In [17]:
```
import pandas as pd
import numpy as np
import datetime as dt
import seaborn as sb
import plotly.express as px
import plotly.graph_objects as go

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import classification_report
from imblearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
```

In [ ]:

In [3]:
```
df = pd.read_csv('Automobile_insurance_fraud.csv')
df.head()
```

Out[3]:

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

5 rows × 40 columns

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
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         0
incident_state                0
incident_city                 0
incident_location             0
incident_hour_of_the_day      0
number_of_vehicles_involved   0
property_damage               0
bodily_injuries               0
witnesses                     0
police_report_available       0
total_claim_amount            0
injury_claim                  0
property_claim                0
vehicle_claim                 0
auto_make                     0
auto_model                    0
auto_year                     0
fraud_reported                0
_c39                       1000
dtype: int64
```

In [5]:
```
df = df.drop(['_c39'], axis = 1)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 39 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   months_as_customer           1000 non-null   int64
 1   age                          1000 non-null   int64
 2   policy_number                1000 non-null   int64
 3   policy_bind_date             1000 non-null   object
 4   policy_state                 1000 non-null   object
 5   policy_csl                   1000 non-null   object
 6   policy_deductable            1000 non-null   int64
 7   policy_annual_premium        1000 non-null   float64
 8   umbrella_limit               1000 non-null   int64
 9   insured_zip                  1000 non-null   int64
 10  insured_sex                  1000 non-null   object
 11  insured_education_level      1000 non-null   object
 12  insured_occupation           1000 non-null   object
 13  insured_hobbies              1000 non-null   object
 14  insured_relationship         1000 non-null   object
 15  capital-gains                1000 non-null   int64
 16  capital-loss                 1000 non-null   int64
 17  incident_date                1000 non-null   object
 18  incident_type                1000 non-null   object
 19  collision_type               1000 non-null   object
 20  incident_severity            1000 non-null   object
 21  authorities_contacted        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
dtypes: float64(1), int64(17), object(21)
memory usage: 304.8+ KB
```

In [7]:
```
df['policy_bind_date'] = pd.to_datetime(df['policy_bind_date'])
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_11400\1924804640.py:1: UserWarning: Parsing dates in DD/MM/YYYY format when dayfirst=False
(the default) was specified. This may lead to inconsistently parsed dates! Specify a format to ensure consistent parsing.
  df['policy_bind_date'] = pd.to_datetime(df['policy_bind_date'])
```

In [8]:
```
df.describe()
```

Out[8]:

	months_as_customer	age	policy_number	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	capital-gains	capital-loss	i
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1.000000e+03	1000.000000	1000.000000	1000.000000	
mean	203.954000	38.948000	546238.648000	1136.000000	1256.406150	1.101000e+06	501214.488000	25126.100000	-26793.700000	
std	115.113174	9.140287	257063.005276	611.864673	244.167395	2.297407e+06	71701.610941	27872.187708	28104.096686	
min	0.000000	19.000000	100804.000000	500.000000	433.330000	-1.000000e+06	430104.000000	0.000000	-111100.000000	
25%	115.750000	32.000000	335980.250000	500.000000	1089.607500	0.000000e+00	448404.500000	0.000000	-51500.000000	
50%	199.500000	38.000000	533135.000000	1000.000000	1257.200000	0.000000e+00	466445.500000	0.000000	-23250.000000	
75%	276.250000	44.000000	759099.750000	2000.000000	1415.695000	0.000000e+00	603251.000000	51025.000000	0.000000	
max	479.000000	64.000000	999435.000000	2000.000000	2047.590000	1.000000e+07	620962.000000	100500.000000	0.000000	

In [9]:
```
for i in df.columns:
    if df[i].dtype == 'object':
        print(i, ":", df[i].nunique())
```

```
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 : 4
incident_severity : 4
authorities_contacted : 5
incident_state : 7
incident_city : 7
incident_location : 1000
property_damage : 3
police_report_available : 3
auto_make : 14
auto_model : 39
fraud_reported : 2
```

In [10]:
```
drop_columns = ['policy_state', 'policy_csl', 'incident_date', 'incident_state', 'incident_city', 'incident_location']
df = df.drop(drop_columns, axis = 1)
df.head()
```

Out[10]:

	months_as_customer	age	policy_number	policy_bind_date	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insured_educatio
0	328	48	521585	2014-10-17	1000	1406.91	0	466132	MALE	
1	228	42	342868	2006-06-27	2000	1197.22	5000000	468176	MALE	
2	134	29	687698	2000-06-09	2000	1413.14	5000000	430632	FEMALE	
3	256	41	227811	1990-05-25	2000	1415.74	6000000	608117	FEMALE	
4	228	44	367455	2014-06-06	1000	1583.91	6000000	610706	MALE	As

5 rows × 33 columns

In [11]:
```
for i in df.columns:
    if df[i].dtype == 'object':
        print(i, ":", df[i].nunique())
```

```
insured_sex : 2
insured_education_level : 7
insured_occupation : 14
insured_hobbies : 20
insured_relationship : 6
incident_type : 4
collision_type : 4
incident_severity : 4
authorities_contacted : 5
property_damage : 3
police_report_available : 3
auto_make : 14
auto_model : 39
fraud_reported : 2
```

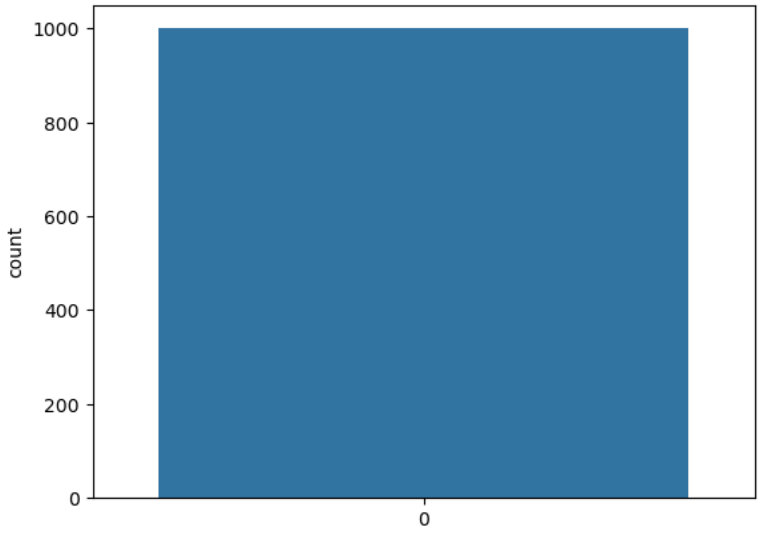
In [12]:
```
df['fraud_reported'] = df['fraud_reported'].str.replace('Y', '1')
df['fraud_reported'] = df['fraud_reported'].str.replace('N', '0')
df['fraud_reported'] = df['fraud_reported'].astype(int)
```

In [13]:
```
df['fraud_reported'].unique()
```

Out[13]:
```
array([1, 0])
```

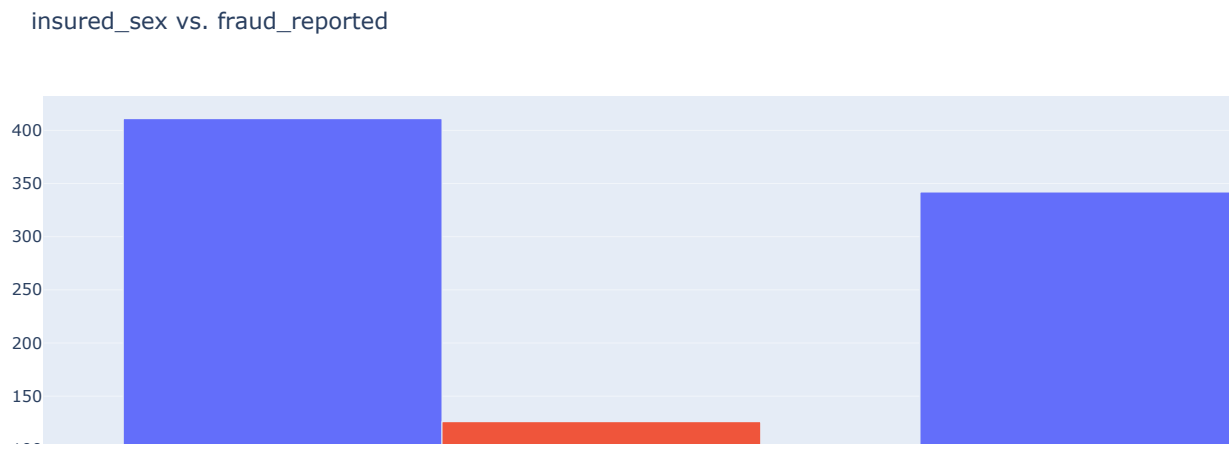
In [18]:
```
sb.countplot(df['fraud_reported'])
```

Out[18]:
```
<Axes: ylabel='count'>
```

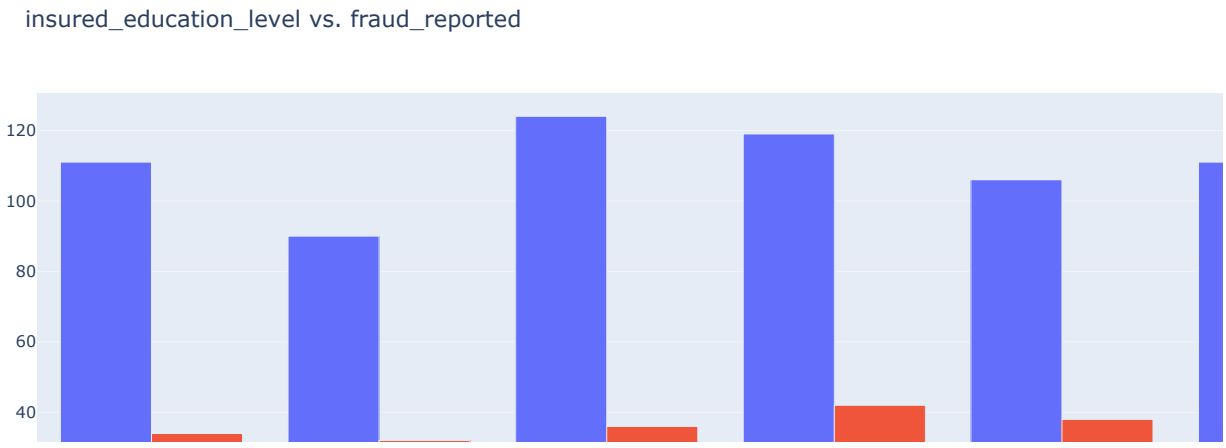


In [19]:
```
def vis_data(df, x, y = 'fraud_reported', graph = 'countplot'):
    if graph == 'hist':
        fig = px.histogram(df, x = x)
        fig.update_layout(title = 'Distribution of {x}'.format(x = x))
        fig.show()
    elif graph == 'bar':
      fig = px.bar(df, x = x, y = y)
      fig.update_layout(title = '{x} vs. {y}'.format(x = x, y = y))
      fig.show()
    elif graph == 'countplot':
      a = df.groupby([x,y]).count()
      a.reset_index(inplace = True)
      no_fraud = a[a['fraud_reported'] == 0]
      yes_fraud = a[a['fraud_reported'] == 1]
      trace1 = go.Bar(x = no_fraud[x], y = no_fraud['policy_number'], name = 'No Fraud')
      trace2 = go.Bar(x = yes_fraud[x], y = yes_fraud['policy_number'], name = 'Fraud')
      fig = go.Figure(data = [trace1, trace2])
      fig.update_layout(title = '{x} vs. {y}'.format(x=x, y = y))
      fig.update_layout(barmode = 'group')
      fig.show()
```

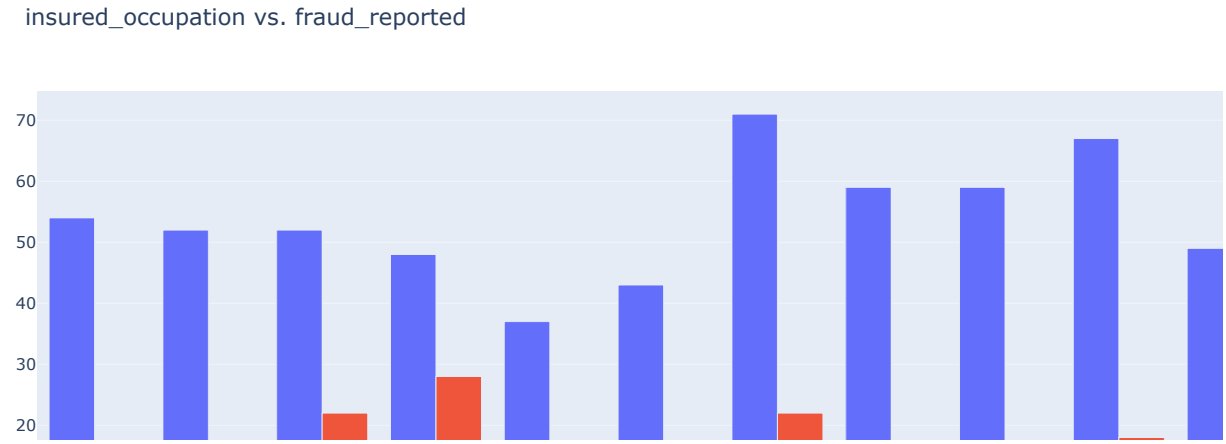
In [20]:
```
vis_data(df, 'insured_sex')
```



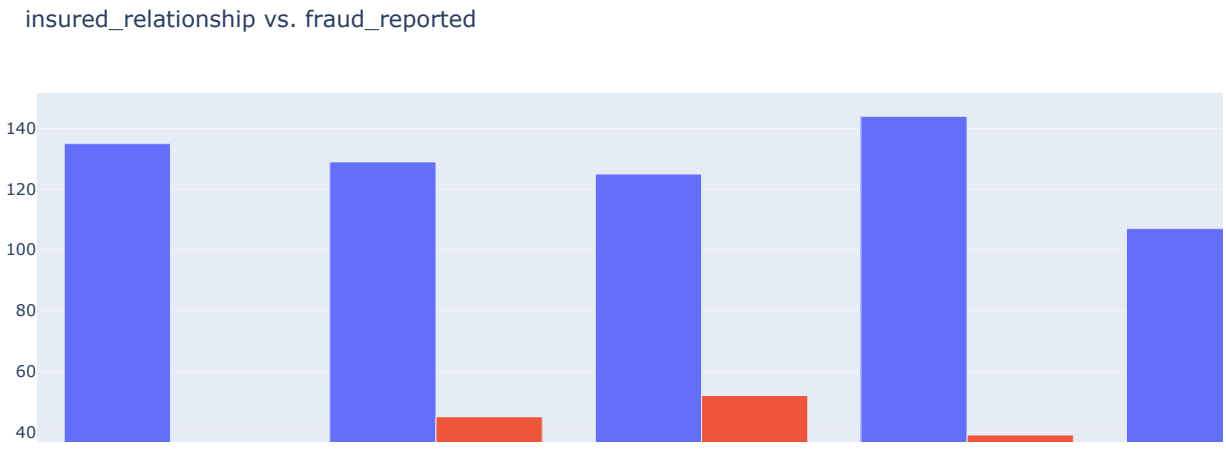
In [21]:
```
vis_data(df, 'insured_education_level')
```



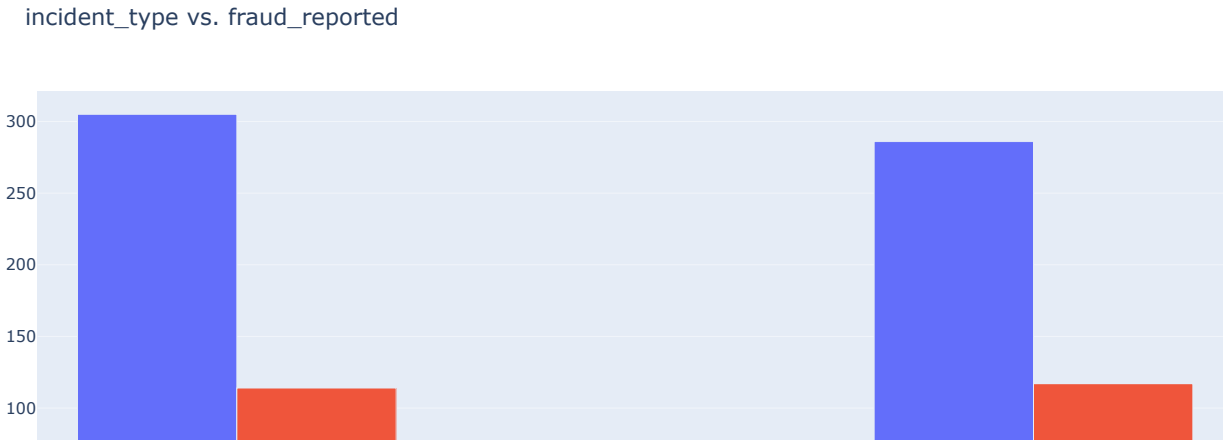
In [22]:
```
vis_data(df, 'insured_occupation')
```



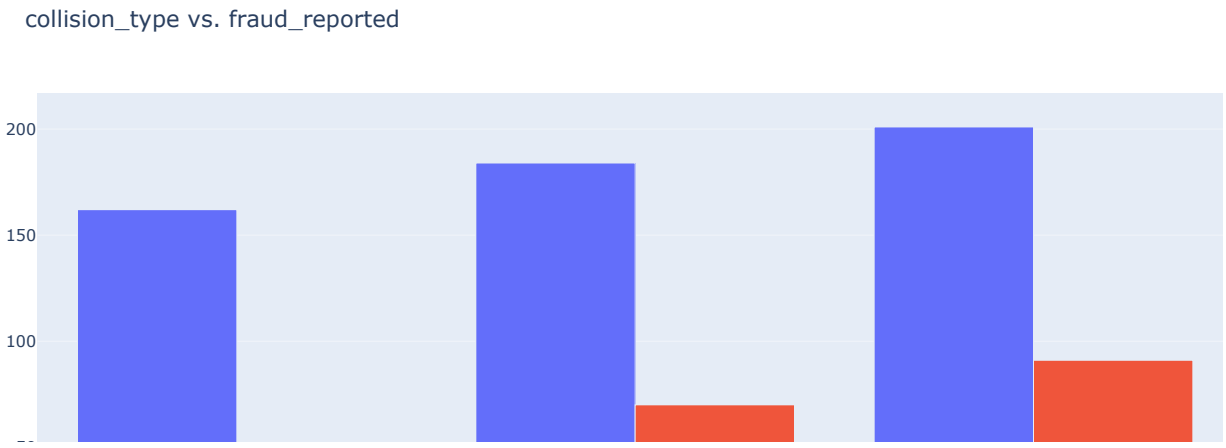
In [23]:
```
vis_data(df, 'insured_relationship')
```



In [24]:
```
vis_data(df, 'incident_type')
```

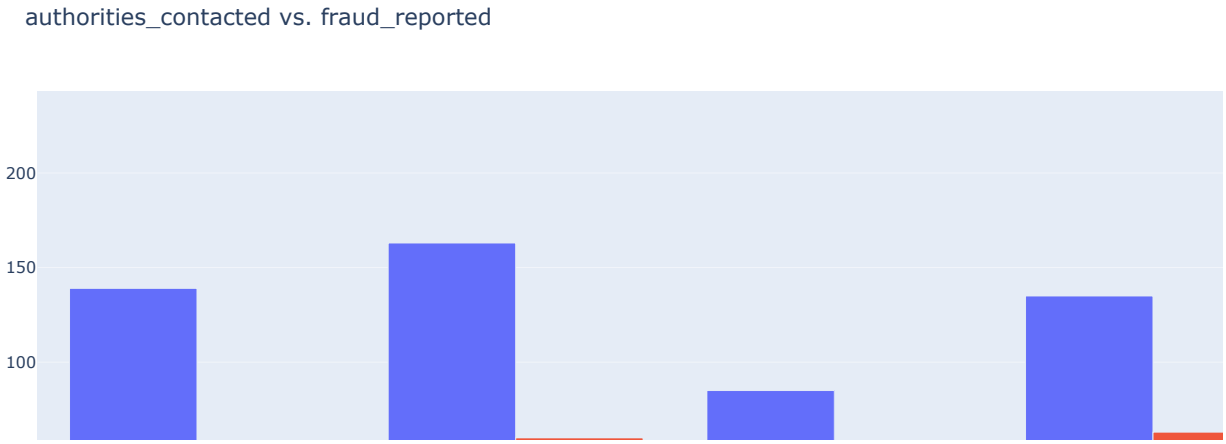


In [25]:
```
vis_data(df, 'collision_type')
```

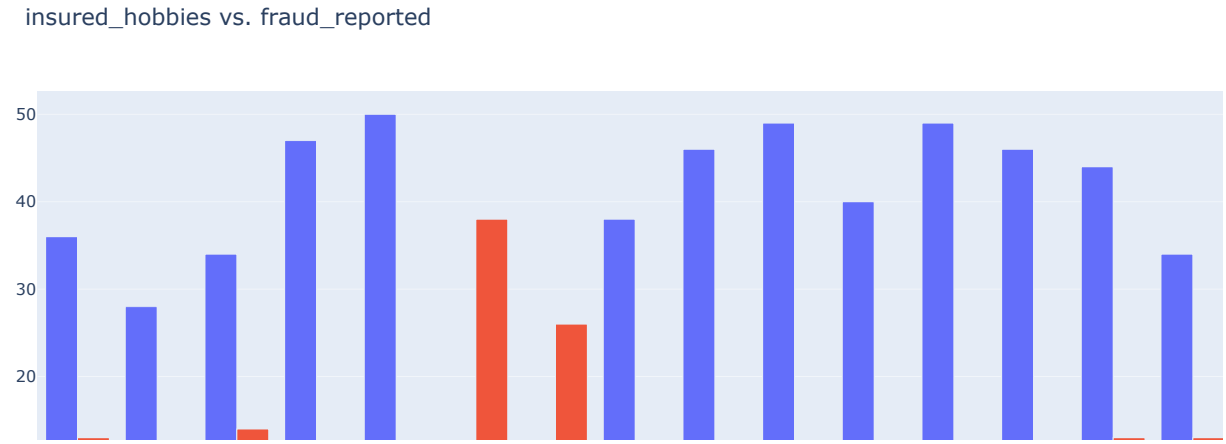


In [26]:
```
vis_data(df, 'incident_severity')
```

In [27]:
```
vis_data(df, 'authorities_contacted')
```



In [28]:
```
vis_data(df, 'insured_hobbies')
```



In [29]:
```
hobbies = df['insured_hobbies'].unique()
for hobby in hobbies:
  if (hobby != 'chess') & (hobby != 'cross-fit'):
    df['insured_hobbies'] = df['insured_hobbies'].str.replace(hobby, 'other')

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

Out[29]:
```
array(['other', 'chess', 'cross-fit'], dtype=object)
```

In [30]:
```
df.head()
```

Out[30]:

	months_as_customer	age	policy_number	policy_bind_date	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insured_educatio
0	328	48	521585	2014-10-17	1000	1406.91	0	466132	MALE	
1	228	42	342868	2006-06-27	2000	1197.22	5000000	468176	MALE	
2	134	29	687698	2000-06-09	2000	1413.14	5000000	430632	FEMALE	
3	256	41	227811	1990-05-25	2000	1415.74	6000000	608117	FEMALE	
4	228	44	367455	2014-06-06	1000	1583.91	6000000	610706	MALE	As

5 rows × 33 columns

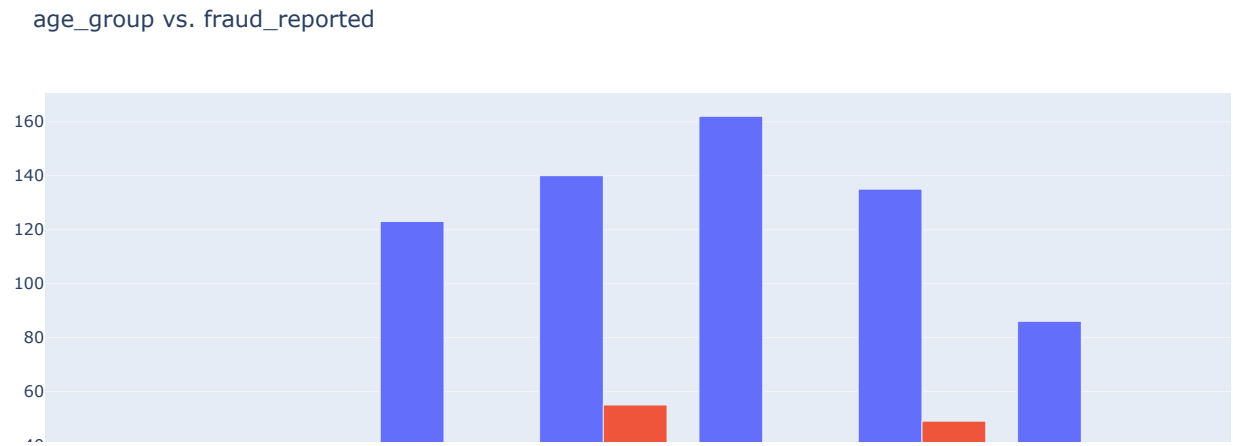
In [31]:
```
vis_data(df, 'age', 'anything', 'hist')
```

In [32]:
```
df['age'].describe()
```

Out[32]:
```
count    1000.000000
mean       38.948000
std         9.140287
min        19.000000
25%        32.000000
50%        38.000000
75%        44.000000
max        64.000000
Name: age, dtype: float64
```

In [33]:
```
bin_labels = ['15-20', '21-25', '26-30', '31-35', '36-40', '41-45', '46-50', '51-55', '56-60', '61-65']
bins = [15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65]

df['age_group'] = pd.cut(df['age'], bins = bins, labels = bin_labels, include_lowest = True)
```

In [34]:
```
vis_data(df, 'age_group')
```



In [35]:
```
vis_data(df, 'months_as_customer', 'not', 'hist')
```

In [36]:
```
df['months_as_customer'].describe()
```

Out[36]:
```
count    1000.000000
mean      203.954000
std       115.113174
min         0.000000
25%       115.750000
50%       199.500000
75%       276.250000
max       479.000000
Name: months_as_customer, dtype: float64
```

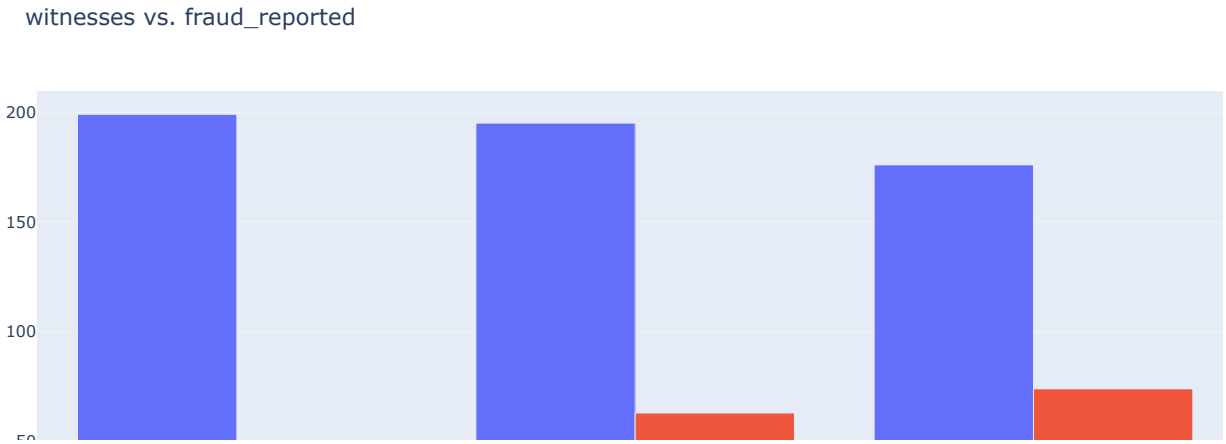
In [37]:
```
bins = [0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500]
bin_labels = ['0-50','51-100','101-150','151-200','201-250','251-300','301-350','351-400','401-450','451-500']

df['months_as_customer_groups'] = pd.cut(df['months_as_customer'], bins = 10, labels = bin_labels, include_lowest= True)
```

In [38]:
```
vis_data(df, 'months_as_customer_groups')
```

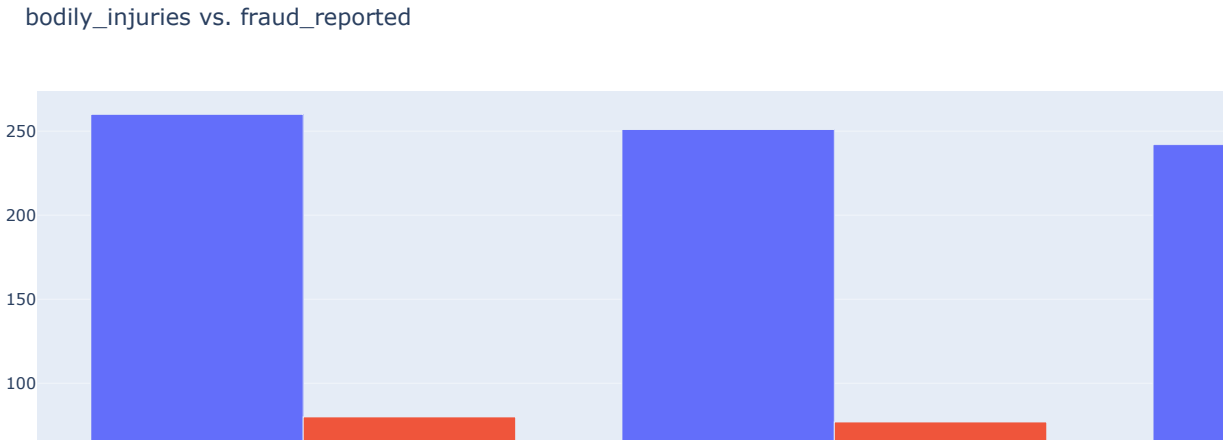
In [39]:
```
vis_data(df, 'auto_make')
```

In [40]:
```
vis_data(df, 'number_of_vehicles_involved')
```

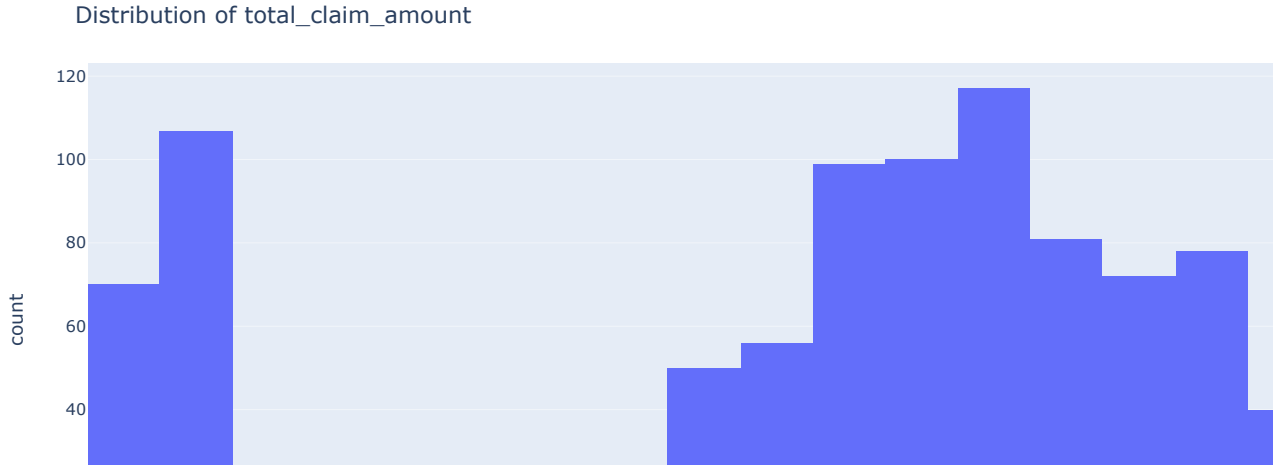
In [41]:
```
vis_data(df, 'witnesses', 'fraud_reported')
```



In [42]:
```
vis_data(df, 'bodily_injuries')
```

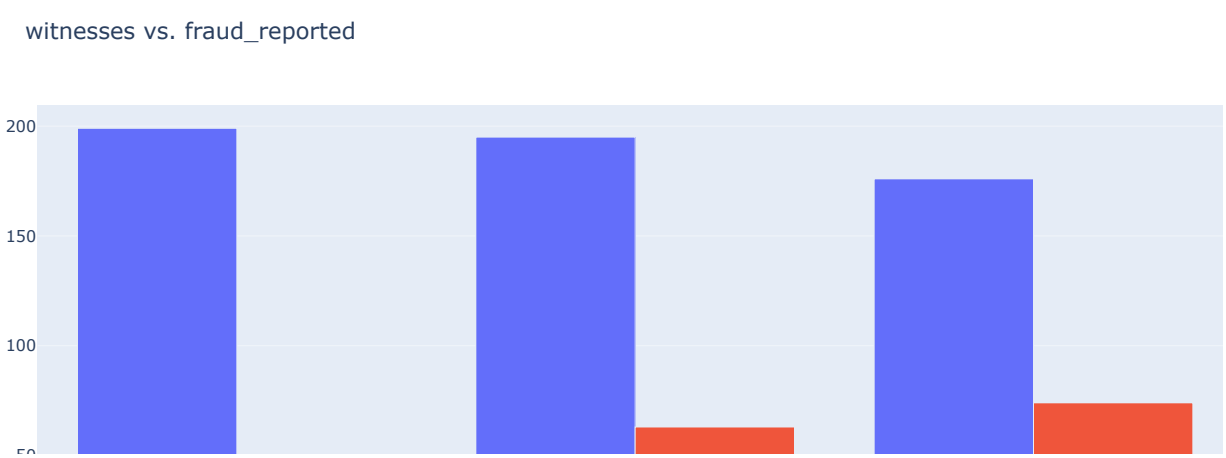


In [43]:
```
vis_data(df, 'total_claim_amount', 'y', 'hist')
```

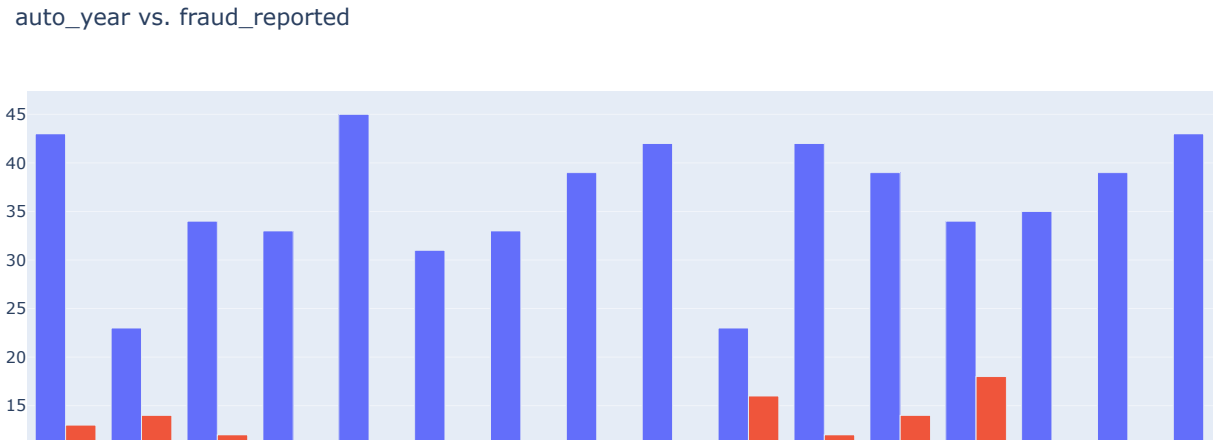


In [45]:
```
vis_data(df, 'number_of_vehicles_involved')
```

In [46]:
```
vis_data(df, 'witnesses')
```



In [47]:
```
vis_data(df, 'auto_year')
```



In [48]:
```
df['policy_annual_premium'].describe()
```

Out[48]:
```
count    1000.000000
mean     1256.406150
std       244.167395
min       433.330000
25%      1089.607500
50%      1257.200000
75%      1415.695000
max      2047.590000
Name: policy_annual_premium, dtype: float64
```

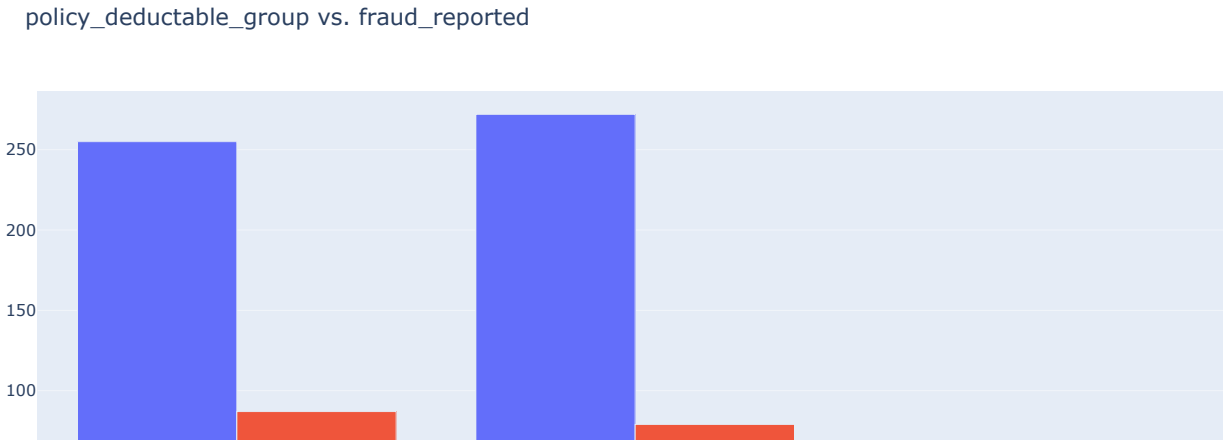
In [49]:
```
bins = list(np.linspace(0,2500, 6, dtype = int))
bin_labels = ['very low', 'low', 'medium', 'high', 'very high']

df['policy_annual_premium_groups'] = pd.cut(df['policy_annual_premium'], bins = bins, labels=bin_labels)
```

In [ ]:
```
vis_data(df, 'policy_annual_premium_groups')
```

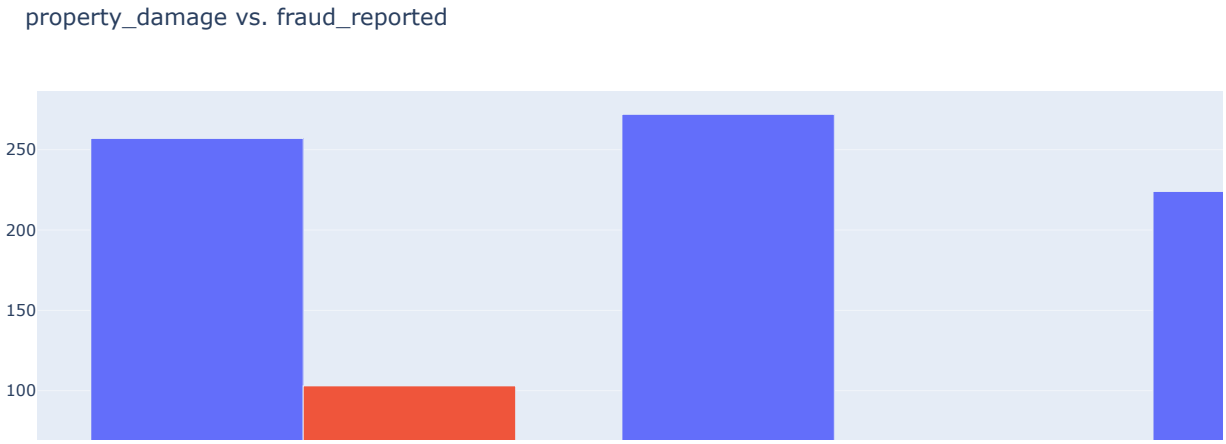
In [50]:
```
df['policy_deductable'].describe()
```

Out[50]:
```
count    1000.000000
mean     1136.000000
std       611.864673
min       500.000000
25%       500.000000
50%      1000.000000
75%      2000.000000
max      2000.000000
Name: policy_deductable, dtype: float64
```

In [51]:
```
bins = list(np.linspace(0,2000, 5, dtype = int))
bin_labels = ['0-500', '501-1000', '1001-1500', '1501-2000']

df['policy_deductable_group'] = pd.cut(df['policy_deductable'], bins = bins, labels = bin_labels)

vis_data(df, 'policy_deductable_group')
```

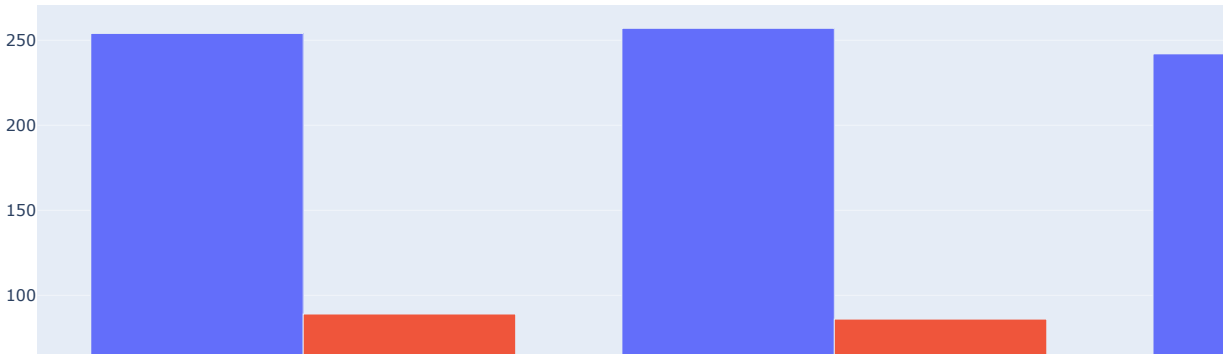


In [52]:
```
vis_data(df, 'property_damage')
```



In [53]:
```
vis_data(df, 'police_report_available')
```

police_report_available vs. fraud_reported



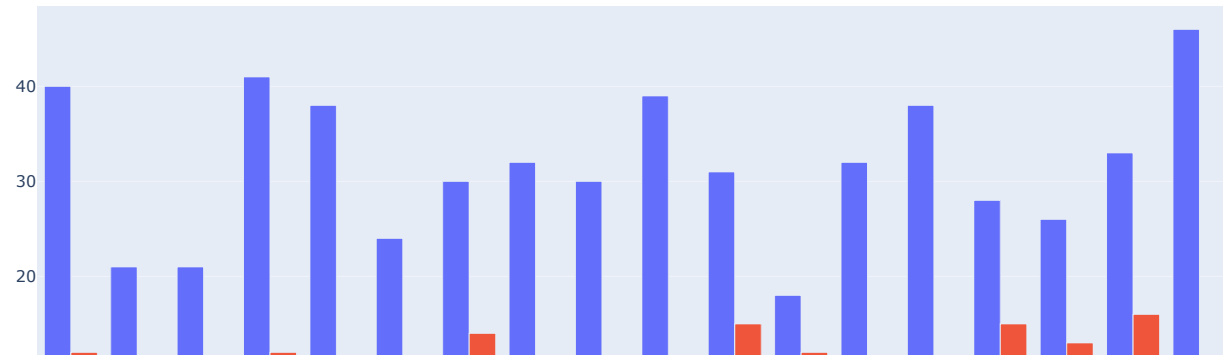
In [ ]:

In [ ]:

In [ ]:

In [44]:
```
vis_data(df, 'incident_hour_of_the_day')
```

incident_hour_of_the_day vs. fraud_reported



In [54]:
```
df = df.drop(['age', 'months_as_customer', 'policy_deductable', 'policy_annual_premium'], axis = 1)
df.columns
```

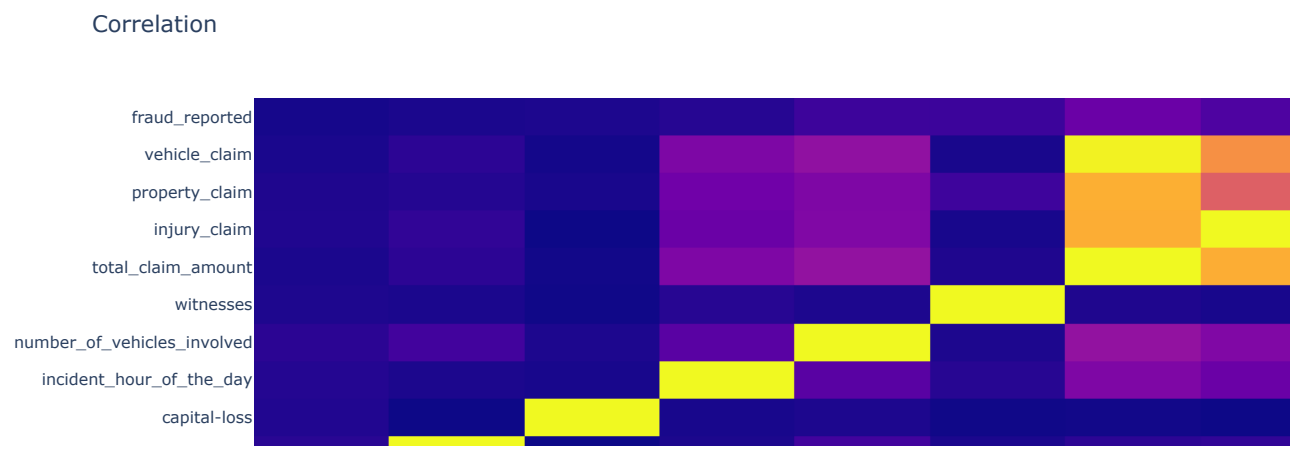
Out[54]:
```
Index(['policy_number', 'policy_bind_date', 'umbrella_limit', 'insured_zip',
       'insured_sex', 'insured_education_level', 'insured_occupation',
       'insured_hobbies', 'insured_relationship', 'capital-gains',
       'capital-loss', 'incident_type', 'collision_type', 'incident_severity',
       'authorities_contacted', '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', 'age_group',
       'months_as_customer_groups', 'policy_annual_premium_groups',
       'policy_deductable_group'],
      dtype='object')
```

In [55]:
```
required_columns = ['policy_number', 'insured_sex', 'insured_education_level', 'insured_occupation',
       'insured_hobbies', 'capital-gains', 'capital-loss', 'incident_type', 'collision_type', 'incident_severity',
       'authorities_contacted', 'incident_hour_of_the_day', 'number_of_vehicles_involved',
       'witnesses', 'total_claim_amount',
       'injury_claim', 'property_claim', 'vehicle_claim',
       'fraud_reported', 'age_group',
       'months_as_customer_groups', 'policy_annual_premium_groups']

print(len(required_columns))
```

```
22
```

In [56]:
```
df1 = df[required_columns]

corr_matrix = df1.corr()

fig = go.Figure(data = go.Heatmap(
                                z = corr_matrix.values,
                                x = list(corr_matrix.columns),
                                y = list(corr_matrix.index)))

fig.update_layout(title = 'Correlation')

fig.show()
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_11400\1864175827.py:3: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid colum
ns or specify the value of numeric_only to silence this warning.
```



In [57]:
```
t = df['total_claim_amount'].iloc[1]
a = df['vehicle_claim'].iloc[1]
b = df['property_claim'].iloc[1]
c = df['injury_claim'].iloc[1]

print(t)
a+b+c
```

```
5070
```

Out[57]:
```
5070
```

In [58]:
```
required_columns = ['insured_sex', 'insured_occupation',
       'insured_hobbies', 'capital-gains', 'capital-loss', 'incident_type', 'collision_type', 'incident_severity',
```

```
       'authorities_contacted', 'incident_hour_of_the_day', 'number_of_vehicles_involved',
       'witnesses', 'total_claim_amount', 'fraud_reported', 'age_group',
       'months_as_customer_groups', 'policy_annual_premium_groups']

print(len(required_columns))
```

```
17
```

In [59]:
```
df1 = df1[required_columns]
df1.head()
```

Out[59]:

	insured_sex	insured_occupation	insured_hobbies	capital-gains	capital-loss	incident_type	collision_type	incident_severity	authorities_contacted	incident_hour_of_the_da
0	MALE	craft-repair	other	53300	0	Single Vehicle Collision	Side Collision	Major Damage	Police	
1	MALE	machine-op-inspct	other	0	0	Vehicle Theft	?	Minor Damage	Police	
2	FEMALE	sales	other	35100	0	Multi-vehicle Collision	Rear Collision	Minor Damage	Police	
3	FEMALE	armed-forces	other	48900	-62400	Single Vehicle Collision	Front Collision	Major Damage	Police	
4	MALE	sales	other	66000	-46000	Vehicle Theft	?	Minor Damage	None	2

In [60]:
```
cat_cols = ['age_group', 'months_as_customer_groups', 'policy_annual_premium_groups']
for col in cat_cols:
  df1[col] = df1[col].astype('object')

columns_to_encode = []
for col in df1.columns:
  if df1[col].dtype == 'object':
    columns_to_encode.append(col)

columns_to_encode
```

Out[60]:
```
['insured_sex',
 'insured_occupation',
 'insured_hobbies',
 'incident_type',
 'collision_type',
 'incident_severity',
 'authorities_contacted',
 'age_group',
 'months_as_customer_groups',
 'policy_annual_premium_groups']
```

In [61]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   insured_sex                  1000 non-null   object
 1   insured_occupation           1000 non-null   object
 2   insured_hobbies              1000 non-null   object
 3   capital-gains                1000 non-null   int64
 4   capital-loss                 1000 non-null   int64
 5   incident_type                1000 non-null   object
 6   collision_type               1000 non-null   object
 7   incident_severity            1000 non-null   object
 8   authorities_contacted        1000 non-null   object
 9   incident_hour_of_the_day     1000 non-null   int64
 10  number_of_vehicles_involved  1000 non-null   int64
 11  witnesses                    1000 non-null   int64
 12  total_claim_amount           1000 non-null   int64
 13  fraud_reported               1000 non-null   int32
 14  age_group                    1000 non-null   object
 15  months_as_customer_groups    1000 non-null   object
 16  policy_annual_premium_groups 1000 non-null   object
dtypes: int32(1), int64(6), object(10)
memory usage: 129.0+ KB
```

In [62]:
```
df1.head()
```

Out[62]:

	insured_sex	insured_occupation	insured_hobbies	capital-gains	capital-loss	incident_type	collision_type	incident_severity	authorities_contacted	incident_hour_of_the_da
0	MALE	craft-repair	other	53300	0	Single Vehicle Collision	Side Collision	Major Damage	Police	
1	MALE	machine-op-inspct	other	0	0	Vehicle Theft	?	Minor Damage	Police	
2	FEMALE	sales	other	35100	0	Multi-vehicle Collision	Rear Collision	Minor Damage	Police	
3	FEMALE	armed-forces	other	48900	-62400	Single Vehicle Collision	Front Collision	Major Damage	Police	
4	MALE	sales	other	66000	-46000	Vehicle Theft	?	Minor Damage	None	2

In [63]:
```
df2 = pd.get_dummies(df1, columns = columns_to_encode)

df2.head()
```

Out[63]:

	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicles_involved	witnesses	total_claim_amount	fraud_reported	insured_sex_FEMALE	insured_sex_MALE
0	53300	0	5	1	2	71610	1	0	1
1	0	0	8	1	0	5070	1	0	1
2	35100	0	7	3	3	34650	0	1	0
3	48900	-62400	5	1	2	63400	1	1	0
4	66000	-46000	20	1	1	6500	0	0	1

5 rows × 68 columns

In [64]:
```
features = []
for col in df2.columns:
  if col != 'fraud_reported':
    features.append(col)

target = 'fraud_reported'

X = df2[features]
y = df2[target]
```

In [65]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X = sc.fit_transform(X)
```

In [66]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1)
```

In [67]:
```
lr = LogisticRegression()

lr.fit(X_train, y_train)
preds = lr.predict(X_test)

score = lr.score(X_test, y_test)
print(score)
```

```
0.848
```

In [68]:
```
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

           0       0.87      0.93      0.90       180
           1       0.78      0.64      0.70        70

    accuracy                           0.85       250
   macro avg       0.82      0.79      0.80       250
weighted avg       0.84      0.85      0.84       250
```

In [69]:
```
oversample = SMOTE(random_state=9)
```

In [70]:
```
X_train, X_test, y_train, y_test = train_test_split(X_over, y_over, random_state = 1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[70], line 1
----> 1 X_train, X_test, y_train, y_test = train_test_split(X_over, y_over, random_state = 1)

NameError: name 'X_over' is not defined
```

In [71]:
```
X_over, y_over = oversample.fit_resample(X_train, y_train)
```

In [72]:
```
lr.fit(X_train, y_train)
preds = lr.predict(X_test)
score = lr.score(X_test, y_test)
print(score)
print()
print(classification_report(y_test, preds))
```

```
0.848

              precision    recall  f1-score   support

           0       0.87      0.93      0.90       180
           1       0.78      0.64      0.70        70

    accuracy                           0.85       250
   macro avg       0.82      0.79      0.80       250
weighted avg       0.84      0.85      0.84       250
```

In [73]:
```
dtc = DecisionTreeClassifier()

dtc.fit(X_train, y_train)
preds = dtc.predict(X_test)

score = dtc.score(X_test, y_test)
print(score)
print()
print(classification_report(y_test, preds))
```

```
0.764

              precision    recall  f1-score   support

           0       0.83      0.84      0.84       180
           1       0.58      0.56      0.57        70

    accuracy                           0.76       250
   macro avg       0.71      0.70      0.70       250
weighted avg       0.76      0.76      0.76       250
```

In [74]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [75]:
```
rfc = RandomForestClassifier(random_state = 1)
rfc.fit(X_train, y_train)
```

Out[75]:
```
RandomForestClassifier(random_state=1)
```

In [76]:
```
preds = rfc.predict(X_test)

score = rfc.score(X_test, y_test)
print(score*100)
print()
print(classification_report(y_test, preds))
```

```
81.6

              precision    recall  f1-score   support

           0       0.85      0.91      0.88       180
           1       0.71      0.59      0.64        70

    accuracy                           0.82       250
   macro avg       0.78      0.75      0.76       250
weighted avg       0.81      0.82      0.81       250
```

In [77]:
```
svc = SVC(kernel='linear')
svc.fit(X_train, y_train)

preds = svc.predict(X_test)

print('Score:' , svc.score(X_test, y_test))
print('Classification report:', classification_report(y_test, preds))
```

```
Score: 0.88
Classification report:               precision    recall  f1-score   support

           0       0.96      0.87      0.91       180
           1       0.73      0.91      0.81        70

    accuracy                           0.88       250
   macro avg       0.85      0.89      0.86       250
weighted avg       0.90      0.88      0.88       250
```

In [78]:
```
degrees = [2,3,4,5,6,7,8]
kernels = ['poly', 'rbf', 'sigmoid']
c_value = [1,2,3]
```

In [79]:
```
scores = {}
for degree in degrees:
    for kernel in kernels:
        for c in c_value:
            svc_t = SVC(kernel = kernel, degree = degree, C = c)
            svc_t.fit(X_train, y_train)

            preds = svc_t.predict(X_test)
            score = svc_t.score(X_test,y_test)
#             print('Score with degree as {d}, kernel as {k}, C as {c} is:'.format(d = degree, k = kernel, c = c), score)
            scores['Score with degree as {d}, kernel as {k}, C as {c} is best'.format(d = degree, k = kernel, c = c)] = score

print(max(scores, key=scores.get))
```

```
Score with degree as 2, kernel as sigmoid, C as 3 is best
```

In [80]:
```
svc_tuned = SVC(kernel='sigmoid', degree = 2, C = 3)
svc_tuned.fit(X_train, y_train)

preds = svc_tuned.predict(X_test)

print('Score:' , svc_tuned.score(X_test, y_test))
print('Classification report:', classification_report(y_test, preds))
```

```
Score: 0.896
Classification report:               precision    recall  f1-score   support

           0       0.95      0.91      0.93       180
           1       0.78      0.87      0.82        70

    accuracy                           0.90       250
   macro avg       0.86      0.89      0.88       250
weighted avg       0.90      0.90      0.90       250
```

In [81]:
```
rfc_tuned = RandomForestClassifier(n_estimators = 1000, random_state = 1, min_samples_split = 2)
rfc_tuned.fit(X_train, y_train)
preds_tuned = rfc_tuned.predict(X_test)
score = rfc_tuned.score(X_test, y_test)
print(score)
```

```
0.832
```

In [82]:
```
n_estimators = [100, 300, 500, 800, 1200]
max_depth = [5, 8, 15, 25, 30]
min_samples_split = [2, 5, 10, 15, 100]
min_samples_leaf = [1, 2, 5, 10]

hyper = dict(n_estimators = n_estimators, max_depth = max_depth,
             min_samples_split = min_samples_split,
             min_samples_leaf = min_samples_leaf)

grid = GridSearchCV(rfc, hyper, cv = 3, verbose = 1,
                    n_jobs = -1)
best = grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 500 candidates, totalling 1500 fits
```

In [86]:
```
print(best)
```

```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=1), n_jobs=-1,
             param_grid={'max_depth': [5, 8, 15, 25, 30],
                         'min_samples_leaf': [1, 2, 5, 10],
                         'min_samples_split': [2, 5, 10, 15, 100],
                         'n_estimators': [100, 300, 500, 800, 1200]},
             verbose=1)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: