Amazon.com - Employee Access. Classification model

By : Rajesh Sahu 20223076 MCA Third Semester

In [1]:

```
#Importing Required Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
ng Dataset
=pd.read_csv('/Users/rajeshsahu/Desktop/Data Science Learning/DataSciencePython Pro
```

In [3]:

```
#printing dataset
dataset
```

Out[3]:

	ACTION	RESOURCE	MGR_ID	ROLE_ROLLUP_1	ROLE_ROLLUP_2	ROLE_DEPTNAME	F
0	1	39353	85475	117961	118300	123472	
1	1	17183	1540	117961	118343	123125	
2	1	36724	14457	118219	118220	117884	
3	1	36135	5396	117961	118343	119993	
4	1	42680	5905	117929	117930	119569	
...	...	...	...	...	...	...	
32764	1	23497	16971	117961	118300	119993	
32765	1	25139	311198	91261	118026	122392	
32766	1	34924	28805	117961	118327	120299	
32767	1	80574	55643	118256	118257	117945	
32768	1	14354	59575	117916	118150	117920	

32769 rows × 10 columns

In [4]:

```
#printing Info of dataset
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32769 entries, 0 to 32768
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ACTION            32769 non-null  int64
 1   RESOURCE          32769 non-null  int64
 2   MGR_ID            32769 non-null  int64
 3   ROLE_ROLLUP_1     32769 non-null  int64
 4   ROLE_ROLLUP_2     32769 non-null  int64
 5   ROLE_DEPTNAME     32769 non-null  int64
 6   ROLE_TITLE        32769 non-null  int64
 7   ROLE_FAMILY_DESC  32769 non-null  int64
 8   ROLE_FAMILY       32769 non-null  int64
 9   ROLE_CODE         32769 non-null  int64
dtypes: int64(10)
memory usage: 2.5 MB
```

In [5]:

```
#Describing dataset
dataset.describe()
```

Out[5]:

	ACTION	RESOURCE	MGR_ID	ROLE_ROLLUP_1	ROLE_ROLLUP_2	ROLE_D
count	32769.000000	32769.000000	32769.000000	32769.000000	32769.000000	32
mean	0.942110	42923.916171	25988.957979	116952.627788	118301.823156	118
std	0.233539	34173.892702	35928.031650	10875.563591	4551.588572	18
min	0.000000	0.000000	25.000000	4292.000000	23779.000000	4
25%	1.000000	20299.000000	4566.000000	117961.000000	118102.000000	118
50%	1.000000	35376.000000	13545.000000	117961.000000	118300.000000	118
75%	1.000000	74189.000000	42034.000000	117961.000000	118386.000000	120
max	1.000000	312153.000000	311696.000000	311178.000000	286791.000000	286

In [6]:

```
#finding correlation between all the features
corr_df = dataset.corr(method='pearson')
```

In [7]:

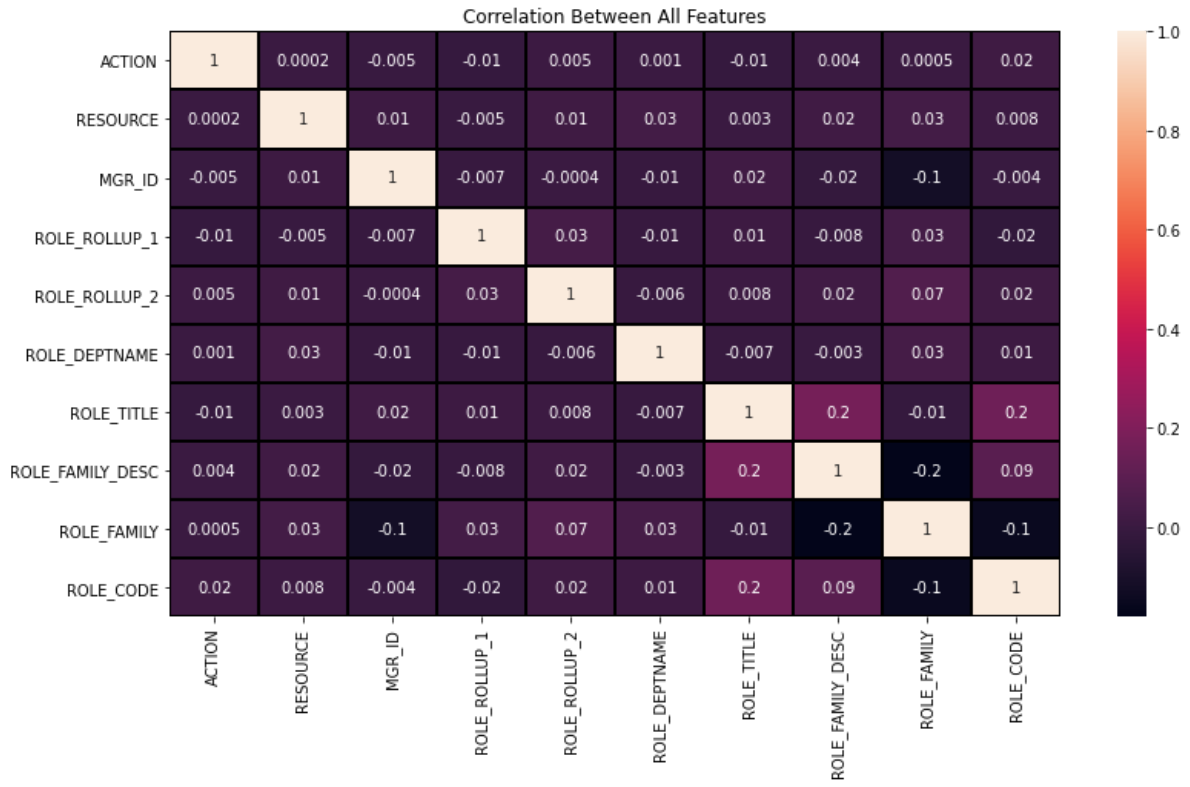
```
#printing correlation metrix
corr_df
```

Out[7]:

	ACTION	RESOURCE	MGR_ID	ROLE_ROLLUP_1	ROLE_ROLLUP_2	ROL
ACTION	1.000000	0.000185	-0.005167	-0.013702	0.005179	
RESOURCE	0.000185	1.000000	0.011088	-0.005016	0.013438	
MGR_ID	-0.005167	0.011088	1.000000	-0.007132	-0.000364	
ROLE_ROLLUP_1	-0.013702	-0.005016	-0.007132	1.000000	0.033358	
ROLE_ROLLUP_2	0.005179	0.013438	-0.000364	0.033358	1.000000	
ROLE_DEPTNAME	0.001025	0.030004	-0.009551	-0.009548	-0.006056	
ROLE_TITLE	-0.010169	0.002936	0.017864	0.010207	0.008305	
ROLE_FAMILY_DESC	0.003565	0.021029	-0.018488	-0.007546	0.018873	
ROLE_FAMILY	0.000502	0.031060	-0.118254	0.029468	0.069558	
ROLE_CODE	0.017147	0.007733	-0.004067	-0.024927	0.015117	

In [8]:

```
plt.figure(figsize=(13,7))
sns.heatmap(corr_df,annot=True,fmt='0.1g',linecolor='black',linewidths=1).set(ti
plt.show()
```



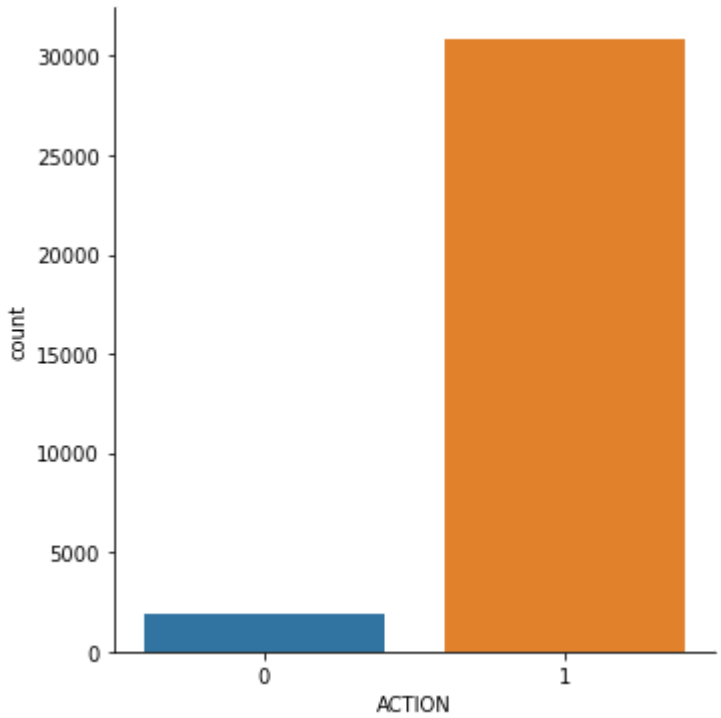
None of the feature is highly correlated with others.

In [9]:

```
#plotting Actions
sns.catplot(data=dataset,x=dataset.ACTION,kind='count')
```

Out[9]:

```
<seaborn.axisgrid.FacetGrid at 0x7fecf2aed9a0>
```



In [10]:

```
#counting 1 and 0 action
print(dataset.groupby('ACTION').get_group(0).value_counts())
```

```
ACTION  RESOURCE  MGR_ID  ROLE_ROLLUP_1  ROLE_ROLLUP_2  ROLE_DEPTNAME
ROLE_TITLE  ROLE_FAMILY_DESC  ROLE_FAMILY  ROLE_CODE
0       0         71215   117961         118386         118404
118784      118785            290919       118786       1
        42202     3838    117961         118225         119924
118321      118448            290919       118322       1
        42870     5228    119596         119597         168533
120789      301124            118424       120791       1
        42710     18096   118256         118257         118623
118995      286106            292795       118997       1
                  18073   118256         118257         118623
118995      286106            292795       118997       1

..
        25749     15510   118169         118170         118171
130479      185784            119784       130481       1
                  14348   118169         118170         118623
117946      144221            292795       117948       1
                  4561    118169         118170         118575
118636      215667            118638       118639       1
        25748     3053    117961         118225         120663
121594      131163            4673         121596       1
        312132    4084    117961         118052         118992
118321      240983            290919       118322       1
Length: 1897, dtype: int64
```

In [11]:

```
dataset.groupby('ACTION').describe()
```

Out[11]:

	RESOURCE								MGR_
	count	mean	std	min	25%	50%	75%	max	count
ACTION									
0	1897.0	42898.348972	35796.888757	0.0	20897.0	34924.0	52015.00	312132.0	1897
1	30872.0	42925.487205	34072.248084	0.0	20299.0	35376.0	74278.25	312153.0	30872

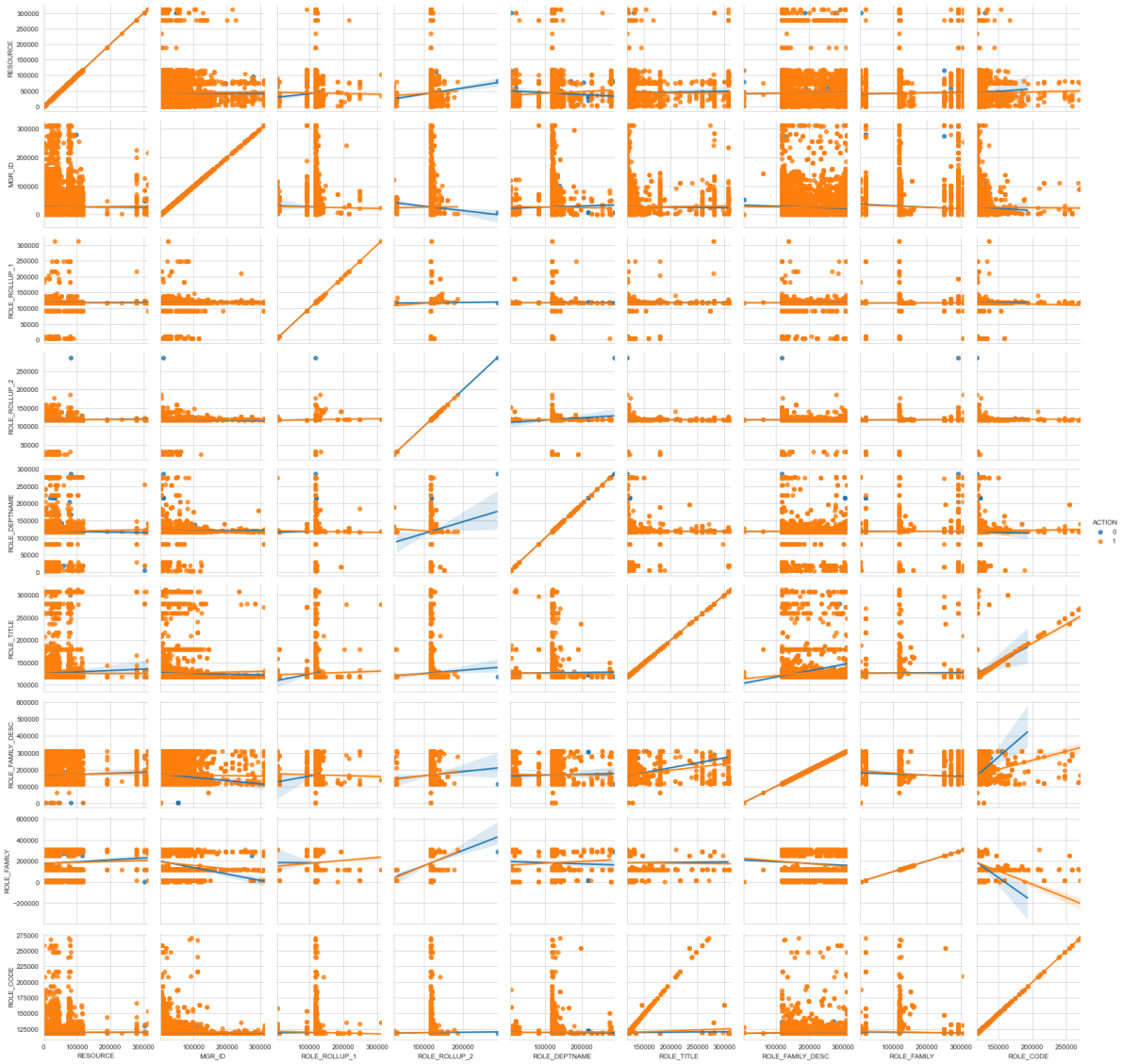
2 rows × 72 columns

No. of Action 0 is 1892 No. of Action 1 is 30872 No. of Action 0 is 5.8 % of the total train dataset

In [12]:

```
#plotting Regression graph between all the variables
plt.figure(figsize=(40,29))
sns.set_style(style='whitegrid')
dataset_plot = sns.PairGrid(dataset,hue='ACTION')
dataset_plot.map(sns.regplot)
dataset_plot.add_legend()
plt.show()
```

```
<Figure size 2880x2088 with 0 Axes>
```



In [ ]:

```

```

In [13]:

```
#Analysing Action 0
action_0 = dataset.groupby('ACTION').get_group(0)
```

In [14]:

```
action_0
```

Out[14]:

	ACTION	RESOURCE	MGR_ID	ROLE_ROLLUP_1	ROLE_ROLLUP_2	ROLE_DEPTNAME	F
5	0	45333	14561	117951	117952	118008	
41	0	74310	49521	117961	118300	118301	
55	0	78106	50613	117916	118150	118810	
66	0	35043	14800	117961	117962	118352	
68	0	15716	18073	118256	118257	118623	
...	...	...	...	...	...	...	
32645	0	25749	14348	118169	118170	118623	
32661	0	25993	7243	117961	118343	118609	
32681	0	14925	20361	117961	117962	118352	
32707	0	41470	14289	118602	118603	117941	
32713	0	77202	6982	117961	118300	118783	

1897 rows × 10 columns

In [15]:

```
#printing action 1 sampled data
action_0.shape
```

Out[15]:

```
(1897, 10)
```

In [16]:

```
1 action_0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1897 entries, 5 to 32713
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ACTION            1897 non-null   int64
 1   RESOURCE          1897 non-null   int64
 2   MGR_ID            1897 non-null   int64
 3   ROLE_ROLLUP_1     1897 non-null   int64
 4   ROLE_ROLLUP_2     1897 non-null   int64
 5   ROLE_DEPTNAME     1897 non-null   int64
 6   ROLE_TITLE        1897 non-null   int64
 7   ROLE_FAMILY_DESC  1897 non-null   int64
 8   ROLE_FAMILY       1897 non-null   int64
 9   ROLE_CODE         1897 non-null   int64
dtypes: int64(10)
memory usage: 163.0 KB
```

There is no null value in the 0 group and no of the rows in group is 1897

In [17]:

```
1 action_0.describe()
```

Out[17]:

	ACTION	RESOURCE	MGR_ID	ROLE_ROLLUP_1	ROLE_ROLLUP_2	ROLE_DEPTI
count	1897.0	1897.000000	1897.000000	1897.000000	1897.000000	1897.0
mean	0.0	42898.348972	26737.799157	117553.749605	118206.734844	118834.3
std	0.0	35796.888757	32238.816523	6509.927394	6976.743628	16208.2
min	0.0	0.000000	25.000000	5110.000000	31010.000000	6104.0
25%	0.0	20897.000000	4965.000000	117961.000000	118052.000000	118317.0
50%	0.0	34924.000000	16588.000000	117961.000000	118300.000000	118810.0
75%	0.0	52015.000000	46663.000000	118169.000000	118386.000000	120323.0
max	0.0	312132.000000	310767.000000	130570.000000	286791.000000	286792.0

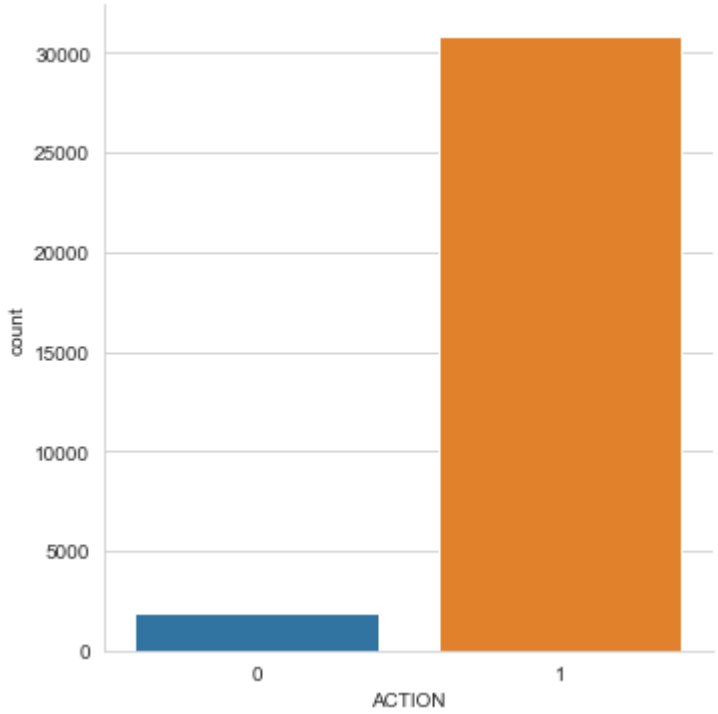
It represents the min max standard deviation and count of each column

In [18]:

```
#plotting Actions
sns.catplot(data=dataset,x=dataset.ACTION,kind='count')
```

Out[18]:

```
<seaborn.axisgrid.FacetGrid at 0x7fed315bc2e0>
```



In [19]:

```
#storing all action 1 rows in a df
action_1 = dataset.groupby('ACTION').get_group(1)
```

In [20]:

```
1 action_1
```

Out[20]:

	ACTION	RESOURCE	MGR_ID	ROLE_ROLLUP_1	ROLE_ROLLUP_2	ROLE_DEPTNAME	F
0	1	39353	85475	117961	118300	123472	
1	1	17183	1540	117961	118343	123125	
2	1	36724	14457	118219	118220	117884	
3	1	36135	5396	117961	118343	119993	
4	1	42680	5905	117929	117930	119569	
...	...	...	...	...	...	...	
32764	1	23497	16971	117961	118300	119993	
32765	1	25139	311198	91261	118026	122392	
32766	1	34924	28805	117961	118327	120299	
32767	1	80574	55643	118256	118257	117945	
32768	1	14354	59575	117916	118150	117920	

30872 rows × 10 columns

In [21]:

```
1 #sampling action 1 rows
2 sampled_action_1 = action_1.sample(1897)
```

In [22]:

```
1 #printing action 1 sampled data
2 sampled_action_1.shape
```

Out[22]:

```
(1897, 10)
```

In [23]:

```
1 #Joining both datasets action_0 and sampled_action_1
2 new_dataset = action_0.append(sampled_action_1,sort=True)
3 len(new_dataset)
```

Out[23]:

```
3794
```

In [24]:

```
#mixing up the rows
new_dataset = new_dataset.sample(len(new_dataset))
new_dataset
```

Out[24]:

	ACTION	MGR_ID	RESOURCE	ROLE_CODE	ROLE_DEPTNAME	ROLE_FAMILY	ROLE_FA
24782	1	44062	79501	118322	118910	290919	
11587	1	4370	4675	118786	118514	290919	
3553	0	7370	29119	118322	118481	290919	
25754	0	17970	33209	118055	148450	117887	
30688	0	4511	19962	118786	119181	290919	
...	...	...	...	...	...	...	
28317	0	257604	75639	118322	119214	290919	
16100	0	14758	74478	118705	120054	118704	
29853	1	49425	40069	118836	119598	118424	
5225	1	17058	25993	120419	119488	249618	
5120	1	7074	73806	118786	120096	290919	

3794 rows × 10 columns

In [25]:

```
#extracting features
X_feature = new_dataset.drop('ACTION',axis=1).values
```

In [26]:

```
Y_target = new_dataset['ACTION'].values
```

In [27]:

```
#Importing Modules for Prediction
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
logReg = LogisticRegression()
sc = StandardScaler()
```

In [28]:

```
#spliting training and test data
X_train,X_test,Y_train,Y_test = train_test_split(X_feature,Y_target)
```

In [29]:

```
#fitting the standard scaler module
sc.fit(X_train)
```

Out[29]:

```
StandardScaler()
```

In [30]:

```
#transforming training data into stander scaler
X_train = sc.transform(X_train)
```

In [31]:

```
#training the logistic regression model
logReg.fit(X_train,Y_train)
```

Out[31]:

```
LogisticRegression()
```

In [32]:

```
#making prediction from the model
Y_predict = logReg.predict(sc.transform(X_test))
```

In [33]:

```
#Eveluating ther Logistic Regression model
from sklearn.metrics import accuracy_score,average_precision_score,recall_score,
```

In [34]:

```
print('Logistic Regression Model with Standardization')
print('Accuracy Score is      ',accuracy_score(Y_test,Y_predict))
print('Precision Score is     ',average_precision_score(Y_test,Y_predict))
print('Recall Score is        ',recall_score(Y_test,Y_predict))
print('Confusion Metrix is    ')
print(confusion_matrix(Y_test,Y_predict))
print('F1 Score is            ',f1_score(Y_test,Y_predict))
print (classification_report(Y_test,Y_predict))
```

```
Logistic Regression Model with Standardization
Accuracy Score is      0.5458377239199157
Precision Score is     0.5272285549888402
Recall Score is        0.45588235294117646
Confusion Metrix is
[[301 172]
 [259 217]]
F1 Score is            0.5017341040462426
              precision    recall  f1-score   support

           0       0.54      0.64      0.58       473
           1       0.56      0.46      0.50       476

    accuracy                           0.55       949
   macro avg       0.55      0.55      0.54       949
weighted avg       0.55      0.55      0.54       949
```

In [35]:

```
#using knn classifier
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
```

In [36]:

```
#fitting knn model
knn.fit(X_train,Y_train)
```

Out[36]:

```
KNeighborsClassifier()
```

In [37]:

```
#making prediction from the KNNeighbors model
Y_predict = knn.predict(sc.transform(X_test))
```

In [38]:

```
print('KN Neighbors Model with Standardization')
print('Accuracy Score is      ',accuracy_score(Y_test,Y_predict))
print('Precision Score is     ',average_precision_score(Y_test,Y_predict))
print('Recall Score is        ',recall_score(Y_test,Y_predict))
print('Confusion Metrix is    ')
print(confusion_matrix(Y_test,Y_predict))
print('F1 Score is            ',f1_score(Y_test,Y_predict))
print (classification_report(Y_test,Y_predict))
```

```
KN Neighbors Model with Standardization
Accuracy Score is      0.6080084299262382
Precision Score is     0.567558554219657
Recall Score is        0.5945378151260504
Confusion Metrix is
[[294 179]
 [193 283]]
F1 Score is            0.6034115138592749
              precision    recall  f1-score   support

           0       0.60      0.62      0.61       473
           1       0.61      0.59      0.60       476

    accuracy                           0.61       949
   macro avg       0.61      0.61      0.61       949
weighted avg       0.61      0.61      0.61       949
```

In [39]:

```
#importing Decision tree Model with criterion gini
from sklearn.tree import DecisionTreeClassifier
dectree = DecisionTreeClassifier(criterion='gini',random_state = 20)
```

In [40]:

```
dectree.fit(X_train,Y_train)
```

Out[40]:

```
DecisionTreeClassifier(random_state=20)
```

In [41]:

```
Y_predict = dectree.predict(sc.transform(X_test))
```

In [42]:

```
print('Decision Tree(gini) Model with Standardization')
print('Accuracy Score is      ',accuracy_score(Y_test,Y_predict))
print('Precision Score is     ',average_precision_score(Y_test,Y_predict))
print('Recall Score is        ',recall_score(Y_test,Y_predict))
print('Confusion Metrix is    ')
print(confusion_matrix(Y_test,Y_predict))
print('F1 Score is            ',f1_score(Y_test,Y_predict))
print (classification_report(Y_test,Y_predict))
```

```
Decision Tree(gini) Model with Standardization
Accuracy Score is      0.7007376185458377
Precision Score is     0.6434283073196292
Recall Score is        0.6848739495798319
Confusion Metrix is
[[339 134]
 [150 326]]
F1 Score is            0.6965811965811967
              precision    recall  f1-score   support

           0       0.69      0.72      0.70       473
           1       0.71      0.68      0.70       476

    accuracy                           0.70       949
   macro avg       0.70      0.70      0.70       949
weighted avg       0.70      0.70      0.70       949
```

In [43]:

```
#decision tree with entropy
dt = DecisionTreeClassifier(criterion='entropy',random_state=20)
```

In [44]:

```
dt.fit(X_train,Y_train)
```

Out[44]:

```
DecisionTreeClassifier(criterion='entropy', random_state=20)
```

In [45]:

```
Y_predict = dt.predict(sc.transform(X_test))
```

In [46]:

```
print('Decision Tree(entropy) Model with Standardization')
print('Accuracy Score is     ',accuracy_score(Y_test,Y_predict))
print('Precision Score is    ',average_precision_score(Y_test,Y_predict))
print('Recall Score is       ',recall_score(Y_test,Y_predict))
print('Confusion Metrix is   ')
print(confusion_matrix(Y_test,Y_predict))
print('F1 Score is           ',f1_score(Y_test,Y_predict))
print (classification_report(Y_test,Y_predict))
```

```
Decision Tree(entropy) Model with Standardization
Accuracy Score is      0.7133825079030558
Precision Score is     0.6566513162745949
Recall Score is        0.6827731092436975
Confusion Metrix is
[[352 121]
 [151 325]]
F1 Score is            0.7049891540130152
              precision    recall  f1-score   support

           0       0.70      0.74      0.72       473
           1       0.73      0.68      0.70       476

    accuracy                           0.71       949
   macro avg       0.71      0.71      0.71       949
weighted avg       0.71      0.71      0.71       949
```

In [47]:

```
#importing Naive Bayes Model
from sklearn.naive_bayes import BernoulliNB
NB = BernoulliNB()
```

In [48]:

```
NB.fit(X_train,Y_train)
```

Out[48]:

```
BernoulliNB()
```

In [49]:

```
Y_predict = NB.predict(sc.transform(X_test))
```

In [50]:

```
print('Bernouli Naive Bayes) Model with Standardization')
print('Accuracy Score is     ',accuracy_score(Y_test,Y_predict))
print('Precision Score is    ',average_precision_score(Y_test,Y_predict))
print('Recall Score is       ',recall_score(Y_test,Y_predict))
print('Confusion Metrix is   ')
print(confusion_matrix(Y_test,Y_predict))
print('F1 Score is           ',f1_score(Y_test,Y_predict))
print (classification_report(Y_test,Y_predict))
```

```
Bernouli Naive Bayes) Model with Standardization
Accuracy Score is      0.5447839831401475
Precision Score is     0.5263553455323675
Recall Score is        0.4852941176470588
Confusion Metrix is
[[286 187]
 [245 231]]
F1 Score is            0.516778523489933
              precision    recall  f1-score   support

           0       0.54      0.60      0.57       473
           1       0.55      0.49      0.52       476

    accuracy                           0.54       949
   macro avg       0.55      0.54      0.54       949
weighted avg       0.55      0.54      0.54       949
```

In [51]:

```
#using svm model
from sklearn import svm
clf = svm.SVC()
```

In [52]:

```
clf.fit(X_train,Y_train)
```

Out[52]:

```
SVC()
```

In [53]:

```
Y_predict = clf.predict(sc.transform(X_test))
```

In [54]:

```
print('SVM Classifier Model with Standardization')
print('Accuracy Score is      ',accuracy_score(Y_test,Y_predict))
print('Precision Score is     ',average_precision_score(Y_test,Y_predict))
print('Recall Score is        ',recall_score(Y_test,Y_predict))
print('Confusion Metrix is    ')
print(confusion_matrix(Y_test,Y_predict))
print('F1 Score is            ',f1_score(Y_test,Y_predict))
print (classification_report(Y_test,Y_predict))
```

```
SVM Classifier Model with Standardization
Accuracy Score is      0.6143308746048473
Precision Score is     0.5715555538245712
Recall Score is        0.6218487394957983
Confusion Metrix is
[[287 186]
 [180 296]]
F1 Score is            0.6179540709812109
              precision    recall  f1-score   support

           0       0.61      0.61      0.61       473
           1       0.61      0.62      0.62       476

    accuracy                           0.61       949
   macro avg       0.61      0.61      0.61       949
weighted avg       0.61      0.61      0.61       949
```

In [55]:

```
#using randome forest classifier
from sklearn.ensemble import RandomForestClassifier
ranfo_gini = RandomForestClassifier(criterion='gini')
```

In [56]:

```
ranfo_gini.fit(X_train,Y_train)
```

Out[56]:

```
RandomForestClassifier()
```

In [57]:

```
Y_predict = ranfo_gini.predict(sc.transform(X_test))
```

In [58]:

```
print('Random Forest(gini) Model with Standardization')
print('Accuracy Score is      ',accuracy_score(Y_test,Y_predict))
print('Precision Score is     ',average_precision_score(Y_test,Y_predict))
print('Recall Score is        ',recall_score(Y_test,Y_predict))
print('Confusion Metrix is    ')
print(confusion_matrix(Y_test,Y_predict))
print('F1 Score is            ',f1_score(Y_test,Y_predict))
print (classification_report(Y_test,Y_predict))
```

```
Random Forest(gini) Model with Standardization
Accuracy Score is      0.761854583772392
Precision Score is     0.6994860177914533
Recall Score is        0.7710084033613446
Confusion Metrix is
[[356 117]
 [109 367]]
F1 Score is            0.7645833333333333
              precision    recall  f1-score   support

           0       0.77      0.75      0.76       473
           1       0.76      0.77      0.76       476

    accuracy                           0.76       949
   macro avg       0.76      0.76      0.76       949
weighted avg       0.76      0.76      0.76       949
```

In [59]:

```
ranfo_entropy = RandomForestClassifier(criterion='entropy')
```

In [60]:

```
ranfo_entropy.fit(X_train,Y_train)
```

Out[60]:

```
RandomForestClassifier(criterion='entropy')
```

In [61]:

```
Y_predict = ranfo_entropy.predict(sc.transform(X_test))
```

In [62]:

```
print('Random Forest (entropy) Model with Standardization')
print('Accuracy Score is      ',accuracy_score(Y_test,Y_predict))
print('Precision Score is     ',average_precision_score(Y_test,Y_predict))
print('Recall Score is        ',recall_score(Y_test,Y_predict))
print('Confusion Metrix is    ')
print(confusion_matrix(Y_test,Y_predict))
print('F1 Score is            ',f1_score(Y_test,Y_predict))
print (classification_report(Y_test,Y_predict))
```

```
Random Forest (entropy) Model with Standardization
Accuracy Score is      0.7692307692307693
Precision Score is     0.7077584962414187
Recall Score is        0.773109243697479
Confusion Metrix is
[[362 111]
 [108 368]]
F1 Score is            0.7706806282722513
              precision    recall  f1-score   support

           0       0.77      0.77      0.77       473
           1       0.77      0.77      0.77       476

    accuracy                           0.77       949
   macro avg       0.77      0.77      0.77       949
weighted avg       0.77      0.77      0.77       949
```

Clearly - Randome Forest Algorithm with criterion gini gives best accuracy and performs better .

In [63]:

```
#Importing test Dataset
test_dataset = pd.read_csv('/Users/rajeshsahu/Desktop/Data Science Learning/Data
```

In [64]:

```
1 test_dataset
```

Out[64]:

	id	RESOURCE	MGR_ID	ROLE_ROLLUP_1	ROLE_ROLLUP_2	ROLE_DEPTNAME	F
0	1	32642	7792	118573	118574	117945	
1	2	4696	14638	117961	118343	118514	
2	3	22662	1760	118887	118888	120171	
3	4	75078	7576	117961	118052	120671	
4	5	39879	55668	117902	118041	117945	
...	...	...	...	...	...	...	
912358	912359	32642	7792	118573	118574	117945	
912359	912360	41594	25262	126974	126975	118320	
912360	912361	45784	141	117961	118225	122273	
912361	912362	15716	2456	118752	118753	118754	
912362	912363	43443	7022	119062	119091	118535	

912363 rows × 10 columns

In [65]:

```
1 #randome forest prediction with test dataset
2 test_dataset.drop('id',axis=1,inplace=True)
```

In [66]:

```
1 test_dataset.head()
```

Out[66]:

	RESOURCE	MGR_ID	ROLE_ROLLUP_1	ROLE_ROLLUP_2	ROLE_DEPTNAME	ROLE_TITLE	R
0	32642	7792	118573	118574	117945	136261	
1	4696	14638	117961	118343	118514	118321	
2	22662	1760	118887	118888	120171	118396	
3	75078	7576	117961	118052	120671	118321	
4	39879	55668	117902	118041	117945	135951	

In [67]:

```
1 test_dataset.shape
```

Out[67]:

```
(912363, 9)
```

In [68]:

```
print(ranfo_gini.predict(sc.transform(test_dataset)))
```

```
[1 1 1 ... 1 1 1]
```

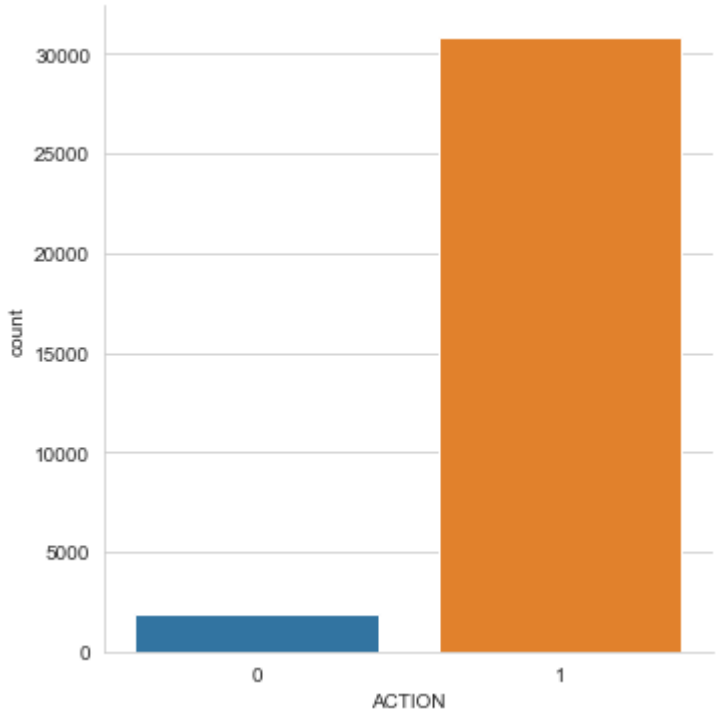
random forest prediction with biased training dataset

In [69]:

```
##random forest prediction with biased training dataset
#plotting Actions
sns.catplot(data=dataset,x=dataset.ACTION,kind='count')
```

Out[69]:

```
<seaborn.axisgrid.FacetGrid at 0x7fed31352790>
```



In [70]:

```
X = dataset.drop('ACTION',axis=1).values
```

In [71]:

```
Y = dataset.ACTION.values
```

In [72]:

```
#spliting training and test data
X_train,X_test,Y_train,Y_test = train_test_split(X,Y)
```

In [73]:

```
sc_biased = StandardScaler()
```

In [74]:

```
sc_biased.fit(X_train)
```

Out[74]:

```
StandardScaler()
```

In [75]:

```
X_train = sc_biased.transform(X_train)
X_test = sc_biased.transform(X_test)
```

In [76]:

```
ranfo_gini_biased = RandomForestClassifier(criterion='gini')
```

In [77]:

```
ranfo_gini_biased.fit(X_train,Y_train)
```

Out[77]:

```
RandomForestClassifier()
```

In [78]:

```
Y_predict = ranfo_gini_biased.predict(X_test)
```

In [79]:

```
print('Random Forest(gini) Model with Biased dataset')
print('Accuracy Score is      ',accuracy_score(Y_test,Y_predict))
print('Precision Score is     ',average_precision_score(Y_test,Y_predict))
print('Recall Score is        ',recall_score(Y_test,Y_predict))
print('Confusion Metrix is    ')
print(confusion_matrix(Y_test,Y_predict))
print('F1 Score is            ',f1_score(Y_test,Y_predict))
print (classification_report(Y_test,Y_predict))
```

```
Random Forest(gini) Model with Biased dataset
Accuracy Score is       0.9459294519711949
Precision Score is      0.9600679716197801
Recall Score is         0.9829892221789378
Confusion Metrix is
[[ 180  312]
 [ 131 7570]]
F1 Score is             0.9715715844189181
              precision    recall  f1-score   support

           0       0.58      0.37      0.45       492
           1       0.96      0.98      0.97      7701

    accuracy                           0.95      8193
   macro avg       0.77      0.67      0.71      8193
weighted avg       0.94      0.95      0.94      8193
```

Here the stats are facinating like -

Accuracy Score is 0.9461735627975101 Precision Score is 0.9581451130890075 Recall Score is 0.9854469854469855

But The confusion matrix is telling the truth that -

112 rows out of total 280 rows of class 0, it has predicted as class 1 which is appoximately 40%.i.e.

40% times it is predicting wrong results for class 0.

Comparison between baised and unbiased prediction results

	precision	recall	f1-score	support
for Biased datset macro avg	0.78	0.66	0.70	8193
for Unbiased dataset macro avg	0.76	0.76	0.76	949

clearly Unbiased dataset performs better

In [ ]:

```
1
```