Insurance Claims Analysis

May 8, 2023

1 Insurance Claims Analysis - Group Project ACT SCI 657

The data is sourced from Kaggle, it can be accessed through the following link:

https://www.kaggle.com/datasets/buntyshah/auto-insurance-claims-data?datasetId=45152&sortBy=voteCount

[ ]:
```
# Import the required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import␣
↪accuracy_score,confusion_matrix,classification_report
```

2 Data Reading

[ ]:
```
# Read the xls file
df=pd.read_excel('insurance_claims.xlsx')
```

[ ]:
```
# Preview the data
df.head()
```

[ ]:
```
   months_as_customer  age  policy_number policy_bind_date policy_state  \
0                 328   48         521585       2014-10-17           OH
1                 228   42         342868       2006-06-27           IN
2                 134   29         687698       2000-09-06           OH
3                 256   41         227811       1990-05-25           IL
4                 228   44         367455       2014-06-06           IL

  policy_csl  policy_deductable  policy_annual_premium  umbrella_limit  \
0    250/500               1000                1406.91               0
1    250/500               2000                1197.22         5000000
```

```
2    100/300                2000                1413.14          5000000
3    250/500                2000                1415.74          6000000
4   500/1000                1000                1583.91          6000000

   insured_zip  … police_report_available total_claim_amount injury_claim  \
0       466132  …                     YES              71610         6510
1       468176  …                       ?               5070          780
2       430632  …                      NO              34650         7700
3       608117  …                      NO              63400         6340
4       610706  …                      NO               6500         1300

  property_claim vehicle_claim  auto_make  auto_model auto_year  \
0          13020         52080       Saab         92x      2004
1            780          3510   Mercedes        E400      2007
2           3850         23100      Dodge         RAM      2007
3           6340         50720  Chevrolet       Tahoe      2014
4            650          4550     Accura         RSX      2009

  fraud_reported _c39
0              Y  NaN
1              Y  NaN
2              N  NaN
3              Y  NaN
4              N  NaN

[5 rows x 40 columns]
```

[ ]:
```python
# List the columns
df.columns
```

[ ]:
```
Index(['months_as_customer', 'age', 'policy_number', 'policy_bind_date',
       'policy_state', 'policy_csl', 'policy_deductable',
       'policy_annual_premium', 'umbrella_limit', 'insured_zip', 'insured_sex',
       'insured_education_level', 'insured_occupation', 'insured_hobbies',
       'insured_relationship', 'capital-gains', 'capital-loss',
       'incident_date', 'incident_type', 'collision_type', 'incident_severity',
       'authorities_contacted', 'incident_state', 'incident_city',
       'incident_location', 'incident_hour_of_the_day',
       'number_of_vehicles_involved', 'property_damage', 'bodily_injuries',
       'witnesses', 'police_report_available', 'total_claim_amount',
       'injury_claim', 'property_claim', 'vehicle_claim', 'auto_make',
       'auto_model', 'auto_year', 'fraud_reported', '_c39'],
      dtype='object')
```

3 Exploratory Data Analysis

[ ]:
```
# Summary the numerical columns
df.describe()
```

[ ]:

	months_as_customer	age	policy_number	policy_deductable
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	203.954000	38.948000	546238.648000	1136.000000
std	115.113174	9.140287	257063.005276	611.864673
min	0.000000	19.000000	100804.000000	500.000000
25%	115.750000	32.000000	335980.250000	500.000000
50%	199.500000	38.000000	533135.000000	1000.000000
75%	276.250000	44.000000	759099.750000	2000.000000
max	479.000000	64.000000	999435.000000	2000.000000

	policy_annual_premium	umbrella_limit	insured_zip	capital-gains
count	1000.000000	1.000000e+03	1000.000000	1000.000000
mean	1256.406150	1.101000e+06	501214.488000	25126.100000
std	244.167395	2.297407e+06	71701.610941	27872.187708
min	433.330000	-1.000000e+06	430104.000000	0.000000
25%	1089.607500	0.000000e+00	448404.500000	0.000000
50%	1257.200000	0.000000e+00	466445.500000	0.000000
75%	1415.695000	0.000000e+00	603251.000000	51025.000000
max	2047.590000	1.000000e+07	620962.000000	100500.000000

	capital-loss	incident_hour_of_the_day	number_of_vehicles_involved
count	1000.000000	1000.000000	1000.00000
mean	-26793.700000	11.644000	1.83900
std	28104.096686	6.951373	1.01888
min	-111100.000000	0.000000	1.00000
25%	-51500.000000	6.000000	1.00000
50%	-23250.000000	12.000000	1.00000
75%	0.000000	17.000000	3.00000
max	0.000000	23.000000	4.00000

	bodily_injuries	witnesses	total_claim_amount	injury_claim
count	1000.000000	1000.000000	1000.00000	1000.000000
mean	0.992000	1.487000	52761.94000	7433.420000
std	0.820127	1.111335	26401.53319	4880.951853
min	0.000000	0.000000	100.00000	0.000000
25%	0.000000	1.000000	41812.50000	4295.000000
50%	1.000000	1.000000	58055.00000	6775.000000
75%	2.000000	2.000000	70592.50000	11305.000000
max	2.000000	3.000000	114920.00000	21450.000000

	property_claim	vehicle_claim	auto_year	_c39
count	1000.000000	1000.000000	1000.000000	0.0

```
mean      7399.570000   37928.950000  2005.103000   NaN
std       4824.726179   18886.252893     6.015861   NaN
min          0.000000      70.000000  1995.000000   NaN
25%       4445.000000   30292.500000  2000.000000   NaN
50%       6750.000000   42100.000000  2005.000000   NaN
75%      10885.000000   50822.500000  2010.000000   NaN
max      23670.000000   79560.000000  2015.000000   NaN
```

[ ]:
```python
# Info the data
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   datetime64[ns]
 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   datetime64[ns]
 18  incident_type                1000 non-null   object
 19  collision_type               1000 non-null   object
 20  incident_severity            1000 non-null   object
 21  authorities_contacted        1000 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: datetime64[ns](2), float64(2), int64(17), object(19)
memory usage: 312.6+ KB
```

[ ]:
```
# Filet the columns with null values
df.isnull().sum()[df.isnull().sum()>0]
```

[ ]:
```
_c39    1000
dtype: int64
```

[ ]:
```
# Drop _c39 column
df.drop('_c39',axis=1,inplace=True)
```

[ ]:
```
# Replace the '?' with nan
df.replace('?',np.nan,inplace=True)
```

[ ]:
```
# Total amount of rows
df.shape
```

[ ]:
```
(1000, 39)
```

[ ]:
```
# Create a list of column names with data type non-numerical
cat_cols=df.select_dtypes(exclude=np.number).columns.tolist()
print(cat_cols)
```

```
['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',
'incident_location', 'property_damage', 'police_report_available', 'auto_make',
'auto_model', 'fraud_reported']
```

[ ]:
```
# Add the policy number, insurance zip, insurance location to the cat_cols list
cat_cols.extend(['policy_number','insured_zip'])
```

[ ]:
```
# Create a funciton to count the unique values in each column
def unique_values(df):
    for i in df.columns:
        print(i,df[i].nunique())
```

[ ]:
```
# Apply the function to the categorical columns
unique_values(df[cat_cols])
```

```
policy_bind_date 951
policy_state 3
policy_csl 3
insured_sex 2
insured_education_level 7
insured_occupation 14
insured_hobbies 20
insured_relationship 6
incident_date 60
incident_type 4
collision_type 3
incident_severity 4
authorities_contacted 5
incident_state 7
incident_city 7
incident_location 1000
property_damage 2
police_report_available 2
auto_make 14
auto_model 39
fraud_reported 2
policy_number 1000
insured_zip 995
```

From the set of categorical variables, we can see that some of them have over 900 different values, which is something to keep in mind as we go further into the prediciton models.

The variables on this condition are:

- policy__blind__date
- incident__location
- policy__number
- insured__zip

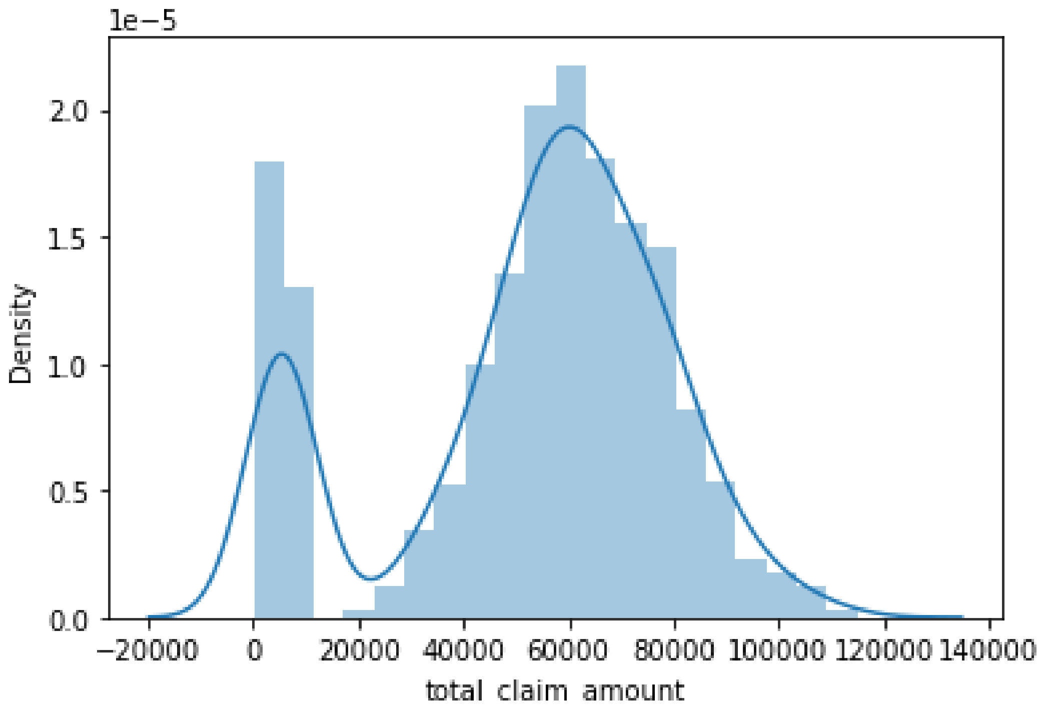
[ ]:
```
# Let's check the target variable by plotting fraud_reported
sns.countplot(df['fraud_reported']);
```

[ ]:
```
# Count the fraud_reported
df['fraud_reported'].value_counts()
```

[ ]:
```
N    753
Y    247
Name: fraud_reported, dtype: int64
```

There is a considerable difference between the fraud and non-fraud claims, but is expected given the nature of these events.

[ ]:
```
# Histogram of total claim amount
sns.distplot(df['total_claim_amount']);
```



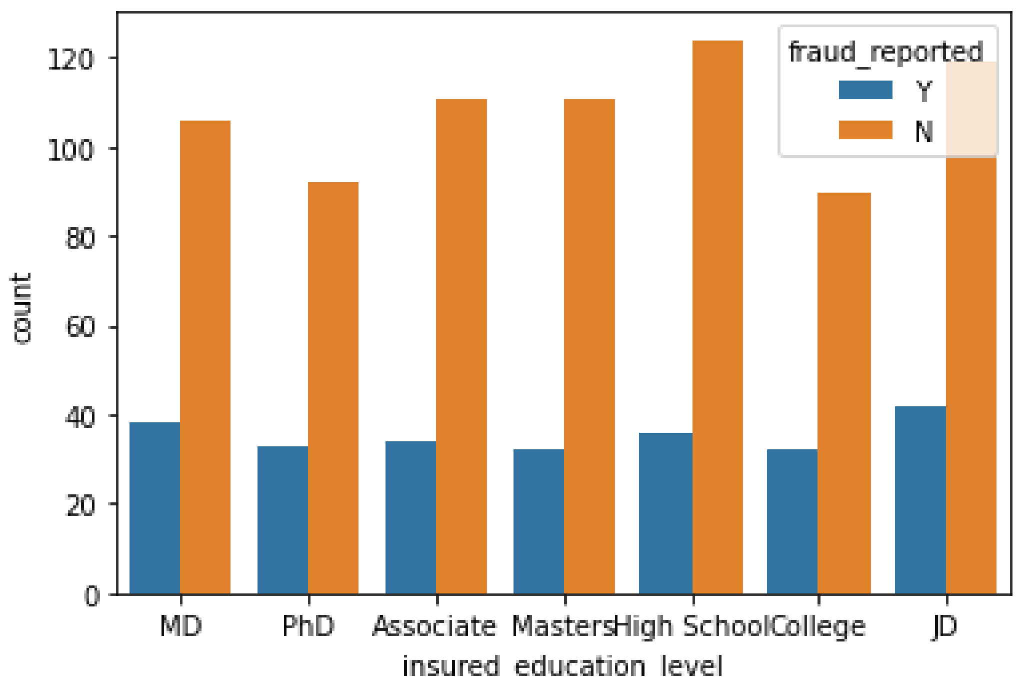

```
[ ]: # Count fraud by Incident State, for Y fraud_reported
     df[df['fraud_reported']=='Y']['incident_state'].value_counts()
```

```
[ ]: SC    73
     NY    58
     WV    39
     NC    34
     VA    25
     OH    10
     PA     8
     Name: incident_state, dtype: int64
```

```
[ ]: # Using a USA map to plot the count of fraud_reported = Y by state, color by␣
      ↪number of fraud_reported

     # Import the required libraries
     import plotly.express as px
     import plotly.graph_objects as go

     # Create a dataframe with the count of fraud_reported = Y by state
     df_state=df[df['fraud_reported']=='Y']['incident_state'].value_counts().
      ↪reset_index()
     df_state.columns=['state','count']
```

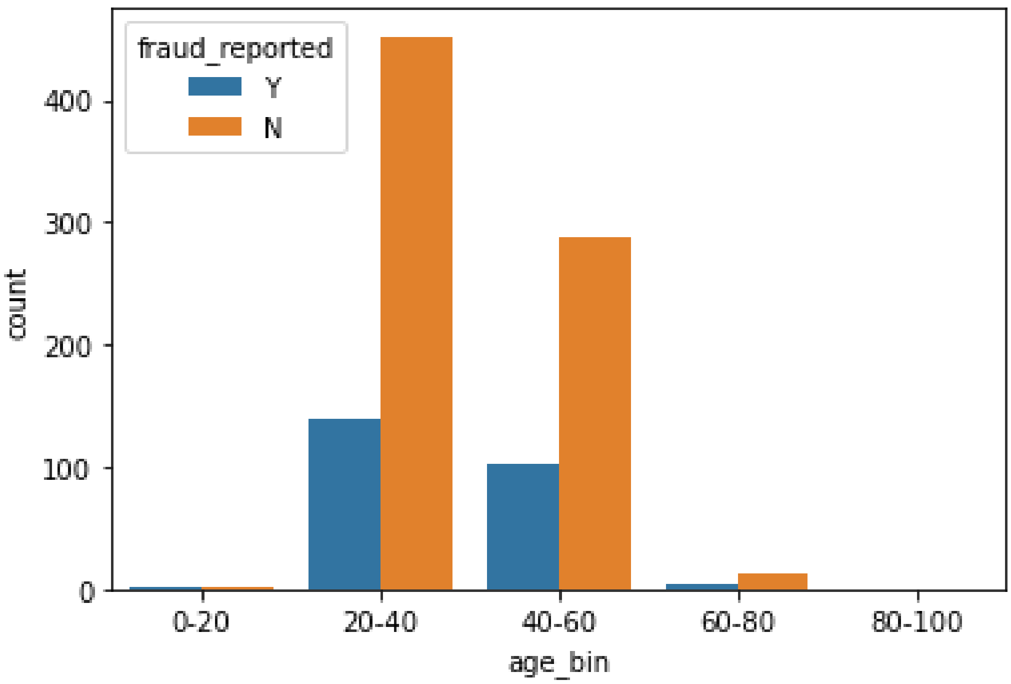
```
# Create a USA map
fig = go.Figure(data=go.Choropleth(
    locations=df_state['state'], # Spatial coordinates
    z = df_state['count'].astype(float), # Data to be color-coded
    locationmode = 'USA-states', # set of locations match entries in `locations`
    colorscale = 'Blues',
    colorbar_title = "Fraud Reported",
))

fig.update_layout(
    title_text = 'Fraud Reported by State',
    geo_scope='usa', # limite map scope to USA
)

fig.show();
```

[ ]:
```
# Plot Breakdown of insuranced education claim group by fraud_reported
sns.countplot(df['insured_education_level'],hue=df['fraud_reported']);
```

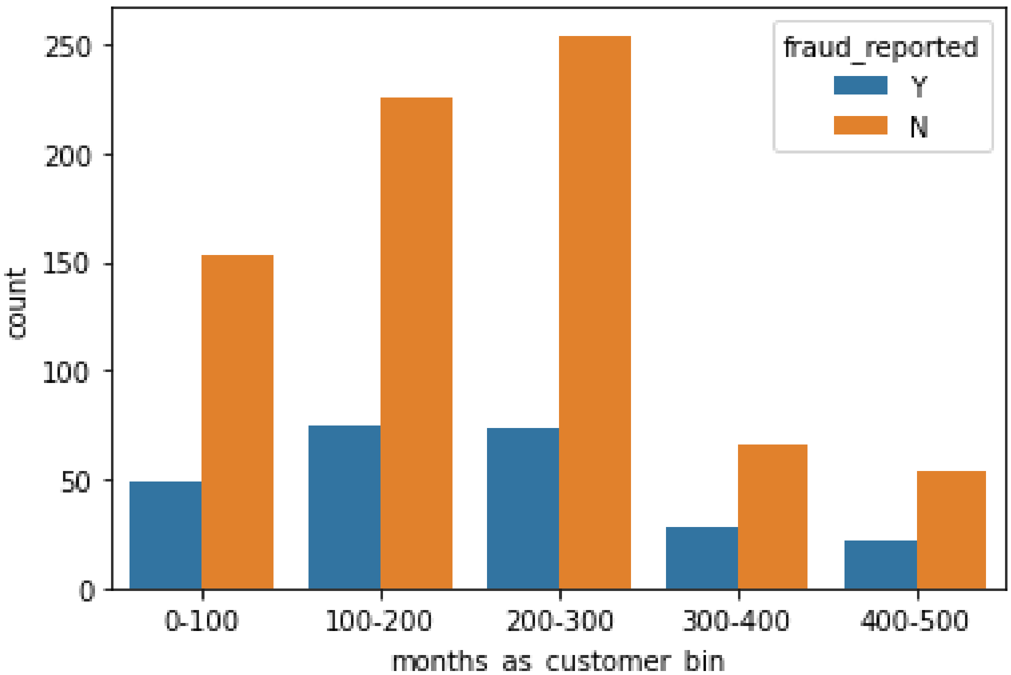


[ ]:
```
# Bin the age column group by fraud_reported
df['age_bin']=pd.
 ↪cut(df['age'],bins=[0,20,40,60,80,100],labels=['0-20','20-40','40-60','60-80','80-100'])
```

```
# Plot Breakdown of insuranced age claim group by fraud_reported
sns.countplot(df['age_bin'],hue=df['fraud_reported']);
```



[ ]:
```
# Bin the months_as_customer column group by fraud_reported
df['months_as_customer_bin']=pd.
 ↪cut(df['months_as_customer'],bins=[0,100,200,300,400,500],labels=['0-100','100-200','200-30

# Plot Breakdown of insuranced months_as_customer claim group by fraud_reported
sns.countplot(df['months_as_customer_bin'],hue=df['fraud_reported']);
```



4 Data Preprocessing

Considering the for non-numerical variables we will have to generate dummy variables, let's drop the high volume distinct values variables. Let's also add incident date given the is a time series variable.

[ ]:
```
cols_to_drop=['policy_number','incident_location','policy_bind_date',␣
↪'insured_zip', 'incident_date', 'age_bin', 'months_as_customer_bin']
```

[ ]:
```
# Excluse the cols_to_drop from the cat_cols list
cat_cols=[i for i in cat_cols if i not in cols_to_drop]
```

[ ]:
```
# Drop the columns
df.drop(cols_to_drop,axis=1,inplace=True)
```

[ ]:
```
# Print the types of the cat_cols
df[cat_cols].dtypes
```

[ ]:
```
policy_state                object
policy_csl                  object
insured_sex                 object
insured_education_level     object
insured_occupation          object
insured_hobbies             object
```

```
insured_relationship        object
incident_type               object
collision_type              object
incident_severity           object
authorities_contacted       object
incident_state              object
incident_city               object
property_damage             object
police_report_available     object
auto_make                   object
auto_model                  object
fraud_reported              object
dtype: object
```

[ ]:
```
# Conver the cat_cols to object type
df[cat_cols]=df[cat_cols].astype('string')
```

[ ]:
```
# Set the column of cat_cols as factors
for i in cat_cols:
    df[i]=df[i].astype('category')
```

[ ]:
```
# Replace the nan values with mode
for i in cat_cols:
    df[i].fillna(df[i].mode()[0],inplace=True)
```

[ ]:
```
# Convert the categorical columns to numeric to pass them using string indexer
cat_cols=df.select_dtypes(exclude=np.number).columns.tolist()
cat_cols

# Create a function to convert the categorical columns to numeric
def string_indexer(df,cols):
    for col in cols:
        le=LabelEncoder()
        df[col]=le.fit_transform(df[col])
    return df

# Apply the function to the categorical columns
df_indexed=string_indexer(df,cat_cols)

# Preview the data
df_indexed.head()
```

[ ]:
```
   months_as_customer  age  policy_state  policy_csl  policy_deductable  \
0                 328   48             2           1               1000
1                 228   42             1           1               2000
2                 134   29             2           0               2000
3                 256   41             0           1               2000
```

```
4                 228   44             0            2                1000

   policy_annual_premium  umbrella_limit  insured_sex  \
0                1406.91               0            1
1                1197.22         5000000            1
2                1413.14         5000000            0
3                1415.74         6000000            0
4                1583.91         6000000            1

   insured_education_level  insured_occupation  …  witnesses  \
0                        4                   2  …          2
1                        4                   6  …          0
2                        6                  11  …          3
3                        6                   1  …          2
4                        0                  11  …          1

   police_report_available  total_claim_amount  injury_claim  property_claim  \
0                        1               71610          6510           13020
1                        0                5070           780             780
2                        0               34650          7700            3850
3                        0               63400          6340            6340
4                        0                6500          1300             650

   vehicle_claim  auto_make  auto_model  auto_year  fraud_reported
0          52080         10           1       2004               1
1           3510          8          12       2007               1
2          23100          4          30       2007               0
3          50720          3          34       2014               1
4           4550          0          31       2009               0

[5 rows x 34 columns]
```

5 GLM - Gamma Regression for Claim Amount Prediction

The purpose of the following section is to generate a prediction model for the claim amount. The model will be based on a Gamma Regression, which is a generalized linear model (GLM) for predicting continuous positive variables.

As a business problem, insurance companies need to be able to predict the claim amount in order to set the premium for the policy. The premium is the amount of money that the policy holder pays to the insurance company in order to be covered. As other option, the interest to predict claim amount might be rooted on the need to predict the amount of money that the insurance company will have to pay to the policy holder, in a case of a claim, so the company can set aside the resources to react to the claim.

[ ]:
```python
# Import the sklearn required libraries
from sklearn.linear_model import GammaRegressor
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
[ ]: # Load the data and split into train and test sets
X = df_indexed.drop('total_claim_amount', axis=1)
y = df_indexed['total_claim_amount']

# Scale the predictor variables
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.4,␣
 ↪random_state=42)

# Create a gamma regression model
gamma_model = GammaRegressor()

# Fit the model on the training data
gamma_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = gamma_model.predict(X_test)
```

```
Mean Squared Error: 116374358.92575014
```

```
[ ]: # Import the performance metrics library
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
[ ]: # Calculate the Model Performance Metrics
print('R2 Score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))
print('MSE:', mean_squared_error(y_test, y_pred))

# Create a function to calculate the adjusted R2
def adj_r2(X,y):
    r2 = gamma_model.score(X,y)
    n = X.shape[0]
    p = X.shape[1]
    adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
    return adjusted_r2

# Calculate the adjusted R2
print('Adjusted R2:', adj_r2(X_test, y_test))
```

```
R2 Score: 0.8286642792363856
MAE: 9110.823275061093
MSE: 116374358.9257501
```

```
Adjusted R2: 0.6368596131291319
```

The model performance metrics suggest that the gamma regression model is a good fit for the data, with an R2 score of 0.83 indicating that the model explains approximately 83% of the variance in the claim amount.

However, the model's predictions are off by an average of $9,110.82, as indicated by the mean absolute error (MAE) of 9110.82.

Additionally, the mean squared error (MSE) of 116,374,358.93 suggests that the model's predictions have a larger spread of errors compared to the MAE.

Finally, the adjusted R2 of 0.637 indicates that the model's performance may be slightly impacted by the number of predictor variables used. Overall, the results suggest that the gamma regression model is a good starting point for predicting claim amount, but there may be room for improvement with further model refinement.

5.1 Feature Selection

The Gamma model had a considerable amount of variables, for efficiency purposes, and better understanding of the most important variables, we will proceed to generate feature selection through backward elimination.

[ ]:
```python
# Count the variables in the model
print('Number of variables in the model:', len(gamma_model.coef_))
```

```
Number of variables in the model: 33
```

[ ]:
```python
# Perform a backward feature selection through recursive feature elimination
from sklearn.feature_selection import RFE

# Create the RFE with a gamma regression estimator and 10 features to select
rfe = RFE(estimator=GammaRegressor(), n_features_to_select=10, verbose=1)

# Fit the eliminator to the data
rfe.fit(X_scaled, y)
```

```
Fitting estimator with 33 features.
Fitting estimator with 32 features.
Fitting estimator with 31 features.
Fitting estimator with 30 features.
Fitting estimator with 29 features.
Fitting estimator with 28 features.
Fitting estimator with 27 features.
Fitting estimator with 26 features.
Fitting estimator with 25 features.
Fitting estimator with 24 features.
Fitting estimator with 23 features.
Fitting estimator with 22 features.
Fitting estimator with 21 features.
Fitting estimator with 20 features.
```

```
Fitting estimator with 19 features.
Fitting estimator with 18 features.
Fitting estimator with 17 features.
Fitting estimator with 16 features.
Fitting estimator with 15 features.
Fitting estimator with 14 features.
Fitting estimator with 13 features.
Fitting estimator with 12 features.
Fitting estimator with 11 features.
```

[ ]: RFE(estimator=GammaRegressor(), n_features_to_select=10, verbose=1)

[ ]:
```python
# Print the features and their ranking (high = dropped early on)
print(dict(zip(X.columns, rfe.ranking_)))
```

```
{'months_as_customer': 16, 'age': 22, 'policy_state': 20, 'policy_csl': 6,
'policy_deductable': 8, 'policy_annual_premium': 7, 'umbrella_limit': 10,
'insured_sex': 14, 'insured_education_level': 2, 'insured_occupation': 24,
'insured_hobbies': 1, 'insured_relationship': 13, 'capital-gains': 12, 'capital-
loss': 17, 'incident_type': 1, 'collision_type': 23, 'incident_severity': 1,
'authorities_contacted': 1, 'incident_state': 3, 'incident_city': 19,
'incident_hour_of_the_day': 1, 'number_of_vehicles_involved': 1,
'property_damage': 21, 'bodily_injuries': 18, 'witnesses': 11,
'police_report_available': 5, 'injury_claim': 1, 'property_claim': 1,
'vehicle_claim': 1, 'auto_make': 9, 'auto_model': 15, 'auto_year': 4,
'fraud_reported': 1}
```

[ ]:
```python
# Print the features that are not eliminated
print(X.columns[rfe.support_])
```

```
Index(['insured_hobbies', 'incident_type', 'incident_severity',
       'authorities_contacted', 'incident_hour_of_the_day',
       'number_of_vehicles_involved', 'injury_claim', 'property_claim',
       'vehicle_claim', 'fraud_reported'],
      dtype='object')
```

[ ]:
```python
# Create a dataframe with the features
df_features = pd.DataFrame({'feature':X.columns, 'rank':rfe.ranking_})

# Print the top 10 features
print(df_features.sort_values('rank').head(10))
```

```
                        feature  rank
16            incident_severity     1
28                vehicle_claim     1
27               property_claim     1
26                 injury_claim     1
21  number_of_vehicles_involved     1
20     incident_hour_of_the_day     1
```

```
17        authorities_contacted      1
14                incident_type      1
10              insured_hobbies      1
32               fraud_reported      1
```

[ ]:
```python
# Run the model with the selected features

# Load the data and split into train and test sets

# Select the features from the RFE
X = df_indexed[X.columns[rfe.support_]]
y = df_indexed['total_claim_amount']

# Scale the predictor variables
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.4,␣
 ↪random_state=42)

# Create a gamma regression model
gamma_model = GammaRegressor()

# Fit the model on the training data
gamma_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = gamma_model.predict(X_test)
```

[ ]:
```python
# Calculate the Model Performance Metrics
print('R2 Score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))
print('MSE:', mean_squared_error(y_test, y_pred))

# Create a function to calculate the adjusted R2
def adj_r2(X,y):
    r2 = gamma_model.score(X,y)
    n = X.shape[0]
    p = X.shape[1]
    adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
    return adjusted_r2

# Calculate the adjusted R2
print('Adjusted R2:', adj_r2(X_test, y_test))
```

```
R2 Score: 0.8375504064524888
MAE: 8829.271884314403
```

```
MSE: 110338738.60385936
Adjusted R2: 0.6640010561580645
```

The RFE model shows a slight improvement compared to the original model, with a higher R2 score, lower MAE, and lower MSE. Additionally, the adjusted R2 score increased from 0.64 to 0.66, which suggests that the selected features better explain the variation in the response variable.

Overall, the RFE feature selection method was able to identify a subset of features that are more relevant for predicting the total claim amount, resulting in a slightly more accurate model.

6 Count Data - Poisson Regression for Amount of Vehicles Involved in the Claim

The purpose of the following section is to generate a prediction model for the amount of vehicules involved in the claims. The model will be based on a Poisson Regression, which is a generalized linear model (GLM) for predicting count data.

By using a Poisson model to predict the number of vehicles involved in an accident claim, the insurance company can estimate the expected number of vehicles involved in a claim and assess the associated risk and potential financial impact. This information can be used to adjust premiums, determine appropriate reserves for future claims, and identify areas for risk mitigation.

Additionally, the insurance company can use the Poisson model to analyze the impact of different variables on the number of vehicles involved in a claim, such as the policyholder's age, sex, education level, occupation, or location. This information can be used to identify high-risk policyholders or regions and develop targeted risk management strategies.

[ ]:
```
# Describe the number of vehicle involded
df['number_of_vehicles_involved'].describe()
```

[ ]:
```
count    1000.00000
mean        1.83900
std         1.01888
min         1.00000
25%         1.00000
50%         1.00000
75%         3.00000
max         4.00000
Name: number_of_vehicles_involved, dtype: float64
```

[ ]:
```
from sklearn.linear_model import PoissonRegressor
```

[ ]:
```
# Fit a Poisson Model to Predict the Amount of vehicules involved

# Load the data and split into train and test sets
X = df_indexed.drop('number_of_vehicles_involved', axis=1)
y = df_indexed['number_of_vehicles_involved']

# Split the data into train and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.40,␣
↪random_state=42)

# Scale the predictor variables
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Fit a Poisson regression model on the training data
poisson_model = PoissonRegressor()
poisson_model.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred = poisson_model.predict(X_test_scaled)
```

[ ]:
```
# Evaluate the model performance
print('R2 Score:', r2_score(y_test, y_pred))
print('MAE:', mean_absolute_error(y_test, y_pred))
print('MSE:', mean_squared_error(y_test, y_pred))

# Create a funciton to calculate the adjusted R2
def adj_r2(X,y):
    r2 = poisson_model.score(X,y)
    n = X.shape[0]
    p = X.shape[1]
    adjusted_r2 = 1-(1-r2)*(n-1)/(n-p-1)
    return adjusted_r2

# Calculate the adjusted R2
print('Adjusted R2:', adj_r2(X_test_scaled, y_test))
```

```
R2 Score: 0.7227489970960406
MAE: 0.47486946204652375
MSE: 0.2938427426089932
Adjusted R2: 0.7168411325544817
```

The R2 score of 0.72 indicates that the Poisson model has a moderate level of fit and is able to explain around 72% of the variance in the number of vehicles involved in a claim.

The MAE value of 0.47 suggests that the model is able to predict the number of vehicles involved with an average error of 0.47, which may or may not be acceptable depending on the context of the problem.

The MSE value of 0.29 suggests that the model's predictions have a moderate level of variance, which means that there is some amount of variability in the predictions around the true values.

The adjusted R2 value of 0.71 indicates that the model has a good level of fit, taking into account the number of predictors used in the model. Overall, the model seems to have a moderate level of fit, but it may benefit from further refinement or the inclusion of additional predictors.

```
[ ]: # Predict the number of vehicles involved in an accident for a random␣
     ↪observation
     print('Predicted number of vehicles involved:', poisson_model.
     ↪predict(X_test_scaled[0].reshape(1,-1))[0])
     print('Actual number of vehicles involved:', y_test.iloc[0])
```

```
Predicted number of vehicles involved: 1.4452792352608677
Actual number of vehicles involved: 1
```

```
[ ]: # Calculate the probability of accidents involving 1, 2, 3, and 4 vehicles

     from scipy.stats import poisson

     # Create a function to calculate the probability of accidents involving 1, 2,␣
     ↪3, and 4 vehicles given the model, and returning a single percentage per␣
     ↪number of vehicles
     def prob_vehicles(model, X, num_vehicles):
         # Convert X to a DataFrame
         X_df = pd.DataFrame(X, columns=X_test.columns)
         # Create a dataframe with the features
         df_features = pd.DataFrame({'feature':X_df.columns, 'coef':model.coef_})
         # Create a dataframe with the features and their exponential
         df_features['exp_coef'] = df_features['coef'].apply(lambda x: np.exp(x))
         # Create a dataframe with the features and their exponential multiplied by␣
     ↪the number of vehicles
         df_features['exp_coef_num_vehicles'] = df_features['exp_coef'].apply(lambda␣
     ↪x: x**num_vehicles)
         # Calculate the probability of accidents involving 1, 2, 3, and 4 vehicles
         prob = df_features['exp_coef_num_vehicles'].prod()
         return prob

     # Calculate the probability of accidents involving 1, 2, 3, and 4 vehicles
     print('Probability of accidents involving 1 vehicle:',␣
     ↪prob_vehicles(poisson_model, X_test_scaled[0].reshape(1,-1), 1))
     print('Probability of accidents involving 2 vehicles:',␣
     ↪prob_vehicles(poisson_model, X_test_scaled[0].reshape(1,-1), 2))
     print('Probability of accidents involving 3 vehicles:',␣
     ↪prob_vehicles(poisson_model, X_test_scaled[0].reshape(1,-1), 3))
     print('Probability of accidents involving 4 vehicles:',␣
     ↪prob_vehicles(poisson_model, X_test_scaled[0].reshape(1,-1), 4))
```

```
Probability of accidents involving 1 vehicle: 0.7766158989555773
Probability of accidents involving 2 vehicles: 0.6031322545105795
Probability of accidents involving 3 vehicles: 0.46840209802583743
Probability of accidents involving 4 vehicles: 0.36376851643101427
```

The results suggest that the probability of an accident involving 1 vehicle is the highest among the four options, with a probability of approximately 0.78. The probability decreases as the number of vehicles involved in the accident increases, with a probability of approximately 0.60 for accidents involving 2 vehicles, approximately 0.47 for accidents involving 3 vehicles, and approximately 0.36 for accidents involving 4 vehicles.

These probabilities may provide insights for insurance companies and policy makers when assessing risk and designing policies related to auto accidents. It may be helpful to investigate the causes of accidents involving multiple vehicles and to determine ways to mitigate these risks, such as improving road infrastructure and safety measures.

7 Neural Network for Fraud Prediction

The following section is focused on generating a prediction model for the fraud probability. The model will be based on a Neural Network Model, which is a machine learning model that can be used to predict the probability of an event occurring.

As a business problem, a machine learning model can help insurance companies to detect and prevent fraud more accurately and efficiently, reducing financial losses and maintaining lower premiums for policyholders. By analyzing historical data and identifying patterns indicative of fraud, the model can prioritize investigations and allocate resources more effectively, improving overall risk management for the company.

[ ]:
```
# Define the dependent and independent variables
X = df.drop('fraud_reported', axis=1)
y = df['fraud_reported']
```

[ ]:
```
# Drop fraud_reported from the cat_cols list
cat_cols=[i for i in cat_cols if i not in ['fraud_reported']]
```

[ ]:
```
# Get the dummy variables for the categorical columns
X=pd.get_dummies(X,columns=cat_cols,drop_first=True)
```

[ ]:
```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,␣
 ↪random_state=52)
```

[ ]:
```
# Fit a neural network model to predict fraud_reported using Keras

# Import the required libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping

# Create a function to create a neural network model
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=X_train.shape[1], activation='relu'))
```

```
    model.add(Dense(8, activation='relu'))
    model.add(Dense(4, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # Compile model
    adam=Adam(lr=0.0001)
    model.compile(loss='binary_crossentropy', optimizer=adam,␣
 ↪metrics=['accuracy'])
    return model

# Create a model
model = create_model()

# Print the model summary
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                1584

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 4)                 36

 dense_3 (Dense)             (None, 1)                 5

=================================================================
Total params: 1,761
Trainable params: 1,761
Non-trainable params: 0
_________________________________________________________________
None
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                1584

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 4)                 36

 dense_3 (Dense)             (None, 1)                 5

=================================================================
Total params: 1,761
Trainable params: 1,761
```

```
Non-trainable params: 0
_________________________________________________________________
None
```

[ ]:
```
# Fit the model
model.fit(X_train, y_train, epochs=200, batch_size=20, verbose=1)
```

```
Epoch 1/200
25/25 [==============================] - 0s 899us/step - loss: 6947.1743 -
accuracy: 0.6580
Epoch 2/200
25/25 [==============================] - 0s 898us/step - loss: 2474.5608 -
accuracy: 0.6360
Epoch 3/200
25/25 [==============================] - 0s 965us/step - loss: 908.8033 -
accuracy: 0.5800
Epoch 4/200
25/25 [==============================] - 0s 792us/step - loss: 745.7484 -
accuracy: 0.5840
Epoch 5/200
25/25 [==============================] - 0s 815us/step - loss: 552.3362 -
accuracy: 0.5800
Epoch 6/200
25/25 [==============================] - 0s 836us/step - loss: 541.5812 -
accuracy: 0.5740
Epoch 7/200
25/25 [==============================] - 0s 878us/step - loss: 586.6372 -
accuracy: 0.5820
Epoch 8/200
25/25 [==============================] - 0s 881us/step - loss: 542.7558 -
accuracy: 0.5820
Epoch 9/200
25/25 [==============================] - 0s 819us/step - loss: 456.3873 -
accuracy: 0.6080
Epoch 10/200
25/25 [==============================] - 0s 837us/step - loss: 492.1421 -
accuracy: 0.6000
Epoch 11/200
25/25 [==============================] - 0s 836us/step - loss: 519.8999 -
accuracy: 0.6120
Epoch 12/200
25/25 [==============================] - 0s 794us/step - loss: 472.0103 -
accuracy: 0.6100
Epoch 13/200
25/25 [==============================] - 0s 877us/step - loss: 398.6445 -
accuracy: 0.6040
Epoch 14/200
25/25 [==============================] - 0s 836us/step - loss: 380.9908 -
```

```
accuracy: 0.6020
Epoch 15/200
25/25 [==============================] - 0s 839us/step - loss: 392.1096 -
accuracy: 0.6140
Epoch 16/200
25/25 [==============================] - 0s 814us/step - loss: 398.7802 -
accuracy: 0.6080
Epoch 17/200
25/25 [==============================] - 0s 1ms/step - loss: 381.8946 -
accuracy: 0.6140
Epoch 18/200
25/25 [==============================] - 0s 1ms/step - loss: 367.1260 -
accuracy: 0.5940
Epoch 19/200
25/25 [==============================] - 0s 1ms/step - loss: 336.5479 -
accuracy: 0.6120
Epoch 20/200
25/25 [==============================] - 0s 943us/step - loss: 402.0947 -
accuracy: 0.6280
Epoch 21/200
25/25 [==============================] - 0s 877us/step - loss: 322.5425 -
accuracy: 0.6160
Epoch 22/200
25/25 [==============================] - 0s 814us/step - loss: 343.7301 -
accuracy: 0.6160
Epoch 23/200
25/25 [==============================] - 0s 879us/step - loss: 316.4546 -
accuracy: 0.6160
Epoch 24/200
25/25 [==============================] - 0s 835us/step - loss: 318.9844 -
accuracy: 0.6160
Epoch 25/200
25/25 [==============================] - 0s 826us/step - loss: 250.8297 -
accuracy: 0.6200
Epoch 26/200
25/25 [==============================] - 0s 818us/step - loss: 267.4481 -
accuracy: 0.6020
Epoch 27/200
25/25 [==============================] - 0s 814us/step - loss: 336.0552 -
accuracy: 0.6280
Epoch 28/200
25/25 [==============================] - 0s 857us/step - loss: 250.4970 -
accuracy: 0.6120
Epoch 29/200
25/25 [==============================] - 0s 835us/step - loss: 280.1104 -
accuracy: 0.6100
Epoch 30/200
25/25 [==============================] - 0s 836us/step - loss: 237.4940 -
```

```
accuracy: 0.6260
Epoch 31/200
25/25 [==============================] - 0s 815us/step - loss: 284.4987 -
accuracy: 0.5940
Epoch 32/200
25/25 [==============================] - 0s 839us/step - loss: 399.9630 -
accuracy: 0.6040
Epoch 33/200
25/25 [==============================] - 0s 846us/step - loss: 280.7903 -
accuracy: 0.6120
Epoch 34/200
25/25 [==============================] - 0s 1ms/step - loss: 200.7180 -
accuracy: 0.5940
Epoch 35/200
25/25 [==============================] - 0s 981us/step - loss: 267.8877 -
accuracy: 0.6000
Epoch 36/200
25/25 [==============================] - 0s 1ms/step - loss: 236.3749 -
accuracy: 0.5960
Epoch 37/200
25/25 [==============================] - 0s 865us/step - loss: 168.1265 -
accuracy: 0.5800
Epoch 38/200
25/25 [==============================] - 0s 841us/step - loss: 202.3585 -
accuracy: 0.5640
Epoch 39/200
25/25 [==============================] - 0s 876us/step - loss: 141.2833 -
accuracy: 0.5520
Epoch 40/200
25/25 [==============================] - 0s 835us/step - loss: 143.3056 -
accuracy: 0.5480
Epoch 41/200
25/25 [==============================] - 0s 898us/step - loss: 146.7035 -
accuracy: 0.5360
Epoch 42/200
25/25 [==============================] - 0s 815us/step - loss: 163.3881 -
accuracy: 0.5700
Epoch 43/200
25/25 [==============================] - 0s 796us/step - loss: 143.7869 -
accuracy: 0.5840
Epoch 44/200
25/25 [==============================] - 0s 839us/step - loss: 247.4911 -
accuracy: 0.5560
Epoch 45/200
25/25 [==============================] - 0s 877us/step - loss: 208.7947 -
accuracy: 0.5760
Epoch 46/200
25/25 [==============================] - 0s 856us/step - loss: 126.1005 -
```

```
accuracy: 0.6100
Epoch 47/200
25/25 [==============================] - 0s 793us/step - loss: 141.9734 -
accuracy: 0.6080
Epoch 48/200
25/25 [==============================] - 0s 983us/step - loss: 144.8161 -
accuracy: 0.6200
Epoch 49/200
25/25 [==============================] - 0s 1ms/step - loss: 125.4669 -
accuracy: 0.6080
Epoch 50/200
25/25 [==============================] - 0s 945us/step - loss: 152.8707 -
accuracy: 0.6160
Epoch 51/200
25/25 [==============================] - 0s 814us/step - loss: 129.0500 -
accuracy: 0.6420
Epoch 52/200
25/25 [==============================] - 0s 814us/step - loss: 72.6257 -
accuracy: 0.6340
Epoch 53/200
25/25 [==============================] - 0s 835us/step - loss: 98.1095 -
accuracy: 0.6400
Epoch 54/200
25/25 [==============================] - 0s 796us/step - loss: 84.0623 -
accuracy: 0.6340
Epoch 55/200
25/25 [==============================] - 0s 815us/step - loss: 56.1537 -
accuracy: 0.6460
Epoch 56/200
25/25 [==============================] - 0s 796us/step - loss: 91.9431 -
accuracy: 0.6440
Epoch 57/200
25/25 [==============================] - 0s 814us/step - loss: 81.1964 -
accuracy: 0.6440
Epoch 58/200
25/25 [==============================] - 0s 814us/step - loss: 101.5889 -
accuracy: 0.6460
Epoch 59/200
25/25 [==============================] - 0s 814us/step - loss: 91.1583 -
accuracy: 0.6640
Epoch 60/200
25/25 [==============================] - 0s 794us/step - loss: 55.9413 -
accuracy: 0.6360
Epoch 61/200
25/25 [==============================] - 0s 797us/step - loss: 139.1858 -
accuracy: 0.6600
Epoch 62/200
25/25 [==============================] - 0s 796us/step - loss: 82.1207 -
```

```
accuracy: 0.6520
Epoch 63/200
25/25 [==============================] - 0s 982us/step - loss: 103.8644 -
accuracy: 0.6680
Epoch 64/200
25/25 [==============================] - 0s 1ms/step - loss: 134.1967 -
accuracy: 0.6680
Epoch 65/200
25/25 [==============================] - 0s 981us/step - loss: 53.8054 -
accuracy: 0.6740
Epoch 66/200
25/25 [==============================] - 0s 794us/step - loss: 54.0155 -
accuracy: 0.6880
Epoch 67/200
25/25 [==============================] - 0s 798us/step - loss: 193.8517 -
accuracy: 0.6700
Epoch 68/200
25/25 [==============================] - 0s 794us/step - loss: 60.0211 -
accuracy: 0.6900
Epoch 69/200
25/25 [==============================] - 0s 814us/step - loss: 77.5990 -
accuracy: 0.6780
Epoch 70/200
25/25 [==============================] - 0s 835us/step - loss: 55.0239 -
accuracy: 0.6660
Epoch 71/200
25/25 [==============================] - 0s 791us/step - loss: 62.3729 -
accuracy: 0.6720
Epoch 72/200
25/25 [==============================] - 0s 797us/step - loss: 104.9686 -
accuracy: 0.6620
Epoch 73/200
25/25 [==============================] - 0s 796us/step - loss: 68.0541 -
accuracy: 0.6760
Epoch 74/200
25/25 [==============================] - 0s 775us/step - loss: 165.1224 -
accuracy: 0.6880
Epoch 75/200
25/25 [==============================] - 0s 796us/step - loss: 86.6320 -
accuracy: 0.6660
Epoch 76/200
25/25 [==============================] - 0s 773us/step - loss: 101.0446 -
accuracy: 0.6580
Epoch 77/200
25/25 [==============================] - 0s 817us/step - loss: 55.4335 -
accuracy: 0.6880
Epoch 78/200
25/25 [==============================] - 0s 795us/step - loss: 56.9533 -
```

```
accuracy: 0.6880
Epoch 79/200
25/25 [==============================] - 0s 1ms/step - loss: 101.0332 -
accuracy: 0.6740
Epoch 80/200
25/25 [==============================] - 0s 1ms/step - loss: 100.4065 -
accuracy: 0.6820
Epoch 81/200
25/25 [==============================] - 0s 1ms/step - loss: 102.4389 -
accuracy: 0.6680
Epoch 82/200
25/25 [==============================] - 0s 1ms/step - loss: 82.6608 - accuracy:
0.6920
Epoch 83/200
25/25 [==============================] - 0s 836us/step - loss: 84.9176 -
accuracy: 0.6760
Epoch 84/200
25/25 [==============================] - 0s 836us/step - loss: 84.8463 -
accuracy: 0.6840
Epoch 85/200
25/25 [==============================] - 0s 800us/step - loss: 65.7322 -
accuracy: 0.6760
Epoch 86/200
25/25 [==============================] - 0s 794us/step - loss: 58.2357 -
accuracy: 0.6760
Epoch 87/200
25/25 [==============================] - 0s 793us/step - loss: 96.6498 -
accuracy: 0.6780
Epoch 88/200
25/25 [==============================] - 0s 856us/step - loss: 121.9459 -
accuracy: 0.6800
Epoch 89/200
25/25 [==============================] - 0s 841us/step - loss: 55.6262 -
accuracy: 0.6780
Epoch 90/200
25/25 [==============================] - 0s 793us/step - loss: 68.2254 -
accuracy: 0.6600
Epoch 91/200
25/25 [==============================] - 0s 835us/step - loss: 53.3755 -
accuracy: 0.6880
Epoch 92/200
25/25 [==============================] - 0s 814us/step - loss: 46.2688 -
accuracy: 0.6880
Epoch 93/200
25/25 [==============================] - 0s 795us/step - loss: 46.6621 -
accuracy: 0.6900
Epoch 94/200
25/25 [==============================] - 0s 962us/step - loss: 57.2078 -
```

```
accuracy: 0.6860
Epoch 95/200
25/25 [==============================] - 0s 1ms/step - loss: 49.6593 - accuracy:
0.6960
Epoch 96/200
25/25 [==============================] - 0s 905us/step - loss: 80.0890 -
accuracy: 0.6880
Epoch 97/200
25/25 [==============================] - 0s 794us/step - loss: 70.6270 -
accuracy: 0.6700
Epoch 98/200
25/25 [==============================] - 0s 835us/step - loss: 69.6082 -
accuracy: 0.6840
Epoch 99/200
25/25 [==============================] - 0s 794us/step - loss: 67.1859 -
accuracy: 0.6840
Epoch 100/200
25/25 [==============================] - 0s 773us/step - loss: 64.1913 -
accuracy: 0.6800
Epoch 101/200
25/25 [==============================] - 0s 793us/step - loss: 71.5268 -
accuracy: 0.6800
Epoch 102/200
25/25 [==============================] - 0s 772us/step - loss: 71.3787 -
accuracy: 0.6920
Epoch 103/200
25/25 [==============================] - 0s 815us/step - loss: 50.0667 -
accuracy: 0.6840
Epoch 104/200
25/25 [==============================] - 0s 815us/step - loss: 58.8049 -
accuracy: 0.6880
Epoch 105/200
25/25 [==============================] - 0s 795us/step - loss: 64.7363 -
accuracy: 0.6640
Epoch 106/200
25/25 [==============================] - 0s 794us/step - loss: 88.3265 -
accuracy: 0.7020
Epoch 107/200
25/25 [==============================] - 0s 835us/step - loss: 97.6908 -
accuracy: 0.7000
Epoch 108/200
25/25 [==============================] - 0s 772us/step - loss: 54.4678 -
accuracy: 0.6980
Epoch 109/200
25/25 [==============================] - 0s 793us/step - loss: 58.5719 -
accuracy: 0.6960
Epoch 110/200
25/25 [==============================] - 0s 1ms/step - loss: 69.8249 - accuracy:
```

```
0.7120
Epoch 111/200
25/25 [==============================] - 0s 1ms/step - loss: 86.3784 - accuracy:
0.7080
Epoch 112/200
25/25 [==============================] - 0s 1ms/step - loss: 42.0706 - accuracy:
0.7000
Epoch 113/200
25/25 [==============================] - 0s 794us/step - loss: 35.6461 -
accuracy: 0.7080
Epoch 114/200
25/25 [==============================] - 0s 793us/step - loss: 44.5446 -
accuracy: 0.6740
Epoch 115/200
25/25 [==============================] - 0s 793us/step - loss: 53.1757 -
accuracy: 0.6720
Epoch 116/200
25/25 [==============================] - 0s 793us/step - loss: 31.8403 -
accuracy: 0.7100
Epoch 117/200
25/25 [==============================] - 0s 794us/step - loss: 63.2532 -
accuracy: 0.6660
Epoch 118/200
25/25 [==============================] - 0s 799us/step - loss: 51.0646 -
accuracy: 0.6920
Epoch 119/200
25/25 [==============================] - 0s 818us/step - loss: 159.9407 -
accuracy: 0.6780
Epoch 120/200
25/25 [==============================] - 0s 792us/step - loss: 102.6931 -
accuracy: 0.6980
Epoch 121/200
25/25 [==============================] - 0s 834us/step - loss: 26.7488 -
accuracy: 0.7080
Epoch 122/200
25/25 [==============================] - 0s 794us/step - loss: 48.9884 -
accuracy: 0.6980
Epoch 123/200
25/25 [==============================] - 0s 794us/step - loss: 69.0593 -
accuracy: 0.6960
Epoch 124/200
25/25 [==============================] - 0s 2ms/step - loss: 80.7219 - accuracy:
0.6860
Epoch 125/200
25/25 [==============================] - 0s 954us/step - loss: 28.2026 -
accuracy: 0.7060
Epoch 126/200
25/25 [==============================] - 0s 882us/step - loss: 34.3212 -
```

```
accuracy: 0.7160
Epoch 127/200
25/25 [==============================] - 0s 813us/step - loss: 43.6475 -
accuracy: 0.6760
Epoch 128/200
25/25 [==============================] - 0s 814us/step - loss: 86.1366 -
accuracy: 0.6980
Epoch 129/200
25/25 [==============================] - 0s 776us/step - loss: 106.6821 -
accuracy: 0.6880
Epoch 130/200
25/25 [==============================] - 0s 816us/step - loss: 171.6767 -
accuracy: 0.6700
Epoch 131/200
25/25 [==============================] - 0s 798us/step - loss: 114.7948 -
accuracy: 0.7080
Epoch 132/200
25/25 [==============================] - 0s 793us/step - loss: 130.5965 -
accuracy: 0.6880
Epoch 133/200
25/25 [==============================] - 0s 772us/step - loss: 39.1164 -
accuracy: 0.7100
Epoch 134/200
25/25 [==============================] - 0s 792us/step - loss: 57.6070 -
accuracy: 0.7100
Epoch 135/200
25/25 [==============================] - 0s 793us/step - loss: 37.7200 -
accuracy: 0.7060
Epoch 136/200
25/25 [==============================] - 0s 815us/step - loss: 57.3084 -
accuracy: 0.7060
Epoch 137/200
25/25 [==============================] - 0s 1ms/step - loss: 79.1852 - accuracy:
0.7040
Epoch 138/200
25/25 [==============================] - 0s 1ms/step - loss: 76.8136 - accuracy:
0.6980
Epoch 139/200
25/25 [==============================] - 0s 960us/step - loss: 74.8812 -
accuracy: 0.7060
Epoch 140/200
25/25 [==============================] - 0s 814us/step - loss: 51.2769 -
accuracy: 0.7160
Epoch 141/200
25/25 [==============================] - 0s 815us/step - loss: 123.5782 -
accuracy: 0.6940
Epoch 142/200
25/25 [==============================] - 0s 772us/step - loss: 97.9099 -
```

```
accuracy: 0.6900
Epoch 143/200
25/25 [==============================] - 0s 813us/step - loss: 100.2137 -
accuracy: 0.7260
Epoch 144/200
25/25 [==============================] - 0s 818us/step - loss: 163.4574 -
accuracy: 0.6940
Epoch 145/200
25/25 [==============================] - 0s 836us/step - loss: 262.9234 -
accuracy: 0.6940
Epoch 146/200
25/25 [==============================] - 0s 989us/step - loss: 190.6122 -
accuracy: 0.7120
Epoch 147/200
25/25 [==============================] - 0s 793us/step - loss: 50.9315 -
accuracy: 0.7060
Epoch 148/200
25/25 [==============================] - 0s 792us/step - loss: 93.2805 -
accuracy: 0.6980
Epoch 149/200
25/25 [==============================] - 0s 794us/step - loss: 61.3404 -
accuracy: 0.7280
Epoch 150/200
25/25 [==============================] - 0s 753us/step - loss: 47.1367 -
accuracy: 0.6960
Epoch 151/200
25/25 [==============================] - 0s 838us/step - loss: 128.6801 -
accuracy: 0.7220
Epoch 152/200
25/25 [==============================] - 0s 1ms/step - loss: 103.4937 -
accuracy: 0.7220
Epoch 153/200
25/25 [==============================] - 0s 1ms/step - loss: 72.6324 - accuracy:
0.7020
Epoch 154/200
25/25 [==============================] - 0s 920us/step - loss: 53.5139 -
accuracy: 0.7120
Epoch 155/200
25/25 [==============================] - 0s 773us/step - loss: 69.2190 -
accuracy: 0.7080
Epoch 156/200
25/25 [==============================] - 0s 772us/step - loss: 79.1296 -
accuracy: 0.7080
Epoch 157/200
25/25 [==============================] - 0s 781us/step - loss: 48.9102 -
accuracy: 0.7020
Epoch 158/200
25/25 [==============================] - 0s 817us/step - loss: 31.8400 -
```

```
accuracy: 0.7260
Epoch 159/200
25/25 [==============================] - 0s 756us/step - loss: 30.7672 -
accuracy: 0.7280
Epoch 160/200
25/25 [==============================] - 0s 813us/step - loss: 57.0757 -
accuracy: 0.7260
Epoch 161/200
25/25 [==============================] - 0s 836us/step - loss: 86.2038 -
accuracy: 0.7220
Epoch 162/200
25/25 [==============================] - 0s 794us/step - loss: 44.8307 -
accuracy: 0.7140
Epoch 163/200
25/25 [==============================] - 0s 794us/step - loss: 29.2588 -
accuracy: 0.7300
Epoch 164/200
25/25 [==============================] - 0s 773us/step - loss: 54.9562 -
accuracy: 0.7200
Epoch 165/200
25/25 [==============================] - 0s 796us/step - loss: 49.0006 -
accuracy: 0.7300
Epoch 166/200
25/25 [==============================] - 0s 1ms/step - loss: 64.4708 - accuracy:
0.7020
Epoch 167/200
25/25 [==============================] - 0s 983us/step - loss: 23.1144 -
accuracy: 0.7280
Epoch 168/200
25/25 [==============================] - 0s 860us/step - loss: 55.4248 -
accuracy: 0.7280
Epoch 169/200
25/25 [==============================] - 0s 798us/step - loss: 49.9142 -
accuracy: 0.7220
Epoch 170/200
25/25 [==============================] - 0s 793us/step - loss: 83.9214 -
accuracy: 0.7120
Epoch 171/200
25/25 [==============================] - 0s 794us/step - loss: 107.3120 -
accuracy: 0.7240
Epoch 172/200
25/25 [==============================] - 0s 795us/step - loss: 40.4449 -
accuracy: 0.7240
Epoch 173/200
25/25 [==============================] - 0s 795us/step - loss: 64.2503 -
accuracy: 0.7020
Epoch 174/200
25/25 [==============================] - 0s 781us/step - loss: 144.2501 -
```

```
accuracy: 0.7080
Epoch 175/200
25/25 [==============================] - 0s 751us/step - loss: 46.9898 -
accuracy: 0.7220
Epoch 176/200
25/25 [==============================] - 0s 837us/step - loss: 21.0693 -
accuracy: 0.7380
Epoch 177/200
25/25 [==============================] - 0s 796us/step - loss: 41.0686 -
accuracy: 0.7260
Epoch 178/200
25/25 [==============================] - 0s 752us/step - loss: 69.5288 -
accuracy: 0.7120
Epoch 179/200
25/25 [==============================] - 0s 877us/step - loss: 91.4239 -
accuracy: 0.7020
Epoch 180/200
25/25 [==============================] - 0s 2ms/step - loss: 177.0196 -
accuracy: 0.7360
Epoch 181/200
25/25 [==============================] - 0s 965us/step - loss: 114.6377 -
accuracy: 0.7100
Epoch 182/200
25/25 [==============================] - 0s 881us/step - loss: 103.1633 -
accuracy: 0.7140
Epoch 183/200
25/25 [==============================] - 0s 796us/step - loss: 37.5359 -
accuracy: 0.7360
Epoch 184/200
25/25 [==============================] - 0s 815us/step - loss: 38.0279 -
accuracy: 0.7200
Epoch 185/200
25/25 [==============================] - 0s 794us/step - loss: 47.0279 -
accuracy: 0.7180
Epoch 186/200
25/25 [==============================] - 0s 836us/step - loss: 52.1423 -
accuracy: 0.7260
Epoch 187/200
25/25 [==============================] - 0s 779us/step - loss: 36.5526 -
accuracy: 0.7240
Epoch 188/200
25/25 [==============================] - 0s 835us/step - loss: 41.8313 -
accuracy: 0.7340
Epoch 189/200
25/25 [==============================] - 0s 818us/step - loss: 47.0684 -
accuracy: 0.7220
Epoch 190/200
25/25 [==============================] - 0s 793us/step - loss: 28.6111 -
```

```
accuracy: 0.7300
Epoch 191/200
25/25 [==============================] - 0s 800us/step - loss: 59.4901 -
accuracy: 0.7340
Epoch 192/200
25/25 [==============================] - 0s 794us/step - loss: 95.7343 -
accuracy: 0.7220
Epoch 193/200
25/25 [==============================] - 0s 941us/step - loss: 84.1668 -
accuracy: 0.7180
Epoch 194/200
25/25 [==============================] - 0s 1ms/step - loss: 34.4738 - accuracy:
0.7420
Epoch 195/200
25/25 [==============================] - 0s 901us/step - loss: 128.3673 -
accuracy: 0.7360
Epoch 196/200
25/25 [==============================] - 0s 814us/step - loss: 141.1569 -
accuracy: 0.7020
Epoch 197/200
25/25 [==============================] - 0s 773us/step - loss: 87.6433 -
accuracy: 0.7300
Epoch 198/200
25/25 [==============================] - 0s 796us/step - loss: 42.1844 -
accuracy: 0.7200
Epoch 199/200
25/25 [==============================] - 0s 836us/step - loss: 34.2162 -
accuracy: 0.7240
Epoch 200/200
25/25 [==============================] - 0s 798us/step - loss: 58.8181 -
accuracy: 0.7280
```

[ ]: <keras.callbacks.History at 0x145d278d550>

[ ]:
```
# Predict the model
y_pred = model.predict(X_test)
```

```
16/16 [==============================] - 0s 669us/step
16/16 [==============================] - 0s 669us/step
```

[ ]:
```
# Print the accuracy score
print(accuracy_score(y_test,y_pred.round()))
```

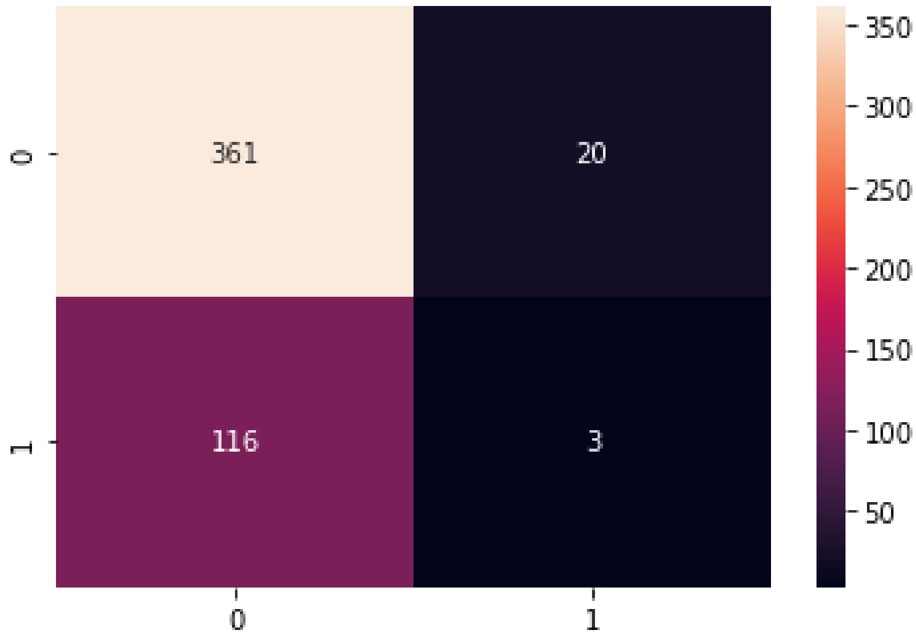
```
0.728
```

[ ]:
```
# Print the confusion matrix
print(confusion_matrix(y_test,y_pred.round()))
```

```
[[361  20]
```

```
 [116   3]]
```

[ ]:
```python
# Plot the confusion matrix
sns.heatmap(confusion_matrix(y_test,y_pred.round()),annot=True,fmt='d');
```



The model accuracy does not to seem that great, with only a score of 73% accuracy. Let's see if we can improve it by hyperparameter tuning.

[ ]:
```python
# Import required libraries
import numpy as np
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam

# Define a function to create a Keras model
def create_model(learning_rate=0.01, activation='relu'):
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=X_train.shape[1], activation=activation))
    model.add(Dense(8, activation=activation))
    model.add(Dense(4, activation=activation))
    model.add(Dense(1, activation='sigmoid'))

    # Compile model
```

```python
    optimizer = Adam(lr=learning_rate)
    model.compile(loss='binary_crossentropy', optimizer=optimizer,␣
↪metrics=['accuracy'])
    return model

# Define hyperparameters
param_grid = {'batch_size': [20, 40, 60, 80, 100],
                'epochs': [100, 200, 300, 400, 500],
                'learning_rate': [0.01, 0.001, 0.0001],
                'activation': ['relu', 'tanh']}

# Create a KerasClassifier object
model = KerasClassifier(build_fn=create_model)

# Create a GridSearchCV object
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)

# Fit the GridSearchCV object with the data
grid_result = grid.fit(X_train, y_train)

# Print the best score and best parameters
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Epoch 1/100
7/7 [==============================] - 1s 1ms/step - loss: 38319.8867 -
accuracy: 0.4680
Epoch 2/100
7/7 [==============================] - 0s 1ms/step - loss: 25369.8164 -
accuracy: 0.5080
Epoch 3/100
7/7 [==============================] - 0s 1ms/step - loss: 11500.5107 -
accuracy: 0.6280
Epoch 4/100
7/7 [==============================] - 0s 1ms/step - loss: 1643.5712 - accuracy:
0.7040
Epoch 5/100
7/7 [==============================] - 0s 1ms/step - loss: 2228.5325 - accuracy:
0.7300
Epoch 6/100
7/7 [==============================] - 0s 1ms/step - loss: 1689.7468 - accuracy:
0.7040
Epoch 7/100
7/7 [==============================] - 0s 1ms/step - loss: 500.6108 - accuracy:
0.5720
Epoch 8/100
7/7 [==============================] - 0s 1ms/step - loss: 483.5487 - accuracy:
0.6080
Epoch 9/100
```

```
7/7 [==============================] - 0s 1ms/step - loss: 464.6248 - accuracy:
0.6800
Epoch 10/100
7/7 [==============================] - 0s 1ms/step - loss: 476.2349 - accuracy:
0.6120
Epoch 11/100
7/7 [==============================] - 0s 1ms/step - loss: 337.5928 - accuracy:
0.6680
Epoch 12/100
7/7 [==============================] - 0s 1ms/step - loss: 271.8499 - accuracy:
0.6240
Epoch 13/100
7/7 [==============================] - 0s 1ms/step - loss: 182.0222 - accuracy:
0.6640
Epoch 14/100
7/7 [==============================] - 0s 2ms/step - loss: 173.7761 - accuracy:
0.6400
Epoch 15/100
7/7 [==============================] - 0s 2ms/step - loss: 143.0125 - accuracy:
0.6160
Epoch 16/100
7/7 [==============================] - 0s 2ms/step - loss: 214.9517 - accuracy:
0.6540
Epoch 17/100
7/7 [==============================] - 0s 1ms/step - loss: 162.1337 - accuracy:
0.6360
Epoch 18/100
7/7 [==============================] - 0s 1ms/step - loss: 130.4715 - accuracy:
0.6560
Epoch 19/100
7/7 [==============================] - 0s 1ms/step - loss: 182.2525 - accuracy:
0.6420
Epoch 20/100
7/7 [==============================] - 0s 1ms/step - loss: 189.3147 - accuracy:
0.6520
Epoch 21/100
7/7 [==============================] - 0s 1ms/step - loss: 171.3895 - accuracy:
0.6760
Epoch 22/100
7/7 [==============================] - 0s 2ms/step - loss: 184.2287 - accuracy:
0.6460
Epoch 23/100
7/7 [==============================] - 0s 1ms/step - loss: 133.2804 - accuracy:
0.6400
Epoch 24/100
7/7 [==============================] - 0s 1ms/step - loss: 126.1146 - accuracy:
0.6300
Epoch 25/100
```

```
7/7 [==============================] - 0s 1ms/step - loss: 227.6079 - accuracy:
0.6620
Epoch 26/100
7/7 [==============================] - 0s 1ms/step - loss: 105.3978 - accuracy:
0.6460
Epoch 27/100
7/7 [==============================] - 0s 1ms/step - loss: 118.8184 - accuracy:
0.6620
Epoch 28/100
7/7 [==============================] - 0s 1ms/step - loss: 117.8343 - accuracy:
0.5960
Epoch 29/100
7/7 [==============================] - 0s 1ms/step - loss: 189.8777 - accuracy:
0.6560
Epoch 30/100
7/7 [==============================] - 0s 1ms/step - loss: 221.6559 - accuracy:
0.6520
Epoch 31/100
7/7 [==============================] - 0s 1ms/step - loss: 141.7795 - accuracy:
0.6540
Epoch 32/100
7/7 [==============================] - 0s 1ms/step - loss: 98.8320 - accuracy:
0.6420
Epoch 33/100
7/7 [==============================] - 0s 2ms/step - loss: 98.3862 - accuracy:
0.6780
Epoch 34/100
7/7 [==============================] - 0s 2ms/step - loss: 99.5913 - accuracy:
0.6820
Epoch 35/100
7/7 [==============================] - 0s 2ms/step - loss: 126.5798 - accuracy:
0.6580
Epoch 36/100
7/7 [==============================] - 0s 1ms/step - loss: 167.7500 - accuracy:
0.6860
Epoch 37/100
7/7 [==============================] - 0s 1ms/step - loss: 288.4189 - accuracy:
0.6340
Epoch 38/100
7/7 [==============================] - 0s 2ms/step - loss: 646.2736 - accuracy:
0.6900
Epoch 39/100
7/7 [==============================] - 0s 2ms/step - loss: 125.7064 - accuracy:
0.6120
Epoch 40/100
7/7 [==============================] - 0s 1ms/step - loss: 174.7963 - accuracy:
0.6540
Epoch 41/100
```

```
7/7 [==============================] - 0s 1ms/step - loss: 138.9755 - accuracy:
0.6400
Epoch 42/100
7/7 [==============================] - 0s 1ms/step - loss: 193.0526 - accuracy:
0.7120
Epoch 43/100
7/7 [==============================] - 0s 1ms/step - loss: 207.9409 - accuracy:
0.6700
Epoch 44/100
7/7 [==============================] - 0s 1ms/step - loss: 63.1655 - accuracy:
0.6520
Epoch 45/100
7/7 [==============================] - 0s 1ms/step - loss: 91.2643 - accuracy:
0.6800
Epoch 46/100
7/7 [==============================] - 0s 1ms/step - loss: 55.4114 - accuracy:
0.6900
Epoch 47/100
7/7 [==============================] - 0s 1ms/step - loss: 130.1207 - accuracy:
0.6960
Epoch 48/100
7/7 [==============================] - 0s 1ms/step - loss: 81.7118 - accuracy:
0.7040
Epoch 49/100
7/7 [==============================] - 0s 1ms/step - loss: 123.6458 - accuracy:
0.6940
Epoch 50/100
7/7 [==============================] - 0s 1ms/step - loss: 133.5079 - accuracy:
0.6900
Epoch 51/100
7/7 [==============================] - 0s 1ms/step - loss: 92.7610 - accuracy:
0.6940
Epoch 52/100
7/7 [==============================] - 0s 1ms/step - loss: 87.0867 - accuracy:
0.6940
Epoch 53/100
7/7 [==============================] - 0s 2ms/step - loss: 148.1695 - accuracy:
0.7180
Epoch 54/100
7/7 [==============================] - 0s 2ms/step - loss: 85.7722 - accuracy:
0.6920
Epoch 55/100
7/7 [==============================] - 0s 2ms/step - loss: 54.7273 - accuracy:
0.7140
Epoch 56/100
7/7 [==============================] - 0s 1000us/step - loss: 72.1869 -
accuracy: 0.6840
Epoch 57/100
```

```
7/7 [==============================] - 0s 1ms/step - loss: 97.4154 - accuracy:
0.7120
Epoch 58/100
7/7 [==============================] - 0s 1000us/step - loss: 81.3191 -
accuracy: 0.7040
Epoch 59/100
7/7 [==============================] - 0s 1ms/step - loss: 91.3966 - accuracy:
0.7120
Epoch 60/100
7/7 [==============================] - 0s 1ms/step - loss: 40.3106 - accuracy:
0.7080
Epoch 61/100
7/7 [==============================] - 0s 1ms/step - loss: 39.9780 - accuracy:
0.7140
Epoch 62/100
7/7 [==============================] - 0s 1ms/step - loss: 57.8788 - accuracy:
0.7080
Epoch 63/100
7/7 [==============================] - 0s 1ms/step - loss: 164.5726 - accuracy:
0.7260
Epoch 64/100
7/7 [==============================] - 0s 1ms/step - loss: 126.4698 - accuracy:
0.6920
Epoch 65/100
7/7 [==============================] - 0s 1ms/step - loss: 177.5610 - accuracy:
0.7280
Epoch 66/100
7/7 [==============================] - 0s 1ms/step - loss: 205.7392 - accuracy:
0.6940
Epoch 67/100
7/7 [==============================] - 0s 1ms/step - loss: 60.7550 - accuracy:
0.6960
Epoch 68/100
7/7 [==============================] - 0s 1ms/step - loss: 75.2348 - accuracy:
0.6960
Epoch 69/100
7/7 [==============================] - 0s 1ms/step - loss: 76.4552 - accuracy:
0.6960
Epoch 70/100
7/7 [==============================] - 0s 2ms/step - loss: 128.4090 - accuracy:
0.7180
Epoch 71/100
7/7 [==============================] - 0s 1ms/step - loss: 92.6421 - accuracy:
0.7120
Epoch 72/100
7/7 [==============================] - 0s 1000us/step - loss: 106.2934 -
accuracy: 0.7000
Epoch 73/100
```

```
7/7 [==============================] - 0s 1ms/step - loss: 80.4447 - accuracy:
0.7140
Epoch 74/100
7/7 [==============================] - 0s 2ms/step - loss: 73.0985 - accuracy:
0.6860
Epoch 75/100
7/7 [==============================] - 0s 1ms/step - loss: 80.0589 - accuracy:
0.7140
Epoch 76/100
7/7 [==============================] - 0s 1ms/step - loss: 99.7002 - accuracy:
0.7120
Epoch 77/100
7/7 [==============================] - 0s 1ms/step - loss: 116.1769 - accuracy:
0.7380
Epoch 78/100
7/7 [==============================] - 0s 1ms/step - loss: 155.7042 - accuracy:
0.6820
Epoch 79/100
7/7 [==============================] - 0s 1ms/step - loss: 139.0850 - accuracy:
0.6960
Epoch 80/100
7/7 [==============================] - 0s 2ms/step - loss: 429.3281 - accuracy:
0.7400
Epoch 81/100
7/7 [==============================] - 0s 2ms/step - loss: 315.3177 - accuracy:
0.6220
Epoch 82/100
7/7 [==============================] - 0s 2ms/step - loss: 293.5067 - accuracy:
0.7460
Epoch 83/100
7/7 [==============================] - 0s 1ms/step - loss: 174.0258 - accuracy:
0.6800
Epoch 84/100
7/7 [==============================] - 0s 1ms/step - loss: 100.3365 - accuracy:
0.6720
Epoch 85/100
7/7 [==============================] - 0s 1ms/step - loss: 148.6979 - accuracy:
0.7000
Epoch 86/100
7/7 [==============================] - 0s 1ms/step - loss: 32.4058 - accuracy:
0.6780
Epoch 87/100
7/7 [==============================] - 0s 1ms/step - loss: 66.3409 - accuracy:
0.7200
Epoch 88/100
7/7 [==============================] - 0s 1ms/step - loss: 104.2426 - accuracy:
0.7280
Epoch 89/100
```

```
7/7 [==============================] - 0s 1ms/step - loss: 160.2844 - accuracy:
0.6800
Epoch 90/100
7/7 [==============================] - 0s 1ms/step - loss: 148.5122 - accuracy:
0.6760
Epoch 91/100
7/7 [==============================] - 0s 1ms/step - loss: 209.4482 - accuracy:
0.7480
Epoch 92/100
7/7 [==============================] - 0s 1ms/step - loss: 280.5800 - accuracy:
0.6860
Epoch 93/100
7/7 [==============================] - 0s 1ms/step - loss: 215.6759 - accuracy:
0.7180
Epoch 94/100
7/7 [==============================] - 0s 1ms/step - loss: 374.7006 - accuracy:
0.7380
Epoch 95/100
7/7 [==============================] - 0s 1ms/step - loss: 193.8938 - accuracy:
0.6100
Epoch 96/100
7/7 [==============================] - 0s 1ms/step - loss: 190.0665 - accuracy:
0.7280
Epoch 97/100
7/7 [==============================] - 0s 1ms/step - loss: 37.0257 - accuracy:
0.7080
Epoch 98/100
7/7 [==============================] - 0s 917us/step - loss: 131.2587 -
accuracy: 0.7180
Epoch 99/100
7/7 [==============================] - 0s 1ms/step - loss: 245.2310 - accuracy:
0.6780
Epoch 100/100
7/7 [==============================] - 0s 1ms/step - loss: 185.2444 - accuracy:
0.6940
Best: 0.754058 using {'activation': 'relu', 'batch_size': 80, 'epochs': 100,
'learning_rate': 0.001}
```

After model optimization we are able to improve accuracy from 73% to 75%, which is still an improvement. However, the model is still not very accurate, a good reason might be the imbalance of the data, which is something to keep in mind, but it's also the nature of fraud.