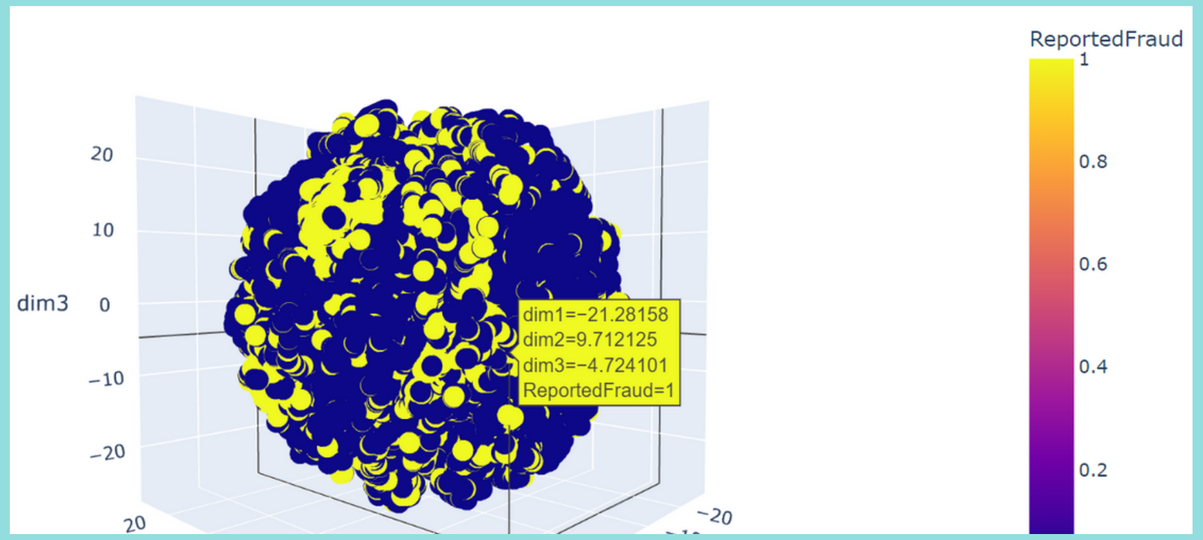
Since there was only 7.94% of variance explained from the three features of PCA we can't expect any differentiator for Fraud and Not Fraud cases.

T-SNE for visualizing and Analysis

Running t-sne with multiple perplexity and iteration values

2D-Plot (Final plot after running for multiple values for hyperparameter)

3D-Plot



Conclusion: Maybe the the data is distributed randomly in the higher domensional space.

Search for the Best Model

To choose which model to be used for training and predicting on the Test dataset. I have built multiple models and hyperparameter stunned the models.

Naive Bayes

```
# Applying grid search to hyperparameter tune the BernoulliNB model

#Instance for Naive Bayes classifier
nb = BernoulliNB()

#params for Bernoulli Naive bayes model
nb_params = [{'alpha':[10**x for x in range(-5,5)], 'fit_prior':[True,False]}]

#Instance for Naive Bayes Classifier and f1 metric
f1score = make_scorer(f1_score,greater_is_better=True, needs_proba=False)
nbclf = GridSearchCV(nb, nb_params, cv=10, scoring=f1score)

#training the model to find the best parameter
nbclf.fit(x_train, y_train)
```

```
GridSearchCV(cv=10, estimator=BernoulliNB(),
             param_grid=[{'alpha': [1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100,
                                    1000, 10000],
                          'fit_prior': [True, False]}],
             scoring=make_scorer(f1_score))
```

```
print("The best parameter after GridSearch comes out to be: ",nbclf.best_params_)
print("With the F1 statistic of: ",nbclf.best_score_)
```

```
The best parameter after GridSearch comes out to be:  {'alpha': 1, 'fit_prior': True}
With the F1 statistic of:  0.6626685857926666
```

KNN

```
# min max sclaer
scaler = MinMaxScaler()
scaler.fit(x_train)
x_trainknn = scaler.transform(x_train)
x_testknn = scaler.transform(x_test)
```

```
# Applying grid search to hyperparameter tune the KNN model

#Instance for KNN model
knn = KNeighborsClassifier()

#Hyperparameter for search
knn_params = [{'n_neighbors':list(range(1,20,2)), 'weights':['uniform','distance']}]

#Instance for Grid Search CV
f1score = make_scorer(f1_score, greater_is_better=True, needs_proba=False)
clf = GridSearchCV(knn, knn_params, cv=10, scoring=f1score)

#training the model to find the best parameter
clf.fit(x_trainknn, y_train)
```

```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19],
                          'weights': ['uniform', 'distance']}],
             scoring=make_scorer(f1_score))
```

```
print("The best parameter after GridSearch comes out to be: ",clf.best_params_)
print("With the F1 statistic of: ",clf.best_score_)
```

```
The best parameter after GridSearch comes out to be:  {'n_neighbors': 7, 'weights': 'uniform'}
With the F1 statistic of:  0.8702983984872918
```

LogisticRegression

```
#Creating Instance for StandardScaler
scaler = StandardScaler()

scaler.fit(x_train)

x_trainlr = scaler.transform(x_train)
```

```
#Hyper-parameter tunning for logistic regression

#Instance for the LogisticRegression
lr = LogisticRegression(n_jobs=-1)

#params for search
lr_params = [{'penalty':['l1','l2','elasticnet'],'C':[10**x for x in range(-3,3)],'max_iter':[100,150,200]}]

#Instance for Grid Search CV
f1score = make_scorer(f1_score, greater_is_better=True, needs_proba=False)
lrclf = GridSearchCV(lr, lr_params, cv=10, scoring=f1score)

#training the model to find the best parameter
lrclf.fit(x_trainlr, y_train)
```

```
GridSearchCV(cv=10, estimator=LogisticRegression(n_jobs=-1),
             param_grid=[{'C': [0.001, 0.01, 0.1, 1, 10, 100],
                          'max_iter': [100, 150, 200],
                          'penalty': ['l1', 'l2', 'elasticnet']}],
             scoring=make_scorer(f1_score))
```

```
print("The best parameter after GridSearch comes out to be: ",lrclf.best_params_)
print("With the F1 statistic of: ",lrclf.best_score_)
```

```
The best parameter after GridSearch comes out to be:  {'C': 10, 'max_iter': 100, 'penalty': 'l2'}
With the F1 statistic of:  0.6696432931568516
```

DecisionTreeClassifier

```
#Hyper-parameter tunning for DecisionTreeClassifier

#Instance for the DecisionTreeClassifier
dt = DecisionTreeClassifier()

#params for search
dt_params = [{'criterion':['gini','entropy','log_loss'],'max_depth':[x for x in range(2,30)],'min_samples_split':[2,3,4,5]}]

#Instance for Grid Search CV
f1score = make_scorer(f1_score, greater_is_better=True, needs_proba=False)
dtclf = GridSearchCV(dt, dt_params, cv=10, scoring=f1score)

#training the model to find the best parameter
dtclf.fit(x_train, y_train)
```

```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid=[{'criterion': ['gini', 'entropy', 'log_loss'],
                          'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
                                        14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
                                        24, 25, 26, 27, 28, 29],
                          'min_samples_split': [2, 3, 4, 5]}],
             scoring=make_scorer(f1_score))
```

```
print("The best parameter after GridSearch comes out to be: ",dtclf.best_params_)
print("With the F1 statistic of: ",dtclf.best_score_)
```

```
The best parameter after GridSearch comes out to be:  {'criterion': 'gini', 'max_depth': 15, 'min_samples_split': 2}
With the F1 statistic of:  0.7411575027932199
```

Random Forest Classifier

```
#Hyper-parameter tunning for RandomForestClassifier

#Instance for the RandomForestClassifier
rf = RandomForestClassifier(n_jobs=-1)

#params for search
rf_params = [{'criterion':['gini','entropy'],'n_estimators':[x for x in range(100,1000,50)],'min_samples_split':[2,3,4]}]

#Instance for Grid Search CV
f1score = make_scorer(f1_score, greater_is_better=True, needs_proba=False)
rfclf = GridSearchCV(rf, rf_params, cv=10, scoring=f1score)

#training the model to find the best parameter
rfclf.fit(x_train, y_train)
```

```
GridSearchCV(cv=10, estimator=RandomForestClassifier(n_jobs=-1),
             param_grid=[{'criterion': ['gini', 'entropy'],
                          'min_samples_split': [2, 3, 4],
                          'n_estimators': [100, 150, 200, 250, 300, 350, 400,
                                           450, 500, 550, 600, 650, 700, 750,
                                           800, 850, 900, 950]}],
             scoring=make_scorer(f1_score))
```

```
print("The best parameter after GridSearch comes out to be: ",rfclf.best_params_)
print("With the F1 statistic of: ",rfclf.best_score_)
```

```
The best parameter after GridSearch comes out to be:  {'criterion': 'gini', 'min_samples_split': 3, 'n_estimators': 600}
With the F1 statistic of:  0.8475012117567303
```

Search for the Best Model

A random model was made as a Benchmark model to evaluate other models.
All the models were hyperparameter stunned and then decided to choose the best model based on the metric.

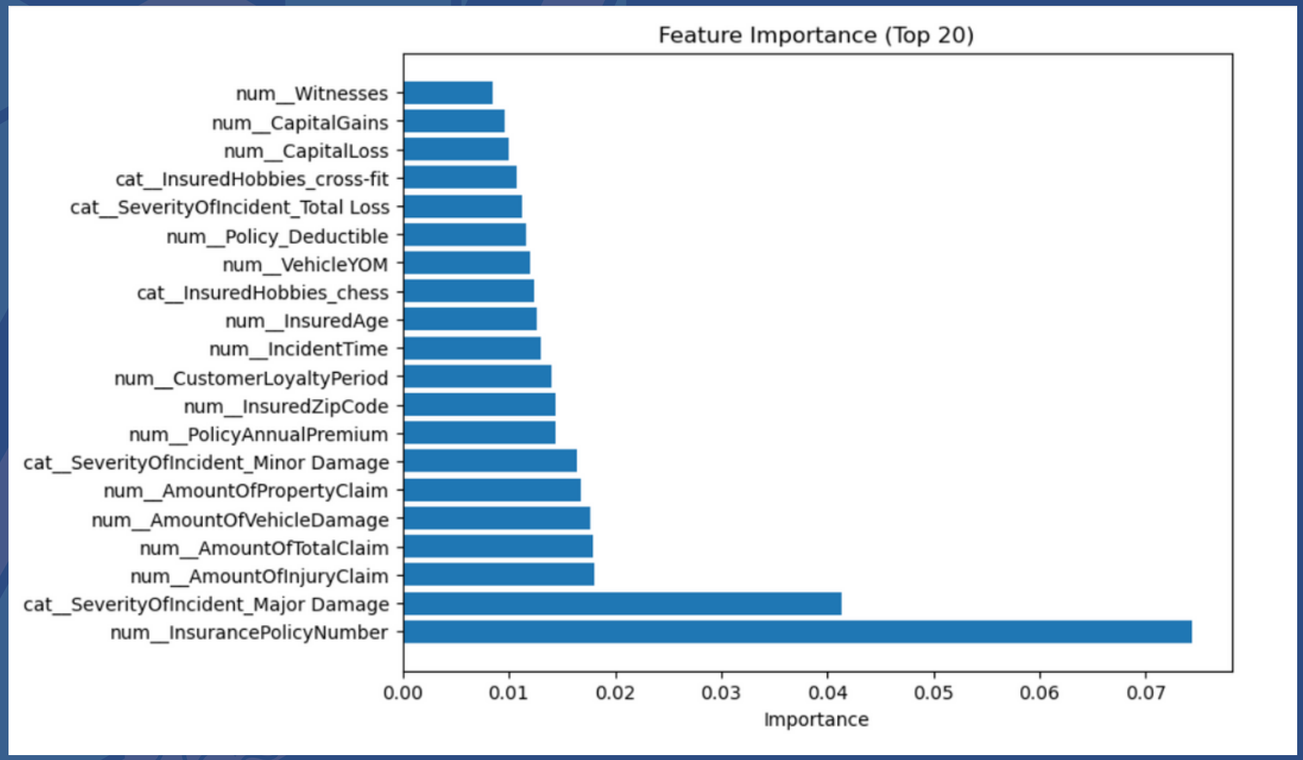
	Models	F1-Score
0	RandomModel	0.340100
1	Naive Bayes	0.662669
2	KNN	0.870298
3	Logistic Regression	0.669643
4	Decision Tree Classifier	0.741158
5	RandomForest Classifier	0.847501

Conclusion: So I am using the RandomForest classifier as my model to predict the unseen data. Since it has the highest f1-score and also has higher log-loss i.e...able to classify cases with higher probability.

Training On Best Model (RF)

After performing the training on the Cleaned and preprocessed data, below are the results from RANDOM FOREST, then from this trained model got the predictions for the Test Data with a Hackathon score of 0.81 f1-score.

```
In [13]: # train data logloss and f1-score
         print("The Log-Loss for a Random Model is: ",log_loss(y_train,train_pred_prob))

         print("The F1-Score for a Random Model is: ",f1_score(y_train,train_pred))

The Log-Loss for a Random Model is:  0.050221244027252906
The F1-Score for a Random Model is:  1.0

In [14]: print(classification_report(y_train,train_pred))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     16813
           1       1.00      1.00      1.00     16868

    accuracy                           1.00     33681
   macro avg       1.00      1.00      1.00     33681
weighted avg       1.00      1.00      1.00     33681

In [15]: # train data logloss and f1-score
         print("The Log-Loss for a Random Model is: ",log_loss(y_test,test_pred_prob))

         print("The F1-Score for a Random Model is: ",f1_score(y_test,test_pred))

The Log-Loss for a Random Model is:  0.1626158713109998
The F1-Score for a Random Model is:  0.979937903033198

In [16]: print(classification_report(y_test,test_pred))

              precision    recall  f1-score   support

           0       0.98      0.98      0.98      4238
           1       0.98      0.98      0.98      4183

    accuracy                           0.98      8421
   macro avg       0.98      0.98      0.98      8421
weighted avg       0.98      0.98      0.98      8421
```



Thank you for your time :)