Insurance Fraud Project

May 24, 2023

Objective of this project is to find the best ML Model that can predict Car Insurance Frauds. This dataset was found on Kaggle.com.

First I'll explore the dataset in order to show its characteristic with EDA. Second, I'll preprocess the data, then I'll train and test 6 ML/DL models.

0.1 Importing necessary libraries and the dataset

[1]:

```
# Importing necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_auc_score
from imblearn.under_sampling import RandomUnderSampler
from tabulate import tabulate
from sklearn.metrics import roc_auc_score, confusion_matrix
from matplotlib.colors import ListedColormap

# Reading the dataset
df = pd.read_csv("C:/Users/Utente/Desktop/Progetto/fraud_oracle.csv")
```

[2]:

```
df.head()
```

[2]:

```
  Month  WeekOfMonth  DayOfWeek    Make AccidentArea DayOfWeekClaimed  \
0   Dec            5  Wednesday   Honda        Urban          Tuesday
1   Jan            3  Wednesday   Honda        Urban           Monday
2   Oct            5     Friday   Honda        Urban         Thursday
3   Jun            2   Saturday  Toyota        Rural           Friday
4   Jan            5     Monday   Honda        Urban          Tuesday
```

```
  MonthClaimed  WeekOfMonthClaimed     Sex MaritalStatus  …  AgeOfVehicle  \
0          Jan                   1  Female        Single  …       3 years
1          Jan                   4    Male        Single  …       6 years
2          Nov                   2    Male       Married  …       7 years
3          Jul                   1    Male       Married  …   more than 7
4          Feb                   2  Female        Single  …       5 years

  AgeOfPolicyHolder PoliceReportFiled WitnessPresent AgentType  \
0          26 to 30                No             No  External
1          31 to 35               Yes             No  External
2          41 to 50                No             No  External
3          51 to 65               Yes             No  External
4          31 to 35                No             No  External

   NumberOfSuppliments  AddressChange_Claim  NumberOfCars  Year  BasePolicy
0                 none               1 year        3 to 4  1994   Liability
1                 none            no change     1 vehicle  1994   Collision
2                 none            no change     1 vehicle  1994   Collision
3          more than 5            no change     1 vehicle  1994   Liability
4                 none            no change     1 vehicle  1994   Collision

[5 rows x 33 columns]
```

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15420 entries, 0 to 15419
Data columns (total 33 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Month                 15420 non-null  object
 1   WeekOfMonth           15420 non-null  int64
 2   DayOfWeek             15420 non-null  object
 3   Make                  15420 non-null  object
 4   AccidentArea          15420 non-null  object
 5   DayOfWeekClaimed      15420 non-null  object
 6   MonthClaimed          15420 non-null  object
 7   WeekOfMonthClaimed    15420 non-null  int64
 8   Sex                   15420 non-null  object
 9   MaritalStatus         15420 non-null  object
 10  Age                   15420 non-null  int64
 11  Fault                 15420 non-null  object
 12  PolicyType            15420 non-null  object
 13  VehicleCategory       15420 non-null  object
 14  VehiclePrice          15420 non-null  object
 15  FraudFound_P          15420 non-null  int64
 16  PolicyNumber          15420 non-null  int64
 17  RepNumber             15420 non-null  int64
```

```
 18  Deductible            15420 non-null  int64
 19  DriverRating          15420 non-null  int64
 20  Days_Policy_Accident  15420 non-null  object
 21  Days_Policy_Claim     15420 non-null  object
 22  PastNumberOfClaims    15420 non-null  object
 23  AgeOfVehicle          15420 non-null  object
 24  AgeOfPolicyHolder     15420 non-null  object
 25  PoliceReportFiled     15420 non-null  object
 26  WitnessPresent        15420 non-null  object
 27  AgentType             15420 non-null  object
 28  NumberOfSuppliments   15420 non-null  object
 29  AddressChange_Claim   15420 non-null  object
 30  NumberOfCars          15420 non-null  object
 31  Year                  15420 non-null  int64
 32  BasePolicy            15420 non-null  object
dtypes: int64(9), object(24)
memory usage: 3.9+ MB
```

[4]:
```
df.describe()
```

[4]:
```
        WeekOfMonth  WeekOfMonthClaimed           Age  FraudFound_P  \
count  15420.000000        15420.000000  15420.000000  15420.000000
mean       2.788586            2.693969     39.855707      0.059857
std        1.287585            1.259115     13.492377      0.237230
min        1.000000            1.000000      0.000000      0.000000
25%        2.000000            2.000000     31.000000      0.000000
50%        3.000000            3.000000     38.000000      0.000000
75%        4.000000            4.000000     48.000000      0.000000
max        5.000000            5.000000     80.000000      1.000000

       PolicyNumber     RepNumber    Deductible  DriverRating          Year
count  15420.000000  15420.000000  15420.000000  15420.000000  15420.000000
mean    7710.500000      8.483268    407.704280      2.487808   1994.866472
std     4451.514911      4.599948     43.950998      1.119453      0.803313
min        1.000000      1.000000    300.000000      1.000000   1994.000000
25%     3855.750000      5.000000    400.000000      1.000000   1994.000000
50%     7710.500000      8.000000    400.000000      2.000000   1995.000000
75%    11565.250000     12.000000    400.000000      3.000000   1996.000000
max    15420.000000     16.000000    700.000000      4.000000   1996.000000
```

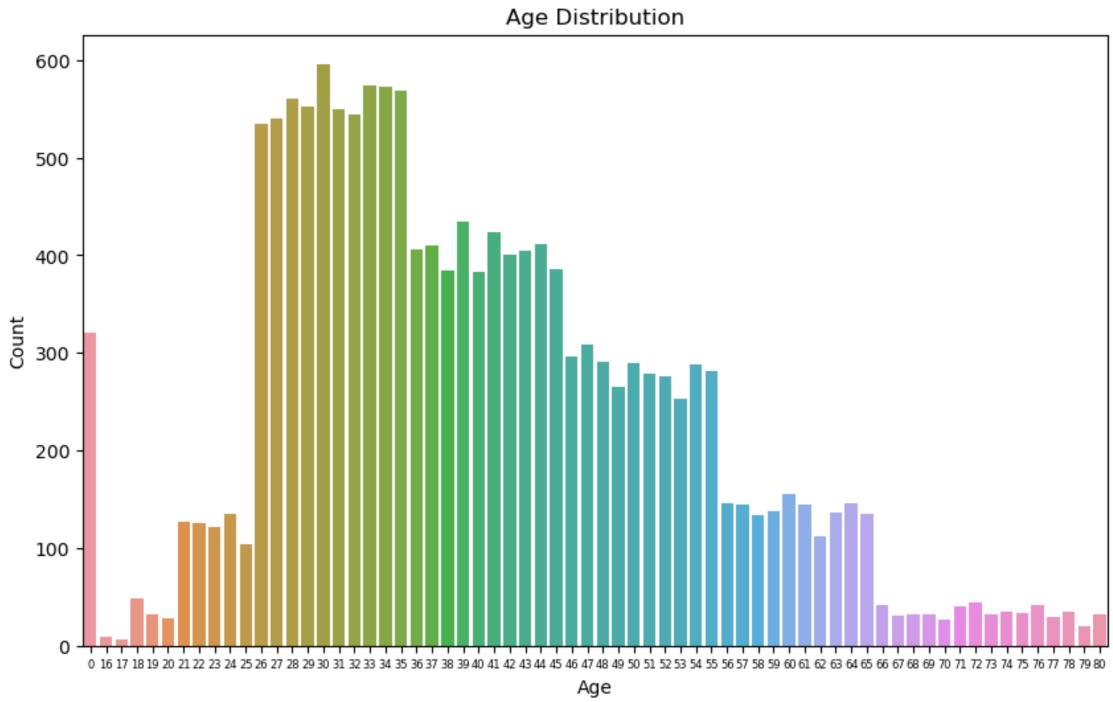
0.2 Data Preprocessing and EDA

[5]:
```
# Function to check missing values
def check_na(data):
    na = data.isna().sum()
    return na
```

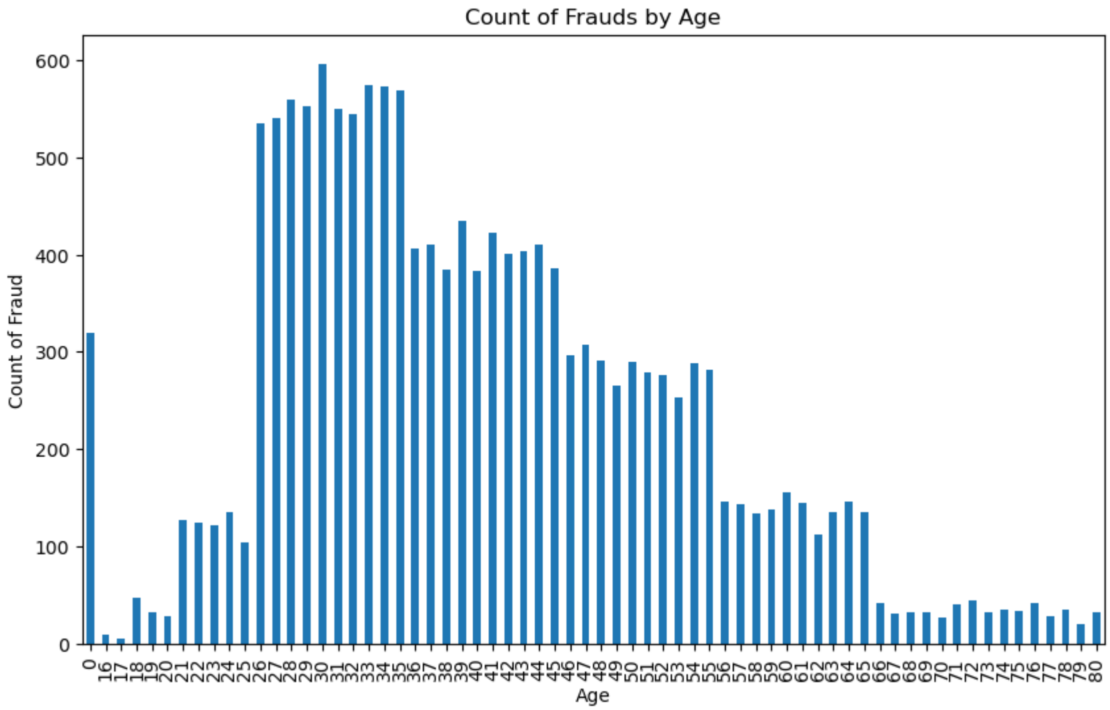
```
check_na(df)
```

[5]:
```
Month                    0
WeekOfMonth              0
DayOfWeek                0
Make                     0
AccidentArea             0
DayOfWeekClaimed         0
MonthClaimed             0
WeekOfMonthClaimed       0
Sex                      0
MaritalStatus            0
Age                      0
Fault                    0
PolicyType               0
VehicleCategory          0
VehiclePrice             0
FraudFound_P             0
PolicyNumber             0
RepNumber                0
Deductible               0
DriverRating             0
Days_Policy_Accident     0
Days_Policy_Claim        0
PastNumberOfClaims       0
AgeOfVehicle             0
AgeOfPolicyHolder        0
PoliceReportFiled        0
WitnessPresent           0
AgentType                0
NumberOfSuppliments      0
AddressChange_Claim      0
NumberOfCars             0
Year                     0
BasePolicy               0
dtype: int64
```

[6]:
```
# Create the countplot for Age Variable
plt.figure(figsize=(10, 6))
ax = sns.countplot(x='Age', data=df)
ax.tick_params(axis='x', labelsize=6)
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Age Distribution')
plt.show()
```

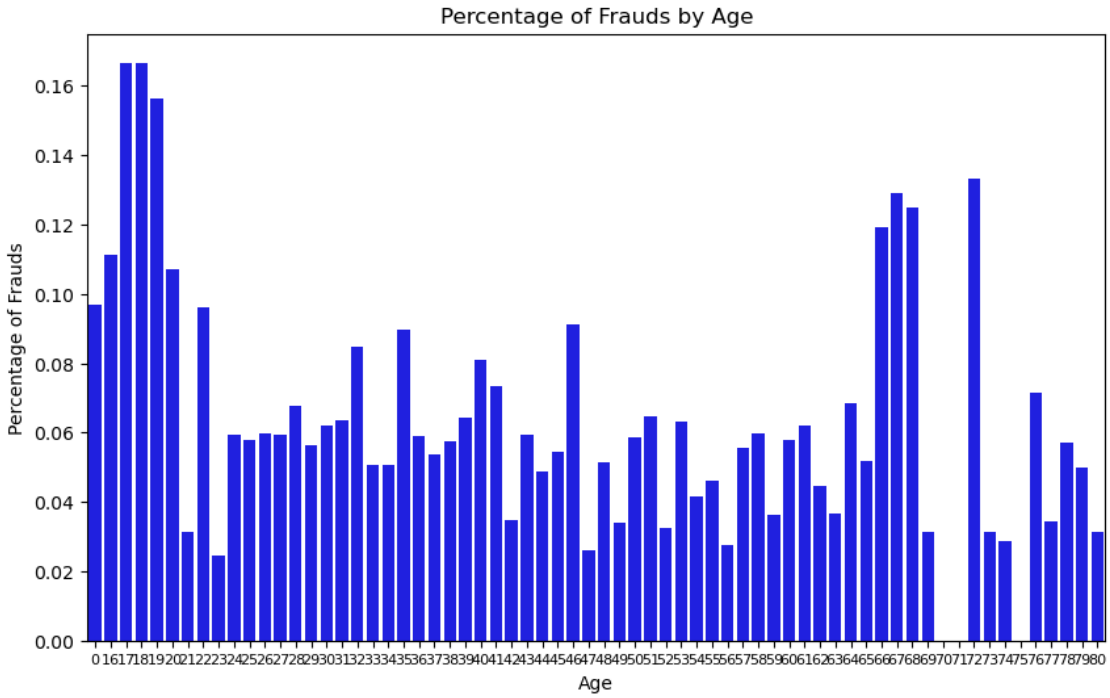


[7]:
```
plt.figure(figsize=(10, 6))
df.groupby('Age')['FraudFound_P'].count().plot(kind='bar')
plt.title('Count of Frauds by Age')
plt.xlabel('Age')
plt.ylabel('Count of Fraud')
plt.show()
```

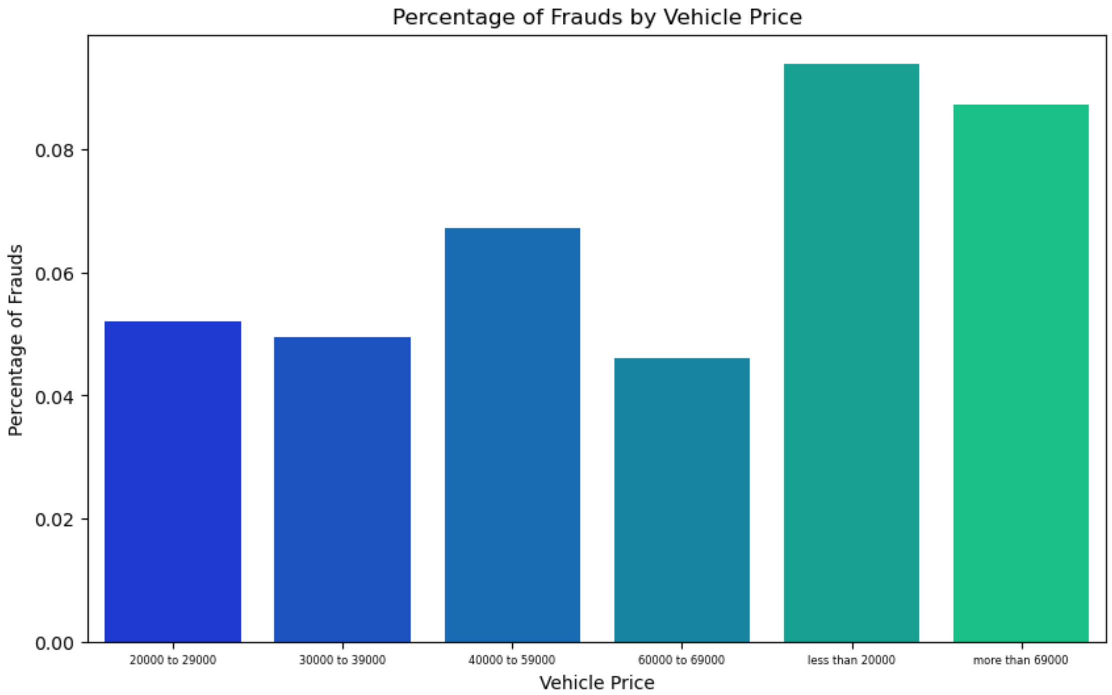


[8]:

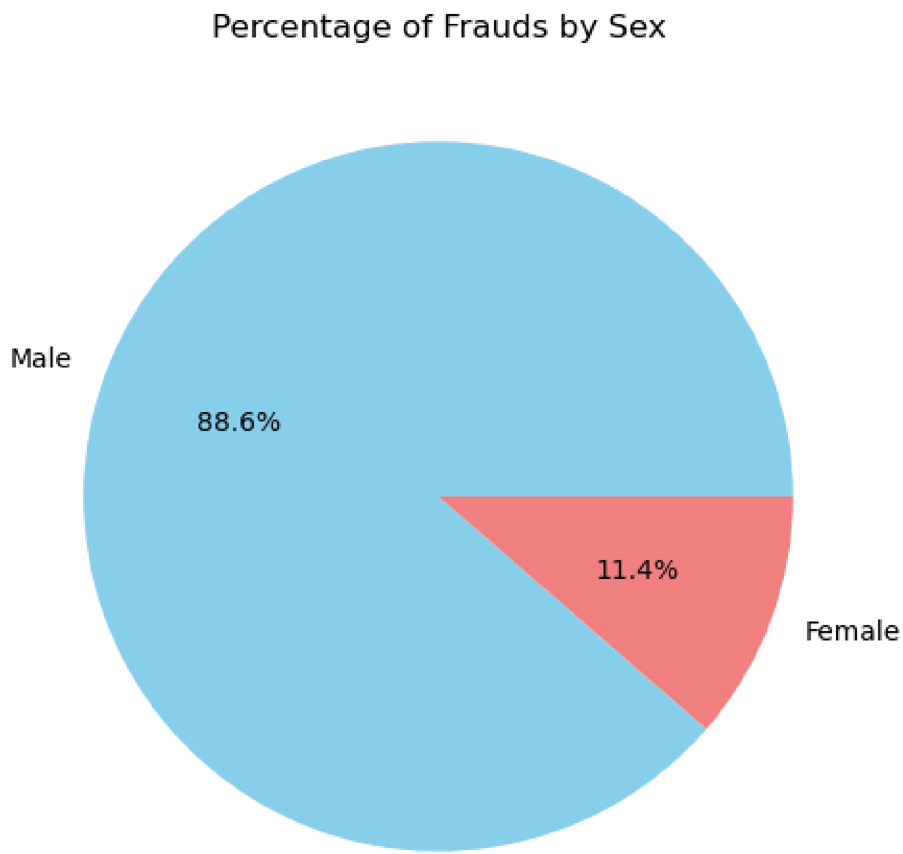
```
# Calculate the percentage of frauds for each age category
fraud_percentage = df[df['FraudFound_P'] == 1].groupby('Age')['FraudFound_P'].
    ↪count() / df.groupby('Age')['FraudFound_P'].count()

# Create the bar plot
plt.figure(figsize=(10, 6))
ax = sns.barplot(x=fraud_percentage.index, y=fraud_percentage.values,␣
    ↪color='blue')
ax.tick_params(axis='x', labelsize=8)
plt.xlabel('Age')
plt.ylabel('Percentage of Frauds')
plt.title('Percentage of Frauds by Age')

plt.show()
```



[9]:
```
# Calculate the percentage of frauds for each age category
fraud_percentage = df[df['FraudFound_P'] == 1].
  ↪groupby('VehiclePrice')['FraudFound_P'].count() / df.
  ↪groupby('VehiclePrice')['FraudFound_P'].count()

# Create the bar plot
plt.figure(figsize=(10, 6))
pal = sns.color_palette("winter", len(fraud_percentage))
ax = sns.barplot(x=fraud_percentage.index, y=fraud_percentage.values,␣
  ↪palette=pal)
ax.tick_params(axis='x', labelsize=6)
plt.xlabel('Vehicle Price')
plt.ylabel('Percentage of Frauds')
plt.title('Percentage of Frauds by Vehicle Price')

# Display the plot
plt.show()
```



[10]:
```
# Calculate the percentage of frauds for each sex category
fraud_percentage = df[df['FraudFound_P'] == 1]['Sex'].
    ↪value_counts(normalize=True) * 100

# Create the pie plot
plt.figure(figsize=(8, 6))
colors = ['skyblue', 'lightcoral']
plt.pie(fraud_percentage, labels=fraud_percentage.index, autopct='%1.1f%%',␣
    ↪colors=colors)
plt.title('Percentage of Frauds by Sex')

# Display the plot
plt.show()
```




```
[11]: # Creating dummy variables for categorical columns
columns_dummies = ['AccidentArea', 'AgeOfPolicyHolder', 'BasePolicy', 'Make',
                   'MaritalStatus', 'Sex', 'Days_Policy_Claim',␣
 ↪'PastNumberOfClaims',
                   'VehiclePrice', 'AgeOfVehicle', 'AddressChange_Claim',␣
 ↪'PoliceReportFiled',
                   'WitnessPresent', 'AgentType', 'NumberOfSuppliments',
                   'NumberOfCars', 'VehicleCategory', 'Fault', 'PolicyType']

df = pd.concat([df.drop(columns_dummies, axis=1),
                pd.get_dummies(df[columns_dummies], dtype=float)], axis=1)

# Mapping categorical values to numerical values
day_mapping = {
    'Monday': 1,
    'Tuesday': 2,
    'Wednesday': 3,
```

```python
    'Thursday': 4,
    'Friday': 5,
    'Saturday': 6,
    'Sunday': 7
}

df['DayOfWeek'] = df['DayOfWeek'].map(day_mapping)
df['DayOfWeekClaimed'] = df['DayOfWeekClaimed'].map(day_mapping)
print(df['DayOfWeek'])

month_mapping = {
    'Jan': 1,
    'Feb': 2,
    'Mar': 3,
    'Apr': 4,
    'May': 5,
    'Jun': 6,
    'Jul': 7,
    'Aug': 8,
    'Sep': 9,
    'Oct': 10,
    'Nov': 11,
    'Dec': 12
}

df['Month'] = df['Month'].map(month_mapping)
df['MonthClaimed'] = df['MonthClaimed'].map(month_mapping)

# Checking missing values again
def check_na(data):
    na = data.isna().sum()
    return na

check_na(df)

# Dropping rows with missing values
df = df.dropna()
check_na(df)
df.head()
```

```
0        3
1        3
2        5
3        6
4        1
        ..
15415    5
15416    4
```

```
15417    4
15418    1
15419    3
Name: DayOfWeek, Length: 15420, dtype: int64
```

[11]:

```
   Month  WeekOfMonth  DayOfWeek  DayOfWeekClaimed  MonthClaimed  \
0     12            5          3               2.0           1.0
1      1            3          3               1.0           1.0
2     10            5          5               4.0          11.0
3      6            2          6               5.0           7.0
4      1            5          1               2.0           2.0

   WeekOfMonthClaimed  Age  FraudFound_P  PolicyNumber  RepNumber  …  \
0                   1   21             0             1         12  …
1                   4   34             0             2         15  …
2                   2   47             0             3          7  …
3                   1   65             0             4          4  …
4                   2   27             0             5          3  …

   Fault_Third Party  PolicyType_Sedan - All Perils  \
0                0.0                            0.0
1                0.0                            0.0
2                0.0                            0.0
3                1.0                            0.0
4                1.0                            0.0

  PolicyType_Sedan - Collision  PolicyType_Sedan - Liability  \
0                          0.0                           0.0
1                          0.0                           0.0
2                          0.0                           0.0
3                          0.0                           1.0
4                          0.0                           0.0

   PolicyType_Sport - All Perils  PolicyType_Sport - Collision  \
0                            0.0                           0.0
1                            0.0                           1.0
2                            0.0                           1.0
3                            0.0                           0.0
4                            0.0                           1.0

   PolicyType_Sport - Liability  PolicyType_Utility - All Perils  \
0                           1.0                              0.0
1                           0.0                              0.0
2                           0.0                              0.0
3                           0.0                              0.0
4                           0.0                              0.0
```

```
   PolicyType_Utility - Collision  PolicyType_Utility - Liability
0                             0.0                             0.0
1                             0.0                             0.0
2                             0.0                             0.0
3                             0.0                             0.0
4                             0.0                             0.0

[5 rows x 109 columns]
```

[12]:

```
# Plotting the count of fraud and non-fraud transactions
ax = sns.countplot(x='FraudFound_P', data=df)
plt.xlabel('Fraud')
plt.ylabel('Count')
plt.title('Number of non-fraud and fraud transactions')

# Adding count numbers inside each bin
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2, height),
                ha='center', va='bottom', fontweight='bold', color='black')

# Displaying the plot
plt.show()
```

0.3 Rebalancing the Dataset

[13]:
```
# Calculating the ratio of fraud transactions
unbalance = df.FraudFound_P[df['FraudFound_P']==1].count()/df['FraudFound_P'].
  ↪count()
print(unbalance)
```

0.0598612101952137

[14]:
```
X = df.drop('FraudFound_P', axis=1)
y = df['FraudFound_P']

rus = RandomUnderSampler()

X_resampled, y_resampled = rus.fit_resample(X, y)
df = pd.concat([X_resampled, y_resampled], axis=1)
ax = sns.countplot(x='FraudFound_P', data=df)

# add labels and title
plt.xlabel('Fraud')
```

```
plt.ylabel('Count')
plt.title('Number of fraud and non-fraud transactions')

# add count numbers inside each bin
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height}', (p.get_x() + p.get_width() / 2, height),
                ha='center', va='bottom', fontweight='bold', color='black')

# display the plot
plt.show()
```

0.4 Splitting, Scaling and Encoding Data

[15]:
```
# Splitting the data into training and testing sets
X = df.drop('FraudFound_P', axis=1)
y = df['FraudFound_P']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
# Scaling numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.select_dtypes(include=['float64',␣
↪'int64']))
X_test_scaled = scaler.transform(X_test.select_dtypes(include=['float64',␣
↪'int64']))

# Encoding target variable
encoder = LabelEncoder()
y_train_enc = encoder.fit_transform(y_train)
y_test_enc = encoder.transform(y_test)
```

0.5 Training and Evaluating Models

[16]:
```
# Define the hyperparameter grids for each model
lr_param_grid = {'C': [0.1, 1, 10], 'penalty': ['l2']}
rf_param_grid = {'n_estimators': [100, 200, 300], 'max_depth': [None, 5, 10]}
dt_param_grid = {'max_depth': [None, 5, 10], 'min_samples_split': [2, 5, 10]}
nn_param_grid = {'hidden_layer_sizes': [(100,), (100, 50)], 'alpha': [0.0001, 0.
↪001, 0.01]}
gb_param_grid = {'n_estimators': [100, 200, 300], 'learning_rate': [0.1, 0.05,␣
↪0.01]}
svc_param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}

# Define the models
models = [
    (LogisticRegression(max_iter=1000), lr_param_grid),
    (RandomForestClassifier(), rf_param_grid),
    (DecisionTreeClassifier(), dt_param_grid),
    (MLPClassifier(max_iter=1000), nn_param_grid),
    (SVC(), svc_param_grid),
    (GradientBoostingClassifier(), gb_param_grid)
]
```

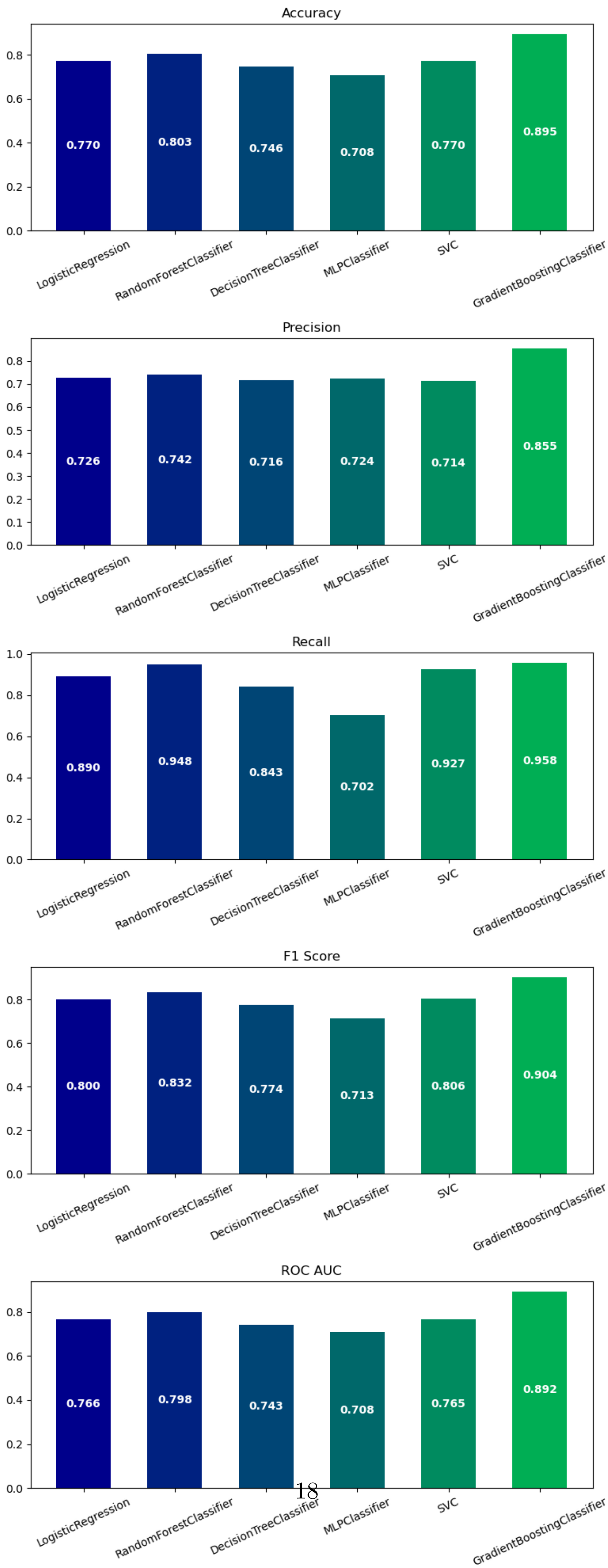
[17]:
```
# Perform model selection and evaluation
results = []
metrics = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']

for model, param_grid in models:
    scoring = 'roc_auc'

    grid_search = GridSearchCV(model, param_grid, cv=5, scoring=scoring,␣
↪n_jobs=-1)
    grid_search.fit(X_train_scaled, y_train_enc)

    best_model = grid_search.best_estimator_
    best_model.fit(X_train_scaled, y_train_enc)  # Fit the best model
```

```
    y_pred = best_model.predict(X_test_scaled)

    row = []
    for metric in metrics:
        score = eval(metric + '_score')(y_test_enc, y_pred)
        row.append(score)

    results.append(row)

df_results = pd.DataFrame(results, columns=metrics, index=[best_model.__class__.
 ↪__name__ for best_model, _ in models])

# Display the results in a table
print(tabulate(df_results, headers='keys', tablefmt='psql'))
```

```
+----------------------------+------------+-------------+----------+----------+-
----------+
|                            |   accuracy |   precision |   recall |       f1 |
roc_auc |
|----------------------------+------------+-------------+----------+----------+-
----------|
| LogisticRegression         |   0.77027  |    0.726496 | 0.890052 | 0.8      |
0.766255 |
| RandomForestClassifier     |   0.802703 |    0.741803 | 0.947644 | 0.832184 |
0.797844 |
| DecisionTreeClassifier     |   0.745946 |    0.715556 | 0.842932 | 0.774038 |
0.742695 |
| MLPClassifier              |   0.708108 |    0.724324 | 0.701571 | 0.712766 |
0.708327 |
| SVC                        |   0.77027  |    0.71371  | 0.926702 | 0.806378 |
0.765027 |
| GradientBoostingClassifier |   0.894595 |    0.85514  | 0.958115 | 0.903704 |
0.892465 |
+----------------------------+------------+-------------+----------+----------+-
----------+
```

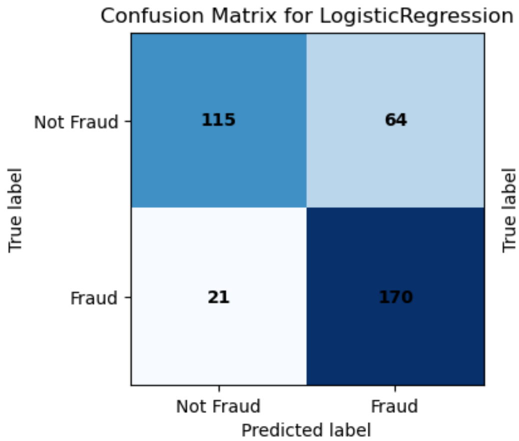
[18]:
```
# Plotting the evaluation metrics for different models
scores = np.array(results)
colors = ListedColormap([
    '#00008B', '#002180', '#004575', '#00686A',
    '#008B5F', '#00AE54', '#00D149', '#00F53E',
    '#00FF00'
])

fig, axs = plt.subplots(5, 1, figsize=(8, 20))
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score', 'ROC AUC']
```

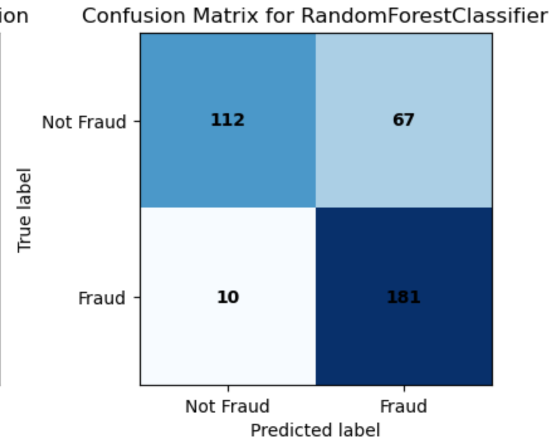
```
for i, metric in enumerate(metrics):
    bars = axs[i].bar([best_model.__class__.__name__ for best_model, _ in␣
↪models], scores[:, i], color=colors(range(len(models))), width=0.6)
    axs[i].set_title(metric)
    axs[i].tick_params(axis='x', rotation=25)
    for bar in bars:
        height = bar.get_height()
        axs[i].text(bar.get_x() + bar.get_width() / 2, height * 0.5, f'{height:.
↪3f}', ha='center', va='center', color='white', fontweight='bold')
plt.tight_layout()
plt.show()
```

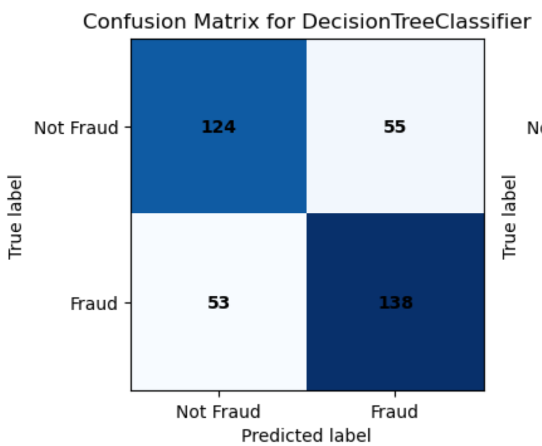

Accuracy
0.770
0.803
0.746
0.708
0.770
0.895
Precision
0.726
0.742
0.716
0.724
0.714
0.855
Recall
0.890
0.948
0.843
0.702
0.927
0.958
F1 Score
0.800
0.832
0.774
0.713
0.806
0.904
ROC AUC
0.766
0.798
0.743
0.708
0.765
0.892
LogisticRegression
RandomForestClassifier
DecisionTreeClassifier
MLPClassifier
SVC
GradientBoostingClassifier

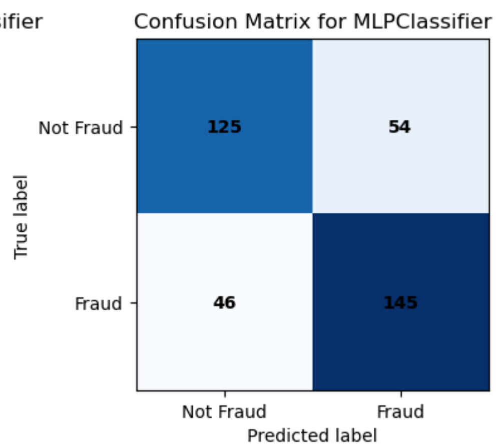
[19]:
```python
# Generating and displaying confusion matrices for different models
num_models = len(models)
num_cols = 2  # Number of columns to display

# Calculate the number of rows needed to display all the models
num_rows = (num_models + num_cols - 1) // num_cols

# Set the figure size based on the number of rows and columns
fig, axes = plt.subplots(num_rows, num_cols, figsize=(8, 6*num_rows))

# Flatten the axes array for easy iteration
axes = axes.flatten()

for i, (best_model, _) in enumerate(models):
    best_model.fit(X_train_scaled, y_train_enc)
    y_pred = best_model.predict(X_test_scaled)
    cm = confusion_matrix(y_test_enc, y_pred)

    # Set the current subplot
    ax = axes[i]

    # Plot the confusion matrix
    im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    ax.set_xticks([0, 1])
    ax.set_yticks([0, 1])
    ax.set_xticklabels(['Not Fraud', 'Fraud'])
    ax.set_yticklabels(['Not Fraud', 'Fraud'])
    ax.set_xlabel('Predicted label')
    ax.set_ylabel('True label')
    ax.set_title(f'Confusion Matrix for {best_model.__class__.__name__}')

    for i in range(2):
        for j in range(2):
            ax.text(j, i, str(cm[i, j]), ha='center', va='center',␣
 ↪color='black', fontweight='bold')

# Remove empty subplots if there are any
if num_models < len(axes):
    for i in range(num_models, len(axes)):
        fig.delaxes(axes[i])

# Adjust the spacing between subplots
plt.tight_layout()
```

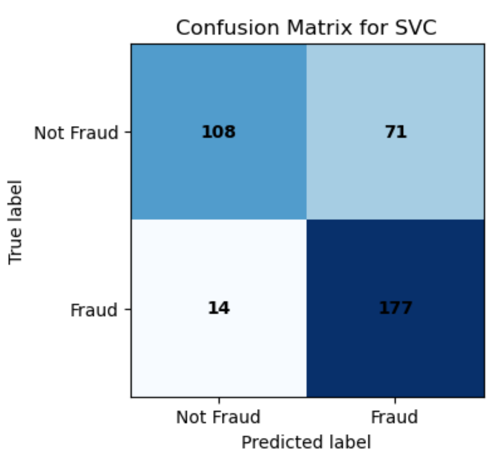
```python
# Display the plot
plt.show()
```

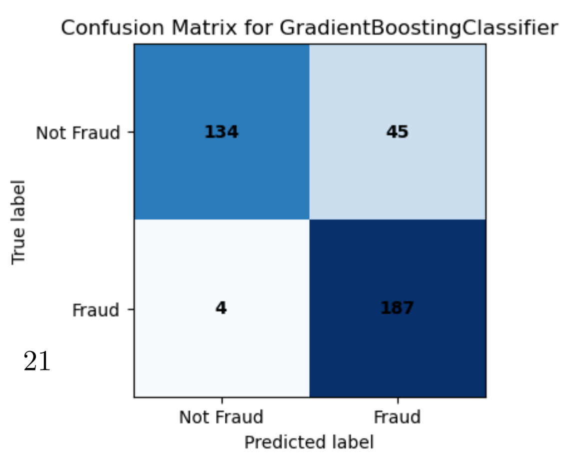

Confusion Matrix for LogisticRegression
Not Fraud
Fraud
True label
115
64
21
170
Not Fraud
Fraud
Predicted label


Confusion Matrix for RandomForestClassifier
Not Fraud
Fraud
True label
112
67
10
181
Not Fraud
Fraud
Predicted label


Confusion Matrix for DecisionTreeClassifier
Not Fraud
Fraud
True label
124
55
53
138
Not Fraud
Fraud
Predicted label


Confusion Matrix for MLPClassifier
Not Fraud
Fraud
True label
125
54
46
145
Not Fraud
Fraud
Predicted label


Confusion Matrix for SVC
Not Fraud
Fraud
True label
108
71
14
177
Not Fraud
Fraud
Predicted label


Confusion Matrix for GradientBoostingClassifier
Not Fraud
Fraud
True label
134
45
4
187
Not Fraud
Fraud
Predicted label

As we can see the best model is the Gradient Boosting Classifier with this parameters:

[20]:
```
best_model_params = grid_search.best_params_
print("Best Model Parameters:")
print(best_model_params)
```

```
Best Model Parameters:
{'learning_rate': 0.1, 'n_estimators': 300}
```