Car insurance data

Content

The columns are resembling practical world features. The outcome column indicates 1 if a customer has claimed his/her loan else 0. The data has 19 features from there 18 of them are corresponding logs which were taken by the company.

0. Initial analysis of the data

0.1 Analysis of null or not valid values

In [5]:
```
# https://www.kaggle.com/sagnik1511/car-insurance-data?select=Car_Insurance
coches=pd.read_csv('Car_Insurance_Claim.csv')
coches.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   10000 non-null  int64  
 1   AGE                  10000 non-null  object 
 2   GENDER               10000 non-null  object 
 3   RACE                 10000 non-null  object 
 4   DRIVING_EXPERIENCE   10000 non-null  object 
 5   EDUCATION            10000 non-null  object 
 6   INCOME               10000 non-null  object 
 7   CREDIT_SCORE         9018 non-null   float64
 8   VEHICLE_OWNERSHIP    10000 non-null  float64
 9   VEHICLE_YEAR         10000 non-null  object 
 10  MARRIED              10000 non-null  float64
 11  CHILDREN             10000 non-null  float64
 12  POSTAL_CODE          10000 non-null  int64  
 13  ANNUAL_MILEAGE       9043 non-null   float64
 14  VEHICLE_TYPE         10000 non-null  object 
 15  SPEEDING_VIOLATIONS  10000 non-null  int64  
 16  DUIS                 10000 non-null  int64  
 17  PAST_ACCIDENTS       10000 non-null  int64  
 18  OUTCOME              10000 non-null  float64
dtypes: float64(6), int64(5), object(8)
memory usage: 1.4+ MB
```

Null values were only found in columns 7 **(CREDIT_SCORE)** and 13 **(ANUAL_MILEAGE)**, but the other coulumns will be analyzed to search for other invalid values

0.2 Analysis of duplicated values

The only relevant column that could have duplicated values is the ID one, since there can only be one row per client

In [6]:
```
coches["ID"].nunique()
```

Out[6]: 10000

All values in the ID column are unique

0.3 Analysis of data types

Some columns will need a codification to work with them, first we check the 20 first values to try to identify patterns in data.

In [7]:
```
coches.head(20)
```

Out[7]:

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCOR
0	569520	65+	female	majority	0-9y	high school	upper class	0.62902
1	750365	16-25	male	majority	0-9y	none	poverty	0.35775
2	199901	16-25	female	majority	0-9y	high school	working class	0.49314
3	478866	16-25	male	majority	0-9y	university	working class	0.20601
4	731664	26-39	male	majority	10-19y	none	working class	0.38836
5	877557	40-64	female	majority	20-29y	high school	upper class	0.61912
6	930134	65+	male	majority	30y+	high school	upper class	0.49294
7	461006	26-39	female	majority	0-9y	university	working class	0.46868
8	68366	40-64	female	majority	20-29y	university	working class	0.52181
9	445911	40-64	female	majority	0-9y	high school	upper class	0.56153
10	275820	65+	male	majority	30y+	high school	upper class	0.62036
11	521399	65+	female	majority	30y+	high school	upper class	0.72983
12	429728	40-64	male	majority	20-29y	high school	upper class	0.63704
13	569640	16-25	female	majority	0-9y	university	upper class	0.59126
14	980181	26-39	male	majority	10-19y	high school	middle class	0.46156
15	906223	26-39	female	majority	0-9y	high school	upper class	0.76279
16	517747	65+	male	majority	30y+	university	upper class	0.79617
17	24851	16-25	male	majority	0-9y	none	poverty	Nal

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCOR
18	104086	26-39	female	majority	0-9y	university	upper class	0.68059
19	240658	16-25	female	majority	0-9y	high school	working	0.41771

First observations

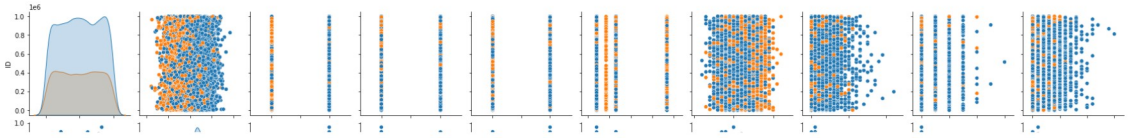
- The column AGE shoud be a numeric discrete value, but actually is a object, because it is defining a period of time.
- The GENDER could be simplified to true - false.
- The RACE can be simplifoed to a numeric value.
- DRIVING_EXPERIENCE have the same problem as AGE.
- EDUCATION and INCOME can be codified like categorical data.
- VEHICLE_YEAR looks like have only befor/after 2015.
- POSTAL_CODE is categorical data and can be codified.

0.4 Observations on a *pairplot*

A pairplot is extremely usefull to notice the most obvious caracteristics of the data

In [8]:
```
sns.pairplot(data=coches,hue="OUTCOME")
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x16606dda190>
```



Conclusions of the pairplot

- Most of the categorical data has been detected as numerical due a bad declaration, this must be fixed. - Some numerical columns dor appear (AGE or DRIVING_EXPERIENCE) because bad declaration.

1. Data cleaning and orgianzation

Every freature will be analize to ensure data consistency.

1.1 AGE encoding

1. We discover the number or classes of the age ranges.
2. Age ranges are encoded into categorical-ordinal data.

In [9]:
```
coches["AGE"].unique()
```

Out[9]:
```
array(['65+', '16-25', '26-39', '40-64'], dtype=object)
```

4 age ranges were found, which means we are going to create 4 categorical classes to organize the data.

In [10]:
```
# From now the modified dataframe will be cdf, so we can make comparations
cdf = coches

cdf['AGE'] = pd.Categorical(
    cdf['AGE'],
    categories=['16-25','26-39','40-64','65+'],
    ordered=True
)

cdf.head(10)
```

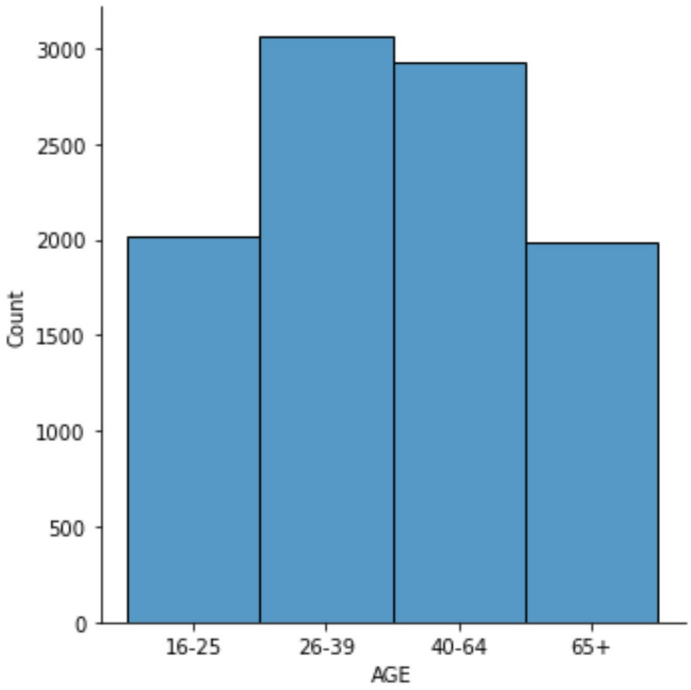
Out[10]:

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE
0	569520	65+	female	majority	0-9y	high school	upper class	0.629027
1	750365	16-25	male	majority	0-9y	none	poverty	0.357757
2	199901	16-25	female	majority	0-9y	high school	working class	0.493146
3	478866	16-25	male	majority	0-9y	university	working class	0.206013
4	731664	26-39	male	majority	10-19y	none	working class	0.388366

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE
5	877557	40-64	female	majority	20-29y	high school	upper class	0.619127
6	930134	65+	male	majority	30y+	high school	upper class	0.492944
7	461006	26-39	female	majority	0-9y	university	working class	0.468689
8	68366	40-64	female	majority	20-29y	university	working class	0.521815

In [11]:
```
sns.displot(cdf['AGE'])
```

Out[11]:
```
<seaborn.axisgrid.FacetGrid at 0x16606dcd760>
```



In [12]:
```
cdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   10000 non-null  int64
 1   AGE                  10000 non-null  category
 2   GENDER               10000 non-null  object
 3   RACE                 10000 non-null  object
 4   DRIVING_EXPERIENCE   10000 non-null  object
 5   EDUCATION            10000 non-null  object
 6   INCOME               10000 non-null  object
 7   CREDIT_SCORE         9018 non-null   float64
 8   VEHICLE_OWNERSHIP    10000 non-null  float64
 9   VEHICLE_YEAR         10000 non-null  object
 10  MARRIED              10000 non-null  float64
 11  CHILDREN             10000 non-null  float64
 12  POSTAL_CODE          10000 non-null  int64
 13  ANNUAL_MILEAGE       9043 non-null   float64
 14  VEHICLE_TYPE         10000 non-null  object
```

```
 15  SPEEDING_VIOLATIONS  10000 non-null  int64
 16  DUIS                 10000 non-null  int64
 17  PAST_ACCIDENTS       10000 non-null  int64
 18  OUTCOME              10000 non-null  float64
dtypes: category(1), float64(6), int64(5), object(7)
memory usage: 1.4+ MB
```

Once clasified as categorical/ordinal we can see the distribution ordered by age range, and now Pandas "knows" AGE is a category, not a object.

1.2 GENDER encoding

To save space GENDER will be converted into m/f values.

In [13]:
```
coches["GENDER"].unique()
```

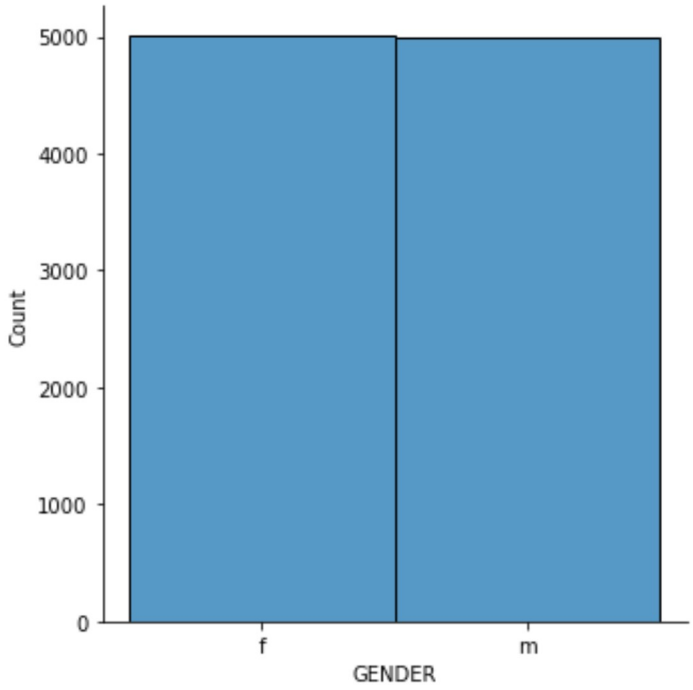
Out[13]:
```
array(['female', 'male'], dtype=object)
```

We confirm there are only 2 vaues.

In [14]:
```
cdf["GENDER"]=cdf["GENDER"].replace({'female':'f','male':'m'})
```

In [15]:
```
sns.displot(cdf["GENDER"])
```

Out[15]:
```
<seaborn.axisgrid.FacetGrid at 0x16610729b80>
```



In [16]:
```
cdf.head(10)
```

Out[16]:

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE
0	569520	65+	f	majority	0-9y	high school	upper class	0.629027

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE
1	750365	16-25	m	majority	0-9y	none	poverty	0.357757
2	199901	16-25	f	majority	0-9y	high school	working class	0.493146
3	478866	16-25	m	majority	0-9y	university	working class	0.206013
4	731664	26-39	m	majority	10-19y	none	working class	0.388366
5	877557	40-64	f	majority	20-29y	high school	upper class	0.619127
6	930134	65+	m	majority	30y+	high school	upper class	0.492944
7	461006	26-39	f	majority	0-9y	university	working class	0.468689
8	68366	40-64	f	majority	20-29y	university	working class	0.521815

1.3 RACE encoding

Similar transformation as GENDER

In [17]:
```
coches["RACE"].unique()
```

Out[17]:
```
array(['majority', 'minority'], dtype=object)
```

We discovered that there are only 2 values, which do not give any information from the label, but since the author of the dataset is from West Bengal, India, we have some extra information.

In [18]:
```
sns.displot(cdf['RACE'])
```

Out[18]:
```
<seaborn.axisgrid.FacetGrid at 0x16606dd64c0>
```

1.4 DRIVING_EXPERIENCE encodign

Similar transformation as AGE

In [19]:
```
coches["DRIVING_EXPERIENCE"].unique()
```

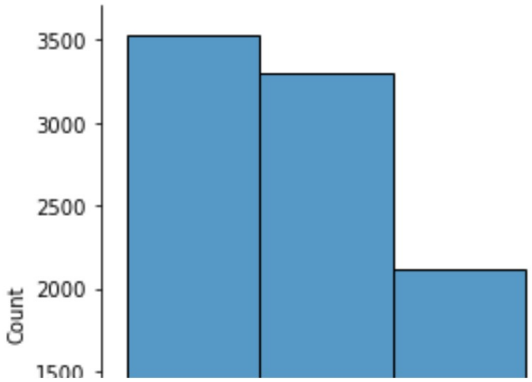
Out[19]:
```
array(['0-9y', '10-19y', '20-29y', '30y+'], dtype=object)
```

In [20]:
```
cdf["DRIVING_EXPERIENCE"] = pd.Categorical(
    cdf["DRIVING_EXPERIENCE"],
    categories=['0-9y','10-19y','20-29y','30y+'],
    ordered=True
)
cdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype   
---  ------               --------------  -----   
 0   ID                   10000 non-null  int64   
 1   AGE                  10000 non-null  category
 2   GENDER               10000 non-null  object  
 3   RACE                 10000 non-null  object  
 4   DRIVING_EXPERIENCE   10000 non-null  category
 5   EDUCATION            10000 non-null  object  
 6   INCOME               10000 non-null  object  
 7   CREDIT_SCORE         9018 non-null   float64 
 8   VEHICLE_OWNERSHIP    10000 non-null  float64 
 9   VEHICLE_YEAR         10000 non-null  object  
 10  MARRIED              10000 non-null  float64 
 11  CHILDREN             10000 non-null  float64 
 12  POSTAL_CODE          10000 non-null  int64   
 13  ANNUAL_MILEAGE       9043 non-null   float64 
 14  VEHICLE_TYPE         10000 non-null  object  
 15  SPEEDING_VIOLATIONS  10000 non-null  int64   
 16  DUIS                 10000 non-null  int64   
 17  PAST_ACCIDENTS       10000 non-null  int64   
 18  OUTCOME              10000 non-null  float64 
dtypes: category(2), float64(6), int64(5), object(6)
memory usage: 1.3+ MB
```

In [21]:
```
sns.displot(cdf["DRIVING_EXPERIENCE"])
```

Out[21]:
```
<seaborn.axisgrid.FacetGrid at 0x16611451ac0>
```



1.5 EDUCATION

We are going to order the education depending on the level.

In [22]:
```
coches["EDUCATION"].unique()
```

Out[22]:
```
array(['high school', 'none', 'university'], dtype=object)
```

In [23]:
```
cdf["EDUCATION"]=cdf["EDUCATION"].replace({'none':'n','high school':'h','ur

cdf["EDUCATION"] = pd.Categorical(
    cdf["EDUCATION"],
    categories=['n','h','u'],
    ordered=True
)
cdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   10000 non-null  int64
 1   AGE                  10000 non-null  category
 2   GENDER               10000 non-null  object
 3   RACE                 10000 non-null  object
 4   DRIVING_EXPERIENCE   10000 non-null  category
 5   EDUCATION            10000 non-null  category
 6   INCOME               10000 non-null  object
 7   CREDIT_SCORE         9018 non-null   float64
 8   VEHICLE_OWNERSHIP    10000 non-null  float64
 9   VEHICLE_YEAR         10000 non-null  object
 10  MARRIED              10000 non-null  float64
 11  CHILDREN             10000 non-null  float64
 12  POSTAL_CODE          10000 non-null  int64
 13  ANNUAL_MILEAGE       9043 non-null   float64
 14  VEHICLE_TYPE         10000 non-null  object
 15  SPEEDING_VIOLATIONS  10000 non-null  int64
 16  DUIS                 10000 non-null  int64
 17  PAST_ACCIDENTS       10000 non-null  int64
 18  OUTCOME              10000 non-null  float64
dtypes: category(3), float64(6), int64(5), object(5)
memory usage: 1.2+ MB
```

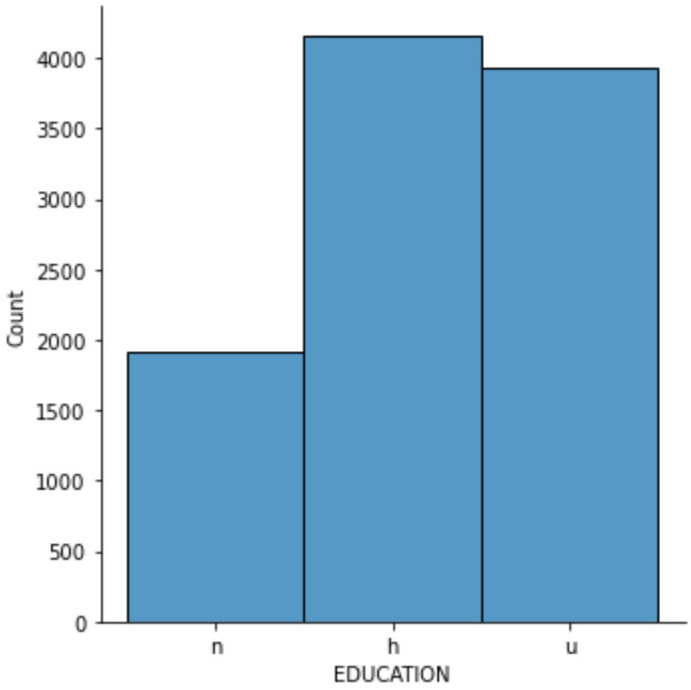
In [24]:
```
cdf.head(10)
```

Out[24]:

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE
0	569520	65+	f	majority	0-9y	h	upper class	0.629027
1	750365	16-25	m	majority	0-9y	n	poverty	0.357757
2	199901	16-25	f	majority	0-9y	h	working class	0.493146
3	478866	16-25	m	majority	0-9y	u	working class	0.206013
4	731664	26-39	m	majority	10-19y	n	working class	0.388366
5	877557	40-64	f	majority	20-29y	h	upper class	0.619127
6	930134	65+	m	majority	30y+	h	upper class	0.492944
7	461006	26-39	f	majority	0-9y	u	working class	0.468689
8	68366	40-64	f	majority	20-29y	u	working class	0.521815
9	445911	40-64	f	majority	0-9y	h	upper class	0.561531

In [25]:
```
sns.displot(cdf["EDUCATION"])
```

Out[25]:
```
<seaborn.axisgrid.FacetGrid at 0x166107189a0>
```



1.6 INCOME encoding

This freature is specially problematic, is a subjetive view of the income (we can not know the real value) so the author of the survey creates 4 categories: upper, working, middle and

poverty. The categories will be changed to poverty (p), lower (l), medium (m) and upper (u) as

In [26]:
```
coches["INCOME"].unique()
```

Out[26]:
```
array(['upper class', 'poverty', 'working class', 'middle class'],
      dtype=object)
```

In [27]:
```
cdf["INCOME"]=cdf["INCOME"].replace({'poverty':'p','working class':'l','mic

cdf["INCOME"] = pd.Categorical(
    cdf["INCOME"],
    categories=['p','l','m','u'],
    ordered=True
)
cdf.head(10)
```

Out[27]:

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE
0	569520	65+	f	majority	0-9y	h	u	0.629027
1	750365	16-25	m	majority	0-9y	n	p	0.357757
2	199901	16-25	f	majority	0-9y	h	l	0.493146
3	478866	16-25	m	majority	0-9y	u	l	0.206013
4	731664	26-39	m	majority	10-19y	n	l	0.388366
5	877557	40-64	f	majority	20-29y	h	u	0.619127
6	930134	65+	m	majority	30y+	h	u	0.492944
7	461006	26-39	f	majority	0-9y	u	l	0.468689
8	68366	40-64	f	majority	20-29y	u	l	0.521815
9	445911	40-64	f	majority	0-9y	h	u	0.561531

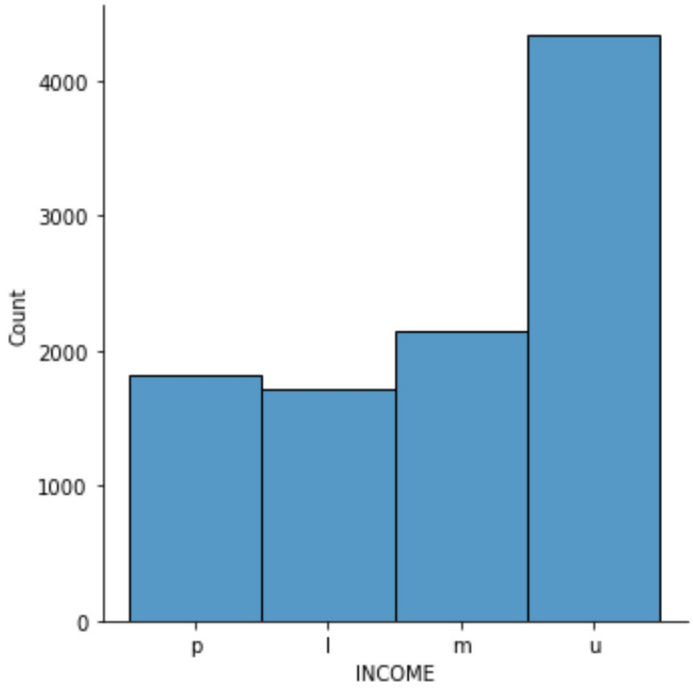
In [28]:
```
cdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   10000 non-null  int64
 1   AGE                  10000 non-null  category
 2   GENDER               10000 non-null  object
 3   RACE                 10000 non-null  object
 4   DRIVING_EXPERIENCE   10000 non-null  category
 5   EDUCATION            10000 non-null  category
 6   INCOME               10000 non-null  category
 7   CREDIT_SCORE         9018 non-null   float64
 8   VEHICLE_OWNERSHIP    10000 non-null  float64
 9   VEHICLE_YEAR         10000 non-null  object
 10  MARRIED              10000 non-null  float64
 11  CHILDREN             10000 non-null  float64
 12  POSTAL_CODE          10000 non-null  int64
 13  ANNUAL_MILEAGE       9043 non-null   float64
 14  VEHICLE_TYPE         10000 non-null  object
 15  SPEEDING_VIOLATIONS  10000 non-null  int64
 16  DUIS                 10000 non-null  int64
```

```
 17  PAST_ACCIDENTS       10000 non-null  int64
 18  OUTCOME              10000 non-null  float64
dtypes: category(4), float64(6), int64(5), object(4)
memory usage: 1.2+ MB
```

In [29]:
```
sns.displot(cdf["INCOME"])
```

Out[29]:
```
<seaborn.axisgrid.FacetGrid at 0x1661070d3a0>
```



1.7 VEHICLE_OWNERSHIP

We just have to turn the values into boolean values instead of numerical.

In [30]:
```
cdf["VEHICLE_OWNERSHIP"] = cdf["VEHICLE_OWNERSHIP"].astype('bool')
```

In [31]:
```
cdf["VEHICLE_OWNERSHIP"]
```

Out[31]:
```
0         True
1        False
2         True
3         True
4         True
         ...
9995      True
9996      True
9997      True
9998     False
9999      True
Name: VEHICLE_OWNERSHIP, Length: 10000, dtype: bool
```

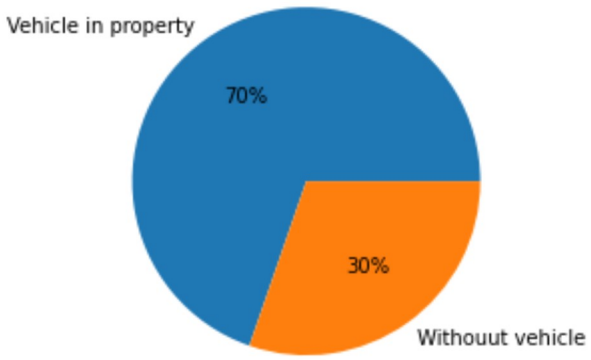
In [32]:
```
cdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
```

```
---  ------                --------------  -----
 0   ID                    10000 non-null  int64
 1   AGE                   10000 non-null  category
 2   GENDER                10000 non-null  object
 3   RACE                  10000 non-null  object
 4   DRIVING_EXPERIENCE    10000 non-null  category
 5   EDUCATION             10000 non-null  category
 6   INCOME                10000 non-null  category
 7   CREDIT_SCORE          9018 non-null   float64
 8   VEHICLE_OWNERSHIP     10000 non-null  bool
 9   VEHICLE_YEAR          10000 non-null  object
 10  MARRIED               10000 non-null  float64
 11  CHILDREN              10000 non-null  float64
 12  POSTAL_CODE           10000 non-null  int64
 13  ANNUAL_MILEAGE        9043 non-null   float64
 14  VEHICLE_TYPE          10000 non-null  object
 15  SPEEDING_VIOLATIONS   10000 non-null  int64
 16  DUIS                  10000 non-null  int64
 17  PAST_ACCIDENTS        10000 non-null  int64
 18  OUTCOME               10000 non-null  float64
dtypes: bool(1), category(4), float64(5), int64(5), object(4)
```

In [33]:

```
valores = cdf["VEHICLE_OWNERSHIP"].value_counts()
plt.pie(valores,labels=["Vehicle in property","Withouut vehicle"],autopct='
plt.show()
```



1.8 VEHICLE_YEAR

The data is unnecesary log, cars are clasified before and after 2015, so data will change to b/a.

In [34]:

```
cdf["VEHICLE_YEAR"].unique()
```

Out[34]:

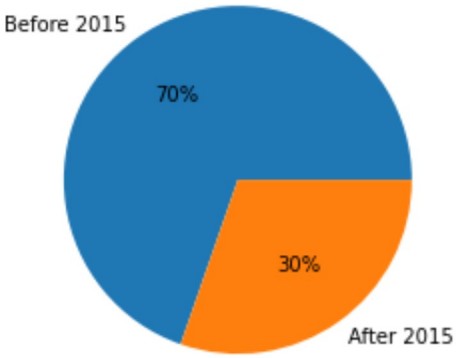
```
array(['after 2015', 'before 2015'], dtype=object)
```

In [35]:

```
cdf["VEHICLE_YEAR"]=cdf["VEHICLE_YEAR"].replace({'after 2015':'a','before 2

cdf["VEHICLE_YEAR"] = pd.Categorical(
    cdf["VEHICLE_YEAR"],
    categories=['b','a'],
    ordered=True
)
cdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   10000 non-null  int64
 1   AGE                  10000 non-null  category
 2   GENDER               10000 non-null  object
 3   RACE                 10000 non-null  object
 4   DRIVING_EXPERIENCE   10000 non-null  category
 5   EDUCATION            10000 non-null  category
 6   INCOME               10000 non-null  category
 7   CREDIT_SCORE         9018 non-null   float64
 8   VEHICLE_OWNERSHIP    10000 non-null  bool
 9   VEHICLE_YEAR         10000 non-null  category
 10  MARRIED              10000 non-null  float64
 11  CHILDREN             10000 non-null  float64
 12  POSTAL_CODE          10000 non-null  int64
 13  ANNUAL_MILEAGE       9043 non-null   float64
 14  VEHICLE_TYPE         10000 non-null  object
 15  SPEEDING_VIOLATIONS  10000 non-null  int64
 16  DUIS                 10000 non-null  int64
 17  PAST_ACCIDENTS       10000 non-null  int64
 18  OUTCOME              10000 non-null  float64
dtypes: bool(1), category(5), float64(5), int64(5), object(3)
memory usage: 1.0+ MB
```

In [36]:
```
valores = cdf["VEHICLE_YEAR"].value_counts()
plt.pie(valores,labels=["Before 2015","After 2015"],autopct='%.0f%%')
plt.show()
```



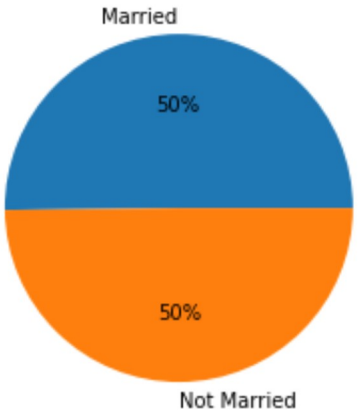
1.9 MARRIED

Same procedure as in VEHICLE_OWNERSHIP

In [37]:
```
cdf["MARRIED"].value_counts()
```

Out[37]:
```
0.0    5018
1.0    4982
Name: MARRIED, dtype: int64
```

In [38]:
```
cdf["MARRIED"] = cdf["MARRIED"].astype('bool')
valores = cdf["MARRIED"].value_counts()
plt.pie(valores,labels=["Married","Not Married"],autopct='%.0f%%')
plt.show()
```



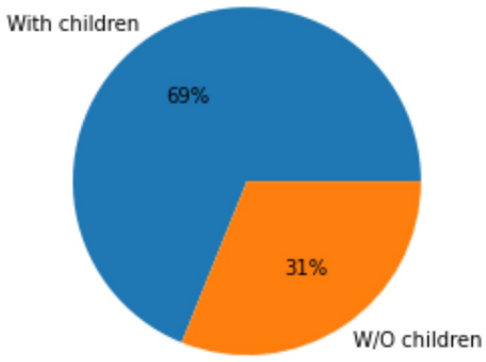
1.10 CHILDREN

Since we only know in the costumer haves children or not, the procedure is the same as in MARRIED.

In [39]:
```
cdf["CHILDREN"].value_counts()
```

Out[39]:
```
1.0    6888
0.0    3112
Name: CHILDREN, dtype: int64
```

In [40]:
```
cdf["CHILDREN"] = cdf["CHILDREN"].astype('bool')
valores = cdf["CHILDREN"].value_counts()
plt.pie(valores,labels=["With children","W/O children"],autopct='%.0f%%')
plt.show()
```



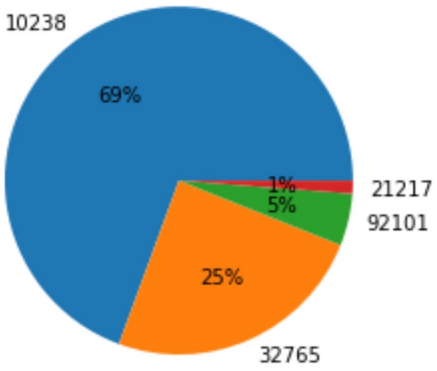
1.11 POSTAL_CODE

The postal code is defined as numerical, but is categorical.

In [41]:
```
cdf["POSTAL_CODE"].value_counts()
```

Out[41]:
```
10238    6940
32765    2456
92101     484
21217     120
Name: POSTAL_CODE, dtype: int64
```

In [42]:
```
cdf["POSTAL_CODE"] = pd.Categorical(cdf["POSTAL_CODE"])
values = cdf["POSTAL_CODE"].value_counts()
plt.pie(values,labels=cdf["POSTAL_CODE"].unique(),autopct='%.0f%%')
plt.show()
```



In [43]:
```
cdf["POSTAL_CODE"]
```

Out[43]:
```
0       10238
1       10238
2       10238
3       32765
4       32765
        ...
9995    10238
9996    32765
9997    10238
9998    10238
9999    10238
Name: POSTAL_CODE, Length: 10000, dtype: category
Categories (4, int64): [10238, 21217, 32765, 92101]
```

1.12 ANNUAL_MILEAGE

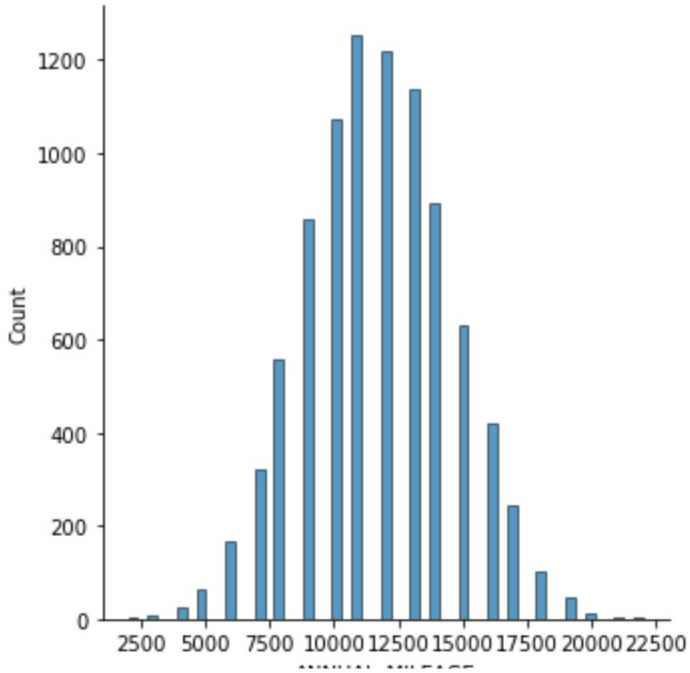
The first thing we notice is that the mileage follows a Gaussian distribution, which makes sense considering the nature of the data. We also find some null data we have to clean, inthis case we will use the Mean method.

In [44]:
```
cdf["ANNUAL_MILEAGE"].unique()
```

Out[44]:
```
array([12000., 16000., 11000., 13000., 14000., 10000.,  8000.,    nan,
       18000., 17000.,  7000., 15000.,  9000.,  5000.,  6000., 19000.,
        4000.,  3000.,  2000., 20000., 21000., 22000.])
```

In [45]:
```
sns.displot(cdf["ANNUAL_MILEAGE"])
```

Out[45]:
```
<seaborn.axisgrid.FacetGrid at 0x16610a2fbb0>
```



In [46]:
```
cdf["ANNUAL_MILEAGE"].isnull().sum()
```

Out[46]:
```
957
```

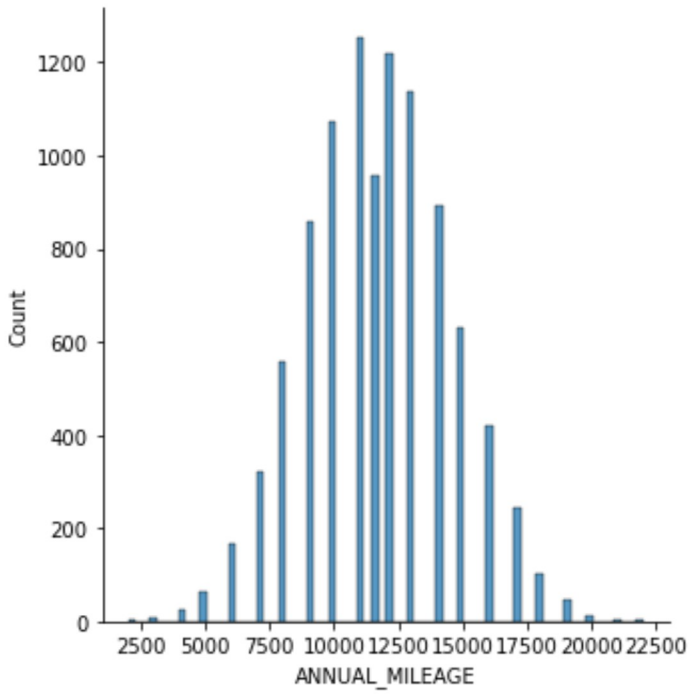
In [47]:
```
cdf["ANNUAL_MILEAGE"]=cdf["ANNUAL_MILEAGE"].fillna(cdf["ANNUAL_MILEAGE"].me
```

In [48]:
```
cdf["ANNUAL_MILEAGE"].isnull().sum()
```

Out[48]:
```
0
```

In [49]:
```
sns.displot(cdf["ANNUAL_MILEAGE"])
```

Out[49]:
```
<seaborn.axisgrid.FacetGrid at 0x16610a6cf10>
```



The null values were less than 10% and we can se how they "break" the distribution.

1.13 VEHICLE_TYPE

Another freathure that is categorical, just 2 types.

In [50]:
```
cdf["VEHICLE_TYPE"].unique()
```

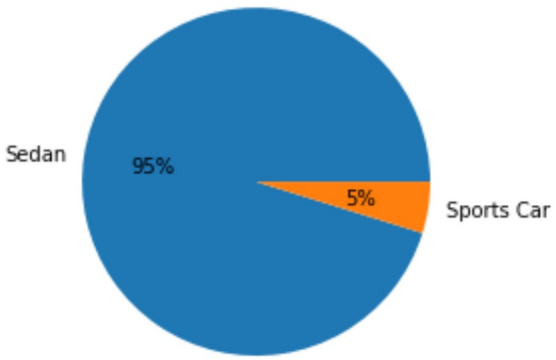
Out[50]:
```
array(['sedan', 'sports car'], dtype=object)
```

In [51]:
```
cdf["VEHICLE_TYPE"]=cdf["VEHICLE_TYPE"].replace({'sedan':'s','sports car':'

cdf["VEHICLE_TYPE"] = pd.Categorical(
    cdf["VEHICLE_TYPE"]
)
```

In [52]:
```
cdf["VEHICLE_TYPE"]
```

Out[52]:
```
0        s
1        s
2        s
3        s
4        s
       ..
9995     s
9996     s
9997     s
9998     s
9999     s
Name: VEHICLE_TYPE, Length: 10000, dtype: category
Categories (2, object): ['s', 'sc']
```

In [53]:
```
values = cdf["VEHICLE_TYPE"].value_counts()
plt.pie(values,labels=["Sedan","Sports Car"],autopct='%.0f%%')
plt.show()
```

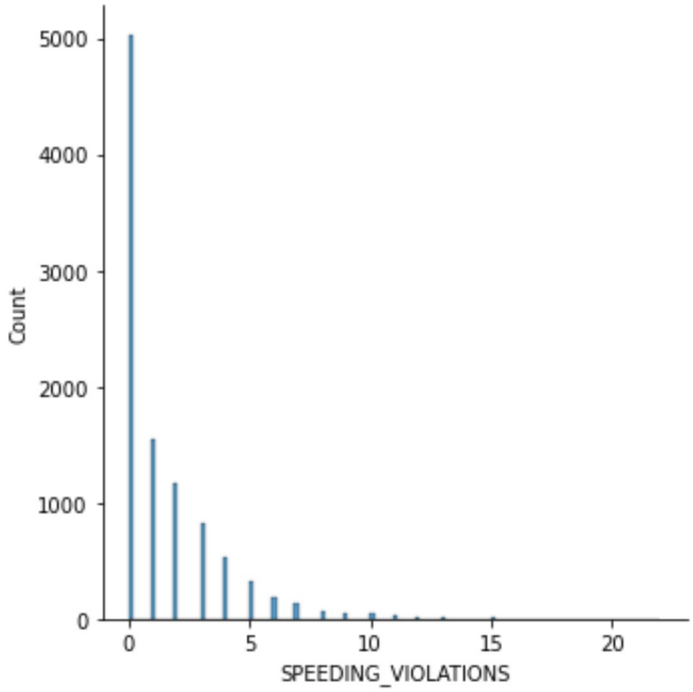


1.14 SPEEDING_VIOLATIONS

Follows a geometric distribution. Do not require any modification.

In [54]:
```
sns.displot(cdf["SPEEDING_VIOLATIONS"])
```

Out[54]:
```
<seaborn.axisgrid.FacetGrid at 0x16610a45bb0>
```



1.15 DUIS

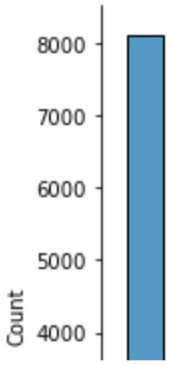
Follows the same pattern as SPEEDING_VIOLATIONS

In [55]:
```
cdf["DUIS"]
```

Out[55]:
```
0       0
1       0
2       0
3       0
4       0
       ..
9995    0
9996    0
9997    0
9998    0
9999    0
Name: DUIS, Length: 10000, dtype: int64
```

In [56]:
```
sns.displot(cdf["DUIS"])
```

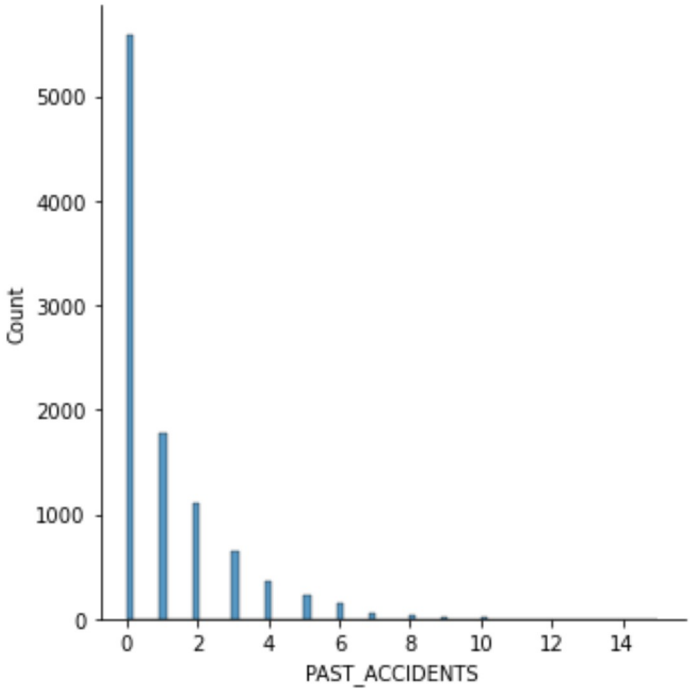
Out[56]:
```
<seaborn.axisgrid.FacetGrid at 0x16610c74ac0>
```



1.16 PAST_ACCIDENTS

In [57]:
```
sns.displot(cdf["PAST_ACCIDENTS"])
```

Out[57]:
```
<seaborn.axisgrid.FacetGrid at 0x16610b27100>
```



1.17 OUTCOME

This is the **Tag**, says if the client claimed the insurance. The only task is to change the data type from float64 to int32

In [58]:
```
cdf["OUTCOME"] = cdf["OUTCOME"].astype("int32")
```

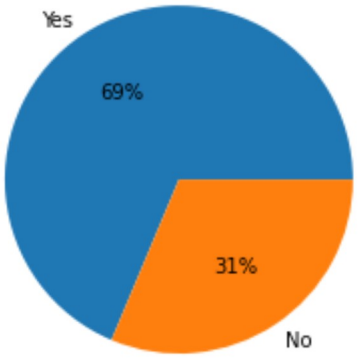
In [59]:
```
cdf["OUTCOME"]
```

Out[59]:
```
0       0
1       1
2       0
3       0
4       1
       ..
9995    0
9996    0
9997    0
```

```
9998    1
9999    0
Name: OUTCOME, Length: 10000, dtype: int32
```

In [60]:

```
values = cdf["OUTCOME"].value_counts()
plt.pie(values,labels=["Yes","No"],autopct='%.0f%%')
plt.show()
```



In [61]:

```
cdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   10000 non-null  int64
 1   AGE                  10000 non-null  category
 2   GENDER               10000 non-null  object
 3   RACE                 10000 non-null  object
 4   DRIVING_EXPERIENCE   10000 non-null  category
 5   EDUCATION            10000 non-null  category
 6   INCOME               10000 non-null  category
 7   CREDIT_SCORE         9018 non-null   float64
 8   VEHICLE_OWNERSHIP    10000 non-null  bool
 9   VEHICLE_YEAR         10000 non-null  category
 10  MARRIED              10000 non-null  bool
 11  CHILDREN             10000 non-null  bool
 12  POSTAL_CODE          10000 non-null  category
 13  ANNUAL_MILEAGE       10000 non-null  float64
 14  VEHICLE_TYPE         10000 non-null  category
 15  SPEEDING_VIOLATIONS  10000 non-null  int64
 16  DUIS                 10000 non-null  int64
 17  PAST_ACCIDENTS       10000 non-null  int64
 18  OUTCOME              10000 non-null  int32
dtypes: bool(3), category(7), float64(2), int32(1), int64(4), object(2)
memory usage: 763.0+ KB
```

1.18 CREDIT_SCORE cleaning

CREDIT_SCORE contains some null values we will have to treat to have the greatest ammount of precission. After a shor investigation, we conclude that if a person dont have a credit history, they have a credit score of 0, so null values have to be cleaned.

In [62]:
```
coches["CREDIT_SCORE"].isnull().sum()
```

Out[62]:
```
982
```

We find 982 null values, which are the 9'82% of the values.

Solution

Because the freature is numerical, we can assign a new value for the missing ones, deleting the rows or the freature is discarded.

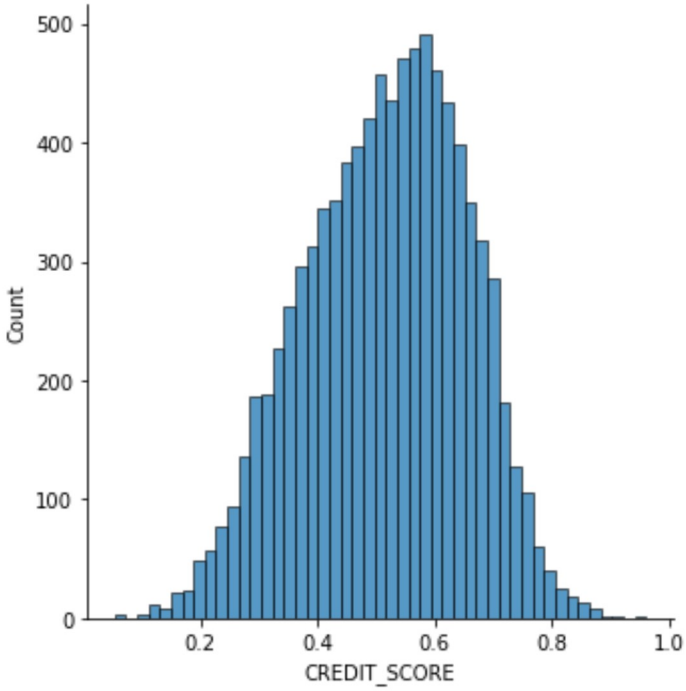
Imputation metods:

- Random: Discarded for being too imprecise for this case.
- Mean: Simple but effective method in the worst case will not impact the result my a lot.
- Mode: Works better for categorical data.
- Median: Not the best method for continuous data, but we can still use it.
- Model: The most precise solution, but also would cost a lot of time.
- Hot Deck: The value is decided based on similar values of the data.

We are going to use the *Hot Deck - KNN* imputation method, because is the most efficient one

In [63]:
```
sns.displot(cdf["CREDIT_SCORE"])
```

Out[63]:
```
<seaborn.axisgrid.FacetGrid at 0x16610e44f70>
```



In [64]:
```
coches["CREDIT_SCORE"]
```

Out[64]:
```
0       0.629027
1       0.357757
2       0.493146
```

```
3       0.206013
4       0.388366
          ...
9995    0.582787
9996    0.522231
9997    0.470940
9998    0.364185
9999    0.435225
Name: CREDIT_SCORE, Length: 10000, dtype: float64
```

In [65]:

```
# Encoding of categorical columns
from sklearn.preprocessing import LabelEncoder,OneHotEncoder

columnsEncode = ["AGE","GENDER","RACE","DRIVING_EXPERIENCE","EDUCATION","IN

#Copy to keep categorical values before encoding
cdf_copy = cdf.copy()

label_encoder =LabelEncoder()
for c in columnsEncode:
    cdf[c]=label_encoder.fit_transform(cdf[c])
cdf.head(10)
```

Out[65]:

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE	V
0	569520	3	0	0	0	0	3	0.629027	
1	750365	0	1	0	0	1	2	0.357757	
2	199901	0	0	0	0	0	0	0.493146	
3	478866	0	1	0	0	2	0	0.206013	
4	731664	1	1	0	1	1	0	0.388366	
5	877557	2	0	0	2	0	3	0.619127	
6	930134	3	1	0	3	0	3	0.492944	
7	461006	1	0	0	0	2	0	0.468689	
8	68366	2	0	0	2	2	0	0.521815	
9	445911	2	0	0	0	0	3	0.561531	

In [66]:

```
import numpy as np
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=2, weights="uniform")
res = imputer.fit_transform(cdf)
res = pd.DataFrame(res)
cdf["CREDIT_SCORE"]=res[7]
```

In [67]:

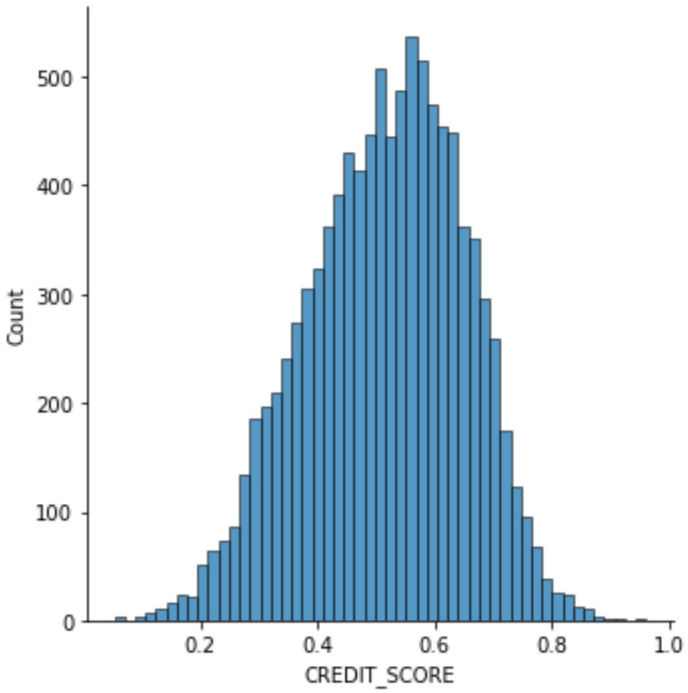
```
cdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   10000 non-null  int64
```

```
 1   AGE                  10000 non-null  int32
 2   GENDER               10000 non-null  int32
 3   RACE                 10000 non-null  int32
 4   DRIVING_EXPERIENCE   10000 non-null  int32
 5   EDUCATION            10000 non-null  int32
 6   INCOME               10000 non-null  int32
 7   CREDIT_SCORE         10000 non-null  float64
 8   VEHICLE_OWNERSHIP    10000 non-null  int64
 9   VEHICLE_YEAR         10000 non-null  int32
 10  MARRIED              10000 non-null  int64
 11  CHILDREN             10000 non-null  int64
 12  POSTAL_CODE          10000 non-null  int64
 13  ANNUAL_MILEAGE       10000 non-null  float64
 14  VEHICLE_TYPE         10000 non-null  int32
 15  SPEEDING_VIOLATIONS  10000 non-null  int64
 16  DUIS                 10000 non-null  int64
 17  PAST_ACCIDENTS       10000 non-null  int64
 18  OUTCOME              10000 non-null  int32
dtypes: float64(2), int32(9), int64(8)
```

In [68]:
```
sns.displot(cdf["CREDIT_SCORE"])
```

Out[68]:
```
<seaborn.axisgrid.FacetGrid at 0x1661422fd90>
```



In [69]:
```
cdf
```

Out[69]:

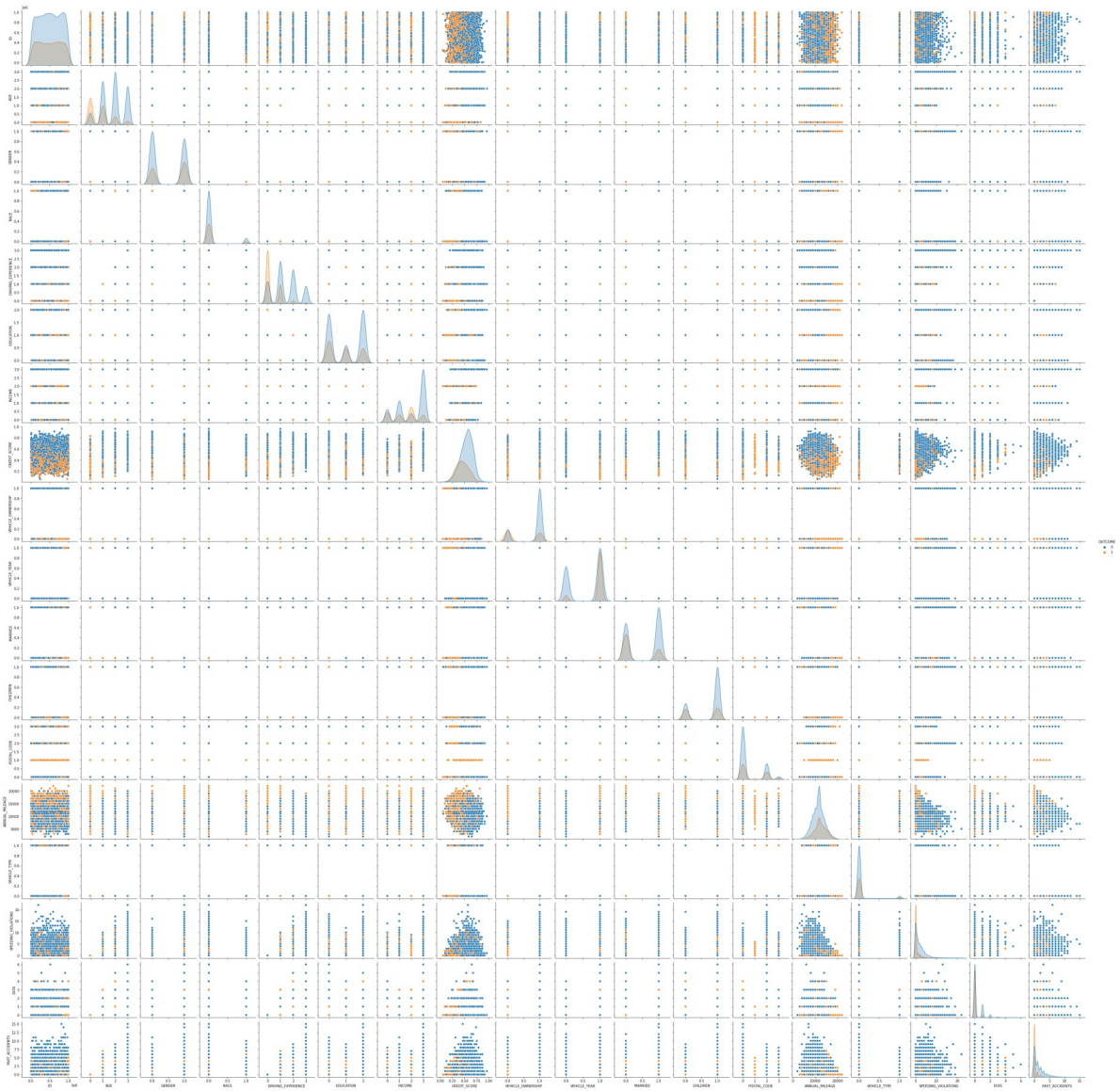
	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE
0	569520	3	0	0	0	0	3	0.629027
1	750365	0	1	0	0	1	2	0.357757
2	199901	0	0	0	0	0	0	0.493146
3	478866	0	1	0	0	2	0	0.206013
4	731664	1	1	0	1	1	0	0.388366
...	...	...	...	...	...	...	...	...

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE
9995	323164	1	0	0	1	2	3	0.582787
9996	910346	1	0	0	1	1	1	0.522231
9997	468409	1	1	0	0	0	1	0.470940
9998	903459	1	0	0	1	0	2	0.364185
9999	442696	1	0	0	0	1	0	0.435225

2. Visualization of the clean data

In [70]:
```
sns.pairplot(data=cdf,hue="OUTCOME")
```

Out[70]:
```
<seaborn.axisgrid.PairGrid at 0x16610e75700>
```



3. Data pipelines

In [71]:
```
#Dataset for pipelines
cdfPipe = pd.read_csv('Car_Insurance_Claim.csv')
cdfPipe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   10000 non-null  int64  
 1   AGE                  10000 non-null  object 
 2   GENDER               10000 non-null  object 
 3   RACE                 10000 non-null  object 
 4   DRIVING_EXPERIENCE   10000 non-null  object 
 5   EDUCATION            10000 non-null  object 
 6   INCOME               10000 non-null  object 
 7   CREDIT_SCORE         9018 non-null   float64
 8   VEHICLE_OWNERSHIP    10000 non-null  float64
 9   VEHICLE_YEAR         10000 non-null  object 
 10  MARRIED              10000 non-null  float64
 11  CHILDREN             10000 non-null  float64
 12  POSTAL_CODE          10000 non-null  int64  
 13  ANNUAL_MILEAGE       9043 non-null   float64
 14  VEHICLE_TYPE         10000 non-null  object 
 15  SPEEDING_VIOLATIONS  10000 non-null  int64  
 16  DUIS                 10000 non-null  int64  
 17  PAST_ACCIDENTS       10000 non-null  int64  
 18  OUTCOME              10000 non-null  float64
dtypes: float64(6), int64(5), object(8)
memory usage: 1.4+ MB
```

3.1 Pipeline for categorical data

Categorical data sometimes have to be transformed to make it accesible for visualization and encodign. If the data is ordinal, we have to establish manually a order, like in AGE or DRIVING_EXPERIENCE, once it is done its easy to transform.

3.2 Pipeline for missing data

The pipeline for mising data can be automatized with the aplication of a imputation method.

In [72]:
```
from sklearn.preprocessing import StandardScaler, OrdinalEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

In [73]:
```
# Imputation by MEAN
numeric_transformer = Pipeline(steps=[
       ('imputer', SimpleImputer(strategy='mean'))
#      ,('scaler', StandardScaler())
])
```

3.3 Pipeline for encoding data

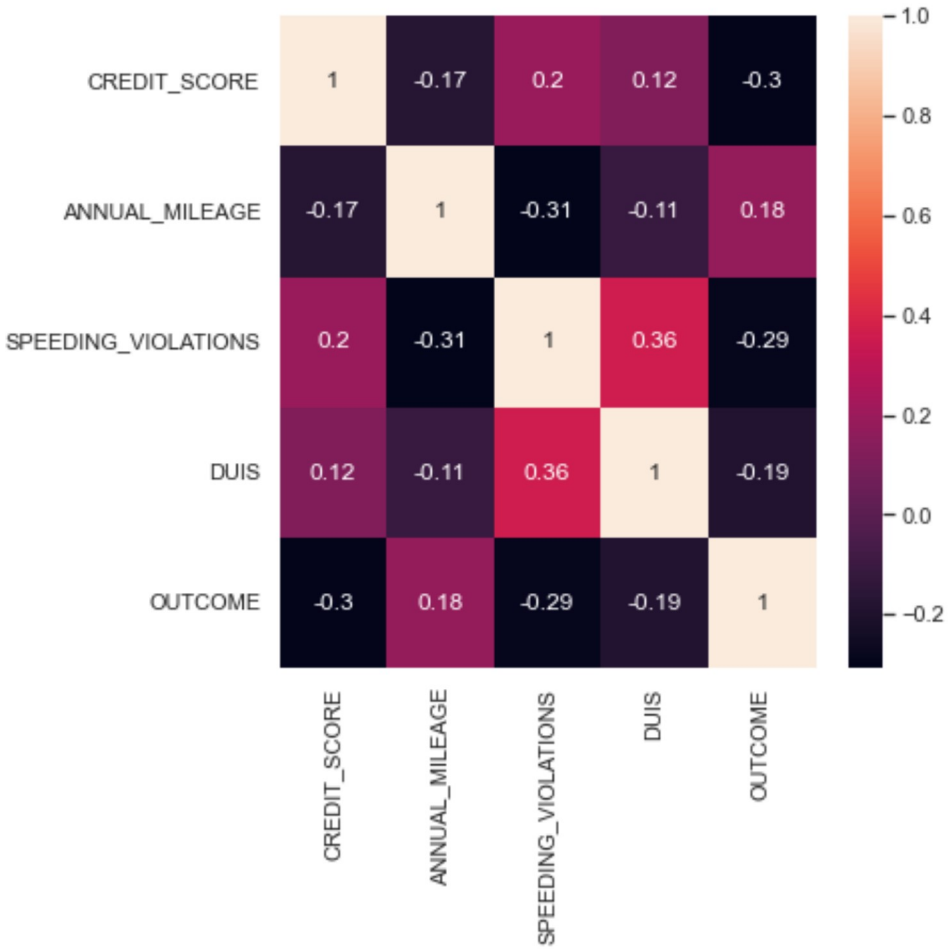
In [74]:

```
categorical_transformer = Pipeline(steps=[
       ('imputer', SimpleImputer(strategy='constant'))
#       ,('encoder', OrdinalEncoder())
])

numeric_features = ["CREDIT_SCORE","ANNUAL_MILEAGE","SPEEDING_VIOLATIONS",'
categorical_features = ["AGE","GENDER","RACE","DRIVING_EXPERIENCE","EDUCATI

preprocessor = ColumnTransformer(
    transformers=[
     ('numeric', numeric_transformer, numeric_features)
    ,('categorical', categorical_transformer, categorical_features)
])
```

4. Data visualization

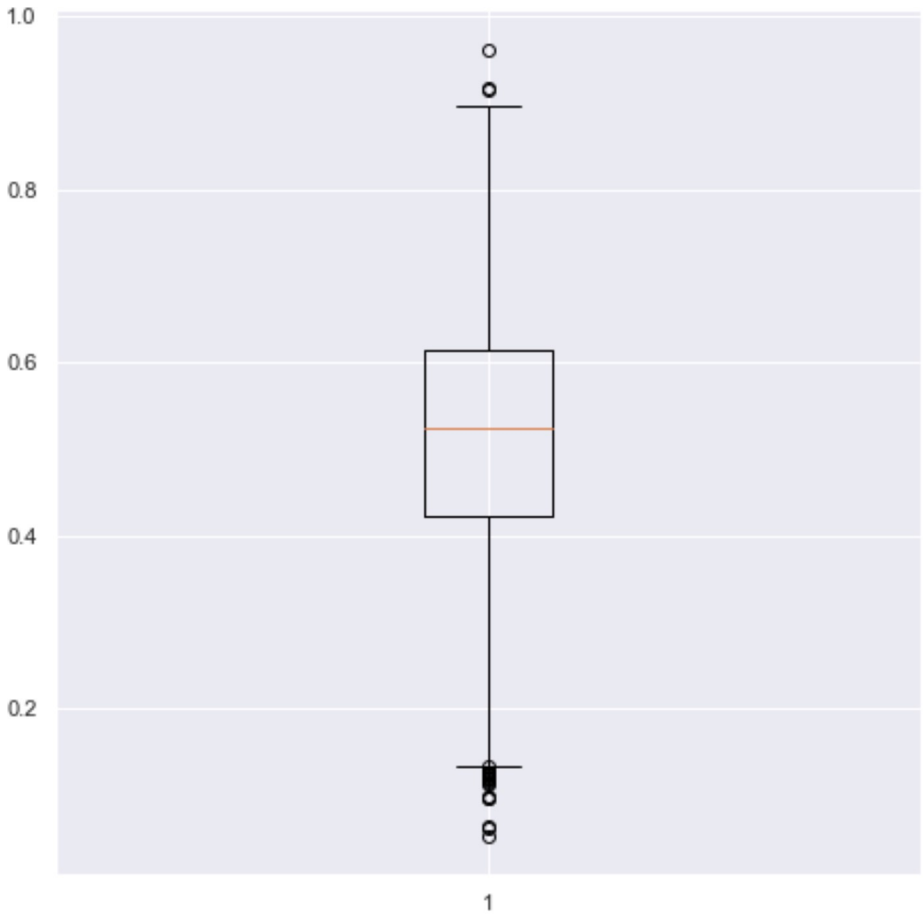
In [107…

```
numeric_features.append("OUTCOME")
corrMatrix = cdf[numeric_features].corr()
sns.set(rc = {'figure.figsize':(6,6)})
sns.heatmap(corrMatrix, annot=True)
plt.show()
```



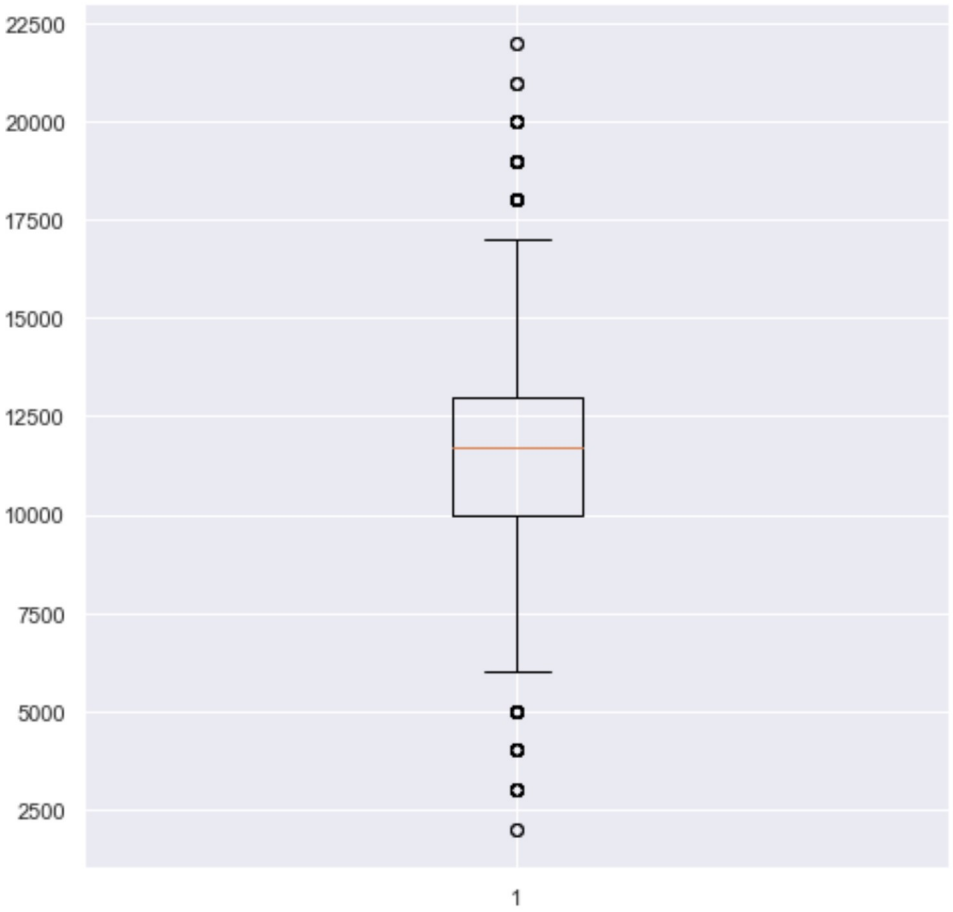
Distribution of the CREDIT_SCORE

In [76]:
```
fig = plt.figure()
# Create an axes instance
ax = fig.add_axes([0,0,1,1])
# Create the boxplot
bp = ax.boxplot([cdf["CREDIT_SCORE"]])
plt.show()
```



Distribution of the ANNUAL_MILEAGE

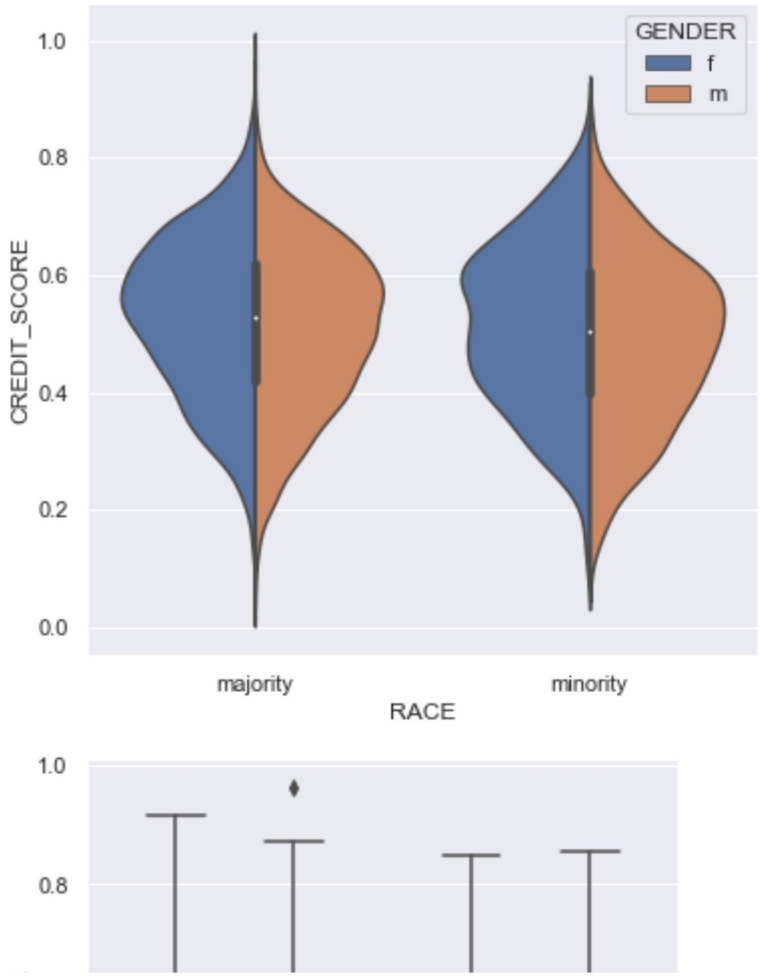
In [77]:
```
fig = plt.figure()
# Create an axes instance
ax = fig.add_axes([0,0,1,1])
# Create the boxplot
bp = ax.boxplot([cdf["ANNUAL_MILEAGE"]])
plt.show()
```



Influence by the race and gender in the CREDIT_SCORE

In [90]:
```
sns.violinplot(data=cdf_copy,
               kind="violin",
               x="RACE",
               y="CREDIT_SCORE",
               hue="GENDER",
               split=True
              )
sns.catplot(data=cdf_copy,
               kind="box",
               x="RACE",
               y="CREDIT_SCORE",
               hue="GENDER"
              )
```

Out[90]:
```
<seaborn.axisgrid.FacetGrid at 0x16636b947f0>
```



Distribution by OUTCOME

In [99]:
```
sns.jointplot(data=cdf_copy, x="CREDIT_SCORE", y="ANNUAL_MILEAGE", hue="OUT
```

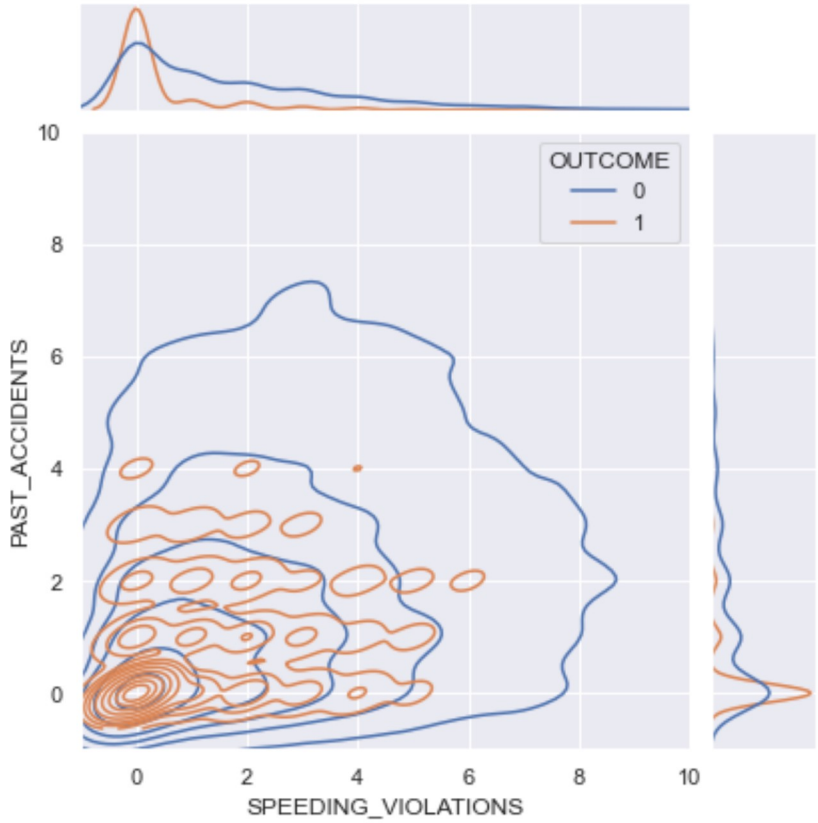
Out[99]:
```
<seaborn.axisgrid.JointGrid at 0x1663dde4f10>
```

In [110…

```
plot = sns.jointplot(data=cdf_copy, x="SPEEDING_VIOLATIONS", y="PAST_ACCIDE
plot.ax_marg_x.set_xlim(-1, 10)
plot.ax_marg_y.set_ylim(-1, 10)
```

Out[110…

```
(-1.0, 10.0)
```



In [ ]: