STARTUP COMPANIES PREDICTION

IMOPRTING IMPORTANT LIBARARIES

In [1]:

```
import pandas as pd   #import pandas
import seaborn as sns
import csv
import numpy as np    #import numpy
import matplotlib.pyplot as plt
import warnings                    #import waarnigs to ingonre any kind of warning while
warnings.filterwarnings("ignore")
```

IMPORTING CSV AS DATAFRAME

In [2]:

```
df=pd.read_csv("startup_companies.csv")
```

CHECKING FIRST FIVE RECODS IN DATAFRAME

In [3]:

```
df.head()
```

Out[3]:

	ID	Name	Category	Subcategory	Country	Launched	Deadline	Goal	Ple
0	1860890148	Grace Jones Does Not Give A F$#% T-Shirt (limi...	Fashion	Fashion	United States	2009-04-21 21:02:48	2009-05-31	1000	
1	709707365	CRYSTAL ANTLERS UNTITLED MOVIE	Film & Video	Shorts	United States	2009-04-23 00:07:53	2009-07-20	80000	
2	1703704063	drawing for dollars	Art	Illustration	United States	2009-04-24 21:52:03	2009-05-03	20	
3	727286	Offline Wikipedia iPhone app	Technology	Software	United States	2009-04-25 17:36:21	2009-07-14	99	
4	1622952265	Pantshirts	Fashion	Fashion	United States	2009-04-27 14:10:39	2009-05-26	1900	

CHECKING LAST FIVE RECODS IN DATAFRAME

In [4]:

```
df.tail()
```

Out[4]:

	ID	Name	Category	Subcategory	Country	Launched	Deadline	Goa
374848	1486845240	Americas Got Talent - Serious MAK	Music	Hip-Hop	United States	2018-01-02 14:13:09	2018-01-16	50
374849	974738310	EVO Planner: The World's First Personalized Fl...	Design	Product Design	United States	2018-01-02 14:15:38	2018-02-09	1500
374850	2106246194	Help save La Gattara, Arizona's first Cat Cafe!	Food	Food	United States	2018-01-02 14:17:46	2018-01-16	1000
374851	1830173355	Digital Dagger Coin	Art	Art	United States	2018-01-02 14:38:17	2018-02-01	65
374852	1339173863	Spirits of the Forest	Games	Tabletop Games	Spain	2018-01-02 15:02:31	2018-01-26	2427

In [5]:

```
newdf=df.copy()
```

In [6]:

```
df['State'].value_counts()
```

Out[6]:

```
Failed        197611
Successful    133851
Canceled       38751
Live            2798
Suspended       1842
Name: State, dtype: int64
```

CHECKING ROWS AND COLUMNW USING SHAPE FUNCTION

In [7]:

```
df.shape
```

Out[7]:

```
(374853, 11)
```

CHECKING DATATYPES AND COLUMNS

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374853 entries, 0 to 374852
Data columns (total 11 columns):
 #   Column       Non-Null Count   Dtype 
---  ------       --------------   ----- 
 0   ID           374853 non-null  int64 
 1   Name         374853 non-null  object
 2   Category     374853 non-null  object
 3   Subcategory  374853 non-null  object
 4   Country      374853 non-null  object
 5   Launched     374853 non-null  object
 6   Deadline     374853 non-null  object
 7   Goal         374853 non-null  int64 
 8   Pledged      374853 non-null  int64 
 9   Backers      374853 non-null  int64 
 10  State        374853 non-null  object
dtypes: int64(4), object(7)
memory usage: 31.5+ MB
```

CHECKING MISSING VALUES

In [9]:

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

Out[9]:

```
ID             0
Name           0
Category       0
Subcategory    0
Country        0
Launched       0
Deadline       0
Goal           0
Pledged        0
Backers        0
State          0
dtype: int64
```

In [10]:

```
df.describe()
```

Out[10]:

	ID	Goal	Pledged	Backers
count	3.748530e+05	3.748530e+05	3.748530e+05	374853.000000
mean	1.074656e+09	4.586378e+04	9.121073e+03	106.690359
std	6.191377e+08	1.158778e+06	9.132054e+04	911.718520
min	5.971000e+03	0.000000e+00	0.000000e+00	0.000000
25%	5.380728e+08	2.000000e+03	3.100000e+01	2.000000
50%	1.075300e+09	5.500000e+03	6.250000e+02	12.000000
75%	1.610149e+09	1.600000e+04	4.051000e+03	57.000000
max	2.147476e+09	1.663614e+08	2.033899e+07	219382.000000

CHECKING UNIQUE VALUES OF ALL COLUMNS

In [11]:

```
df.nunique()
```

Out[11]:

```
ID             374853
Name           372061
Category           15
Subcategory       159
Country            22
Launched       374297
Deadline         3164
Goal            27692
Pledged         39989
Backers          3963
State               5
dtype: int64
```

CHECKING SKEWNESS

In [12]:

```
df.skew()
```

Out[12]:

```
ID          -0.002575
Goal        77.828151
Pledged     82.003924
Backers     86.339961
dtype: float64
```

CONVERTING OBJECT INTO DATETIMEFRAME

In [13]:

```
df['Launch']=pd.to_datetime(df['Launched']).dt.date
df['LaunchTime']=pd.to_datetime(df['Launched']).dt.time
df['Launch1']=pd.to_datetime(df['Launch'])
```

In [14]:

```
df['Deadline']=pd.to_datetime(df['Deadline'])
df['DeadlineYear']=df['Deadline'].dt.year
df['LaunchYear']=df['Launch1'].dt.year
df['Deadlinemonth']=df['Deadline'].dt.month
df['LaunchMonth']=df['Launch1'].dt.month
df['DeadlineDay']=df['Deadline'].dt.day
df['LaunchDay']=df['Launch1'].dt.day
```

In [15]:

```
df.head()
```

Out[15]:

	ID	Name	Category	Subcategory	Country	Launched	Deadline	Goal	Ple
0	1860890148	Grace Jones Does Not Give A F$#% T-Shirt (limi...	Fashion	Fashion	United States	2009-04-21 21:02:48	2009-05-31	1000	
1	709707365	CRYSTAL ANTLERS UNTITLED MOVIE	Film & Video	Shorts	United States	2009-04-23 00:07:53	2009-07-20	80000	
2	1703704063	drawing for dollars	Art	Illustration	United States	2009-04-24 21:52:03	2009-05-03	20	
3	727286	Offline Wikipedia iPhone app	Technology	Software	United States	2009-04-25 17:36:21	2009-07-14	99	
4	1622952265	Pantshirts	Fashion	Fashion	United States	2009-04-27 14:10:39	2009-05-26	1900	

In [16]:

```
df.drop(['Launched','Deadline','Launch','Launch1'],axis=1,inplace=True)
```

In [17]:

```
df.head()
```

Out[17]:

	ID	Name	Category	Subcategory	Country	Goal	Pledged	Backers	
0	1860890148	Grace Jones Does Not Give A F$#% T-Shirt (limi...	Fashion	Fashion	United States	1000	625	30	F
1	709707365	CRYSTAL ANTLERS UNTITLED MOVIE	Film & Video	Shorts	United States	80000	22	3	F
2	1703704063	drawing for dollars	Art	Illustration	United States	20	35	3	Succ
3	727286	Offline Wikipedia iPhone app	Technology	Software	United States	99	145	25	Succ
4	1622952265	Pantshirts	Fashion	Fashion	United States	1900	387	10	F

CHECKING NAN VALUES IN COLUMNS

In [18]:

```
df['Category'].value_counts()
```

Out[18]:

```
Film & Video    62694
Music           49529
Publishing      39378
Games           35225
Technology      32562
Design          30065
Art             28151
Food            24599
Fashion         22812
Theater         10911
Comics          10819
Photography     10778
Crafts           8809
Journalism       4754
Dance            3767
Name: Category, dtype: int64
```

In [19]:

```
df['Subcategory'].value_counts()
```

Out[19]:

```
Product Design     22310
Documentary        16138
Tabletop Games     14178
Music              13339
Shorts             12357
                   ...
Residencies           69
Letterpress           49
Chiptune              35
Literary Spaces       27
Taxidermy             13
Name: Subcategory, Length: 159, dtype: int64
```

In [20]:

```
df['Country'].value_counts()
```

Out[20]:

```
United States     292618
United Kingdom     33671
Canada             14756
Australia           7839
Germany             4171
France              2939
Italy               2878
Netherlands         2868
Spain               2276
Sweden              1757
Mexico              1752
New Zealand         1447
Denmark             1113
Ireland              811
Switzerland          760
Norway               708
Hong Kong            618
Belgium              617
Austria              597
Singapore            555
Luxembourg            62
Japan                 40
Name: Country, dtype: int64
```

In [21]:

```
df.columns
```

Out[21]:

```
Index(['ID', 'Name', 'Category', 'Subcategory', 'Country', 'Goal', 'Pledge
d',
       'Backers', 'State', 'LaunchTime', 'DeadlineYear', 'LaunchYear',
       'Deadlinemonth', 'LaunchMonth', 'DeadlineDay', 'LaunchDay'],
      dtype='object')
```

DIVIDING DATA INTO CATEGROICAL AND NUMERICAL COLUMNS

In [22]:

```
numcol=df.select_dtypes(["int64","float64"]).columns
catcol=df.select_dtypes(["object"]).columns
```

numcol SHOWS NUMERICAL COLUMNS

In [23]:

```
numcol
```

Out[23]:

```
Index(['ID', 'Goal', 'Pledged', 'Backers', 'DeadlineYear', 'LaunchYear',
       'Deadlinemonth', 'LaunchMonth', 'DeadlineDay', 'LaunchDay'],
      dtype='object')
```

catcol SHOWS CATEGROICAL COLUMNS

In [24]:

```
catcol
```

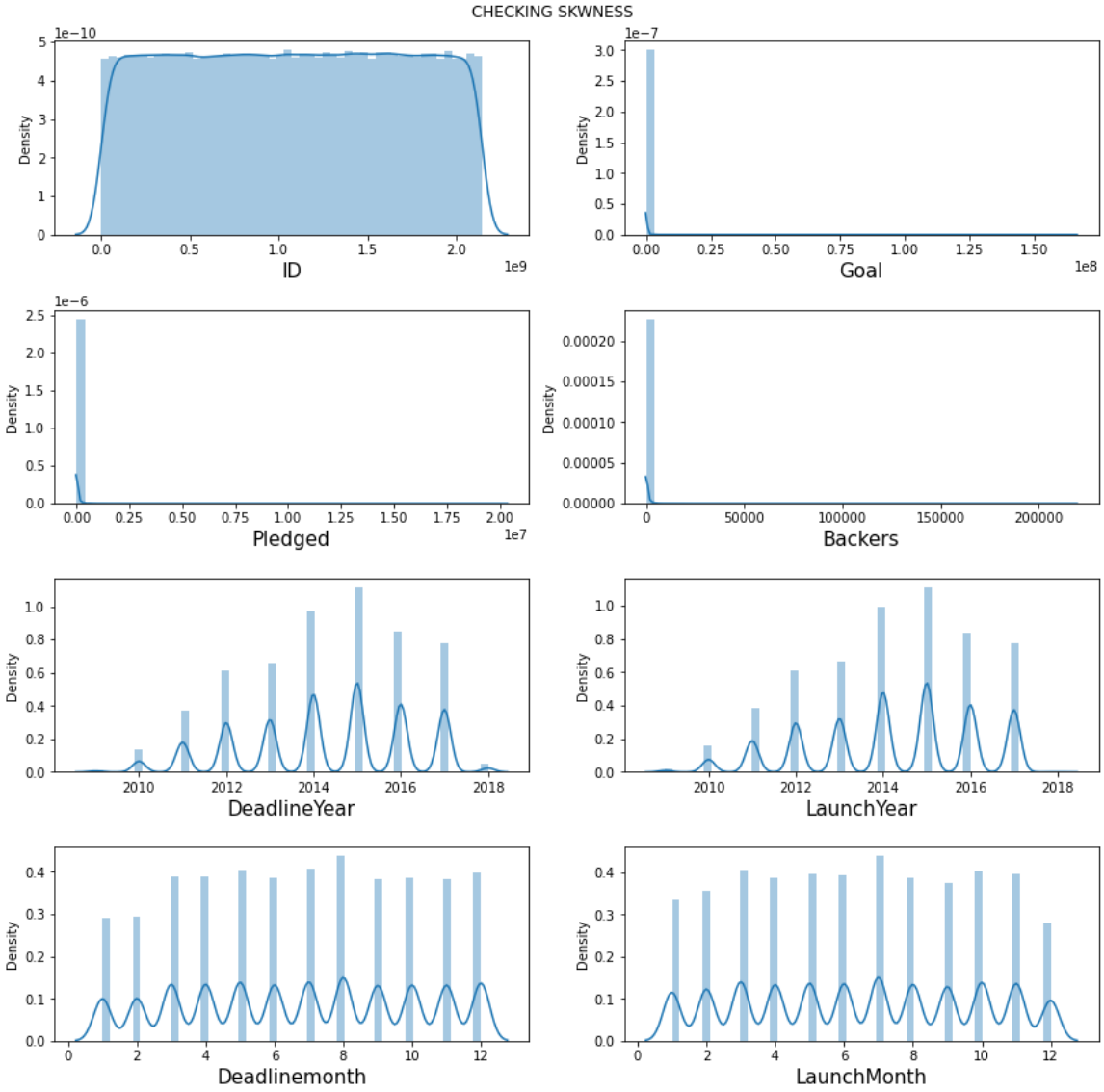
Out[24]:

```
Index(['Name', 'Category', 'Subcategory', 'Country', 'State', 'LaunchTim
e'], dtype='object')
```

CHECKING SKWNESS

In [25]:

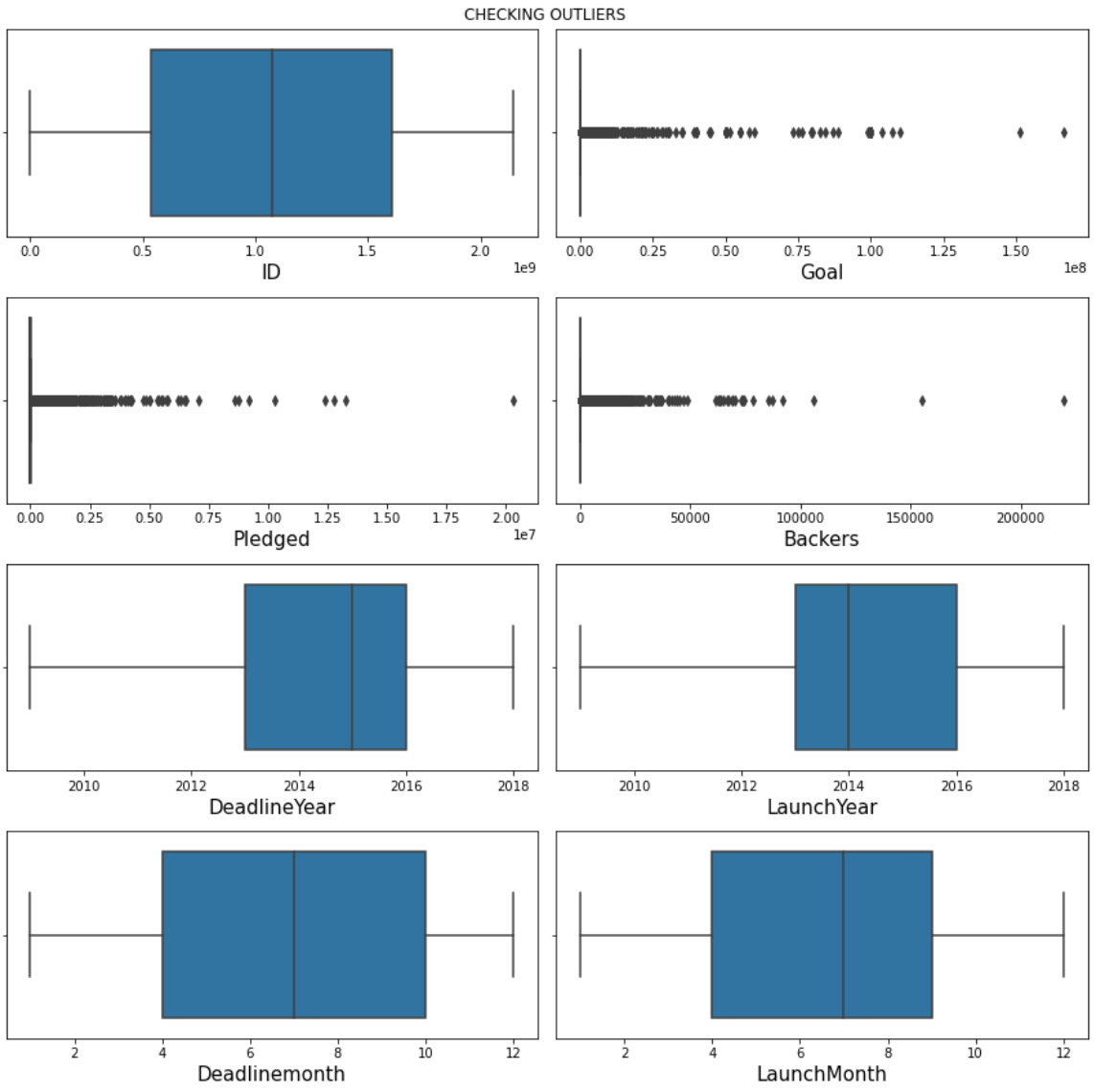
```
plt.figure(figsize=(12,12))
plt.suptitle("CHECKING SKWNESS")
pltn=1
for i in numcol:
    if pltn<=8:
        ax=plt.subplot(4,2,pltn)
        sns.distplot(df[i])
        plt.xlabel(i,fontsize=15)
    pltn=pltn+1
plt.tight_layout()
```



CHECKING OUTLIERS

In [26]:

```
plt.figure(figsize=(12,12))
plt.suptitle("CHECKING OUTLIERS")
pltn=1
for i in numcol:
    if pltn<=8:
        ax=plt.subplot(4,2,pltn)
        sns.boxplot(df[i])
        plt.xlabel(i,fontsize=15)
    pltn=pltn+1
plt.tight_layout()
```



REMOVING OUTLIERS FROM Pledged,Goal,Backers

In [27]:

```
q25=df["Pledged"].quantile(0.25)
q75=df["Pledged"].quantile(0.75)
iqr=q75-q25
ul=q75+1.5*iqr
ll=q25-1.5*iqr
df['Pledged']=np.where(df['Pledged']>ul,ul,np.where(df['Pledged']<ll,ll,df['Pledged']))
```

In [28]:

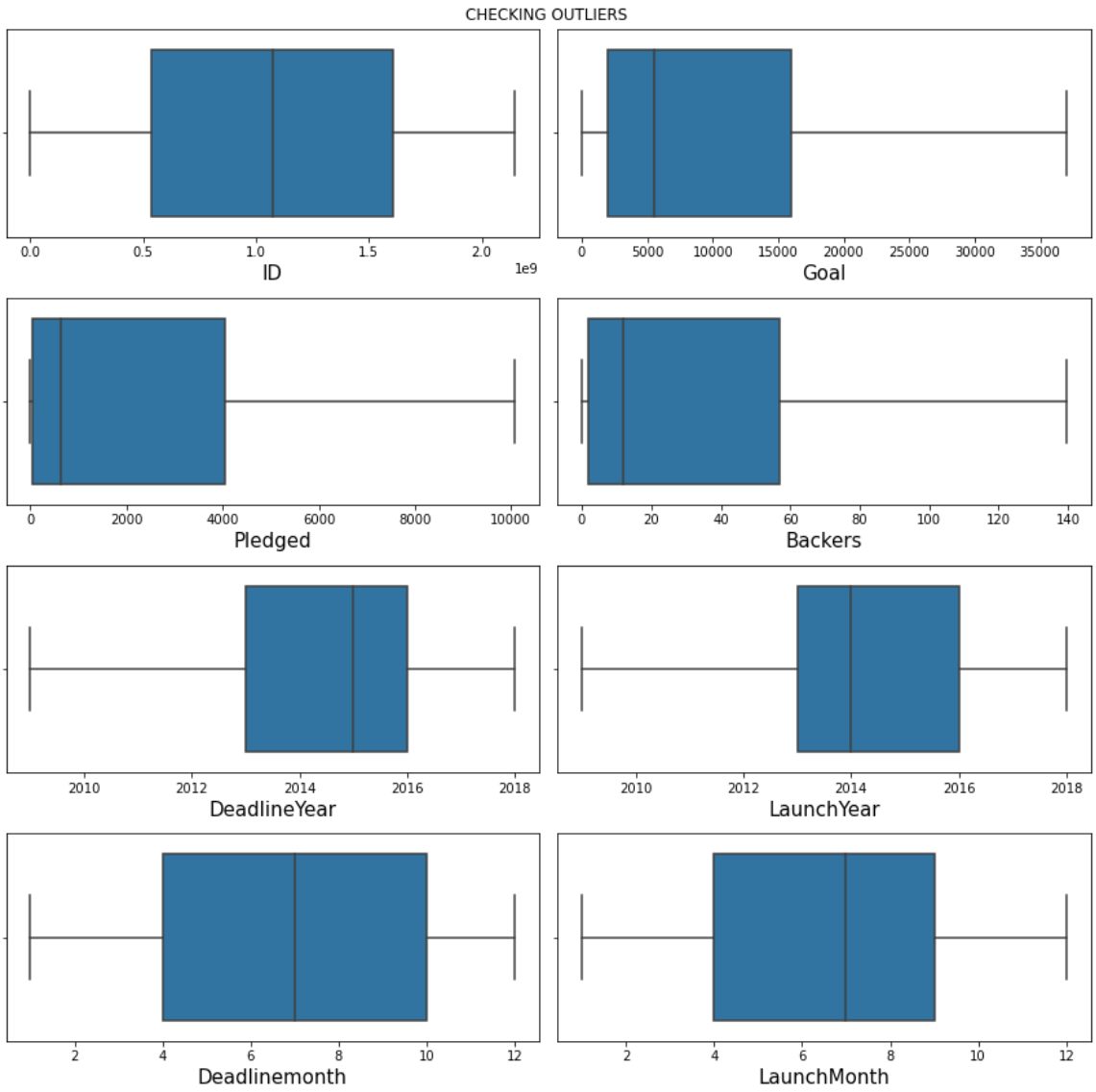
```
q25=df["Goal"].quantile(0.25)
q75=df["Goal"].quantile(0.75)
iqr=q75-q25
ul=q75+1.5*iqr
ll=q25-1.5*iqr
df['Goal']=np.where(df['Goal']>ul,ul,np.where(df['Goal']<ll,ll,df['Goal']))
```

In [29]:

```
q25=df["Backers"].quantile(0.25)
q75=df["Backers"].quantile(0.75)
iqr=q75-q25
ul=q75+1.5*iqr
ll=q25-1.5*iqr
df['Backers']=np.where(df['Backers']>ul,ul,np.where(df['Backers']<ll,ll,df['Backers']))
```

In [30]:

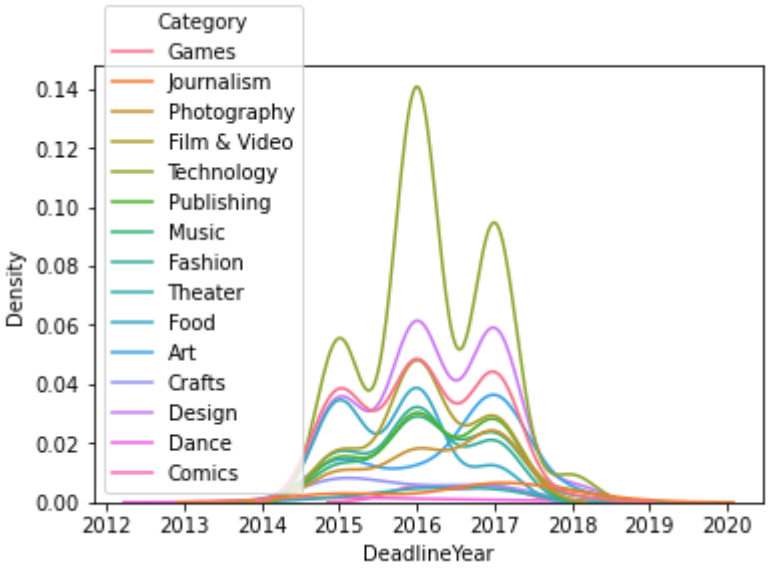
```
plt.figure(figsize=(12,12))
plt.suptitle("CHECKING OUTLIERS")
pltn=1
for i in numcol:
    if pltn<=8:
        ax=plt.subplot(4,2,pltn)
        sns.boxplot(df[i])
        plt.xlabel(i,fontsize=15)
    pltn=pltn+1
plt.tight_layout()
```



In [31]:

```
plt.figure(figsize=(10,8))
fig,axes=plt.subplots(1,1)
sns.kdeplot(data=df[df['Country']=='Austria'],x='DeadlineYear',hue='Category')
plt.show()
```

```
<Figure size 720x576 with 0 Axes>
```

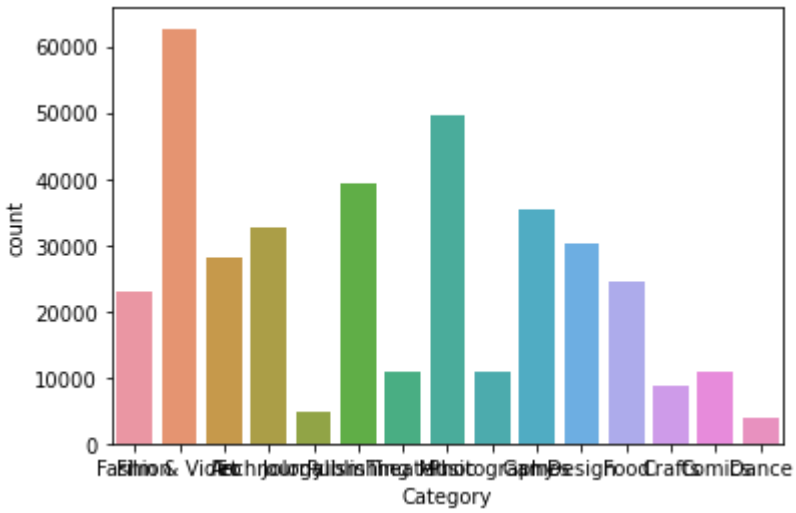


In [32]:

```
sns.countplot(x='Category',data=df)
```

Out[32]:

```
<AxesSubplot:xlabel='Category', ylabel='count'>
```



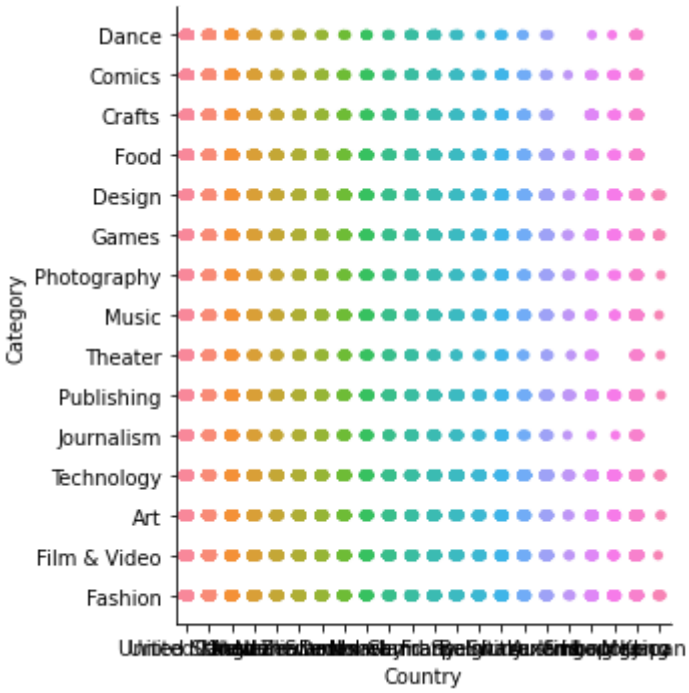
In [33]:

```
plt.figure(figsize=(16,8))
sns.catplot(x='Country',y='Category',data=df)
```

Out[33]:

```
<seaborn.axisgrid.FacetGrid at 0x1c753e32c40>

<Figure size 1152x576 with 0 Axes>
```

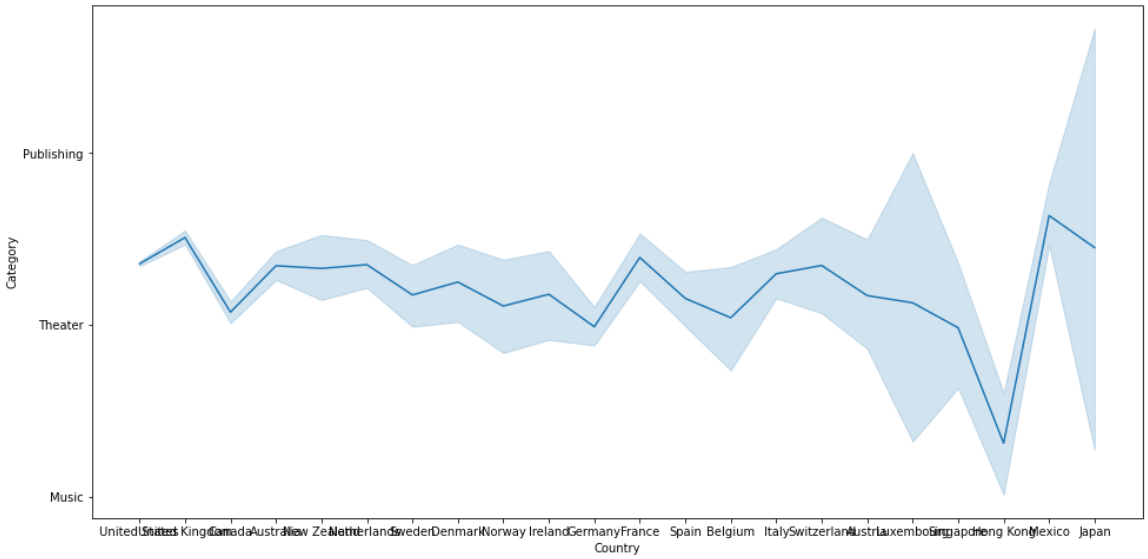


In [34]:

```
plt.figure(figsize=(16,8))
sns.lineplot(x='Country',y='Category',data=df)
```

Out[34]:

```
<AxesSubplot:xlabel='Country', ylabel='Category'>
```



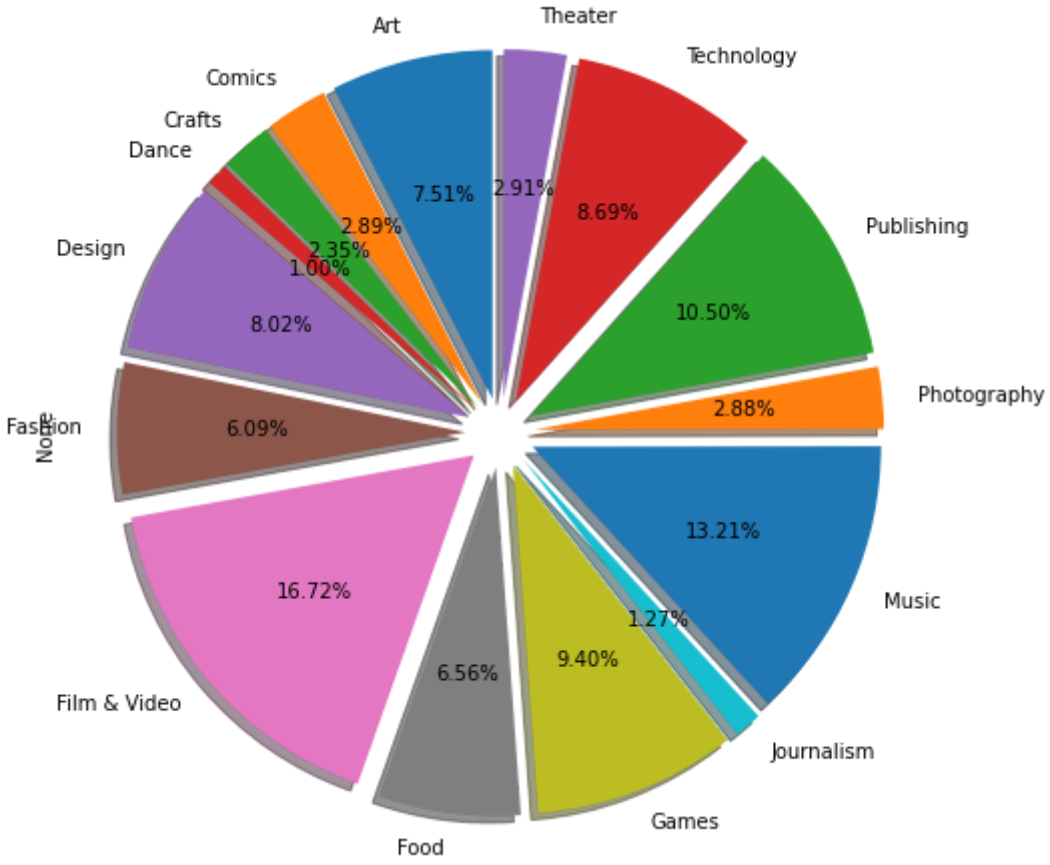
SHOWING PIE PLOT ON CATEGORY

In [35]:

```
plt.figure(figsize=(16,8))
df.groupby('Category').size().plot(kind='pie',autopct='%1.2f%%',shadow=True,startangle=90
```

Out[35]:

```
<AxesSubplot:ylabel='None'>
```

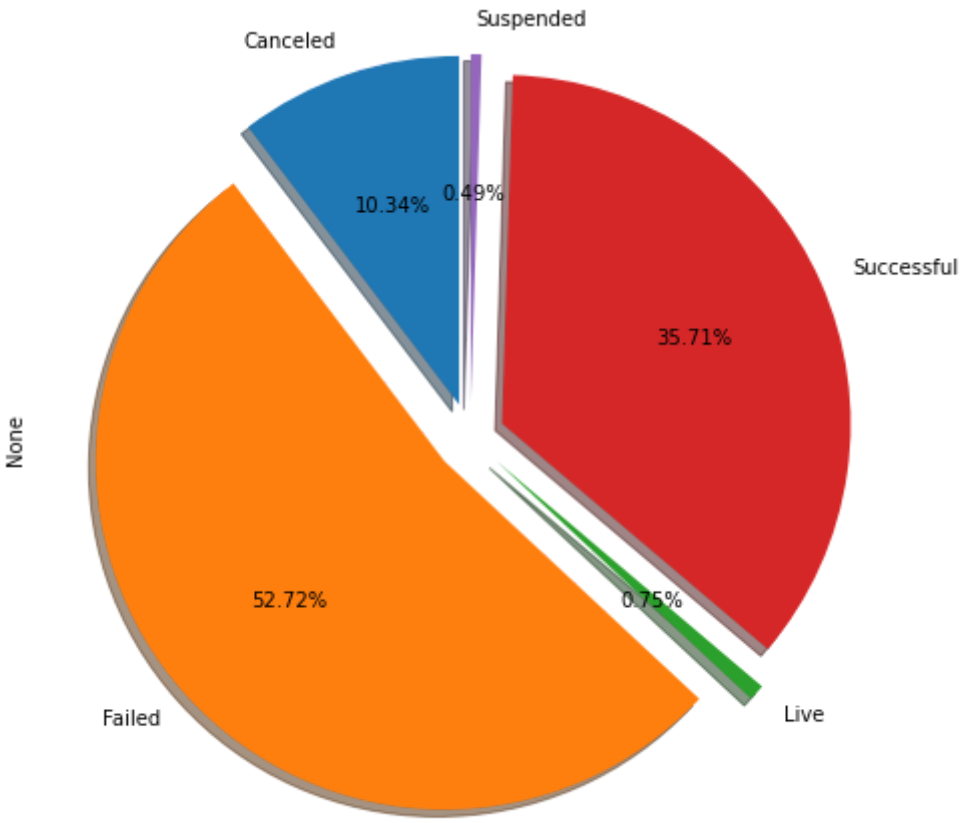


In [36]:

```
plt.figure(figsize=(26,8))
df.groupby('State').size().plot(kind='pie',autopct='%1.2f%%',shadow=True,startangle=90,ex
```

Out[36]:

```
<AxesSubplot:ylabel='None'>
```

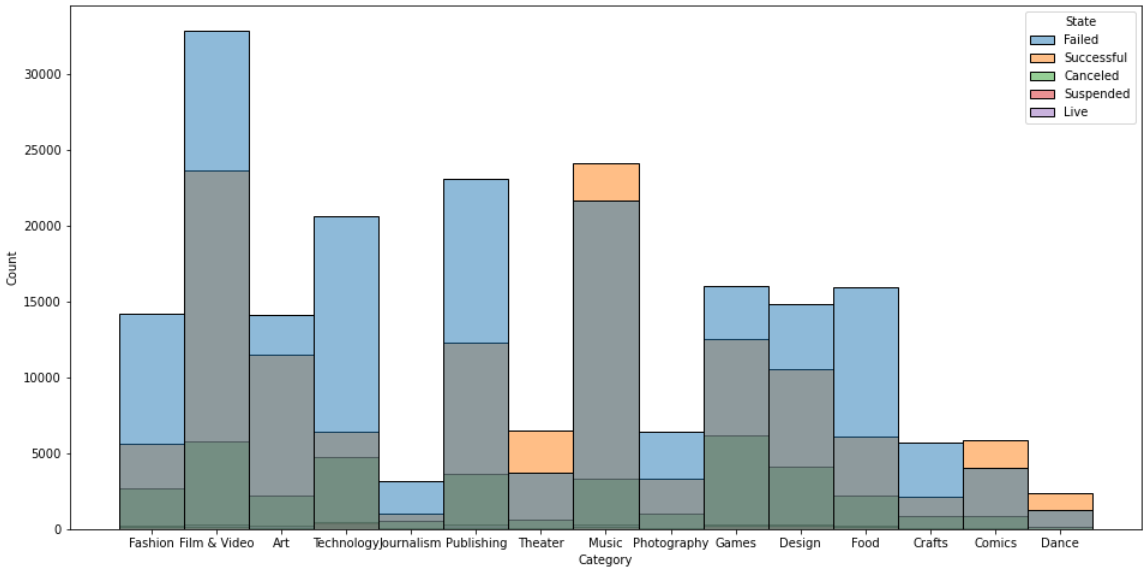


In [37]:

```
plt.figure(figsize=(16,8))
sns.histplot(x='Category',hue='State',data=df)
```

Out[37]:

```
<AxesSubplot:xlabel='Category', ylabel='Count'>
```

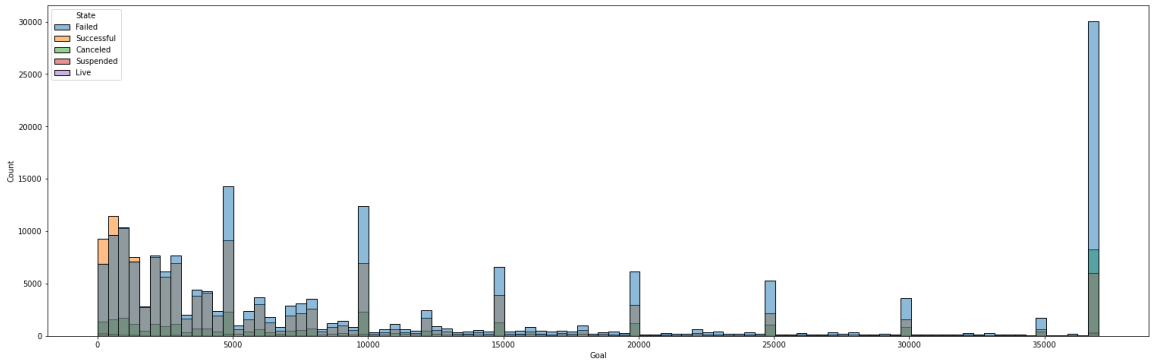


In [38]:

```
plt.figure(figsize=(26,8))
sns.histplot(x='Goal',hue='State',data=df)
```

Out[38]:

```
<AxesSubplot:xlabel='Goal', ylabel='Count'>
```

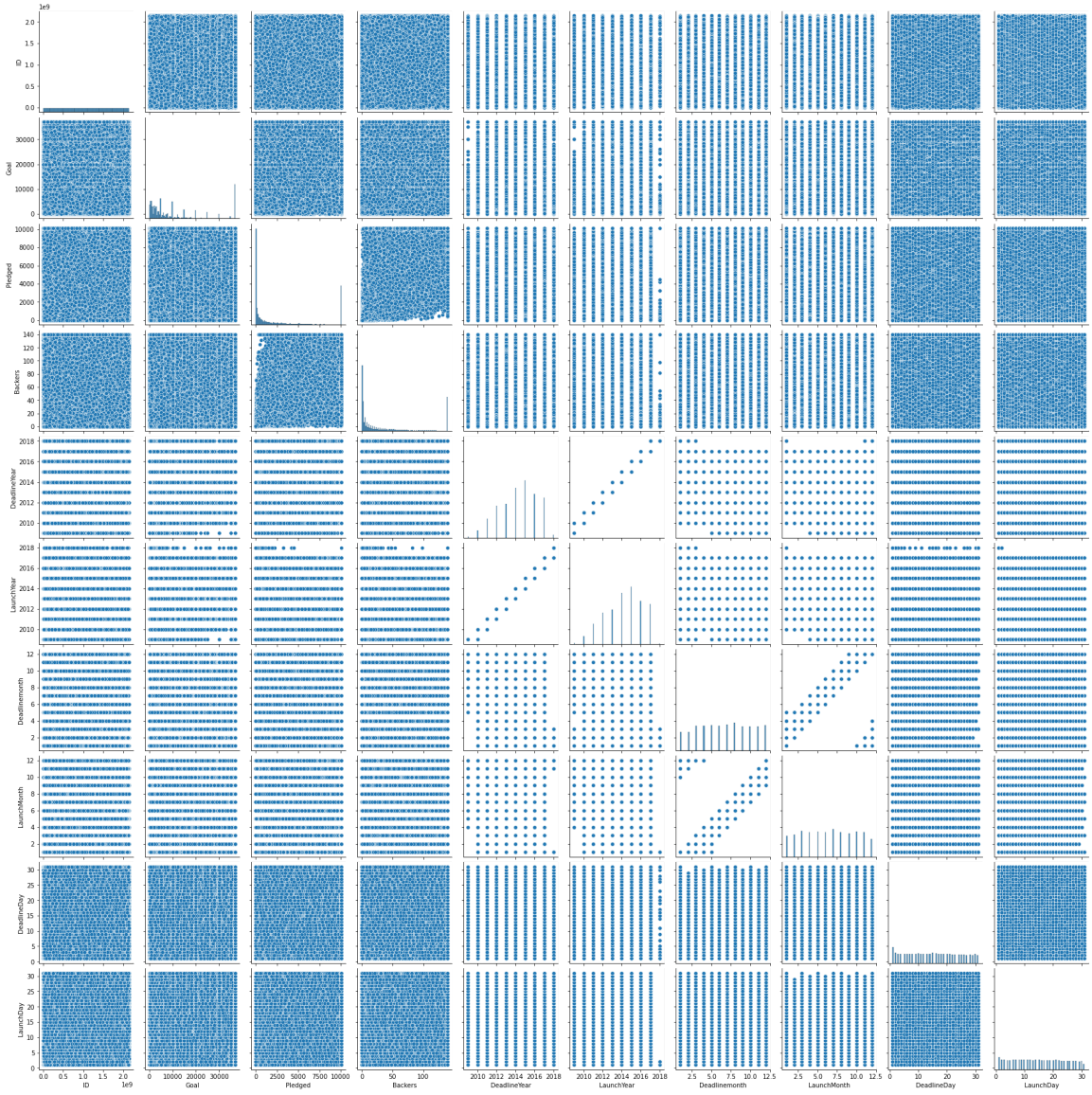


In [39]:

```
sns.pairplot(df)
```

Out[39]:

```
<seaborn.axisgrid.PairGrid at 0x1c75f0ade50>
```



CONVERTING CATEGORICAL COLUMNS INTO NUMERICAL

In [40]:

```
from sklearn.preprocessing import OrdinalEncoder
oe=OrdinalEncoder()
df[catcol]=oe.fit_transform(df[catcol])
```

In [41]:

```
df.columns
```

Out[41]:

```
Index(['ID', 'Name', 'Category', 'Subcategory', 'Country', 'Goal', 'Pledge
d',
       'Backers', 'State', 'LaunchTime', 'DeadlineYear', 'LaunchYear',
       'Deadlinemonth', 'LaunchMonth', 'DeadlineDay', 'LaunchDay'],
      dtype='object')
```

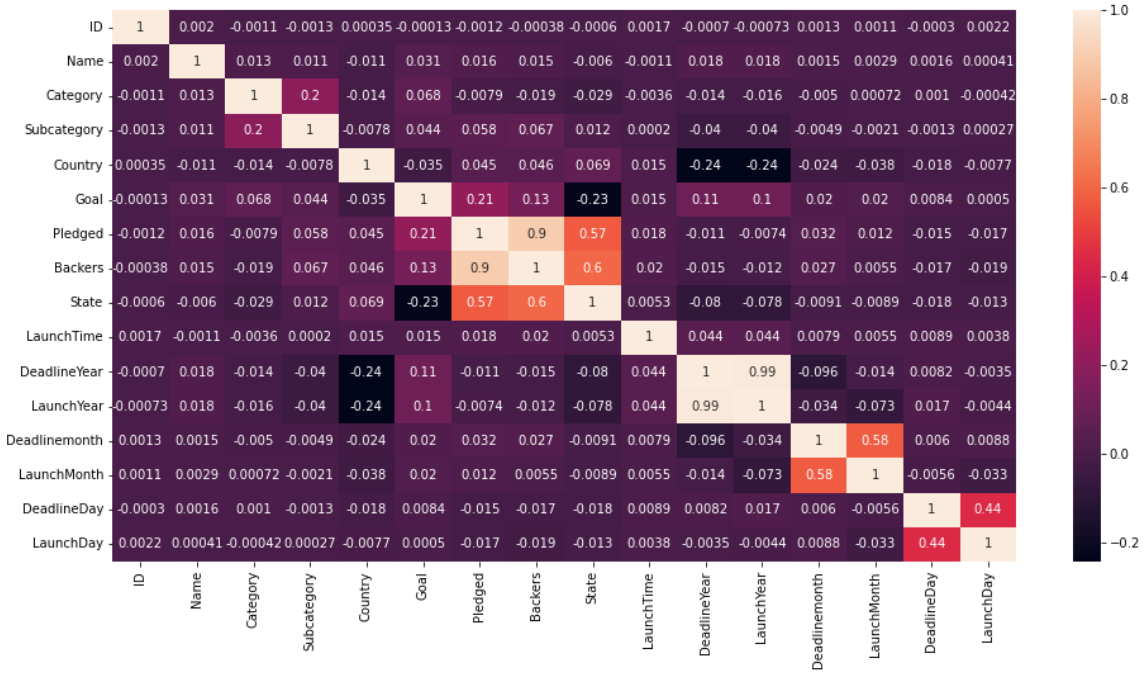
CHECKING CORREALTION

In [42]:

```
plt.figure(figsize=(16,8))
sns.heatmap(df.corr(),annot=True)
```

Out[42]:

<AxesSubplot:>



In [43]:

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

In [44]:

```
df
```

Out[44]:

	Category	Subcategory	Country	Goal	Pledged	Backers	State	LaunchTime	Dea
0	5.0	52.0	21.0	1000.0	625.0	30.0	1.0	70698.0	
1	6.0	129.0	21.0	37000.0	22.0	3.0	1.0	473.0	
2	0.0	70.0	21.0	20.0	35.0	3.0	3.0	73649.0	
3	13.0	131.0	21.0	99.0	145.0	25.0	3.0	58323.0	
4	5.0	52.0	21.0	1900.0	387.0	10.0	1.0	46262.0	
...	...	...	...	...	...	...	...	...	
374848	10.0	68.0	21.0	500.0	0.0	0.0	2.0	46398.0	
374849	4.0	113.0	21.0	15000.0	269.0	8.0	2.0	46535.0	
374850	7.0	58.0	21.0	10000.0	165.0	3.0	2.0	46653.0	
374851	0.0	10.0	21.0	650.0	7.0	1.0	2.0	47810.0	
374852	8.0	136.0	17.0	24274.0	4483.0	82.0	2.0	49200.0	

374853 rows × 14 columns

In [45]:

```
x=df.drop(['State','LaunchTime','LaunchDay','DeadlineDay'],axis=1)
```

In [46]:

```
x.columns
```

Out[46]:

```
Index(['Category', 'Subcategory', 'Country', 'Goal', 'Pledged', 'Backers',
       'DeadlineYear', 'LaunchYear', 'Deadlinemonth', 'LaunchMonth'],
      dtype='object')
```

In [47]:

```
y=df['State']
```

SEGREGATING TRAIN TEST DATA

In [48]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
```

IMPORTING ALL CLASSIFIER

In [49]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
```

CREATING USERDEFIND FUNCTION

In [50]:

```
def mymodel(model):

    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)

    train=model.score(xtrain,ytrain)
    test=model.score(xtest,ytest)

    print(f"Training score:{train}\nTesting score:{test}")
    print(classification_report(ytest,ypred))

    return model
```

In [51]:

```
#KNeighborsClassifier
knn=mymodel(KNeighborsClassifier())
```

```
Training score:0.8821442318319188
Testing score:0.8549566052500533
              precision    recall  f1-score   support

         0.0       0.19      0.05      0.08     11582
         1.0       0.82      0.94      0.88     59365
         2.0       0.67      0.04      0.07       847
         3.0       0.96      0.99      0.97     40097
         4.0       0.00      0.00      0.00       565

    accuracy                           0.85    112456
   macro avg       0.53      0.40      0.40    112456
weighted avg       0.80      0.85      0.82    112456
```

In [52]:

```
#LogisticRegression
lr=mymodel(LogisticRegression())
```

```
Training score:0.8031646703277857
Testing score:0.8029985060823789
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00     11582
         1.0       0.75      0.96      0.84     59365
         2.0       0.00      0.00      0.00       847
         3.0       0.91      0.84      0.87     40097
         4.0       0.00      0.00      0.00       565

    accuracy                           0.80    112456
   macro avg       0.33      0.36      0.34    112456
weighted avg       0.72      0.80      0.75    112456
```

In [53]:

```
#DecisionTreeClassifier
dtc=mymodel(DecisionTreeClassifier())
```

```
Training score:0.9918444189529606
Testing score:0.8016557586967348
              precision    recall  f1-score   support

         0.0       0.20      0.23      0.21     11582
         1.0       0.83      0.81      0.82     59365
         2.0       0.87      0.85      0.86       847
         3.0       0.97      0.96      0.96     40097
         4.0       0.03      0.03      0.03       565

    accuracy                           0.80    112456
   macro avg       0.58      0.58      0.58    112456
weighted avg       0.81      0.80      0.81    112456
```

In [54]:

```
#RandomForestClassifier
rfr=mymodel(RandomForestClassifier())
```

```
Training score:0.991832985895418
Testing score:0.871576438784947
              precision    recall  f1-score   support

         0.0       0.29      0.05      0.08     11582
         1.0       0.84      0.96      0.89     59365
         2.0       0.86      0.99      0.92       847
         3.0       0.96      0.99      0.97     40097
         4.0       0.22      0.01      0.02       565

    accuracy                           0.87    112456
   macro avg       0.63      0.60      0.58    112456
weighted avg       0.82      0.87      0.83    112456
```

In [55]:

```
#AdaBoostClassifier
adc=mymodel(AdaBoostClassifier())
```

```
Training score:0.8417855387066163
Testing score:0.8425517535747314
              precision    recall  f1-score   support

         0.0       0.24      0.01      0.02     11582
         1.0       0.81      0.95      0.87     59365
         2.0       0.87      0.92      0.89       847
         3.0       0.91      0.93      0.92     40097
         4.0       0.00      0.00      0.00       565

    accuracy                           0.84    112456
   macro avg       0.56      0.56      0.54    112456
weighted avg       0.78      0.84      0.80    112456
```

In [56]:

```
#XGBClassifier
xgb=mymodel(XGBClassifier())
```

```
Training score:0.884716669778999
Testing score:0.881153517820303
              precision    recall  f1-score   support

         0.0       0.51      0.02      0.04     11582
         1.0       0.84      0.98      0.90     59365
         2.0       0.87      0.98      0.92       847
         3.0       0.96      0.99      0.98     40097
         4.0       0.53      0.02      0.03       565

    accuracy                           0.88    112456
   macro avg       0.74      0.60      0.57    112456
weighted avg       0.85      0.88      0.84    112456
```

IMPORTING PICKLE

In [59]:

```
import pickle
pickle.dump(knn,open('model.pkl','wb'))
```

In [ ]:

In [ ]: