Fraudulent Claim Detection
By Paras Mehta and Shaik Muhammad Naim

1. Data Loading

```
df = pd.read_csv("insurance_claims.csv")
df.head()  # check the records
df.shape #Check the shape
# Inspect the features in the dataset
df.info()
```

2. Data Cleaning

2.1. Handle null values

2.1.1. Examine the columns to determine if any value or column needs to be treated

```
# Check the number of missing values in each column
df = df.replace('?', pd.NA)
df.isnull().sum()
```

2.1.2. Handle rows containing null values

```
# Handle the rows containing null values
df["collision_type"] = df["collision_type"].fillna('unknown')
df["property_damage"] = df["property_damage"].fillna('unknown')
df["authorities_contacted"] = df["authorities_contacted"].fillna('unknown')
df["police_report_available"] = df["police_report_available"].fillna('unknown')


df.isnull().sum()
```

2.2. Identify and handle redundant values and columns

2.2.1. Examine the columns to determine if any value or column needs to be treated

```
# Write code to display all the columns with their unique values and
counts and check for redundant values
redundant_cols = []
```

```
for col in df.columns:
    nunique = df[col].nunique(dropna =False)
    total = len(df[col])
    if nunique <= 1 :
        redundant_cols.append(col)

    elif nunique/total > 0.95:
        redundant_cols.append(col)

print ("Redundant column")
print (redundant_cols)
for col in df.columns:
    print(f"\nColumn :{col}")
    print("Unique count: ", df[col].nunique(dropna=False))
    print("Sample value: ", df[col].dropna().unique()[:10])
```

2.2.2. Identify and drop any columns that are completely empty

```
# Identify and drop any columns that are completely empty
df = df.drop(columns=['_c39'],axis=1 )
```

2.2.3. Identify and drop rows where features have illogical or invalid values, such as negative values for features that should only have positive values

```
# Identify and drop rows where features have illogical or invalid values,
such as negative values for features that should only have positive values
#Removing the umbrella_limit that is less than 0
df = df[df["umbrella_limit"]>=0]
```

2.2.4. Identify and remove columns where a large proportion of the values are unique or near-unique, as these columns are likely to be identifiers or have very limited predictive power

```
# Identify and remove columns that are likely to be identifiers or have very
limited predictive power
#policy_number → unique per record, no predictive power.
#policy_bind_date → almost unique, timestamp-like, not directly predictive.
#insured_zip → 999 unique values (almost every row unique).
#incident_date → 999 unique values, not predictive unless transformed to
“day of week / month”.
#incident_location → 999 unique values (acts like ID).
#total_claim_amount → redundant (sum of injury_claim + property_claim +
```

```
vehicle_claim).
drop_cols = [
    "policy_number",
    "policy_bind_date",
    "insured_zip",
    "incident_date",
    "incident_location",
    "total_claim_amount"  # redundant with injury/property/vehicle claims
]
df_clean = df.drop(columns=drop_cols)
# Check the dataset
df_clean.head()
```

2.3. Fix Data Types

```
# Fix the data types of the columns with incorrect data types
#Make fraud_reported to 1 or 0 1 means fraud claim and 0 means not
fraud
df_clean["fraud_reported"] = df_clean["fraud_reported"].map({"Y": 1, "N":
0}).astype(int)

# change categorial column to type as category
categorical_cols = [
    "policy_state", "policy_csl", "insured_sex", "insured_education_level",
    "insured_occupation", "insured_hobbies", "insured_relationship",
    "incident_type", "collision_type", "incident_severity",
    "authorities_contacted", "incident_state", "incident_city",
    "property_damage", "police_report_available", "auto_make",
"auto_model"
]
df_clean[categorical_cols] = df_clean[categorical_cols].astype("category")

# Check the features of the data again
df_clean.info()
```

3. Train-Validation Split

3.1. Import required libraries

```
# Import train-test-split
from sklearn.model_selection import train_test_split
```

3.2. Define feature and target variables

```
# Put all the feature variables in X
X = df_clean.drop(columns=["fraud_reported"], axis =1)
# Put the target variable in y
y = df_clean["fraud_reported"]
```

3.3. Split the data

```
# Split the dataset into 70% train and 30% validation and use stratification on the
target variable
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.3, stratify=y,
random_state=42)
# Reset index for all train and test sets
X_train = X_train.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
```

4. EDA on training data

4.1. Perform univariate analysis

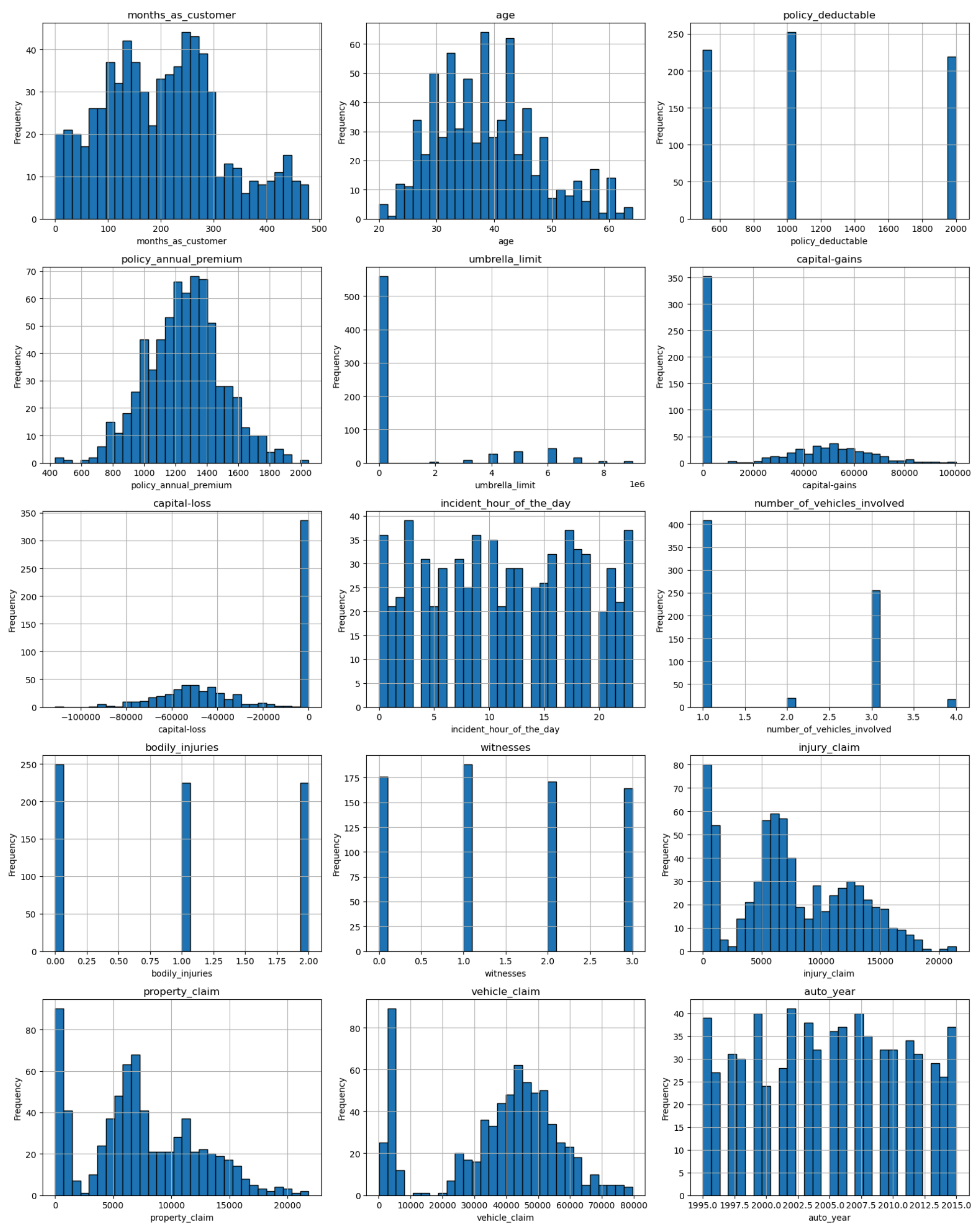
4.1.1. Identify and select numerical columns from training data for univariate analysis

```
# Select numerical columns
numericals_col =
X_train.select_dtypes(include=["int64","float64"]).columns.tolist()
print("numerical column", numericals_col)
```

4.1.2. Visualise the distribution of selected numerical features using appropriate plots to understand their characteristics

```
plt.figure(figsize=(16, 20))  # adjust size for readability
for i, col in enumerate(numericals_col, 1):
    plt.subplot(5, 3, i)  # 5 rows, 3 columns of subplots
    X_train[col].hist(bins=30, edgecolor='black')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel("Frequency")
```

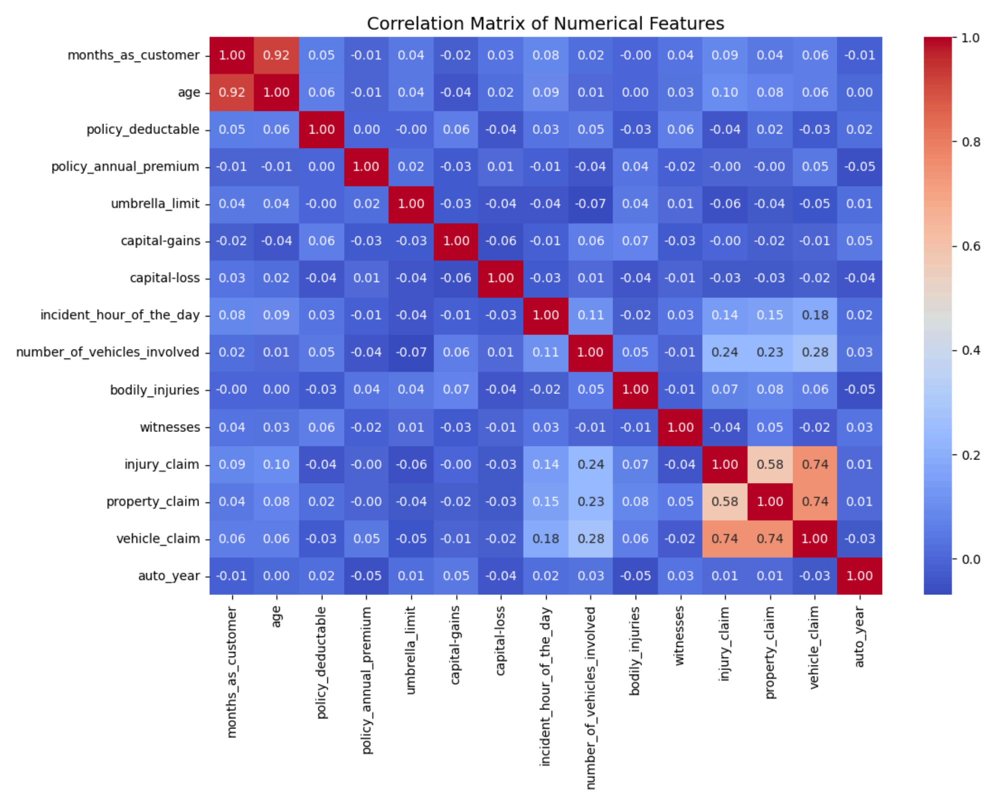
```
plt.tight_layout()
plt.show()
```



4.2. Perform correlation analysis

```
# Create correlation matrix for numerical columns
corr_matrix = X_train[numericals_col].corr()
# Plot Heatmap of the correlation matrix
plt.figure(figsize=(12, 8))
```

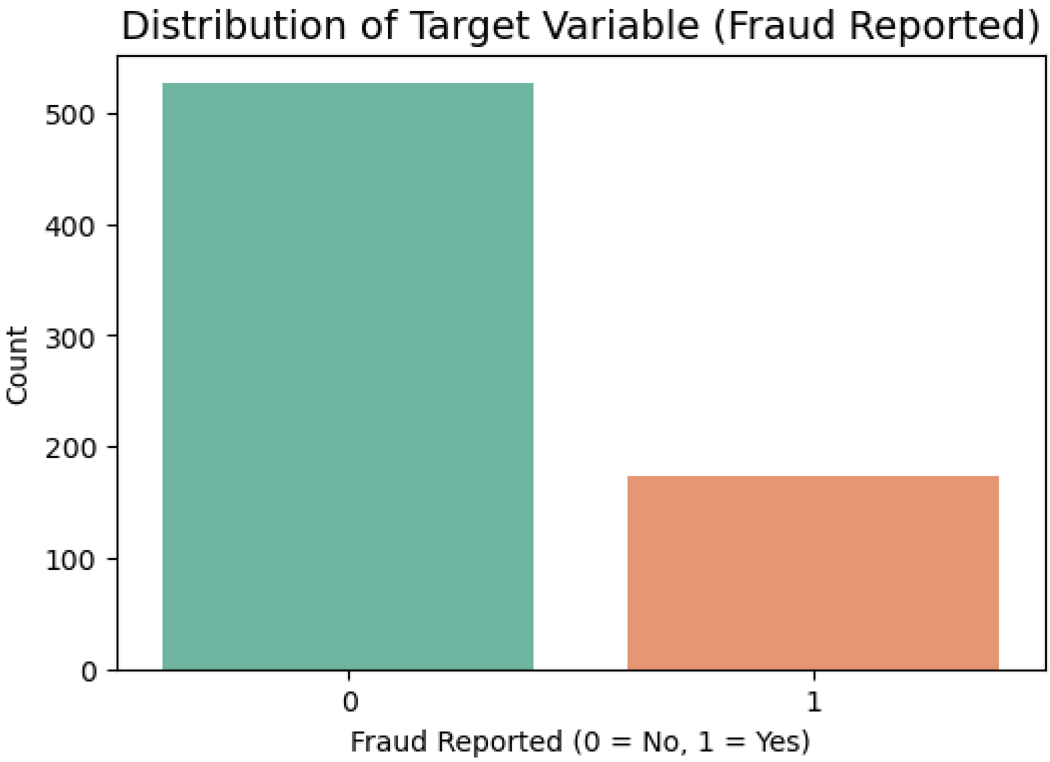
```
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f",
cbar=True)
plt.title("Correlation Matrix of Numerical Features", fontsize=14)
plt.show()
```



4.3. Check class balance

```
# Plot a bar chart to check class balance
print(y_train["fraud_reported"].value_counts())
print("\nClass distribution (%):")
print(y_train["fraud_reported"].value_counts(normalize=True) * 100)
plt.figure(figsize=(6,4))
sns.countplot(x="fraud_reported", data=y_train, palette="Set2")
plt.title("Distribution of Target Variable (Fraud Reported)", fontsize=14)
plt.xlabel("Fraud Reported (0 = No, 1 = Yes)")
plt.ylabel("Count")
plt.show()

—-output–
fraud_reported
0    526
1    173
Name: count, dtype: int64
```

```
Class distribution (%):
fraud_reported
0    75.250358
1    24.749642
Name: proportion, dtype: float64
```



4.4. Perform bivariate analysis

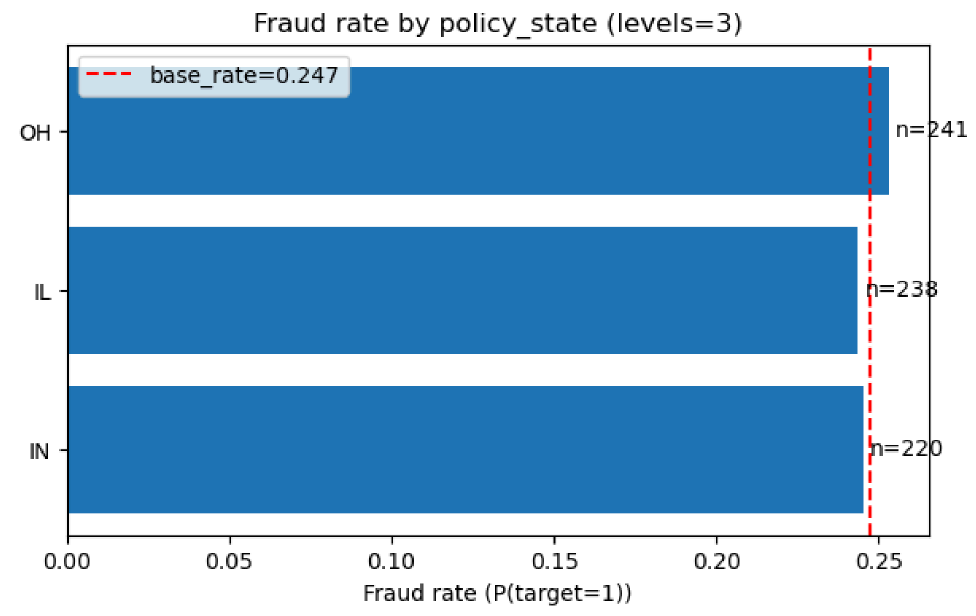
4.4.1. Target likelihood analysis for categorical variables

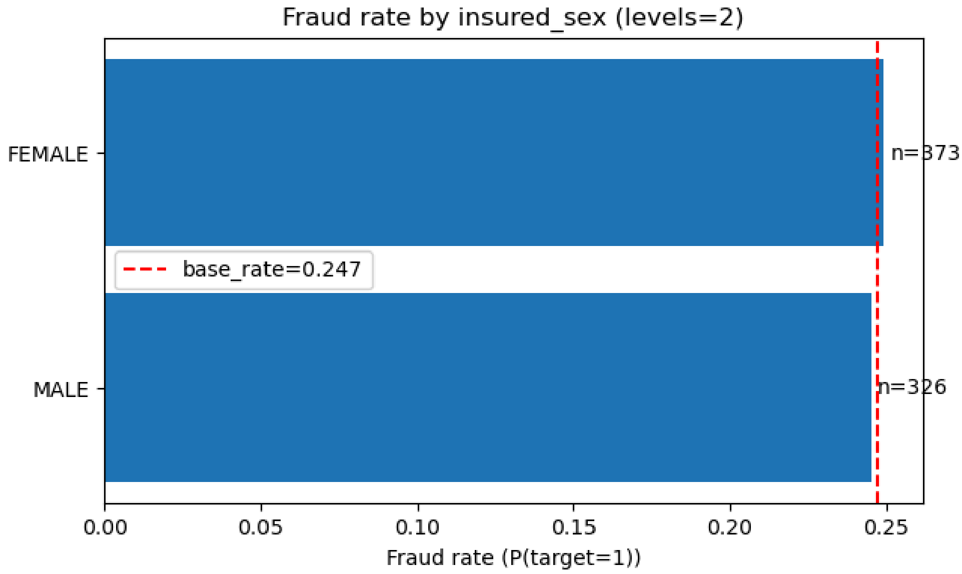
```
# Write a function to calculate and analyse the target variable likelihood for
categorical features


def analyze_cat_target_likelihood(X_train, y_train,
                              categorical_cols=None,
                              min_count=20,
                              rate_diff_threshold=0.05,
                              plot=False,
                              top_n_levels_plot=10,
                              figsize=(7,4)):
    """
    Analyze P(target=1) for each level of categorical features using X_train
and y_train.
```

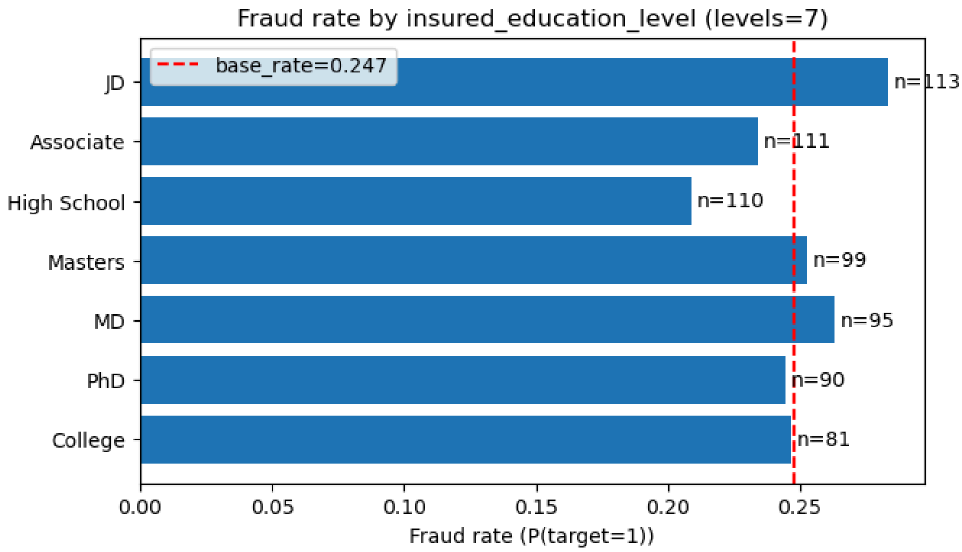
```
    Robust to y_train being a Series or a single-column DataFrame.
    """

    # Ensure y is a 1-D numeric Series
    if isinstance(y_train, pd.DataFrame):
        if y_train.shape[1] == 1:
            y = y_train.iloc[:, 0].reset_index(drop=True)
        else:
            # more than one column - take first column (user should supply
single-target)
            y = y_train.iloc[:, 0].reset_index(drop=True)
    else:
        y = pd.Series(y_train).reset_index(drop=True)

    # Coerce to numeric (0/1)

    y = y.astype(int)

    # Reset X index to align
    X = X_train.reset_index(drop=True).copy()

    # Auto-detect categorical columns if not provided
    if categorical_cols is None:
        categorical_cols = X.select_dtypes(include=[
"category"]).columns.tolist()

    categorical_cols = [c for c in categorical_cols if c in X.columns]

    per_feature = {}
    summary_rows = []

    # scalar base rate
    base_rate = float(y.mean())

    for col in categorical_cols:
        # skip if column has no non-null values
        if X[col].dropna().shape[0] == 0:
            continue

        temp = pd.concat([X[col], y], axis=1)
        temp.columns = [col, "target"]

        grp = temp.groupby(col)["target"].agg(total_count="count",
fraud_cases="sum").reset_index()
        grp["fraud_rate"] = grp["fraud_cases"] / grp["total_count"]
        grp["low_support"] = grp["total_count"] < min_count
        grp = grp.sort_values("fraud_rate",
ascending=False).reset_index(drop=True)

        # feature-level metrics
        max_rate = float(grp["fraud_rate"].max())
        min_rate = float(grp["fraud_rate"].min())
        rate_range = max_rate - min_rate
        n_levels = grp.shape[0]
```

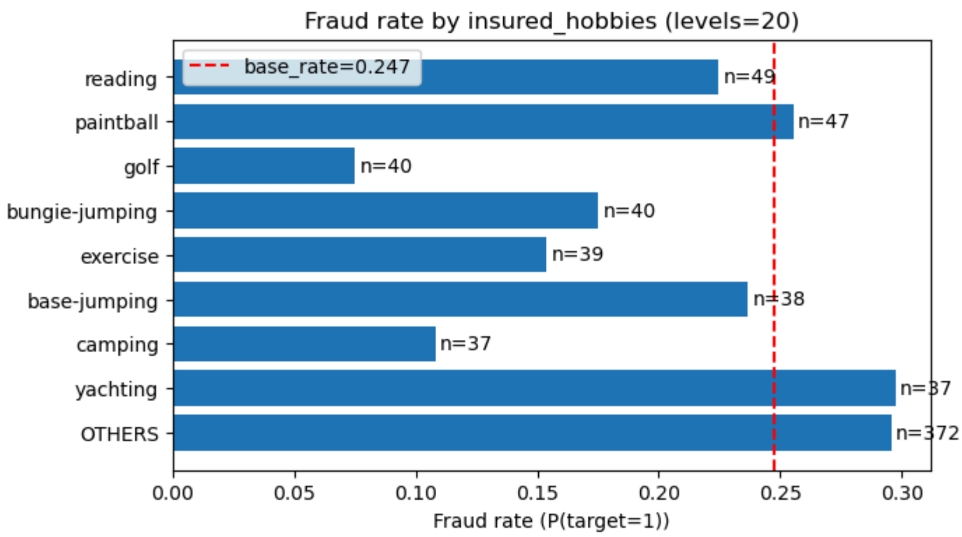
```
        weighted_rate = float((grp["fraud_rate"] * grp["total_count"]).sum() /
grp["total_count"].sum())
        n_low_support_levels = int(grp["low_support"].sum())
        is_weak = rate_range < rate_diff_threshold

        per_feature[col] = grp
        summary_rows.append({
            "feature": col,
            "n_levels": n_levels,
            "max_rate": max_rate,
            "min_rate": min_rate,
            "rate_range": rate_range,
            "weighted_rate": weighted_rate,
            "base_rate": base_rate,
            "is_weak": is_weak,
            "n_low_support_levels": n_low_support_levels
        })

        # optional plotting
        if plot:
            plot_df = grp.sort_values("total_count", ascending=False).copy()
            if plot_df.shape[0] > top_n_levels_plot:
                top = plot_df.head(top_n_levels_plot).copy()
                others = plot_df.iloc[top_n_levels_plot:].copy()
                others_row = {
                    col: "OTHERS",
                    "total_count": others["total_count"].sum(),
                    "fraud_cases": others["fraud_cases"].sum()
                }
                others_row["fraud_rate"] = others_row["fraud_cases"] /
others_row["total_count"] if others_row["total_count"]>0 else np.nan
                others_row["low_support"] =
others["total_count"].lt(min_count).all()
                top = pd.concat([top, pd.DataFrame([others_row])],
ignore_index=True)
                plot_df = top

            plt.figure(figsize=figsize)
            plt.barh(plot_df[col].astype(str), plot_df["fraud_rate"])
            plt.xlabel("Fraud rate (P(target=1))")
            plt.title(f"Fraud rate by {col} (levels={n_levels})")
            plt.gca().invert_yaxis()
            for i, row in enumerate(plot_df.itertuples()):
                # protect against NaN fraud_rate
                if pd.notna(getattr(row, "fraud_rate", None)):
                    plt.text(getattr(row, "fraud_rate") + 0.002, i,
f"n={int(row.total_count)}", va="center")
            # draw base rate as line
            plt.axvline(x=base_rate, color="red", linestyle="--",
label=f"base_rate={base_rate:.3f}")
            plt.legend()
            plt.show()

    summary =
```

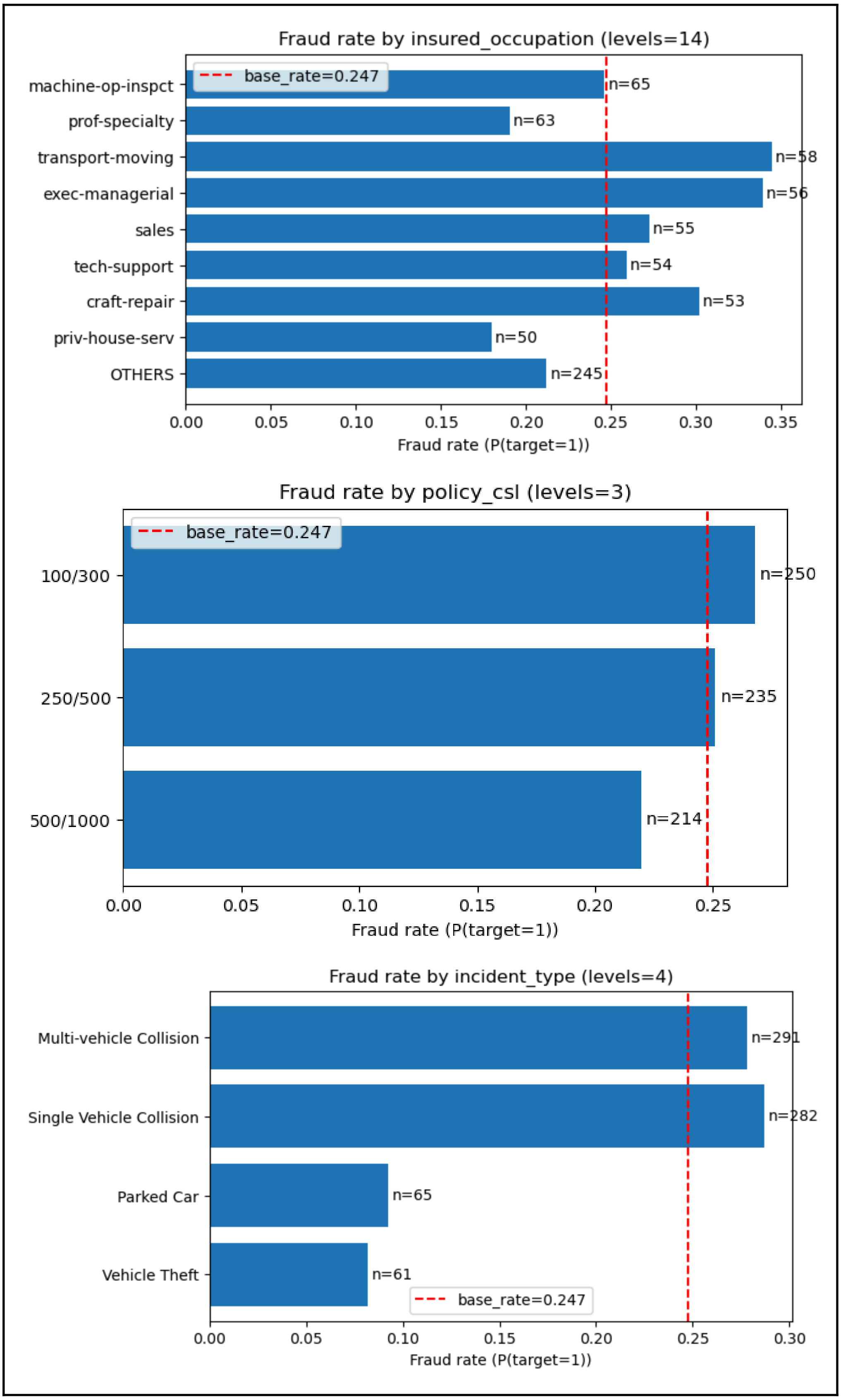
```
pd.DataFrame(summary_rows).set_index("feature").sort_values("rate_rang
e", ascending=False)
    return {"per_feature": per_feature, "summary": summary}


result = analyze_cat_target_likelihood(X_train, y_train,
                                       categorical_cols=None,  # autodetect
                                       min_count=20,
                                       rate_diff_threshold=0.05,
                                       plot=True,
                                       top_n_levels_plot=8)

# View summary (features ranked by how much fraud rate varies across
their levels)
print(result["summary"])
print(result["per_feature"]["property_damage"].sort_values("total_count",
ascending=False))
```



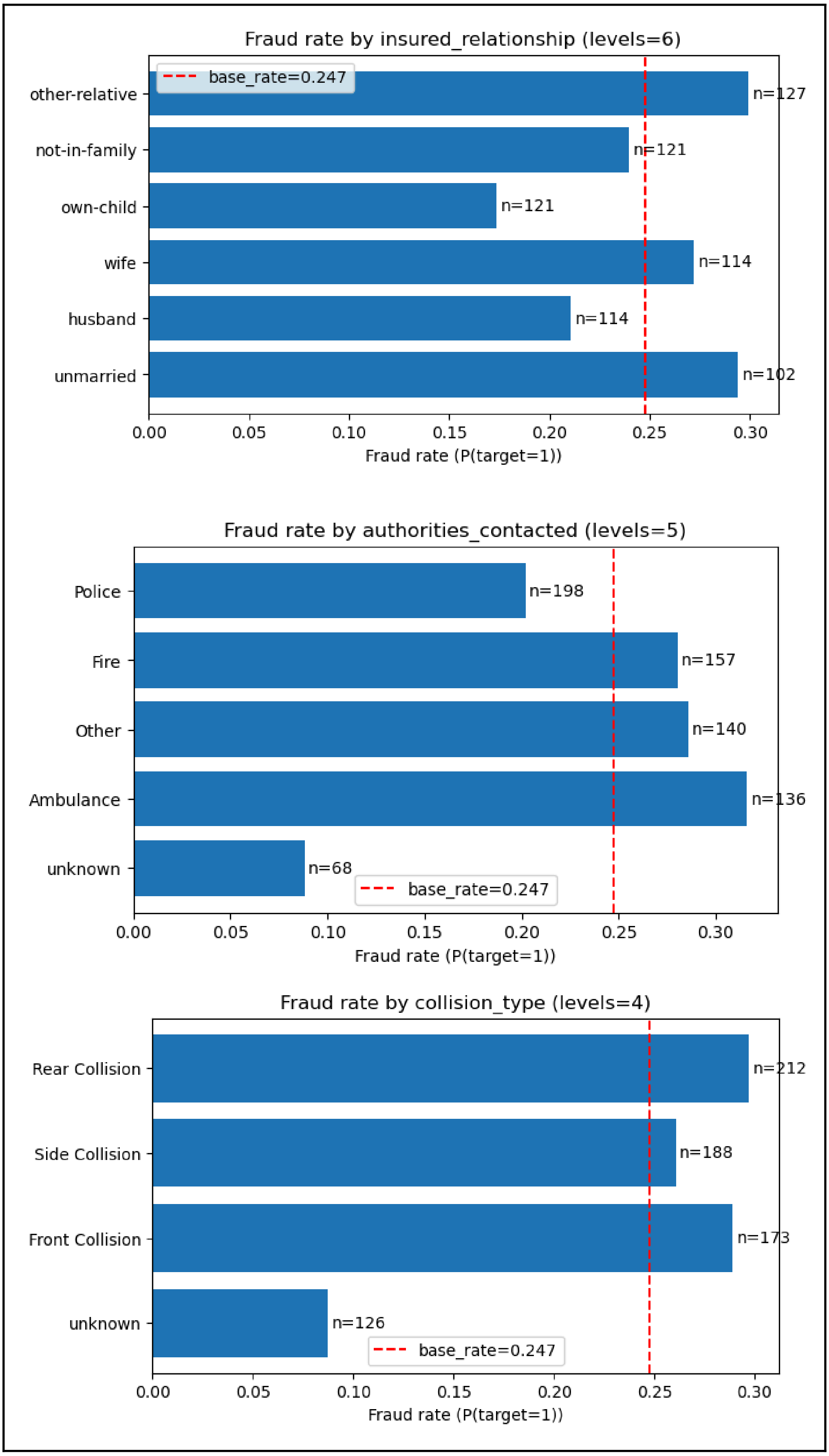

Fraud rate by insured_sex (levels=2)
FEMALE
MALE
n=373
n=326
base_rate=0.247
0.00
0.05
0.10
0.15
0.20
0.25
Fraud rate (P(target=1))

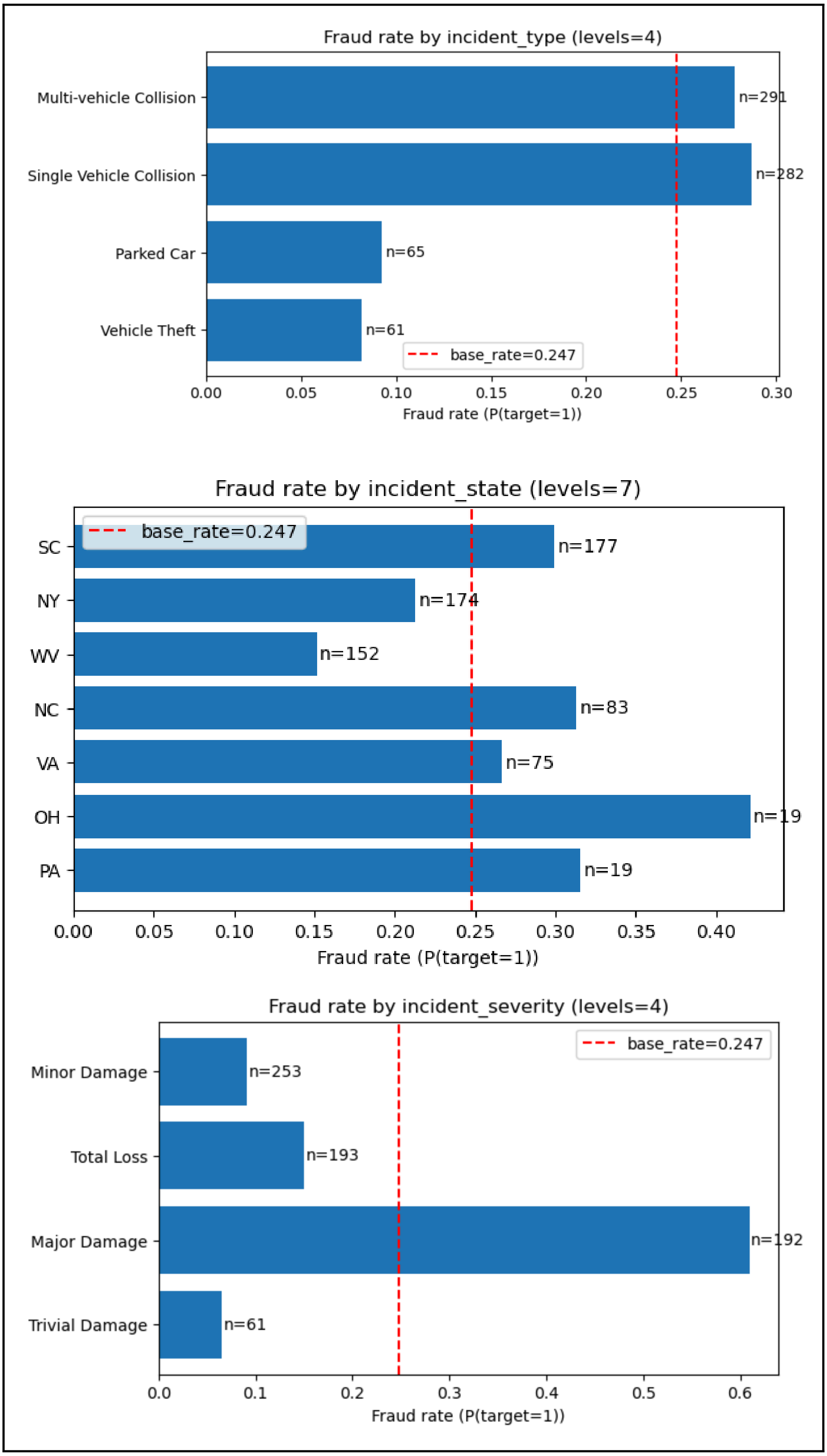

Fraud rate by insured_education_level (levels=7)
JD
Associate
High School
Masters
MD
PhD
College
n=113
n=111
n=110
n=99
n=95
n=90
n=81
base_rate=0.247
0.00
0.05
0.10
0.15
0.20
0.25
Fraud rate (P(target=1))

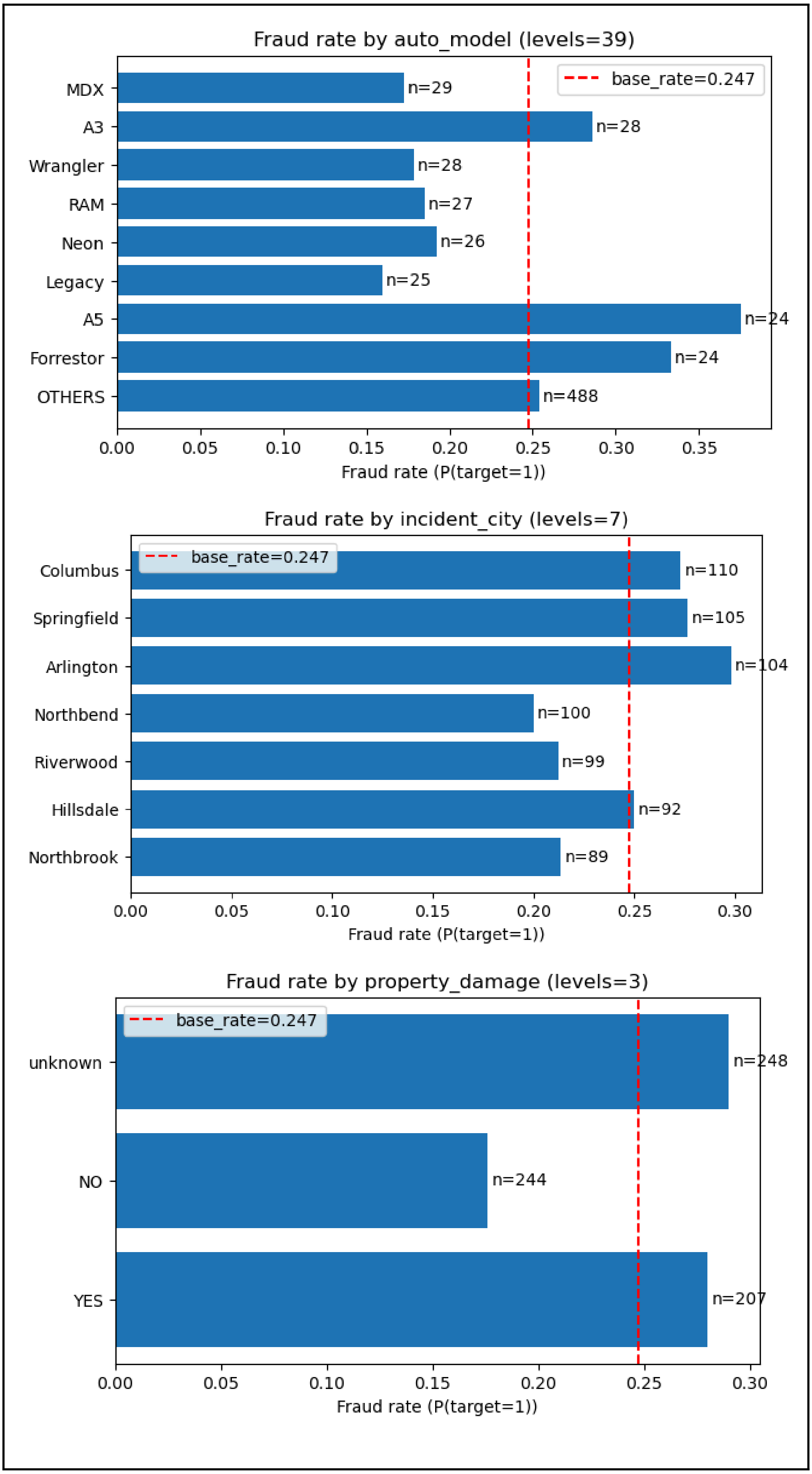

Fraud rate by insured_hobbies (levels=20)
reading
paintball
golf
bungie-jumping
exercise
base-jumping
camping
yachting
OTHERS
n=49
n=47
n=40
n=40
n=39
n=38
n=37
n=37
n=372
base_rate=0.247
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Fraud rate (P(target=1))

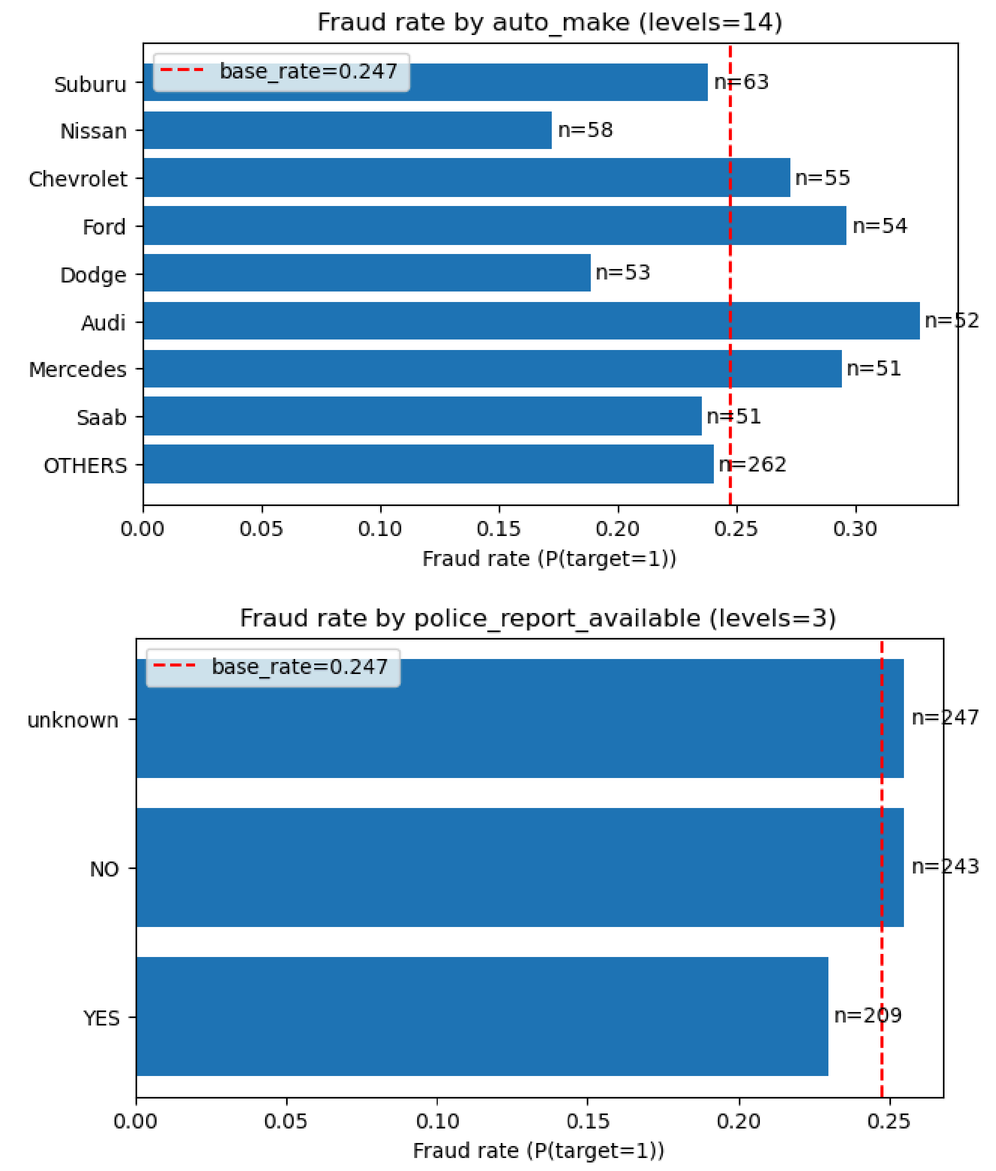


Fraud rate by insured_occupation (levels=14)
base_rate=0.247
machine-op-inspct
prof-specialty
transport-moving
exec-managerial
sales
tech-support
craft-repair
priv-house-serv
OTHERS
n=65
n=63
n=58
n=56
n=55
n=54
n=53
n=50
n=245
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Fraud rate (P(target=1))
Fraud rate by policy_csl (levels=3)
base_rate=0.247
100/300
250/500
500/1000
n=250
n=235
n=214
0.00
0.05
0.10
0.15
0.20
0.25
Fraud rate (P(target=1))
Fraud rate by incident_type (levels=4)
Multi-vehicle Collision
Single Vehicle Collision
Parked Car
Vehicle Theft
n=291
n=282
n=65
n=61
base_rate=0.247
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Fraud rate (P(target=1))


Fraud rate by insured_relationship (levels=6)
base_rate=0.247
other-relative
not-in-family
own-child
wife
husband
unmarried
n=127
n=121
n=121
n=114
n=114
n=102
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Fraud rate (P(target=1))
Fraud rate by authorities_contacted (levels=5)
Police
Fire
Other
Ambulance
unknown
n=198
n=157
n=140
n=136
n=68
base_rate=0.247
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Fraud rate (P(target=1))
Fraud rate by collision_type (levels=4)
Rear Collision
Side Collision
Front Collision
unknown
n=212
n=188
n=173
n=126
base_rate=0.247
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Fraud rate (P(target=1))


Fraud rate by incident_type (levels=4)
Multi-vehicle Collision
Single Vehicle Collision
Parked Car
Vehicle Theft
n=291
n=282
n=65
n=61
base_rate=0.247
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Fraud rate (P(target=1))
Fraud rate by incident_state (levels=7)
base_rate=0.247
SC
NY
WV
NC
VA
OH
PA
n=177
n=174
n=152
n=83
n=75
n=19
n=19
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Fraud rate (P(target=1))
Fraud rate by incident_severity (levels=4)
base_rate=0.247
Minor Damage
Total Loss
Major Damage
Trivial Damage
n=253
n=193
n=192
n=61
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Fraud rate (P(target=1))


Fraud rate by auto_model (levels=39)
base_rate=0.247
MDX
A3
Wrangler
RAM
Neon
Legacy
A5
Forrestor
OTHERS
n=29
n=28
n=28
n=27
n=26
n=25
n=24
n=24
n=488
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35
Fraud rate (P(target=1))
Fraud rate by incident_city (levels=7)
base_rate=0.247
Columbus
Springfield
Arlington
Northbend
Riverwood
Hillsdale
Northbrook
n=110
n=105
n=104
n=100
n=99
n=92
n=89
0.00 0.05 0.10 0.15 0.20 0.25 0.30
Fraud rate (P(target=1))
Fraud rate by property_damage (levels=3)
base_rate=0.247
unknown
NO
YES
n=248
n=244
n=207
0.00 0.05 0.10 0.15 0.20 0.25 0.30
Fraud rate (P(target=1))



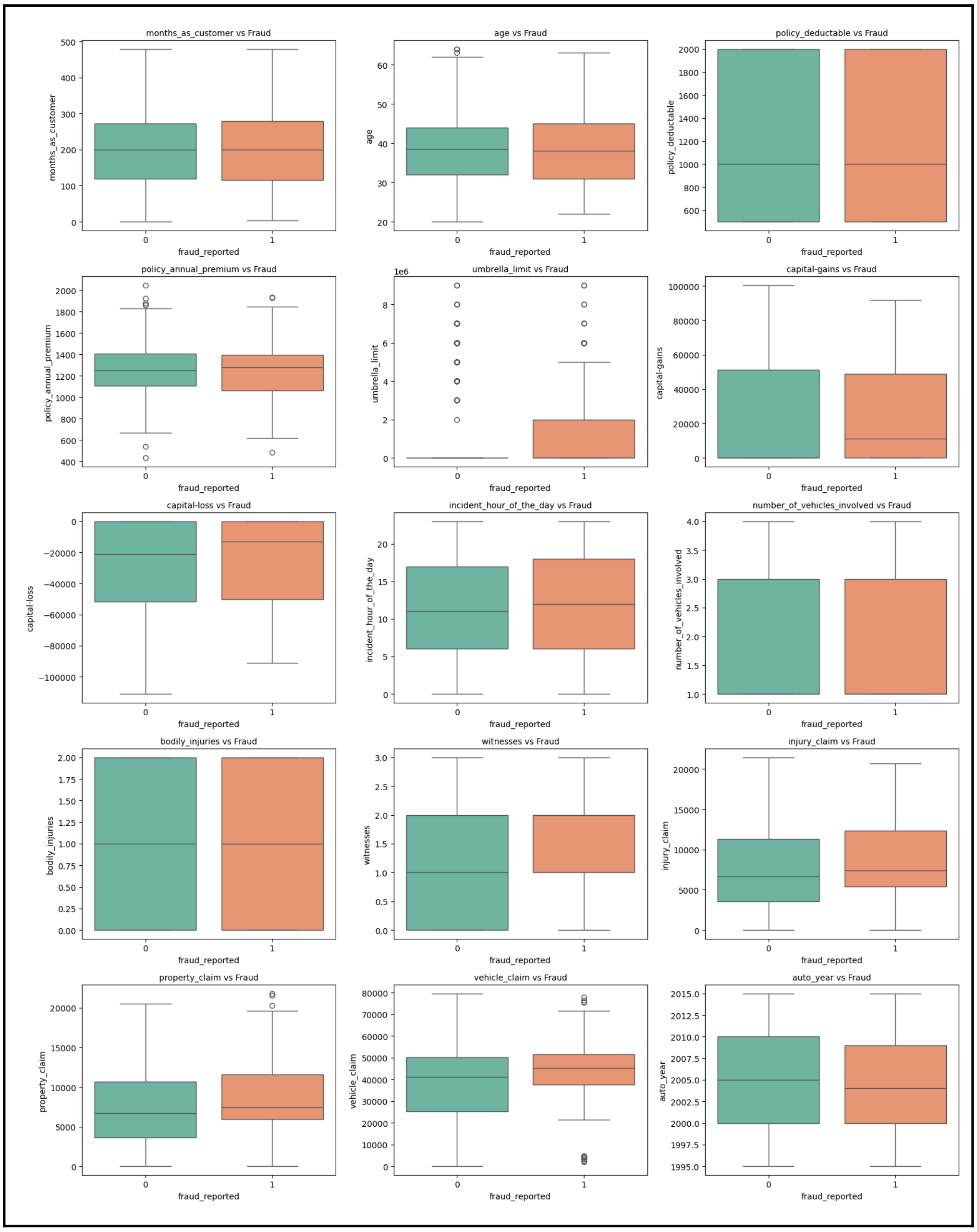

```
#output
n_levels  max_rate  min_rate  rate_range  \
feature
insured_hobbies                20  0.906250  0.075000
0.831250
incident_severity               4  0.609375  0.065574
0.543801
auto_model                     39  0.500000  0.000000
0.500000
incident_state                  7  0.421053  0.151316
0.269737
authorities_contacted           5  0.316176  0.088235
0.227941
collision_type                  4  0.297170  0.087302
```

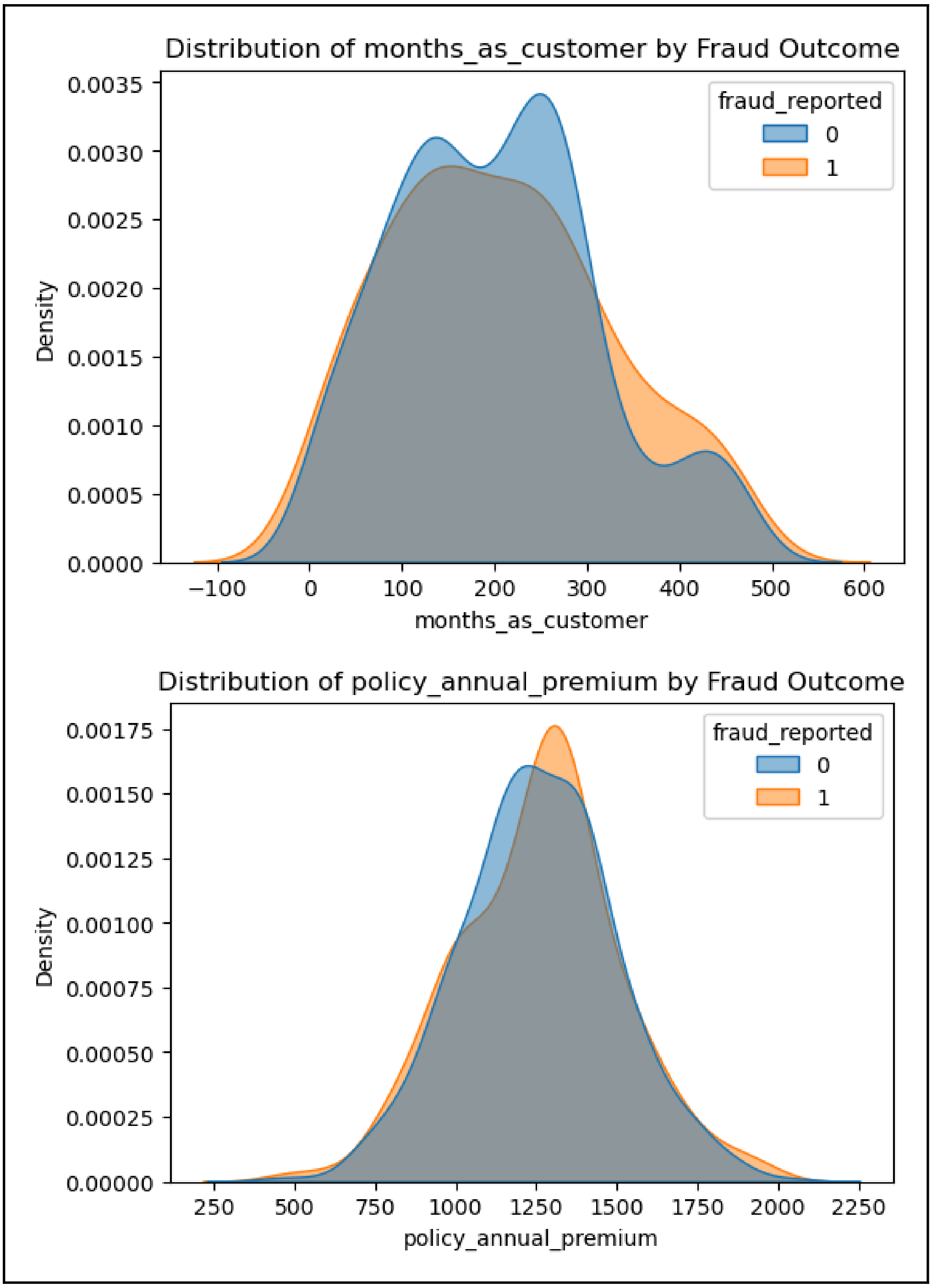
```
0.209868
incident_type                 4  0.287234  0.081967
0.205267
insured_occupation           14  0.344828  0.142857
0.201970
auto_make                    14  0.326923  0.140000
0.186923
insured_relationship          6  0.299213  0.173554
0.125659
property_damage               3  0.290323  0.176230
0.114093
incident_city                 7  0.298077  0.200000
0.098077
insured_education_level       7  0.283186  0.209091
0.074095
policy_csl                    3  0.268000  0.219626
0.048374
police_report_available       3  0.255144  0.229665
0.025479
policy_state                  3  0.253112  0.243697
0.009415
insured_sex                   2  0.249330  0.245399
0.003931

                         weighted_rate  base_rate  is_weak  \
feature
insured_hobbies               0.247496   0.247496    False
incident_severity             0.247496   0.247496    False
auto_model                    0.247496   0.247496    False
incident_state                0.247496   0.247496    False
authorities_contacted         0.247496   0.247496    False
collision_type                0.247496   0.247496    False
incident_type                 0.247496   0.247496    False
insured_occupation            0.247496   0.247496    False
auto_make                     0.247496   0.247496    False
insured_relationship          0.247496   0.247496    False
property_damage               0.247496   0.247496    False
incident_city                 0.247496   0.247496    False
insured_education_level       0.247496   0.247496    False
policy_csl                    0.247496   0.247496     True
police_report_available       0.247496   0.247496     True
policy_state                  0.247496   0.247496     True
insured_sex                   0.247496   0.247496     True

                         n_low_support_levels
feature
insured_hobbies                             1
incident_severity                           0
auto_model                                 25
incident_state                              2
authorities_contacted                       0
collision_type                              0
incident_type                               0
insured_occupation                          0
auto_make                                   0
insured_relationship                        0
property_damage                             0
incident_city                               0
insured_education_level                     0
policy_csl                                  0
```

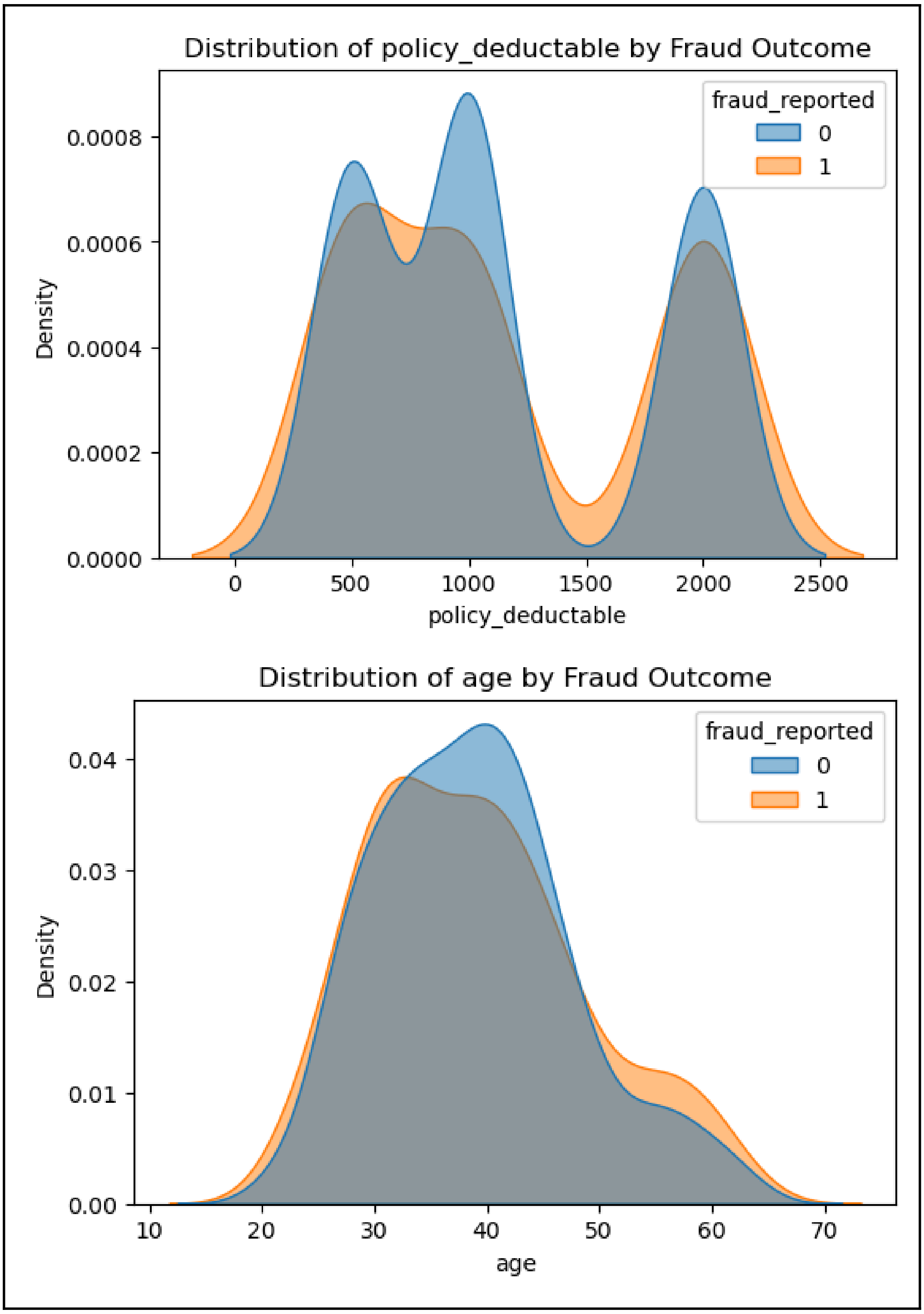
```
police_report_available                    0
policy_state                               0
insured_sex                                0
  property_damage  total_count  fraud_cases  fraud_rate
low_support
0         unknown          248           72    0.290323
False
2              NO          244           43    0.176230
False
1             YES          207           58    0.280193
False
```

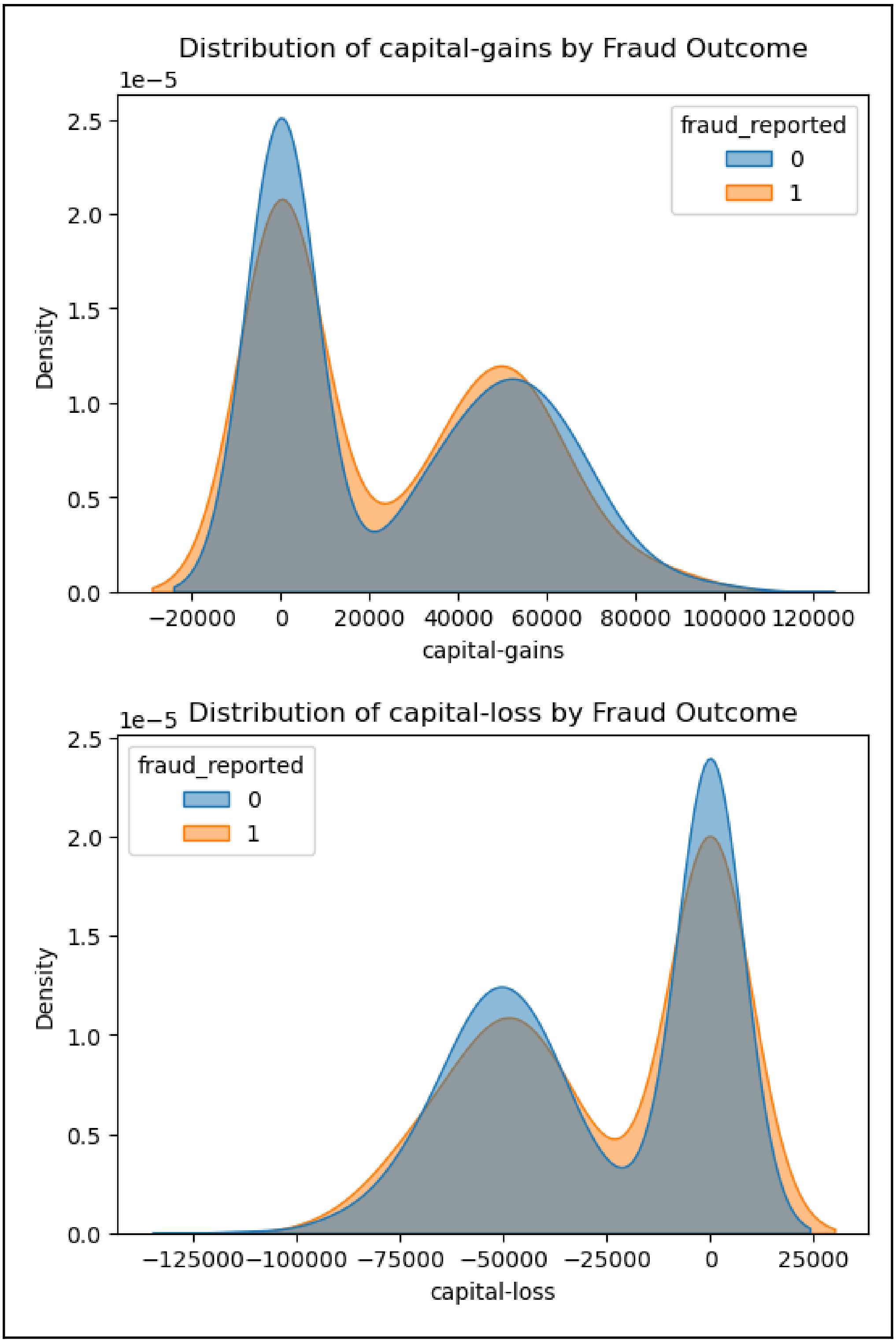
4.4.2. Explore the relationships between numerical features and the target variable to understand their impact on the target outcome using appropriate visualisation techniques to identify trends and potential interactions

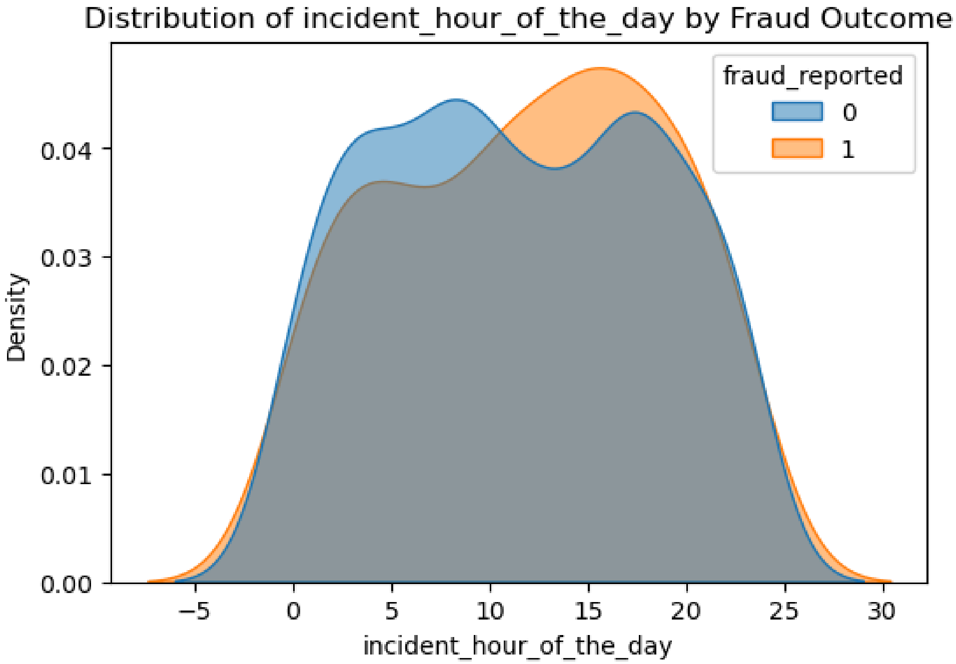
```
# Visualise the relationship between numerical features and the target
variable to understand their impact on the target outcome


# Ensure target is numeric (0/1)
y = y_train if isinstance(y_train, pd.Series) else y_train.squeeze()
Xy = X_train.copy()
Xy["fraud_reported"] = y

# Select numerical columns
numerical_cols = X_train.select_dtypes(include=["int64",
"float64"]).columns.tolist()

# Plot boxplots for each numerical column vs target
plt.figure(figsize=(16, 20))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(5, 3, i)
    sns.boxplot(x="fraud_reported", y=col, data=Xy, palette="Set2")
    plt.title(f"{col} vs Fraud", fontsize=10)
plt.tight_layout()
plt.show()

# Plot distribution (histogram + KDE) for each numeric column split by fraud
for col in numerical_cols:
    plt.figure(figsize=(6,4))
    sns.kdeplot(data=Xy, x=col, hue="fraud_reported",
common_norm=False, fill=True, alpha=0.5)
    plt.title(f"Distribution of {col} by Fraud Outcome")
    plt.show()
```

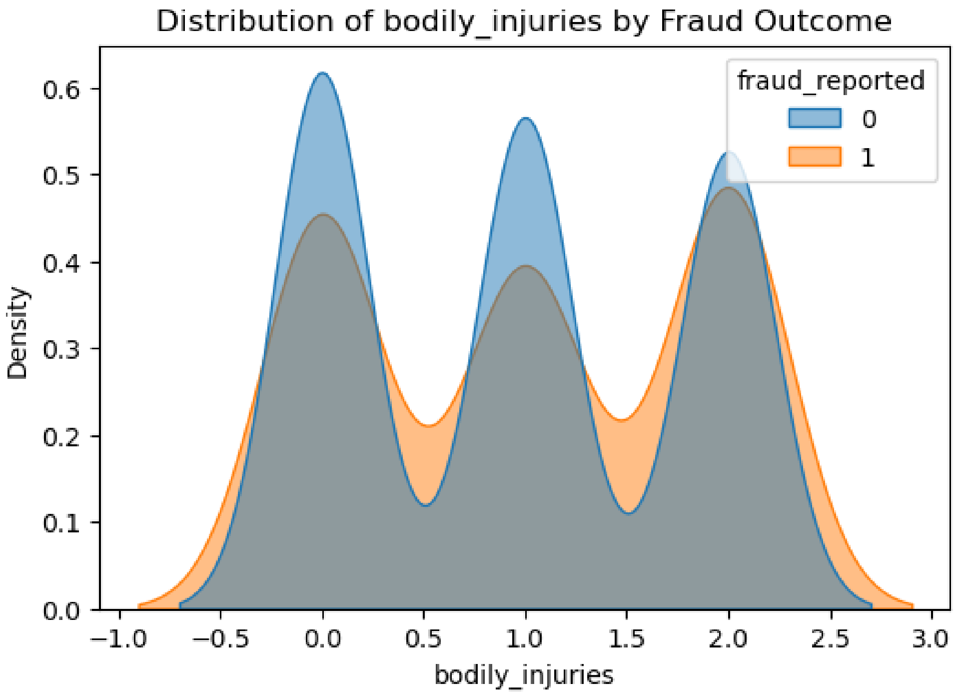

months_as_customer vs Fraud
months_as_customer
fraud_reported
age vs Fraud
age
fraud_reported
policy_deductable vs Fraud
policy_deductable
fraud_reported
policy_annual_premium vs Fraud
policy_annual_premium
fraud_reported
umbrella_limit vs Fraud
umbrella_limit
fraud_reported
capital-gains vs Fraud
capital-gains
fraud_reported
capital-loss vs Fraud
capital-loss
fraud_reported
incident_hour_of_the_day vs Fraud
incident_hour_of_the_day
fraud_reported
number_of_vehicles_involved vs Fraud
number_of_vehicles_involved
fraud_reported
bodily_injuries vs Fraud
bodily_injuries
fraud_reported
witnesses vs Fraud
witnesses
fraud_reported
injury_claim vs Fraud
injury_claim
fraud_reported
property_claim vs Fraud
property_claim
fraud_reported
vehicle_claim vs Fraud
vehicle_claim
fraud_reported
auto_year vs Fraud
auto_year
fraud_reported

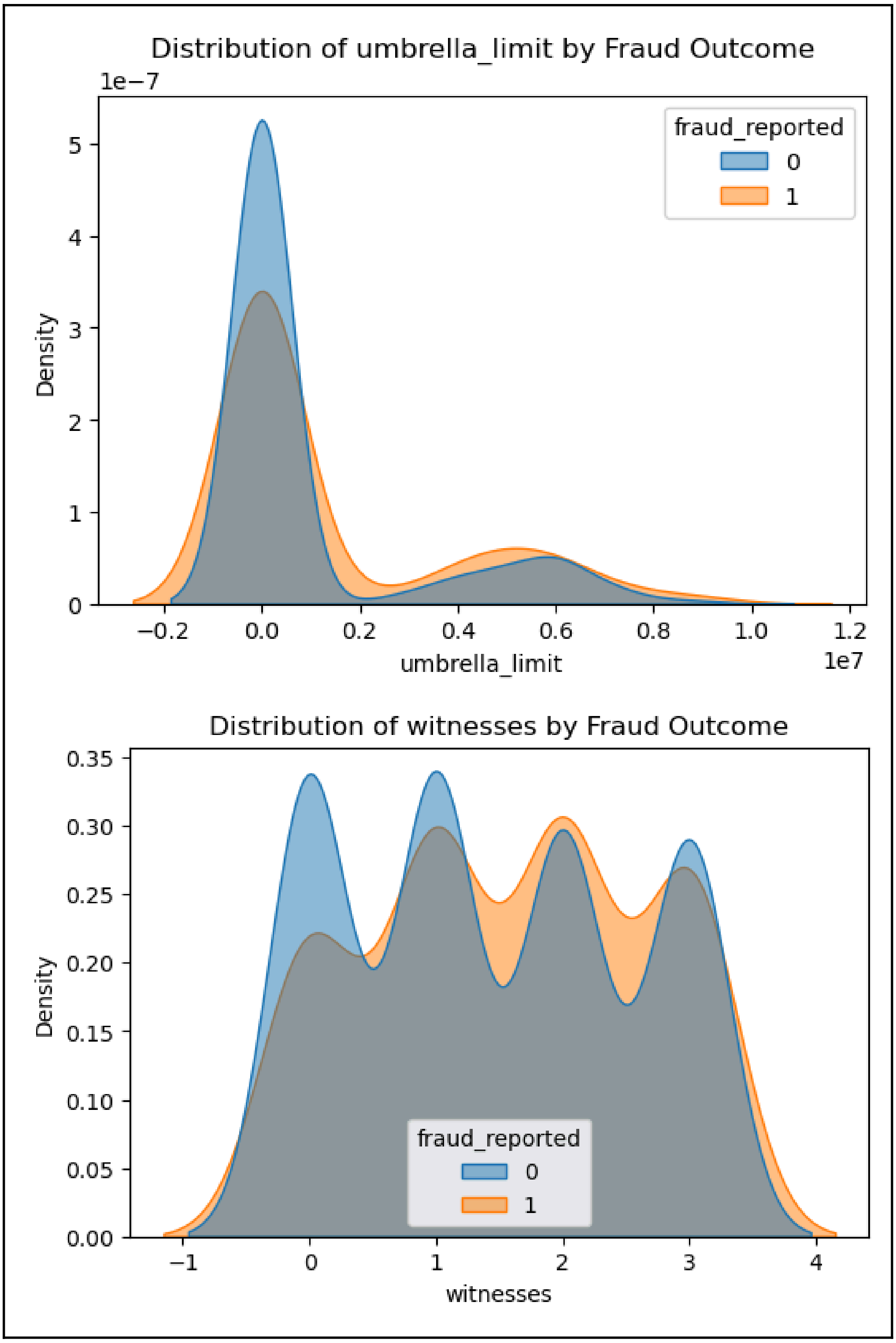

Distribution of months_as_customer by Fraud Outcome
fraud_reported
0
1
Density
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035
−100
0
100
200
300
400
500
600
months_as_customer
Distribution of policy_annual_premium by Fraud Outcome
fraud_reported
0
1
Density
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
0.00175
250
500
750
1000
1250
1500
1750
2000
2250
policy_annual_premium


Distribution of policy_deductable by Fraud Outcome
fraud_reported
0
1
Density
0.0008
0.0006
0.0004
0.0002
0.0000
0
500
1000
1500
2000
2500
policy_deductable
Distribution of age by Fraud Outcome
fraud_reported
0
1
Density
0.04
0.03
0.02
0.01
0.00
10
20
30
40
50
60
70
age

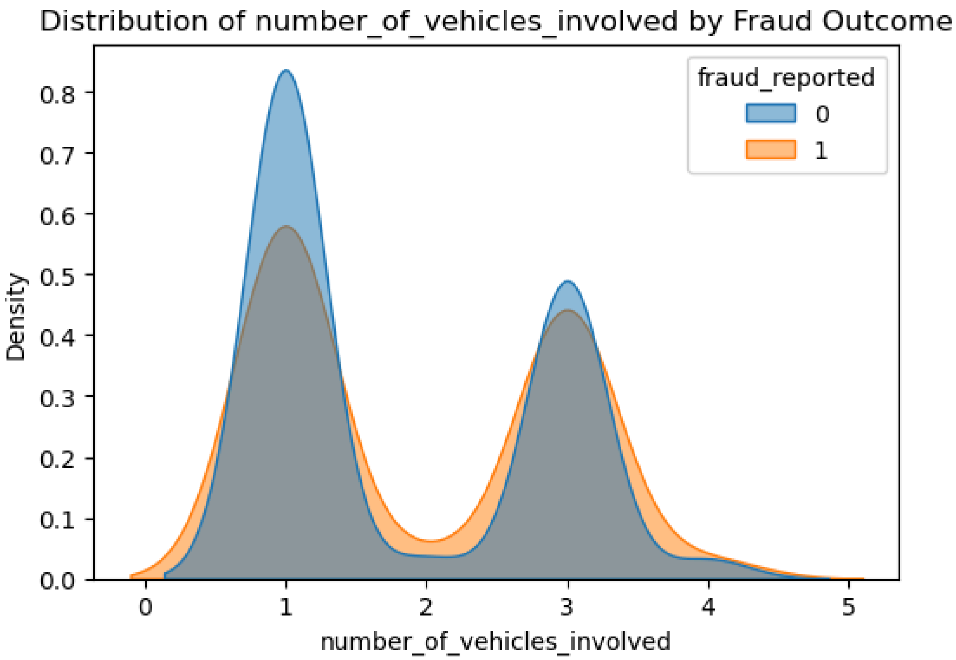

Distribution of capital-gains by Fraud Outcome
1e−5
2.5
2.0
1.5
1.0
0.5
0.0
Density
fraud_reported
0
1
−20000
0
20000
40000
60000
80000
100000
120000
capital-gains
Distribution of capital-loss by Fraud Outcome
1e−5
2.5
2.0
1.5
1.0
0.5
0.0
Density
fraud_reported
0
1
−125000
−100000
−75000
−50000
−25000
0
25000
capital-loss

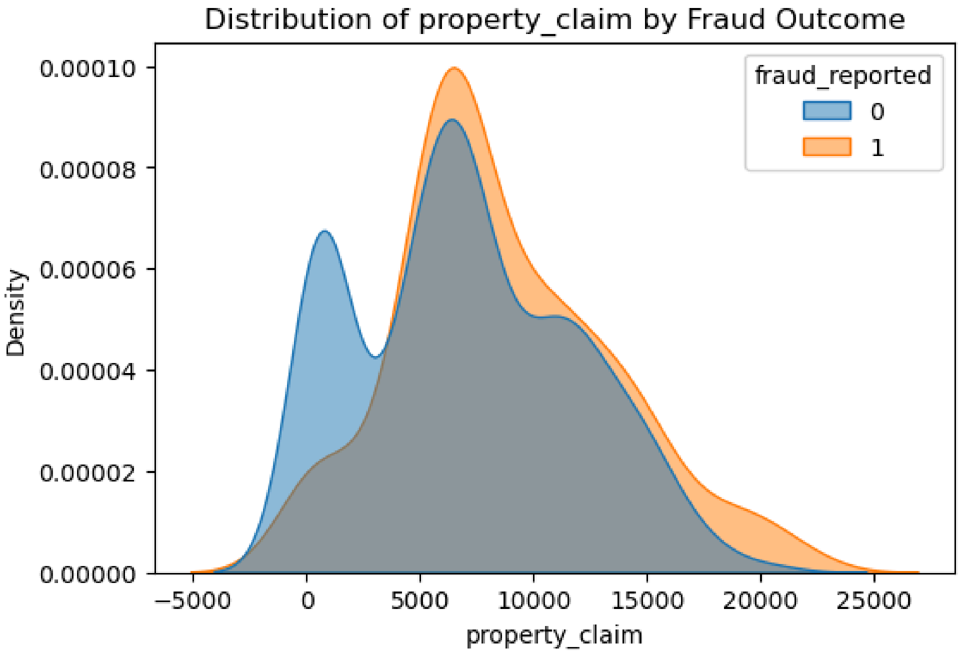

Distribution of incident_hour_of_the_day by Fraud Outcome
fraud_reported
0
1
Density
0.00
0.01
0.02
0.03
0.04
−5
0
5
10
15
20
25
30
incident_hour_of_the_day

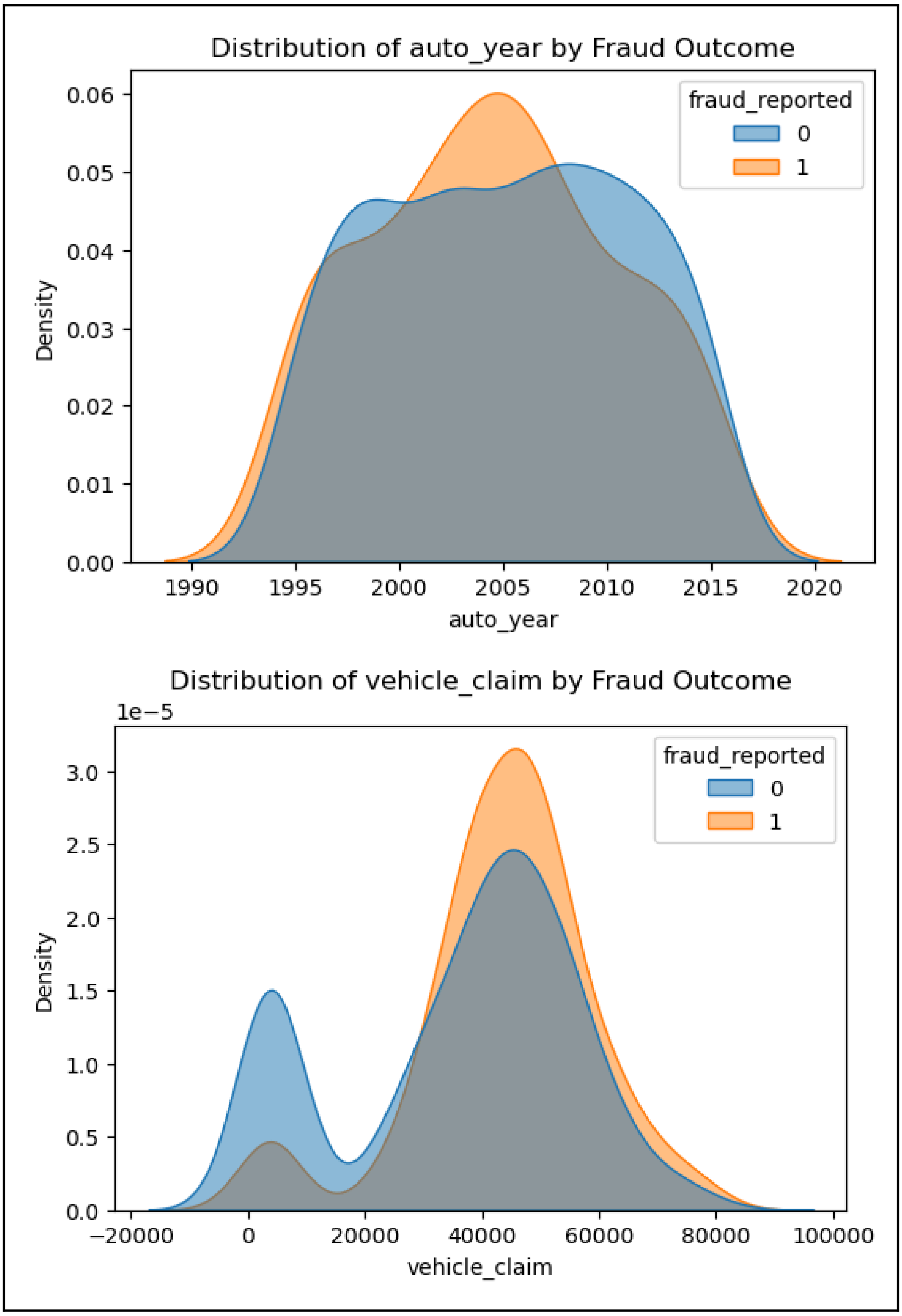

Distribution of bodily_injuries by Fraud Outcome
fraud_reported
0
1
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
bodily_injuries

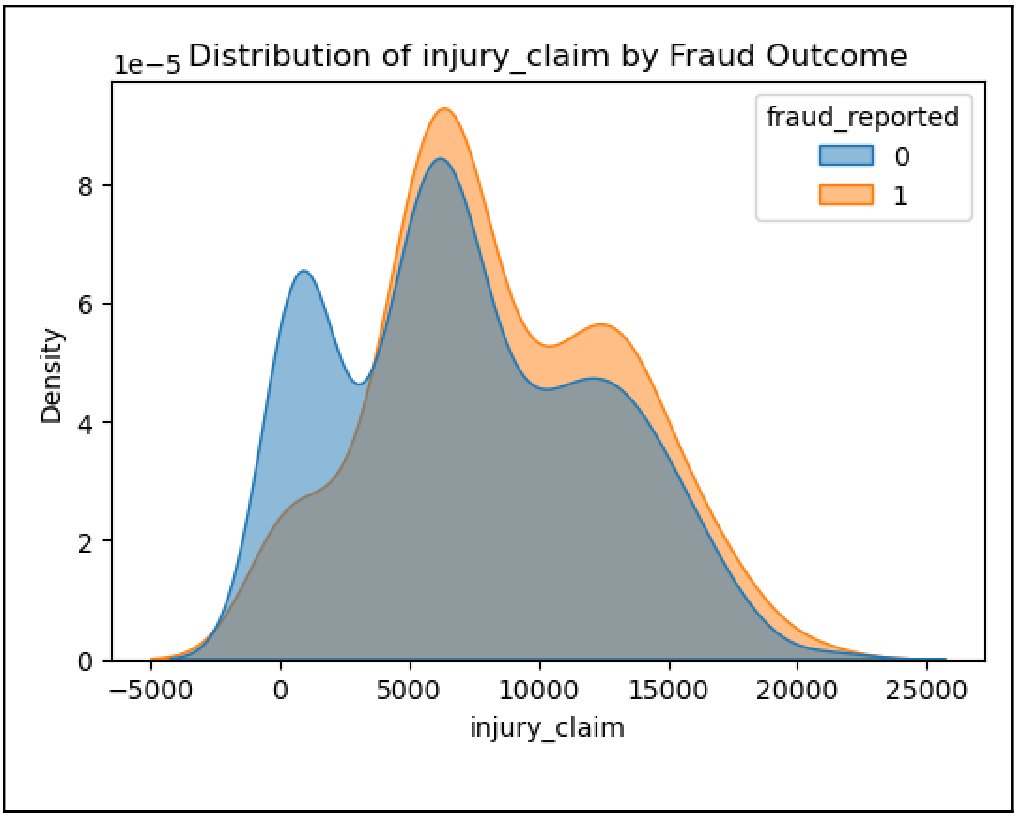


Distribution of umbrella_limit by Fraud Outcome
1e−7
5
4
3
2
1
0
Density
fraud_reported
0
1
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
umbrella_limit
1e7
Distribution of witnesses by Fraud Outcome
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
fraud_reported
0
1
−1
0
1
2
3
4
witnesses


Distribution of number_of_vehicles_involved by Fraud Outcome
fraud_reported
0
1
Density
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
5
number_of_vehicles_involved


Distribution of property_claim by Fraud Outcome
fraud_reported
0
1
Density
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
−5000
0
5000
10000
15000
20000
25000
property_claim


Distribution of auto_year by Fraud Outcome
fraud_reported
0
1
Density
0.06
0.05
0.04
0.03
0.02
0.01
0.00
1990
1995
2000
2005
2010
2015
2020
auto_year
Distribution of vehicle_claim by Fraud Outcome
1e−5
fraud_reported
0
1
Density
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−20000
0
20000
40000
60000
80000
100000
vehicle_claim



5. Feature Engineering

5.1. Perform resampling

```
# Import RandomOverSampler from imblearn library
from imblearn.over_sampling import RandomOverSampler

# Perform resampling on training data

from collections import Counter

# Before resampling
print("Class distribution before resampling:", Counter(y_train))

# Apply RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_train_res, y_train_res = ros.fit_resample(X_train, y_train)

# After resampling
print("Class distribution after resampling:", Counter(y_train_res))

# Shapes
print("Original training shape:", X_train.shape, y_train.shape)
print("Resampled training shape:", X_train_res.shape, y_train_res.shape)
```

```
#output
Class distribution before resampling: Counter({0: 526, 1: 173})
Class distribution after resampling: Counter({0: 526, 1: 526})
Original training shape: (699, 32) (699,)
Resampled training shape: (1052, 32) (1052,)
```

5.2. Feature Creation

```
# Create new features based on your understanding for both training and
validation data
import numpy as np
import pandas as pd

def create_new_features(df):
    df_new = df.copy()

    # --- Policy / Customer ---
    df_new["policy_premium_per_month"] =
df_new["policy_annual_premium"] / 12
    df_new["tenure_bucket"] = pd.cut(
        df_new["months_as_customer"],
        bins=[0, 12, 36, 60, 120, np.inf],
        labels=["<1yr", "1-3yr", "3-5yr", "5-10yr", "10+yr"]
    )

    # --- Claims ---
    df_new["claim_component_sum"] = (
        df_new["injury_claim"] + df_new["property_claim"] +
df_new["vehicle_claim"]
    )
    df_new["claim_component_nonzero"] = (
        (df_new[["injury_claim", "property_claim", "vehicle_claim"]] >
0).sum(axis=1)
    )
    df_new["avg_claim_component"] = (
        df_new["claim_component_sum"] /
(df_new["claim_component_nonzero"] + 1e-5)
    )
    df_new["is_single_component_claim"] = (
        (df_new["claim_component_nonzero"] == 1).astype(int)
    )

    # --- Ratios / interactions ---
    df_new["claim_to_premium_ratio"] = (
        df_new["claim_component_sum"] / (df_new["policy_annual_premium"]
+ 1e-5)
    )
    df_new["age_x_premium"] = df_new["age"] *
df_new["policy_annual_premium"]
```

```
    df_new["claim_per_vehicle"] = (
        df_new["claim_component_sum"] /
(df_new["number_of_vehicles_involved"] + 1e-5)
    )

    # --- Buckets ---
    df_new["age_bucket"] = pd.cut(
        df_new["age"],
        bins=[0, 25, 40, 60, np.inf],
        labels=["young", "adult", "mid-age", "senior"]
    )
    df_new["vehicle_age"] = 2025 - df_new["auto_year"]

    return df_new

# Apply to both train & validation sets
X_train_fe = create_new_features(X_train_res)
X_test_fe   = create_new_features(X_test)

print("Train shape before:", X_train_res.shape, "after feature engineering:",
X_train_fe.shape)
print("Validation shape before:", X_test.shape, "after feature engineering:",
X_test_fe.shape)
```

```
#output

Train shape before: (1052, 32) after feature engineering:
(1052, 43)
Validation shape before: (300, 32) after feature engineering:
(300, 43)
```

5.3. Handle redundant columns

```
# Drop redundant columns from training and validation data
X_train_fe["claim_component_sum"] = (
    X_train_fe["injury_claim"] +
    X_train_fe["property_claim"] +
    X_train_fe["vehicle_claim"]
)

X_test_fe["claim_component_sum"] = (
    X_test_fe["injury_claim"] +
    X_test_fe["property_claim"] +
    X_test_fe["vehicle_claim"]
)
drop_cols = ["injury_claim", "property_claim",
"vehicle_claim","months_as_customer","policy_state","insured_sex"]
```

```
X_train_fe = X_train_fe.drop(columns=drop_cols, errors="ignore")
X_test_fe   = X_test_fe.drop(columns=drop_cols, errors="ignore")

# Check the data
X_train_fe.head()
```

5.4. Combine values in Categorical Columns

```
# Combine categories that have low frequency or provide limited predictive information
def collapse_rare_levels(df, col, min_count=50, new_label="Other"):
    """
    Replace rare levels in a categorical column with 'Other'.
    """
    value_counts = df[col].value_counts()
    rare_levels = value_counts[value_counts < min_count].index
    df[col] = df[col].replace(rare_levels, new_label)
    return df

categorical_cols = [
    "auto_model", "insured_hobbies", "incident_state", "insured_occupation"
]

for col in categorical_cols:
    if col in X_train_fe.columns:
        X_train_fe = collapse_rare_levels(X_train_fe, col, min_count=50)
        X_test_fe   = collapse_rare_levels(X_test_fe, col, min_count=50)

# Inspect results
for col in categorical_cols:
    print(f"\n{col} levels after collapsing:")
    print(X_train_fe[col].value_counts())


###OUTPUT####
auto_model levels after collapsing:
auto_model
Other    1002
A5         50
Name: count, dtype: int64

insured_hobbies levels after collapsing:
insured_hobbies
Other             431
chess             101
paintball          73
reading            65
bungie-jumping     64
exercise           56
skydiving          55
base-jumping       53
```

```
yachting           53
board-games        51
hiking             50
Name: count, dtype: int64

incident_state levels after collapsing:
incident_state
SC       283
NY       257
WV       194
NC       135
VA       109
Other     74
Name: count, dtype: int64

insured_occupation levels after collapsing:
insured_occupation
exec-managerial      99
machine-op-inspct    96
prof-specialty       95
Other                93
transport-moving     92
tech-support         86
craft-repair         83
sales                82
armed-forces         73
adm-clerical         71
other-service        66
priv-house-serv      63
farming-fishing      53
Name: count, dtype: int64
```

5.5. Dummy variable creation

5.5.1. Identify categorical columns for dummy variable creation

```
# Identify the categorical columns for creating dummy variables
# Identify categorical columns
categorical_cols =
X_train_fe.select_dtypes(include=["category"]).columns.tolist()

# Identify numeric columns
numeric_cols = X_train_fe.select_dtypes(include=["int64",
"float64"]).columns.tolist()

print("Categorical columns for dummies:", categorical_cols)
print("Numeric columns:", numeric_cols)

###Output###
Categorical columns for dummies: ['policy_csl',
'insured_education_level', 'insured_occupation',
'insured_hobbies', 'insured_relationship',
'incident_type', 'collision_type', 'incident_severity',
'authorities_contacted', 'incident_state',
```

```
'incident_city', 'property_damage',
'police_report_available', 'auto_make', 'auto_model',
'tenure_bucket', 'age_bucket']
Numeric columns: ['age', 'policy_deductable',
'policy_annual_premium', 'umbrella_limit',
'capital-gains', 'capital-loss',
'incident_hour_of_the_day',
'number_of_vehicles_involved', 'bodily_injuries',
'witnesses', 'auto_year', 'policy_premium_per_month',
'claim_component_sum', 'claim_component_nonzero',
'avg_claim_component', 'is_single_component_claim',
'claim_to_premium_ratio', 'age_x_premium',
'claim_per_vehicle', 'vehicle_age']
```

5.5.2. Create dummy variables for categorical columns in training data

```
# Create dummy variables using the 'get_dummies' for categorical
columns in training data


# Create dummy variables (one-hot encoding)
X_train_dummies = pd.get_dummies(X_train_fe,
columns=categorical_cols, drop_first=True)

print("Original train shape:", X_train_fe.shape)
print("After get_dummies shape:", X_train_dummies.shape)
print("New columns added:", X_train_dummies.shape[1] -
X_train_fe.shape[1])
bool_cols =
X_train_dummies.select_dtypes(include=["bool"]).columns
X_train_dummies[bool_cols] =
X_train_dummies[bool_cols].astype(int)

###OUTPUT###
Original train shape: (1052, 37)
After get_dummies shape: (1052, 104)
New columns added: 67
```

5.5.3. Create dummy variables for categorical columns in validation data

```
# Create dummy variables using the 'get_dummies' for categorical columns
in validation data
X_test_dummies = pd.get_dummies(X_test_fe, columns=categorical_cols,
```

```
drop_first=True)

print("Original train shape:", X_test_fe.shape)
print("After get_dummies shape:", X_test_dummies.shape)
print("New columns added:", X_test_dummies.shape[1] -
X_test_fe.shape[1])
bool_cols = X_test_dummies.select_dtypes(include=["bool"]).columns
X_test_dummies[bool_cols] = X_test_dummies[bool_cols].astype(int)

X_train_dummies, X_test_dummies = X_train_dummies.align(
   X_test_dummies, join="left", axis=1, fill_value=0
)


###Output###
Original train shape: (300, 37)
After get_dummies shape: (300, 79)
New columns added: 42
```

5.6. Feature scaling

```
# Import the necessary scaling tool from scikit-learn
from sklearn.preprocessing import StandardScaler

# Scale the numeric features present in the training data
scaler = StandardScaler()
scaler.fit(X_train_dummies[numeric_cols])
X_train_scaled = X_train_dummies.copy()
X_train_scaled[numeric_cols] = scaler.transform(X_train_dummies[numeric_cols])

# Scale the numeric features present in the validation data
X_test_scaled   = X_test_dummies.copy()
X_test_scaled[numeric_cols]   = scaler.transform(X_test_dummies[numeric_cols])
```

6. Model Building

6.1. Feature selection

6.1.1. Import necessary libraries

```
# Import necessary libraries
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt
```

6.1.2. Perform feature selection

```
# Apply RFECV to identify the most relevant features
logreg = LogisticRegression(max_iter=1000, solver='liblinear',
class_weight='balanced')

# RFECV with 5-fold Stratified CV
rfecv = RFECV(
    estimator=logreg,
    step=1,                 # remove one feature at a time
    cv=StratifiedKFold(5),    # stratified to preserve fraud/non-fraud
ratio
    scoring='roc_auc',        # optimize for ROC-AUC (good for
imbalanced datasets)
    n_jobs=-1
)

# Fit RFECV on training data
rfecv.fit(X_train_scaled, y_train_res)

# Display the features ranking by RFECV in a DataFrame
rfecv_results = pd.DataFrame({
    "Feature": X_train_scaled.columns,
    "Rank": rfecv.ranking_,
    "Selected": rfecv.support_
}).sort_values("Rank")

print("RFECV Feature Ranking:")
print(rfecv_results)

####OUTPUT###
RFECV Feature Ranking:
                                Feature  Rank  Selected
0                                   age     1      True
74              incident_city_Hillsdale     1      True
73               incident_city_Columbus     1      True
72                    incident_state_WV     1      True
71                    incident_state_VA     1      True
..                                  ...   ...       ...
103                   age_bucket_senior     1      True
85                  auto_make_Chevrolet     2     False
29       insured_occupation_craft-repair     3     False
55                 incident_type_Parked Car     4     False
31   insured_occupation_farming-fishing     5     False

[104 rows x 3 columns]
```

6.1.3. Retain the selected features

```
# Put columns selected by RFECV into variable 'col'
col = X_train_scaled.columns[rfecv.support_].tolist()
print("Number of features selected:", len(col))
print("Selected features (col):", col)

##OUTPUT##
Number of features selected: 100
Selected features (col): ['age', 'policy_deductable',
'policy_annual_premium', 'umbrella_limit',
'capital-gains', 'capital-loss',
'incident_hour_of_the_day',
'number_of_vehicles_involved', 'bodily_injuries',
'witnesses', 'auto_year', 'policy_premium_per_month',
'claim_component_sum', 'claim_component_nonzero',
'avg_claim_component', 'is_single_component_claim',
'claim_to_premium_ratio', 'age_x_premium',
'claim_per_vehicle', 'vehicle_age',
'policy_csl_250/500', 'policy_csl_500/1000',
'insured_education_level_College',
'insured_education_level_High School',
'insured_education_level_JD',
'insured_education_level_MD',
'insured_education_level_Masters',
'insured_education_level_PhD',
'insured_occupation_armed-forces',
'insured_occupation_exec-managerial',
'insured_occupation_Other',
'insured_occupation_machine-op-inspct',
'insured_occupation_other-service',
'insured_occupation_priv-house-serv',
'insured_occupation_prof-specialty',
'insured_occupation_sales',
'insured_occupation_tech-support',
'insured_occupation_transport-moving',
'insured_hobbies_Other', 'insured_hobbies_board-games',
'insured_hobbies_bungie-jumping',
'insured_hobbies_chess', 'insured_hobbies_exercise',
'insured_hobbies_hiking', 'insured_hobbies_paintball',
'insured_hobbies_reading', 'insured_hobbies_skydiving',
'insured_hobbies_yachting',
'insured_relationship_not-in-family',
'insured_relationship_other-relative',
'insured_relationship_own-child',
```

```
'insured_relationship_unmarried',
'insured_relationship_wife', 'incident_type_Single
Vehicle Collision', 'incident_type_Vehicle Theft',
'collision_type_Rear Collision', 'collision_type_Side
Collision', 'collision_type_unknown',
'incident_severity_Minor Damage',
'incident_severity_Total Loss',
'incident_severity_Trivial Damage',
'authorities_contacted_Fire',
'authorities_contacted_Other',
'authorities_contacted_Police',
'authorities_contacted_unknown', 'incident_state_NY',
'incident_state_Other', 'incident_state_SC',
'incident_state_VA', 'incident_state_WV',
'incident_city_Columbus', 'incident_city_Hillsdale',
'incident_city_Northbend', 'incident_city_Northbrook',
'incident_city_Riverwood', 'incident_city_Springfield',
'property_damage_YES', 'property_damage_unknown',
'police_report_available_YES',
'police_report_available_unknown', 'auto_make_Audi',
'auto_make_BMW', 'auto_make_Dodge', 'auto_make_Ford',
'auto_make_Honda', 'auto_make_Jeep',
'auto_make_Mercedes', 'auto_make_Nissan',
'auto_make_Saab', 'auto_make_Suburu',
'auto_make_Toyota', 'auto_make_Volkswagen',
'auto_model_A5', 'tenure_bucket_1-3yr',
'tenure_bucket_3-5yr', 'tenure_bucket_5-10yr',
'tenure_bucket_10+yr', 'age_bucket_adult',
'age_bucket_mid-age', 'age_bucket_senior']
```

6.2. Build Logistic Regression Model

6.2.1. Select relevant features and add constant in training data

```
# Select only the columns selected by RFECV

X_train_rfecv = X_train_scaled[col].copy()
X_test_rfecv  = X_test_scaled[col].copy()

# Import statsmodels and add constant
import statsmodels.api as sm
X_train_rfecv_const = sm.add_constant(X_train_rfecv,
has_constant="add")
# Check the data
print("Shape with constant:", X_train_rfecv_const.shape)
```

```
print("First few columns:", X_train_rfecv_const.head())
```

6.2.2. Fit logistic regression model

```
# Fit a logistic Regression model on X_train after adding a constant
and output the summary
X_train_rfecv_const_re = X_train_rfecv_const
logit_model = sm.Logit(y_train_res, X_train_rfecv_const)
result = logit_model.fit()

print(result.summary())
##OUTPUT##

Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.316231
         Iterations: 35
                           Logit Regression Results
=========================================================
======================
Dep. Variable:         fraud_reported   No.
Observations:                 1052
Model:                          Logit   Df Residuals:
953
Method:                           MLE   Df Model:
98
Date:                Tue, 09 Sep 2025   Pseudo R-squ.:
0.5438
Time:                        05:54:25   Log-Likelihood:
-332.68
converged:                      False   LL-Null:
-729.19
Covariance Type:            nonrobust   LLR p-value:
2.979e-109
=========================================================
===================================================
                                            coef    std
err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------
---------------------------------------------------
const                                     3.4216
1421.372      0.002      0.998   -2782.416    2789.259
age                                      -0.5721
0.668     -0.857      0.392      -1.881       0.737
policy_deductable                         0.1336
0.116      1.155      0.248      -0.093       0.360
policy_annual_premium                    -0.1740
nan        nan        nan         nan         nan
umbrella_limit                            0.2921
0.118      2.478      0.013       0.061       0.523
capital-gains                             0.1379
0.115      1.194      0.232      -0.088       0.364
capital-loss                             -0.0721
0.116     -0.620      0.535      -0.300       0.156
incident_hour_of_the_day                 -0.1377
0.117     -1.180      0.238      -0.366       0.091
```

```
number_of_vehicles_involved              -0.6196
0.509     -1.218      0.223      -1.617       0.378
bodily_injuries                           0.2030
0.115      1.759      0.079      -0.023       0.429
witnesses                                 0.4500
0.118      3.817      0.000       0.219       0.681
auto_year                                -0.1431
5.85e+06  -2.45e-08      1.000   -1.15e+07    1.15e+07
policy_premium_per_month                 -0.1740
nan        nan        nan         nan         nan
claim_component_sum                      -3.0214
2.150     -1.405      0.160      -7.235       1.192
claim_component_nonzero                   0.2103
0.536      0.392      0.695      -0.841       1.262
avg_claim_component                       2.6595
2.062      1.290      0.197      -1.382       6.701
is_single_component_claim                 1.4653
1.62e+04   9.02e-05      1.000   -3.18e+04    3.18e+04
claim_to_premium_ratio                    0.7455
0.484      1.541      0.123      -0.203       1.694
age_x_premium                             1.3831
0.832      1.662      0.096      -0.248       3.014
claim_per_vehicle                         0.1044
0.604      0.173      0.863      -1.079       1.288
vehicle_age                               0.1431
5.85e+06   2.45e-08      1.000   -1.15e+07    1.15e+07
policy_csl_250/500                        0.5719
0.278      2.056      0.040       0.027       1.117
policy_csl_500/1000                      -0.2904
0.288     -1.007      0.314      -0.855       0.275
insured_education_level_College          -0.0756
0.471     -0.160      0.873      -0.999       0.848
insured_education_level_High School      -0.2499
0.426     -0.587      0.557      -1.084       0.585
insured_education_level_JD                0.8027
0.407      1.972      0.049       0.005       1.601
insured_education_level_MD                0.5856
0.424      1.381      0.167      -0.245       1.417
insured_education_level_Masters           0.2699
0.425      0.635      0.525      -0.563       1.102
insured_education_level_PhD               0.8223
0.434      1.896      0.058      -0.028       1.672
insured_occupation_armed-forces           0.5292
0.530      0.999      0.318      -0.509       1.568
insured_occupation_exec-managerial        0.3991
0.484      0.824      0.410      -0.550       1.348
insured_occupation_Other                 -1.3333
0.455     -2.929      0.003      -2.225      -0.441
insured_occupation_machine-op-inspct     -0.0128
0.454     -0.028      0.977      -0.902       0.877
insured_occupation_other-service         -0.8327
0.544     -1.531      0.126      -1.899       0.233
insured_occupation_priv-house-serv       -1.3099
0.591     -2.217      0.027      -2.468      -0.152
insured_occupation_prof-specialty         0.1805
0.473      0.382      0.703      -0.746       1.107
insured_occupation_sales                 -0.1975
0.563     -0.351      0.726      -1.301       0.906
insured_occupation_tech-support          -0.2887
0.481     -0.600      0.548      -1.232       0.654
```

```
insured_occupation_transport-moving        0.8429
0.438      1.926      0.054      -0.015       1.700
insured_hobbies_Other                     -0.3305
0.497     -0.666      0.506      -1.304       0.643
insured_hobbies_board-games               -0.3108
0.715     -0.435      0.664      -1.713       1.091
insured_hobbies_bungie-jumping            -1.1174
0.727     -1.537      0.124      -2.542       0.308
insured_hobbies_chess                      6.4090
0.839      7.635      0.000       4.764       8.054
insured_hobbies_exercise                  -0.4941
0.668     -0.740      0.459      -1.803       0.815
insured_hobbies_hiking                    -1.0540
0.698     -1.510      0.131      -2.422       0.314
insured_hobbies_paintball                 -0.2730
0.633     -0.431      0.666      -1.515       0.969
insured_hobbies_reading                   -0.6998
0.667     -1.050      0.294      -2.007       0.607
insured_hobbies_skydiving                 -0.9335
0.740     -1.262      0.207      -2.384       0.517
insured_hobbies_yachting                  -0.0363
0.711     -0.051      0.959      -1.429       1.357
insured_relationship_not-in-family         0.9425
0.409      2.303      0.021       0.140       1.745
insured_relationship_other-relative        0.5917
0.396      1.493      0.135      -0.185       1.369
insured_relationship_own-child            -0.6324
0.418     -1.513      0.130      -1.452       0.187
insured_relationship_unmarried             0.9204
0.420      2.194      0.028       0.098       1.743
insured_relationship_wife                 -0.0186
0.429     -0.043      0.965      -0.859       0.822
incident_type_Single Vehicle Collision    -1.3172
1.166     -1.130      0.259      -3.603       0.968
incident_type_Vehicle Theft               -0.3085
0.785     -0.393      0.694      -1.846       1.229
collision_type_Rear Collision              0.4204
0.296      1.421      0.155      -0.160       1.000
collision_type_Side Collision             -0.5707
0.317     -1.800      0.072      -1.192       0.051
collision_type_unknown                    -0.2411
1.452     -0.166      0.868      -3.087       2.605
incident_severity_Minor Damage            -4.5796
0.378    -12.107      0.000      -5.321      -3.838
incident_severity_Total Loss              -3.6086
0.311    -11.603      0.000      -4.218      -2.999
incident_severity_Trivial Damage          -5.4819
0.876     -6.260      0.000      -7.198      -3.765
authorities_contacted_Fire                -0.2997
0.347     -0.862      0.388      -0.981       0.381
authorities_contacted_Other                0.3026
0.344      0.880      0.379      -0.371       0.976
authorities_contacted_Police               0.0204
0.343      0.060      0.953      -0.651       0.692
authorities_contacted_unknown              0.0175
0.879      0.020      0.984      -1.705       1.740
incident_state_NY                         -0.4862
0.409     -1.189      0.234      -1.288       0.315
incident_state_Other                      -0.1878
0.614     -0.306      0.760      -1.391       1.015
```

```
incident_state_SC                                   -0.1800
0.399     -0.451      0.652      -0.962        0.602
incident_state_VA                                    0.7222
0.463      1.558      0.119      -0.186        1.630
incident_state_WV                                   -1.0632
0.429     -2.479      0.013      -1.904       -0.222
incident_city_Columbus                               0.0793
0.398      0.199      0.842      -0.701        0.859
incident_city_Hillsdale                              0.1840
0.427      0.431      0.666      -0.652        1.020
incident_city_Northbend                             -0.0163
0.414     -0.039      0.969      -0.828        0.795
incident_city_Northbrook                            -0.4012
0.458     -0.875      0.381      -1.300        0.497
incident_city_Riverwood                              0.2578
0.434      0.594      0.552      -0.592        1.108
incident_city_Springfield                            0.2338
0.418      0.560      0.576      -0.585        1.052
property_damage_YES                                  0.6012
0.295      2.037      0.042       0.023        1.180
property_damage_unknown                              0.4298
0.276      1.557      0.119      -0.111        0.971
police_report_available_YES                         -0.4169
0.295     -1.413      0.158      -0.995        0.161
police_report_available_unknown                     -0.3038
0.277     -1.099      0.272      -0.846        0.238
auto_make_Audi                                       1.9622
0.646      3.038      0.002       0.696        3.228
auto_make_BMW                                        1.2412
0.513      2.419      0.016       0.236        2.247
auto_make_Dodge                                      1.1514
0.502      2.293      0.022       0.167        2.136
auto_make_Ford                                       0.4200
0.526      0.799      0.424      -0.610        1.451
auto_make_Honda                                      1.4111
0.585      2.410      0.016       0.264        2.559
auto_make_Jeep                                       1.1065
0.541      2.046      0.041       0.046        2.167
auto_make_Mercedes                                   0.2333
0.515      0.453      0.651      -0.777        1.243
auto_make_Nissan                                    -0.0221
0.547     -0.040      0.968      -1.095        1.051
auto_make_Saab                                       0.5727
0.500      1.145      0.252      -0.408        1.553
auto_make_Suburu                                     0.6681
0.513      1.301      0.193      -0.338        1.674
auto_make_Toyota                                     0.7753
0.599      1.295      0.195      -0.398        1.949
auto_make_Volkswagen                                 0.4392
0.564      0.778      0.436      -0.667        1.545
auto_model_A5                                        0.0999
0.746      0.134      0.894      -1.363        1.563
tenure_bucket_1-3yr                                 -0.8109
0.947     -0.856      0.392      -2.668        1.046
tenure_bucket_3-5yr                                 -0.0136
0.959     -0.014      0.989      -1.894        1.866
tenure_bucket_5-10yr                                -0.2252
0.799     -0.282      0.778      -1.791        1.341
tenure_bucket_10+yr                                 -1.2530
0.785     -1.597      0.110      -2.791        0.285
```

```
age_bucket_adult                                    -0.5675
0.656     -0.865      0.387      -1.853       0.718
age_bucket_mid-age                                  -0.6122
0.846     -0.723      0.469      -2.271       1.047
age_bucket_senior                                   -2.2176
1.399     -1.586      0.113      -4.959       0.524
=========================================================
===================================================
```

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
logsk = LogisticRegression(C=1e9)

logsk.fit(X_train_rfecv, y_train_res)
```

6.2.3. Evaluate VIF of features to assess multicollinearity

```
# Import 'variance_inflation_factor'
from statsmodels.stats.outliers_influence import
variance_inflation_factor
# Make a VIF DataFrame for all the variables present

def calculate_vif(X: pd.DataFrame):
    """
    Create a VIF DataFrame for all numeric columns in DataFrame X.
    """
    # Ensure numeric only
    X_numeric = X.select_dtypes(include=[np.number])

    # Add small constant if any column is constant (to avoid division
by zero)
    X_numeric = X_numeric.copy()
    for col in X_numeric.columns:
        if X_numeric[col].nunique() == 1:
            X_numeric[col] = X_numeric[col] + 1e-6

    vif_data = pd.DataFrame()
    vif_data["Feature"] = X_numeric.columns
    vif_data["VIF"] = [
        variance_inflation_factor(X_numeric.values, i)
        for i in range(X_numeric.shape[1])
    ]
    return vif_data.sort_values("VIF", ascending=False)

# Example: run on your one-hot encoded training set
```

```
vif_df = calculate_vif(X_train_rfecv)
print(vif_df[vif_df['VIF'] > 5])
```

6.2.4. Make predictions on training data

```
# Predict the probabilities on the training data
y_train_pred_prob = logsk.predict_proba(X_train_rfecv)[:, 1]

# Ensure it’s a flat 1-D array
y_train_pred_prob = np.asarray(y_train_pred_prob).reshape(-1,)

print("Shape:", y_train_pred_prob.shape)
print("First 10 predicted probabilities:", y_train_pred_prob[:10])

##OUTPUT##
Shape: (1052,)
First 10 predicted probabilities: [0.7028878  0.13683076
0.17926176 0.03150495 0.06332432 0.49219636
 0.86344362 0.02615324 0.24287583 0.25465523]
```

6.2.5. Create a DataFrame that includes actual fraud reported flags, predicted probabilities, and a column indicating predicted classifications based on a cutoff value of 0.5

```
# Create a new DataFrame containing the actual fraud reported flag
and the probabilities predicted by the model
train_pred_df = pd.DataFrame({
   "fraud_reported": y_train_res.values,   # actual
   "pred_prob": y_train_pred_prob               # predicted probability
(fraud)
})
# Create new column indicating predicted classifications based on a
cutoff value of 0.5
cutoff = 0.5
train_pred_df["pred_class"] = (train_pred_df["pred_prob"] >=
cutoff).astype(int)

print(train_pred_df.head())
##OUTPUT##
  fraud_reported  pred_prob  pred_class
0               0   0.702888           1
1               0   0.136831           0
2               0   0.179262           0
3               0   0.031505           0
4               0   0.063324           0
```

6.2.6. Check the accuracy of the model

```
# Import metrics from sklearn for evaluation
from sklearn import metrics

# Check the accuracy of the model
accuracy = metrics.accuracy_score(train_pred_df["fraud_reported"],
                                  train_pred_df["pred_class"])

print(f"Training Accuracy: {accuracy:.4f}")
##OUTPUT##
Training Accuracy: 0.8888
```

6.2.7. Create a confusion matrix based on the predictions made on the training data

```
# Create confusion matrix
cm = metrics.confusion_matrix(train_pred_df["fraud_reported"],
train_pred_df["pred_class"])

# Print numeric confusion matrix
print("Confusion Matrix:\n", cm)
##OUTPUT##
Confusion Matrix:
 [[465  61]
 [ 56 470]]
```

6.2.8. Create variables for true positive, true negative, false positive and false negative

```
# Create variables for true positive, true negative, false positive and
false negative
tn, fp, fn, tp = cm.ravel()

print("True Negatives :", tn)
print("False Positives:", fp)
print("False Negatives:", fn)
print("True Positives :", tp)
##OUTPUT##
True Negatives : 465
False Positives: 61
False Negatives: 56
True Positives : 470
```

6.2.9. Calculate sensitivity, specificity, precision, recall and F1-score

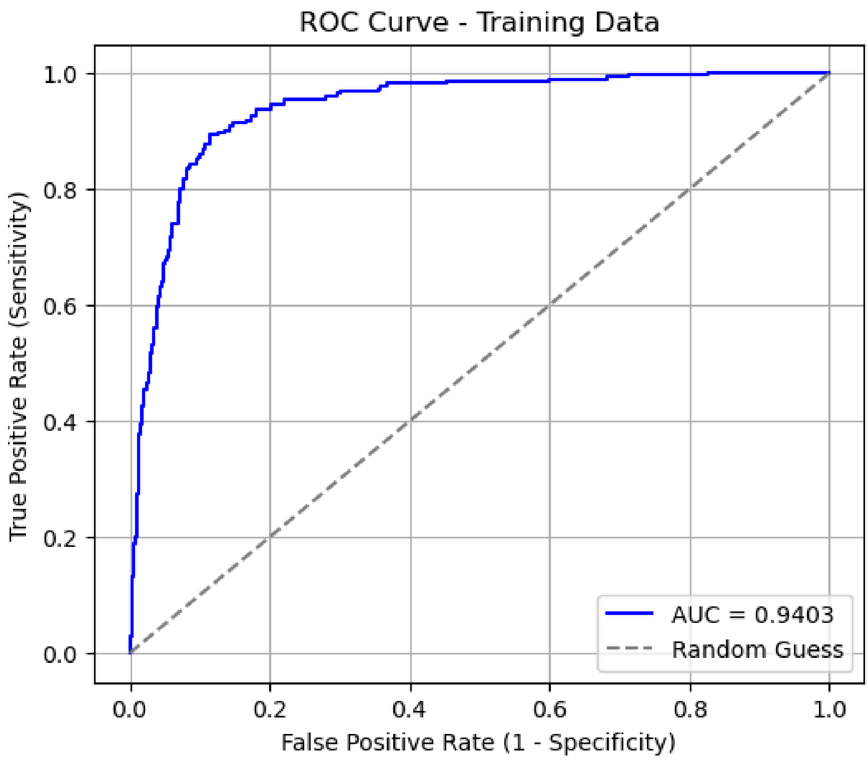
```
# Calculate the sensitivity

sensitivity = tp / (tp + fn)
print(f"Sensitivity (Recall): {sensitivity:.4f}")
# Calculate the specificity
specificity = tn / (tn + fp)
print(f"Specificity: {specificity:.4f}")
# Calculate Precision
precision = tp / (tp + fp)
print(f"Precision  : {precision:.4f}")
# Calculate Recall
recall = tp / (tp + fn)
print(f"Recall     : {recall:.4f}")
# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)
print(f"F1 Score   : {f1_score:.4f}")

##OUTPUT##
Sensitivity (Recall): 0.8935
Specificity: 0.8840
Precision  : 0.8851
Recall     : 0.8935
F1 Score   : 0.8893
```

6.3. Find the Optimal Cutoff

6.3.1. Plot ROC Curve to visualise the trade-off between true positive rate and false positive rate across different classification thresholds

```
# Import libraries or function to plot the ROC curve
from sklearn.metrics import roc_curve, roc_auc_score


# Define ROC function
def plot_roc_curve(y_true, y_prob, title="ROC Curve"):

    # Compute ROC values
    fpr, tpr, thresholds = roc_curve(y_true, y_prob)
    auc_score = roc_auc_score(y_true, y_prob)

    # Plot
    plt.figure(figsize=(6,5))
    plt.plot(fpr, tpr, color="blue", label=f"AUC = {auc_score:.4f}")
    plt.plot([0,1], [0,1], linestyle="--", color="gray", label="Random
Guess")
```

```
    plt.xlabel("False Positive Rate (1 - Specificity)")
    plt.ylabel("True Positive Rate (Sensitivity)")
    plt.title(title)
    plt.legend(loc="lower right")
    plt.grid(True)
    plt.show()

    return auc_score

# Call the ROC function
auc_train = plot_roc_curve(train_pred_df["fraud_reported"],
                           train_pred_df["pred_prob"],
                           title="ROC Curve - Training Data")

print("Training AUC:", auc_train)
```



6.3.2. Predict on training data at various probability cutoffs

```
# Create columns with different probability cutoffs to explore the
impact of cutoff on model performance
# Define cutoffs to test
cutoffs = [0.3, 0.4, 0.5, 0.6, 0.7]

# Create new columns for each cutoff
for c in cutoffs:
    col_name = f"pred_class_{str(c).replace('.', '')}"
```

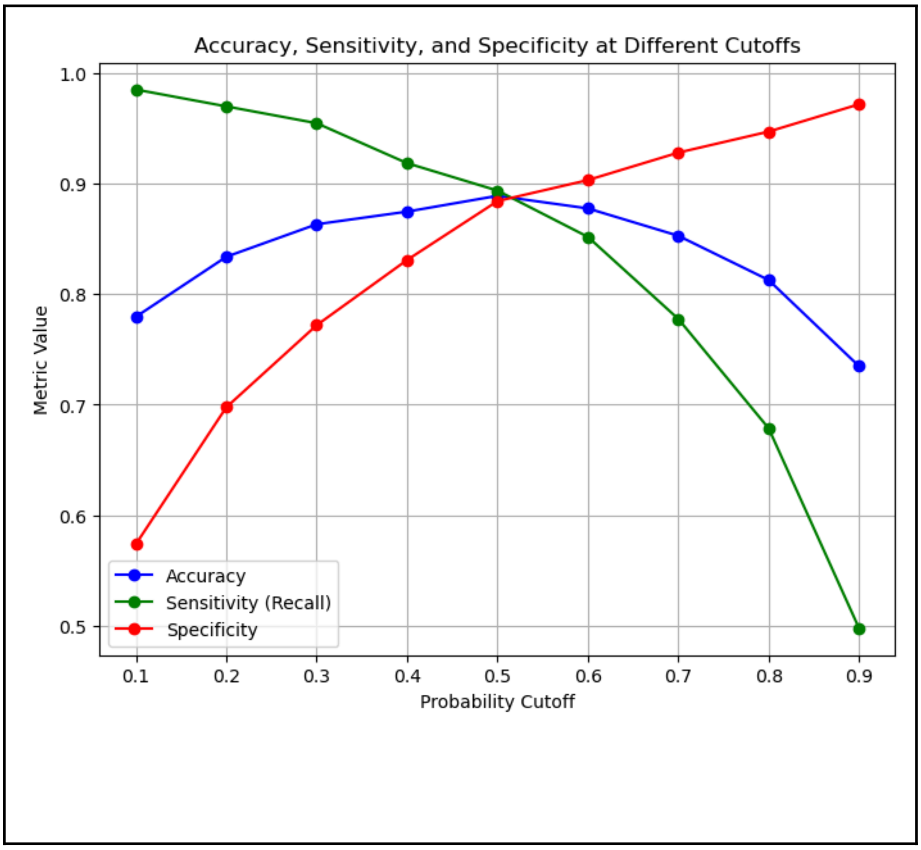
```
    train_pred_df[col_name] = (train_pred_df["pred_prob"] >=
c).astype(int)

print(train_pred_df.head())
##OUTPUT##
fraud_reported  pred_prob  pred_class  pred_class_03
pred_class_04  \
0               0   0.702888          1             1
1
1               0   0.136831          0             0
0
2               0   0.179262          0             0
0
3               0   0.031505          0             0
0
4               0   0.063324          0             0
0

   pred_class_05  pred_class_06  pred_class_07
0              1              1              1
1              0              0              0
2              0              0              0
3              0              0              0
4              0              0              0
```

6.3.3. Plot accuracy, sensitivity, specificity at different values of probability cutoffs

```
# Create a DataFrame to see the values of accuracy, sensitivity, and
specificity at different values of probability cutoffs
from sklearn.metrics import confusion_matrix, accuracy_score

# Define a range of cutoffs (finer granularity than before)
cutoffs = np.arange(0.1, 0.91, 0.1)  # 0.1 to 0.9 in steps of 0.1

results = []

for c in cutoffs:
    # Predicted class at cutoff
    y_pred_class = (train_pred_df["pred_prob"] >= c).astype(int)

    # Confusion matrix
    tn, fp, fn, tp = confusion_matrix(train_pred_df["fraud_reported"],
y_pred_class).ravel()

    # Metrics
    accuracy = accuracy_score(train_pred_df["fraud_reported"],
y_pred_class)
    sensitivity = tp / (tp + fn) if (tp+fn) > 0 else 0
    specificity = tn / (tn + fp) if (tn+fp) > 0 else 0

    results.append({
```

```
        "Cutoff": round(c, 2),
        "Accuracy": round(accuracy, 4),
        "Sensitivity": round(sensitivity, 4),
        "Specificity": round(specificity, 4)
    })

cutoff_df = pd.DataFrame(results)
print(cutoff_df)
##OUTPUT##
   Cutoff  Accuracy  Sensitivity  Specificity
0     0.1    0.7795       0.9848       0.5741
1     0.2    0.8337       0.9696       0.6977
2     0.3    0.8631       0.9544       0.7719
3     0.4    0.8745       0.9183       0.8308
4     0.5    0.8888       0.8935       0.8840
5     0.6    0.8774       0.8517       0.9030
6     0.7    0.8527       0.7776       0.9278
7     0.8    0.8127       0.6787       0.9468
8     0.9    0.7348       0.4981       0.9715
```

```
# Plot accuracy, sensitivity, and specificity at different values of
probability cutoffs
# Plot Accuracy, Sensitivity, Specificity vs Cutoff
plt.figure(figsize=(8,6))

plt.plot(cutoff_df["Cutoff"], cutoff_df["Accuracy"], marker="o",
label="Accuracy", color="blue")
plt.plot(cutoff_df["Cutoff"], cutoff_df["Sensitivity"], marker="o",
label="Sensitivity (Recall)", color="green")
plt.plot(cutoff_df["Cutoff"], cutoff_df["Specificity"], marker="o",
label="Specificity", color="red")

plt.xlabel("Probability Cutoff")
plt.ylabel("Metric Value")
plt.title("Accuracy, Sensitivity, and Specificity at Different Cutoffs")
plt.legend()
plt.grid(True)
plt.show()
```



6.3.4. Create a column for final prediction based on optimal cutoff

```
# Create a column for final prediction based on the optimal cutoff
optimal_cutoff = 0.5
train_pred_df["final_pred"] = (train_pred_df["pred_prob"] >=
optimal_cutoff).astype(int)

print(train_pred_df.head())
```

6.3.5. Calculate the accuracy

```
# Check the accuracy now
accuracy_final = accuracy_score(train_pred_df["fraud_reported"],
train_pred_df["final_pred"])

print(f"Accuracy at optimal cutoff: {accuracy_final:.4f}")

##OUTPUT##
Accuracy at optimal cutoff: 0.8888
```

6.3.6. Create confusion matrix

```
# Create the confusion matrix once again
# Create confusion matrix
cm = metrics.confusion_matrix(train_pred_df["fraud_reported"],
train_pred_df["final_pred"])

# Print numeric confusion matrix
print("Confusion Matrix:\n", cm)
##OUTPUT##
Confusion Matrix:
 [[465  61]
 [ 56 470]]
```

6.3.7. Create variables for true positive, true negative, false positive and false negative

```
# Create variables for true positive, true negative, false positive and
false negative
tn, fp, fn, tp = cm.ravel()

print("True Negatives :", tn)
print("False Positives:", fp)
print("False Negatives:", fn)
print("True Positives :", tp)
##OUTPUT##
True Negatives : 465
False Positives: 61
False Negatives: 56
True Positives : 470
```

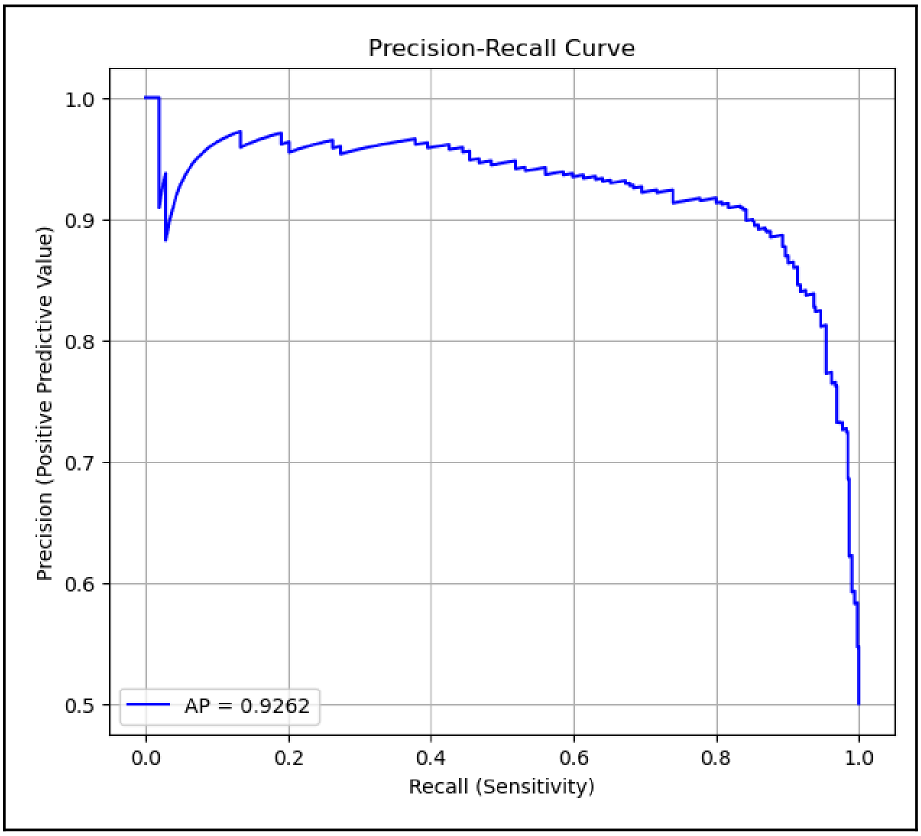
6.3.8. Calculate sensitivity, specificity, precision, recall and F1-score of the model

```
# Calculate the sensitivity

sensitivity = tp / (tp + fn)
print(f"Sensitivity (Recall): {sensitivity:.4f}")
# Calculate the specificity
specificity = tn / (tn + fp)
print(f"Specificity: {specificity:.4f}")
# Calculate Precision
precision = tp / (tp + fp)
print(f"Precision  : {precision:.4f}")
# Calculate Recall
recall = tp / (tp + fn)
print(f"Recall     : {recall:.4f}")
# Calculate F1 Score
```

```
f1_score = 2 * (precision * recall) / (precision + recall)
print(f"F1 Score   : {f1_score:.4f}")

##OUTPUT##
Sensitivity (Recall): 0.8935
Specificity: 0.8840
Precision  : 0.8851
Recall     : 0.8935
F1 Score   : 0.8893
```

6.3.9. Plot precision-recall curve

```
# Import precision-recall curve function
from sklearn.metrics import
precision_recall_curve,average_precision_score
# Plot precision-recall curve
precision, recall, thresholds = precision_recall_curve(
    train_pred_df["fraud_reported"], train_pred_df["pred_prob"]
)
ap_score =
average_precision_score(train_pred_df["fraud_reported"],
train_pred_df["pred_prob"])

plt.figure(figsize=(7,6))
plt.plot(recall, precision, color="blue", label=f"AP = {ap_score:.4f}")
plt.xlabel("Recall (Sensitivity)")
plt.ylabel("Precision (Positive Predictive Value)")
plt.title("Precision-Recall Curve")
plt.legend(loc="lower left")
plt.grid(True)
plt.show()
```



6.4. Build Random Forest Model

6.4.1. Import necessary libraries

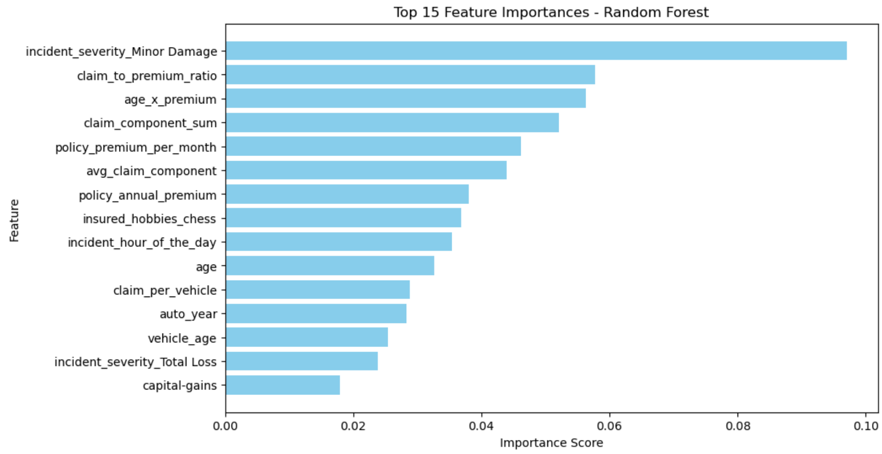
```
# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score,
GridSearchCV
```

6.4.2. Build the random forest model

```
# Build a base random forest model
rf_model = RandomForestClassifier(
    n_estimators=5,        # number of trees
    random_state=42,        # reproducibility
    n_jobs=-1              # use all CPU cores
)
rf_model.fit(X_train_rfecv, y_train_res)
```

6.4.3. Get feature importance scores and select important features

```
# Get feature importance scores from the trained model
importances = rf_model.feature_importances_
# Create a DataFrame to visualise the importance scores
feat_importances = pd.DataFrame({
    "Feature": X_train_rfecv.columns,
    "Importance": importances
}).sort_values(by="Importance", ascending=False)

print(feat_importances.head(10))  # Top 10 features

# Plot top 15 features
plt.figure(figsize=(10,6))
plt.barh(feat_importances["Feature"].head(15)[::-1],
         feat_importances["Importance"].head(15)[::-1],
         color="skyblue")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.title("Top 15 Feature Importances - Random Forest")
plt.show()

##OUTPUT##
                         Feature  Importance
58  incident_severity_Minor Damage    0.097077
16          claim_to_premium_ratio    0.057755
17                   age_x_premium    0.056275
12             claim_component_sum    0.052069
11        policy_premium_per_month    0.046125
14             avg_claim_component    0.043906
2            policy_annual_premium    0.038038
41             insured_hobbies_chess    0.036892
6            incident_hour_of_the_day    0.035328
0                              age    0.032662
```




```
# Select features with high importance scores
top_features = feat_importances[feat_importances["Importance"] >=
0.01]["Feature"].tolist()
```

```
print("Selected top features:\n", top_features)
# Create a new training data with only the selected features
X_train_selected = X_train_rfecv[top_features].copy()
X_test_selected   = X_test_rfecv[top_features].copy()
```

```
##OUTPUT##
Selected top features:
 ['incident_severity_Minor Damage',
'claim_to_premium_ratio', 'age_x_premium',
'claim_component_sum', 'policy_premium_per_month',
'avg_claim_component', 'policy_annual_premium',
'insured_hobbies_chess', 'incident_hour_of_the_day',
'age', 'claim_per_vehicle', 'auto_year', 'vehicle_age',
'incident_severity_Total Loss', 'capital-gains',
'capital-loss', 'insured_hobbies_Other',
'collision_type_unknown', 'policy_csl_500/1000',
'tenure_bucket_10+yr', 'incident_state_WV',
'policy_csl_250/500']
```

6.4.4. Train the model with selected features

```
# Fit the model on the training data with selected features
rf_model.fit(X_train_selected, y_train_res)
```

6.4.5. Generate predictions on the training data

```
# Generate predictions on training data
y_train_rf_pred = rf_model.predict(X_train_selected)
y_train_rf_prob = rf_model.predict_proba(X_train_selected)[:, 1]
```

6.4.6. Check accuracy of the model

```
# Check accuracy of the model
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
print("Training Accuracy:", accuracy_score(y_train_res,
y_train_rf_pred))
print("\nClassification Report:\n", classification_report(y_train_res,
y_train_rf_pred))
```

```
##OUTPUT##
Training Accuracy: 0.9933460076045627

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       526
           1       0.99      1.00      0.99       526

    accuracy                           0.99      1052
   macro avg       0.99      0.99      0.99      1052
weighted avg       0.99      0.99      0.99      1052
```

6.4.7. Create confusion matrix

```
# Create the confusion matrix to visualise the performance
cm = confusion_matrix(y_train_res, y_train_rf_pred)
print("\nConfusion Matrix:\n", cm)

##OUTPUT##
Confusion Matrix:
 [[521   5]
 [  2 524]]
```

6.4.8. Create variables for true positive, true negative, false positive and false negative

```
# Create variables for true positive, true negative, false positive and
false negative
tn, fp, fn, tp = cm.ravel()

print("True Negatives :", tn)
print("False Positives:", fp)
print("False Negatives:", fn)
print("True Positives :", tp)

##OUTPUT##
True Negatives : 521
False Positives: 5
False Negatives: 2
True Positives : 524
```

6.4.9. Calculate sensitivity, specificity, precision, recall and F1-score of the model

```
# Calculate the sensitivity

sensitivity = tp / (tp + fn)
print(f"Sensitivity (Recall): {sensitivity:.4f}")
# Calculate the specificity
specificity = tn / (tn + fp)
print(f"Specificity: {specificity:.4f}")
# Calculate Precision
precision = tp / (tp + fp)
print(f"Precision  : {precision:.4f}")
# Calculate Recall
recall = tp / (tp + fn)
print(f"Recall     : {recall:.4f}")
# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)
print(f"F1 Score   : {f1_score:.4f}")

##OUTPUT##
Sensitivity (Recall): 0.9962
Specificity: 0.9905
Precision  : 0.9905
Recall     : 0.9962
F1 Score   : 0.9934
```

6.4.10. Check if the model is overfitting training data using cross validation

```
# Use cross validation to check if the model is overfitting
from sklearn.model_selection import cross_validate, StratifiedKFold
from sklearn.metrics import make_scorer, accuracy_score,
precision_score, recall_score, f1_score, roc_auc_score
y_arr = np.asarray(y_train_res).reshape(-1,)

# Define scoring metrics
scoring = {
    "accuracy": make_scorer(accuracy_score),
    "precision": make_scorer(precision_score, zero_division=0),
    "recall": make_scorer(recall_score, zero_division=0),
    "f1": make_scorer(f1_score, zero_division=0),
    "roc_auc": "roc_auc"
}

# Stratified K-Fold CV
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Run cross-validation
cv_results = cross_validate(
    rf_model,
    X_train_selected, y_arr,
```

```
    cv=cv,
    scoring=scoring,
    return_train_score=True,
    n_jobs=-1
)

# Build summary DataFrame
summary = pd.DataFrame({
    "Metric": ["Accuracy", "Precision", "Recall", "F1", "ROC-AUC"],
    "Train Mean": [cv_results[f"train_{m}"].mean() for m in
["accuracy","precision","recall","f1","roc_auc"]],
    "CV Mean": [cv_results[f"test_{m}"].mean() for m in
["accuracy","precision","recall","f1","roc_auc"]],
    "Train Std": [cv_results[f"train_{m}"].std() for m in
["accuracy","precision","recall","f1","roc_auc"]],
    "CV Std": [cv_results[f"test_{m}"].std() for m in
["accuracy","precision","recall","f1","roc_auc"]],
})

# Add delta column (train - cv)
summary["Delta"] = summary["Train Mean"] - summary["CV Mean"]

print(summary)

##OUTPUT##
Metric  Train Mean   CV Mean  Train Std    CV Std
Delta
0   Accuracy    0.986217  0.878321   0.003324  0.008904
0.107896
1  Precision    0.982111  0.849797   0.004308  0.018230
0.132315
2     Recall    0.990497  0.920126   0.005415  0.020598
0.070371
3         F1    0.986273  0.883214   0.003320  0.008087
0.103059
4    ROC-AUC    0.998989  0.947585   0.000462  0.013671
0.051404
```

6.5. Hyperparameter Tuning

6.5.1. Use grid search to find the best hyperparameter values

```
# Use grid search to find the best hyperparamter values
param_grid = {
    "n_estimators": [100, 200, 300,400,500],      # number of trees
    "max_depth": [5, 10, 15, 20,25],        # tree depth
    "min_samples_split": [2, 10, 20],      # min samples to split a node
    "min_samples_leaf": [1, 5, 10],        # min samples per leaf
    "max_features": ["sqrt", "log2"],      # features per split
    "class_weight": [None, "balanced"]     # handle imbalance
```

```
}

rf = RandomForestClassifier(random_state=42, n_jobs=-1)
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    scoring="roc_auc",   # optimize for ROC-AUC (better for fraud
detection)
    cv=5,
    n_jobs=-1,
    verbose=2
)
# Fit on training set (using selected features)
grid_search.fit(X_train_selected, y_train_res)



# Best Hyperparameters
print("Best Parameters:", grid_search.best_params_)
print("Best ROC-AUC Score:", grid_search.best_score_)
##OUTPUT##
Best Parameters: {'class_weight': None, 'max_depth': 20,
'max_features': 'sqrt', 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 500}
Best ROC-AUC Score: 0.9865543148076841
```

6.5.2. Build a random forest model based on hyperparameter tuning results

```
# Building random forest model based on results of hyperparameter
tuning
best_params=grid_search.best_params_
rf_tuned = RandomForestClassifier(**best_params)
rf_tuned.fit(X_train_selected, y_train_res)
```

6.5.3. Make predictions on training data

```
# Make predictions on training data
y_train_pred = rf_tuned.predict(X_train_selected)
y_train_prob = rf_tuned.predict_proba(X_train_selected)[:,1]
```

6.5.4. Check accuracy of Random Forest Model

```
# Check the accuracy
print("Training Accuracy:", accuracy_score(y_train_res,
y_train_pred))
print("\nClassification Report:\n", classification_report(y_train_res,
y_train_pred))
##OUTPUT##
Training Accuracy: 1.0

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       526
           1       1.00      1.00      1.00       526

    accuracy                           1.00      1052
   macro avg       1.00      1.00      1.00      1052
weighted avg       1.00      1.00      1.00      1052
```

6.5.5. Create confusion matrix

```
# Create the confusion matrix
cm = confusion_matrix(y_train_res, y_train_pred)
print("\nConfusion Matrix:\n", cm)
##OUTPUT##
Confusion Matrix:
 [[526   0]
 [  0 526]]
```

6.5.6. Create variables for true positive, true negative, false positive and false negative

```
# Create variables for true positive, true negative, false positive and
false negative
tn, fp, fn, tp = cm.ravel()

print("True Negatives :", tn)
print("False Positives:", fp)
print("False Negatives:", fn)
print("True Positives :", tp)
##OUTPUT##
True Negatives : 526
False Positives: 0
False Negatives: 0
True Positives : 526
```

6.5.7. Calculate sensitivity, specificity, precision, recall and F1-score of the model

```
# Calculate the sensitivity

sensitivity = tp / (tp + fn)
print(f"Sensitivity (Recall): {sensitivity:.4f}")
# Calculate the specificity
specificity = tn / (tn + fp)
print(f"Specificity: {specificity:.4f}")
# Calculate Precision
precision = tp / (tp + fp)
print(f"Precision  : {precision:.4f}")
# Calculate Recall
recall = tp / (tp + fn)
print(f"Recall     : {recall:.4f}")
# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)
print(f"F1 Score   : {f1_score:.4f}")
##OUTPUT##
Sensitivity (Recall): 1.0000
Specificity: 1.0000
Precision  : 1.0000
Recall     : 1.0000
F1 Score   : 1.0000
```

7. Prediction and Model Evaluation

7.1. Make predictions over validation data using logistic regression model

7.1.1. Make predictions over validation data

```
# Make predictions on the validation data and store it in the
variable 'y_validation_pred'
y_validation_pred = logsk.predict_proba(X_test_rfecv)[:, 1]
```

7.1.2. Create DataFrame with actual values and predicted values for validation data

	fraud_reported	pred_prob
0	0	0.734447

1	0	0.042975
2	0	0.241139
3	0	0.142840
4	0	0.119074

7.1.3. Make final prediction based on cutoff value

```
# Make final predictions on the validation data using the optimal
cutoff
print("Optimal cutoff:", optimal_cutoff)
test_pred_df["final_pred"] = (test_pred_df["pred_prob"] >=
optimal_cutoff).astype(int)

print(test_pred_df.head())
##OUTPUT##
Optimal cutoff: 0.5
   fraud_reported  pred_prob  final_pred
0               0   0.734447           1
1               0   0.042975           0
2               0   0.241139           0
3               0   0.142840           0
4               0   0.119074           0
```

7.1.4. Check the accuracy of logistic regression model on validation data

```
# Check the accuracy
print("Test Accuracy:", accuracy_score(y_test,
test_pred_df["final_pred"]))
##OUTPUT##
Test Accuracy: 0.74
```

7.1.5. Create confusion matrix

```
# Create the confusion matrix
cm = confusion_matrix(y_test, test_pred_df["final_pred"])
print("\nConfusion Matrix:\n", cm)
##OUTPUT##
Confusion Matrix:
```

```
[[178  48]
 [ 30  44]]
```

7.1.6. Create variables for true positive, true negative, false positive and false negative

```
# Create variables for true positive, true negative, false positive and false negative
tn, fp, fn, tp = cm.ravel()

print("True Negatives :", tn)
print("False Positives:", fp)
print("False Negatives:", fn)
print("True Positives :", tp)
##OUTPUT##
True Negatives : 178
False Positives: 48
False Negatives: 30
True Positives : 44
```

7.1.7. Calculate sensitivity, specificity, precision, recall and f1 score of the model

```
# Calculate the sensitivity

sensitivity = tp / (tp + fn)
print(f"Sensitivity (Recall): {sensitivity:.4f}")
# Calculate the specificity
specificity = tn / (tn + fp)
print(f"Specificity: {specificity:.4f}")
# Calculate Precision
precision = tp / (tp + fp)
print(f"Precision  : {precision:.4f}")
# Calculate Recall
recall = tp / (tp + fn)
print(f"Recall     : {recall:.4f}")
# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)
print(f"F1 Score   : {f1_score:.4f}")
##OUTPUT##
Sensitivity (Recall): 0.5946
Specificity: 0.7876
Precision  : 0.4783
Recall     : 0.5946
F1 Score   : 0.5301
```

7.2. Make predictions over validation data using random forest model

7.2.1. Select the important features and make predictions over validation data

```
# Select the relevant features for validation data
X_test_selected
# Make predictions on the validation data
y_test_pred = rf_tuned.predict(X_test_selected)
y_test_prob = rf_tuned.predict_proba(X_test_selected)[:,1]
```

7.2.2. Check accuracy of random forest model

```
# Check accuracy
print("Test Accuracy:", accuracy_score(y_test, y_test_pred))
##OUTPUT##
Test Accuracy: 0.77
```

7.2.3. Create confusion matrix

```
# Create the confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
print("\nConfusion Matrix:\n", cm)
##OUTPUT##
Confusion Matrix:
 [[198  28]
 [ 41  33]]
```

7.2.4. Create variables for true positive, true negative, false positive and false negative

```
# Create variables for true positive, true negative, false positive and
false negative
tn, fp, fn, tp = cm.ravel()

print("True Negatives :", tn)
print("False Positives:", fp)
print("False Negatives:", fn)
print("True Positives :", tp)
##OUTPUT##
```

```
True Negatives : 198
False Positives: 28
False Negatives: 41
True Positives : 33
```

7.2.5. Calculate sensitivity, specificity, precision, recall and F1-score of the model

```
# Calculate the sensitivity

sensitivity = tp / (tp + fn)
print(f"Sensitivity (Recall): {sensitivity:.4f}")
# Calculate the specificity
specificity = tn / (tn + fp)
print(f"Specificity: {specificity:.4f}")
# Calculate Precision
precision = tp / (tp + fp)
print(f"Precision  : {precision:.4f}")
# Calculate Recall
recall = tp / (tp + fn)
print(f"Recall     : {recall:.4f}")
# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)
print(f"F1 Score   : {f1_score:.4f}")
##OUTPUT##
Sensitivity (Recall): 0.4459
Specificity: 0.8761
Precision  : 0.5410
Recall     : 0.4459
F1 Score   : 0.4889
```

8. Evaluation and Conclusion

Logistic Regression Results

Training Performance Accuracy: 0.8888 (very strong) Confusion Matrix: [[465, 61], [56, 470]] TN = 465, FP = 61, FN = 56, TP = 470 Sensitivity (Recall): 0.8935 - model correctly catches ~89% of fraud cases. Specificity: 0.8840 - also good at correctly identifying legitimate claims. Precision: 0.8851 - 89% of predicted frauds are actual fraud. F1 Score: 0.8893 - balanced performance. Training metrics look very strong and balanced across all measures.

Validation (Test) PerformancE Accuracy: 0.7400 (drops from training → sign of overfitting / weaker generalization) Confusion Matrix: [[178, 48], [30, 44]] TN = 178, FP = 48, FN = 30, TP = 44 Sensitivity (Recall): 0.5946 - only 59% of frauds are detected (misses 41%). Specificity: 0.7876 - does better at identifying legitimate claims. Precision: 0.4783 - less than half of predicted frauds are actually fraud. F1 Score: 0.5301 - moderate balance, but weak compared to training On validation data, the model performance drops significantly, especially in Precision and Recall.

Conclusion for logistic regression Logistic Regression fits the training data very well, showing balanced and strong performance However, on validation data, it generalizes poorly: Accuracy drops to 74%. Recall falls to 59% (model misses many fraud cases). Precision is low (48%), meaning high false positives. This suggests overfitting or that linear decision boundaries are insufficient for capturing fraud patterns.

Random Forest Results

Training Performance (before and after tuning) Baseline RF (n_estimators=10) Accuracy: 0.9933 Confusion Matrix: [[521, 5], [2, 524]] Sensitivity (Recall): 0.9962 Specificity: 0.9905 F1 Score: 0.9934 Already extremely high performance on training set.

After GridSearchCV tuning (best params: depth=20, n_estimators=500, etc.) Training Accuracy: 1.0000 Confusion Matrix: [[526, 0], [0, 526]] Sensitivity, Specificity, Precision, Recall, F1: all = 1.0 Perfect fit on training data → a clear sign of overfitting.

Test (Validation) Performance Accuracy: 0.7400 (same as Logistic Regression test accuracy). Confusion Matrix: [[178, 48], [30, 44]] TN = 178, FP = 48, FN = 30, TP = 44 Sensitivity (Recall): 0.5946 → catches ~59% of fraud (misses 41%). Specificity: 0.7876 → correctly identifies ~79% legitimate claims. Precision: 0.4783 → less than half of flagged frauds are true fraud. F1 Score: 0.5301 → weak balance of precision & recall. Despite perfect training performance, the test metrics collapse to the same level as Logistic Regression.

Conclusion Random Forest massively overfits: It memorizes the training set (100% accuracy). But generalization to test set is poor (only 74% accuracy). Performance on test set (Accuracy = 0.74, Recall = 0.59, Precision = 0.48, F1 = 0.53) is almost identical to Logistic Regression's test performance. This means Random Forest did not improve generalization, even though it overfits more aggressively than Logistic Regression.

```
### Conlusion between Losistic classification and Random forest
```

Logistic Regression vs Random Forest — Model Comparison

Model Performance

Metric	Logistic Regression	Random Forest

Training Accuracy	0.8888	1.0000 (after tuning)
Training Recall	0.8935	1.0000
Training Precision	0.8851	1.0000
Training F1 Score	0.8893	1.0000
Test Accuracy	0.7400	0.7400
Test Recall	0.5946	0.5946
Test Precision	0.4783	0.4783
Test F1 Score	0.5301	0.5301

Conclusion

1. **On training data**
    - Logistic Regression performs **very well** but not perfect.
    - Random Forest (especially after tuning) achieves **perfect fit (100%)**, which is a strong sign of **overfitting**.
2. **On validation/test data**
    - Both models perform **similarly** (Accuracy ≈ 0.74, Recall ≈ 0.59, Precision ≈ 0.48, F1 ≈ 0.53).
    - Neither model generalizes well; Random Forest's extra complexity **did not improve performance** compared to Logistic Regression.
3. **Interpretability vs Complexity**
    - Logistic Regression is **simpler, interpretable, and stable**.
    - Random Forest is more **complex and overfits easily** without improving test results.
4. **Overall**
    - Logistic Regression is a **better baseline** model here: it generalizes almost as well as Random Forest, but with less overfitting risk and easier interpretability.
    - Random Forest needs **stronger regularization/tuning** or may require **additional feature engineering / resampling techniques** to outperform Logistic Regression.