In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df=pd.read_csv("D:\\projects\\DS Internship datasets\\project 2\\Car_Insurance_Claim.csv"
```

In [3]:

```
df.head()
```

Out[3]:

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_S
0	569520	65+	female	majority	0-9y	high school	upper class	0.6
1	750365	16-25	male	majority	0-9y	none	poverty	0.3
2	199901	16-25	female	majority	0-9y	high school	working class	0.4
3	478866	16-25	male	majority	0-9y	university	working class	0.2
4	731664	26-39	male	majority	10-19y	none	working class	0.3

In [4]:

```
df.tail()
```

Out[4]:

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDI
9995	323164	26-39	female	majority	10-19y	university	upper class	
9996	910346	26-39	female	majority	10-19y	none	middle class	
9997	468409	26-39	male	majority	0-9y	high school	middle class	
9998	903459	26-39	female	majority	10-19y	high school	poverty	
9999	442696	26-39	female	majority	0-9y	none	working class	

In [5]:

```
df.describe()
```

Out[5]:

	ID	CREDIT_SCORE	VEHICLE_OWNERSHIP	MARRIED	CHILDREN	P
count	10000.000000	9018.000000	10000.000000	10000.000000	10000.000000	
mean	500521.906800	0.515813	0.697000	0.498200	0.688800	
std	290030.768758	0.137688	0.459578	0.500022	0.463008	
min	101.000000	0.053358	0.000000	0.000000	0.000000	
25%	249638.500000	0.417191	0.000000	0.000000	0.000000	
50%	501777.000000	0.525033	1.000000	0.000000	1.000000	
75%	753974.500000	0.618312	1.000000	1.000000	1.000000	
max	999976.000000	0.960819	1.000000	1.000000	1.000000	

In [6]:

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

Out[6]:

```
2
```

In [7]:

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

Out[7]:

```
ID                     0
AGE                    0
GENDER                 0
RACE                   0
DRIVING_EXPERIENCE     0
EDUCATION              0
INCOME                 0
CREDIT_SCORE         982
VEHICLE_OWNERSHIP      0
VEHICLE_YEAR           0
MARRIED                0
CHILDREN               0
POSTAL_CODE            0
ANNUAL_MILEAGE       957
VEHICLE_TYPE           0
SPEEDING_VIOLATIONS    0
DUIS                   0
PAST_ACCIDENTS         0
OUTCOME                0
dtype: int64
```

In [8]:

```
df.shape
```

Out[8]:

```
(10000, 19)
```

In [9]:

```
df.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
```

In [10]:

```
df[df.isnull().any(axis=1)]
```

Out[10]:

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDI
13	569640	16-25	female	majority	0-9y	university	upper class	
15	906223	26-39	female	majority	0-9y	high school	upper class	
16	517747	65+	male	majority	30y+	university	upper class	
17	24851	16-25	male	majority	0-9y	none	poverty	
18	104086	26-39	female	majority	0-9y	university	upper class	
...	...	...	...	...	...	...	...	
9977	794068	65+	male	minority	0-9y	none	upper class	
9981	366048	26-39	male	majority	0-9y	high school	working class	
9985	595418	16-25	male	minority	0-9y	high school	working class	
9988	479789	26-39	male	majority	10-19y	high school	poverty	
9996	910346	26-39	female	majority	10-19y	none	middle class	

1851 rows × 19 columns

In [11]:

```
df[df["CREDIT_SCORE"].isnull()]
```

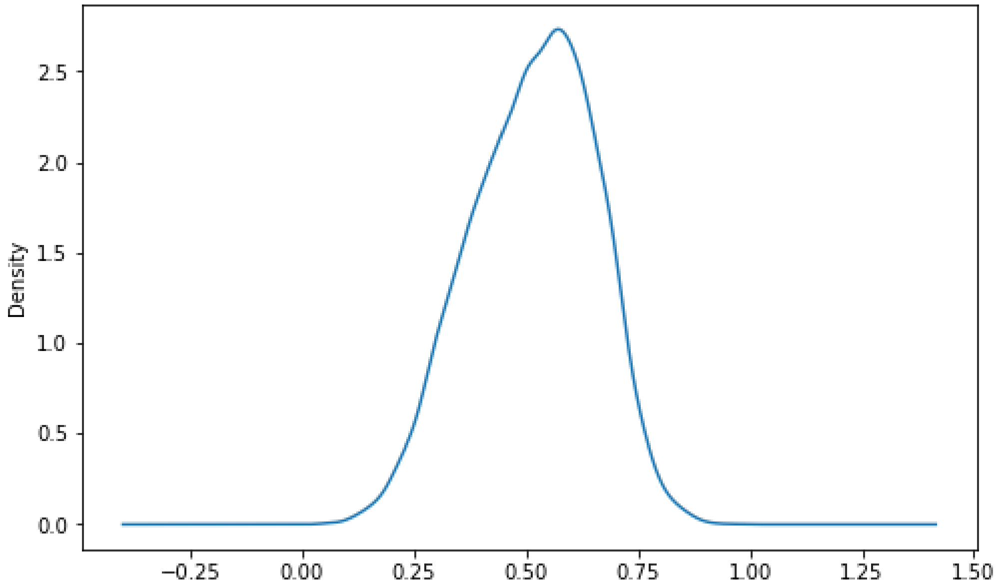
Out[11]:

	ID	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDI
17	24851	16-25	male	majority	0-9y	none	poverty	
23	217	16-25	male	majority	0-9y	none	poverty	
37	511757	40-64	female	majority	10-19y	none	middle class	
38	429947	65+	male	majority	30y+	university	upper class	
47	921097	40-64	female	majority	20-29y	university	upper class	
...	...	...	...	...	...	...	...	
9952	870405	40-64	female	majority	10-19y	university	upper class	
9967	27406	26-39	female	majority	10-19y	high school	middle class	
9981	366048	26-39	male	majority	0-9y	high school	working class	
9985	595418	16-25	male	minority	0-9y	high school	working class	
9988	479789	26-39	male	majority	10-19y	high school	poverty	

982 rows × 19 columns

In [12]:

```
plt.figure(figsize=(8,5))
df['CREDIT_SCORE'].plot(kind='kde')
plt.show()
```



In [13]:

```
df["CREDIT_SCORE"].fillna(df["CREDIT_SCORE"].median(),inplace=True)
```

In [14]:

```
df["ANNUAL_MILEAGE"].fillna(df["ANNUAL_MILEAGE"].median(),inplace=True)
```

In [15]:

```
df.isna().sum()
```

Out[15]:

```
ID                     0
AGE                    0
GENDER                 0
RACE                   0
DRIVING_EXPERIENCE     0
EDUCATION              0
INCOME                 0
CREDIT_SCORE           0
VEHICLE_OWNERSHIP      0
VEHICLE_YEAR           0
MARRIED                0
CHILDREN               0
POSTAL_CODE            0
ANNUAL_MILEAGE         0
VEHICLE_TYPE           0
SPEEDING_VIOLATIONS    0
DUIS                   0
PAST_ACCIDENTS         0
OUTCOME                0
dtype: int64
```

In [16]:

```
df.drop(["ID"],axis=1,inplace=True)
```

In [17]:

```
df.head()
```

Out[17]:

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

In [18]:

```
df["AGE"].unique()#discrete
```

Out[18]:

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

In [19]:

```
df["GENDER"].unique()#nominal
```

Out[19]:

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

In [20]:

```
df["RACE"].unique()#nominal
```

Out[20]:

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

In [21]:

```
df["DRIVING_EXPERIENCE"].unique()#discrete
```

Out[21]:

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

In [22]:

```
df["EDUCATION"].unique()#ordinal
```

Out[22]:

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

In [23]:

```
df["INCOME"].unique()#ordinal
```

Out[23]:

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

In [24]:

```
df["CREDIT_SCORE"].unique()#continous
```

Out[24]:

```
array([0.62902731, 0.35775712, 0.49314579, ..., 0.47094023, 0.36418478,
       0.43522478])
```

In [25]:

```
df["VEHICLE_OWNERSHIP"].unique()#discrete
```

Out[25]:

```
array([1., 0.])
```

In [26]:

```
df["VEHICLE_YEAR"].unique()#nominal
```

Out[26]:

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

In [27]:

```
df["MARRIED"].unique()#discrete
```

Out[27]:

```
array([0., 1.])
```

In [28]:

```
df["CHILDREN"].unique()#discrete
```

Out[28]:

```
array([1., 0.])
```

In [29]:

```
df["POSTAL_CODE"].unique()#nominal
```

Out[29]:

```
array([10238, 32765, 92101, 21217], dtype=int64)
```

In [30]:

```
df["ANNUAL_MILEAGE"].unique()#discrete
```

Out[30]:

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

In [31]:

```
df["VEHICLE_TYPE"].unique()#nominal
```

Out[31]:

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

In [32]:

```
df["SPEEDING_VIOLATIONS"].unique()#continous
```

Out[32]:

```
array([ 0,  2,  3,  7,  6,  4, 10, 13,  1,  5,  9,  8, 12, 11, 15, 17, 19,
       18, 16, 14, 22], dtype=int64)
```

In [33]:

```
df["DUIS"].unique()#continous
```

Out[33]:

```
array([0, 2, 1, 3, 4, 5, 6], dtype=int64)
```

In [34]:

```
df["PAST_ACCIDENTS"].unique()#continous
```

Out[34]:

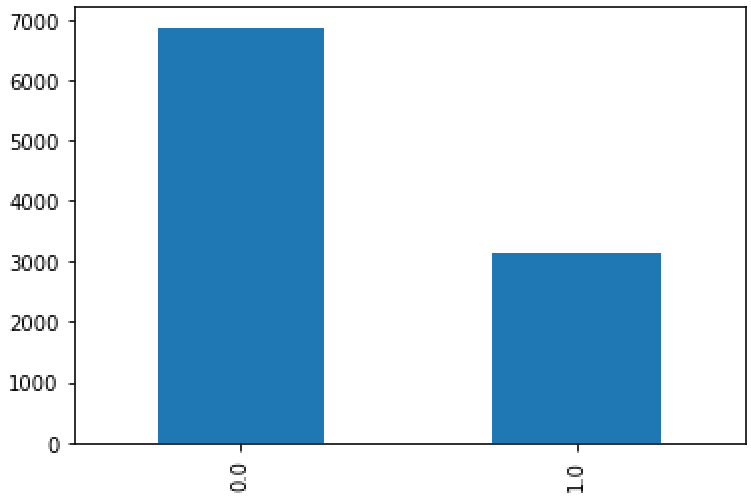
```
array([ 0,  1,  3,  7,  2,  5,  4,  6,  8, 10, 11,  9, 12, 14, 15],
      dtype=int64)
```

In [35]:

```
df["OUTCOME"].value_counts().plot(kind='bar')#discrete
```

Out[35]:

```
<AxesSubplot:>
```



In [36]:

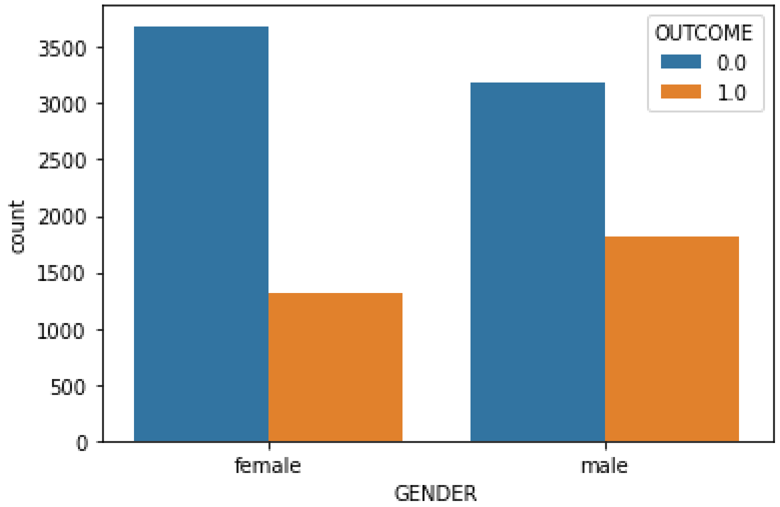
```
df["GENDER"].value_counts()
```

Out[36]:

```
female    5010
male      4990
Name: GENDER, dtype: int64
```

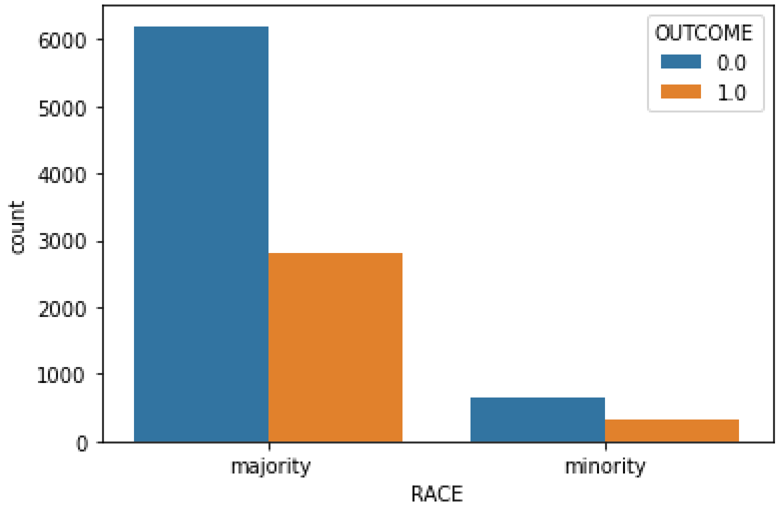
In [37]:

```
sns.countplot(data=df,x="GENDER",hue="OUTCOME")
plt.show()
```



In [38]:

```
sns.countplot(data=df,x="RACE",hue="OUTCOME")
plt.show()
```



In [39]:

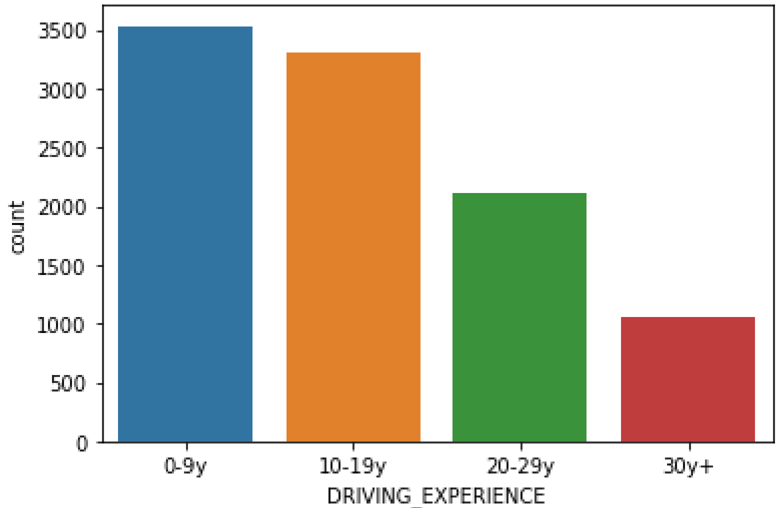
```
df["DRIVING_EXPERIENCE"].value_counts()
```

Out[39]:

```
0-9y      3530
10-19y    3299
20-29y    2119
30y+      1052
Name: DRIVING_EXPERIENCE, dtype: int64
```

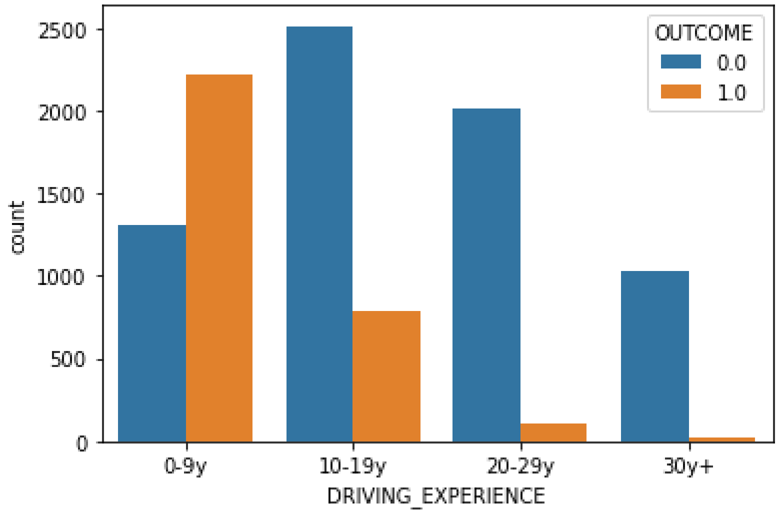
In [40]:

```
sns.countplot(df["DRIVING_EXPERIENCE"])
plt.show()
```



In [41]:

```
sns.countplot(data=df,x="DRIVING_EXPERIENCE",hue="OUTCOME")
plt.show()
```



In [42]:

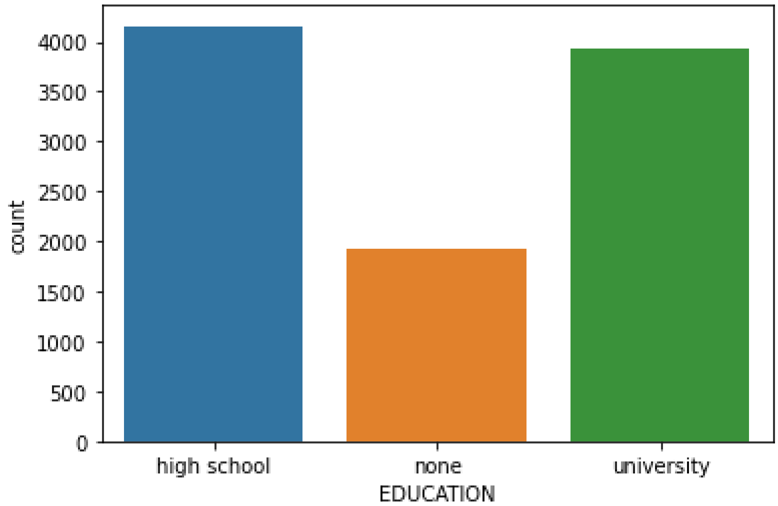
```
df["EDUCATION"].value_counts()
```

Out[42]:

```
high school    4157
university     3928
none           1915
Name: EDUCATION, dtype: int64
```

In [43]:

```
sns.countplot(df["EDUCATION"])
plt.show()
```



In [44]:

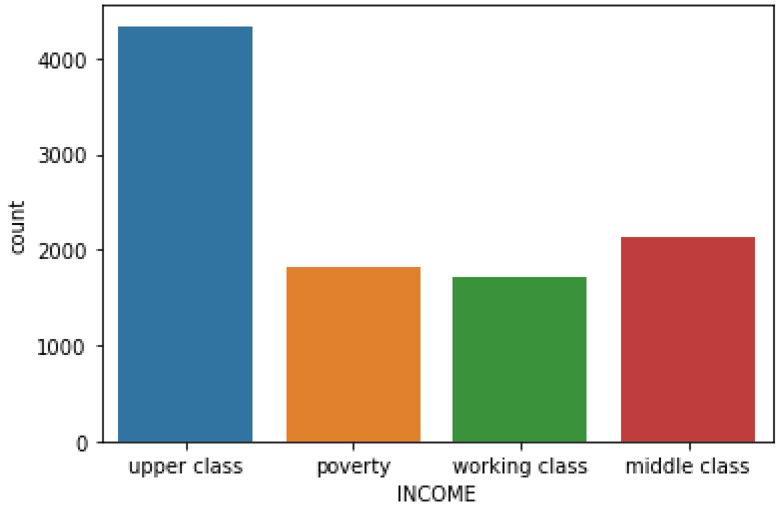
```
df["INCOME"].value_counts()
```

Out[44]:

```
upper class      4336
middle class     2138
poverty          1814
working class    1712
Name: INCOME, dtype: int64
```

In [45]:

```
sns.countplot(df["INCOME"])
plt.show()
```

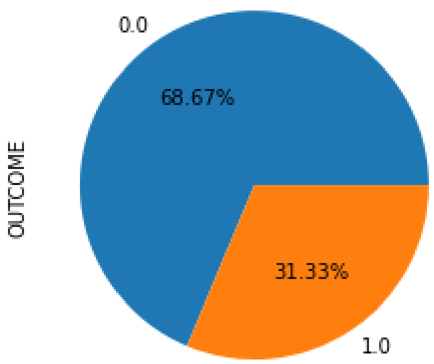


In [46]:

```
df["OUTCOME"].value_counts().plot(kind='pie',autopct="%0.2f%%")
```

Out[46]:

<AxesSubplot:ylabel='OUTCOME'>



In [47]:

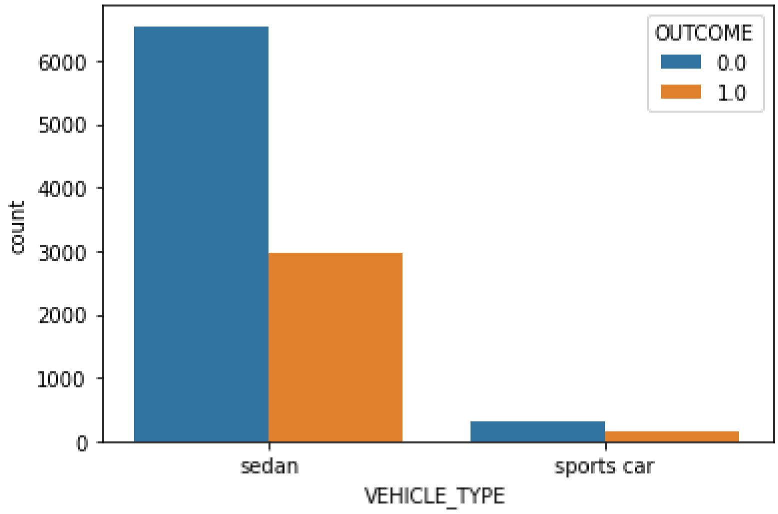
```
df["VEHICLE_TYPE"].value_counts()
```

Out[47]:

```
sedan         9523
sports car     477
Name: VEHICLE_TYPE, dtype: int64
```

In [48]:

```
sns.countplot(data=df,x='VEHICLE_TYPE',hue='OUTCOME')
plt.show()
```



In [49]:

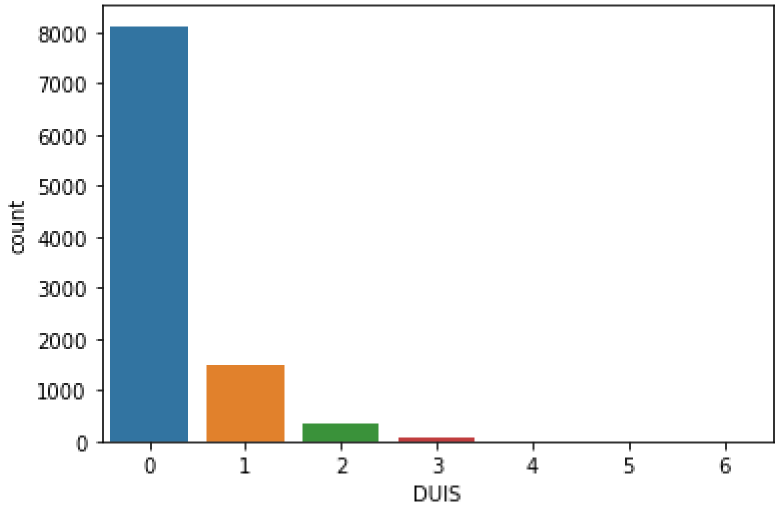
```
df["DUIS"].value_counts()
```

Out[49]:

```
0    8118
1    1470
2     331
3      68
4      10
5       2
6       1
Name: DUIS, dtype: int64
```

In [50]:

```
sns.countplot(df["DUIS"])
plt.show()
```

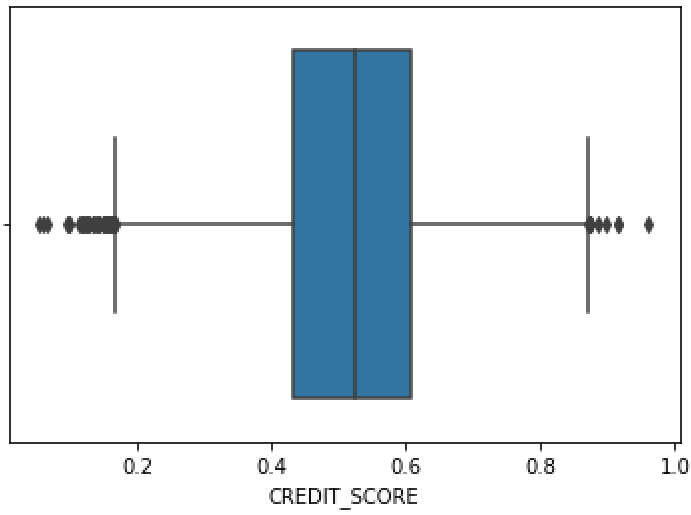


In [51]:

```
n=df.select_dtypes(exclude='object')
```

In [52]:

```
for i in n.columns:
    sns.boxplot(data=n,x=i)
    plt.show()
```



In [53]:

```
df.dtypes
```

Out[53]:

```
AGE                      object
GENDER                   object
RACE                     object
DRIVING_EXPERIENCE       object
EDUCATION                object
INCOME                   object
CREDIT_SCORE            float64
VEHICLE_OWNERSHIP       float64
VEHICLE_YEAR             object
MARRIED                 float64
CHILDREN                float64
POSTAL_CODE               int64
ANNUAL_MILEAGE          float64
VEHICLE_TYPE             object
SPEEDING_VIOLATIONS       int64
DUIS                      int64
PAST_ACCIDENTS            int64
OUTCOME                 float64
dtype: object
```

In [54]:

```
from sklearn.preprocessing import LabelEncoder
lr=LabelEncoder()
```

In [55]:

```
df['AGE']=lr.fit_transform(df['AGE'])
df['GENDER']=lr.fit_transform(df['GENDER'])
df['RACE']=lr.fit_transform(df['RACE'])
df['DRIVING_EXPERIENCE']=lr.fit_transform(df['DRIVING_EXPERIENCE'])
df['EDUCATION']=lr.fit_transform(df['EDUCATION'])
df['INCOME']=lr.fit_transform(df['INCOME'])
df['VEHICLE_YEAR']=lr.fit_transform(df['VEHICLE_YEAR'])
df['VEHICLE_TYPE']=lr.fit_transform(df['VEHICLE_TYPE'])
```

In [56]:

```
df.dtypes
```

Out[56]:

```
AGE                     int32
GENDER                  int32
RACE                    int32
DRIVING_EXPERIENCE      int32
EDUCATION               int32
INCOME                  int32
CREDIT_SCORE          float64
VEHICLE_OWNERSHIP     float64
VEHICLE_YEAR            int32
MARRIED               float64
CHILDREN              float64
POSTAL_CODE             int64
ANNUAL_MILEAGE        float64
VEHICLE_TYPE            int32
SPEEDING_VIOLATIONS     int64
DUIS                    int64
PAST_ACCIDENTS          int64
OUTCOME               float64
dtype: object
```

In [57]:

```
df.head()
```

Out[57]:

	AGE	GENDER	RACE	DRIVING_EXPERIENCE	EDUCATION	INCOME	CREDIT_SCORE	VE
0	3	0	0	0	0	2	0.629027	
1	0	1	0	0	1	1	0.357757	
2	0	0	0	0	0	3	0.493146	
3	0	1	0	0	2	3	0.206013	
4	1	1	0	1	1	3	0.388366	

In [58]:

```
car_insurance=df.values
```

In [59]:

```
df.shape
```

Out[59]:

```
(10000, 18)
```

In [60]:

```
x=car_insurance[:,0:17]
y=car_insurance[:,17]
```

In [61]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
```

Logistic Regression

In [62]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

In [63]:

```
logistic_model = LogisticRegression().fit(x_train,y_train)
ypredicted = logistic_model.predict(x_test)
ypredicted
```

Out[63]:

```
array([1., 0., 0., ..., 0., 0., 0.])
```

Evaluation for Logistic Regression

In [64]:

```
Varience=np.var(ypredicted)
Varience
```

Out[64]:

```
0.20391822222222222
```

In [65]:

```
SE=np.mean((np.mean(ypredicted)-y)**2)
bias=SE-Varience
bias
```

Out[65]:

```
0.012007022222222219
```

In [66]:

```
print("Confusion Matrix")
matrix = confusion_matrix(y_test,ypredicted)
print(matrix)
```

```
Confusion Matrix
[[1842  203]
 [ 302  653]]
```

In [67]:

```
print("\nClassification Report")
report = classification_report(y_test,ypredicted)
print(report)
```

```

Classification Report
              precision    recall  f1-score   support

         0.0       0.86      0.90      0.88      2045
         1.0       0.76      0.68      0.72       955

    accuracy                           0.83      3000
   macro avg       0.81      0.79      0.80      3000
weighted avg       0.83      0.83      0.83      3000

```

In [68]:

```
lr_accuracy = accuracy_score(y_test, ypredicted)
lr_accuracy
print('Logistic Regression Accuracy of Scikit Model: {:.2f}%'.format(lr_accuracy*100))
```

```
Logistic Regression Accuracy of Scikit Model: 83.17%
```

In [69]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedStratifiedKFold
```

In [70]:

```
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2']
# define models and parameters
model = LogisticRegression()
c_values = [100, 1000, 1.0, 0.1, 0.01]
# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='a
grid_result = grid_search.fit(x_train,y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean*100, stdev*100, param))
```

```
Best: 0.845333 using {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
84.533333 (1.070275) with: {'C': 100, 'penalty': 'l2', 'solver': 'newton-c
g'}
80.647619 (2.359734) with: {'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
80.528571 (1.225661) with: {'C': 100, 'penalty': 'l2', 'solver': 'liblinea
r'}
84.519048 (1.053974) with: {'C': 1000, 'penalty': 'l2', 'solver': 'newton-
cg'}
80.819048 (2.513618) with: {'C': 1000, 'penalty': 'l2', 'solver': 'lbfgs'}
80.528571 (1.225661) with: {'C': 1000, 'penalty': 'l2', 'solver': 'libline
ar'}
84.509524 (0.994326) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'newton-c
g'}
80.833333 (2.426914) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'lbfgs'}
80.528571 (1.225661) with: {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinea
r'}
84.390476 (0.895137) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'newton-c
g'}
81.076190 (2.496728) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}
80.528571 (1.233407) with: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinea
r'}
83.533333 (0.972362) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'newton-
cg'}
80.600000 (1.885642) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
80.509524 (1.139688) with: {'C': 0.01, 'penalty': 'l2', 'solver': 'libline
ar'}
```

In [71]:

```
grid_result.score(x_train,y_train)*100
```

Out[71]:

```
84.81428571428572
```

In [73]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
```

In [75]:

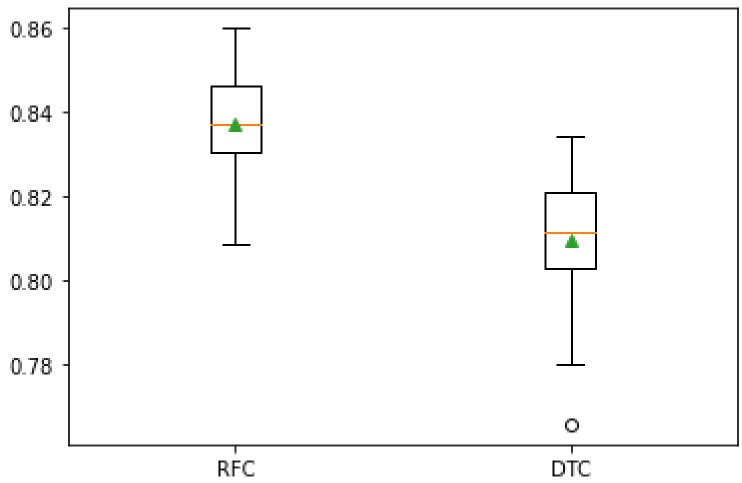
```
# Spot-Check Algorithms
models = []
models.append(('RF', RandomForestClassifier()))
models.append(('GNB', GaussianNB()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('DSC', DecisionTreeClassifier(random_state = 1, max_depth=2)))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = RepeatedStratifiedKFold(n_splits=10, n_repeats = 3, random_state=1)
    cv_results = cross_val_score(model,x_train,y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %.2f (%.3f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
RF: 0.84 (0.011)
GNB: 0.73 (0.017)
KNN: 0.79 (0.014)
DSC: 0.81 (0.015)
SVM: 0.70 (0.004)
```

In [76]:

```
# evaluate model 1
model1 = RandomForestClassifier()
cv1 = RepeatedStratifiedKFold(n_splits = 10, n_repeats = 3, random_state = 1)
scores1 = cross_val_score(model1,x_train,y_train, scoring = 'accuracy', cv = cv1, n_jobs
print('RFC Mean Accuracy: %.1f%% +/-(%.3f)' % (np.mean(scores1*100), np.std(scores1)))
# evaluate model 2
model2 = DecisionTreeClassifier(random_state = 1, max_depth=2)
cv2 = RepeatedStratifiedKFold(n_splits = 10, n_repeats = 3, random_state = 1)
scores3 = cross_val_score(model2,x_train,y_train, scoring = 'accuracy', cv = cv2, n_jobs
print('DecisionTreeClassifier Mean Accuracy: %.1f%% +/-(%.3f)' % (np.mean(scores3*100), r
# plot the results
plt.boxplot([scores1, scores3], labels=['RFC', 'DTC'], showmeans=True)
plt.show()
```

```
RFC Mean Accuracy: 83.7% +/-(0.012)
DecisionTreeClassifier Mean Accuracy: 80.9% +/-(0.015)
```



In [77]:

```
from mlxtend.evaluate import paired_ttest_5x2cv
# check if difference between algorithms is real
t, p = paired_ttest_5x2cv(estimator1=model1,
                          estimator2=model2,
                          X=x,
                          y=y,
                          scoring='accuracy',
                          random_seed=1)
# summarize
print(f'The P-value is = {p:.3f}')
print(f'The t-statistics is = {t:.3f}')
# interpret the result
if p <= 0.05:
    print('Since p<0.05, We can reject the null-hypothesis that both models perform equal
else:
    print('Since p>0.05, we cannot reject the null hypothesis and may conclude that the p
```

```
The P-value is = 0.021
The t-statistics is = 3.339
Since p<0.05, We can reject the null-hypothesis that both models perform e
qually well on this dataset. We may conclude that the two algorithms are s
ignificantly different.
```

In [ ]:

```

```