Milestone 2: Data Collection & Preparation

Machine learning relies heavily on the **quality and relevance of data**. A well-prepared dataset is the foundation for training reliable and accurate models. In this section, we focus on acquiring and beginning to explore the dataset needed for fraud detection in auto insurance claims.

Activity 1: Collect the Dataset

For this project, we are using a **.CSV dataset** available from a trusted open-source platform. Open datasets help standardize experimentation and improve reproducibility.

- **Source**: Kaggle - Auto Insurance Claims Data
- **Dataset Format**: CSV (Comma-Separated Values)
- **Purpose**: To train a machine learning model that detects whether an insurance claim is fraudulent or not.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import SMOTE
import joblib
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

●For checking the null values, df.isna().any( ) function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
df = pd.read_csv("insurance_claims.csv")
```

```
df.head()
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	...	police_report_availabl
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	0	466132	...	YE
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	5000000	468176	...	
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.14	5000000	430632	...	N
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.74	6000000	608117	...	N
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.91	6000000	610706	...	N

5 rows × 40 columns

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Handling missing values

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

	0
months_as_customer	0
age	0
policy_number	0
policy_bind_date	0
policy_state	0
policy_csl	0
policy_deductable	0
policy_annual_premium	0
umbrella_limit	0
insured_zip	0
insured_sex	0
insured_education_level	0
insured_occupation	0
insured_hobbies	0
insured_relationship	0
capital-gains	0
capital-loss	0
incident_date	0
incident_type	0
collision_type	0
incident_severity	0
authorities_contacted	91
incident_state	0
incident_city	0

```
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

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

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

```
df['authorities_contacted'] = df['authorities_contacted'].fillna('Unknown')
df.isna().any()
```

	0
months_as_customer	False
age	False
policy_number	False
policy_bind_date	False
policy_state	False
policy_csl	False
policy_deductable	False
policy_annual_premium	False

```
df.drop(columns=['_c39'], inplace=True)
df.drop(columns=['incident_location'], inplace=True)
df.drop(columns=['policy_number'], inplace=True)
df.drop(columns=['insured_zip'], inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 36 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   months_as_customer       1000 non-null   int64
 1   age                      1000 non-null   int64
 2   policy_bind_date         1000 non-null   object
 3   policy_state             1000 non-null   object
 4   policy_csl               1000 non-null   object
 5   policy_deductable        1000 non-null   int64
 6   policy_annual_premium    1000 non-null   float64
 7   umbrella_limit           1000 non-null   int64
 8   insured_sex              1000 non-null   object
 9   insured_education_level  1000 non-null   object
 10  insured_occupation       1000 non-null   object
 11  insured_hobbies          1000 non-null   object
 12  insured_relationship     1000 non-null   object
 13  capital-gains            1000 non-null   int64
 14  capital-loss             1000 non-null   int64
 15  incident_date            1000 non-null   object
 16  incident_type            1000 non-null   object
```

```python
non_numeric_cols = df.select_dtypes(exclude=['number']).columns
print(non_numeric_cols)
```

```
Index(['policy_bind_date', 'policy_state', 'policy_csl', 'insured_sex',
       'insured_education_level', 'insured_occupation', 'insured_hobbies',
       'insured_relationship', 'incident_date', 'incident_type',
       'collision_type', 'incident_severity', 'authorities_contacted',
       'incident_state', 'incident_city', 'property_damage',
       'police_report_available', 'auto_make', 'auto_model', 'fraud_reported'],
      dtype='object')
```

```python
df[(df['total_claim_amount'] < 0) | (df['vehicle_claim'] < 0)]
```

months_as_customer	age	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_pre

0 rows × 36 columns

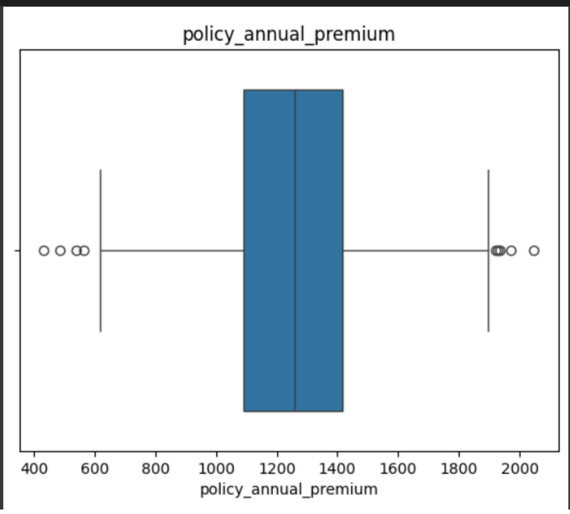
Activity 2.2: Handling Outliers

With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of Age feature with some mathematical formula.

·To find upper bound we have to multiply IQR (Interquartile range) with 1.5 and add it with 3rd quantile. To find lower bound instead of adding, subtract it with 1st quantile. Take image attached below as your reference.

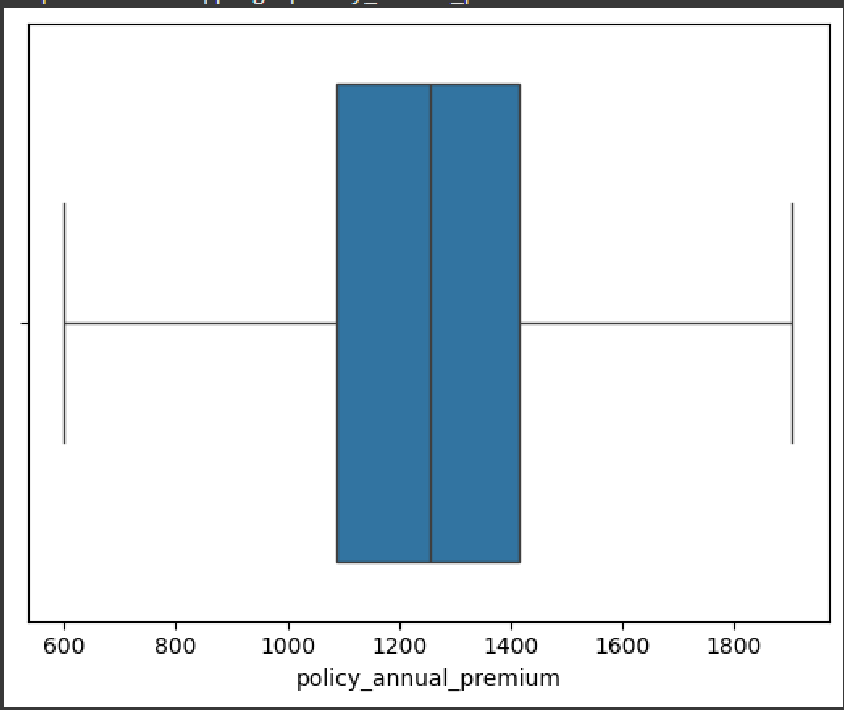
·To handle the outliers transformation technique is used. Here log transformation is used. We have created a function to visualize the distribution and probability plot of Age feature.

```python
for col in df.select_dtypes(include=['int64', 'float64']).columns:
    sns.boxplot(data=df, x=col)
    plt.title(col)
    plt.show()
```



```python
# Columns for outlier handling
num_features = ['age', 'policy_annual_premium', 'umbrella_limit', 'total_claim_amount', 'property_claim']

# Step 1: Calculate IQR for each selected feature
IQR = []
for col in num_features:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR.append(Q3 - Q1)

# Step 2: Compute upper and lower bounds
upper = []
lower = []
for i, col in enumerate(num_features):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    upper.append(Q3 + 1.5 * IQR[i])
    lower.append(Q1 - 1.5 * IQR[i])

# Step 3: Cap the outliers
df['age'] = np.where(df['age'] > upper[0], upper[0],
             np.where(df['age'] < lower[0], lower[0], df['age']))

df['policy_annual_premium'] = np.where(df['policy_annual_premium'] > upper[1], upper[1],
                               np.where(df['policy_annual_premium'] < lower[1], lower[1], df['policy_annual_premium']))

df['umbrella_limit'] = np.where(df['umbrella_limit'] > upper[2], upper[2],
                        np.where(df['umbrella_limit'] < lower[2], lower[2], df['umbrella_limit']))

df['total_claim_amount'] = np.where(df['total_claim_amount'] > upper[3], upper[3],
                            np.where(df['total_claim_amount'] < lower[3], lower[3], df['total_claim_amount']))

df['property_claim'] = np.where(df['property_claim'] > upper[4], upper[4],
                        np.where(df['property_claim'] < lower[4], lower[4], df['property_claim']))

# Step 4: Visualize the boxplots after capping
for col in num_features:
    print(f"Boxplot after capping: {col}")
    sns.boxplot(data=df, x=col)
    plt.show()
```

Boxplot after capping: policy_annual_premium



Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.