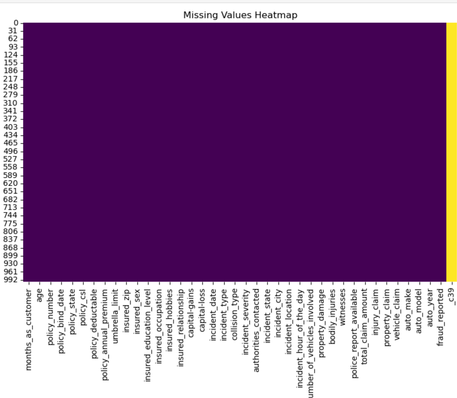
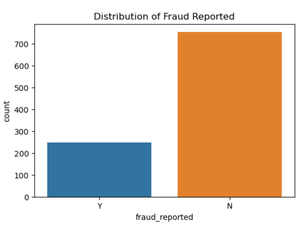
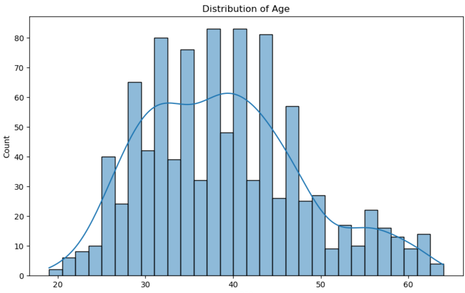
In [3]:
```ipython3
# Step 1: Import the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import cross_val_score, GridSearchCV
import joblib
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

# Step 2: Load the dataset
url = "https://raw.githubusercontent.com/dsrscientist/Data-Science-ML-Capstone-Projects/master/Automobile_insurance_fraud.csv"
df = pd.read_csv(url)

# Step 3: Exploratory Data Analysis (EDA)
# Check for missing values
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()

# Distribution of target variable
plt.figure(figsize=(6, 4))
sns.countplot(x='fraud_reported', data=df)
plt.title('Distribution of Fraud Reported')
plt.show()

# Distribution of age
plt.figure(figsize=(10, 6))
sns.histplot(x='age', data=df, kde=True, bins=30)
plt.title('Distribution of Age')
plt.show()

# Step 4: Preprocessing and Feature Engineering
df = df.drop('_c39', axis=1)

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object']).columns

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Step 5: Building a RandomForestClassifier model
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

model.fit(X_train, y_train)

# Step 6: Model Evaluation
train_accuracy = model.score(X_train, y_train)
test_accuracy = model.score(X_test, y_test)

print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Testing Accuracy: {test_accuracy:.2f}")

cv_accuracy = cross_val_score(model, X, y, cv=5, scoring='accuracy').mean()
print(f"Cross-Validation Accuracy: {cv_accuracy:.2f}")

y_pred = model.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Step 7: Hyperparameter Tuning
param_grid = {
    'classifier__n_estimators': [50, 100, 200],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [2, 5, 10],
    'classifier__min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(model, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print("Best Hyperparameters:")
print(best_params)

# Step 8: Save the best model for production
best_model = grid_search.best_estimator_
joblib.dump(best_model, 'insurance_fraud_detection_model.pkl')
```








```
Training Accuracy: 1.00
Testing Accuracy: 0.72
Cross-Validation Accuracy: 0.76
Classification Report:
              precision    recall  f1-score   support

           N       0.74      0.96      0.83       145
           Y       0.50      0.11      0.18        55

    accuracy                           0.73       200
   macro avg       0.62      0.53      0.51       200
weighted avg       0.67      0.72      0.65       200

Best Hyperparameters:
{'classifier__max_depth': None, 'classifier__min_samples_leaf': 1, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 200}
['insurance_fraud_detection_model.pkl']
```

Out[3]:

— Title: Census Income Prediction Model Documentation

1. Introduction:
   - This document provides documentation for the Census Income Prediction Model.
   - The model aims to predict income levels based on various features using a RandomForestClassifier.
2. Libraries Used:
   - pandas: Data manipulation and analysis.
   - numpy: Numerical operations.
   - matplotlib, seaborn: Data visualization.
   - scikit-learn: Machine learning tools.
   - joblib: Saving the best-tuned model.
3. Dataset:
   - The model uses the Census Income dataset from the given link.
4. Exploratory Data Analysis (EDA): a. Display Basic Statistics:
   - Provides an overview of the dataset's numerical features. b. Visualize Target Variable Distribution:
   - Displays the distribution of the 'Income' target variable. c. Visualize Correlation Matrix:
   - Heatmap to visualize the correlation between features.
5. Preprocessing and Feature Engineering: a. Target and Features:
   - 'Income' is the target column, and other columns are features. b. Handling Categorical Variables:
   - Label Encoding is applied to transform categorical variables into numerical format.
6. Data Splitting:
   - The dataset is split into training and testing sets for model evaluation.
7. Model Building and Testing: a. RandomForestClassifier:
   - Model is instantiated and trained on the training data.
   - Evaluation metrics (Accuracy, Classification Report) are printed for the test set.
8. Hyperparameter Tuning:
   - GridSearchCV is used to find the best hyperparameters for the RandomForestClassifier.
   - The best-tuned model is saved using joblib.
9. Conclusion:
   - The Census Income Prediction Model has been developed, tested, and fine-tuned for improved performance.

—

In [ ]: