```
import kagglehub
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import glob # Import the glob module

path_to_directory = kagglehub.dataset_download("buntyshah/auto-insurance-claims-data")

print("Path to dataset files:", path_to_directory)

csv_files = glob.glob(f"{path_to_directory}/*.csv")

if len(csv_files) > 0:
    path_to_csv_file = csv_files[0]
    df = pd.read_csv(path_to_csv_file)
    print("Successfully loaded the CSV file.")
else:
    print("No CSV file found in the downloaded directory.")
```

```
Path to dataset files: /kaggle/input/auto-insurance-claims-data
Successfully loaded the CSV file.
```

```
df
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premiu
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.9
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.2
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.1
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.7
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.9
...	...	...	...	...	...	...	...	...
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.8
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.7
997	130	34	918516	2003-02-17	OH	250/500	500	1383.4
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.9
999	456	60	556080	1996-11-11	OH	250/500	1000	766.1

1000 rows × 40 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 40 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   months_as_customer           1000 non-null   int64
 1   age                          1000 non-null   int64
 2   policy_number                1000 non-null   int64
 3   policy_bind_date             1000 non-null   object
 4   policy_state                 1000 non-null   object
 5   policy_csl                   1000 non-null   object
 6   policy_deductable            1000 non-null   int64
 7   policy_annual_premium        1000 non-null   float64
 8   umbrella_limit               1000 non-null   int64
 9   insured_zip                  1000 non-null   int64
 10  insured_sex                  1000 non-null   object
 11  insured_education_level      1000 non-null   object
 12  insured_occupation           1000 non-null   object
 13  insured_hobbies              1000 non-null   object
 14  insured_relationship         1000 non-null   object
 15  capital-gains                1000 non-null   int64
 16  capital-loss                 1000 non-null   int64
 17  incident_date                1000 non-null   object
 18  incident_type                1000 non-null   object
 19  collision_type               1000 non-null   object
 20  incident_severity            1000 non-null   object
```

```
 21  authorities_contacted        909 non-null    object
 22  incident_state               1000 non-null   object
 23  incident_city                1000 non-null   object
 24  incident_location            1000 non-null   object
 25  incident_hour_of_the_day     1000 non-null   int64
 26  number_of_vehicles_involved  1000 non-null   int64
 27  property_damage              1000 non-null   object
 28  bodily_injuries              1000 non-null   int64
 29  witnesses                    1000 non-null   int64
 30  police_report_available      1000 non-null   object
 31  total_claim_amount           1000 non-null   int64
 32  injury_claim                 1000 non-null   int64
 33  property_claim               1000 non-null   int64
 34  vehicle_claim                1000 non-null   int64
 35  auto_make                    1000 non-null   object
 36  auto_model                   1000 non-null   object
 37  auto_year                    1000 non-null   int64
 38  fraud_reported               1000 non-null   object
 39  _c39                         0 non-null      float64
dtypes: float64(2), int64(17), object(21)
memory usage: 312.6+ KB
```

```
df.nunique()
```

	0
months_as_customer	391
age	46
policy_number	1000
policy_bind_date	951
policy_state	3
policy_csl	3
policy_deductable	3
policy_annual_premium	991
umbrella_limit	11
insured_zip	995
insured_sex	2
insured_education_level	7
insured_occupation	14
insured_hobbies	20
insured_relationship	6
capital-gains	338
capital-loss	354
incident_date	60
incident_type	4
collision_type	4
incident_severity	4
authorities_contacted	4
incident_state	7
incident_city	7
incident_location	1000
incident_hour_of_the_day	24
number_of_vehicles_involved	4
property_damage	3
bodily_injuries	3
witnesses	4
police_report_available	3
total_claim_amount	763
injury_claim	638
property_claim	626
vehicle_claim	726
auto_make	14
auto_model	39
auto_year	21
fraud_reported	2
_c39	0

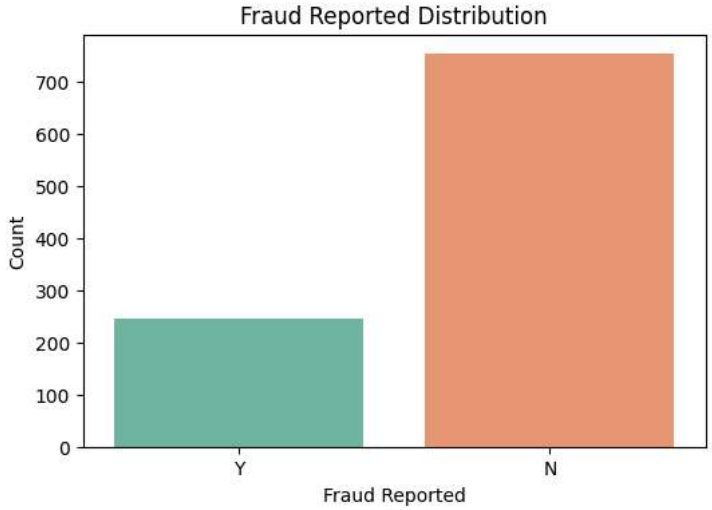
dtype: int64

EDA

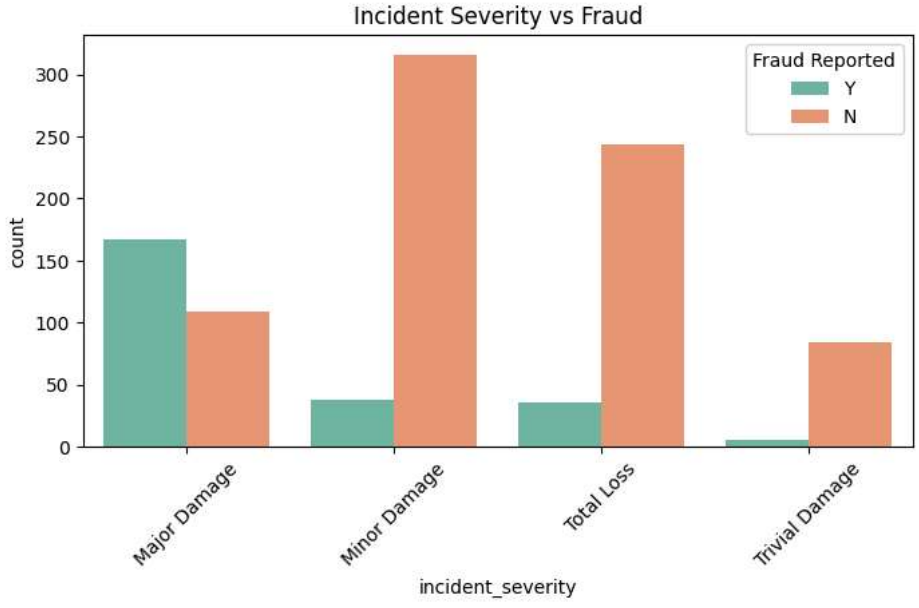
```
# Class distribution
plt.figure(figsize=(6, 4))
```

```
plt.figure(figsize=(8, 4))
sns.countplot(x='fraud_reported', data=df, palette='Set2')
plt.title('Fraud Reported Distribution')
plt.xlabel('Fraud Reported')
plt.ylabel('Count')
plt.show()
```

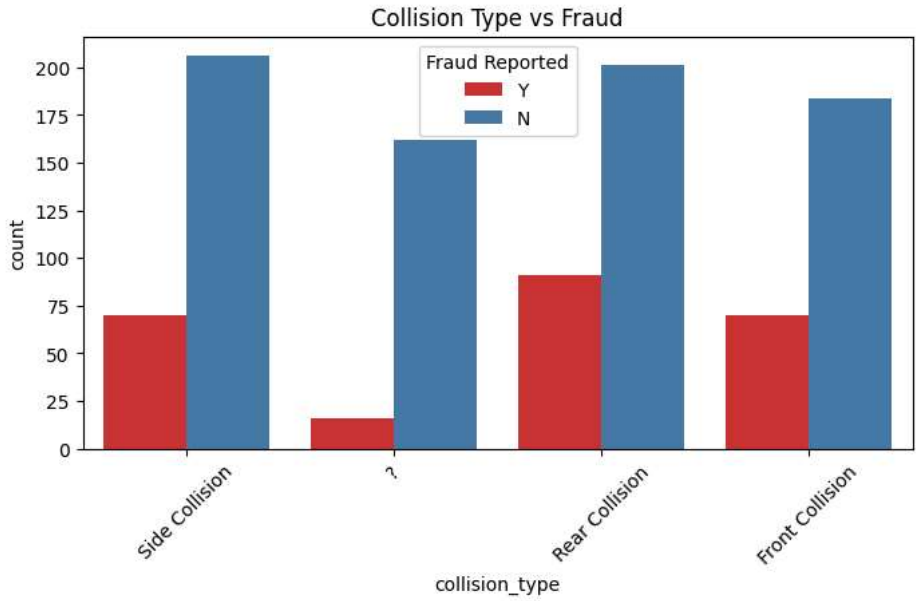
```
<ipython-input-74-11b09e768cdc>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set

  sns.countplot(x='fraud_reported', data=df, palette='Set2')
```




```
plt.figure(figsize=(8, 4))
sns.countplot(data=df, x='incident_severity', hue='fraud_reported', palette='Set2')
plt.title('Incident Severity vs Fraud')
plt.xticks(rotation=45)
plt.legend(title='Fraud Reported')
plt.show()
```




```
plt.figure(figsize=(8, 4))
sns.countplot(data=df, x='collision_type', hue='fraud_reported', palette='Set1')
plt.title('Collision Type vs Fraud')
plt.xticks(rotation=45)
plt.legend(title='Fraud Reported')
plt.show()
```

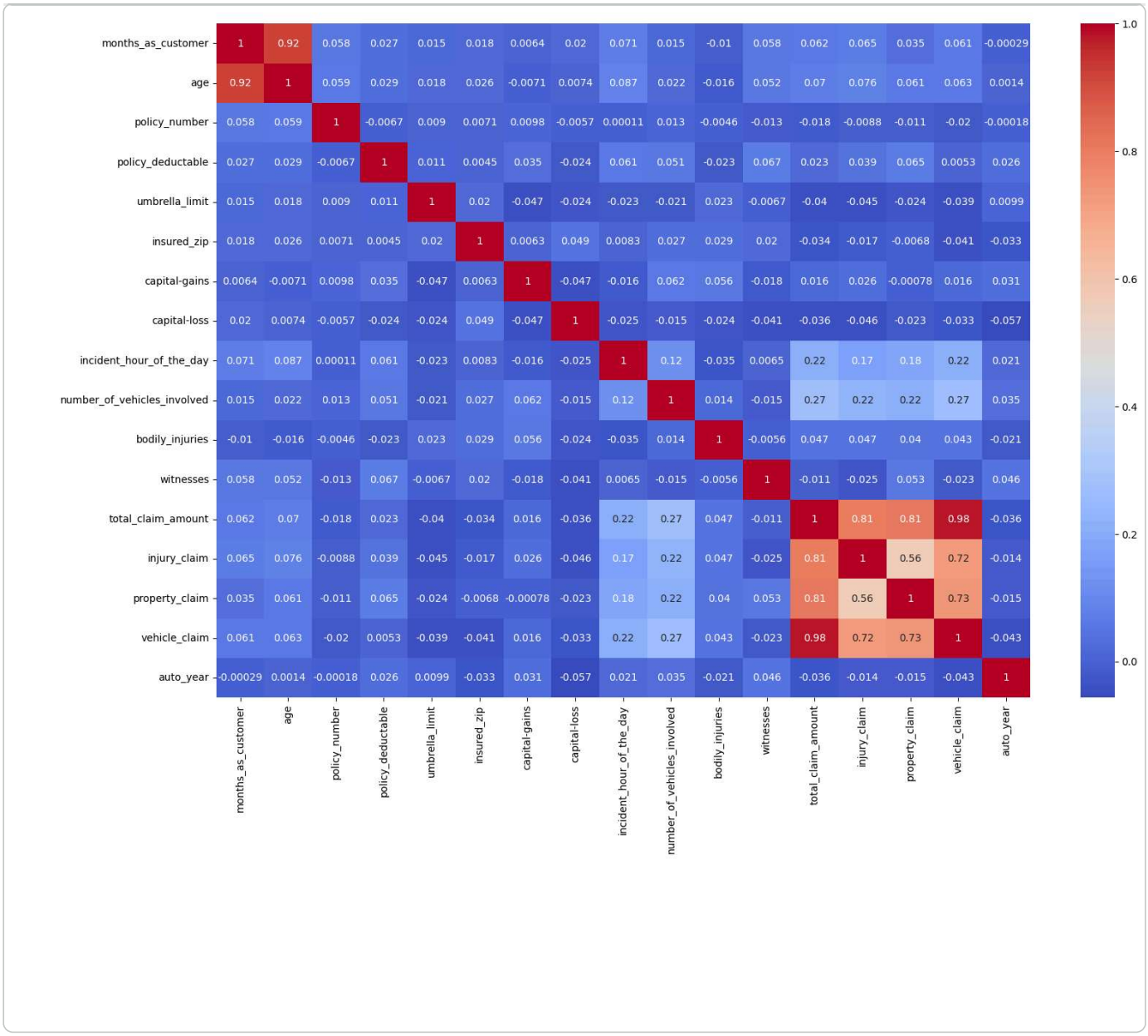



```
df['collision_type'].value_counts()
```

collision_type	count
Rear Collision	292
Side Collision	276
Front Collision	254
?	178

dtype: int64

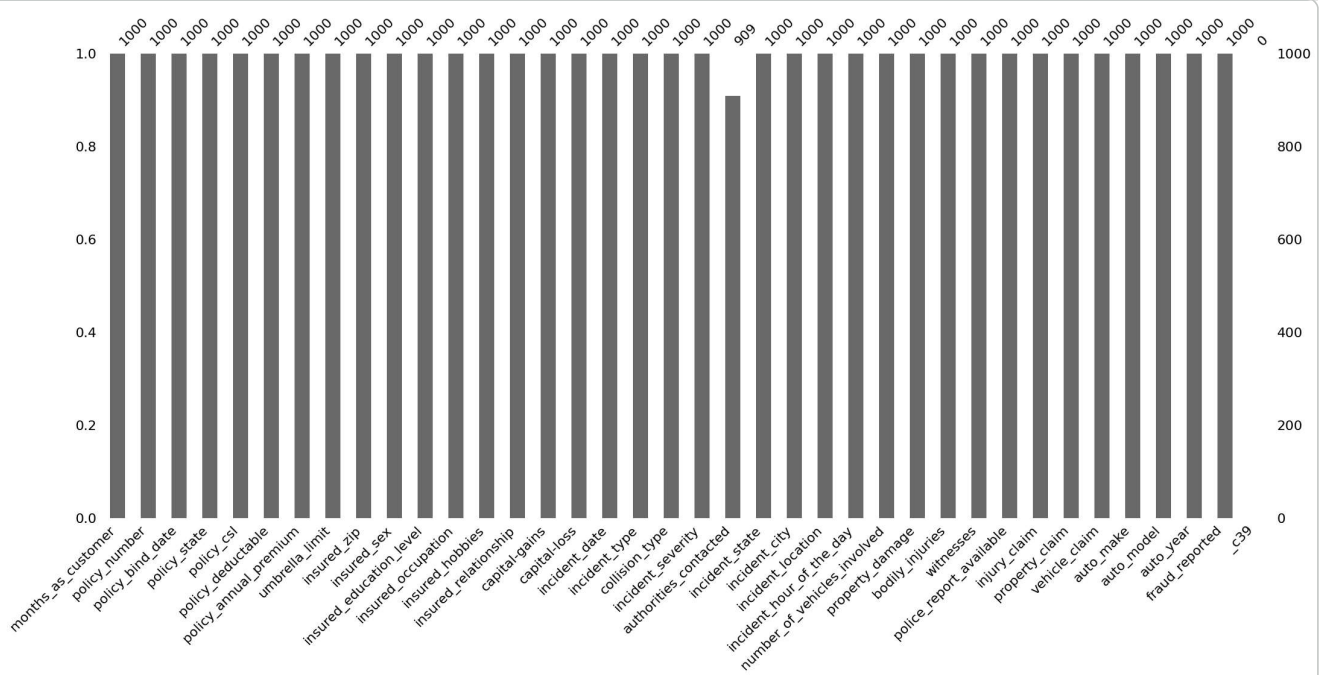
```
corr = df.select_dtypes('int').corr()

plt.figure(figsize = (18, 12))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```




```
df.drop(columns = ['age', 'total_claim_amount'], inplace = True, axis = 1)
```

Preprocessing

```
import missingno as msno

msno.bar(df)
plt.show()
```




```
df['authorities_contacted'].unique()
```

```
array(['Police', nan, 'Fire', 'Other', 'Ambulance'], dtype=object)
```

```
df['authorities_contacted'].fillna('Other', inplace=True)
```

```
<ipython-input-82-d8b2a0360c42>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  df['authorities_contacted'].fillna('Other', inplace=True)
```

```
df['authorities_contacted'].isna().sum()
```

```
np.int64(0)
```

Feature engineering and selection

```
df['policy_bind_date'] = pd.to_datetime(df['policy_bind_date'], errors='coerce')
df['incident_date'] = pd.to_datetime(df['incident_date'], errors='coerce')
df['days_since_bind'] = (df['incident_date'] - df['policy_bind_date']).dt.days
```

```
cols_to_drop = ['policy_number','policy_bind_date',
                'policy_state','insured_zip',
                'incident_location','incident_date',
                'incident_state','incident_city',
                'insured_hobbies','auto_make',
                'auto_model','auto_year', '_c39']

df.drop(cols_to_drop, inplace = True, axis = 1)
```

Encoding categorical columns

```
df['fraud_reported'] = df['fraud_reported'].map({'Y': 1, 'N': 0})

X = df.drop('fraud_reported', axis = 1)
y = df['fraud_reported']
```

```
cat_df = X.select_dtypes('object')
```

```
for col in cat_df.columns:
    print(f"{col}: \n{cat_df[col].unique()}\n")
```

```
policy_csl:
['250/500' '100/300' '500/1000']

insured_sex:
['MALE' 'FEMALE']

insured_education_level:
['MD' 'PhD' 'Associate' 'Masters' 'High School' 'College' 'JD']

insured_occupation:
['craft-repair' 'machine-op-inspct' 'sales' 'armed-forces' 'tech-support'
 'prof-specialty' 'other-service' 'priv-house-serv' 'exec-managerial'
 'protective-serv' 'transport-moving' 'handlers-cleaners' 'adm-clerical'
 'farming-fishing']

insured_relationship:
['husband' 'other-relative' 'own-child' 'unmarried' 'wife' 'not-in-family']

incident_type:
['Single Vehicle Collision' 'Vehicle Theft' 'Multi-vehicle Collision'
 'Parked Car']

collision_type:
['Side Collision' '?' 'Rear Collision' 'Front Collision']

incident_severity:
['Major Damage' 'Minor Damage' 'Total Loss' 'Trivial Damage']

authorities_contacted:
['Police' 'Other' 'Fire' 'Ambulance']

property_damage:
['YES' '?' 'NO']

police_report_available:
['YES' '?' 'NO']
```

```
cat_df['collision_type'] = cat_df['collision_type'].replace('?', "Other")

cols_with_question = [
    'property_damage',
    'police_report_available'
]

# Replace '?' with 'Not Specified'
for col in cols_with_question:
    cat_df[col] = cat_df[col].replace('?', 'Not specified')
```

```
cat_df = pd.get_dummies(cat_df)
```

```
num_df = X.select_dtypes(include = ['number'])
X = pd.concat([num_df, cat_df], axis = 1)
```

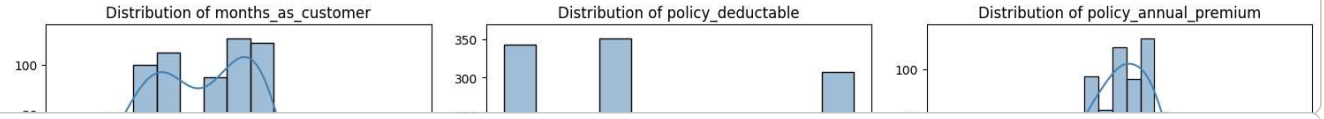
```
X.columns
```

```
Index(['months_as_customer', 'policy_deductable', 'policy_annual_premium',
       'umbrella_limit', 'capital-gains', 'capital-loss',
       'incident_hour_of_the_day', 'number_of_vehicles_involved',
       'bodily_injuries', 'witnesses', 'injury_claim', 'property_claim',
       'vehicle_claim', 'days_since_bind', 'policy_csl_100/300',
       'policy_csl_250/500', 'policy_csl_500/1000', 'insured_sex_FEMALE',
       'insured_sex_MALE', 'insured_education_level_Associate',
       '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_adm-clerical',
       'insured_occupation_armed-forces', 'insured_occupation_craft-repair',
       'insured_occupation_exec-managerial',
       'insured_occupation_farming-fishing',
       'insured_occupation_handlers-cleaners',
       'insured_occupation_machine-op-inspct',
```

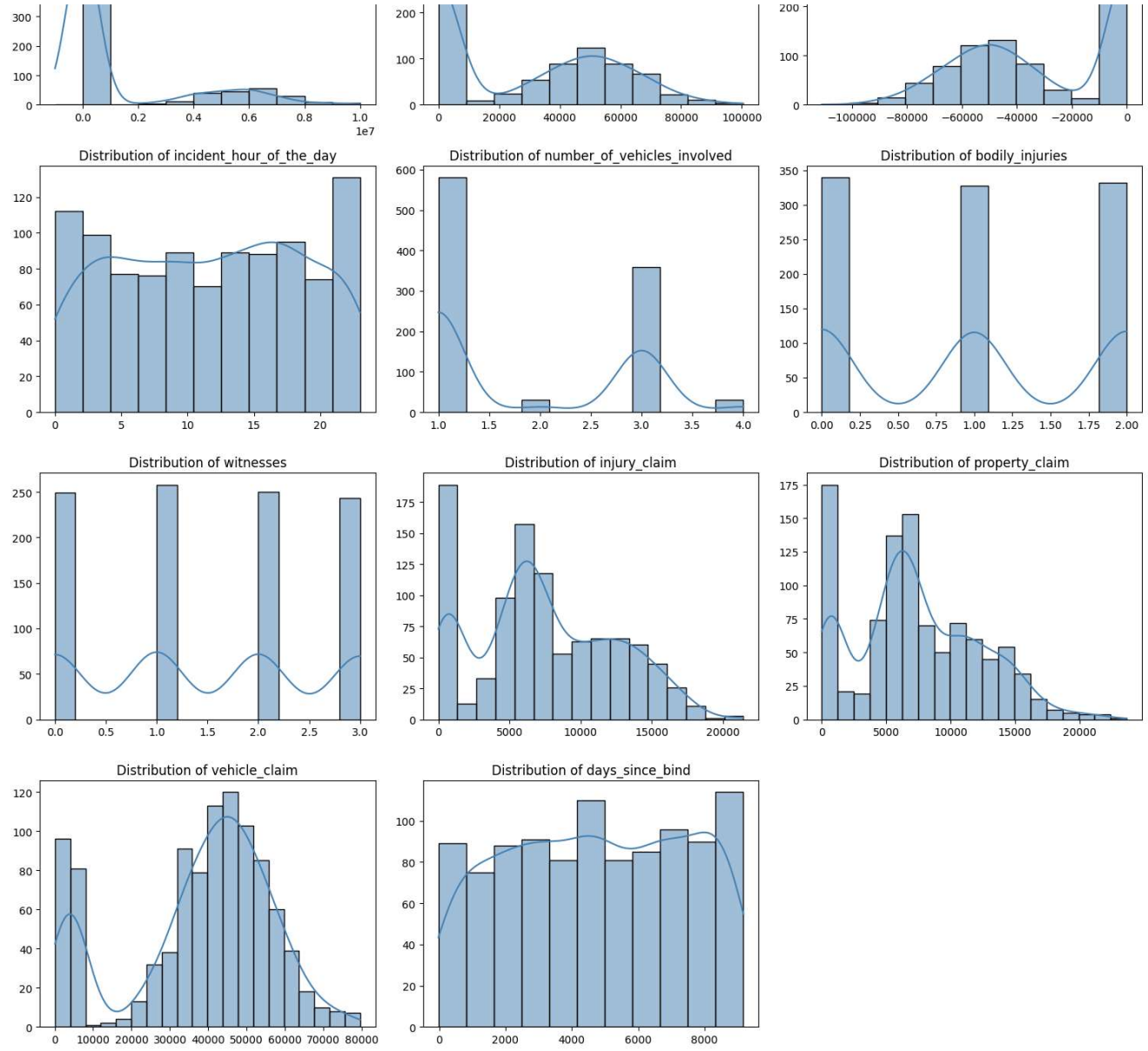
```
       'insured_occupation_other-service',
       'insured_occupation_priv-house-serv',
       'insured_occupation_prof-specialty',
       'insured_occupation_protective-serv', 'insured_occupation_sales',
       'insured_occupation_tech-support',
       'insured_occupation_transport-moving', 'insured_relationship_husband',
       'insured_relationship_not-in-family',
       'insured_relationship_other-relative', 'insured_relationship_own-child',
       'insured_relationship_unmarried', 'insured_relationship_wife',
       'incident_type_Multi-vehicle Collision', 'incident_type_Parked Car',
       'incident_type_Single Vehicle Collision', 'incident_type_Vehicle Theft',
       'collision_type_Front Collision', 'collision_type_Other',
       'collision_type_Rear Collision', 'collision_type_Side Collision',
       'incident_severity_Major Damage', 'incident_severity_Minor Damage',
       'incident_severity_Total Loss', 'incident_severity_Trivial Damage',
       'authorities_contacted_Ambulance', 'authorities_contacted_Fire',
       'authorities_contacted_Other', 'authorities_contacted_Police',
       'property_damage_NO', 'property_damage_Not specified',
       'property_damage_YES', 'police_report_available_NO',
       'police_report_available_Not specified', 'police_report_available_YES'],
      dtype='object')
```

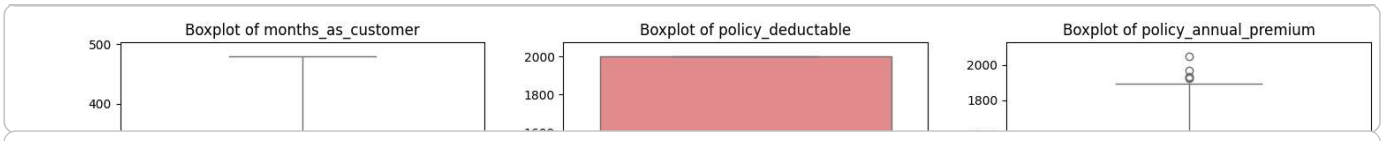
Normality check, Outlier detection & Scaling

```
import math

numeric_cols = X.select_dtypes(include=['int64', 'float64']).columns

num_cols = len(numeric_cols)
cols_per_row = 3
num_rows = math.ceil(num_cols / cols_per_row)

fig, axes = plt.subplots(num_rows, cols_per_row, figsize=(cols_per_row * 5, num_rows * 4))
axes = axes.flatten()

for i, col in enumerate(numeric_cols):
    sns.histplot(X[col], kde=True, ax=axes[i], color='steelblue')
    axes[i].set_title(f'Distribution of {col}')
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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




```
fig, axes = plt.subplots(num_rows, cols_per_row, figsize=(cols_per_row * 5, num_rows * 4))
axes = axes.flatten()

# Plot each variable
for i, col in enumerate(numeric_cols):
    sns.boxplot(y=X[col], ax=axes[i], color='lightcoral')
    axes[i].set_title(f'Boxplot of {col}')
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

# Hide unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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





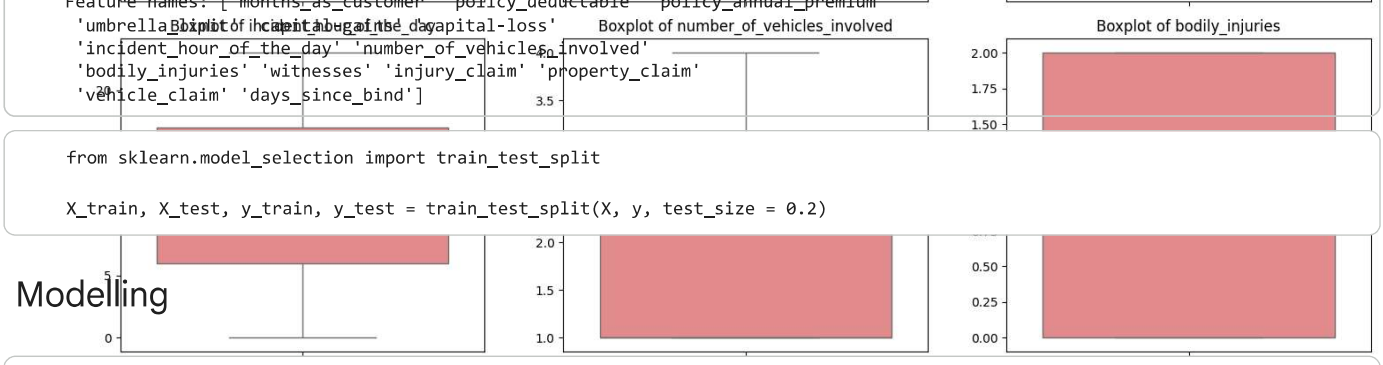

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler = scaler.fit(num_df)
scaled_data = scaler.transform(num_df)

scaled_num_df = pd.DataFrame(data = scaled_data, columns = num_df.columns, index = num_df.index)
X = pd.concat([scaled_num_df, cat_df], axis = 1)
```

```
from google.colab import drive
drive.mount('/content/drive')

import joblib
with open("/content/drive/MyDrive/Auto Insurance Fraud Detection/scaler.joblib", "wb") as f:
    joblib.dump(scaler, f)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
print("Feature names:", scaler.feature_names_in_)
```

```
Feature names: ['months_as_customer' 'policy_deductable' 'policy_annual_premium'
 'umbrella_limit' 'capital-gains' 'capital-loss'
 'incident_hour_of_the_day' 'number_of_vehicles_involved'
 'bodily_injuries' 'witnesses' 'injury_claim' 'property_claim'
 'vehicle_claim' 'days_since_bind']
```




```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

Modelling

```
!pip install catboost
```

```
Requirement already satisfied: catboost in /usr/local/lib/python3.11/dist-packages (1.2.8)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from catboost) (1.15.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (4.58.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly->catboost) (9.1.2)
```


```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.metrics import (
    classification_report, confusion_matrix, roc_auc_score,
    precision_recall_curve, roc_curve, auc, f1_score,
    precision_score, recall_score)
```

```
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
```

```
    average_precision_score, confusion_matrix
)

def train_and_evaluate_model(model, model_name, X_train, y_train, X_test, y_test):
    print(f"\n🔍 Evaluating: {model_name}")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    # Metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_proba)
    pr_auc = average_precision_score(y_test, y_proba)
    cm = confusion_matrix(y_test, y_pred)

    # Print results
    print(f"{'Metric':<20} {'Value':>10}")
    print("-" * 32)
    print(f"{'Accuracy':<20} {accuracy:>10.4f}")
    print(f"{'Precision':<20} {precision:>10.4f}")
    print(f"{'Recall':<20} {recall:>10.4f}")
    print(f"{'F1 Score':<20} {f1:>10.4f}")
    print(f"{'ROC AUC':<20} {roc_auc:>10.4f}")
    print(f"{'PR AUC':<20} {pr_auc:>10.4f}")
    print("\nConfusion Matrix:")
    print(cm)
```

```
# Logistic Regression
lr = LogisticRegression(class_weight='balanced', random_state=42)
train_and_evaluate_model(lr, "Logistic Regression", X_train, y_train, X_test, y_test)
```

```
🔍 Evaluating: Logistic Regression
Metric                    Value
--------------------------------
Accuracy                 0.8200
Precision                0.5424
Recall                   0.7805
F1 Score                 0.6400
ROC AUC                  0.8139
PR AUC                   0.4995

Confusion Matrix:
[[132  27]
 [  9  32]]
```

```
# Decision Tree
dt = DecisionTreeClassifier(class_weight='balanced', random_state=42)
train_and_evaluate_model(dt, "Decision Tree", X_train, y_train, X_test, y_test)
```

```
🔍 Evaluating: Decision Tree
Metric                    Value
--------------------------------
Accuracy                 0.7750
Precision                0.4545
Recall                   0.4878
F1 Score                 0.4706
ROC AUC                  0.6684
PR AUC                   0.3267

Confusion Matrix:
[[135  24]
 [ 21  20]]
```

```
# Random Forest
rf = RandomForestClassifier(class_weight='balanced', random_state=42)
train_and_evaluate_model(rf, "Random Forest", X_train, y_train, X_test, y_test)
```

```
🔍 Evaluating: Random Forest
Metric                    Value
--------------------------------
Accuracy                 0.8100
Precision                0.5714
```

```
Recall                  0.2927
F1 Score                0.3871
ROC AUC                 0.8608
PR AUC                  0.5732

Confusion Matrix:
[[150   9]
 [ 29  12]]
```

```
# XGBoost
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
train_and_evaluate_model(xgb, "XGBoost", X_train, y_train, X_test, y_test)
```

```
🔍 Evaluating: XGBoost
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [07:33:31] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Metric                   Value
--------------------------------
Accuracy                0.8000
Precision               0.5172
Recall                  0.3659
F1 Score                0.4286
ROC AUC                 0.8170
PR AUC                  0.5009

Confusion Matrix:
[[145  14]
 [ 26  15]]
```

```
# LightGBM
lgb = LGBMClassifier(class_weight='balanced', random_state=42)
train_and_evaluate_model(lgb, "LightGBM", X_train, y_train, X_test, y_test)
```

```
🔍 Evaluating: LightGBM
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 206, number of negative: 594
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000084 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1951
[LightGBM] [Info] Number of data points in the train set: 800, number of used features: 68
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=-0.000000
[LightGBM] [Info] Start training from score -0.000000
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Metric                   Value
--------------------------------
Accuracy                0.8400
Precision               0.6452
Recall                  0.4878
F1 Score                0.5556
ROC AUC                 0.8233
PR AUC                  0.4920

Confusion Matrix:
[[148  11]
 [ 21  20]]
```

```
# CatBoost
cat = CatBoostClassifier(verbose=0, random_state=42)
train_and_evaluate_model(cat, "CatBoost", X_train, y_train, X_test, y_test)
```

```
🔍 Evaluating: CatBoost
Metric                    Value
--------------------------------
Accuracy                  0.8200
Precision                 0.5862
Recall                    0.4146
F1 Score                  0.4857
ROC AUC                   0.8420
PR AUC                    0.5142

Confusion Matrix:
[[147  12]
 [ 24  17]]
```

```
Start coding or generate with AI.
```

Hyperparameter tuning

```
from sklearn.model_selection import GridSearchCV

param_grid_rf = {
    'n_estimators': [150, 250],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'criterion': ['gini', 'entropy'],
    'bootstrap': [True, False]
}

grid_rf = GridSearchCV(
    RandomForestClassifier(class_weight='balanced', random_state=42),
    param_grid_rf,
    scoring='f1',
    cv=5,
    n_jobs=-1
)

grid_rf.fit(X_train, y_train)
best_rf = grid_rf.best_estimator_
best_rf
```

```
RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=10, min_samples_leaf=2, n_estimators=250,
                       random_state=42)
```

```
train_and_evaluate_model(best_rf, "Random Forest", X_train, y_train, X_test, y_test)
```

```
🔍 Evaluating: Random Forest
Metric                    Value
--------------------------------
Accuracy                  0.8700
Precision                 0.6596
Recall                    0.7561
F1 Score                  0.7045
ROC AUC                   0.8397
PR AUC                    0.5379

Confusion Matrix:
[[143  16]
 [ 10  31]]
```

```
param_grid_lr = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],          # Regularization strength
    'penalty': ['l1', 'l2'],                      # Type of regularization
    'solver': ['liblinear', 'saga'],              # Solvers that support L1 and L2
    'fit_intercept': [True, False],               # Whether to fit the intercept
    'class_weight': [None, 'balanced'],           # Handle imbalance
    'max_iter': [100, 200, 500],                  # Maximum number of iterations
    'tol': [1e-4, 1e-3, 1e-2],                    # Tolerance for stopping criteria
    'dual': [False]                               # Only applicable for 'liblinear'
}
```

```
grid_lr = GridSearchCV(
    LogisticRegression(random_state=42),
    param_grid_lr,
    scoring='f1',
    cv=5,
    n_jobs=-1
)

grid_lr.fit(X_train, y_train)
best_lr = grid_lr.best_estimator_
best_lr
```

```
LogisticRegression(C=0.1, class_weight='balanced', fit_intercept=False,
                   penalty='l1', random_state=42, solver='saga', tol=0.01)
```

```
train_and_evaluate_model(best_lr, "Logistic Regression", X_train, y_train, X_test, y_test)
```

```
🔍 Evaluating: Logistic Regression
Metric                    Value
--------------------------------
Accuracy                 0.8700
Precision                0.6531
Recall                   0.7805
F1 Score                 0.7111
ROC AUC                  0.8308
PR AUC                   0.5275

Confusion Matrix:
[[142  17]
 [  9  32]]
```

```
param_grid_dt = {
    'criterion': ['gini', 'entropy'],          # Loss function to measure the quality of a split
    'splitter': ['best', 'random'],                        # Strategy used to choose the split at each node
    'max_features': [None, 'sqrt', 'log2'],                # Number of features to consider when looking for best split
    'class_weight': [None, 'balanced'],
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],# Handling class imbalance
}


grid_dt = GridSearchCV(
    DecisionTreeClassifier(random_state=42),
    param_grid_dt,
    scoring='f1',
    cv=5,
    n_jobs=-1
)

grid_dt.fit(X_train, y_train)
best_dt = grid_dt.best_estimator_
best_dt
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=3, min_samples_leaf=4, random_state=42,
                       splitter='random')
```

```
train_and_evaluate_model(best_dt, "Decision Trees", X_train, y_train, X_test, y_test)
```

```
🔍 Evaluating: Decision Trees
Metric                    Value
--------------------------------
Accuracy                 0.8750
Precision                0.6667
Recall                   0.7805
F1 Score                 0.7191
ROC AUC                  0.8205
PR AUC                   0.5203
```

```
Confusion Matrix:
[[143  16]
 [  9  32]]
```

```
import pickle

with open('model_dt.pkl', 'wb') as f:
    pickle.dump(best_dt, f)

# Load the model
with open('model_dt.pkl', 'rb') as f:
    model_loaded = pickle.load(f)
```

```
# Save to a specific folder in Drive
with open('/content/drive/MyDrive/Auto Insurance Fraud Detection/model_dt.pkl', 'wb') as f:
    pickle.dump(best_dt, f)
```

```
# Save to a specific folder in Drive

with open('/content/drive/MyDrive/Auto Insurance Fraud Detection/model_dt.joblib', 'wb') as f:
    joblib.dump(best_dt, f)
```

```
param_grid_xgb = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'scale_pos_weight': [1, 5, 10]  # For imbalance
}

grid_xgb = GridSearchCV(
    XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42),
    param_grid_xgb,
    scoring='f1',
    cv=5,
    n_jobs=-1
)

grid_xgb.fit(X_train, y_train)
best_xgb = grid_xgb.best_estimator_
best_xgb
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [07:40:49] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.1, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=3,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=100,
              n_jobs=None, num_parallel_tree=None, random_state=42, ...)
```

```
train_and_evaluate_model(best_xgb, "XG Boost", X_train, y_train, X_test, y_test)
```

```
🔍 Evaluating: XG Boost
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [07:40:50] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Metric                    Value
--------------------------------
Accuracy                 0.7700
Precision                0.4648
Recall                   0.8049
F1 Score                 0.5893
ROC AUC                  0.8239
PR AUC                   0.5027
```