# Project Report

# Insurance claims Project

- This project consists of different statistics Hypothesis related to differe insrurance claims and categories of customers
- Raw data has been filtered and generated some visual graphs and statistical values

# Given data

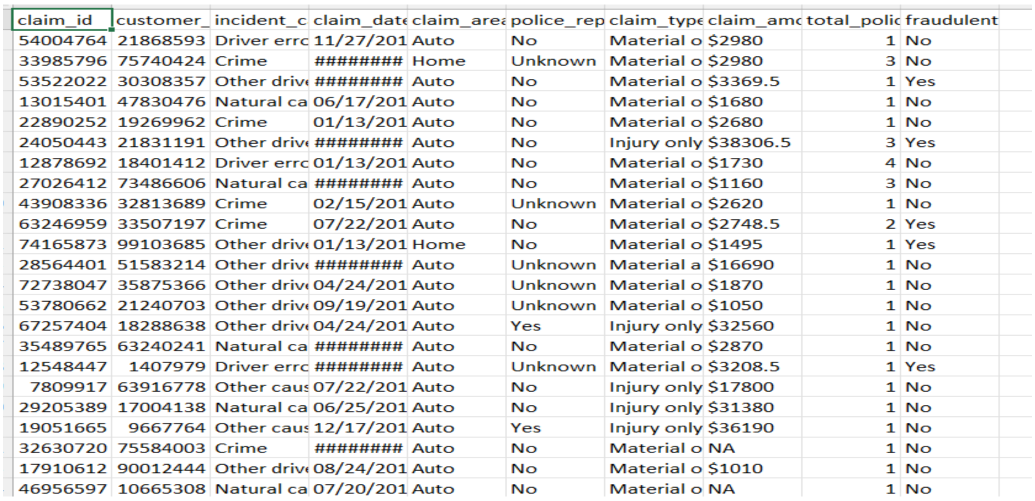
There are 2 csv files.

1.claims.csv

2.cust_demographics.csv

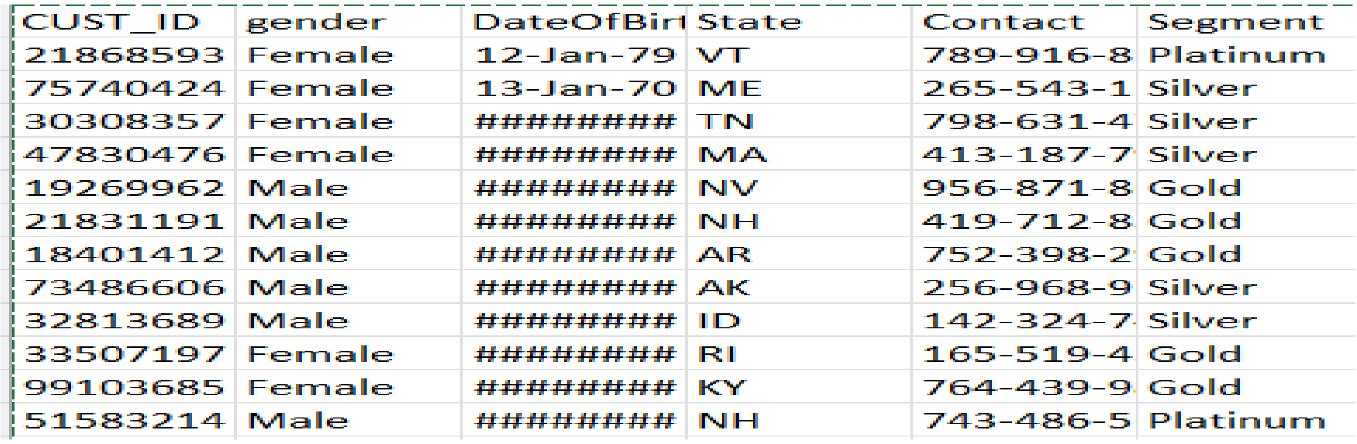
As shown in figs.

1.Claims.csv



| claim_id | customer_ | incident_c | claim_date | claim_area | police_rep | claim_type | claim_amc | total_polic | fraudulent |
|---|---|---|---|---|---|---|---|---|---|
| 54004764 | 21868593 | Driver errc | 11/27/201 | Auto | No | Material o | $2980 | 1 | No |
| 33985796 | 75740424 | Crime | ######## | Home | Unknown | Material o | $2980 | 3 | No |
| 53522022 | 30308357 | Other driv | ######## | Auto | No | Material o | $3369.5 | 1 | Yes |
| 13015401 | 47830476 | Natural ca | 06/17/201 | Auto | No | Material o | $1680 | 1 | No |
| 22890252 | 19269962 | Crime | 01/13/201 | Auto | No | Material o | $2680 | 1 | No |
| 24050443 | 21831191 | Other driv | ######## | Auto | No | Injury only | $38306.5 | 3 | Yes |
| 12878692 | 18401412 | Driver errc | 01/13/201 | Auto | No | Material o | $1730 | 4 | No |
| 27026412 | 73486606 | Natural ca | ######## | Auto | No | Material o | $1160 | 3 | No |
| 43908336 | 32813689 | Crime | 02/15/201 | Auto | Unknown | Material o | $2620 | 1 | No |
| 63246959 | 33507197 | Crime | 07/22/201 | Auto | No | Material o | $2748.5 | 2 | Yes |
| 74165873 | 99103685 | Other driv | 01/13/201 | Home | No | Material o | $1495 | 1 | Yes |
| 28564401 | 51583214 | Other driv | ######## | Auto | Unknown | Material a | $16690 | 1 | No |
| 72738047 | 35875366 | Other driv | 04/24/201 | Auto | Unknown | Material o | $1870 | 1 | No |
| 53780662 | 21240703 | Other driv | 09/19/201 | Auto | Unknown | Material o | $1050 | 1 | No |
| 67257404 | 18288638 | Other driv | 04/24/201 | Auto | Yes | Injury only | $32560 | 1 | No |
| 35489765 | 63240241 | Natural ca | ######## | Auto | No | Material o | $2870 | 1 | No |
| 12548447 | 1407979 | Driver errc | ######## | Auto | Unknown | Material o | $3208.5 | 1 | Yes |
| 7809917 | 63916778 | Other caus | 07/22/201 | Auto | No | Injury only | $17800 | 1 | No |
| 29205389 | 17004138 | Natural ca | 06/25/201 | Auto | No | Injury only | $31380 | 1 | No |
| 19051665 | 9667764 | Other caus | 12/17/201 | Auto | Yes | Injury only | $36190 | 1 | No |
| 32630720 | 75584003 | Crime | ######## | Auto | No | Material o | NA | 1 | No |
| 17910612 | 90012444 | Other driv | 08/24/201 | Auto | No | Material o | $1010 | 1 | No |
| 46956597 | 10665308 | Natural ca | 07/20/201 | Auto | No | Material o | NA | 1 | No |

2..cust_demographics.csv



| CUST_ID | gender | DateOfBirt | State | Contact | Segment |
|---|---|---|---|---|---|
| 21868593 | Female | 12-Jan-79 | VT | 789-916-8 | Platinum |
| 75740424 | Female | 13-Jan-70 | ME | 265-543-1 | Silver |
| 30308357 | Female | ######### | TN | 798-631-4 | Silver |
| 47830476 | Female | ######### | MA | 413-187-7 | Silver |
| 19269962 | Male | ######### | NV | 956-871-8 | Gold |
| 21831191 | Male | ######### | NH | 419-712-8 | Gold |
| 18401412 | Male | ######### | AR | 752-398-2 | Gold |
| 73486606 | Male | ######### | AK | 256-968-9 | Silver |
| 32813689 | Male | ######### | ID | 142-324-7 | Silver |
| 33507197 | Female | ######### | RI | 165-519-4 | Gold |
| 99103685 | Female | ######### | KY | 764-439-9 | Gold |
| 51583214 | Male | ######### | NH | 743-486-5 | Platinum |

# Importing required packages & data files

- Here I impoted the required packages like pandas, numpy and other libraries
- Then I extract the two csv files which is the raw data we use in this project.

### Importing packages:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
from datetime import timedelta
%matplotlib inline
import seaborn as sns
from scipy import stats
```

### Extracting Files:

```
df_in= pd.read_csv("claims.csv")
df_cust= pd.read_csv("cust_demographics.csv")
```

# Combining the two files into a single dataset

There are two csv files to perform required computations we have to combine them as single data set so I merge these files through “inner join” to reduce the number of null values and duplicaes in data.

```
df=pd.merge(right = df_in , left = df_cust,

        right_on= 'customer_id', left_on= 'CUST_ID', how = 'inner')
```

# Performed Audit for datatypes

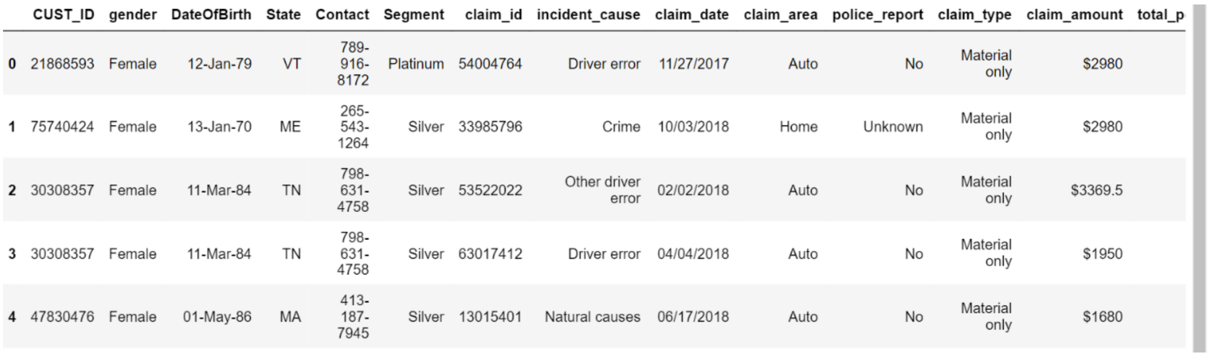
There are some columns with mismatch datatypes so I change those datatypes as suitable to perform computations in future. As shown in below,

```
df['DateOfBirth'] = pd.to_datetime(df['DateOfBirth'])

df['claim_date'] = pd.to_datetime(df['claim_date'])

df["Contact"] = pd.to_numeric(df.Contact.str.replace("-",""),downcast='float')

df["claim_amount"] = pd.to_numeric(df.claim_amount.str.replace("$",""),downcast='float
```

Now I have the datatypes of columns are as follows,

```
#   Column               Non-Null Count  Dtype

---  ------               --------------  -----
 0   CUST_ID              1085 non-null   int64
 1   gender               1085 non-null   object
 2   DateOfBirth          1085 non-null   datetime64[ns]
 3   State                1085 non-null   object
 4   Contact              1085 non-null   float32
 5   Segment              1085 non-null   object
 6   claim_id             1085 non-null   int64
 7   incident_cause       1085 non-null   object
 8   claim_date           1085 non-null   datetime64[ns]
 9   claim_area           1085 non-null   object
 10  police_report        1085 non-null   object
 11  claim_type           1085 non-null   object
 12  claim_amount         1020 non-null   float32
 13  total_policy_claims  1075 non-null   float64
 14  fraudulent           1085 non-null
```

And data as here:



| | CUST_ID | gender | DateOfBirth | State | Contact | Segment | claim_id | incident_cause | claim_date | claim_area | police_report | claim_type | claim_amount | total_p |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 21868593 | Female | 12-Jan-79 | VT | 789-916-8172 | Platinum | 54004764 | Driver error | 11/27/2017 | Auto | No | Material only | $2980 | |
| **1** | 75740424 | Female | 13-Jan-70 | ME | 265-543-1264 | Silver | 33985796 | Crime | 10/03/2018 | Home | Unknown | Material only | $2980 | |
| **2** | 30308357 | Female | 11-Mar-84 | TN | 798-631-4758 | Silver | 53522022 | Other driver error | 02/02/2018 | Auto | No | Material only | $3369.5 | |
| **3** | 30308357 | Female | 11-Mar-84 | TN | 798-631-4758 | Silver | 63017412 | Driver error | 04/04/2018 | Auto | No | Material only | $1950 | |
| **4** | 47830476 | Female | 01-May-86 | MA | 413-187-7945 | Silver | 13015401 | Natural causes | 06/17/2018 | Auto | No | Material only | $1680 | |

# ➢ Create an alert flag (1,0)

Now I generate one flag column where the another column “police_report” has  a value “yes” then flag has return “1” else ” 0”

```
df["flag"] = np.where(df.police_report == 'No',0,np.where(df.police_report == "Yes",1,np.nan))
```

# ➢ Removes duplicates

In our dataset, a single customer claimed multiple times in different categories so to reduce ambiguity I make the cust_id column unique by applying the below syntax.

```
df=df.groupby("CUST_ID").first().reset_index(drop = True)
```

```
    in the above syntax, I use first() to retain the latest entry of the
customer
```

# ➢ Handle missing values

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

```
gender                     0
DateOfBirth                0
State                      0
Contact                    0
Segment                    0
claim_id                   0
incident_cause             0
claim_date                 0
claim_area                 0
claim_type                 0
claim_amount              65
total_policy_claims       10
fraudulent                 0
flag                     292
dtype: int64
```

There are some null values in our data so I replace them by filling those null values with mode for categorical columns and mean for numeric column by using below code.

```
df["total_policy_claims"]=df["total_policy_claims"].fillna(df["total_policy_claims"].mode()[0])

df["claim_amount"] = df["claim_amount"].fillna(df["claim_amount"].mean())

df["flag"]=df["flag"].fillna(df["flag"].mode()[0])
```

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

## output:

```
gender                   0
DateOfBirth              0
State                    0
Contact                  0
Segment                  0
claim_id                 0
incident_cause           0
claim_date               0
claim_area               0
claim_type               0
claim_amount             0
total_policy_claims      0
fraudulent               0
flag                     0
dtype: int64
```

## ➢ Categorize the customers as age groups:

To categorize customers based on age first we have to calculate their age when they claim their amount so I created a column “age” by using claim_date and Date of Birth  of the customer and then I create categories by using the where clause as follows,

```
df["age"]=round((df.claim_date - df.DateOfBirth).apply(lambda a: a.days)/365.25, 0)
```

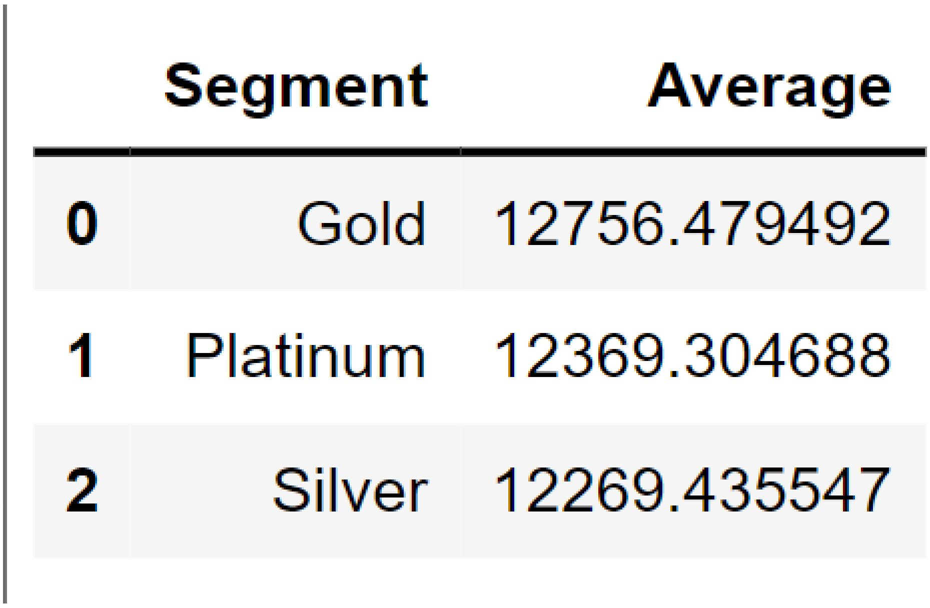
```
df['Age_group'] = np.where(df.age <18,'Childen',np.where(df.age<30,'Youth',np.where(df.age<60,'Adult','Senior')))
```

## computations I performed:

- ➢ The average amount claimed by customers from various Segments

```
df.groupby('Segment')['claim_amount'].mean()
```

output:



|   | Segment | Average |
|---|---|---|
| **0** | Gold | 12756.479492 |
| **1** | Platinum | 12369.304688 |
| **2** | Silver | 12269.435547 |

- ➢ Number of adults from TX, DE and AK claimed insurance for driver related issues

There are two conditions are

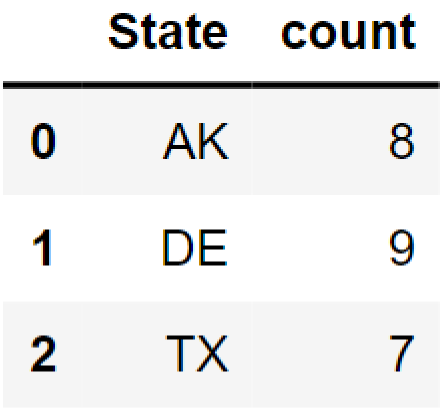
1.customer should be in state TX or DE or AK

2.insurance cause should contain string “driver”

So, I use the following syntax to find the desired output,

```
df.loc[(df.State.isin(['TX','DE','AK']))&(df.incident_cause.str.lower().str.contains("driver")) &
(df.Age_group == 'Adult')].groupby(by = "State")['Age_group'].count().reset_index(name="count"
```

output:

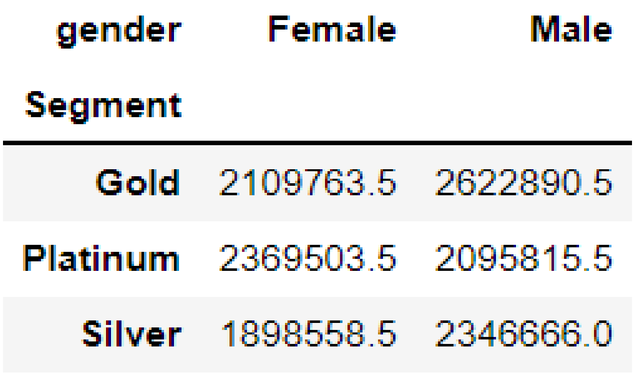


|   | State | count |
|---|---|---|
| 0 | AK | 8 |
| 1 | DE | 9 |
| 2 | TX | 7 |

- ➢ 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.

To plot pie chart I need a dataset so, I create a pivot table with index as Segment and columns as gender to get summarise of data.

```
pie_chart =pie_.pivot(index = "Segment", columns = "gender", values ="claim_amount")
```

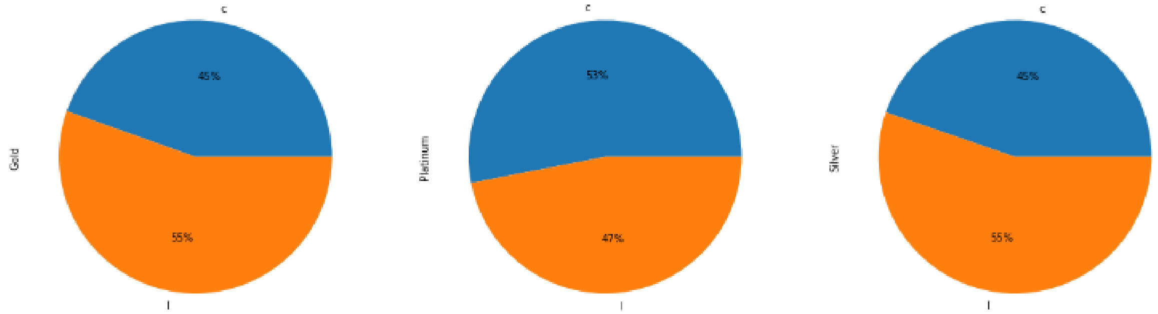


| gender | Female | Male |
|---|---|---|
| Segment | | |
| Gold | 2109763.5 | 2622890.5 |
| Platinum | 2369503.5 | 2095815.5 |
| Silver | 1898558.5 | 2346666.0 |

Ans then use plot pie chart by using following syntax,

```
pie_chart.T.plot(kind = "pie",subplots=True,legend =
False,figsize=(20,10),labels="claim_amount",autopct='%1.0f%%')

plt.show()
```

output:

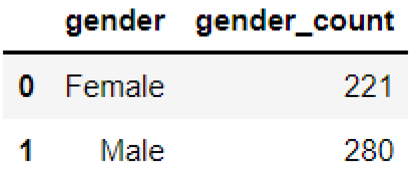


- ➢ Among males and females, which gender had claimed the most for any type of driver-related issues? E.g.This metric can be compared using a bar chart

To plot a bar graph I create a new data table with gender and the count then I use the following syntax,

```
bar_=df.loc[(df.incident_cause.str.lower().str.contains("driver"))].groupby("gender")[["gender"]].count().add_suffix("_count").reset_index()
```

output:

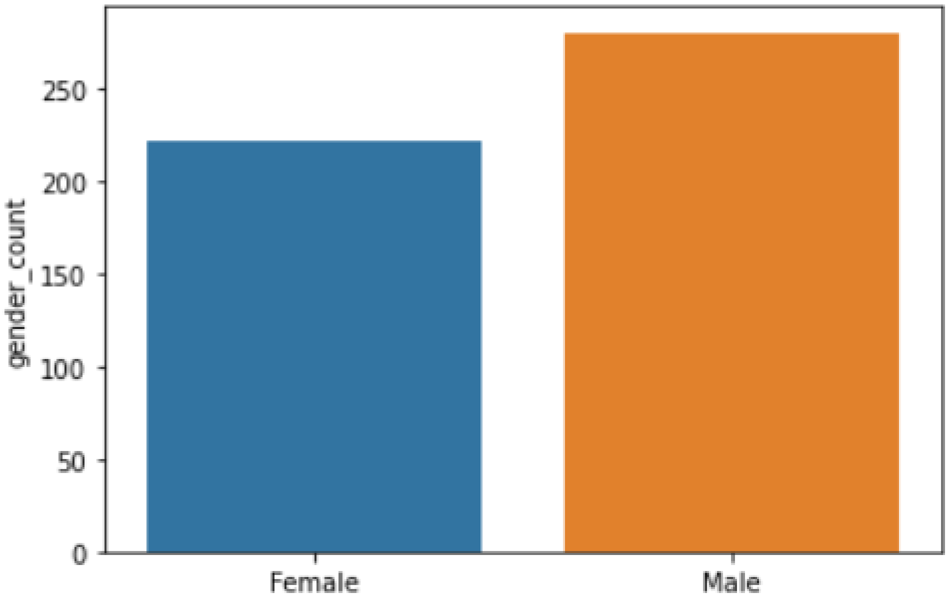


|   | gender | gender_count |
|---|---|---|
| 0 | Female | 221 |
| 1 | Male | 280 |

Syntax for plot bar graph

```
sns.barplot(x = 'gender',
        y = 'gender_count',
        data = bar_)
plt.show()
```

output:



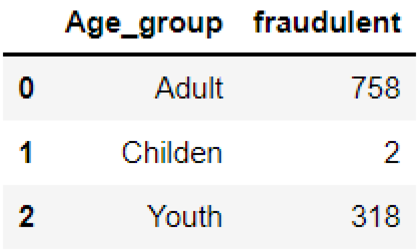
- Which age group had the maximum fraudulent policy claims? Visualize it on a bar chart.

To plot bar-graph I create new data set with columns Age_group and Fraudulent by following syntax,

Syntax:

```
bar_fraud = df.groupby("Age_group")[["fraudulent"]].count().reset_index()
bar_fraud
```

output:



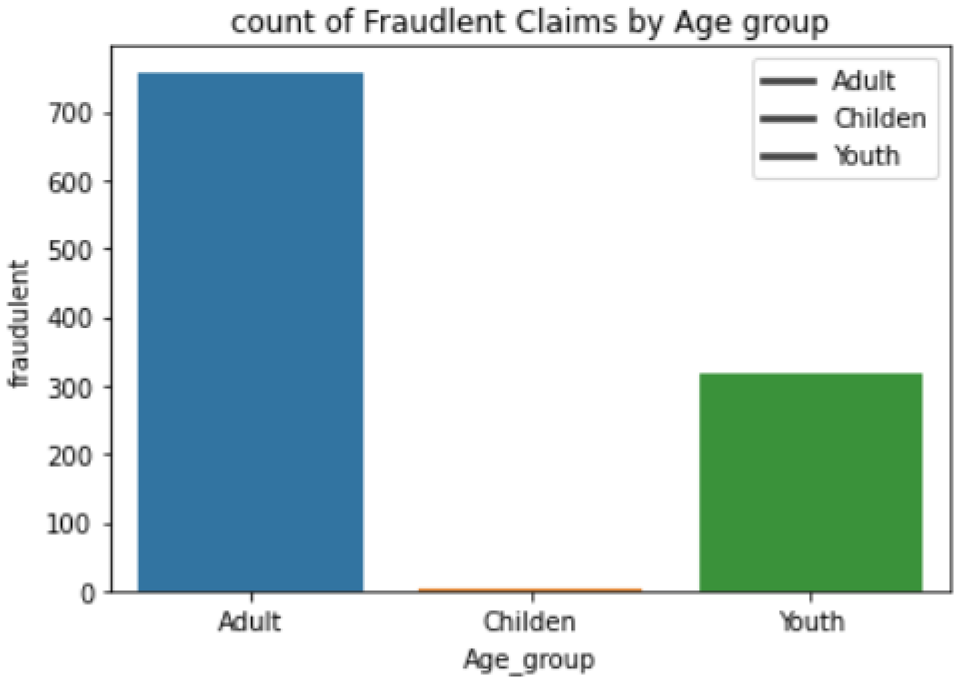
| | Age_group | fraudulent |
|---|---|---|
| **0** | Adult | 758 |
| **1** | Childen | 2 |
| **2** | Youth | 318 |

Now ploting the bar graph by wising following syntax,

## Syntax:

```
sns.barplot(x='Age_group',y='fraudulent',data=bar_fraud)

plt.legend(['Adult','Childen','Youth'])

plt.title('count of Fraudlent Claims by Age group')

plt.show()
```

## output:



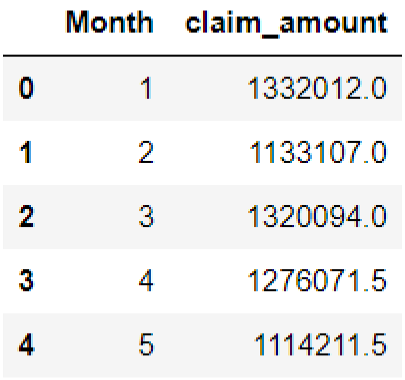
- Visualize the monthly trend of the total amount that has been claimed by the customers. Ensure that on the “month” axis, the month is in a chronological order not alphabetical order.

To plot trendline I create new data table by using following syntax,

Syntax:

```
monthly_claims=
pd.DataFrame(df.groupby(df['claim_date'].dt.month).claim_amount.sum().reset_index())

monthly_claims.columns = ['Month','claim_amount']

monthly_claims.head()
```

output:



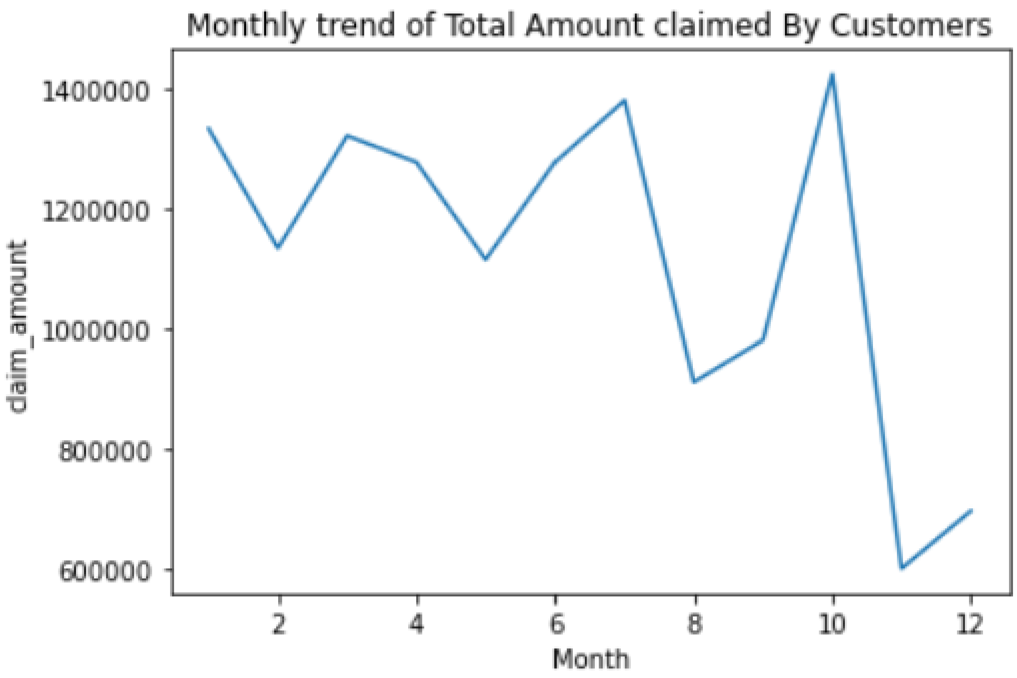
|   | Month | claim_amount |
|---|---|---|
| **0** | 1 | 1332012.0 |
| **1** | 2 | 1133107.0 |
| **2** | 3 | 1320094.0 |
| **3** | 4 | 1276071.5 |
| **4** | 5 | 1114211.5 |

To plot trend line I use lineplot as follows,

Syntax:

```
sns.lineplot(x='Month',y='claim_amount',data=monthly_claims)

plt.ticklabel_format(style = 'plain',axis = 'y')

plt.title('Monthly trend of Total Amount claimed By Customers')

plt.show()
```

output:



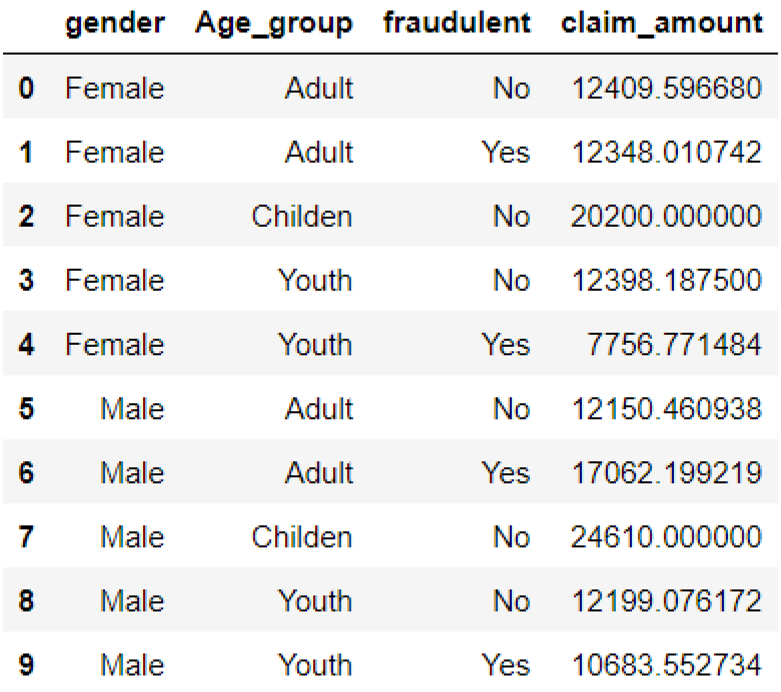
- What is the average claim amount for gender and age categories and suitably represent the above using a facetted bar chart, one facet that represents fraudulent claims and the other for non-fraudulent claims.

To make facetted bar chart I created following table by using the data in the above question with below syntax

Syntax:

```
avg_claim_amt= df.groupby(['gender','Age_group','fraudulent']).claim_amount.mean().reset_index()

avg_claim_amt
```

output:



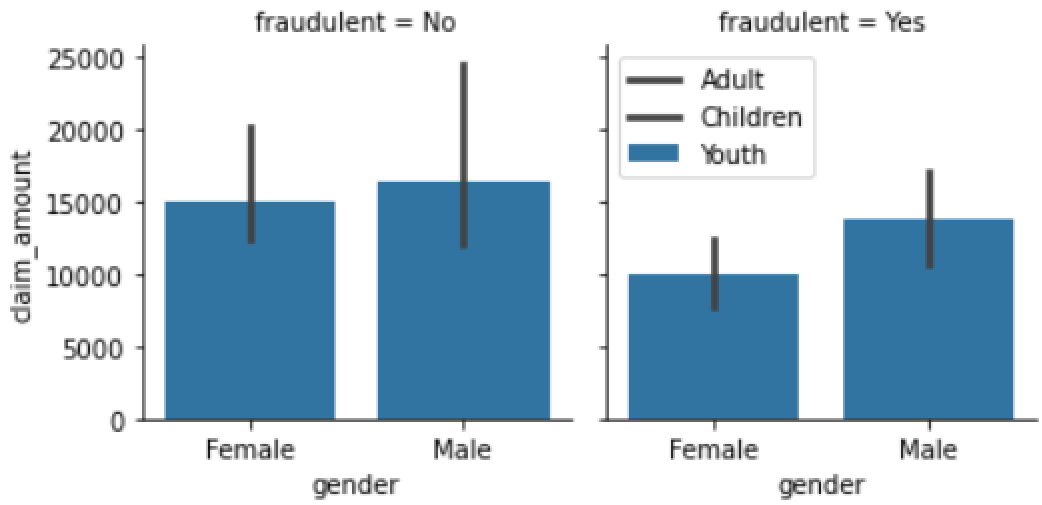
|   | gender | Age_group | fraudulent | claim_amount |
|---|---|---|---|---|
| **0** | Female | Adult | No | 12409.596680 |
| **1** | Female | Adult | Yes | 12348.010742 |
| **2** | Female | Childen | No | 20200.000000 |
| **3** | Female | Youth | No | 12398.187500 |
| **4** | Female | Youth | Yes | 7756.771484 |
| **5** | Male | Adult | No | 12150.460938 |
| **6** | Male | Adult | Yes | 17062.199219 |
| **7** | Male | Childen | No | 24610.000000 |
| **8** | Male | Youth | No | 12199.076172 |
| **9** | Male | Youth | Yes | 10683.552734 |

To make facetted bar chart I use syntax as follows,

## Syntax:

```
facet= sns.FacetGrid(data =avg_claim_amt, col = 'fraudulent')

facet.map(sns.barplot,'gender', 'claim_amount',hue_order =['Adult','Children','Youth'])

plt.legend(['Adult','Children','Youth'])

plt.ticklabel_format(style = 'plain',axis = 'y')

plt.show()
```

## output:



- similarity in the amount claimed by males and females

to find similarity first we have to get claim amount for both males and females.

male = df.loc[df.gender == 'Male','claim_amount']

female = df.loc[df.gender == 'Female','claim_amount']

here we have two independent samples scores they male claim amount and female claim amount so for this type of calculation to find similarity we can use T_test

“This is a test for the null hypothesis that 2 independent samples have identical average (expected) values. This test assumes that the populations have identical variances by default.”

### Syntax:

stats.ttest_ind(male,female)

### output:

```
Ttest_indResult(statistic=0.8848699385883664, pvalue=0.3764244709509263
5)
```

- relationship between age category and segment

  here we have to find the relation between two categorical columns so we have to use a crosstab table to compute the data

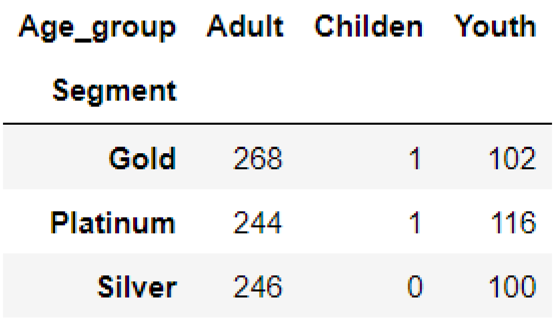
crosstab():

“This method is used to compute a simple cross-ta This method is used to compute a simple cross-tabulation of two (or more) factors. By default, computes a frequency table of the factors unless an array of values and an aggregation function are passed.”

### Syntax:

crosstab= pd.crosstab(df.Segment,df.Age_group) crosstab

output:



| Age_group | Adult | Childen | Youth |
|---|---|---|---|
| **Segment** | | | |
| **Gold** | 268 | 1 | 102 |
| **Platinum** | 244 | 1 | 116 |
| **Silver** | 246 | 0 | 100 |

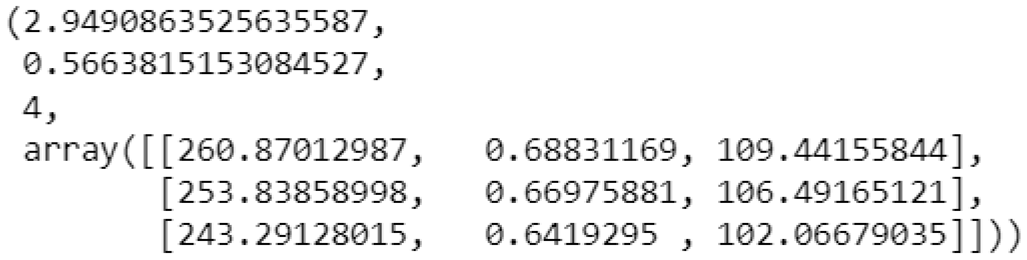
Here I support to use “chi-square test” because we have to work on cross table which is created to know the relationship between two independent categorical variables

>>>Chi-square test of independence of variables in a contingency or cross table.

Syntax for statistical summary:

stats.chi2_contingency(crosstab)

output:



```
(2.9490863525635587,
 0.5663815153084527,
 4,
 array([[260.87012987,   0.68831169, 109.44155844],
        [253.83858998,   0.66975881, 106.49165121],
        [243.29128015,   0.6419295 , 102.06679035]]))
```

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

to find max and min date I use following syntax,

Syntax:

print('Max Date:',df.claim_date.max(), '| Min Date:',df.claim_date.min())

output:

```
Max Date: 2018-10-30 00:00:00 | Min Date: 2017-01-01 00:00:00
```

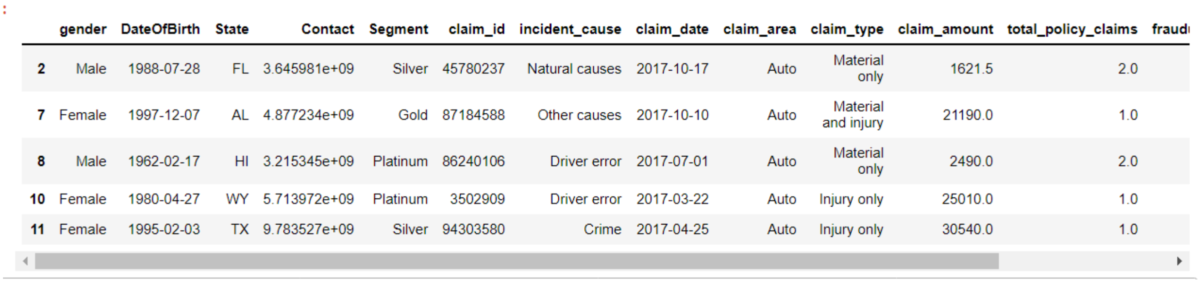
To segrigate our data from minimum date  and maximum date I use below syntax,

## Syntax:

dates = df[(df.claim_date>'2017-01-01') & (df.claim_date<'2018-01-01')]

dates.head()

## output:



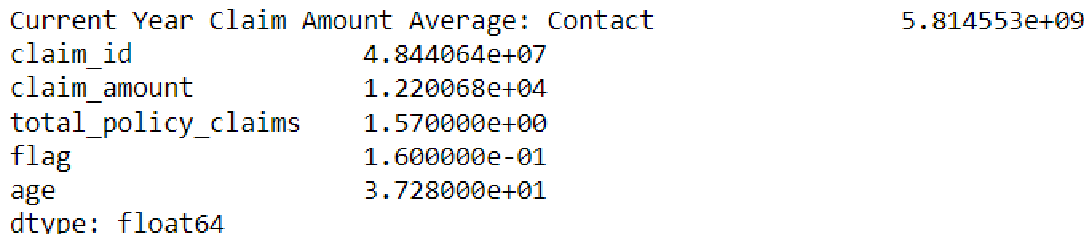
|  | gender | DateOfBirth | State | Contact | Segment | claim_id | incident_cause | claim_date | claim_area | claim_type | claim_amount | total_policy_claims | fraud |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | Male | 1988-07-28 | FL | 3.645981e+09 | Silver | 45780237 | Natural causes | 2017-10-17 | Auto | Material only | 1621.5 | 2.0 |  |
| 7 | Female | 1997-12-07 | AL | 4.877234e+09 | Gold | 87184588 | Other causes | 2017-10-10 | Auto | Material and injury | 21190.0 | 1.0 |  |
| 8 | Male | 1962-02-17 | HI | 3.215345e+09 | Platinum | 86240106 | Driver error | 2017-07-01 | Auto | Material only | 2490.0 | 2.0 |  |
| 10 | Female | 1980-04-27 | WY | 5.713972e+09 | Platinum | 3502909 | Driver error | 2017-03-22 | Auto | Injury only | 25010.0 | 1.0 |  |
| 11 | Female | 1995-02-03 | TX | 9.783527e+09 | Silver | 94303580 | Crime | 2017-04-25 | Auto | Injury only | 30540.0 | 1.0 |  |

To find average claim amunt I use the following syntax,

## Syntax:

print('Current Year Claim Amount Average:', round(dates.mean(),2))

## output:



```
Current Year Claim Amount Average: Contact                5.814553e+09
claim_id               4.844064e+07
claim_amount           1.220068e+04
total_policy_claims    1.570000e+00
flag                   1.600000e-01
age                    3.728000e+01
dtype: float64
```

## To find the statistics and p-value I use one T-test

- A one-sample t-test is used to test whether or not the mean of a population is equal to some value. The motivation for performing a one-sample t-test. The formula to perform a one-sample t-test. The assumptions that should be met to perform a one-sample t-test.

## Syntax for statistical summary:

stats.ttest_1samp(dates.claim_amount,10000)

## output:

```
Ttest_1sampResult(statistic=4.003476859412519, pvalue=7.046475870365297
e-05)
```

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

To find the mean for each category first we have to make individual age group means as follows,

## Syntax:

```
adult_amt = df.loc[df.Age_group == 'Adult','claim_amount']

children_amt = df.loc[df.Age_group == 'Childen','claim_amount']

youth_amt = df.loc[df.Age_group == 'Youth','claim_amount']

print(' mean of Adult  : ',adult_amt.mean(),'| Mean of Youth: ',youth_amt.mean() ,

    '| Mean of Children : ',children_amt.mean())
```

Output:

```
mean of Adult  :  12807.7373046875 | Mean of Youth:  11604.1796875 | Me
an of Children :  22405.0
```

here I use stats.f_oneway() to find statistical summary

**stats.f_oneway()**

The one-way ANOVA tests the null hypothesis that two or more groups have the same population mean. The test is applied to samples from two or more groups, possibly with differin g sizes

Syntax:

```
stats.f_oneway(adult_amt,youth_amt,children_amt)
```

output:

```
F_onewayResult(statistic=1.4629916552752953, pvalue=0.2320031702525453)
```

- Is there any relationship between the total number of policy claims and the claimed amount

Here I use stats.pearsonr to get appropriate statistical summary

stats.pearsonr>>>

this correlation coefficient and p-value for testing non-correlations and this correlation coefficient [1] measure the linear relationship between two datasets.

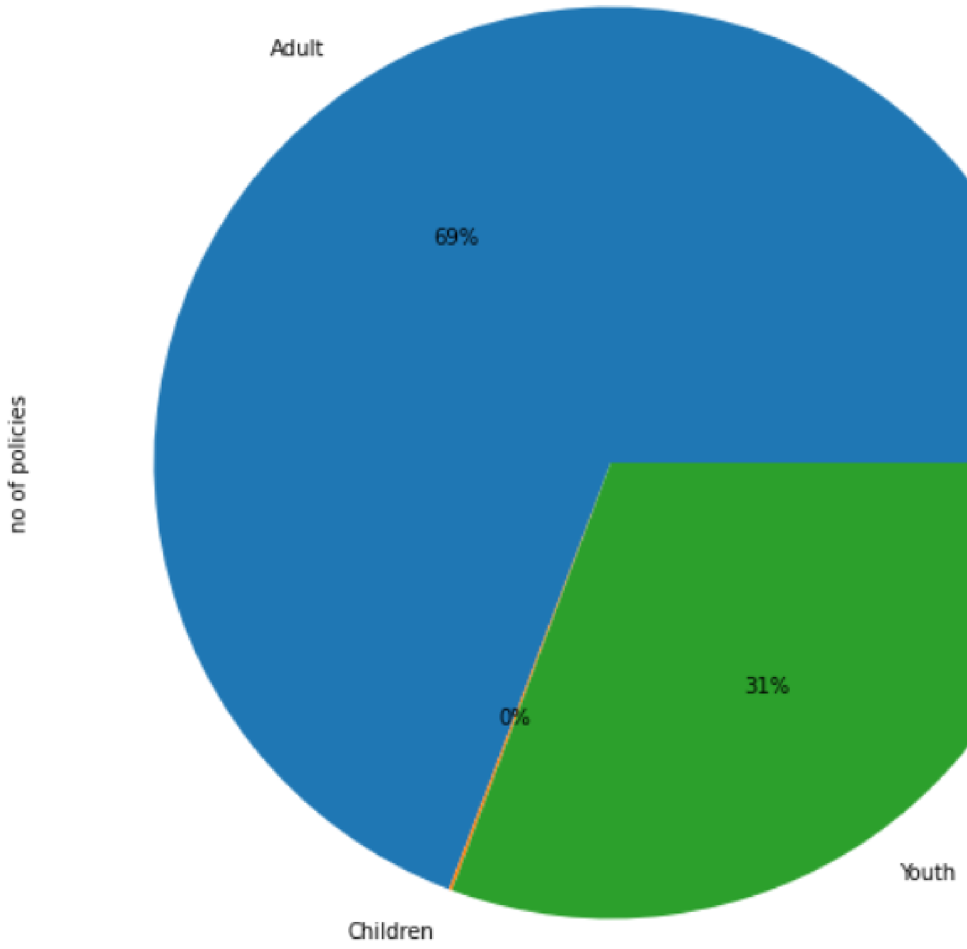
Syntax:

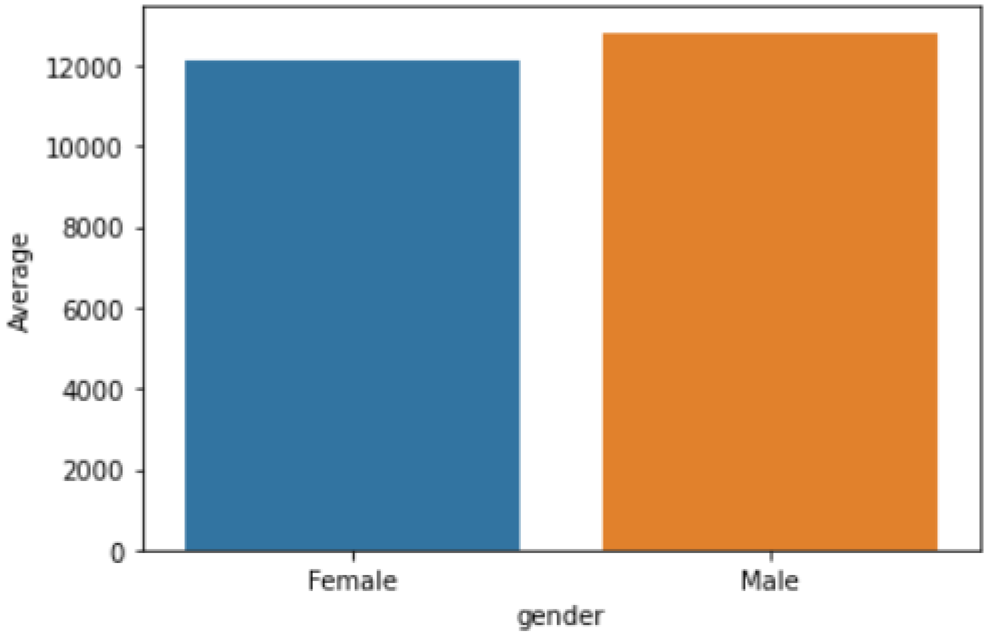
```
stats.pearsonr(df.total_policy_claims, df.claim_amount)
```

output:

```
(-0.01480092663146236, 0.6273782423038327)
```

# Some visual graphs to understand data set


Adult
69%
no of policies
0%
31%
Youth
Children


12000
10000
8000
6000
4000
2000
0
Average
Female
Male
gender