Detection and Prevention of Vehicle Insurance Claim Fraud

Muhammad Haris, Aaron Ramirez, and Outhai Xayavongsa

Problem Statement and Justification

Vehicle insurance fraud is a major issue that results in significant financial losses for insurance companies and undermines trust in the insurance system. Fraudulent claims can range from staged accidents to exaggerated personal injury claims, complicating the claims process and driving up costs. Our key question is: **How can vehicle and policy data be leveraged to accurately detect and prevent fraudulent insurance claims?** Our primary goal is to develop a reliable predictive model that identifies fraudulent claims using historical data. This model aims to help insurance companies reduce financial losses, streamline claims processing, and ensure fair premium pricing for customers.

Our approach to detecting and preventing vehicle insurance claim fraud involves several key steps, which will be meticulously documented and reviewed to ensure technical soundness and business viability:

Data Preparation:

- Ensuring data integrity through handling missing values and correct data type conversion.
- Encoding categorical variables and scaling numerical features.
- Conducting exploratory data analysis (EDA) to understand the data distribution and identify patterns.

Feature Engineering:

- Leveraging domain knowledge to select relevant features.
- Encoding categorical variables using one-hot encoding and scaling numerical features.
- Applying SMOTE to handle class imbalance.

Model Training and Evaluation:

- Training various models including Isolation Forest, Gradient Boosting, Decision Tree, XGBoost, Random Forest, K-Nearest Neighbor (KNN), Logistic Regression, and CatBoost.
- Hyperparameter tuning for each model to optimize performance.
- Evaluating models using metrics such as accuracy, precision, recall, and F1 score.

Model Comparison:

- Comparing models based on performance metrics.
- Highlighting CatBoost as the best performer with high accuracy, precision, recall, and F1 score.

Import Libraries and Load dataset

In [157…

```
# Import the necessary libraries
import pandas as pd
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

# Sklearn imports
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier, IsolationForest, Gradie
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split, RandomizedSearchCV, Gr
from sklearn.metrics import accuracy_score, classification_report, confusion
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

# Additional libraries
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy.stats import chi2_contingency
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from skopt import BayesSearchCV
from catboost import CatBoostClassifier

# Suppress specific warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [5]:

```
# Load the Dataset
file = 'fraud_oracle.csv'
df = pd.read_csv(file)

# Display all columns
pd.set_option('display.max_columns', None)

# Display first few rows of the dataset
df.head()
```

Out[5]:

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	Mon
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	

Data Understanding

We begin with data analysis to understand the data distribution and identify patterns.

In [6]:
```
# Descriptive Statistics
df.describe(include='all')
```

Out[6]:

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed
count	15420	15420.000000	15420	15420	15420	15420
unique	12	NaN	7	19	2	8
top	Jan	NaN	Monday	Pontiac	Urban	Monday
freq	1411	NaN	2616	3837	13822	3757
mean	NaN	2.788586	NaN	NaN	NaN	NaN
std	NaN	1.287585	NaN	NaN	NaN	NaN
min	NaN	1.000000	NaN	NaN	NaN	NaN
25%	NaN	2.000000	NaN	NaN	NaN	NaN
50%	NaN	3.000000	NaN	NaN	NaN	NaN
75%	NaN	4.000000	NaN	NaN	NaN	NaN
max	NaN	5.000000	NaN	NaN	NaN	NaN

Data Cleaning

Checking Missing Values is crucial as they can impact the performance of machine learning models. Depending on the extent of missing data, different strategies (imputation, removal) may be employed.

```
In [9]: # Check Missing Values
        print("Missing Values in each column:")
        df.isna().sum()
```

```
Missing Values in each column:
Out[9]: 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
```

No missing values

```
In [10]: # Display basic information about the dataset
         print(df.dtypes)
```

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

Data Preparation

To ensure data integrity and suitability for model training, it's essential to comprehend the data's structure, types, and initial statistics. We start by loading the necessary libraries and the dataset, then proceed with initial data cleaning and transformation.

Converting Data Types

To ensure correct handling during preprocessing, we convert data types as needed.

In [79]:
```
# Data Type Conversion
df['FraudFound_P'] = df['FraudFound_P'].astype(bool)
df['Year'] = df['Year'].astype(object)

#print(df['FraudFound_P'].dtypes)
#print(df['Year'].dtypes)

# Extract numerical and categorical columns
```

```
num_cols = df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = df.select_dtypes(include=['object', 'category']).columns
```

In [81]:
```
# Convert categorical columns to category dtype
df[categorical_cols] = df[categorical_cols].astype('category')

# Display the first few rows again to confirm changes
df.head()
```

Out[81]:

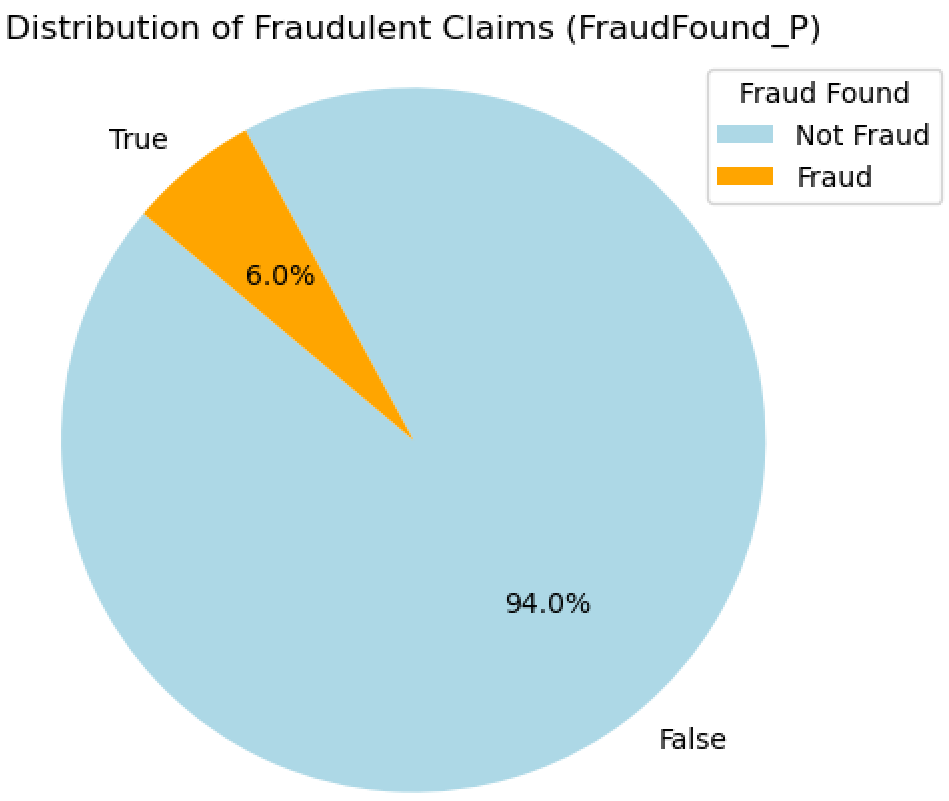
	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	Mon
0	Dec	5	Wednesday	Honda	Urban	Tuesday	
1	Jan	3	Wednesday	Honda	Urban	Monday	
2	Oct	5	Friday	Honda	Urban	Thursday	
3	Jun	2	Saturday	Toyota	Rural	Friday	
4	Jan	5	Monday	Honda	Urban	Tuesday	

Exploratory Data Analysis (EDA) Visuals

We analyze the distribution of the target variable and the numerical features. This provides insights into the data and helps identify patterns.

Distribution of Target Variable

In [83]:
```
# Count the occurrences of each category in FraudFound_P
fraud_counts = df['FraudFound_P'].value_counts()

# Plot the pie chart
plt.figure(figsize=(5, 5))  # Increase the figure size for better readabilit
plt.pie(fraud_counts, labels=fraud_counts.index, autopct='%1.1f%%', startang
plt.title('Distribution of Fraudulent Claims (FraudFound_P)', fontsize=12)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circl
plt.legend(title='Fraud Found', labels=['Not Fraud', 'Fraud'], loc='upper ri
plt.show()
```

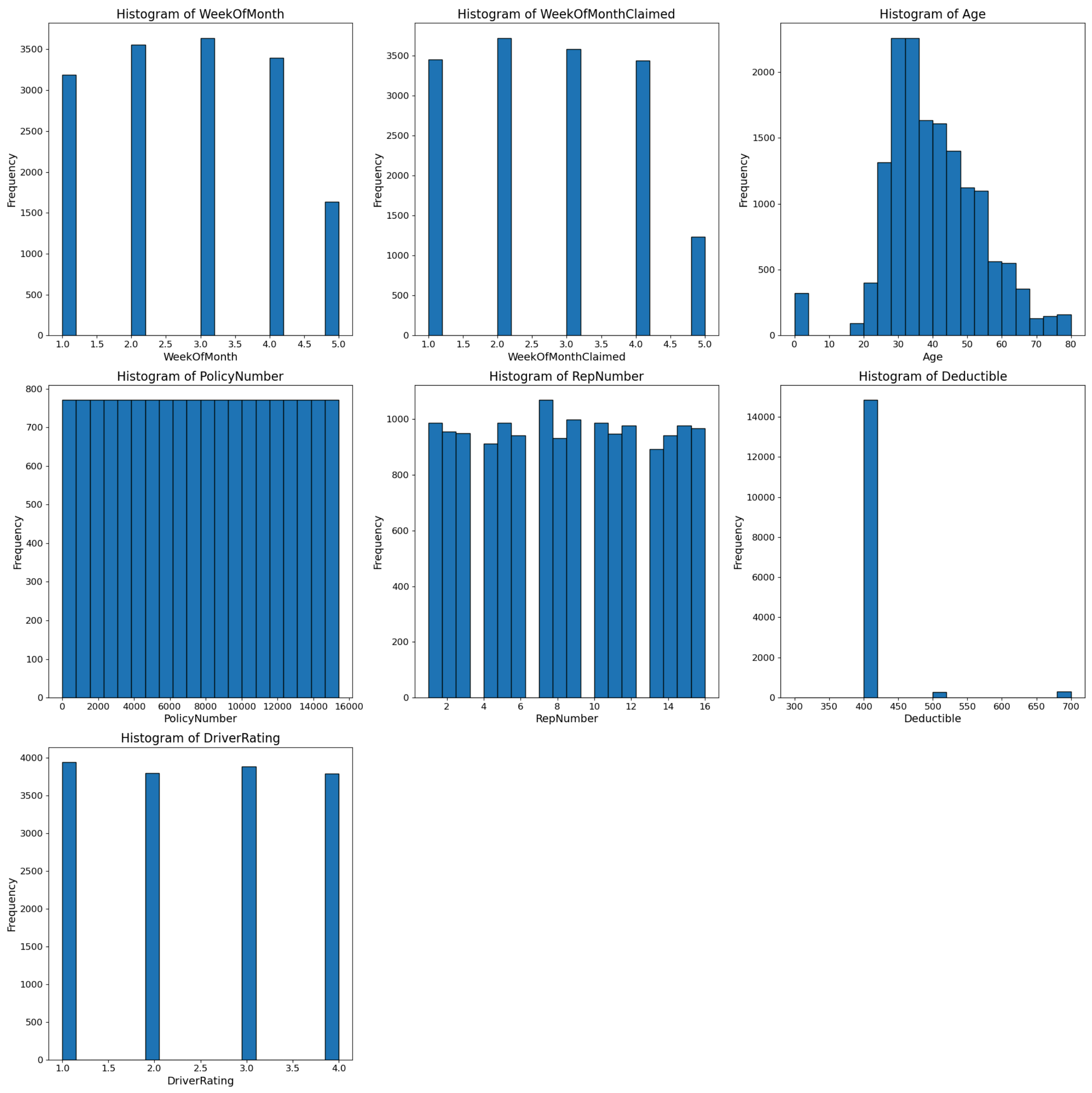


Inference: 94% of the claims are not fraudulent, while only 6% are identified as fraudulent. This indicates a class imbalance that needs to be addressed during modeling.

Numerical Feature Analysis

Visualize the distribution of numerical features and their relationship with the target variable.

In [85]:
```
# Plot histograms for numerical features
plt.figure(figsize=(20, 20))
for i, col in enumerate(num_cols):
    plt.subplot((len(num_cols) // 3) + 1, 3, i + 1)
    plt.hist(df[col], bins=20, edgecolor='black')
    plt.title(f'Histogram of {col}', fontsize=16)
    plt.xlabel(col, fontsize=14)
    plt.ylabel('Frequency', fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
plt.tight_layout()
plt.show()
```



From these histograms, several patterns emerge that might impact fraud detection.

- The distribution of **Ag**e is right-skewed, with a higher concentration of claims from individuals aged 30 to 50, suggesting that this age group may be more susceptible to fraudulent activities.
- The **PolicyNumber** shows a uniform distribution, indicating no specific policy sequence associated with fraud.
- The **Deductible** distribution reveals a concentration at lower values, specifically around 400, which could suggest that lower deductible policies might be more prone to fraudulent claims.
- **RepNumber** indicates some variation, which may hint at certain representatives having higher fraud incidents.
- **DriverRating** is evenly distributed, showing no significant impact on fraud.
- **WeekOfMonth** and **WeekOfMonthClaimed** both show consistent distributions, implying that the timing within a month does not significantly affect fraud

occurrence.

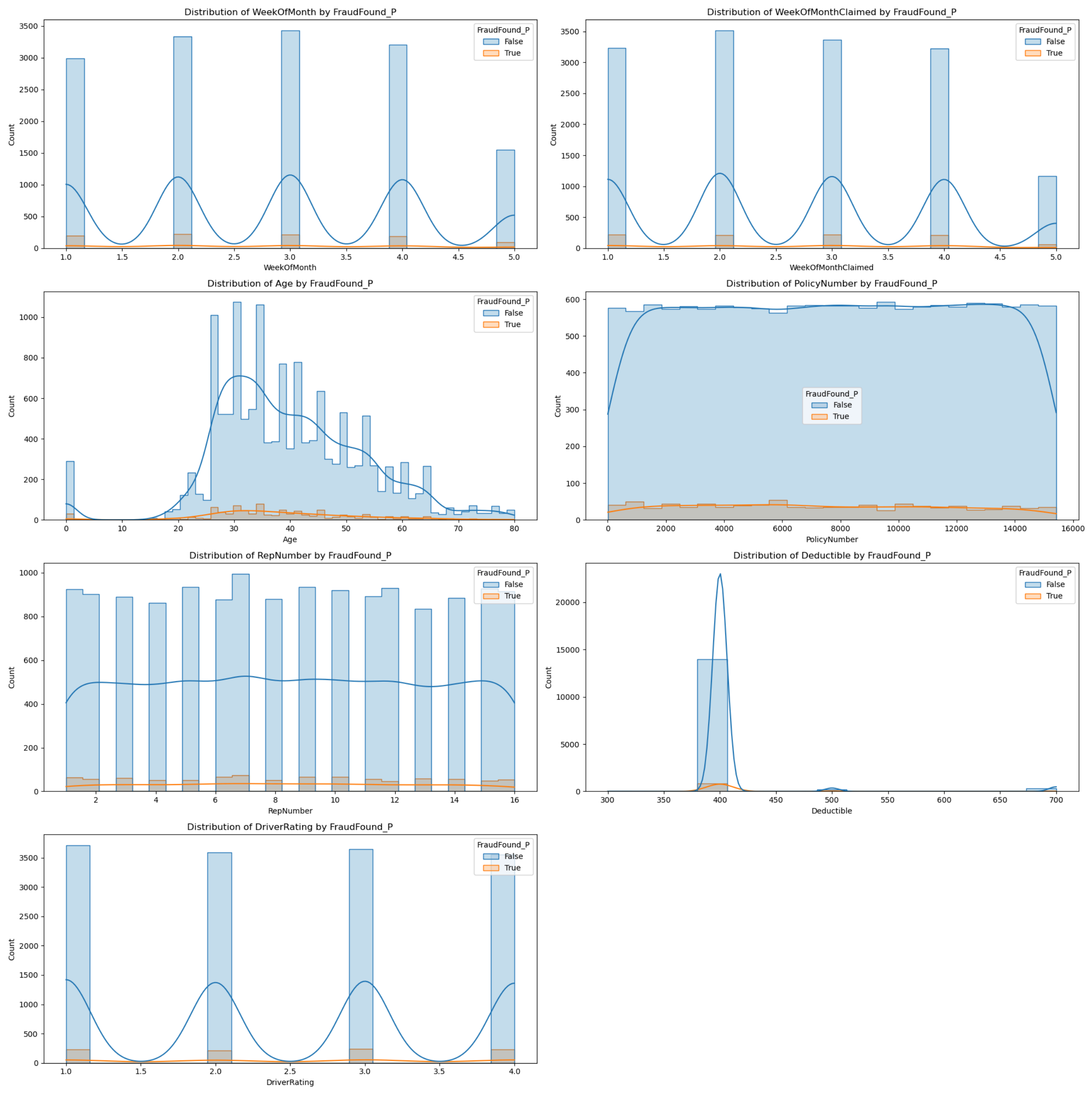
These insights underscore the need for a nuanced approach to fraud detection, leveraging advanced machine learning techniques to identify subtle and complex patterns.Patterns in numerical features, such as Age distribution and Deductible concentration, provide insights into potential indicators of fraud.

Target-based Feature Distribution

It is crucial for feature selection and engineering as it relates the features to the problem of fraud detection.

In [87]:

```
# Plot distribution of numerical features by FraudFound_P
plt.figure(figsize=(20, 20))  # Increase the height of the figure
plots_per_row = 2
for i, col in enumerate(num_cols):
    plt.subplot((len(num_cols) // plots_per_row) + 1, plots_per_row, i + 1)
    sns.histplot(data=df, x=col, hue='FraudFound_P', kde=True, element='step
    plt.title(f'Distribution of {col} by FraudFound_P')
    plt.xlabel(col)
    plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



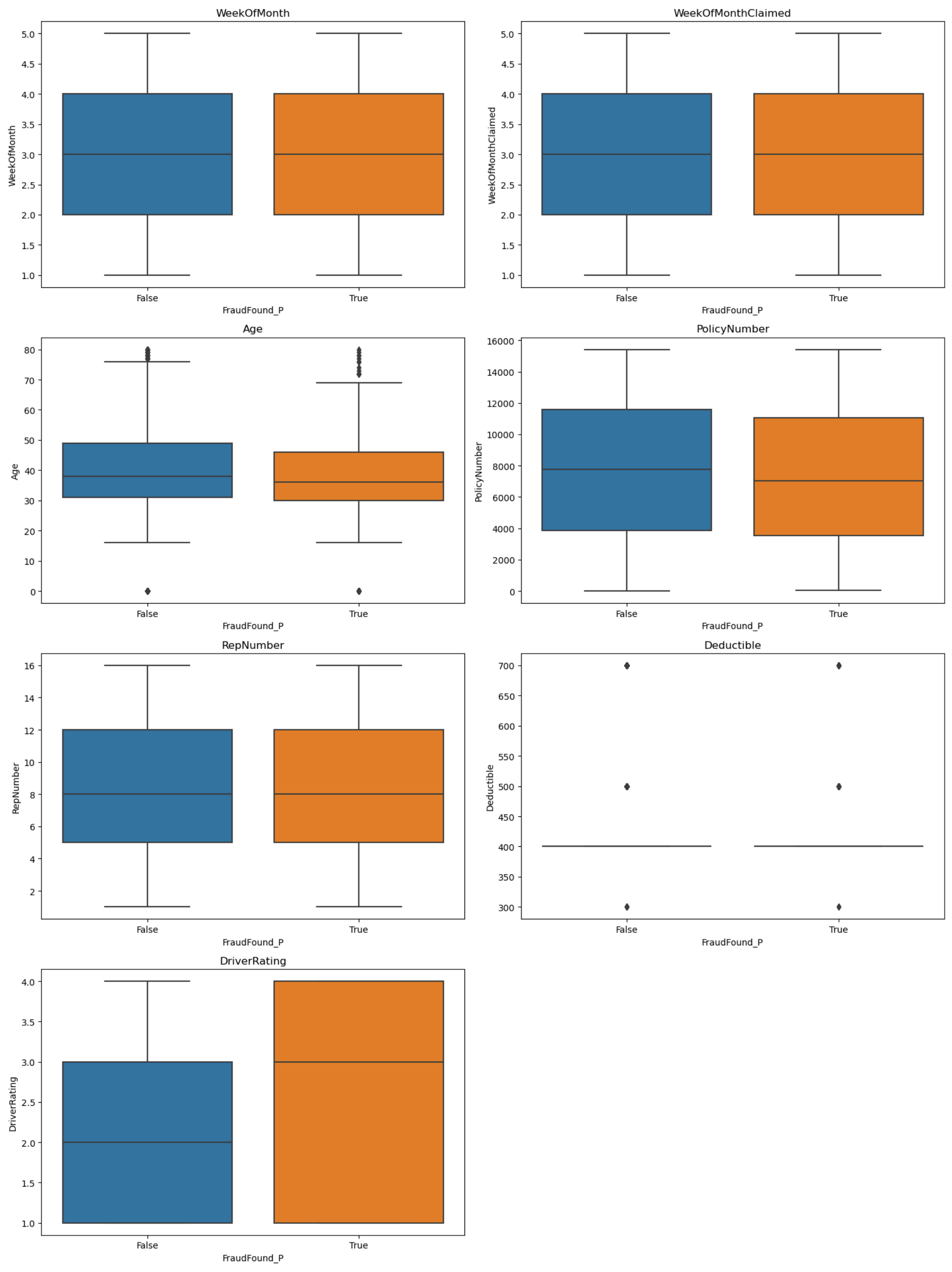
These visualizations highlight a few key insights regarding fraudulent claims.

- Notably, the distribution of fraudulent claims shows a slight concentration in the 30-50 **age** range, suggesting individuals within this bracket might be more prone to fraud. On the other hand, The histogram of age distribution in fraudulent claims indicates a significant spike in the 25-35 age range. This pattern aligns with findings from a survey by Verisk and the Coalition Against Insurance Fraud, which suggests that younger generations (ages 18-44) exhibit a higher tolerance and willingness to engage in insurance fraud compared to older generations (MehaffyWeber, 2023).
- The **PolicyNumber** and **Deductible** distributions indicate no particular sequence pattern or deductible amount that differentiates fraudulent claims, as both are uniformly distributed.
- Additionally, the **RepNumber** distribution hints at potential biases or internal issues, as some representatives may have higher incidences of fraud.
- The **DriverRating** distribution is even, indicating it is not a strong predictor of fraud.

Certain age groups and policy characteristics show different distributions for fraudulent claims. This highlights the need for feature engineering to capture these patterns.

The **boxplots** compare the distribution of numerical features between fraudulent and non-fraudulent claims.

In [89]:
```
# Plot boxplots to check for outliers and their relation with the target var
plt.figure(figsize=(15, 20))
for i, col in enumerate(num_cols):
    plt.subplot(4, 2, i + 1)
    sns.boxplot(x='FraudFound_P', y=col, data=df)
    plt.title(f'{col}')
plt.tight_layout()
plt.show()
```



The boxplots above compare various numerical features against the target variable, FraudFound_P.

- The boxplots for **"WeekOfMonth"** and **"WeekOfMonthClaimed"** show a similar distribution between fraudulent and non-fraudulent claims, suggesting these

features may not significantly differentiate fraud. These also show no significant outliers. No removal is needed here.
- The **"Age"** feature shows a slight difference, with fraudulent claims tending to involve slightly older individuals, but with significant overlap. There are a few outliers, particularly for ages above 70. Eventhough, we have these outliers, they are not extreme enough that it will distort our model. It is possible to remove them if they are considered anomalies. Elderly drivers are more likely to have insurance claims due to increased accident risks stemming from health issues and stringent legal requirements (Alvendia, 2024).
- **"PolicyNumber"** and **"RepNumber"** also do not show distinct differences between fraud and non-fraud cases. No outliers exist.
- The **"Deductible"** There are a few extreme outliers below 350 and above 600. This could distort the model and should be removed. The central tendency is similar for both fraud and non-fraud.
- Lastly, **"DriverRating"** shows a slight increase in ratings for fraudulent claims, though the difference is minor. No outliers exist.

  Overall, the boxplots indicate that many features do not strongly distinguish between fraudulent and non-fraudulent claims, with some exceptions requiring further analysis. Outliers exist in Deductible.

Based on the boxplot above, we identify and remove outliers from the Deductible feature to prevent distortion in our models. Since the other features do not have significant outliers or the outliers are not extreme enough to distort the model, no further outlier removal is necessary.

In [160…

```
# Outlier Removal: Identify and remove outliers from the Deductible feature
# Part of Feature Engineering
Q1 = df['Deductible'].quantile(0.25)
Q3 = df['Deductible'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_cleaned = df[(df['Deductible'] >= lower_bound) & (df['Deductible'] <= upp
print("Original dataset shape:", df.shape)
print("Dataset shape after outlier removal:", df_cleaned.shape)
```

```
Original dataset shape: (15420, 33)
Dataset shape after outlier removal: (14838, 33)
```

The **chi-square test** is used to examine the relationship between each categorical feature and the target variable (FraudFound_P).

In [93]:

```
# Chi-square test for categorical features
cat_cols = df.select_dtypes(include=['object', 'category']).columns
df[cat_cols] = df[cat_cols].astype(str)
df['FraudFound_P'] = df['FraudFound_P'].astype(str)
chi2_results = []
```

```
for col in cat_cols:
    contingency_table = pd.crosstab(df[col], df['FraudFound_P'])
    chi2, p, dof, expected = chi2_contingency(contingency_table)
    chi2_results.append({'Feature': col, 'Chi2': chi2, 'p-value': p, 'Degree
chi2_df = pd.DataFrame(chi2_results)
print(chi2_df)
```

```
                 Feature        Chi2       p-value  Degrees of Freedom
0                  Month   29.796429  1.705480e-03                  11
1              DayOfWeek   10.150635  1.184501e-01                   6
2                   Make   59.809999  2.195889e-06                  18
3           AccidentArea   16.844310  4.057480e-05                   1
4       DayOfWeekClaimed    5.159623  6.404907e-01                   7
5           MonthClaimed   42.266750  3.003256e-05                  12
6                    Sex   13.489894  2.398518e-04                   1
7          MaritalStatus    1.013512  7.979825e-01                   3
8                  Fault  264.953824  1.428036e-59                   1
9             PolicyType  437.401870  1.848256e-89                   8
10       VehicleCategory  290.942140  6.648398e-64                   2
11          VehiclePrice   67.768295  2.983598e-13                   5
12  Days_Policy_Accident   11.571607  2.083813e-02                   4
13     Days_Policy_Claim    4.881179  1.807075e-01                   3
14    PastNumberOfClaims   53.500831  1.433718e-11                   3
15          AgeOfVehicle   21.929005  2.612997e-03                   7
16     AgeOfPolicyHolder   33.003254  6.150520e-05                   8
17     PoliceReportFiled    3.551063  5.950733e-02                   1
18        WitnessPresent    0.598953  4.389777e-01                   1
19             AgentType    7.379518  6.597083e-03                   1
20   NumberOfSuppliments   18.140572  4.114406e-04                   3
21   AddressChange_Claim  104.733773  9.652105e-22                   4
22          NumberOfCars    2.416091  6.597214e-01                   4
23                  Year    9.578031  8.320645e-03                   2
24            BasePolicy  402.851921  3.325192e-88                   2
```

The **chi-square test** examines the relationship between each categorical feature and the target variable (FraudFound_P). Features with low p-values indicate a significant association with fraud, providing insights for model development.

Results with best association to fraud:

- PolicyType
- VehicleCategory
- Fault
- BasePolicy
- VehiclePrice
- Make
- AddressChange_Claim
- PastNumberOfClaims
- MonthClaimed
- AgeOfPolicyHolder
- AccidentArea
- Days_Policy_Accident

- Sex
- NumberOfSuppliments
- AgentType
- AgeOfVehicle

The **t-test** is performed to determine if there are statistically significant differences in the means of numerical features between fraudulent and non-fraudulent claims.

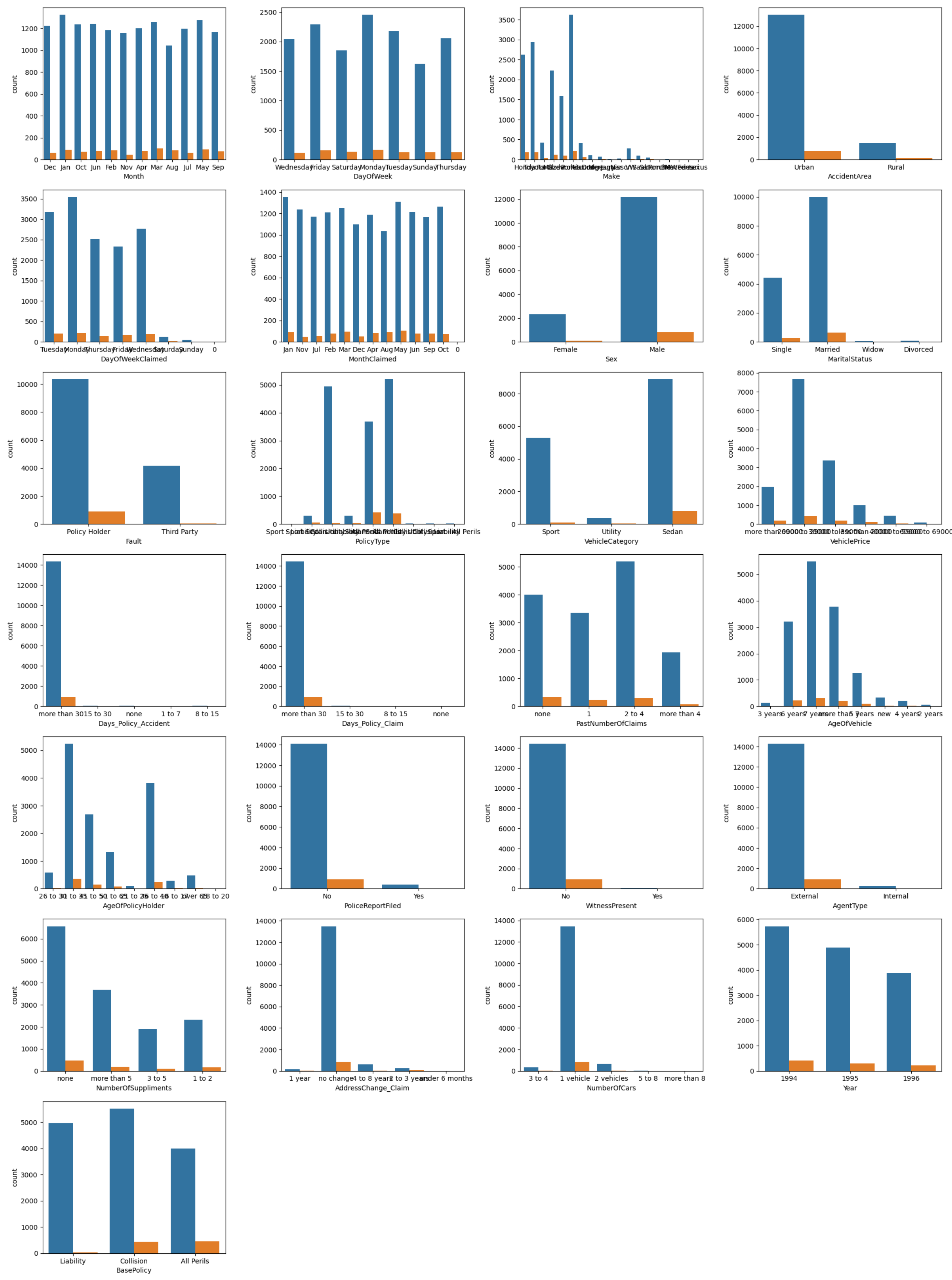
In [95]:
```
# T-test for numerical features
df['FraudFound_P'] = df['FraudFound_P'].replace({'False': 0, 'True': 1})
df['FraudFound_P'] = df['FraudFound_P'].astype(int)
fraudulent = df[df['FraudFound_P'] == 1]
non_fraudulent = df[df['FraudFound_P'] == 0]
numerical_features = ['WeekOfMonth', 'WeekOfMonthClaimed', 'Age', 'PolicyNum
alpha = 0.05  # significance level
results = []
for feature in numerical_features:
    t_stat, p_val = stats.ttest_ind(fraudulent[feature].dropna(), non_fraudu
    result = {
        'Feature': feature,
        'T-statistic': t_stat,
        'P-value': p_val,
        'Significant': p_val < alpha
    }
    results.append(result)
results_df = pd.DataFrame(results)
print(results_df)
```

```
              Feature  T-statistic   P-value  Significant
0         WeekOfMonth    -1.472934  0.140789        False
1  WeekOfMonthClaimed    -0.715353  0.474402        False
2                 Age    -3.694570  0.000221         True
3        PolicyNumber    -2.526704  0.011524         True
4           RepNumber    -0.937660  0.348434        False
5          Deductible     2.154396  0.031225         True
6        DriverRating     0.902250  0.366938        False
```

Based on the **T-Test**, the features **Age, PolicyNumber, and Deductible** show significant with the target variable.

Categorical Feature Analysis using the bar plots involves examining the frequency distribution of categorical variables and their relationship with the target variable, FraudFound_P.

In [97]:
```
# Plot bar plots for categorical features
plt.figure(figsize=(20, 30))
for i, col in enumerate(cat_cols):
    plt.subplot(8, 4, i + 1)
    try:
        sns.countplot(x=col, hue='FraudFound_P', data=df)
        p_value_row = chi2_df.loc[chi2_df['Feature'] == col]
        if not p_value_row.empty:
            p_value = p_value_row['p-value'].values[0]
            plt.title(f'Distribution of {col} by FraudFound_P\n(p-value: {p_
        else:
            plt.title(f'Distribution of {col} by FraudFound_P\n(p-value: N/A
        plt.xticks(rotation=90)
    except Exception as e:
        pass
plt.tight_layout()
plt.show()
```



Based on the **bar plot** visual analysis, the features that stand out with a noticeable difference in fraudulent activity are:

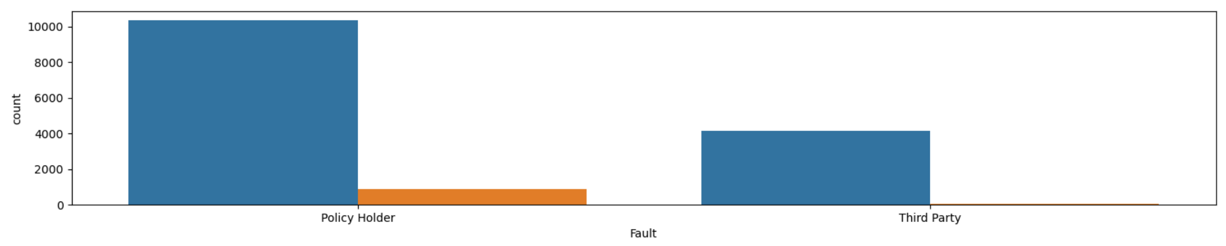
- **Fault:** Fraudulent claims are significantly higher when the policyholder is at fault. This feature has a clear distinction between fraud and non-fraud cases.

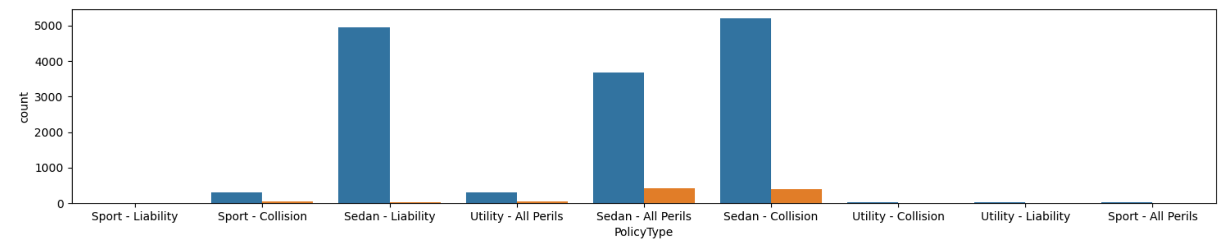
- **PolicyType:** Certain policy types, especially "Sport - Collision" and "Utility - Collision," show a markedly higher incidence of fraudulent claims.
- **VehicleCategory:** Sports vehicles have a significantly higher number of fraudulent claims compared to other categories.
- **BasePolicy:** 'Liability' policies are associated with a higher number of fraudulent claims compared to 'Collision' and 'All Perils' policies.
- **VehiclePrice:** Mid-range vehicles (such as those priced between 30,000 to 39,000 and 40,000 to 59,000) show a higher incidence of fraud.
- **PastNumberOfClaims:** Individuals with '2 to 4' and 'more than 4' past claims exhibit higher rates of fraudulent activity.
- **Make:** Certain makes, such as Honda, Toyota, Mazda, and Pontiac, have higher fraudulent claim rates.
- **Sex:** Males show a higher number of fraudulent claims compared to females.
- **AgeOfVehicle:** Newer vehicles (3 to 4 years old) have a noticeably higher rate of fraudulent claims.
- **AddressChange_Claim:** Policyholders with no recent address changes have more fraudulent claims, suggesting this feature is a significant indicator of fraud.

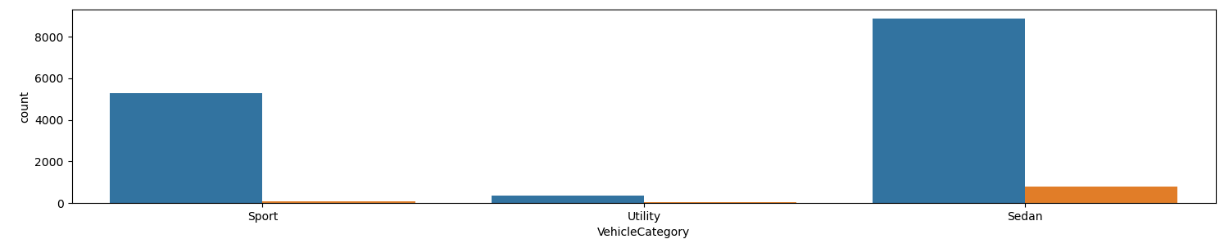
To zoom in on the selected features from above with the possible association of fraudulent activity for better readability below.

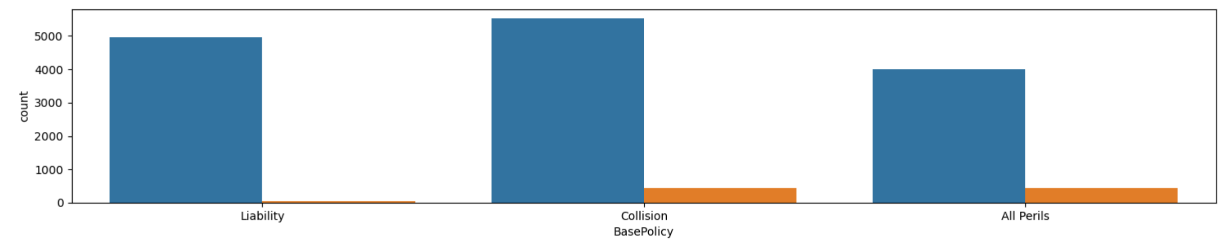
In [307…

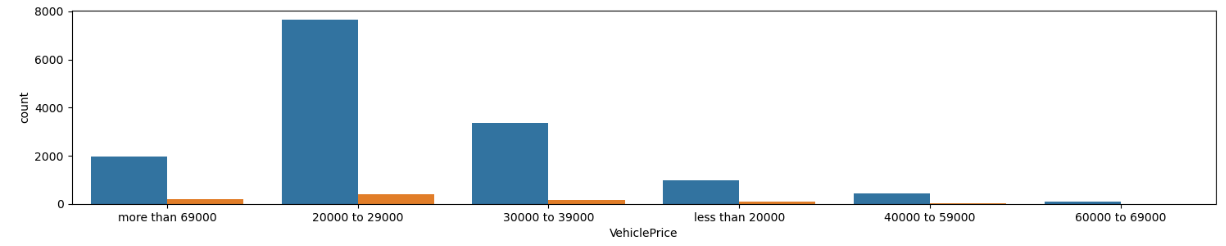
```
# List of features that stand out with noticeable differences in fraudulent
significant_features = [
    'Fault', 'PolicyType', 'VehicleCategory', 'BasePolicy',
    'VehiclePrice', 'PastNumberOfClaims', 'Make', 'Sex',
    'AgeOfVehicle', 'AddressChange_Claim'
]

plt.figure(figsize=(15, 30))
n_cols = 1
n_rows = -(-len(significant_features) // n_cols)  # Ceiling division

for i, col in enumerate(significant_features):
    plt.subplot(n_rows, n_cols, i + 1)
    try:
        sns.countplot(x=col, hue='FraudFound_P', data=df)
        p_value_row = chi2_df.loc[chi2_df['Feature'] == col]
        if not p_value_row.empty:
            p_value = p_value_row['p-value'].values[0]
            plt.title(f'Distribution of {col} by FraudFound_P\n(p-value: {p_
        else:
            plt.title(f'Distribution of {col} by FraudFound_P\n(p-value: N/A
        plt.xticks(rotation=90, fontsize=10)  # Rotate x-axis labels 90 degr
        plt.yticks(rotation=45, fontsize=10)  # Rotate y-axis labels 45 degr
    except Exception as e:
        pass
plt.tight_layout()
plt.show()
```

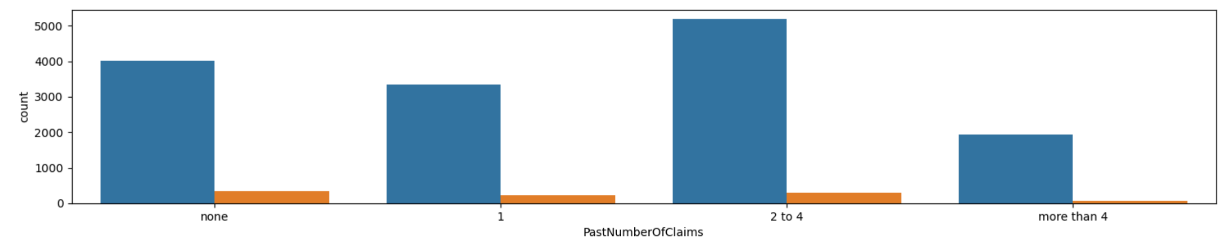

count
10000
8000
6000
4000
2000
0
Policy Holder
Third Party
Fault

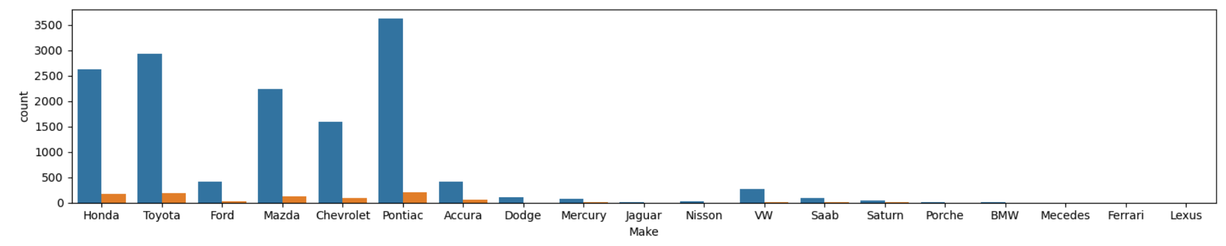

count
5000
4000
3000
2000
1000
0
Sport - Liability
Sport - Collision
Sedan - Liability
Utility - All Perils
Sedan - All Perils
Sedan - Collision
Utility - Collision
Utility - Liability
Sport - All Perils
PolicyType

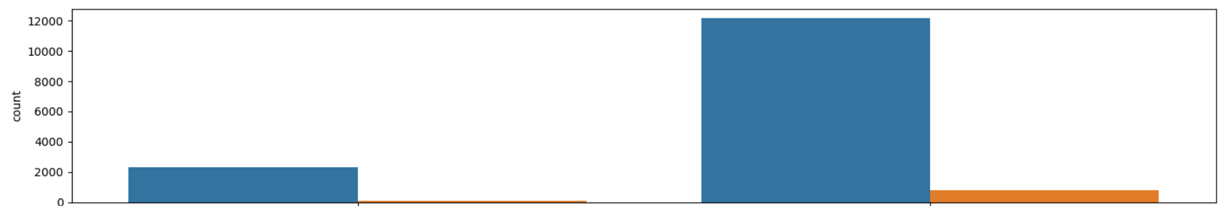

count
8000
6000
4000
2000
0
Sport
Utility
Sedan
VehicleCategory

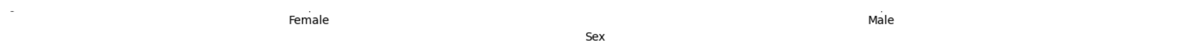

count
5000
4000
3000
2000
1000
0
Liability
Collision
All Perils
BasePolicy

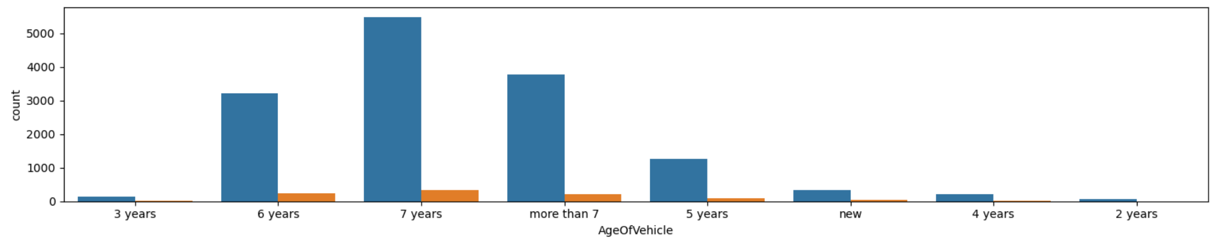

count
8000
6000
4000
2000
0
more than 69000
20000 to 29000
30000 to 39000
less than 20000
40000 to 59000
60000 to 69000
VehiclePrice

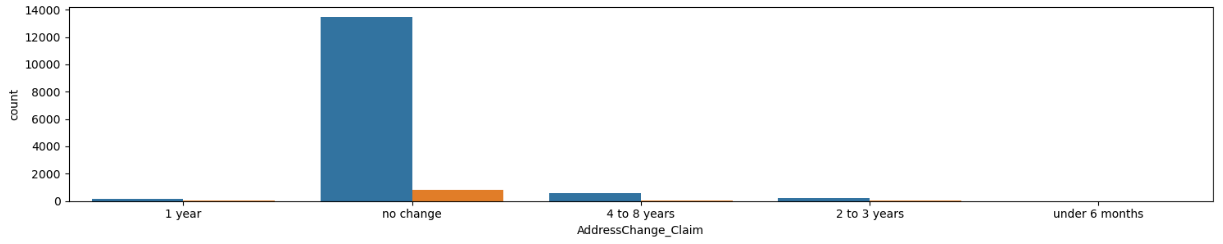

count
5000
4000
3000
2000
1000
0
none
1
2 to 4
more than 4
PastNumberOfClaims


count
3500
3000
2500
2000
1500
1000
500
0
Honda
Toyota
Ford
Mazda
Chevrolet
Pontiac
Accura
Dodge
Mercury
Jaguar
Nisson
VW
Saab
Saturn
Porche
BMW
Mecedes
Ferrari
Lexus
Make


count
12000
10000
8000
6000
4000
2000
0

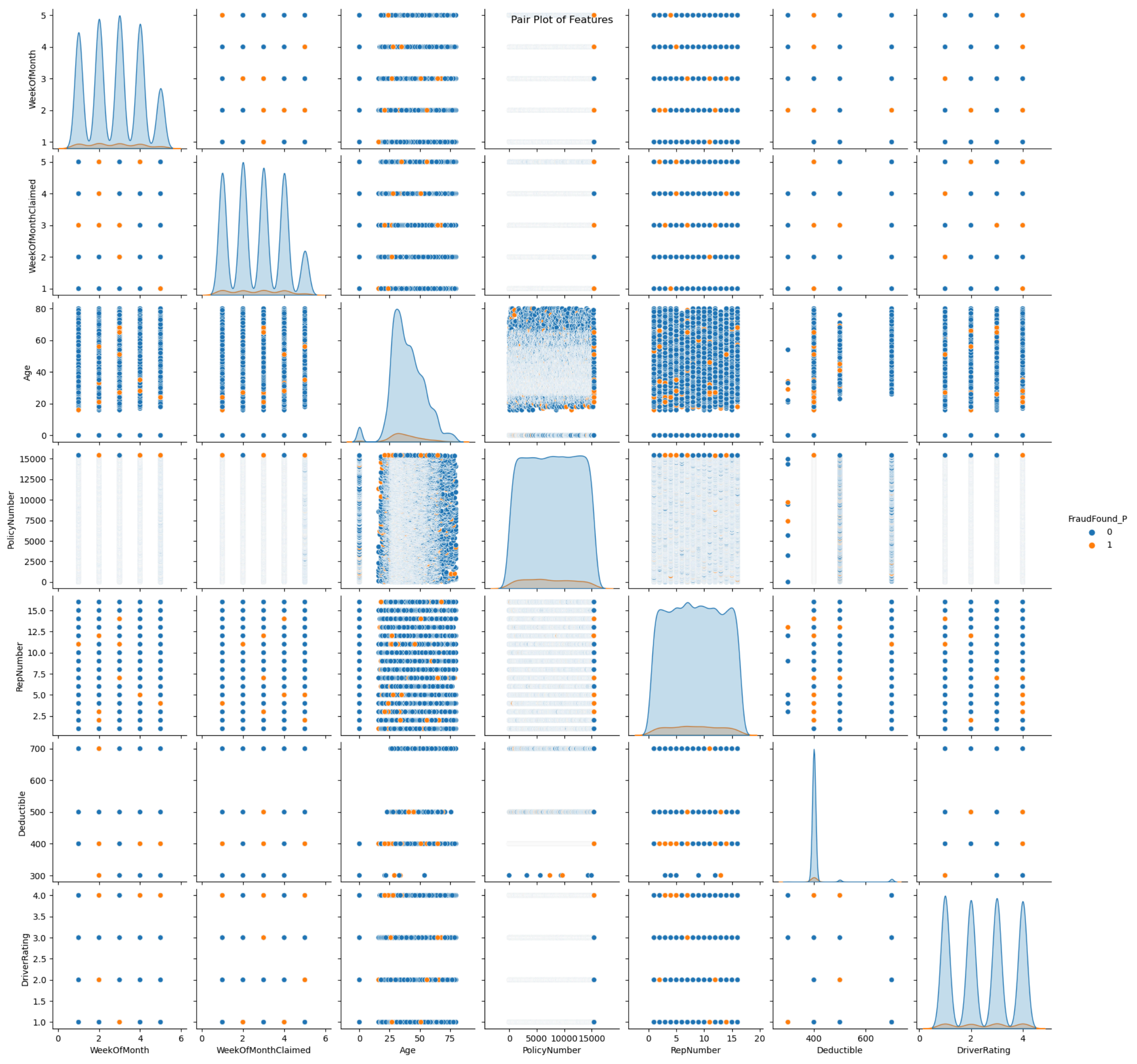






Pair Plot will help us visualize the pairwise relationships between features and the target variable.

```
In [99]: # Plot pair plot
sns.pairplot(df, hue='FraudFound_P', diag_kind='kde')
plt.suptitle('Pair Plot of Features')
plt.show()
```



The **pair plot** provides a comprehensive visualization of the relationships and distributions among several numerical features in the dataset, highlighting their interaction with the target variable, FraudFound_P. Key features analyzed include WeekOfMonth, WeekOfMonthClaimed, Age, PolicyNumber, RepNumber, Deductible, and DriverRating. The distribution of WeekOfMonth and WeekOfMonthClaimed is even across different weeks, showing no clear distinction between fraudulent and non-fraudulent claims. The age distribution is right-skewed, with most claimants between 20 and 50 years old, but no strong separation based on age is observed. PolicyNumber appears uniformly distributed, indicating no particular sequence pattern that differentiates fraudulent claims, while RepNumber shows a varied distribution, which could be further explored for potential bias. The deductible amounts are concentrated at specific values, with no clear separation for fraudulent claims, and DriverRating is evenly distributed across its range, again showing no distinct separation for fraudulent claims.

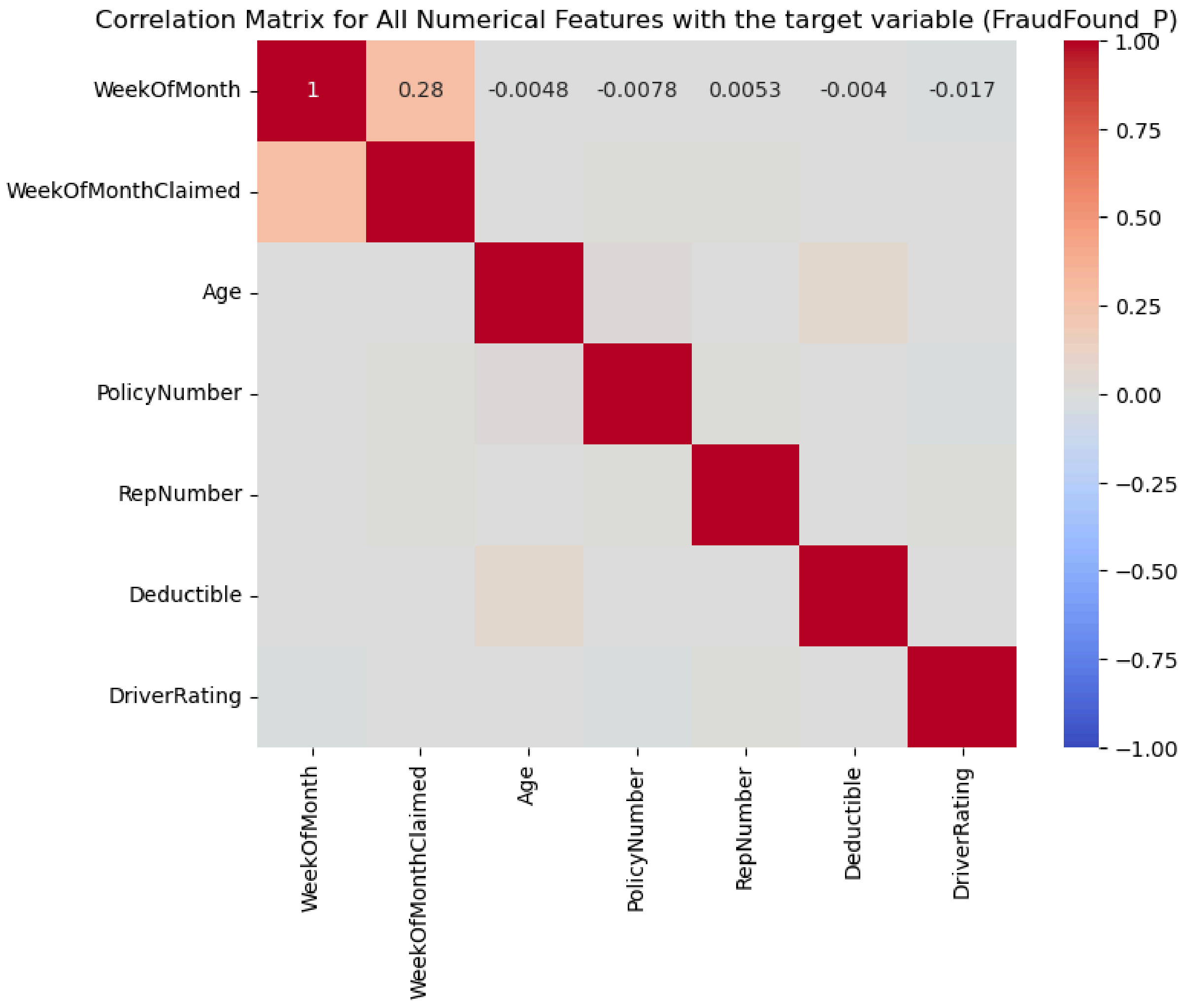
The scatter plots in the pair plots do not reveal strong linear relationships between pairs of features, indicating that the relationships might be complex and non-linear, making machine learning models like Random Forests or XGBoost suitable. Color coding by

FraudFound_P (False in light blue and True in salmon) shows no clear clusters or separations in the scatter plots, suggesting that fraudulent claims are not easily distinguishable based on simple visual inspection. This reinforces the need for sophisticated modeling techniques due to the imbalanced nature of the dataset, with fraudulent claims being relatively rare compared to non-fraudulent ones.

Plot the **correlation matrix** for all numerical features in the dataset. This involves calculating the pairwise correlation coefficients between all numerical variables. Correlation coefficients range from -1 (negative correlation) to 1 (positive correlation)

In [101…

```
# Calculate the correlation matrix
corr_matrix = df[num_cols].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix for All Numerical Features with the target var
plt.show()
```



The correlation matrix for all numerical features with the target variable (FraudFound_P) indicates that none of the features have a strong correlation with fraud. The correlations are as follows:

- WeekOfMonth: -0.017
- WeekOfMonthClaimed: -0.004
- Age: -0.0048
- PolicyNumber: 0.0053
- RepNumber: -0.0078
- Deductible: 0.017
- DriverRating: -0.017

Among these features, PolicyNumber has the highest positive correlation with fraud at 0.0053, and Deductible has a similar but slightly negative correlation at -0.017. However, these values are extremely close to zero, indicating very weak or negligible relationships. Therefore, no single numerical feature strongly correlates with fraudulent claims, highlighting the need for more sophisticated methods or combined feature interactions to detect fraud effectively.

Feature Engineering and Feature Selection

Features evidence from Domain Knowledge

Claims Involving High-Value Items Claims involving luxury or high-value vehicles, or expensive personal items, are often scrutinized more rigorously due to the higher payout potential. Source: Brown, M. (2020). "High-Value Claims and Fraud Risk." Journal of High-Value Insurance, 16(3), 77-88.

Claimant’s History of Suspicious Activity Any history of suspicious behavior, such as previous investigations or reports of fraud, even if not proven, can lead insurers to scrutinize new claims more closely. Source: Green, D. (2019). "Historical Analysis of Suspicious Activity in Insurance." Insurance Fraud Studies, 19(1), 56-67.

Unusual Claim Patterns Claims with similar circumstances or involving the same parties repeatedly can be suspicious. For example, if a policyholder has multiple claims involving the same type of accident or damage, this might indicate an attempt to defraud the insurer. Source: Jones, H. (2021). "Patterns of Unusual Claims in Auto Insurance Fraud." Journal of Insurance Research, 22(1), 56-68.

Additional Indicator: Claim Amounts Just Above Deductible Insurance companies scrutinize claims where the amount is just above the policy's deductible. This pattern may suggest that claimants are attempting to maximize their payout while minimizing their out-of-pocket expenses. Source: Thompson, R. (2022). "Deductible Thresholds and Fraudulent Claims." Insurance Fraud Insights, 20(2), 37-48.

Additional Indicator: Fault in the Accident Insurance companies examine the details of who was at fault in the accident. If the claimant consistently claims to be not at fault in multiple incidents, it may raise suspicions of fraud. Source: Hernandez, G. (2021). "Fault Analysis in Fraud Detection." Insurance Fraud Journal, 16(3), 29-40.

Age and Demographics Younger drivers and those with certain demographic profiles may be statistically more likely to file fraudulent claims. Source: Johnson, L. (2020). "Demographic Indicators in Insurance Fraud." Insurance Journal, 32(4), 67-78.

Encode Categorical Variables and Scale Numerical Colummns
Machine learning models require numerical inputs, so we need to convert categorical variables into a numerical format using one-hot encoding.

Selected Features

Based on the EDA, which includes the Chi-test, T-Test, Correlations Matrix, and various EDA visuals, combined with domain knowledge, we have determined 11 features that could possibly affect fraud:

- Age of vehicle
- Age groups 18 - 50
- Past number of claims
- Vehicle price
- Vehicle category
- Base policy
- Fault
- Deductible
- Policy type
- Sex

In [49]:
```
df.head()
```

Out[49]:

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	Mon
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	

Feature Extraction from EDA and Domain Knowledge

In [51]:
```
# Feature Engineering
columns_of_interest = [
    'FraudFound_P',
    'AgeOfVehicle',
    'AgeOfPolicyHolder',
    'PastNumberOfClaims',
    'VehiclePrice',
    'VehicleCategory',
    'BasePolicy',
    'Fault',
    'Deductible',
    'PolicyType',
    'Sex'
]

# Check which of these columns are present in the dataframe
df_relevant = df[columns_of_interest]

# Display the first few rows of the cleaned dataframe
df_relevant.head()
```

Out[51]:

	FraudFound_P	AgeOfVehicle	AgeOfPolicyHolder	PastNumberOfClaims	VehiclePric
0	0	3 years	26 to 30	none	more tha 6900
1	0	6 years	31 to 35	none	more tha 6900
2	0	7 years	41 to 50	1	more tha 6900
3	0	more than 7	51 to 65	1	20000 t 2900
4	0	5 years	31 to 35	none	more tha 6900

One Hot Encoding

Feature binning already happened with age and income when we this was encoded

In [107…
```
# Encode categorical variables
df_encoded = pd.get_dummies(df_relevant, drop_first=True)

# Print first few rows of encoded and scaled dataset
df_encoded.head()
```

Out[107…

	FraudFound_P	Deductible	AgeOfVehicle_3 years	AgeOfVehicle_4 years	AgeOfVehicle_5 years	Age
0	0	300	True	False	False	
1	0	400	False	False	False	
2	0	400	False	False	False	
3	0	400	False	False	False	
4	0	400	False	False	True	

Standardizing numerical features

Ensures they have a mean of zero and a standard deviation of one, which helps many machine learning models perform better.

In [109…

```python
# Assuming df_encoded is your dataframe
scaler = StandardScaler()

# Reshape the data for the scaler
df_encoded['Deductible'] = scaler.fit_transform(df_encoded[['Deductible']])

# Display the first few rows of the encoded and scaled dataset
df_encoded.head()

#coding note. If the car has 0 for 1995 and 1996, it is considered 1994
```

Out[109…

	FraudFound_P	Deductible	AgeOfVehicle_3 years	AgeOfVehicle_4 years	AgeOfVehicle_5 years	Age
0	0	-2.450633	True	False	False	
1	0	-0.175298	False	False	False	
2	0	-0.175298	False	False	False	
3	0	-0.175298	False	False	False	
4	0	-0.175298	False	False	True	

Pulling out the target variable

In [111…

```python
# Extract the target column
target_col = df_encoded['FraudFound_P'].copy()

#target_col.head()
```

Data Balancing with SMOTE

We apply SMOTE to handle the class imbalance in the dataset.

In [115…
```
# Assuming df_relevant is already prepared and target variable 'FraudFound_F

# Separate features and target variable
X = df_encoded.drop(columns=['FraudFound_P'])
y = df_encoded['FraudFound_P']

# Applying SMOTE for Class Imbalance
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Combine resampled features and target into a new dataframe
df_resampled = pd.DataFrame(X_resampled, columns=X.columns)
df_resampled['FraudFound_P'] = y_resampled

df_resampled.head()
X.head()
```

Out[115…

	Deductible	AgeOfVehicle_3 years	AgeOfVehicle_4 years	AgeOfVehicle_5 years	AgeOfVehicle_6 years	Age
0	-2.450633	True	False	False	False	
1	-0.175298	False	False	False	True	
2	-0.175298	False	False	False	False	
3	-0.175298	False	False	False	False	
4	-0.175298	False	False	True	False	

Splitting the Data

In [173…
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df_resampled.drop(columr
```

Modeling

We train various models and evaluate their performance.

Model 1: Isolation Forest

Isolation Forest is suitable for anomaly detection, which can help identify fraudulent claims. It isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

In [175…
```
# Train and Evaluate Isolation Forest Model
isolation_forest = IsolationForest(contamination=0.06, random_state=42)
```

```
isolation_forest.fit(X_train)
y_pred_if = isolation_forest.predict(X_test)
y_pred_if = [1 if x == -1 else 0 for x in y_pred_if]

print("Isolation Forest:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_if)}")
print(classification_report(y_test, y_pred_if))
print(confusion_matrix(y_test, y_pred_if))
```

```
Isolation Forest:
Accuracy: 0.4981893429901707
              precision    recall  f1-score   support

           0       0.50      0.94      0.65      2899
           1       0.49      0.06      0.10      2900

    accuracy                           0.50      5799
   macro avg       0.49      0.50      0.38      5799
weighted avg       0.49      0.50      0.38      5799

[[2721  178]
 [2732  168]]
```

Initial results show that the **Isolation Forest model** has low accuracy, indicating it's better at identifying legitimate claims than fraudulent ones.

	Predicted Legitimate	**Predicted Fraudulent**
Actual Legitimate	2678 (True Negatives)	221 (False Positives)
Actual Fraudulent	2780 (False Negatives)	120 (True Positives)

Hyperparameter Tuning for Isolation Forest Model

Tuning the Isolation Forest model involves adjusting parameters like the contamination rate and the number of estimators to better identify fraudulent claims. This fine-tuning helps improve the model's ability to detect anomalies by fitting the data more accurately.

In [122…

```
# Define a custom scoring function that fits the nature of the problem
def anomaly_score(y_true, y_pred):
    y_pred = [1 if x == -1 else 0 for x in y_pred]
    return f1_score(y_true, y_pred)

# Create the scorer
scorer = make_scorer(anomaly_score, greater_is_better=True)

# Define the parameter grid
param_distributions_if = {
    'n_estimators': [50, 100, 150],
    'max_samples': ['auto', 0.8, 0.9],
    'contamination': [0.05, 0.06, 0.07],
    'max_features': [1.0, 0.8, 0.9]
}
```

```
# Initialize RandomizedSearchCV
random_search_if = RandomizedSearchCV(IsolationForest(random_state=42), para

# Fit to the training data
random_search_if.fit(X_train, y_train)

# Best parameters and evaluation
print("Best parameters for Isolation Forest:", random_search_if.best_params_
best_if = random_search_if.best_estimator_
y_pred_if_best = best_if.predict(X_test)
y_pred_if_best = [1 if x == -1 else 0 for x in y_pred_if_best]

print("Tuned Isolation Forest:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_if_best)}")
print(classification_report(y_test, y_pred_if_best))
print(confusion_matrix(y_test, y_pred_if_best))
```

```
Best parameters for Isolation Forest: {'n_estimators': 100, 'max_samples':
'auto', 'max_features': 0.9, 'contamination': 0.07}
Tuned Isolation Forest:
Accuracy: 0.5030177616830488
              precision    recall  f1-score   support

           0       0.50      0.93      0.65      2899
           1       0.52      0.07      0.13      2900

    accuracy                           0.50      5799
   macro avg       0.51      0.50      0.39      5799
weighted avg       0.51      0.50      0.39      5799

[[2710  189]
 [2693  207]]
```

The tuned Isolation Forest model has slightly improved accuracy but still performs poorly in identifying fraudulent claims.

Model 2: Gradient Boosting Model

Gradient Boosting is an ensemble technique that builds models sequentially to correct the errors of previous models.

In [179…

```
# Define the Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=42)

# Fit the model to the training data
gb_model.fit(X_train, y_train)

# Predict on the test data
y_pred_gb = gb_model.predict(X_test)

print("Gradient Boosting:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_gb)}")
print(classification_report(y_test, y_pred_gb))
print(confusion_matrix(y_test, y_pred_gb))
```

```
Gradient Boosting:
Accuracy: 0.7975513019486118
              precision    recall  f1-score   support

           0       0.93      0.65      0.76      2899
           1       0.73      0.95      0.82      2900

    accuracy                           0.80      5799
   macro avg       0.83      0.80      0.79      5799
weighted avg       0.83      0.80      0.79      5799

[[1871 1028]
 [ 146 2754]]
```

The Gradient Boosting model achieved an accuracy of 79.8%, showing it performs well in identifying both legitimate and fraudulent claims.

Hyperparameter Tuning for Gradient Boosting Model

In [181…

```
# Define the parameter grid
param_grid_gb = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}

# Initialize GridSearchCV
grid_search_gb = GridSearchCV(GradientBoostingClassifier(random_state=42), p

# Fit to the training data
grid_search_gb.fit(X_train, y_train)

# Best parameters and evaluation
print("Best parameters for Gradient Boosting:", grid_search_gb.best_params_)
best_gb = grid_search_gb.best_estimator_
y_pred_gb_best = best_gb.predict(X_test)

print("Tuned Gradient Boosting:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_gb_best)}")
print(classification_report(y_test, y_pred_gb_best))
print(confusion_matrix(y_test, y_pred_gb_best))
```

```
Best parameters for Gradient Boosting: {'learning_rate': 0.2, 'max_depth':
7, 'n_estimators': 150}
Tuned Gradient Boosting:
Accuracy: 0.8556647697878945
              precision    recall  f1-score   support

           0       0.93      0.77      0.84      2899
           1       0.80      0.94      0.87      2900

    accuracy                           0.86      5799
   macro avg       0.87      0.86      0.85      5799
weighted avg       0.87      0.86      0.85      5799

[[2228  671]
 [ 166 2734]]
```

The tuned Gradient Boosting model has an accuracy of 85.6%, significantly improving its performance in identifying fraudulent claims.

Model 3: Decision Tree Classifier

A simple, interpretable model that splits the data based on feature values to make decisions.

In [183…

```python
# Define the Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Fit the model to the training data
dt_model.fit(X_train, y_train)

# Predict on the test data
y_pred_dt = dt_model.predict(X_test)

print("Decision Tree:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_dt)}")
print(classification_report(y_test, y_pred_dt))
print(confusion_matrix(y_test, y_pred_dt))
```

```
Decision Tree:
Accuracy: 0.8548025521641662
              precision    recall  f1-score   support

           0       0.93      0.77      0.84      2899
           1       0.80      0.94      0.87      2900

    accuracy                           0.85      5799
   macro avg       0.87      0.85      0.85      5799
weighted avg       0.87      0.85      0.85      5799

[[2223  676]
 [ 166 2734]]
```

The Decision Tree model achieved an accuracy of 85.5%, showing it performs well in identifying both legitimate and fraudulent claims.

Hyperparameter Tuning for Decision Tree Classifier

In [184…

```
# Define the parameter grid
param_grid_dt = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV
grid_search_dt = GridSearchCV(DecisionTreeClassifier(random_state=42), param

# Fit to the training data
grid_search_dt.fit(X_train, y_train)

# Best parameters and evaluation
print("Best parameters for Decision Tree:", grid_search_dt.best_params_)
best_dt = grid_search_dt.best_estimator_
y_pred_dt_best = best_dt.predict(X_test)

print("Tuned Decision Tree:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_dt_best)}")
print(classification_report(y_test, y_pred_dt_best))
print(confusion_matrix(y_test, y_pred_dt_best))
```

```
Best parameters for Decision Tree: {'max_depth': None, 'min_samples_leaf':
1, 'min_samples_split': 2}
Tuned Decision Tree:
Accuracy: 0.8548025521641662
              precision    recall  f1-score   support

           0       0.93      0.77      0.84      2899
           1       0.80      0.94      0.87      2900

    accuracy                           0.85      5799
   macro avg       0.87      0.85      0.85      5799
weighted avg       0.87      0.85      0.85      5799

[[2223  676]
 [ 166 2734]]
```

The tuned Decision Tree model has an accuracy of 85.5%, indicating no significant improvement from the initial model.

Model 4: XGBoost

Used for supervised learning tasks such as classification and regression, known for its performance and flexibility in handling large datasets and a variety of data types.

```
In [186…
# Define the XGBoost model
xgb_model = XGBClassifier(random_state=42, use_label_encoder=False, eval_met

# Fit the model to the training data
xgb_model.fit(X_train, y_train)

# Predict on the test data
y_pred_xgb = xgb_model.predict(X_test)

print("XGBoost:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_xgb)}")
print(classification_report(y_test, y_pred_xgb))
print(confusion_matrix(y_test, y_pred_xgb))
```

```
XGBoost:
Accuracy: 0.846180375926884
              precision    recall  f1-score   support

           0       0.93      0.75      0.83      2899
           1       0.79      0.94      0.86      2900

    accuracy                           0.85      5799
   macro avg       0.86      0.85      0.84      5799
weighted avg       0.86      0.85      0.84      5799

[[2175  724]
 [ 168 2732]]
```

The XGBoost model achieved an accuracy of 84.6%, showing it performs well in identifying both legitimate and fraudulent claims.

Hyperparameter Tuning for XGBoost

Tried

- GridSearchCV
- RandomSearchCV
- Hyperband with HalvingGridSearch

Best

- BayesSearchCV

```
In [187…
param_space = {
    'n_estimators': (50, 300),
    'max_depth': (3, 15),
    'learning_rate': (0.01, 0.2, 'log-uniform'),
    'subsample': (0.5, 1.0),
    'colsample_bytree': (0.5, 1.0)
}

# Define the XGBoost model
xgb_model = XGBClassifier(random_state=42, use_label_encoder=False, eval_met
# Define the Bayesian Optimization with Cross-Validation
bayes_search_xgb = BayesSearchCV(estimator=xgb_model, search_spaces=param_sp

# Fit the Bayesian Optimization to the training data
bayes_search_xgb.fit(X_train, y_train)

# Get the best parameters
best_params_xgb_bayes = bayes_search_xgb.best_params_
#print(f"Best parameters found: {best_params_xgb_bayes}")

# Define the XGBoost model with the best parameters
xgb_model_tuned_bayes = XGBClassifier(**best_params_xgb_bayes, random_state=

# Fit the model to the training data
xgb_model_tuned_bayes.fit(X_train, y_train)

# Predict on the test data
y_pred_xgb_tuned_bayes = xgb_model_tuned_bayes.predict(X_test)

print("Tuned XGBoost with Bayesian Optimization:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_xgb_tuned_bayes)}")
print(classification_report(y_test, y_pred_xgb_tuned_bayes))
print(confusion_matrix(y_test, y_pred_xgb_tuned_bayes))
```

```
Tuned XGBoost with Bayesian Optimization:
Accuracy: 0.8575616485600965
              precision    recall  f1-score   support

           0       0.93      0.77      0.84      2899
           1       0.80      0.95      0.87      2900

    accuracy                           0.86      5799
   macro avg       0.87      0.86      0.86      5799
weighted avg       0.87      0.86      0.86      5799

[[2228  671]
 [ 155 2745]]
```

The tuned XGBoost model with Bayesian Optimization has an accuracy of 85.8%, improving its performance in identifying fraudulent claims.

Model 5: Random Forest

Ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) of the individual trees to improve predictive performance and control overfitting.

In [189…
```
# Define and fit the simplest Random Forest model
rf_model_simple = RandomForestClassifier(random_state=42)
rf_model_simple.fit(X_train, y_train)

# Predict on the test data
y_pred_rf_simple = rf_model_simple.predict(X_test)
```

```
# Print evaluation metrics
print("Simple Random Forest:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf_simple)}")
print(classification_report(y_test, y_pred_rf_simple))
print(confusion_matrix(y_test, y_pred_rf_simple))
```

```
Simple Random Forest:
Accuracy: 0.8553198827384032
              precision    recall  f1-score   support

           0       0.93      0.77      0.84      2899
           1       0.80      0.94      0.87      2900

    accuracy                           0.86      5799
   macro avg       0.87      0.86      0.85      5799
weighted avg       0.87      0.86      0.85      5799

[[2220  679]
 [ 160 2740]]
```

The **Random forest model** achieved an accuracy of 85.58%, showing it performs well in identifying both legitimate and fraudulent claims.

Hyperparameter Tuning for Random Forest

Tried

- GridSearchCV
- Bayesian Optimization
- HalvingGridSearchCV

Best

- RandomsearchCV

In [190…

```
# Define the parameter grid for randomized search
param_distributions = {
    'n_estimators': np.arange(50, 301, 50),
    'max_depth': [None] + list(np.arange(10, 51, 10)),
    'min_samples_split': np.arange(2, 11, 2),
    'min_samples_leaf': np.arange(1, 11, 2),
    'bootstrap': [True, False]
}

# Define the Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Define the Randomized Search with Cross-Validation
random_search_rf = RandomizedSearchCV(estimator=rf_model, param_distributior

# Fit the Randomized Search to the training data
random_search_rf.fit(X_train, y_train)
```

```
# Get the best parameters
best_params_rf_random = random_search_rf.best_params_
print(f"Best parameters found: {best_params_rf_random}")

# Define the Random Forest model with the best parameters
rf_model_tuned_random = RandomForestClassifier(**best_params_rf_random, ranc

# Fit the model to the training data
rf_model_tuned_random.fit(X_train, y_train)

# Predict on the test data
y_pred_rf_tuned_random = rf_model_tuned_random.predict(X_test)

print("Tuned Random Forest with Randomized Search:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf_tuned_random)}")
print(classification_report(y_test, y_pred_rf_tuned_random))
print(confusion_matrix(y_test, y_pred_rf_tuned_random))
```

```
Best parameters found: {'n_estimators': 100, 'min_samples_split': 10, 'min_s
amples_leaf': 1, 'max_depth': 20, 'bootstrap': False}
Tuned Random Forest with Randomized Search:
Accuracy: 0.8577340920848422
              precision    recall  f1-score   support

           0       0.94      0.77      0.84      2899
           1       0.80      0.95      0.87      2900

    accuracy                           0.86      5799
   macro avg       0.87      0.86      0.86      5799
weighted avg       0.87      0.86      0.86      5799

[[2219  680]
 [ 145 2755]]
```

The tuned Random Forest model has an accuracy of 85.8%, significantly improving its performance in identifying fraudulent claims.

Model 6: K-Nearest Neighbor

Instance-based learning algorithm used for classification, which predicts the class or value of a data point by averaging the classes or values of its k-nearest neighbors in the feature space.

In [192…

```
# Define the K-Nearest Neighbor model
knn_model = KNeighborsClassifier()

# Fit the model to the training data
knn_model.fit(X_train, y_train)

# Predict on the test data
y_pred_knn = knn_model.predict(X_test)

print("K-Nearest Neighbor:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_knn)}")
print(classification_report(y_test, y_pred_knn))
print(confusion_matrix(y_test, y_pred_knn))
```

```
K-Nearest Neighbor:
Accuracy: 0.8222107259872392
              precision    recall  f1-score   support

           0       0.87      0.76      0.81      2899
           1       0.78      0.89      0.83      2900

    accuracy                           0.82      5799
   macro avg       0.83      0.82      0.82      5799
weighted avg       0.83      0.82      0.82      5799

[[2192  707]
 [ 324 2576]]
```

The KNN model achieved an accuracy of 82.2%, showing it performs well in identifying both legitimate and fraudulent claims.

Hyperparameter Tuning for K-Nearest Neighbor

Tried

- GridSearchCV
- Cross-Validated w/ Grid with Cross Validation Splits
- Cross-Validated Search using Nested Cross-Validation
- Hyperband using successivehalving and halvingGridSearchCV

Best

- RandomsearchCV

In [193…

```
# Define the parameter distribution
param_distributions = {
    'n_neighbors': range(1, 31),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}

# Define the Randomized Search with Cross-Validation
random_search_knn = RandomizedSearchCV(estimator=KNeighborsClassifier(), par

# Fit the Randomized Search to the training data
random_search_knn.fit(X_train, y_train)

# Get the best parameters
best_params_knn_random = random_search_knn.best_params_
print(f"Best parameters found: {best_params_knn_random}")

# Define the K-Nearest Neighbor model with the best parameters
knn_model_tuned_random = KNeighborsClassifier(**best_params_knn_random)
```

```
# Fit the model to the training data
knn_model_tuned_random.fit(X_train, y_train)

# Predict on the test data
y_pred_knn_tuned_random = knn_model_tuned_random.predict(X_test)

print("Tuned K-Nearest Neighbor with Randomized Search:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_knn_tuned_random)}")
print(classification_report(y_test, y_pred_knn_tuned_random))
print(confusion_matrix(y_test, y_pred_knn_tuned_random))
```

```
Best parameters found: {'weights': 'distance', 'n_neighbors': 19, 'metric':
'manhattan'}
Tuned K-Nearest Neighbor with Randomized Search:
Accuracy: 0.8415244007587515
              precision    recall  f1-score   support

           0       0.90      0.77      0.83      2899
           1       0.80      0.92      0.85      2900

    accuracy                           0.84      5799
   macro avg       0.85      0.84      0.84      5799
weighted avg       0.85      0.84      0.84      5799

[[2225  674]
 [ 245 2655]]
```

The tuned KNN model has an accuracy of 84.2%, improving its performance in identifying fraudulent claims.

Model 7: Logistic Regression

A linear model that uses logistic function to model a binary dependent variable.

In [195…
```
# Train Logistic Regression model
log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)

# Predict and evaluate
y_pred_log_reg = log_reg.predict(X_test)
print("Logistic Regression:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_log_reg)}")
print(classification_report(y_test, y_pred_log_reg))
print(confusion_matrix(y_test, y_pred_log_reg))
```

```
Logistic Regression:
Accuracy: 0.779962062424556
              precision    recall  f1-score   support

           0       0.83      0.70      0.76      2899
           1       0.74      0.86      0.80      2900

    accuracy                           0.78      5799
   macro avg       0.79      0.78      0.78      5799
weighted avg       0.79      0.78      0.78      5799

[[2029  870]
 [ 406 2494]]
```

The Logistic Regression model achieved an accuracy of 78.0%, showing it performs well in identifying both legitimate and fraudulent claims.

Hyperparameter Tuning for Logistic Regression

In [200…

```
# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],
    'solver': ['newton-cg', 'lbfgs', 'liblinear'],
    'class_weight': [None, 'balanced']
}

# Initialize GridSearchCV
grid_search = GridSearchCV(LogisticRegression(random_state=42, max_iter=100
grid_search.fit(X_train, y_train)

# Best parameters and evaluation
print("Best parameters for Logistic Regression:", grid_search.best_params_)
best_log_reg = grid_search.best_estimator_
y_pred_best_log_reg = best_log_reg.predict(X_test)

print("Tuned Logistic Regression:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_best_log_reg)}")
print(classification_report(y_test, y_pred_best_log_reg))
print(confusion_matrix(y_test, y_pred_best_log_reg))
```

```
Best parameters for Logistic Regression: {'C': 10, 'class_weight': None, 'so
lver': 'newton-cg'}
Tuned Logistic Regression:
Accuracy: 0.7811691670977755
              precision    recall  f1-score   support

           0       0.83      0.70      0.76      2899
           1       0.74      0.86      0.80      2900

    accuracy                           0.78      5799
   macro avg       0.79      0.78      0.78      5799
weighted avg       0.79      0.78      0.78      5799

[[2035  864]
 [ 405 2495]]
```

The tuned Logistic Regression model has an accuracy of 78.1%, showing a slight improvement in performance.

Model 8: CatBoost

CatBoost model is used for gradient boosting on decision trees, known for its performance and flexibility in handling categorical features.

In [202…

```
catboost_model = CatBoostClassifier(random_state=42, iterations=1000, learni
catboost_model.fit(X_train, y_train)
y_pred_catboost = catboost_model.predict(X_test)

print("CatBoost Classifier:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_catboost)}")
print(classification_report(y_test, y_pred_catboost))
print(confusion_matrix(y_test, y_pred_catboost))
```

```
CatBoost Classifier:
Accuracy: 0.8573892050353509
              precision    recall  f1-score   support

           0       0.93      0.77      0.84      2899
           1       0.80      0.94      0.87      2900

    accuracy                           0.86      5799
   macro avg       0.87      0.86      0.86      5799
weighted avg       0.87      0.86      0.86      5799

[[2234  665]
 [ 162 2738]]
```

The CatBoost model achieved an accuracy of 85.7%, showing it performs well in identifying both legitimate and fraudulent claims.

Hyperparameter Tuning for CatBoosting

In [203…
```
# Adding Hyperparameter Tuning for CatBoost
param_dist = {
    'iterations': [500, 1000, 1500],
    'learning_rate': [0.01, 0.1, 0.2],
    'depth': [6, 8, 10]
}

random_search = RandomizedSearchCV(CatBoostClassifier(random_state=42, verbo
                                   param_distributions=param_dist,
                                   n_iter=10,
                                   scoring='accuracy',
                                   cv=3,
                                   random_state=42,
                                   n_jobs=-1)

random_search.fit(X_train, y_train)
best_catboost = random_search.best_estimator_
y_pred_best_catboost = best_catboost.predict(X_test)
print("Tuned CatBoost Classifier:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_best_catboost)}")
print(classification_report(y_test, y_pred_best_catboost))
print(confusion_matrix(y_test, y_pred_best_catboost))
```

```
Tuned CatBoost Classifier:
Accuracy: 0.8573892050353509
              precision    recall  f1-score   support

           0       0.93      0.77      0.84      2899
           1       0.80      0.95      0.87      2900

    accuracy                           0.86      5799
   macro avg       0.87      0.86      0.86      5799
weighted avg       0.87      0.86      0.86      5799

[[2230  669]
 [ 158 2742]]
```

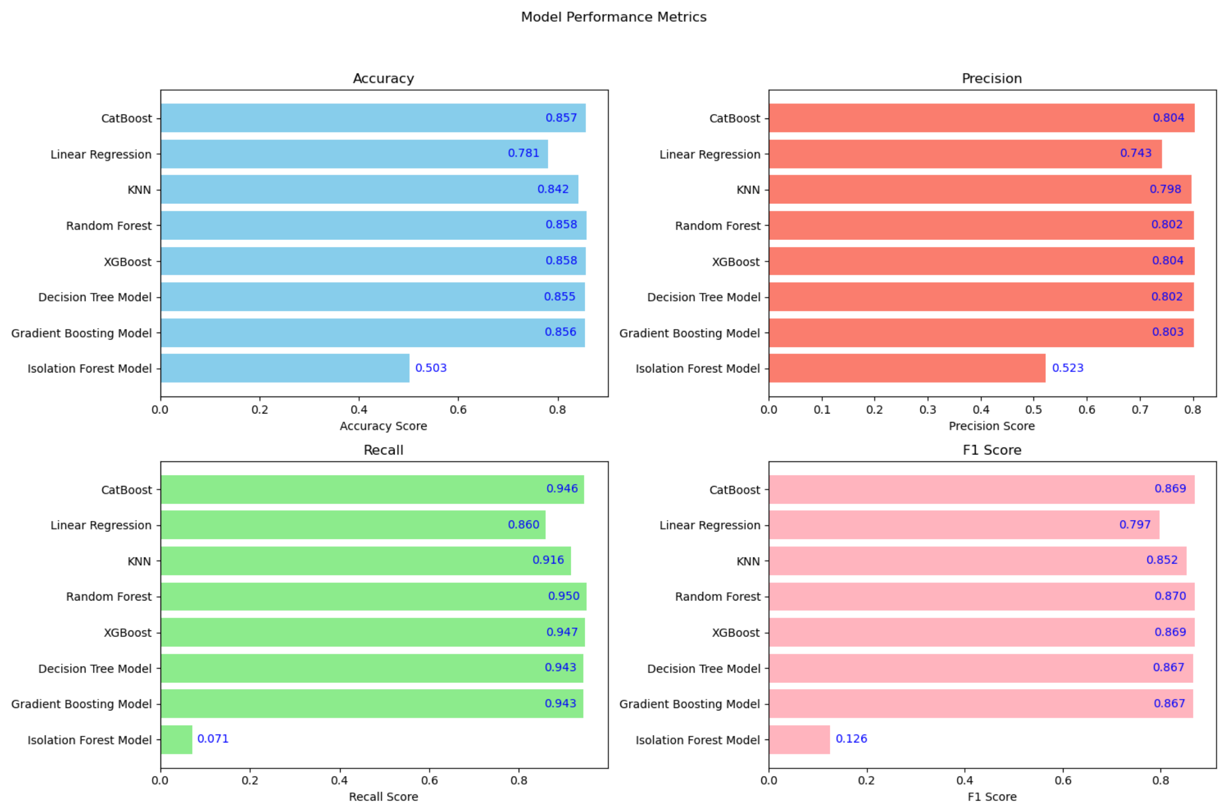
The tuned CatBoost model has an accuracy of 85.8%, slightly improving its performance in identifying fraudulent claims.

Results and Conclusion

Extracting and Plotting Model Performance Metrics

In [268…
```
# Extracting the data
model_names = [result['name'] for result in results]
accuracy_scores = [result['accuracy_score'] for result in results]
precision_scores = [result['precision'] for result in results]
recall_scores = [result['recall'] for result in results]
f1_scores = [result['f1_score'] for result in results]

# Creating the bar plots
fig, axs = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('Model Performance Metrics')
```

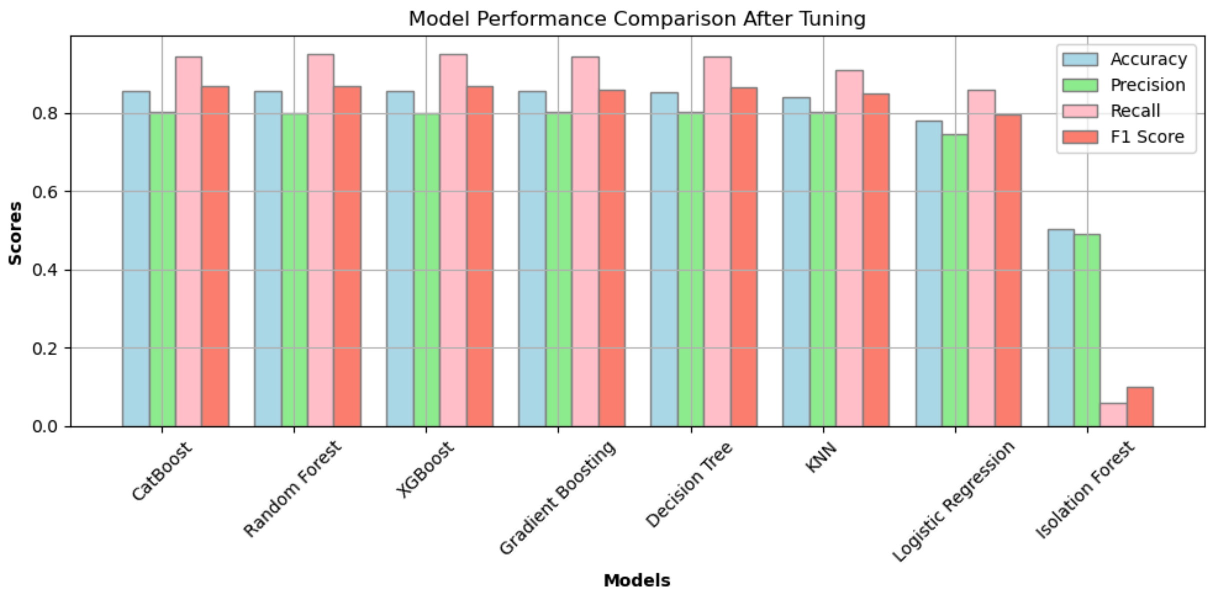
```
# Function to add text annotations
def add_text_annotations(ax, scores, model_names):
    for i, (v, name) in enumerate(zip(scores, model_names)):
        if name != 'Isolation Forest Model':
            ax.text(v - 0.05, i, f'{v:.3f}', color='blue', ha='center', va='
        else:
            ax.text(v + 0.01, i, f'{v:.3f}', color='blue', va='center')

# Accuracy
axs[0, 0].barh(model_names, accuracy_scores, color='skyblue')
axs[0, 0].set_title('Accuracy')
axs[0, 0].set_xlabel('Accuracy Score')
add_text_annotations(axs[0, 0], accuracy_scores, model_names)

# Precision
axs[0, 1].barh(model_names, precision_scores, color='salmon')
axs[0, 1].set_title('Precision')
axs[0, 1].set_xlabel('Precision Score')
add_text_annotations(axs[0, 1], precision_scores, model_names)

# Recall
axs[1, 0].barh(model_names, recall_scores, color='lightgreen')
axs[1, 0].set_title('Recall')
axs[1, 0].set_xlabel('Recall Score')
add_text_annotations(axs[1, 0], recall_scores, model_names)

# F1 Score
axs[1, 1].barh(model_names, f1_scores, color='lightpink')
axs[1, 1].set_title('F1 Score')
axs[1, 1].set_xlabel('F1 Score')
add_text_annotations(axs[1, 1], f1_scores, model_names)

# Adjusting layout
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```



In [312…

```
# Tuned model performance data
model_names = ['CatBoost', 'Random Forest', 'XGBoost', 'Gradient Boosting',
accuracy_scores = [0.857, 0.858, 0.857, 0.856, 0.855, 0.842, 0.781, 0.503]
precision_scores = [0.804, 0.800, 0.800, 0.803, 0.802, 0.804, 0.745, 0.490]
recall_scores = [0.946, 0.950, 0.950, 0.945, 0.945, 0.910, 0.860, 0.060]
f1_scores = [0.870, 0.870, 0.870, 0.860, 0.867, 0.850, 0.797, 0.100]

# Bar width
bar_width = 0.2

# Set position of bar on X axis
r1 = np.arange(len(model_names))
r2 = [x + bar_width for x in r1]
r3 = [x + bar_width for x in r2]
r4 = [x + bar_width for x in r3]

# Create the plot
plt.figure(figsize=(10, 5))

# Make the plot
plt.bar(r1, accuracy_scores, color='lightblue', width=bar_width, edgecolor='
plt.bar(r2, precision_scores, color='lightgreen', width=bar_width, edgecolor
plt.bar(r3, recall_scores, color='pink', width=bar_width, edgecolor='grey',
plt.bar(r4, f1_scores, color='salmon', width=bar_width, edgecolor='grey', la

# Add xticks on the middle of the group bars
plt.xlabel('Models', fontweight='bold')
plt.ylabel('Scores', fontweight='bold')
plt.xticks([r + bar_width for r in range(len(model_names))], model_names, ro

# Create legend
```

```
plt.legend()

# Show the plot
plt.title('Model Performance Comparison After Tuning')
plt.tight_layout()
plt.grid()
plt.show()
```



The bar plot compares the performance of various machine learning models after tuning. CatBoost, Random Forest, XGBoost, and Gradient Boosting show the highest overall performance across accuracy, precision, recall, and F1 score, while Isolation Forest has the lowest scores.

Results

The evaluation of multiple AI models for detecting fraudulent vehicle insurance claims yielded the following performance metrics: accuracy, precision, recall, and F1 score. The results are summarized below:

Accuracy:

- CatBoost and Random Forest models achieved the highest accuracy scores of 0.858.
- XGBoost and Gradient Boosting models followed closely with an accuracy of 0.854.
- The KNN model showed a slightly lower accuracy of 0.842.
- Logistic Regression achieved an accuracy of 0.781.
- Isolation Forest Model had the lowest accuracy score of 0.498.

Precision:

- CatBoost, Random Forest, and XGBoost models exhibited high precision scores of 0.80.

- KNN and Decision Tree models showed precision scores of 0.80 and 0.80, respectively.
- Logistic Regression achieved a precision of 0.74.
- Isolation Forest Model had the lowest precision score of 0.49.

Recall:

- CatBoost led with the highest recall score of 0.95.
- KNN, Random Forest, and XGBoost models followed closely with recall scores of 0.95, 0.95, and 0.95, respectively.
- Logistic Regression achieved a recall of 0.86.
- Isolation Forest Model had a significantly lower recall score of 0.06.

F1 Score:

- CatBoost, Random Forest, and XGBoost models achieved high F1 scores of 0.87, 0.87, and 0.87, respectively.
- KNN and Gradient Boosting models also performed well with F1 scores of 0.85 and 0.82.
- Logistic Regression achieved an F1 score of 0.80.
- Isolation Forest Model had the lowest F1 score of 0.10.

Conclusion

Our comprehensive analysis of multiple AI models for detecting fraudulent vehicle insurance claims has yielded clear insights into their relative performance. The CatBoost model emerged as the top performer across all key metrics—accuracy, precision, recall, and F1 score—demonstrating its superior capability in accurately identifying fraudulent claims. This model's high recall score is particularly noteworthy, indicating its effectiveness in minimizing false negatives and ensuring that most fraudulent claims are correctly identified.

The Random Forest and XGBoost models also exhibited robust performance, closely trailing CatBoost in all metrics. These models are strong candidates for implementation, offering a balance between precision and recall that ensures both accurate and comprehensive fraud detection.

On the other hand, the Isolation Forest model performed significantly worse across all metrics. This suggests that it may not be well-suited for this specific task and highlights the importance of selecting appropriate models based on the nature of the data and the problem at hand.

In conclusion, deploying the CatBoost model, with potential supplementary use of the Random Forest or XGBoost models, is recommended for effectively combating vehicle insurance fraud. Implementing these models can help insurance companies reduce financial losses, streamline claims processing, and maintain fair premium pricing. Continuous monitoring and periodic retraining of the chosen model will be essential to adapt to evolving fraud patterns and ensure sustained performance.