DTSC 5502 - Principles and Techniques for Data Science

Automated Vehicle Insurance Fraud Detection with Supervised Machine Learning

Group – 2

Sai Kumar Akkamshetty – 11702531

Uma Abhishek Polikanda – 11704249

Aakash Reddy Varala – 11637262

Hanumantha Reddy Varikuti – 11610681

Project Title: "Automated Vehicle Insurance Fraud Detection with Supervised Machine Learning."

Participants & Roles:

Sai Kumar Akkamshetty – SaiKumarAkkamshetty@my.unt.edu

Role: Code Development, Dataset Selection, Data Processing and Preparation

Uma Abhishek Polikanda – UmaAbhishekPolakonda@my.unt.edu

Role: Code Development, Feature Extraction, Model Prediction and Evaluation

Aakash Reddy Varala – AakashReddyVarala@my.unt.edu

Role: Code Development, Model Prediction and Evaluation, Data Analysis

Hanumantha Reddy Varikuti – HanumanthaReddyVarikuti@my.unt.edu

Role: Project Documentation, Dataset Selection, References and Related Projects

Workflow:

Our project collaboration has been fostered through regular in-person Wednesday meetings and additional online sessions via Microsoft Teams. We plan to enhance our code development and project documentation using Jupyter notebooks. The workflow is organized into specific phases, commencing with Data Pre-processing and Exploration, followed by Model Development and Training. Next, we transit into the Testing and Evaluation phase, succeeded by Performance Analysis. The concluding steps involve thorough Documentation and the generation of a comprehensive Final Report. This structured methodology ensures a systematic and efficient progression through each project stage.

Abstract:

The main goal of our project is to build an automated fraud detection model that can identify fraudulent vehicle insurance claims. With the increasing complexity of insurance claims, it has become critical to detect fraudulent activities efficiently. Our model will explore previous auto insurance claims data, extracting similar features and patterns that point to potential fraud. By using advanced machine learning algorithms, we want to create a predictive model that can detect fake insurance claims in real time.

We will use a real-world dataset sourced from Kaggle, involving structured insurance claims data. Our project will use supervised Learning methods, with an initial focus on Random Forest, to classify claims as either genuine or fraudulent. This project will analyze insurance claims data to identify unusual patterns and mark possible fraud cases. By automating this project insurance companies can greatly reduce financial losses due to fraudulent claims, improve the overall efficiency in claim processing and help in maintaining the integrity and sustainability of the Auto Insurance firms.

This project not only helps to overcome the financial losses but also helps in maintaining the integrity and sustainability of the Vehicle Insurance firms.

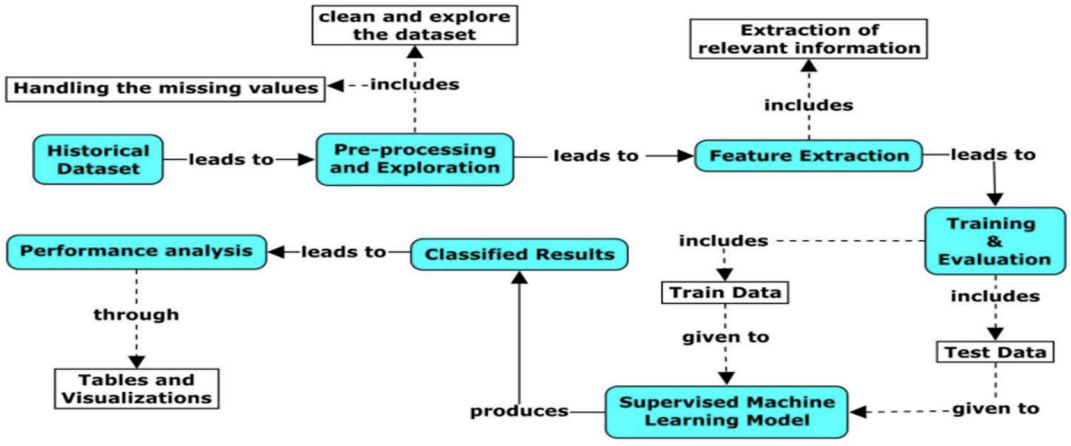
Data Specification:

The dataset (carclaims.csv) contains 33 features with 15,420 samples that are used to predict the target variable (fraud found). The data set contains 923 fraud detections/cases. These features are utilized in training machine learning models that include month, week of month, day of week, make, accident area, day of week claimed, month claimed, week of month claimed, sex, marital status, age, fault, policy type, vehicle category, vehicle price, policy number, Reg number, deductible, driver rating, days: policy accident, days: policy claim, past number of claims, age of vehicle, age of policy holder, police report file, witness present, age and time, number of supplements, address change claims, number of cars, year, base policy, fraud(Y/N).

Project Design:

Technologies and Design Methods used in our project are as follows:

Jupyter Notebook, Scikit-Learn (Random Forest, Logistic Regression, Decision Tree, KNN Classifier)



A visual representation of our approach to vehicle insurance fraud detection with Machine learning.

To determine if the claim is false or not, we used a variety of supervised machine learning methods.

Random forest:

In essence, Random Forest is used for both classification and regression issues. It is an ensemble classification method and a supervised machine learning classifier. The more trees there are, the more precise the outcome would be. The RF machine learning method is simple, easy to use, and capable of achieving outstanding outcomes in most cases without the need for hyper tuning. Over-fitting is one of the decision tree algorithm's main problems. The decision tree appears to have remembered the data. Random

Forest is utilized to prevent this and is an illustration of ensemble learning in action. The use of several repetitions of one or more algorithms is referred to as "ensemble learning." A "random forest" is a collection of decision trees.

Decision tree:

The decision tree approach is also included in the supervised learning category. Regression and classification problems can be solved with DT. However, it is used in this work to overcome categorization problems. DT breaks down the input into ever-smaller bits in an attempt to solve the problem, which results in the prediction of a target value (diagnosis). A decision tree (DT) consists of decision nodes and leaf nodes, each of which is linked to a class label and traits that are shown on the interior node of the tree.

Logistic regression:

It is a condensed version of "linear regression," a potent tool for visualizing data. The likelihood of an illness or other health concern because of a plausible cause is ascertained using logistic regression. The link between independent factors (X), also known as exposures or predictors, and a binary dependent (target) variable (Y), also known as the outcome or response variable, is examined using both basic and multivariate logistic regressions. It is often applied to forecast changes in the dependent variable that will be binary or multiclass.

KNN:

K-Nearest Neighbor is one of the most straightforward machine learning algorithms, based on the supervised learning approach (KNN). The model is saved, and when a new data point is given, the model searches for similarities between the data points. K-nearest neighbors are identified by calculating the distance between each data point with respect to the new data point, and based on the similarity, it produces the output. This method is also referred to as lazy learning. This indicates that new data can be reliably and quickly categorized using the K-NN approach.

Model Evaluation:

The Receiver Operating Characteristic suggests that while the model is accurate in terms of correctly predicting the majority class, it's not doing well in predicting the minority class. The F1 Score and Recall are both measures of a model's performance in terms of handling both classes in a balanced way. A low score indicates that the model is not predicting the minority class correctly.

This could be due to an imbalance in the classes in the data. We may consider using techniques like Up sampling the minority class or Down sampling the majority class to handle the class imbalance.

- Up sampling is a technique used to address class imbalance by increasing the number of minority class samples.
- Down sampling is a technique used to address class imbalance by decreasing the number of majority class samples.

After performing sampling, the Upsampled Random Forest Classifier model has an accuracy of 98.66%. The F1 Score is 0.9868, Precision is 0.9739, and Recall is 1.0. The ROC AUC score is 0.9866. These scores suggest that the model's performance is excellent. The accuracy, F1 Score, Precision, Recall, and ROC

AUC are all very high, indicating that the model's ability to correctly and classify instances is much better than random chance.

The choice of Logistic Regression, Random Forest Classifier, KNN Classifier, and Decision Tree models for our Vehicle Insurance Fraud Detection project is due to their versatility in handling both numerical and categorical data, which is common in insurance fraud detection. These models are also interpretable, especially Logistic Regression and Decision Trees, providing insights into feature importance. Random Forest Classifier and KNN Classifier are known for their high performance on various classification problems, including fraud detection.

Project Milestones:

We performed the following tasks important tasks in our project:

- Data Pre-processing and Exploration.
- Model development and Training.
- Testing and Evaluation.
- Performance Analysis.
- Documentation and Final Report.

We have included the evaluation metrics such as F-1 score, Recall score, Precision and ROC curve to analyze in detail characteristics of the model which we have taken as our reference (mentioned in the reference page). We also included two other classifier models such as KNN classifier and Decision Tree models to find out the best model that gives accurate results.

Results:

The evaluation results for the proposed machine learning methods are concluded as below:

	Model	Accuracy Score	F1_Score	Precision	Recall_Score
0	UpSampled Logistic Regression	0.729480	0.752430	0.693507	0.822296
1	DownSampled Logistic Regression	0.738095	0.760396	0.700730	0.831169
2	UpSampled RandomForest Classifier	0.987860	0.988004	0.976293	1.000000
3	DownSampled RandomForest Classifier	0.759740	0.785300	0.709790	0.878788
4	UpSampled KNeighborsClassifier	0.877086	0.890527	0.802658	1.000000
5	DownSampled KNeighborsClassifier	0.642857	0.651163	0.636364	0.666667
6	UpSampled DecisionTreeClassifier	0.967720	0.968725	0.939347	1.000000
7	DownSampled DecisionTreeClassifier	0.709957	0.708696	0.711790	0.705628

Conclusion:

As the different countries around the world evolve into a more economical based one, stimulating their economy is the goal. To fight these fraudsters and money launderers was quite a complex task before the era of machine learning but thanks to machine learning and AI we can fight these kinds of attacks. The proposed solution can be used in insurance companies to find out if a certain insurance claim made is a fraud or not. The model was designed after testing multiple algorithms to come up with the best model that will detect if a claim is fraudulent or not. This is aimed at the insurance companies as a pitch to come up with a more tailored model for their liking to their own systems. The model should be simple enough to calculate big datasets, yet complex enough to have a decent successful percentile.

Future Developments:

1. **Advanced Sampling Techniques**: While we used up sampling to handle class imbalance, there are other advanced techniques like SMOTE or ADASYN that you could explore.
2. **Feature Engineering**: Creating new features based on existing ones can often improve model performance. For example, creating interaction features, binning variables, or creating polynomial features.
3. **Ensemble Methods**: Combining predictions from multiple models can often yield better results than any individual model. Techniques like bagging, boosting, or stacking could be used.
4. **Deep Learning Models**: If you have a large amount of data, deep learning models can sometimes outperform traditional machine learning models.

Repository Link: https://github.com/Hanumanthareddy12/Automated-Vehicle-Insurance-Fraud-Detection

Reference:

[1] Viaene, S., Ayuso, M., Guillen, M., Van Gheel, D., & Dedene, G. (2007). Strategies for detecting fraudulent claims in the automobile insurance industry. European Journal of Operational Research, 176(1), 565-583. https://doi.org/10.1016/j.ejor.2005.08.005

[2] Kini, R. Chelluru, K. Naik, D. Naik, S. Aswale and P. Shetgaonkar, "Automobile Insurance Fraud Detection: An Overview," 2022 3rd International Conference on Intelligent Engineering and Management(ICIEM),London,UnitedKingdom,2022,pp.7-12,doi: 10.1109/ICIEM54221.2022.9853043.

[3] Majhi, S. K., Bhatachharya, S., Pradhan, R., & Biswal, S. (2019). Fuzzy clustering using salp swarm algorithm for automobile insurance fraud detection. Journal of Intelligent & Fuzzy Systems, 36(3), 2333-2344. https://doi.org/10.3233/JIFS-169944

[4] Nabrawi, E., & Alanazi, A. (2023). Fraud detection in healthcare insurance claims using machine learning. Risks (Basel), 11(9), 160. https://doi.org/10.3390/risks11090160

[5] Amr, Tarek. *Hands-On Machine Learning with scikit-learn and Scientific Python Toolkits: A practical guide to implementing supervised and unsupervised machine learning algorithms in Python*. Packt Publishing Ltd, 2020.

[6] PACKT, P. (Producer), & . (2018). Hands-on Supervised Machine Learning with Python. [Video/DVD] PACKT Publishing. https://video.alexanderstreet.com/watch/hands-on-supervised-machine-learning-with-python

[7] Rashmi. (2022). Vehicle Insurance Fraud Detection. Kaggle. Retrieved from Kaggle

Appendix

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, f1_score,
precision_score, recall_score, roc_curve, accuracy_score,auc
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.utils import resample
import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv("carclaims.csv")

print(df.head())

print("Number of Features Available:",df.shape[1])
print("Number of Samples Available :",df.shape[0])

df.isnull().sum()


plt.figure(figsize=(10,8))
bars = plt.bar(df.FraudFound.value_counts().index,
df.FraudFound.value_counts().values)
plt.title("Fraud Type")
plt.xlabel("Type")
plt.ylabel("Count")

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.05, yval, ha='center',
va='bottom')


plt.show()

# Replacing the variables with 0 and 1

df.loc[df['FraudFound'] == 'No','FraudFound'] = 0
df.loc[df['FraudFound'] == 'Yes','FraudFound'] = 1

df['FraudFound'] = df['FraudFound'].astype(int)

#Plotting the graph separately showing frauds found in each Car Makers

make = df.groupby('Make')['FraudFound'].sum().sort_values(ascending=False)
```

```
plt.figure(figsize=(20,8))
plt.title("Car Make Vs Frauds")


cmap = plt.get_cmap('Blues')
colors = [cmap(i) for i in np.linspace(0, 1, len(make.index))]

ax = sns.barplot(y=make.index, x=make.values, orient='h', palette=colors[::-1])
for i, v in enumerate(make.values):
    ax.text(v + 0.2, i + .25, str(format(int(v), ',d')), color='black',
fontweight='light')

plt.xlabel("Number of Fraud")
plt.ylabel("Car Make")
plt.show()

# Plotting the number of claims found in each Car Makers.

make_count = df['Make'].value_counts().sort_values(ascending=False)
plt.figure(figsize=(20,8))
plt.title("Car Make Count")


cmap = plt.get_cmap('Blues')
colors = [cmap(i) for i in np.linspace(0, 1, len(make_count.index))]

ax = sns.barplot(y=make_count.index, x=make_count.values, orient='h',
palette=colors[::-1])
for i, v in enumerate(make_count.values):
    ax.text(v + 0.2, i + .25, str(format(int(v), ',d')), color='black',
fontweight='light')

plt.ylabel("Car Make")
plt.xlabel("Count of Cars")
plt.show()

# Plotting the number of frauds found in each Policy Holder's Age

policyAge =
df.groupby('AgeOfPolicyHolder')['FraudFound'].sum().sort_values(ascending=True)
plt.figure(figsize=(20,8))
plt.title("Policy Holder's Age Vs Frauds")


cmap = plt.get_cmap('Blues')
colors = [cmap(i) for i in np.linspace(0, 1, len(policyAge.index))]

ax = sns.barplot(x=policyAge.index, y=policyAge.values, palette=colors)
for p in ax.patches:
    ax.annotate(str(format(int(p.get_height()), ',d')), (p.get_x()+0.24,
p.get_height()*1.01))

plt.xlabel("Policy Holder's Age")
plt.ylabel("Number of Fraud")
plt.show()
```

```
#Plotting the graph separately showing frauds found in each gender

gender = df.groupby('Sex')['FraudFound'].sum()
plt.figure(figsize=(10,8))
plt.title("Gender Vs Frauds")

ax = sns.barplot(x=gender.index, y=gender.values)
for p in ax.patches:
    ax.annotate(str(format(int(p.get_height()), ',d')), (p.get_x()+0.24,
p.get_height()*1.01))

plt.xlabel("Gender")
plt.ylabel("Number of Fraud")
plt.show()

#Plotting the graph separately showing frauds found in each Area

accidentArea = df.groupby('AccidentArea')['FraudFound'].sum()

plt.figure(figsize=(10,8))
plt.title("AccidentArea Vs Frauds")

plt.pie(accidentArea.values,labels=accidentArea.index,  autopct='%.0f%%')

plt.show()

#Plotting the pie chart separately showing frauds found in Type of Faults
fault = df.groupby('Fault')['FraudFound'].sum()

plt.figure(figsize=(10,8))
plt.title("Fault Vs Frauds")
plt.pie(fault.values,labels=fault.index,  autopct='%.0f%%')
plt.show()

#Plotting the graph separately showing frauds found in NumberofCars involved
cars = df.groupby('NumberOfCars')['FraudFound'].sum()
plt.figure(figsize=(20,8))
plt.title("Cars Involved Vs Frauds")

ax = sns.barplot(x=cars.index,y=cars.values)
for p in ax.patches:
    ax.annotate(str(format(int(p.get_height()), ',d')), (p.get_x()+0.4,
p.get_height()*1.01))
plt.xlabel("Cars Involved")
plt.ylabel("Number of Fraud");
plt.yticks([])
plt.show()

#Plotting the graph separately showing frauds found in MaritalStatus
fraud = df[df['FraudFound'] == 1]
plt.figure(figsize=(10,5))
plt.title("Marital Status Vs Frauds")
sns.countplot(x=fraud['MaritalStatus']);
plt.xlabel("Marital Status")
plt.ylabel("Number of Fraud");
```

```
le = LabelEncoder()

cols = df.select_dtypes('O').columns

df[cols]= df[cols].apply(le.fit_transform)
df['Year'] = le.fit_transform(df.Year)

plt.figure(figsize=(20,20))
sns.heatmap(df.corr(),annot=True,linewidth=0.5,fmt="0.2f")

df_new = df[['Make', 'AccidentArea','Sex',
       'MaritalStatus','Fault', 'VehicleCategory',\
       'VehiclePrice', 'Year',\
       'DriverRating', 'Days:Policy-Accident', 'Days:Policy-Claim',\
       'PastNumberOfClaims', 'AgeOfVehicle', 'AgeOfPolicyHolder',\
       'PoliceReportFiled', 'WitnessPresent', 'AgentType',\
       'NumberOfSuppliments', 'AddressChange-Claim', 'NumberOfCars',\
       'BasePolicy', 'FraudFound']]

plt.figure(figsize=(20,20))
sns.heatmap(df_new.corr(),annot=True,linewidth=0.5,fmt="0.2f")

X = df_new.drop(['FraudFound'], axis=1)
y = df_new['FraudFound']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42, stratify=y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape

def conf_matrix(y_test,y_pred):
    con_matrix = confusion_matrix(y_test,y_pred)
    con_matrix = pd.DataFrame(con_matrix,range(2),range(2))

    plt.figure(figsize=(5,5))
    plt.title("Confusion Matrix")
    sns.heatmap(con_matrix,annot=True,cbar=False,fmt='g')



lr = LogisticRegression()
lr.fit(X_train,y_train)
lr_pred = lr.predict(X_test)


print('Accuracy of LogisticRegression model:',accuracy_score(y_test, lr_pred))
print('F1 Score for LogisticRegression model:', f1_score(y_test, lr_pred))
print('Precision for LogisticRegression model:', precision_score(y_test, lr_pred))
print('Recall for LogisticRegression model:', recall_score(y_test, lr_pred))


fpr, tpr, thresholds = roc_curve(y_test, lr_pred)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for LogisticRegression:', roc_auc)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
```

```
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for LogisticRegression')
plt.legend(loc="lower right")
plt.show()

conf_matrix(y_test,lr_pred)

rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)
rfc_pred = rfc.predict(X_test)

acc_rfc = accuracy_score(y_test, rfc_pred)
print('Accuracy of RandomForestClassifier model:',accuracy_score(y_test, rfc_pred))
print('F1 Score for RandomForestClassifier model:', f1_score(y_test, rfc_pred))
print('Precision for RandomForestClassifier model:', precision_score(y_test,
rfc_pred))
print('Recall for RandomForestClassifier model:', recall_score(y_test, rfc_pred))


fpr, tpr, thresholds = roc_curve(y_test, rfc_pred)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for LogisticRegression:', roc_auc)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for RandomForestClassifier')
plt.legend(loc="lower right")
plt.show()

conf_matrix(y_test,rfc_pred)

knn = KNeighborsClassifier()
knn.fit(X_train,y_train)
knn_pred = rfc.predict(X_test)

acc_knn = accuracy_score(y_test, knn_pred)
print('Accuracy of KNeighborsClassifier model:',accuracy_score(y_test, knn_pred))
print('F1 Score for KNeighborsClassifier model:', f1_score(y_test, knn_pred))
print('Precision for KNeighborsClassifier model:', precision_score(y_test, knn_pred))
print('Recall for KNeighborsClassifier model:', recall_score(y_test, knn_pred))


fpr, tpr, thresholds = roc_curve(y_test, knn_pred)
roc_auc = auc(fpr, tpr)
```

```
print(f'ROC AUC for KNeighborsClassifier:', roc_auc)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for KNeighborsClassifier')
plt.legend(loc="lower right")
plt.show()

conf_matrix(y_test,knn_pred)

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train,y_train)
dt_pred = rfc.predict(X_test)

acc_dt = accuracy_score(y_test, dt_pred)
print('Accuracy of DecisionTreeClassifier model:',accuracy_score(y_test, dt_pred))
print('F1 Score for DecisionTreeClassifier model:', f1_score(y_test, dt_pred))
print('Precision for DecisionTreeClassifier model:', precision_score(y_test,
dt_pred))
print('Recall for DecisionTreeClassifier model:', recall_score(y_test, dt_pred))


fpr, tpr, thresholds = roc_curve(y_test, dt_pred)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for DecisionTreeClassifier:', roc_auc)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for DecisionTreeClassifier')
plt.legend(loc="lower right")
plt.show()

conf_matrix(y_test,dt_pred)

mn = df_new.FraudFound.value_counts()[0]

df_majority = df_new[df_new.FraudFound==0]
df_minority = df_new[df_new.FraudFound==1]

df_minority_upsampled = resample(df_minority,replace=True,n_samples =
mn,random_state=42)
```

```
df_upsampled = pd.concat([df_majority,df_minority_upsampled])
df_upsampled.FraudFound.value_counts()

X_up = df_upsampled.drop('FraudFound',axis=1)
y_up = df_upsampled[['FraudFound']]
X_train_up,X_test_up,y_train_up,y_test_up = train_test_split(X_up,y_up,stratify=y_up)
X_train_up.shape, X_test_up.shape, y_train_up.shape, y_test_up.shape

# Upsampled Logistic Regression model
lr.fit(X_train_up,y_train_up)
lr_pred_up = lr.predict(X_test_up)

acc_lr_up = accuracy_score(y_test_up, lr_pred_up)
f1_lr_up = f1_score(y_test_up, lr_pred_up)
pre_lr_up = precision_score(y_test_up, lr_pred_up)
recall_lr_up = recall_score(y_test_up, lr_pred_up)

print("Accuracy of Upsampled LogisticRegression model:",acc_lr_up)
print(f'F1 Score for Upsampled LogisticRegression model:', f1_lr_up)
print(f'Precision for Upsampled LogisticRegression model:', pre_lr_up)
print(f'Recall for Upsampled LogisticRegression model:', recall_lr_up)


fpr, tpr, thresholds = roc_curve(y_test_up, lr_pred_up)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Upsampled LogisticRegression:', roc_auc)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Upsampled LogisticRegression')
plt.legend(loc="lower right")
plt.show()

conf_matrix(y_test_up,lr_pred_up)

# Upsampled RandomForestClassifier model
rfc.fit(X_train_up,y_train_up)
rfc_pred_up = rfc.predict(X_test_up)

acc_rfc_up = accuracy_score(y_test_up, rfc_pred_up)
f1_rfc_up = f1_score(y_test_up, rfc_pred_up)
pre_rfc_up = precision_score(y_test_up, rfc_pred_up)
recall_rfc_up = recall_score(y_test_up, rfc_pred_up)

print("Accuracy of Upsampled RandomForestClassifier model:",acc_rfc_up)
print(f'F1 Score for Upsampled RandomForestClassifier model:', f1_rfc_up)
print(f'Precision for Upsampled RandomForestClassifier model:', pre_rfc_up)
print(f'Recall for Upsampled RandomForestClassifier model:', recall_rfc_up)
```

```
fpr, tpr, thresholds = roc_curve(y_test_up, rfc_pred_up)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Upsampled RandomForestClassifier:', roc_auc)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Upsampled RandomForestClassifier')
plt.legend(loc="lower right")
plt.show()

conf_matrix(y_test_up,rfc_pred_up)

# Upsampled KNeighborsClassifier model
knn.fit(X_train_up,y_train_up)
knn_pred_up = knn.predict(X_test_up)

acc_knn_up = accuracy_score(y_test_up, knn_pred_up)
f1_knn_up = f1_score(y_test_up, knn_pred_up)
pre_knn_up = precision_score(y_test_up, knn_pred_up)
recall_knn_up = recall_score(y_test_up, knn_pred_up)

print("Accuracy of Upsampled KNeighborsClassifier model:",acc_knn_up)
print(f'F1 Score for Upsampled KNeighborsClassifier model:', f1_knn_up)
print(f'Precision for Upsampled KNeighborsClassifier model:', pre_knn_up)
print(f'Recall for Upsampled KNeighborsClassifier model:', recall_knn_up)

fpr, tpr, thresholds = roc_curve(y_test_up, knn_pred_up)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Upsampled KNeighborsClassifier:', roc_auc)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Upsampled KNeighborsClassifier')
plt.legend(loc="lower right")
plt.show()

conf_matrix(y_test_up,knn_pred_up)

# Upsampled DecisionTreeClassifier model
dt.fit(X_train_up,y_train_up)
dt_pred_up = dt.predict(X_test_up)
```

```
acc_dt_up = accuracy_score(y_test_up, dt_pred_up)
f1_dt_up = f1_score(y_test_up, dt_pred_up)
pre_dt_up = precision_score(y_test_up, dt_pred_up)
recall_dt_up = recall_score(y_test_up, dt_pred_up)

print("Accuracy of Upsampled DecisionTreeClassifier model:", acc_dt_up)
print(f'F1 Score for Upsampled DecisionTreeClassifier model:', f1_dt_up)
print(f'Precision for Upsampled DecisionTreeClassifier model:', pre_dt_up )
print(f'Recall for Upsampled DecisionTreeClassifier model:', recall_dt_up)


fpr, tpr, thresholds = roc_curve(y_test_up, dt_pred_up)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Upsampled DecisionTreeClassifier:', roc_auc)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Upsampled DecisionTreeClassifier')
plt.legend(loc="lower right")
plt.show()

conf_matrix(y_test_up,dt_pred_up)

mj = df_new.FraudFound.value_counts()[1]

df_majority = df_new[df_new.FraudFound==0]
df_minority = df_new[df_new.FraudFound==1]

df_majority_downsampled = resample(df_majority,replace=False,n_samples =
mj,random_state=42)

df_downsampled = pd.concat([df_minority,df_majority_downsampled])
df_downsampled.FraudFound.value_counts()

X_dwn = df_downsampled.drop('FraudFound',axis=1)
y_dwn = df_downsampled[['FraudFound']]
X_train_dwn,X_test_dwn,y_train_dwn,y_test_dwn =
train_test_split(X_dwn,y_dwn,stratify=y_dwn)
X_train_dwn.shape, X_test_dwn.shape, y_train_dwn.shape, y_test_dwn.shape

# DownSampled Logistic Regression model
lr.fit(X_train_dwn,y_train_dwn)
lr_pred_dwn = lr.predict(X_test_dwn)

acc_lr_dwn = accuracy_score(y_test_dwn, lr_pred_dwn)
f1_lr_dwn = f1_score(y_test_dwn, lr_pred_dwn)
pre_lr_dwn = precision_score(y_test_dwn, lr_pred_dwn)
recall_lr_dwn = recall_score(y_test_dwn, lr_pred_dwn)
```

```
print('Accuracy of Downsampled LogisticRegression model:', acc_lr_dwn)
print('F1 Score for Downsampled LogisticRegression model:', f1_lr_dwn)
print('Precision for Downsampled LogisticRegression model:', pre_lr_dwn)
print('Recall for Downsampled LogisticRegression model:', recall_lr_dwn)


fpr, tpr, thresholds = roc_curve(y_test_dwn, lr_pred_dwn)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Downsampled LogisticRegression:', roc_auc)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Downsampled LogisticRegression')
plt.legend(loc="lower right")
plt.show()

conf_matrix(y_test_dwn,lr_pred_dwn)

# DownSampled RandomForestClassifier model
rfc.fit(X_train_dwn,y_train_dwn)
rfc_pred_dwn = rfc.predict(X_test_dwn)

acc_rfc_dwn = accuracy_score(y_test_dwn, rfc_pred_dwn)
f1_rfc_dwn = f1_score(y_test_dwn, rfc_pred_dwn)
pre_rfc_dwn = precision_score(y_test_dwn, rfc_pred_dwn)
recall_rfc_dwn = recall_score(y_test_dwn, rfc_pred_dwn)

print('Accuracy of Downsampled RandomForestClassifier model:', acc_rfc_dwn)
print('F1 Score for Downsampled RandomForestClassifier model:', f1_rfc_dwn)
print('Precision for Downsampled RandomForestClassifier model:', pre_rfc_dwn)
print('Recall for Downsampled RandomForestClassifier model:', recall_rfc_dwn )


fpr, tpr, thresholds = roc_curve(y_test_dwn, rfc_pred_dwn)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Downsampled RandomForestClassifier:', roc_auc)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Downsampled
RandomForestClassifier')
plt.legend(loc="lower right")
plt.show()
```

```
conf_matrix(y_test_dwn,rfc_pred_dwn)

# DownSampled KNeighborsClassifier model
knn.fit(X_train_dwn,y_train_dwn)
knn_pred_dwn = knn.predict(X_test_dwn)

acc_knn_dwn = accuracy_score(y_test_dwn, knn_pred_dwn)
f1_knn_dwn = f1_score(y_test_dwn, knn_pred_dwn)
pre_knn_dwn = precision_score(y_test_dwn, knn_pred_dwn)
recall_knn_dwn = recall_score(y_test_dwn, knn_pred_dwn)

print('Accuracy of Downsampled KNeighborsClassifier model:', acc_knn_dwn)
print('F1 Score for Downsampled KNeighborsClassifier model:', f1_knn_dwn)
print('Precision for Downsampled KNeighborsClassifier model:', pre_knn_dwn)
print('Recall for Downsampled KNeighborsClassifier model:', recall_knn_dwn)


fpr, tpr, thresholds = roc_curve(y_test_dwn, knn_pred_dwn)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Downsampled KNeighborsClassifier:', roc_auc)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Downsampled KNeighborsClassifier')
plt.legend(loc="lower right")
plt.show()

conf_matrix(y_test_dwn,knn_pred_dwn)

# DownSampled DecisionTreeClassifier model
dt.fit(X_train_dwn,y_train_dwn)
dt_pred_dwn = dt.predict(X_test_dwn)

acc_dt_dwn = accuracy_score(y_test_dwn, dt_pred_dwn)
f1_dt_dwn =f1_score(y_test_dwn, dt_pred_dwn)
pre_dt_dwn = precision_score(y_test_dwn, dt_pred_dwn)
recall_dt_dwn = recall_score(y_test_dwn, dt_pred_dwn)

print('Accuracy of Downsampled DecisionTreeClassifier model:', acc_dt_dwn)
print('F1 Score for Downsampled DecisionTreeClassifier model:', f1_dt_dwn)
print('Precision for Downsampled DecisionTreeClassifier model:', pre_dt_dwn)
print('Recall for Downsampled DecisionTreeClassifier model:', recall_dt_dwn)


fpr, tpr, thresholds = roc_curve(y_test_dwn, dt_pred_dwn)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC for Downsampled DecisionTreeClassifier:', roc_auc)
```

```
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Receiver Operating Characteristic for Downsampled
DecisionTreeClassifier')
plt.legend(loc="lower right")
plt.show()

conf_matrix(y_test_dwn,dt_pred_dwn)



data = {'Model':['UpSampled Logistic Regression','DownSampled Logistic Regression',
                 'UpSampled RandomForest Classifier','DownSampled RandomForest
Classifier',
                 'UpSampled KNeighborsClassifier', 'DownSampled
KNeighborsClassifier',
                 'UpSampled DecisionTreeClassifier','DownSampled
DecisionTreeClassifier'],

        'Accuracy Score':[acc_lr_up, acc_lr_dwn, acc_rfc_up, acc_rfc_dwn, acc_knn_up,
acc_knn_dwn,
                          acc_dt_up, acc_dt_dwn],

        'F1_Score':[f1_lr_up, f1_lr_dwn, f1_rfc_up, f1_rfc_dwn, f1_knn_up,
f1_knn_dwn,
                          f1_dt_up, f1_dt_dwn],

        'Precision':[pre_lr_up, pre_lr_dwn, pre_rfc_up, pre_rfc_dwn, pre_knn_up,
pre_knn_dwn,
                          pre_dt_up, pre_dt_dwn],

        'Recall_Score':[recall_lr_up, recall_lr_dwn, recall_rfc_up, recall_rfc_dwn,
recall_knn_up, recall_knn_dwn,
                          recall_dt_up, recall_dt_dwn]}

comparision_table=pd.DataFrame(data)
comparision_table
```