Import Libaries

In [1]:

```python
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
```

Read And Check The Data

In [2]:

```python
df = pd.read_csv('creditcard.csv')
df.head()
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.3
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.6
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.1
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.1

5 rows × 31 columns

Look Into More Details To The Data

In [3]:

```python
df.describe()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	
mean	94813.859575	1.165980e-15	3.416908e-16	-1.373150e-15	2.086869e-15	9.604066e-16	1.490107e-15	-5.556467e-16	1.177556e-16	
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	

8 rows × 31 columns

In [4]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [5]:

```
print('number of rows:', df.shape[0], ', number of columns:', df.shape[1])
```

```
number of rows: 284807 , number of columns: 31
```

Check Missing Data

In [6]:

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

Out[6]:

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [7]:

```
print('Frauds :',(len(df[df['Class'] == 1])/df.shape[0])*100,'%')
print('Non Frauds :',(len(df[df['Class'] == 0])/df.shape[0])*100,'%')
```

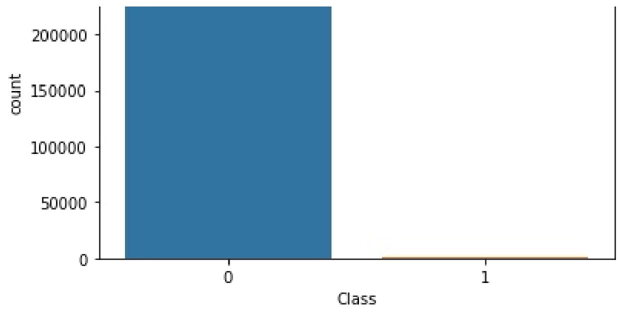
```
Frauds : 0.1727485630620034 %
Non Frauds : 99.82725143693798 %
```

In [8]:

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

Out[8]:

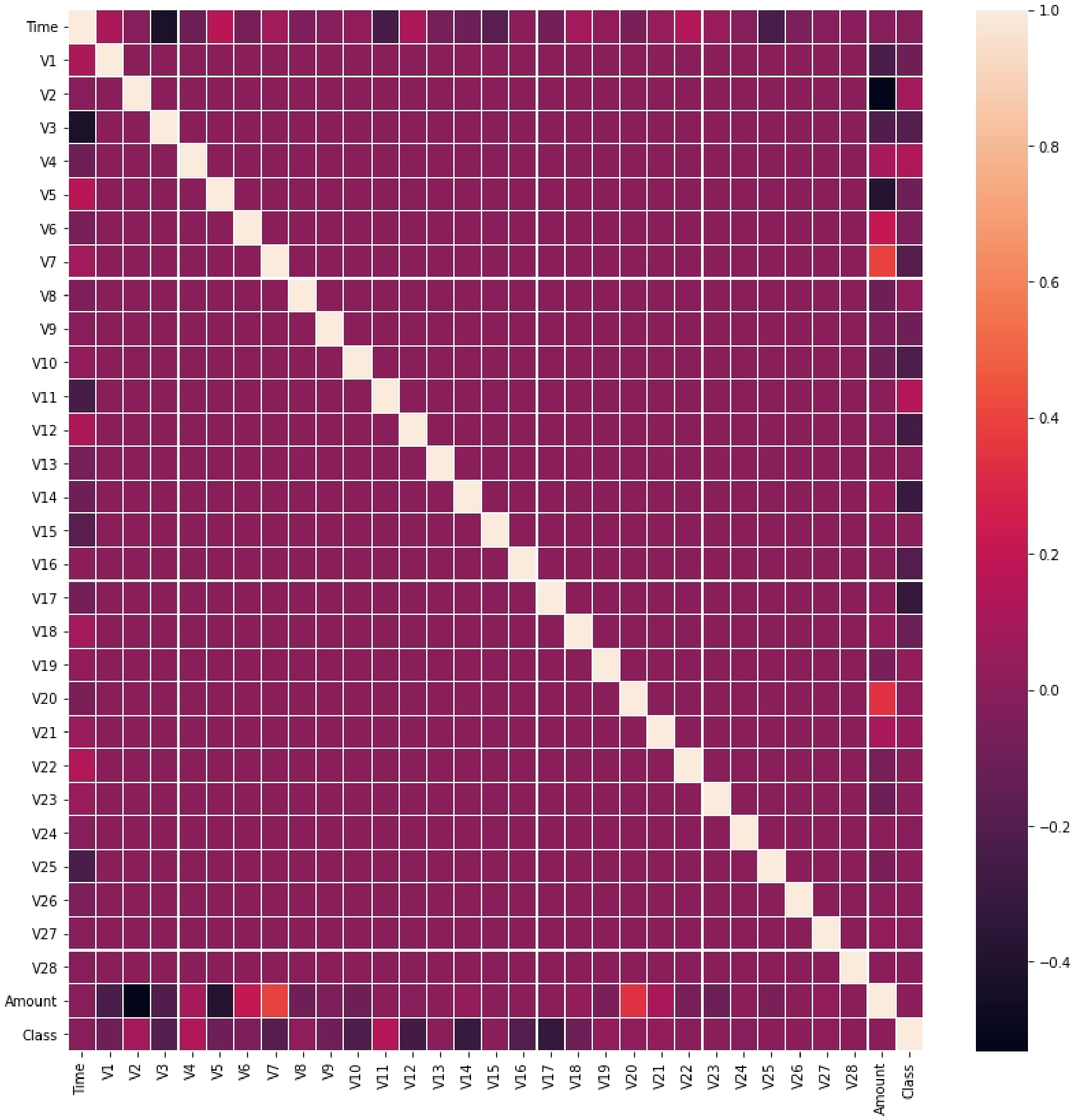
```
<AxesSubplot:xlabel='Class', ylabel='count'>
```



Features Correlation

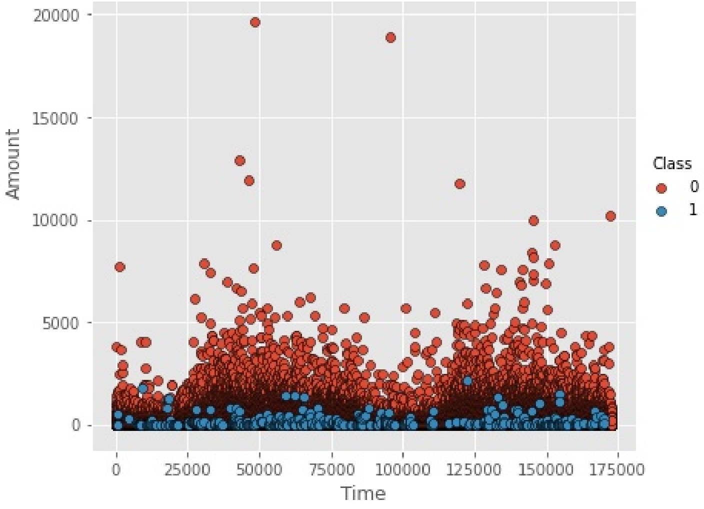
In [9]:

```
plt.figure(figsize = (14,14))
sns.heatmap(df.corr(),xticklabels=df.corr().columns,yticklabels=df.corr().columns,linewidths=.1)
plt.show()
```



In [10]:

```
plt.style.use("ggplot")
sns.FacetGrid(df, hue="Class", height = 6).map(plt.scatter, "Time", "Amount", edgecolor="k").add_legend()
plt.show()
```



Standardize The Variables

```
In [11]: scaler = StandardScaler()
         scalered_features = scaler.fit_transform(df.drop('Class', axis=1))
         scalered_features
```

```
Out[11]: array([[-1.99658302, -0.69424232, -0.04407492, ...,  0.33089162,
                 -0.06378115,  0.24496426],
                [-1.99658302,  0.60849633,  0.16117592, ..., -0.02225568,
                  0.04460752, -0.34247454],
                [-1.99656197, -0.69350046, -0.81157783, ..., -0.13713686,
                 -0.18102083,  1.16068593],
                ...,
                [ 1.6419735 ,  0.98002374, -0.18243372, ...,  0.01103672,
                 -0.0804672 , -0.0818393 ],
                [ 1.6419735 , -0.12275539,  0.32125034, ...,  0.26960398,
                  0.31668678, -0.31324853],
                [ 1.64205773, -0.27233093, -0.11489898, ..., -0.00598394,
                  0.04134999,  0.51435531]])
```

```
In [12]: df_normalize = pd.DataFrame(scalered_features, columns = df.columns[:-1])
         df_normalize
```

Out[12]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	
0	-1.996583	-0.694242	-0.044075	1.672773	0.973366	-0.245117	0.347068	0.193679	0.082637	0.331128	...	0.326118	-0.024923	0.3
1	-1.996583	0.608496	0.161176	0.109797	0.316523	0.043483	-0.061820	-0.063700	0.071253	-0.232494	...	-0.089611	-0.307377	-0.8
2	-1.996562	-0.693500	-0.811578	1.169468	0.268231	-0.364572	1.351454	0.639776	0.207373	-1.378675	...	0.680975	0.337632	1.0
3	-1.996562	-0.493325	-0.112169	1.182516	-0.609727	-0.007469	0.936150	0.192071	0.316018	-1.262503	...	-0.269855	-0.147443	0.0
4	-1.996541	-0.591330	0.531541	1.021412	0.284655	-0.295015	0.071999	0.479302	-0.226510	0.744326	...	0.529939	-0.012839	1.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	1.641931	-6.065842	6.099286	-6.486245	-1.459641	-3.886611	-1.956690	-3.975628	6.116573	1.742559	...	1.914365	0.290602	0.1
284803	1.641952	-0.374121	-0.033356	1.342145	-0.521651	0.629040	0.794446	0.019667	0.246886	0.532299	...	0.077330	0.291625	1.2
284804	1.641974	0.980024	-0.182434	-2.143205	-0.393984	1.905833	2.275262	-0.239939	0.593140	0.393630	...	0.001811	0.315913	0.7
284805	1.641974	-0.122755	0.321250	0.463320	0.487192	-0.273836	0.468155	-0.554672	0.568631	0.356887	...	0.165300	0.361112	1.1
284806	1.642058	-0.272331	-0.114899	0.463866	-0.357570	-0.009089	-0.487602	1.274769	-0.347176	0.442532	...	0.496739	0.355411	0.8

284807 rows × 30 columns

Split Data In Train And Test Set

```
In [58]: X_train,X_test,y_train,y_test = train_test_split(scalered_features, df['Class'], test_size=0.20)
```

KNN

In [59]:

```
knn_model = KNeighborsClassifier(n_neighbors=1)
knn_model.fit(X_train, y_train)
```

Out[59]:

```
KNeighborsClassifier(n_neighbors=1)
```

In [60]:

```
train_preds = knn_model.predict(X_train)
acc = accuracy_score(y_train, train_preds)
print('train accuracy for k = 1 : ',acc)
test_preds = knn_model.predict(X_test)
acc = accuracy_score(y_test, test_preds)
print('test accuracy for k = 1 : ',acc)
```

```
train accuracy for k = 1 :  1.0
test accuracy for k = 1 :  0.9993153330290369
```

In [61]:

```
print(confusion_matrix(y_test, test_preds))
```

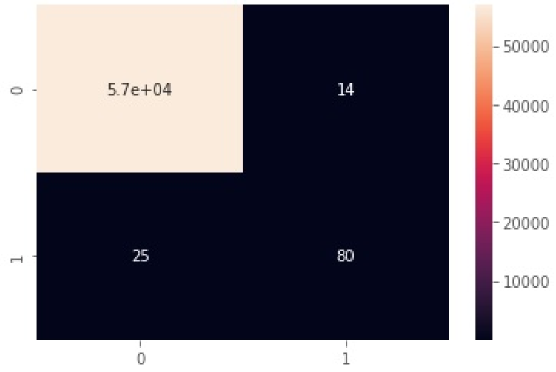
```
[[56843    14]
 [   25    80]]
```

In [62]:

```
sns.heatmap(confusion_matrix(y_test, test_preds), annot = True)
```

Out[62]:

```
<AxesSubplot:>
```



In [63]:

```
print(classification_report(y_test, test_preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56857
           1       0.85      0.76      0.80       105

    accuracy                           1.00     56962
   macro avg       0.93      0.88      0.90     56962
weighted avg       1.00      1.00      1.00     56962
```

In [14]:

```
knn_model = KNeighborsClassifier(n_neighbors=2)
knn_model.fit(X_train, y_train)
```

Out[14]:

```
KNeighborsClassifier(n_neighbors=2)
```

In [15]:

```
train_preds = knn_model.predict(X_train)
acc = accuracy_score(y_train, train_preds)
print('train accuracy for k = 2 : ',acc)
test_preds = knn_model.predict(X_test)
acc = accuracy_score(y_test, test_preds)
print('test accuracy for k = 2 : ',acc)
```

```
train accuracy for k = 2 :  0.9996664399043209
test accuracy for k = 2 :  0.9994908886626171
```

In [16]:

```
print(confusion_matrix(y_test, test_preds))
```

```
[[56861     1]
```

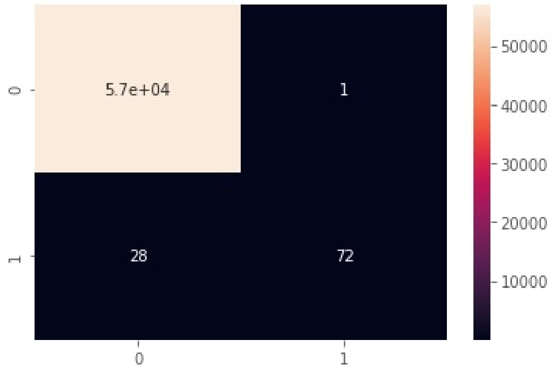
```
 [   28    72]]
```

In [17]:

```
sns.heatmap(confusion_matrix(y_test, test_preds), annot = True)
```

Out[17]:

```
<AxesSubplot:>
```



In [18]:

```
print(classification_report(y_test, test_preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56862
           1       0.99      0.72      0.83       100

    accuracy                           1.00     56962
   macro avg       0.99      0.86      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

In [19]:

```
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train, y_train)
```

Out[19]:

```
KNeighborsClassifier(n_neighbors=3)
```

In [20]:

```
train_preds = knn_model.predict(X_train)
acc = accuracy_score(y_train, train_preds)
print('train accuracy for k = 3 : ',acc)
test_preds = knn_model.predict(X_test)
acc = accuracy_score(y_test, test_preds)
print('test accuracy for k = 3 : ',acc)
```

```
train accuracy for k = 3 :  0.9996752178015756
test accuracy for k = 3 :  0.9994733330992591
```

In [21]:

```
print(confusion_matrix(y_test, test_preds))
```

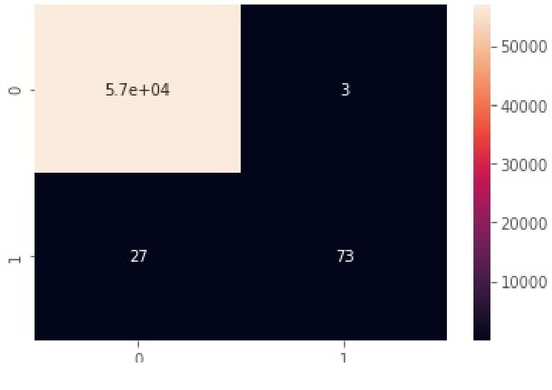
```
[[56859     3]
 [   27    73]]
```

In [22]:

```
sns.heatmap(confusion_matrix(y_test, test_preds), annot = True)
```

Out[22]:

```
<AxesSubplot:>
```



In [23]:

```
print(classification_report(y_test, test_preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56862
           1       0.96      0.73      0.83       100

    accuracy                           1.00     56962
   macro avg       0.98      0.86      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```

In [24]:

```
knn_model = KNeighborsClassifier(n_neighbors=4)
knn_model.fit(X_train, y_train)
```

Out[24]:

```
KNeighborsClassifier(n_neighbors=4)
```

In [25]:

```
train_preds = knn_model.predict(X_train)
acc = accuracy_score(y_train, train_preds)
print('train accuracy for k = 4 : ',acc)
test_preds = knn_model.predict(X_test)
acc = accuracy_score(y_test, test_preds)
print('test accuracy for k = 4 : ',acc)
```

```
train accuracy for k = 4 :  0.9996225504180474
test accuracy for k = 4 :  0.9994908886626171
```

In [26]:

```
print(confusion_matrix(y_test, test_preds))
```

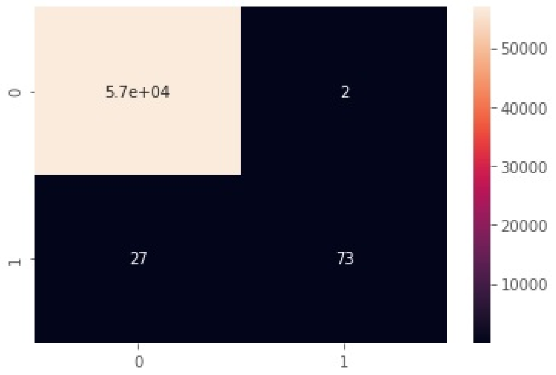
```
[[56860     2]
 [   27    73]]
```

In [27]:

```
sns.heatmap(confusion_matrix(y_test, test_preds), annot = True)
```

Out[27]:

```
<AxesSubplot:>
```



In [28]:

```
print(classification_report(y_test, test_preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56862
           1       0.97      0.73      0.83       100

    accuracy                           1.00     56962
   macro avg       0.99      0.86      0.92     56962
weighted avg       1.00      1.00      1.00     56962
```

In [29]:

```
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
```

Out[29]:

```
KNeighborsClassifier()
```

In [30]:

```
train_preds = knn_model.predict(X_train)
acc = accuracy_score(y_train, train_preds)
print('train accuracy for k = 5 : ',acc)
test_preds = knn_model.predict(X_test)
acc = accuracy_score(y_test, test_preds)
print('test accuracy for k = 5 : ',acc)
```

```
train accuracy for k = 5 :  0.99961816146942
test accuracy for k = 5 :  0.9994733330992591
```

In [31]:

```
print(confusion_matrix(y_test, test_preds))
```

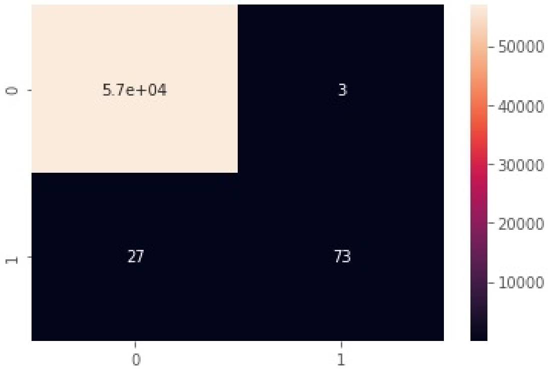
```
[[56859     3]
 [   27    73]]
```

In [32]:

```
sns.heatmap(confusion_matrix(y_test, test_preds), annot = True)
```

Out[32]:

```
<AxesSubplot:>
```



In [33]:

```
print(classification_report(y_test, test_preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56862
           1       0.96      0.73      0.83       100

    accuracy                           1.00     56962
   macro avg       0.98      0.86      0.91     56962
weighted avg       1.00      1.00      1.00     56962
```