Task 2

In task 2, we are going to use ANN to predict whether the transaction is a fraud or not.

Loading required libraries. tensorflow and kera packages are for ANN models.

In [29]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# sklearn packages
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout

# plotting
import seaborn as sns

# will show plots without doing plt.show()
%matplotlib inline
```

Loading creditcard data.

In [2]:
```
# Read dataset to pandas dataframe
card = pd.read_csv('creditcard.csv')
```

Credit card data set has 30 columns. columns V1 to V28 are called Principal Component Analysis varicables, or PCA, which is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

In [3]:
```
card.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 [4]:
```
card.head()
```

Out[4]:

	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
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
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
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
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

5 rows × 31 columns

In [5]:
```
card.describe(include='all')
```

Out[5]:

	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	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-15	-1.698953e-15	-1.893285e-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

The target has two values 0 and 1 where 0 is the transaction is clean and 1 is a fraud

In [6]:
```
card.Class.unique()
```

Out[6]:
```
array([0, 1], dtype=int64)
```

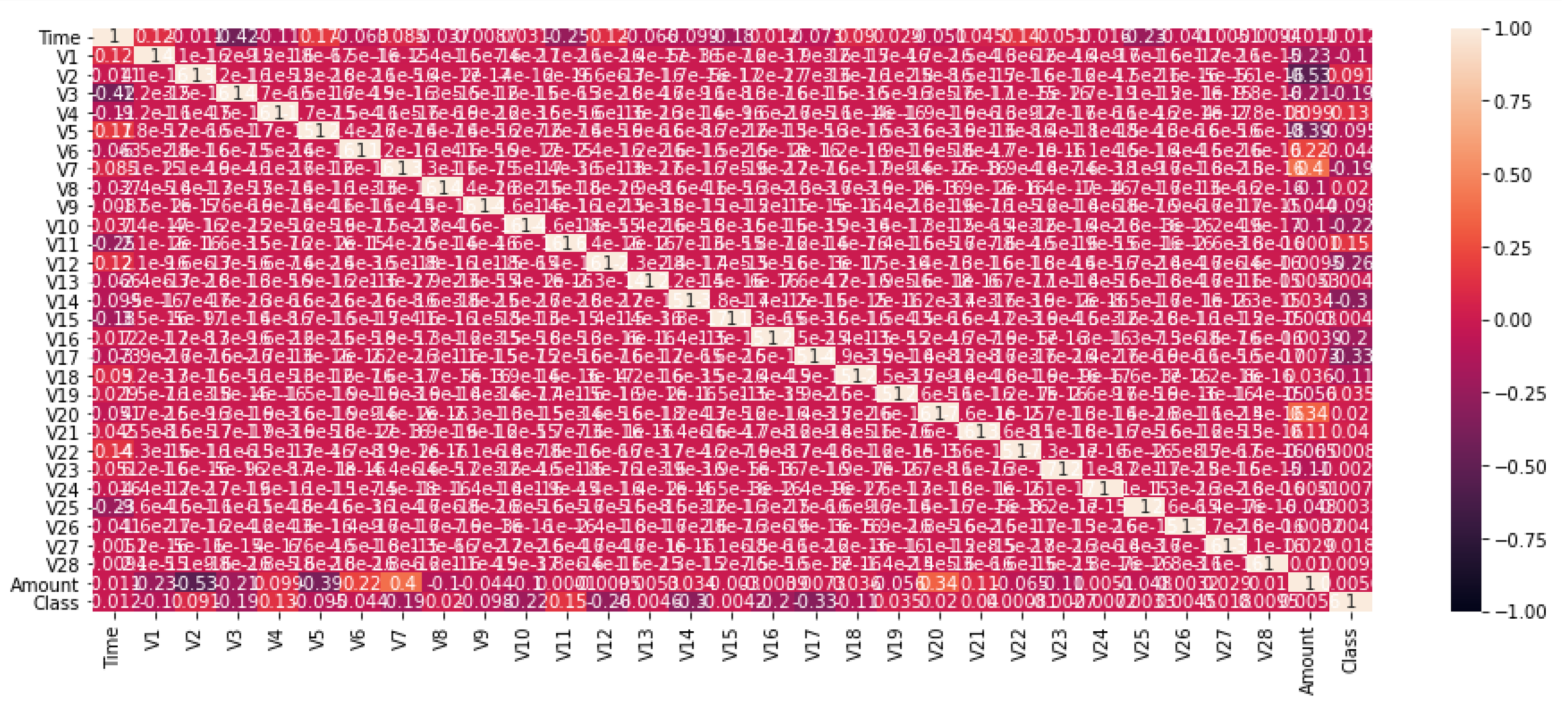
Class column has 284315 0s and only 492 1s which makes the data highly unbalanced.

In [7]:
```
card['Class'].value_counts()
```

Out[7]:
```
0    284315
1       492
Name: Class, dtype: int64
```

Correleation plot below suggests that variables are not correlated except for few columns

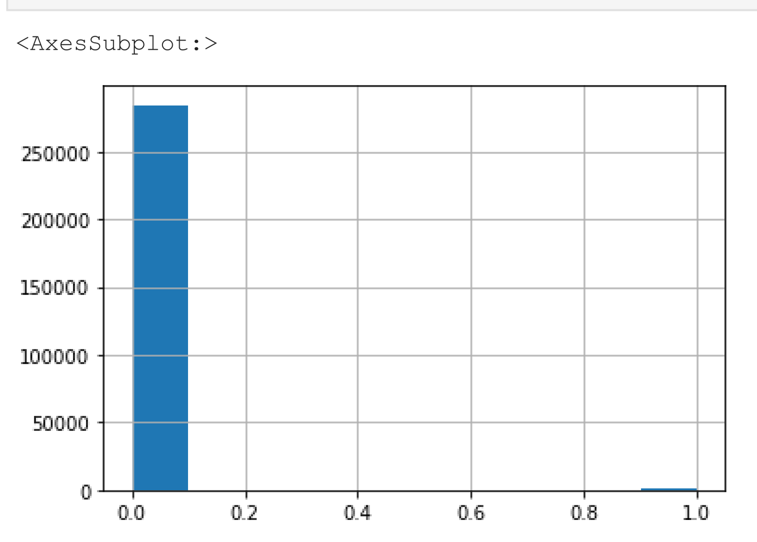
In [8]:
```
plt.figure(figsize=(16, 6))
_ = sns.heatmap(card.corr(), vmin=-1, vmax=1, annot=True)
```



Bar chart below shows how the target is unbalanced. You can barely see 1s bar.

In [9]:
```
card['Class'].hist()
```

Out[9]:
```
<AxesSubplot:>
```



Because of all PCA variables are normalized, we need to normalize Amount column.

In [12]:
```
scaler = StandardScaler()
card['NormalizedAmount'] = scaler.fit_transform(card['Amount'].values.reshape(-1, 1))
```

In [13]:
```
card.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 32 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  
 31  NormalizedAmount  284807 non-null  float64
dtypes: float64(31), int64(1)
memory usage: 69.5 MB
```

Dropping Amount and Time because we don't need time to predict fraud and the amount column we already normalized it in a new column

In [14]:
```
card = card.drop(['Amount', 'Time'], axis = 1)
```

Splitting the data to train and test subset with 0.8 ratio.

In [15]:
```
#gather up names of all the columns
cols = card.columns

#set the prediction column and the feature columns for KNN
prediction_col = 'Class'
feature_cols = [c for c in cols if c != prediction_col]

x = card[feature_cols]
y = card[prediction_col]

#split the dataset into the train and test data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=41)
```

Here we fit the 5-layer model, where input_dim means the number of input columns, unit is the number of nodes in each layer. and using ReLU as activation function for the hidden layers and use Sigmoid function in the output layer for a binary classification problem.

In [30]:
```
model = Sequential([
Dense(input_dim = 29, units = 16, activation = 'relu'),
Dense(units = 24, activation = 'relu'),
Dropout(0.5),
Dense(units = 20, activation = 'relu'),
Dense(units = 24, activation = 'relu'),
Dense(units =1, activation = 'sigmoid'),])
```

In [57]:
```
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
model.fit(x_train, y_train, batch_size = 15, epochs = 5)
```

```
Epoch 1/5
15190/15190 [==============================] - 17s 1ms/step - loss: 0.0034 - accuracy: 0.9994
Epoch 2/5
15190/15190 [==============================] - 16s 1ms/step - loss: 0.0032 - accuracy: 0.9994
Epoch 3/5
15190/15190 [==============================] - 16s 1ms/step - loss: 0.0032 - accuracy: 0.9994
Epoch 4/5
15190/15190 [==============================] - 17s 1ms/step - loss: 0.0031 - accuracy: 0.9994
Epoch 5/5
15190/15190 [==============================] - 17s 1ms/step - loss: 0.0030 - accuracy: 0.9994
```

Out[57]:
```
<keras.callbacks.History at 0x2787eeabd30>
```

The model accuracy is 99% which is impressive. the lose is 0.3%

In [58]:
```
predictions = model.predict(x_test)
```

Printing the predictions.

In [59]:
```
print(predictions)
```

```
[[2.71920264e-09]
 [1.93046674e-07]
 [2.59850026e-07]
 ...
 [1.15511135e-07]
 [1.77368929e-06]
 [3.84221903e-06]]
```

In order to print confusion matrix we need to make predictions binary. So, we need to change y_test and predictions from continuous to classes.

In [63]:
```
cutoff = 0.7                                   # decide on a cutoff limit
y_pred_classes = np.zeros_like(predictions)    # initialise a matrix full with zeros
y_pred_classes[predictions > cutoff] = 1
```

In [65]:
```
y_test_classes = np.zeros_like(predictions)
y_test_classes[predictions > cutoff] = 1
```

In [67]:
```
print(confusion_matrix(y_test_classes,y_pred_classes))
print(classification_report(y_test_classes,y_pred_classes))
```

```
[[56891     0]
 [    0    71]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     56891
         1.0       1.00      1.00      1.00        71

    accuracy                           1.00     56962
   macro avg       1.00      1.00      1.00     56962
weighted avg       1.00      1.00      1.00     56962
```

True positive is 56891 and false negative is 71. Precision 1.0, recall 1.0, F1 score 1.0.

Some of code are adopted from references

References

https://www.geeksforgeeks.org/python-pandas-split-strings-into-two-list-columns-using-str-split/

https://stackoverflow.com/questions/39173813/pandas-convert-dtype-object-to-int

https://towardsdatascience.com/credit-card-fraud-detection-9bc8db79b956

https://stackoverflow.com/questions/37179332/error-in-keras-name-dense-is-not-defined

https://datascience.stackexchange.com/questions/46019/continuous-variable-not-supported-in-confusion-matrix