

Week 2 - Preprocessing, part 2

1. Lesson: None

2. Weekly graph question

✅ The Storytelling With Data book mentions planning on a "Who, What, and How" for your data story. Write down a possible Who, What, and How for your data, using the ideas in the book.

The *Car Claims* dataset, available on [Kaggle](#) and provided by **Angoss Knowledge Seeker**, contains 15,420 automobile insurance claims. Of these, 14,497 are legitimate and 923 are fraudulent, reflecting a significant class imbalance.

Who (Audience)

- **Fraud Investigators & Analysts:** Improve fraud detection models.
- **Claims Adjusters & Underwriters:** Assess claim validity.
- **Insurance Executives & Risk Managers:** Reduce fraud losses.

What (Key Message)

- **Fraud is rare (6%) but costly.**
- **Patterns exist** in claim amount, number of vehicles, and incident type.
- **Better detection models** can minimize losses while ensuring fair claims.

How (Visualization Approach)









- **Bar charts:** Fraud vs. non-fraud distribution.
- **Heatmaps:** Feature correlations.
- **Decision Trees:** Key fraud indicators.
- **Time trends:** Fraud patterns over time.
- **Geospatial maps:** Fraud hotspots.

3. Homework - work with your own data




```
In [32]: import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import kagglehub
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

This week, you will do the same types of exercises as last week, but you should use your own datasets that you found last semester.

Here are some types of analysis you can do Use Google, documentation, and ChatGPT to help you:

-  Summarize the datasets using `info()` and `describe()`
-  Are there any duplicate rows?
-  Are there any duplicate values in a given column (when this would be inappropriate?)
-  What are the mean, median, and mode of each column?
-  Are there any missing or null values?
 -  Do you want to fill in the missing value with a mean value? A value of your choice? Remove that row?
-  Identify any other inconsistent data (e.g. someone seems to be taking an action before they are born.)
-  Encode any categorical variables (e.g. with one-hot encoding.)

Conclusions:

-  Are the data usable? If not, find some new data!
-  Do you need to modify or correct the data in some way?
-  Is there any class imbalance? (Categories that have many more items than other categories).

```
In [33]: # set variables
kaggle_dataset_path = "khusheekapoor/vehicle-insurance-fraud-detection"
kaggle_dataset_file_name = "carclaims.csv"
print(f"Path to kaggle dataset: {kaggle_dataset_path}")
print(f"Kaggle dataset file name: {kaggle_dataset_file_name}")

# download the data set
kaggle_dataset_local_path = kagglehub.dataset_download(kaggle_dataset_path)
print(f"Path to downloaded file: {kaggle_dataset_local_path}")

# read csv file to pandas dataframe
kaggle_dataset_local_path_to_file = os.path.join(kaggle_dataset_local_path, kaggle_dataset_file_name)
kaggle_dataset_raw = pd.read_csv(kaggle_dataset_local_path_to_file)

# copy dataframe for EDA
dataset = kaggle_dataset_raw.copy()

# show top rows
dataset.head()
```

Path to kaggle dataset: khusheekapoor/vehicle-insurance-fraud-detection

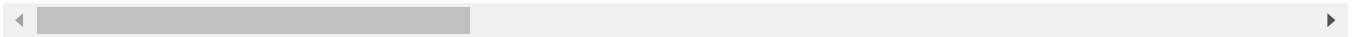
Kaggle dataset file name: carclaims.csv

Path to downloaded file: /home/codespace/.cache/kagglehub/datasets/khusheekapoor/vehicle-insurance-fraud-detection/versions/1

Out[33]:

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

5 rows × 33 columns



Summarize the datasets using info() and describe()

In [34]:

```
# Summarize the datasets using info()
dataset.info()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15420 entries, 0 to 15419
Data columns (total 33 columns):
Column Non-Null Count Dtype
--- -
0 Month 15420 non-null object
1 WeekOfMonth 15420 non-null int64
2 DayOfWeek 15420 non-null object
3 Make 15420 non-null object
4 AccidentArea 15420 non-null object
5 DayOfWeekClaimed 15420 non-null object
6 MonthClaimed 15420 non-null object
7 WeekOfMonthClaimed 15420 non-null int64
8 Sex 15420 non-null object
9 MaritalStatus 15420 non-null object
10 Age 15420 non-null int64
11 Fault 15420 non-null object
12 PolicyType 15420 non-null object
13 VehicleCategory 15420 non-null object
14 VehiclePrice 15420 non-null object
15 PolicyNumber 15420 non-null int64
16 RepNumber 15420 non-null int64
17 Deductible 15420 non-null int64
18 DriverRating 15420 non-null int64
19 Days:Policy-Accident 15420 non-null object
20 Days:Policy-Claim 15420 non-null object
21 PastNumberOfClaims 15420 non-null object
22 AgeOfVehicle 15420 non-null object
23 AgeOfPolicyHolder 15420 non-null object
24 PoliceReportFiled 15420 non-null object
25 WitnessPresent 15420 non-null object
26 AgentType 15420 non-null object
27 NumberOfSuppliments 15420 non-null object
28 AddressChange-Claim 15420 non-null object
29 NumberOfCars 15420 non-null object
30 Year 15420 non-null int64
31 BasePolicy 15420 non-null object
32 FraudFound 15420 non-null object
dtypes: int64(8), object(25)
memory usage: 3.9+ MB

In [35]: *# add target feature as integer*

```
dataset['FraudFound_Int'] = dataset['FraudFound'].map({'Yes': 1, 'No': 0})
```

In [36]: *# Summarize the datasets using describe()*

```
dataset.describe().transpose()
```

Out[36]:

	count	mean	std	min	25%	50%	75%	max
WeekOfMonth	15420.0	2.788586	1.287585	1.0	2.00	3.0	4.00	5.0
WeekOfMonthClaimed	15420.0	2.693969	1.259115	1.0	2.00	3.0	4.00	5.0
Age	15420.0	39.855707	13.492377	0.0	31.00	38.0	48.00	80.0
PolicyNumber	15420.0	7710.500000	4451.514911	1.0	3855.75	7710.5	11565.25	15420.0
RepNumber	15420.0	8.483268	4.599948	1.0	5.00	8.0	12.00	16.0
Deductible	15420.0	407.704280	43.950998	300.0	400.00	400.0	400.00	700.0
DriverRating	15420.0	2.487808	1.119453	1.0	1.00	2.0	3.00	4.0
Year	15420.0	1994.866472	0.803313	1994.0	1994.00	1995.0	1996.00	1996.0
FraudFound_Int	15420.0	0.059857	0.237230	0.0	0.00	0.0	0.00	1.0

Are there any duplicate rows?

In [37]: *# Are there any duplicate rows?*

```
# Note: There is no information describing primary key, therefore first duplicates check for c  
print("Number of duplicated Rows: ", dataset.duplicated().sum())
```

```
# show duplicated if any:
```

```
if dataset.duplicated().sum() > 0:  
    dataset_duplicates = dataset[dataset.duplicated()]  
    print("Duplicated Rows: ", dataset_duplicates)  
    # clean up from duplicates  
    deleted_duplicates = dataset.drop_duplicates()  
    print("Deleted Duplicated Rows:", dataset_duplicates)
```

```
else:  
    print("No duplicates found.")
```

Number of duplicated Rows: 0

No duplicates found.

What are the mean, median, and mode of each column?

In [38]: *# What are the mean, median, and mode of each column?*

```
# select only numeric columns
```

```
numeric_dataset = dataset.select_dtypes(include=['number'])
```

```
# compute mean, median, and mode only for numeric columns
```

```
summary_stats = numeric_dataset.agg(['mean', 'median']).transpose()
```

```
# compute mode separately
```

```
mode_values = numeric_dataset.mode().iloc[0] # First mode for each column  
summary_stats['mode'] = mode_values  
print("Mean, median, and mode:")  
summary_stats
```

Mean, median, and mode:

Out[38]:

	mean	median	mode
WeekOfMonth	2.788586	3.0	3.0
WeekOfMonthClaimed	2.693969	3.0	2.0
Age	39.855707	38.0	30.0
PolicyNumber	7710.500000	7710.5	1.0
RepNumber	8.483268	8.0	7.0
Deductible	407.704280	400.0	400.0
DriverRating	2.487808	2.0	1.0
Year	1994.866472	1995.0	1994.0
FraudFound_Int	0.059857	0.0	0.0

In [39]:

```
# add some very basic data profiling:
# shape
print(f"This dataset contain {dataset.shape[0]} rows")
print(f"This dataset contain {dataset.shape[1]} columns")

# basic aggregates
data_profiling = dataset.agg(['count', 'nunique', 'min', 'max']).transpose()
data_profiling
```

This dataset contain 15420 rows

This dataset contain 34 columns

Out[39]:

	count	nunique	min	max
Month	15420	12	Apr	Sep
WeekOfMonth	15420	5	1	5
DayOfWeek	15420	7	Friday	Wednesday
Make	15420	19	Accura	VW
AccidentArea	15420	2	Rural	Urban
DayOfWeekClaimed	15420	8	0	Wednesday
MonthClaimed	15420	13	0	Sep
WeekOfMonthClaimed	15420	5	1	5
Sex	15420	2	Female	Male
MaritalStatus	15420	4	Divorced	Widow
Age	15420	66	0	80
Fault	15420	2	Policy Holder	Third Party
PolicyType	15420	9	Sedan - All Perils	Utility - Liability
VehicleCategory	15420	3	Sedan	Utility
VehiclePrice	15420	6	20,000 to 29,000	more than 69,000
PolicyNumber	15420	15420	1	15420
RepNumber	15420	16	1	16
Deductible	15420	4	300	700
DriverRating	15420	4	1	4
Days:Policy-Accident	15420	5	1 to 7	none
Days:Policy-Claim	15420	4	15 to 30	none
PastNumberOfClaims	15420	4	1	none
AgeOfVehicle	15420	8	2 years	new
AgeOfPolicyHolder	15420	9	16 to 17	over 65
PoliceReportFiled	15420	2	No	Yes
WitnessPresent	15420	2	No	Yes
AgentType	15420	2	External	Internal
NumberOfSuppliments	15420	4	1 to 2	none
AddressChange-Claim	15420	5	1 year	under 6 months
NumberOfCars	15420	5	1 vehicle	more than 8
Year	15420	3	1994	1996
BasePolicy	15420	3	All Perils	Liability
FraudFound	15420	2	No	Yes
FraudFound_Int	15420	2	0	1

Are there any duplicate values in a given column (when this would be inappropriate?)

```
In [40]: # get list of columns where 'count' is equal to 'nunique', these are our natural key candidates
data_profiling_unique_columns = data_profiling[data_profiling['count'] == data_profiling['nunique']]
data_profiling_unique_columns
```

```
Out[40]:
```

	count	nunique	min	max
PolicyNumber	15420	15420	1	15420

Based on the data:

- PolicyNumber has 15,420 unique values, which matches the total row count (15,420).
- The min value is 1 and the max value is 15,420, indicating a continuous sequence.
- The data type is int64, confirming it's numeric.

Conclusion:

- PolicyNumber is a valid primary key for uniquely identifying each row in the dataset.
- Not useful for analysis since it carries no meaningful information about the claims.

```
In [41]: # Assume the data must have a unique ID, such as PolicyNumber
# This is how duplicates can be detected
print(f"Duplicates of Policy Numbers: {dataset['PolicyNumber'].duplicated().sum()}")

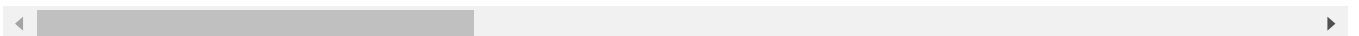
# and deleted
dataset_with_unique_policy_numbers = dataset.drop_duplicates(subset=['PolicyNumber'], keep='first')
dataset_with_unique_policy_numbers.head()
```

Duplicates of Policy Numbers: 0

```
Out[41]:
```

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

5 rows × 34 columns



Are there any duplicate values in a given column (when this would be inappropriate?)

Inappropriate duplicates could be :

- Primary or natural keys: Unique identifiers such as a claim number(if provided), or a composite key formed by combining multiple columns (e.g., PolicyNumber + Timestamp).
- Timestamps: Expected to be unique per claim, but collisions may occur.

Appropriate duplicates are :

- Categorical attributes: Examples include Month , Sex , Marital Status , etc.
- Numerical values: Examples include Deductible Amount , etc.

Are there any missing or null values?

```
In [42]: # Are there any null values?

# check null Values
missing_values_number = dataset.isnull().sum().sum()
if missing_values_number > 0:
    print(f"This dataframe contain null values = {missing_values_number}")
else:
    print("No null values detected.")
```

No null values detected.

Do you want to fill in the missing value with a mean value? A value of your choice? Remove that row?

- **Few missing values or rows:** Remove the rows.
- **Significant percentage of missing values:**
 - **Numerical columns:** Fill with the mean.
 - **Categorical columns:** Fill with "Unknown" and encode.

Identify any other inconsistent data (e.g. someone seems to be taking an action before they are born.)

Based on the `data_profiling` dataframe:

- Column `MonthClaimed` should contain valid month names (e.g., Jan, Feb, ..., Dec), but some rows have a value of 0, which is an inconsistency.
- The column `Age` contains values of 0, which may indicate inconsistencies or errors.
- The column `DayOfWeekClaimed` has 8 distinct values, which may indicate inconsistencies or errors, as there should only be 7 possible values (Monday to Sunday).
- There are duplicated features like `AgeOfPolicyHolder` (categorical age range) and `Age` (exact age).

Encode any categorical variables (e.g. with one-hot encoding.)

```
In [43]: # One-hot encode categorical columns
dataset_encoded = pd.get_dummies(dataset, columns=['Month', 'Make'])
dataset_encoded.head()
```


Out[43]:

	WeekOfMonth	DayOfWeek	AccidentArea	DayOfWeekClaimed	MonthClaimed	WeekOfMonthClaim
0	5	Wednesday	Urban	Tuesday	Jan	
1	3	Wednesday	Urban	Monday	Jan	
2	5	Friday	Urban	Thursday	Nov	
3	2	Saturday	Rural	Friday	Jul	
4	5	Monday	Urban	Tuesday	Feb	



5 rows × 63 columns

Conclusions:

- *Are the data usable? If not, find some new data!*
 - The data appears usable, with minor inconsistencies.
- *Do you need to modify or correct the data in some way?*
 - There are some inconsistencies that can be fixed:
 - `MonthClaimed` : Rows with 0 values can be removed.
 - `Age` : This feature can be dropped, or 0 values can be replaced with the median age from the corresponding `AgeOfPolicyHolder` (categorical age range).
 - `DayOfWeekClaimed` : Rows with 0 values can be removed.
- *Is there any class imbalance? (Categories that have many more items than other categories).*
 - Most categorical columns have fewer than 10 unique values, while `Age` has 64 distinct categories.

4. Storytelling With Data graph

Just like last week: choose any graph in the Introduction of Storytelling With Data.

-  Use matplotlib to reproduce it in a rough way. I don't expect you to spend an enormous amount of time on this; I understand that you likely will not have time to re-create every feature of the graph. However, if you're excited about learning to use matplotlib, this is a good way to do that. You don't have to duplicate the exact values on the graph; just the same rough shape will be enough.
-  If you don't feel comfortable using matplotlib yet, do the best you can and write down what you tried or what Google searches you did to find the answers.

In [44]:

```
df = dataset.copy()

fraud_counts = df.groupby(["WitnessPresent", "FraudFound"]).size().unstack(fill_value=0)

fig, ax = plt.subplots(figsize=(8, 5))

# plot the bars side by side (stacked=False for grouped bars)
```

```

bars = fraud_counts.plot(kind='bar', stacked=False, ax=ax, color=['blue', 'red'], width=0.8)

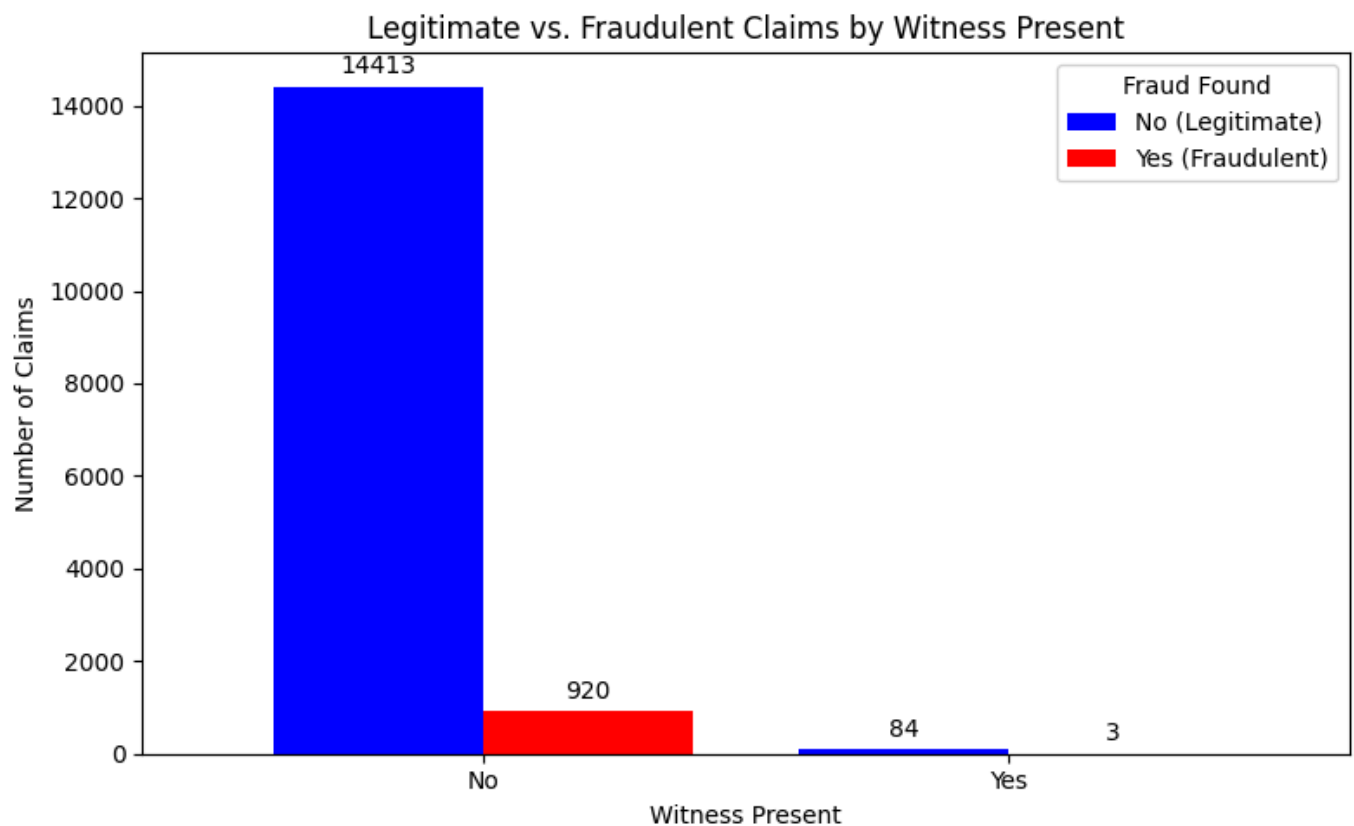
# adjusting labels on bars to prevent overlapping
for bar_container in bars.containers:
    ax.bar_label(bar_container, fmt='%d', padding=3, fontsize=10)

# add labels and title
ax.set_xlabel("Witness Present")
ax.set_ylabel("Number of Claims")
ax.set_title("Legitimate vs. Fraudulent Claims by Witness Present")
ax.legend(title="Fraud Found", labels=["No (Legitimate)", "Yes (Fraudulent)"])

# adjust tick rotation and layout
plt.xticks(rotation=0)
plt.tight_layout()

# show the chart
plt.show()

```



```

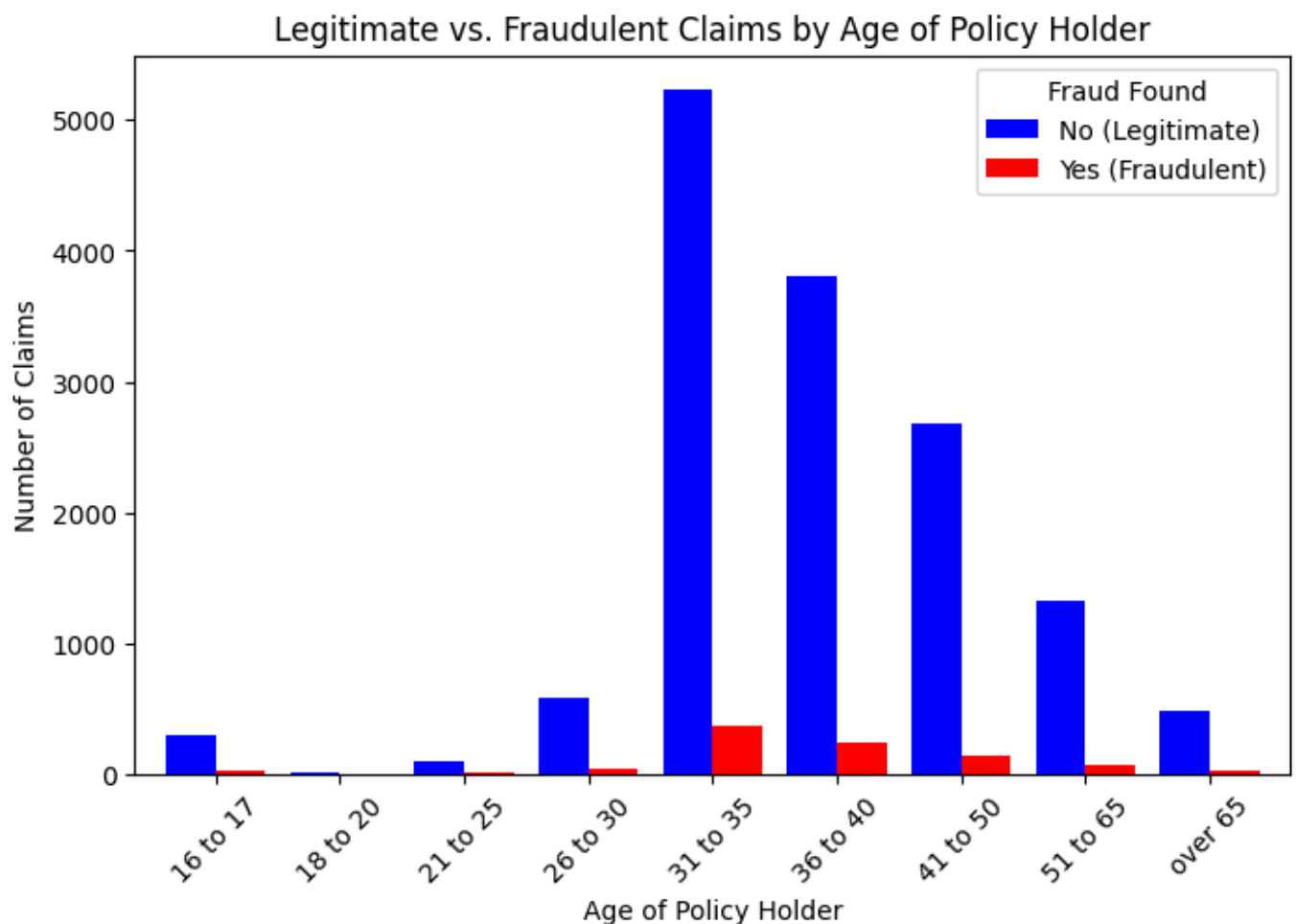
In [45]: fraud_counts = df.groupby(["AgeOfPolicyHolder", "FraudFound"]).size().unstack(fill_value=0)

# Plot grouped bar chart (bars next to each other)
fig, ax = plt.subplots(figsize=(8, 5))
fraud_counts.plot(kind='bar', stacked=False, ax=ax, color=['blue', 'red'], width=0.8)

# add labels and title
ax.set_xlabel("Age of Policy Holder")
ax.set_ylabel("Number of Claims")
ax.set_title("Legitimate vs. Fraudulent Claims by Age of Policy Holder")
ax.legend(title="Fraud Found", labels=["No (Legitimate)", "Yes (Fraudulent)"])

# show the chart
plt.xticks(rotation=45)
plt.show()

```



Write down what you tried or what Google searches you did to find the answers.

I've tried multiple combinations of categorical features, aiming to keep the chart readable while avoiding unnecessary complexity.

I prefer bar charts as they are the easiest to read and interpret. I also searched on Google and found other projects on Kaggle with this dataset, but most of them focus on machine learning rather than descriptive analytics.

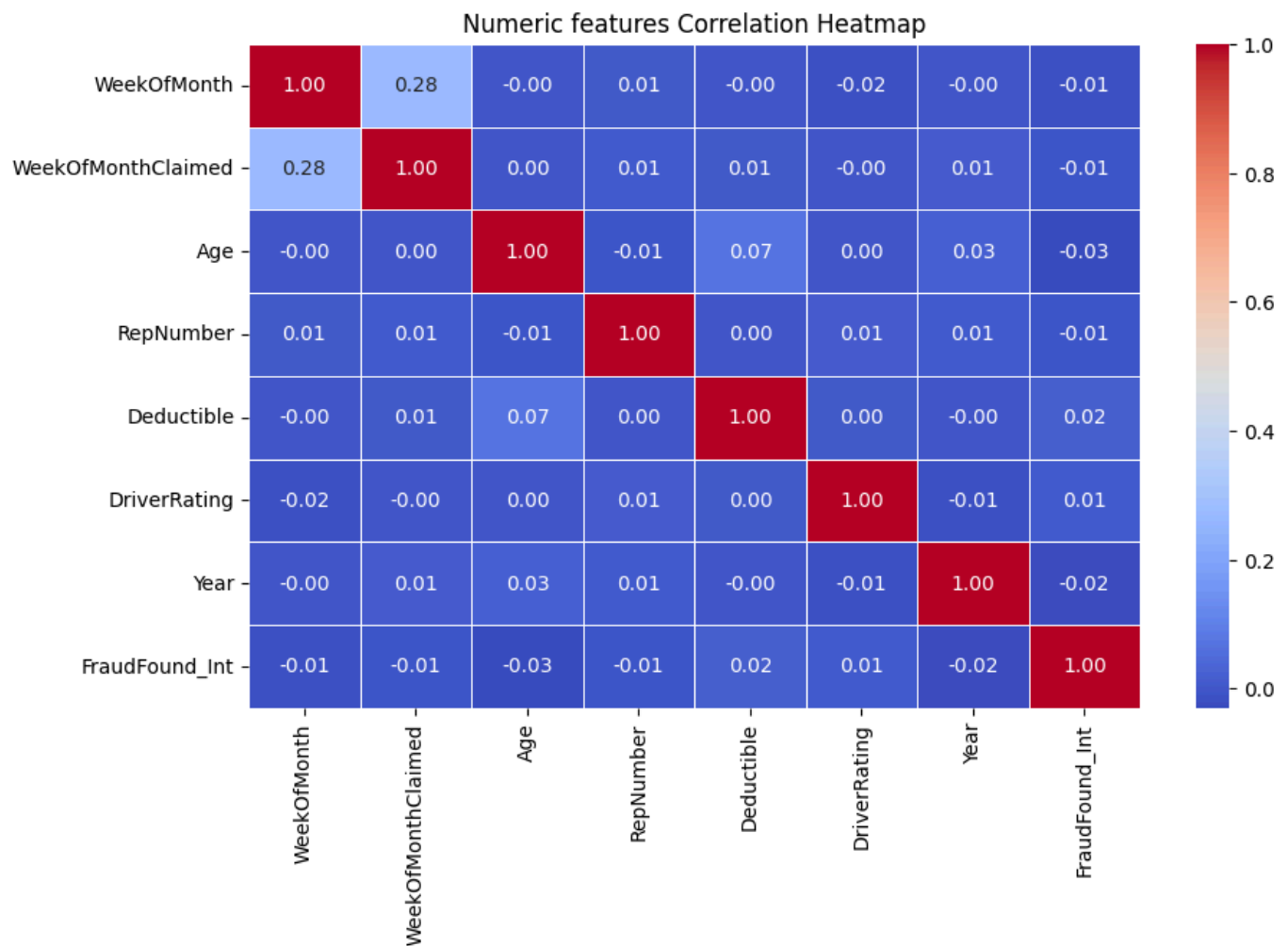
Additionally, I added a heatmap below to visualize correlations and try to identify patterns in the data.

```
In [46]: # Feature Correlation Heatmap
# use only numeric features for the heat map
# do not use `PolicyNumber`, leaving them for the numerologists.
df_numeric = dataset.select_dtypes(include=['number']).drop(columns=['PolicyNumber'])

# plot heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df_numeric.corr(), annot=True, fmt=".2f", cmap="coolwarm", linewidths=0.5)

# add title
plt.title("Numeric features Correlation Heatmap")

# show the heatmap
plt.show()
```



Heatmap observations

The heatmap suggests that no strong correlations exist among the **numeric** features, meaning each variable is relatively independent.

More feature engineering is needed, such as encoding categorical variables or creating new derived features to improve model performance.

References

1. Kaggle. (n.d.). *Vehicle Insurance Fraud Detection*. Retrieved from <https://www.kaggle.com/datasets/khusheekapoor/vehicle-insurance-fraud-detection/data>
2. Nelyapenko, (2025). *ModB-Sem2 Weekly Homework: Week 02. OMDS-ModB2-Week2-Nelyapenko-Sergey.ipynb* *GitHub*. Retrieved from <https://github.com/SergeyNelyapenkoBU/2025-spring-B2/blob/main/OMDS-ModB2-Week2-Nelyapenko-Sergey.ipynb>