insurance-claim-case-study

May 29, 2023

[205]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime as dt
import seaborn as sns
import re
import scipy.stats as stats

from sklearn.preprocessing import OneHotEncoder



# set the graphs to show in the jupyter notebook
%matplotlib inline

# set seabor graphs to a better style
sns.set(style="ticks")

pd.set_option('display.max_columns', None)
```

1 Insurance claim case study

1.0.1 1.Import claims_data.csv and cust_data.csv which is provided to you and combine the two datasets appropriately to create a 360-degree view of the data. Use the same for the subsequent questions.

[206]:
```
input_file_path = "C:\\Projects\\EDA\\data\\input_files//"
input_file_name = "claims.csv"
input_file_path + input_file_name
Claims = pd.read_csv(input_file_path + input_file_name )
```

[207]:
```
input_file_path = "C:\\Projects\\EDA\\data\\input_files//"
input_file_name = "cust_demographics.csv"
input_file_path + input_file_name
Cust_demo = pd.read_csv(input_file_path + input_file_name )
```

[208]:
```
Cust_demo= Cust_demo.rename(columns ={'CUST_ID' : 'customer_id'})
```

[209]:
```
Cust_claims = pd.merge(left =Claims, right = Cust_demo, on= 'customer_id',␣
↪how='left')
```

[178]:
```
Cust_claims.shape
```

[178]:
```
(1100, 15)
```

[210]:
```
Cust_claims.head()
```

[210]:

	claim_id	customer_id	incident_cause	claim_date	claim_area	police_report	claim_type	claim_amount	total_policy_claims	fraudulent	gender	DateOfBirth	State	Contact	Segment
0	54004764	21868593	Driver error	11/27/2017	Auto	No	Material only	$2980	1.0	No	Female	12-Jan-79	VT	789-916-8172	Platinum
1	33985796	75740424	Crime	10/03/2018	Home	Unknown	Material only	$2980	3.0	No	Female	13-Jan-70	ME	265-543-1264	Silver
2	53522022	30308357	Other driver error	02/02/2018	Auto	No	Material only	$3369.5	1.0	Yes	Female	11-Mar-84	TN	798-631-4758	Silver
3	13015401	47830476	Natural causes	06/17/2018	Auto	No	Material only	$1680	1.0	No	Female	01-May-86	MA	413-187-7945	Silver
4	22890252	19269962	Crime	01/13/2018	Auto	No	Material only	$2680	1.0	No	Male	13-May-77	NV	956-871-8691	Gold

1.0.2 2.Perform a data audit for the datatypes and find out if there are any mismatch within the current datatypes of the columns and their business significance.

[211]:
```
Cust_claims['claim_id'] = Cust_claims['claim_id'].astype('object')
Cust_claims['customer_id'] = Cust_claims['customer_id'].astype('object')
Cust_claims['claim_date'] = Cust_claims['claim_date'].astype('datetime64')
#Cust_claims['DateOfBirth'] = Cust_claims['DateOfBirth'].astype('datetime64')
```

[212]:
```
Cust_claims.dtypes
```

[212]:
```
claim_id                    object
customer_id                 object
incident_cause              object
claim_date          datetime64[ns]
claim_area                  object
police_report               object
claim_type                  object
```

```
claim_amount                  object
total_policy_claims          float64
fraudulent                    object
gender                        object
DateOfBirth                   object
State                         object
Contact                       object
Segment                       object
dtype: object
```

1.0.3 3.Convert the column claim_amount to numeric. Use the appropriate modules/attributes to remove the $ sign.

[213]:
```
Cust_claims['claim_amount'] = Cust_claims['claim_amount'].str.replace('$', '')
Cust_claims['claim_amount'] = pd.to_numeric(Cust_claims['claim_amount'])
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_12784\2754579616.py:1: FutureWarning:
The default value of regex will change from True to False in a future version.
In addition, single character regular expressions will *not* be treated as
literal strings when regex=True.
  Cust_claims['claim_amount'] = Cust_claims['claim_amount'].str.replace('$', '')
```

[214]:
```
Cust_claims.dtypes
```

[214]:
```
claim_id                      object
customer_id                   object
incident_cause                object
claim_date            datetime64[ns]
claim_area                    object
police_report                 object
claim_type                    object
claim_amount                 float64
total_policy_claims          float64
fraudulent                    object
gender                        object
DateOfBirth                   object
State                         object
Contact                       object
Segment                       object
dtype: object
```

1.0.4 4.Of all the injury claims, some of them have gone unreported with the police. Create an alert flag (1,0) for all such claims.

[215]:
```
Cust_claims['alert flag'] = 0
Cust_claims.loc [Cust_claims['police_report'] == 'Unknown' , 'alert flag'] = 1
```

1.0.5 5.One customer can claim for insurance more than once and in each claim, multiple categories of claims can be involved. However, customer ID should remain unique.

Retain the most recent observation and delete any duplicated records in the data based on the customer ID column.

[216]:
```
Cust_claims.sort_values(by=['customer_id', 'claim_date'])
Cust_claims=Cust_claims.drop_duplicates(subset='customer_id').
 ↪reset_index(drop=True)
```

1.0.6 6.Check for missing values and impute the missing values with an appropriate value. (mean for continuous and mode for categorical)

[217]:
```
Cust_claims.isnull().sum()
```

[217]:
```
claim_id                0
customer_id             0
incident_cause          0
claim_date              0
claim_area              0
police_report           0
claim_type              0
claim_amount           65
total_policy_claims    10
fraudulent              0
gender                 15
DateOfBirth            15
State                  15
Contact                15
Segment                15
alert flag              0
dtype: int64
```

[218]:
```
for col in Cust_claims.columns:
    if Cust_claims[col].dtype == 'float64':
        Cust_claims[col].fillna(Cust_claims[col].mean(), inplace=True)
    elif Cust_claims[col].dtype == 'object':
        Cust_claims[col].fillna(Cust_claims[col].mode()[0], inplace=True)
```

[219]:
```
Cust_claims.isnull().sum()
```

[219]:
```
claim_id              0
customer_id           0
incident_cause        0
claim_date            0
claim_area            0
police_report         0
claim_type            0
claim_amount          0
total_policy_claims   0
fraudulent            0
gender                0
DateOfBirth           0
State                 0
Contact               0
Segment               0
alert flag            0
dtype: int64
```

1.0.7 7. Calculate the age of customers in years. Based on the age, categorize the customers according to the below criteria

Children < 18

Youth 18-30

Adult 30-60

Senior > 60

[220]:
```
Cust_claims['DateOfBirth'] = pd.to_datetime (Cust_claims['DateOfBirth'], format␣
 ↪= '%d-%b-%y')
Cust_claims['DateOfBirth'] = Cust_claims['DateOfBirth'].dt.
 ↪strftime('19%y-%m-%d')
```

[221]:
```
Cust_claims['DateOfBirth'] = Cust_claims['DateOfBirth'].astype('datetime64')
```

[222]:
```
Cust_claims['birth_Year'] = Cust_claims['DateOfBirth'].dt.year
```

[223]:
```
Cust_claims['current_year']  = dt.datetime.now().year
```

[224]:
```
Cust_claims['Age'] = Cust_claims['current_year'] - Cust_claims['birth_Year']
```

[225]:
```
bins = [0, 18, 30, 60, 120]
labels = ['Children', 'Youth', 'Adult', 'Senior']
Cust_claims['age_category'] = pd.cut(Cust_claims['Age'], bins=bins,␣
 ↪labels=labels)
```

1.0.8 8.What is the average amount claimed by the customers from various segments?

[226]:
```
Average_claim = Cust_claims.groupby(['Segment'])[['claim_amount']].mean()
print(Average_claim)
```

```
          claim_amount
Segment
Gold      12675.262946
Platinum  12368.233815
Silver    12267.349306
```

1.0.9 9.What is the total claim amount based on incident cause for all the claims that have been done at least 20 days prior to 1st of October, 2018.

[227]:
```
cutoff_date = dt.datetime(2018, 10, 1) - dt.timedelta(days=20)
filtered_Cust_claims = Cust_claims[Cust_claims['claim_date'] <= cutoff_date]
```

[228]:
```
total_claimamount = filtered_Cust_claims.
 ↪groupby(['incident_cause'])[['claim_amount']].sum()
print(total_claimamount)
```

```
                  claim_amount
incident_cause
Crime                7.294856e+05
Driver error         3.292987e+06
Natural causes       1.315024e+06
Other causes         3.779919e+06
Other driver error   3.384079e+06
```

1.0.10 10.How many adults from TX, DE and AK claimed insurance for driver related issues and causes?

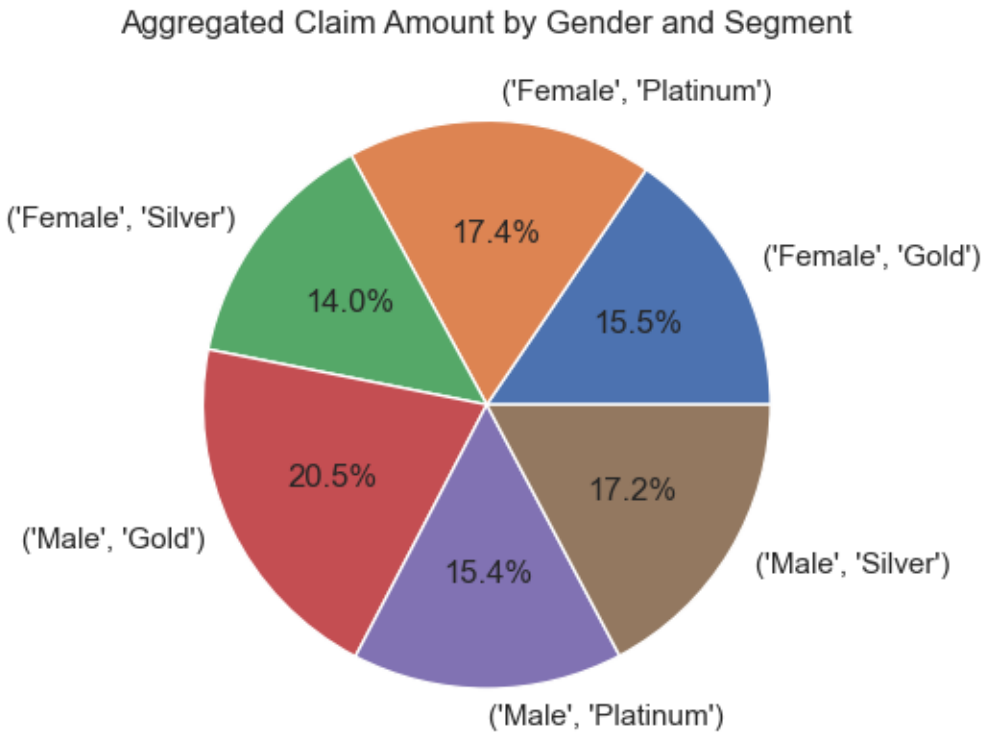
[229]:
```
Adults = Cust_claims[(Cust_claims['age_category'] == 'Adult') &␣
 ↪(Cust_claims['State'].isin(['TX', 'DE', 'AK']))]
```

[230]:
```
driver_related_claims = Adults[Adults['incident_cause'].str.contains('driver',␣
 ↪case=False)]
num_claims = driver_related_claims.shape[0]
print(f"The number of adults from TX, DE, and AK who claimed insurance for␣
 ↪driver-related issues and causes is {num_claims}.")
```

```
The number of adults from TX, DE, and AK who claimed insurance for driver-
related issues and causes is 27.
```

1.0.11 11.Draw a pie chart between the aggregated value of claim amount based on gender and segment. Represent the claim amount as a percentage on the pie chart.

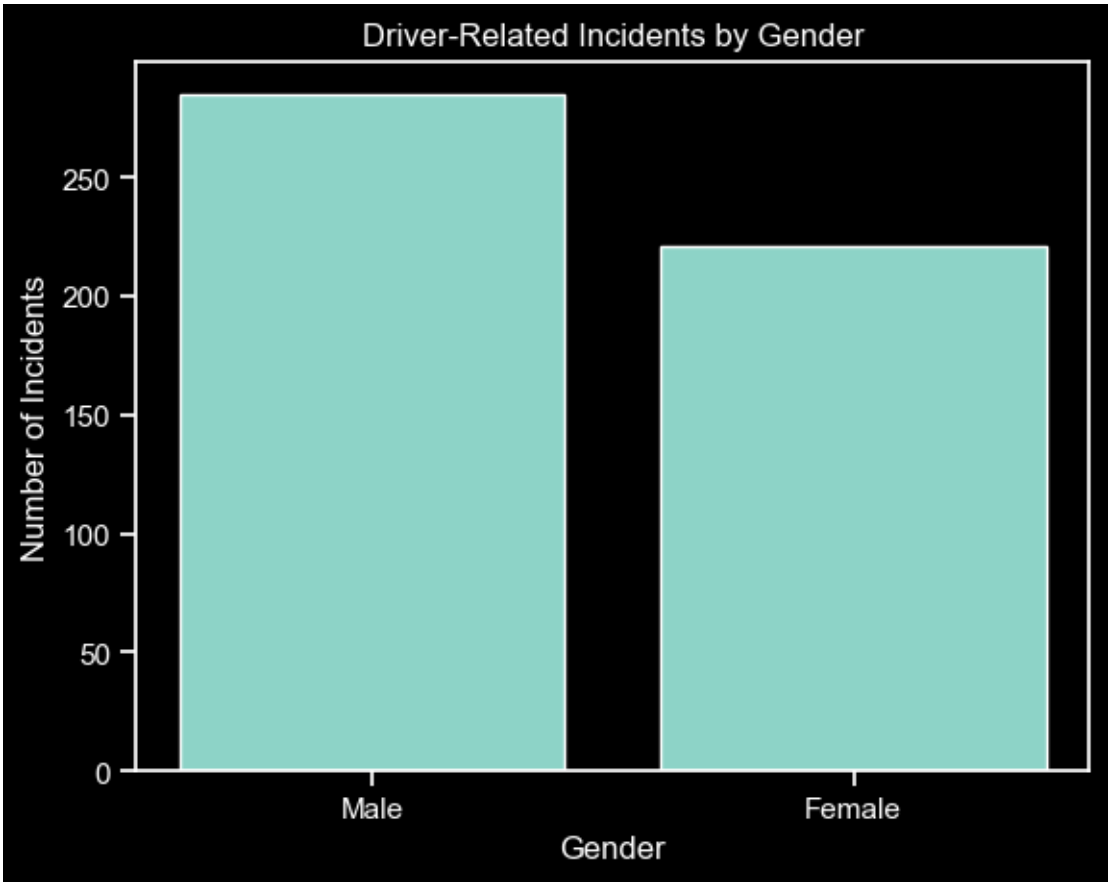
[231]:
```
Grouped = Cust_claims.groupby(['gender','Segment'])[['claim_amount']].sum()

# create a pie chart of the aggregated claim amounts
fig, ax = plt.subplots()
ax.pie(Grouped['claim_amount'], labels=Grouped.index, autopct='%1.1f%%')
ax.set_title('Aggregated Claim Amount by Gender and Segment')
plt.show()
```



1.0.12 12.Among males and females, which gender had claimed the most for any type of driver related issues? E.g. This metric can be compared using a bar chart

[302]:
```
driver_related = Cust_claims[Cust_claims["incident_cause"].str.
 ↪contains("driver", case=False)]
gender_counts = driver_related["gender"].value_counts()
#bar chart
plt.style.use('dark_background')
plt.rcParams.update({'text.color': 'white'})
plt.bar(gender_counts.index, gender_counts.values)
```

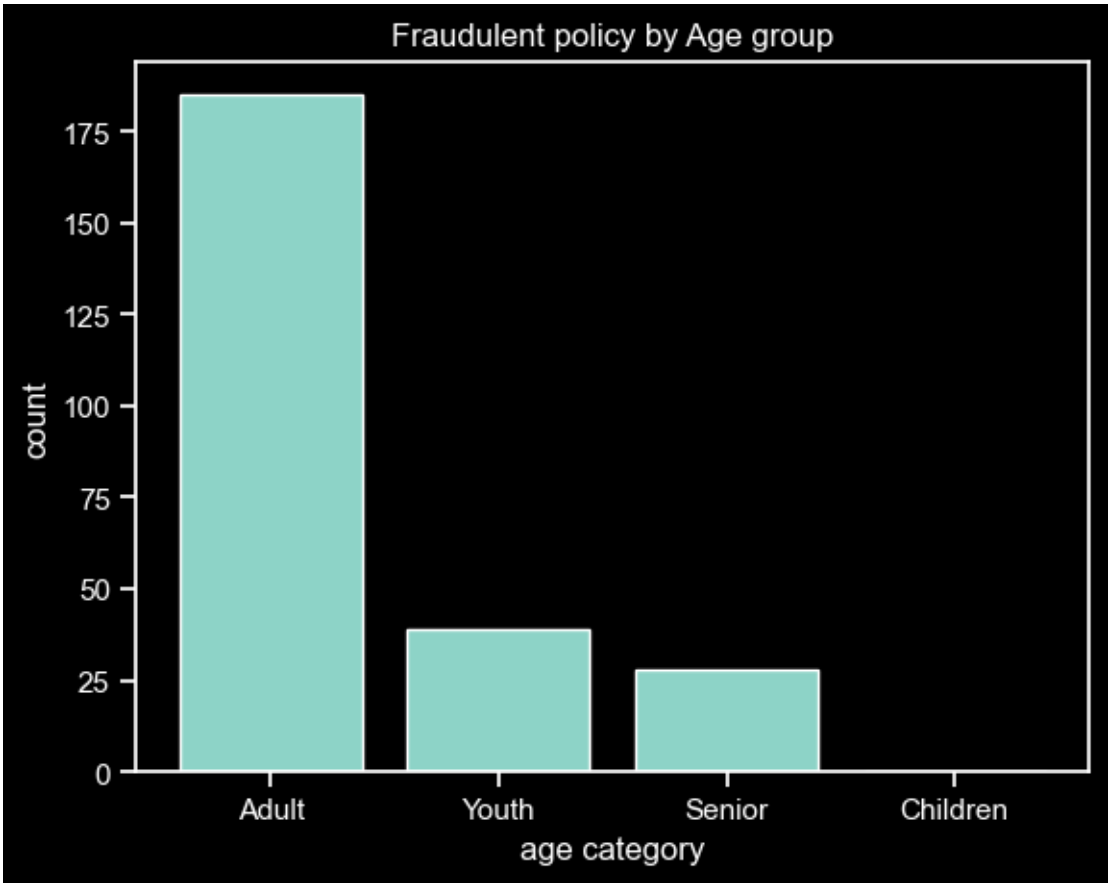
```
plt.title("Driver-Related Incidents by Gender")
plt.xlabel("Gender")
plt.ylabel("Number of Incidents")
plt.show()
```



1.0.13 13.Which age group had the maximum fraudulent policy claims? Visualize it on a bar chart.

[32]:
```
Fraudulent =  Cust_claims[(Cust_claims['fraudulent'] == 'Yes')]
```

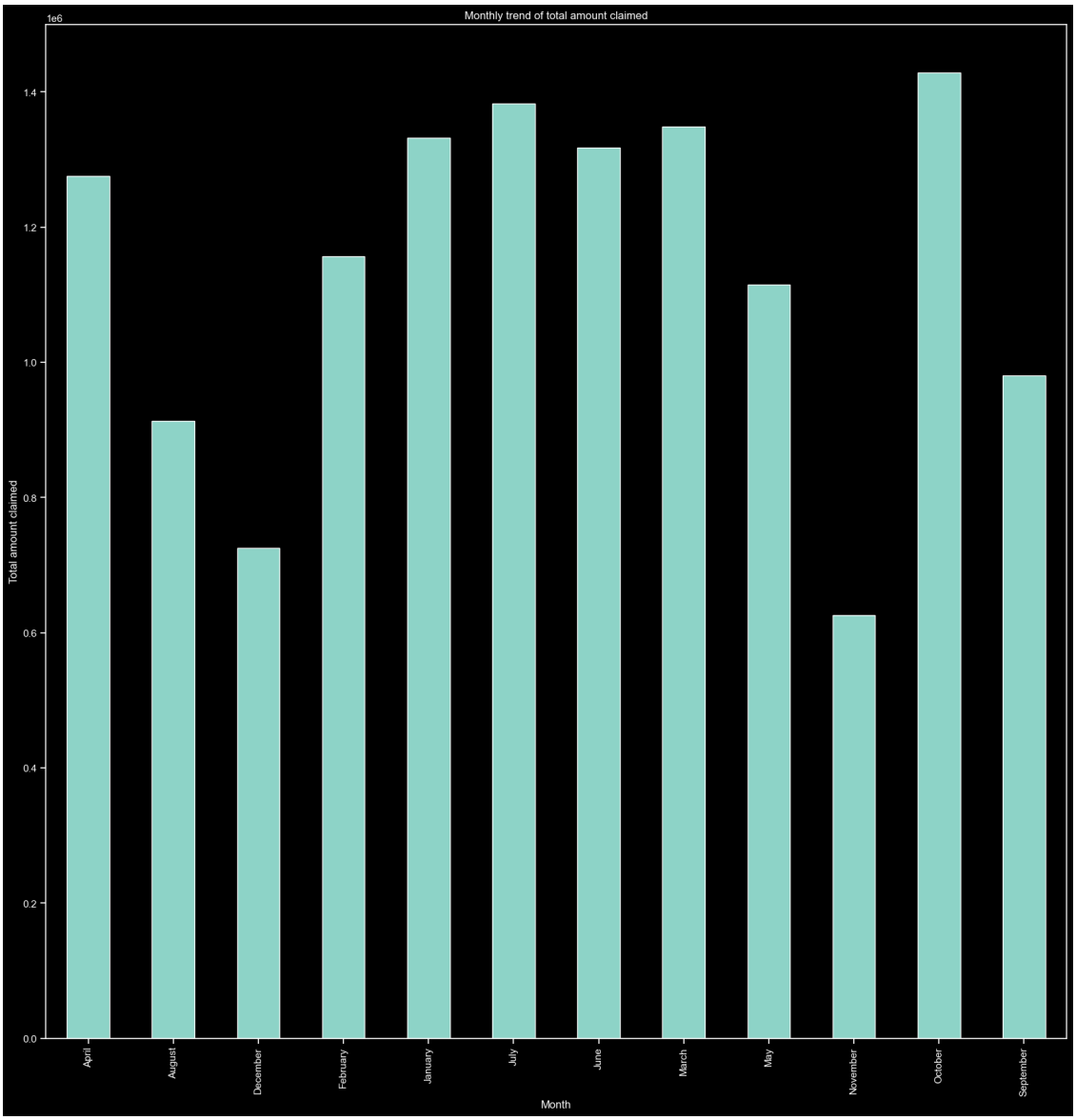
[303]:
```
age_fraudcount = Fraudulent['age_category'].value_counts()
plt.style.use('dark_background')
plt.rcParams.update({'text.color': 'white'})
plt.bar(age_fraudcount.index, age_fraudcount.values)
plt.title("Fraudulent policy by Age group")
plt.xlabel("age category")
plt.ylabel("count")
plt.show()
```



1.0.14 14.Visualize the monthly trend of the total amount that has been claimed by the customers. Ensure that on the "month" axis, the month is in a chronological order not alphabetical order.

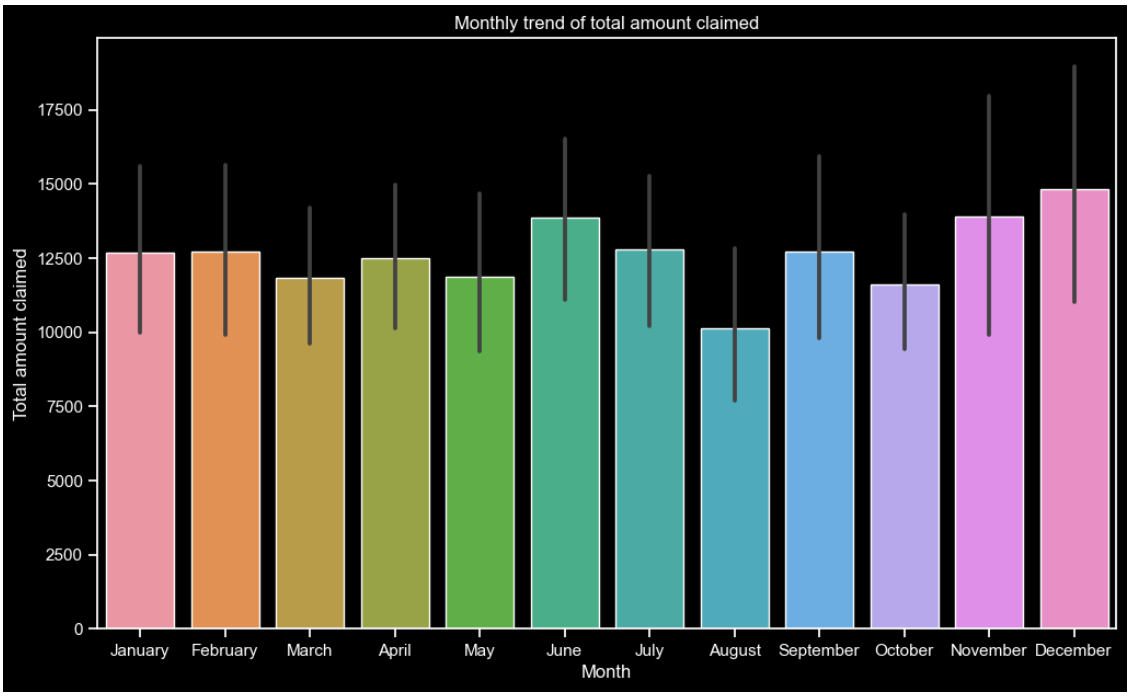
[234]:
```
Cust_claims['Month_name'] = Cust_claims['claim_date'].dt.strftime('%B')
```

[304]:
```
monthly_totals = Cust_claims.groupby('Month_name')['claim_amount'].sum()
plt.style.use('dark_background')
plt.rcParams.update({'text.color': 'white'})
monthly_totals.plot(kind ='bar', figsize=(20,20))
plt.xlabel('Month')
plt.ylabel('Total amount claimed')
plt.title('Monthly trend of total amount claimed')
plt.show()
```



[305]:
```
monthly_totals = Cust_claims.groupby('Month_name')['claim_amount'].sum().
  ↪reset_index()
month_order = ["January", "February", "March", "April", "May", "June", "July",␣
  ↪"August", "September", "October", "November", "December"]

# Set the style and figure size
plt.style.use('dark_background')
plt.rcParams.update({'text.color': 'white'})
plt.figure(figsize=(12, 7))

# Create the bar chart
```

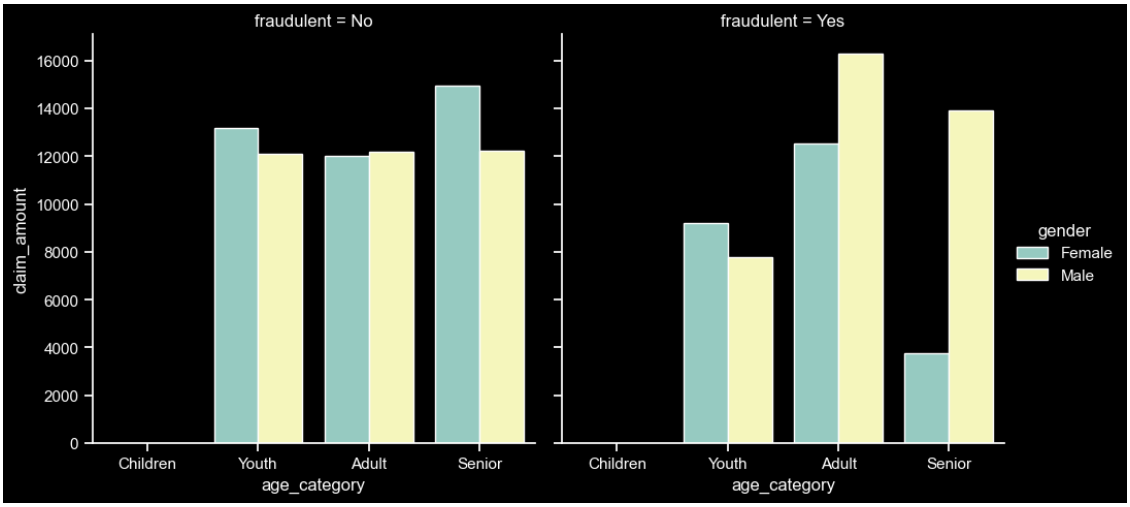
```
sns.barplot(data=monthly_totals, x=Cust_claims["Month_name"],␣
 ↪y=Cust_claims["claim_amount"], order=month_order)
plt.xlabel('Month')
plt.ylabel('Total amount claimed')
plt.title('Monthly trend of total amount claimed')
plt.show()
```



1.0.15 15.What is the average claim amount for gender and age categories and suitably represent the above using a facetted bar chart, one facet that represents fraudulent claims and the other for non-fraudulent claims.

[306]:
```
grouped_data = Cust_claims.groupby(['gender', 'age_category',␣
 ↪'fraudulent'])['claim_amount'].mean().reset_index()
plt.style.use('dark_background')
plt.rcParams.update({'text.color': 'white'})
sns.catplot(data=grouped_data, x='age_category', y='claim_amount',␣
 ↪hue='gender', col='fraudulent', kind='bar')
```

[306]:
```
<seaborn.axisgrid.FacetGrid at 0x22b556ca670>
```



Based on the conclusions from exploratory analysis as well as suitable statistical tests, answer the below questions. Please include a detailed write-up on the parameters taken into consideration, the Hypothesis testing steps, conclusion from the p-values and the business implications of the statements.

[237]:
```
Cust_claims1 = Cust_claims.copy()
```

[239]:
```
Cust_claims1 = Cust_claims1.drop('Month_name', axis =1)
```

[240]:
```
Cust_claims1 = Cust_claims1.drop('Contact', axis =1)
Cust_claims1 = Cust_claims1.drop('State', axis =1)
```

[241]:
```
Cust_claims1.columns
```

[241]:
```
Index(['claim_id', 'customer_id', 'incident_cause', 'claim_date', 'claim_area',
       'police_report', 'claim_type', 'claim_amount', 'total_policy_claims',
       'fraudulent', 'gender', 'DateOfBirth', 'Segment', 'alert flag',
       'birth_Year', 'current_year', 'Age', 'age_category'],
      dtype='object')
```

[259]:
```
Cust_claims1.head()
```

[259]:

	claim_id	customer_id	incident_cause	claim_date	claim_area	\
0	54004764	21868593	1	2017-11-27	1	
1	33985796	75740424	2	2018-10-03	2	
2	53522022	30308357	4	2018-02-02	1	
3	13015401	47830476	3	2018-06-17	1	
4	22890252	19269962	2	2018-01-13	1	

	police_report	claim_type	claim_amount	total_policy_claims	fraudulent	\

```
0              2           1        2980.0                  1.0           0
1              3           1        2980.0                  3.0           0
2              2           1        3369.5                  1.0           1
3              2           1        1680.0                  1.0           0
4              2           1        2680.0                  1.0           0

   gender DateOfBirth  Segment  alert flag  birth_Year  current_year  Age  \
0       1  1979-01-12        1           0        1979          2023   44
1       1  1970-01-13        3           1        1970          2023   53
2       1  1984-03-11        3           0        1984          2023   39
3       1  1986-05-01        3           0        1986          2023   37
4       0  1977-05-13        2           0        1977          2023   46

   age_category
0           2.0
1           2.0
2           2.0
3           2.0
4           2.0
```

[242]:
```
Cust_claims1['incident_cause'] = Cust_claims1['incident_cause'].map({'Driver␣
 ↪error' : 1, 'Crime' : 2, 'Natural causes': 3,  'Other driver error':␣
 ↪4,'Other causes':5})
Cust_claims1['claim_area'] = Cust_claims1['claim_area'].map({'Auto' : 1, 'Home'␣
 ↪:2})
Cust_claims1['police_report'] = Cust_claims1['police_report'].map({'Yes':␣
 ↪1,'No': 2,  'Unknown': 3  })
Cust_claims1['claim_type'] = Cust_claims1['claim_type'].map({'Material only':␣
 ↪1,'Injury only':2, 'Material and injury':3 })
Cust_claims1['fraudulent'] = Cust_claims1['fraudulent'].map({'Yes': 1, 'No' :␣
 ↪0})
Cust_claims1['gender'] = Cust_claims1['gender'].map({'Female':1, 'Male': 0})
Cust_claims1['Segment'] = Cust_claims1['Segment'].map({'Platinum': 1,'Gold':2,␣
 ↪'Silver':3})
Cust_claims1['age_category'] = Cust_claims1['age_category'].map({'Youth':1,␣
 ↪'Adult':2,'Senior':3})
```

[245]:
```
Cust_claims1.dtypes
```

[245]:
```
claim_id                      int64
customer_id                   int64
incident_cause                int64
claim_date           datetime64[ns]
claim_area                    int64
police_report                 int64
claim_type                    int64
```

```
claim_amount                  float64
total_policy_claims           float64
fraudulent                      int64
gender                          int64
DateOfBirth            datetime64[ns]
Segment                         int64
alert flag                      int64
birth_Year                      int64
current_year                    int64
Age                             int64
age_category                  float64
dtype: object
```

1.0.16 16.Is there any similarity in the amount claimed by males and females?

Independent sample ttest

[256]:
```
#Take two samples   0-male, 1- female
variable_name = 'claim_amount'
male = Cust_claims1.loc[Cust_claims1.gender == 0, variable_name]
female= Cust_claims1.loc[Cust_claims1.gender == 1, variable_name]

# display the mean ammount
print('mean spend of male:', male.mean() )
print('mean spend of female:', female.mean() )
```

```
mean spend of male: 12765.032998308823
mean spend of female: 12100.717432755697
```

[257]:
```
# H0: There is no similarity in the amount claimed by males and females
# Ha: There is similarity in the amount claimed by males and females
# CI: 99%, p: 0.01

# perform the test
stats.ttest_ind( male, female )
```

[257]:
```
Ttest_indResult(statistic=0.821612848932144, pvalue=0.4114767802889234)
```

1.1 Conclusion

Since the calculated p_value is greater than 0.01, we fail to reject Null Hypothesis.

we can claim that there is no similarity in the amount claims made by males and females

[286]:
```
np.var(female)
```

[286]:
```
172483455.8140038
```

1.1.1 17.Is there any relationship between age category and segment?

Chi square test

[276]:
```
Cust_claims1[['Segment', 'age_category']].head()
```

[276]:
```
   Segment  age_category
0        1           2.0
1        3           2.0
2        3           2.0
3        3           2.0
4        2           2.0
```

[277]:
```
# get the ob_freq_table from the dataset
obs_freq = pd.crosstab( Cust_claims1.age_category, Cust_claims1.Segment )
obs_freq
```

[277]:
```
Segment          1    2    3
age_category
1.0             69   71   68
2.0            266  272  257
3.0             26   43   21
```

[284]:
```
# Ho: There is no relationship between age category and segment
# Ha: There is a relationship between age category and segment
# CI: 95%, p: 0.05

# perform the test
stats.chi2_contingency( obs_freq )
```

[284]:
```
(6.979146873118188,
 0.13699432980276435,
 4,
 array([[ 68.6989936 ,  73.45654163,  65.84446478],
        [262.57548033, 280.75937786, 251.66514181],
        [ 29.72552608,  31.78408051,  28.49039341]]))
```

1.2 Conclusion

The calculated p_value is 0.1369 which is greater than the defined p_value which 0.05. So, We fail to reject null hypothesis and can claim there is no relationship between age category and segment

1.2.1 18.The current year has shown a significant rise in claim amounts as compared to 2016-17 fiscal average which was $10,000.

One sample ttest

[282]:
```
# sample to be considered
variable_name = 'claim_amount'

# mean values to be compared
pop_mean = 10000
sample = Cust_claims1.loc[:,variable_name]
mean_sample = sample.mean()

# display the means
print('population mean: ', pop_mean, '| sample mean: ', mean_sample )
```

```
population mean:  10000 | sample mean:  12444.72714007783
```

[ ]:
```
# Defining the hypothesis
#H0: The mean claim amount in the current year is equal to $10,000.
#Ha: The mean claim amount in the current year is significantly different from␣
 ↪$10,000
# CI: 99%, p: 0.01
```

[283]:
```
# perform the test
stats.ttest_1samp(a = sample, popmean = pop_mean)
```

[283]:
```
Ttest_1sampResult(statistic=6.051937772221075, pvalue=1.9644042804907075e-09)
```

1.3 Conclusion

since the calculate p value is lesser than the defined p value, we fail to reject the alternate hypothesis

We can conclude the mean claim amount in the current year is significantly different from $10,000

1.3.1 19.Is there any difference between age groups and insurance claims?

Annova test

[263]:
```
Cust_claims1['age_category'].unique()
```

[263]:
```
array([2., 1., 3.])
```

[262]:
```
Cust_claims1['age_category'].value_counts()
```

[262]:
```
2.0    795
1.0    208
3.0     90
Name: age_category, dtype: int64
```

[272]:
```
2
variable_name = 'claim_amount'
```

```
# filter the data based on segments
Youth = Cust_claims1.loc[ Cust_claims1['age_category']== 1, variable_name ]
Adult = Cust_claims1.loc[ Cust_claims1['age_category'] == 2, variable_name ]
Senior = Cust_claims1.loc[ Cust_claims1['age_category'] == 3, variable_name ]

# display the mean of the three sample
print( 'mean of Youth: ', Youth.mean(), '| mean of Adult: ', Adult.mean(), '|␣
 ↪mean of Senior: ', Senior.mean() )
```

```
mean of Youth:  11846.781895390603 | mean of Adult:  12649.412343073196 | mean
of Senior:  12018.592412451366
```

[273]:
```
#Ho: There is no relationship between age group and insurance claimed
#Ha: There is relationship between age group and insurance claimed
# CI: 95%, p: 0.05

# perform the test
stats.f_oneway( Youth, Adult, Senior )
```

[273]:
```
F_onewayResult(statistic=0.34725221644722964, pvalue=0.7067052264087558)
```

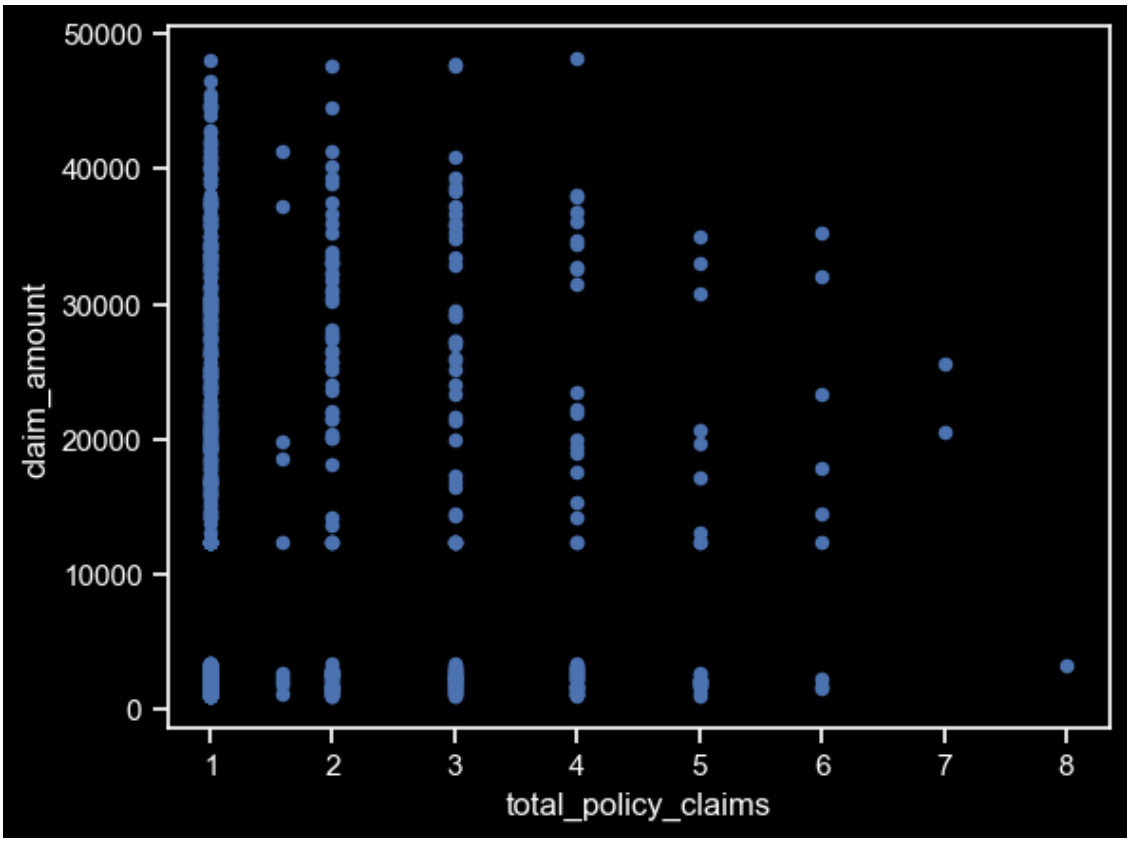
1.4 conclusion

Since the calculate p value(0.70) is greater than the defined p value(0.05)We fail to reject the null hypothesis and can claim there is no relationship between age group and insurance claimed

1.4.1 20.Is there any relationship between total number of policy claims and the claimed amount?

[307]:
```
plt.style.use('dark_background')
plt.rcParams.update({'text.color': 'white'})
Cust_claims1.plot( kind = 'scatter', x ='total_policy_claims', y =␣
 ↪'claim_amount' )
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
```



[253]:
```
# Ho: There is no relationship b/w the total number of policy claimed and the claim amount
# Ha:There is relationship b/w the total number of policy claimed and the claim amount
# CI: 95%, p: 0.05

# perform the test
stats.pearsonr(Cust_claims1['total_policy_claims'],Cust_claims1['claim_amount'])
```

[253]:
```
PearsonRResult(statistic=-0.016409477672452967, pvalue=0.5878721766398625)
```

1.5 conclusion

We fail to reject null hypothesis and because of the neagative value we can claim its negatively correlated

There is no relationsip between Number of policies claimed and the amount claimed

[ ]: