Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Upload dataset

In [2]:
```
data = pd.read_csv('C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\Data Science
\\DL & NLP\\Machine Learning\\Machine Learning\\Credit_Card_Default_Classifica
tion\\creditcard.csv')
```

Analysis the data set

In [3]:
```
# View the shape of dataset
data.shape
```

Out[3]:
```
(284807, 31)
```

In [4]:
```
# View the type of data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
Time      284807 non-null float64
V1        284807 non-null float64
V2        284807 non-null float64
V3        284807 non-null float64
V4        284807 non-null float64
V5        284807 non-null float64
V6        284807 non-null float64
V7        284807 non-null float64
V8        284807 non-null float64
V9        284807 non-null float64
V10       284807 non-null float64
V11       284807 non-null float64
V12       284807 non-null float64
V13       284807 non-null float64
V14       284807 non-null float64
V15       284807 non-null float64
V16       284807 non-null float64
V17       284807 non-null float64
V18       284807 non-null float64
V19       284807 non-null float64
V20       284807 non-null float64
V21       284807 non-null float64
V22       284807 non-null float64
V23       284807 non-null float64
V24       284807 non-null float64
V25       284807 non-null float64
V26       284807 non-null float64
V27       284807 non-null float64
V28       284807 non-null float64
Amount    284807 non-null float64
Class     284807 non-null int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [5]:
```
# View the column names
data.columns
```

Out[5]:
```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [6]:
```
# View the first 10 values
data.head(10)
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-

10 rows × 31 columns

In [7]:
```
# View the last 10 values
data.tail(10)
```

Out[7]:

	Time	V1	V2	V3	V4	V5	V6	V7	
284797	172782.0	-0.241923	0.712247	0.399806	-0.463406	0.244531	-1.343668	0.929369	-(
284798	172782.0	0.219529	0.881246	-0.635891	0.960928	-0.152971	-1.014307	0.427126	(
284799	172783.0	-1.775135	-0.004235	1.189786	0.331096	1.196063	5.519980	-1.518185	2
284800	172784.0	2.039560	-0.175233	-1.196825	0.234580	-0.008713	-0.726571	0.017050	-(
284801	172785.0	0.120316	0.931005	-0.546012	-0.745097	1.130314	-0.235973	0.812722	(
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	(
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	(
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	(
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-(

10 rows × 31 columns

In [8]:
```
# Check if any null values in the data
data.isnull().any()
```

Out[8]:
```
Time      False
V1        False
V2        False
V3        False
V4        False
V5        False
V6        False
V7        False
V8        False
V9        False
V10       False
V11       False
V12       False
V13       False
V14       False
V15       False
V16       False
V17       False
V18       False
V19       False
V20       False
V21       False
V22       False
V23       False
V24       False
V25       False
V26       False
V27       False
V28       False
Amount    False
Class     False
dtype: bool
```

In [9]:
```
data.isnull().sum()
```

Out[9]:
```
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
```

No null values in the given dataset

Exploratory Data Analysis(EDA)

In [10]:

```
data.describe().transpose()
```

Out[10]:

	count	mean	std	min	25%	50%	
Time	284807.0	9.481386e+04	47488.145955	0.000000	54201.500000	84692.000000	139320.
V1	284807.0	3.919560e-15	1.958696	-56.407510	-0.920373	0.018109	1.
V2	284807.0	5.688174e-16	1.651309	-72.715728	-0.598550	0.065486	0.
V3	284807.0	-8.769071e-15	1.516255	-48.325589	-0.890365	0.179846	1.
V4	284807.0	2.782312e-15	1.415869	-5.683171	-0.848640	-0.019847	0.
V5	284807.0	-1.552563e-15	1.380247	-113.743307	-0.691597	-0.054336	0.
V6	284807.0	2.010663e-15	1.332271	-26.160506	-0.768296	-0.274187	0.
V7	284807.0	-1.694249e-15	1.237094	-43.557242	-0.554076	0.040103	0.
V8	284807.0	-1.927028e-16	1.194353	-73.216718	-0.208630	0.022358	0.
V9	284807.0	-3.137024e-15	1.098632	-13.434066	-0.643098	-0.051429	0.
V10	284807.0	1.768627e-15	1.088850	-24.588262	-0.535426	-0.092917	0.
V11	284807.0	9.170318e-16	1.020713	-4.797473	-0.762494	-0.032757	0.
V12	284807.0	-1.810658e-15	0.999201	-18.683715	-0.405571	0.140033	0.
V13	284807.0	1.693438e-15	0.995274	-5.791881	-0.648539	-0.013568	0.
V14	284807.0	1.479045e-15	0.958596	-19.214325	-0.425574	0.050601	0.
V15	284807.0	3.482336e-15	0.915316	-4.498945	-0.582884	0.048072	0.
V16	284807.0	1.392007e-15	0.876253	-14.129855	-0.468037	0.066413	0.
V17	284807.0	-7.528491e-16	0.849337	-25.162799	-0.483748	-0.065676	0.
V18	284807.0	4.328772e-16	0.838176	-9.498746	-0.498850	-0.003636	0.
V19	284807.0	9.049732e-16	0.814041	-7.213527	-0.456299	0.003735	0.
V20	284807.0	5.085503e-16	0.770925	-54.497720	-0.211721	-0.062481	0.
V21	284807.0	1.537294e-16	0.734524	-34.830382	-0.228395	-0.029450	0.
V22	284807.0	7.959909e-16	0.725702	-10.933144	-0.542350	0.006782	0.
V23	284807.0	5.367590e-16	0.624460	-44.807735	-0.161846	-0.011193	0.
V24	284807.0	4.458112e-15	0.605647	-2.836627	-0.354586	0.040976	0.
V25	284807.0	1.453003e-15	0.521278	-10.295397	-0.317145	0.016594	0.
V26	284807.0	1.699104e-15	0.482227	-2.604551	-0.326984	-0.052139	0.
V27	284807.0	-3.660161e-16	0.403632	-22.565679	-0.070840	0.001342	0.
V28	284807.0	-1.206049e-16	0.330083	-15.430084	-0.052960	0.011244	0.
Amount	284807.0	8.834962e+01	250.120109	0.000000	5.600000	22.000000	77.
Class	284807.0	1.727486e-03	0.041527	0.000000	0.000000	0.000000	0.

In [11]:
```
# Find how many 0,1 in Out put data (Class)
data['Class'].value_counts()
```

Out[11]:
```
0    284315
1       492
Name: Class, dtype: int64
```

For predicting the model ,we have to take 0 as non_Fraud and 1 as Fraud happening.

Feature Scaling

In [12]:
```
from sklearn.preprocessing import StandardScaler
data['normalizedAmount'] = StandardScaler().fit_transform(data['Amount'].value
s.reshape(-1,1))
data = data.drop(['Amount'],axis=1)
```

In [13]:
```
data = data.drop(['Time'],axis=1)
data.head()
```

Out[13]:

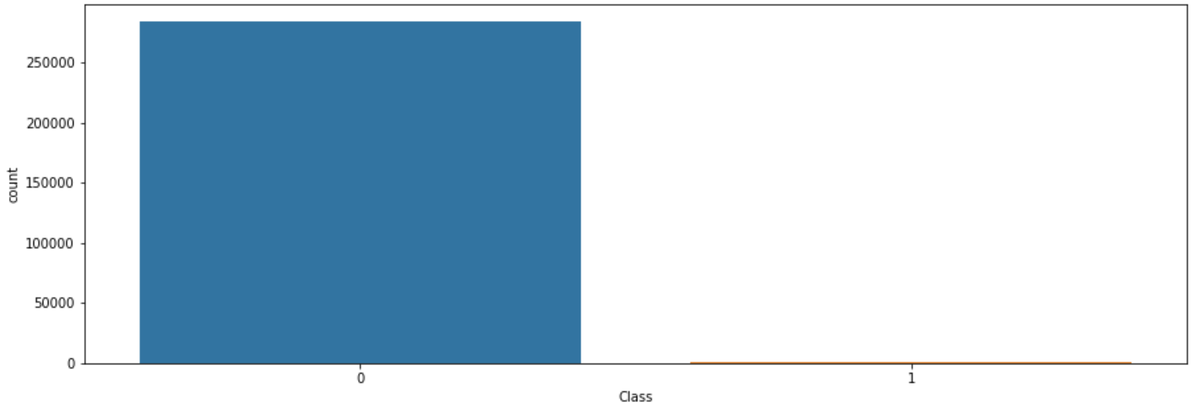
	V1	V2	V3	V4	V5	V6	V7	V8	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36378
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.25542
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.51465
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.38702
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.81773

5 rows × 30 columns

Data Visualizations

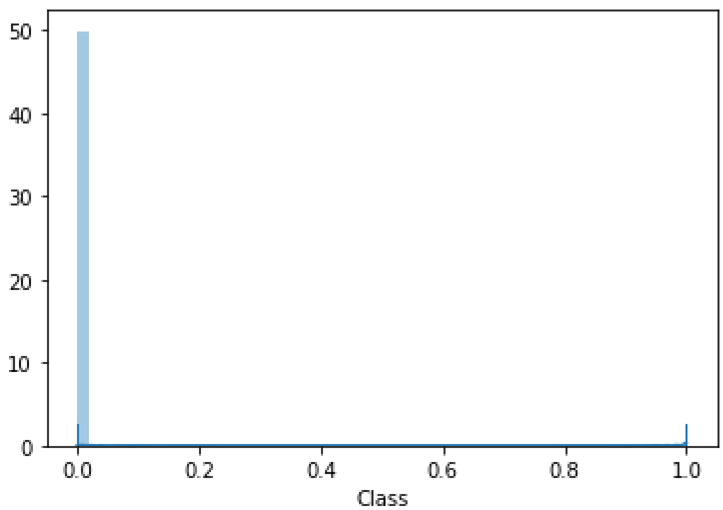
1. Univariate Analysis

In [14]:
```
fig,ax = plt.subplots(figsize=(15,5))
ax = sns.countplot(data['Class'])
plt.show()
```



Distribution of the Class

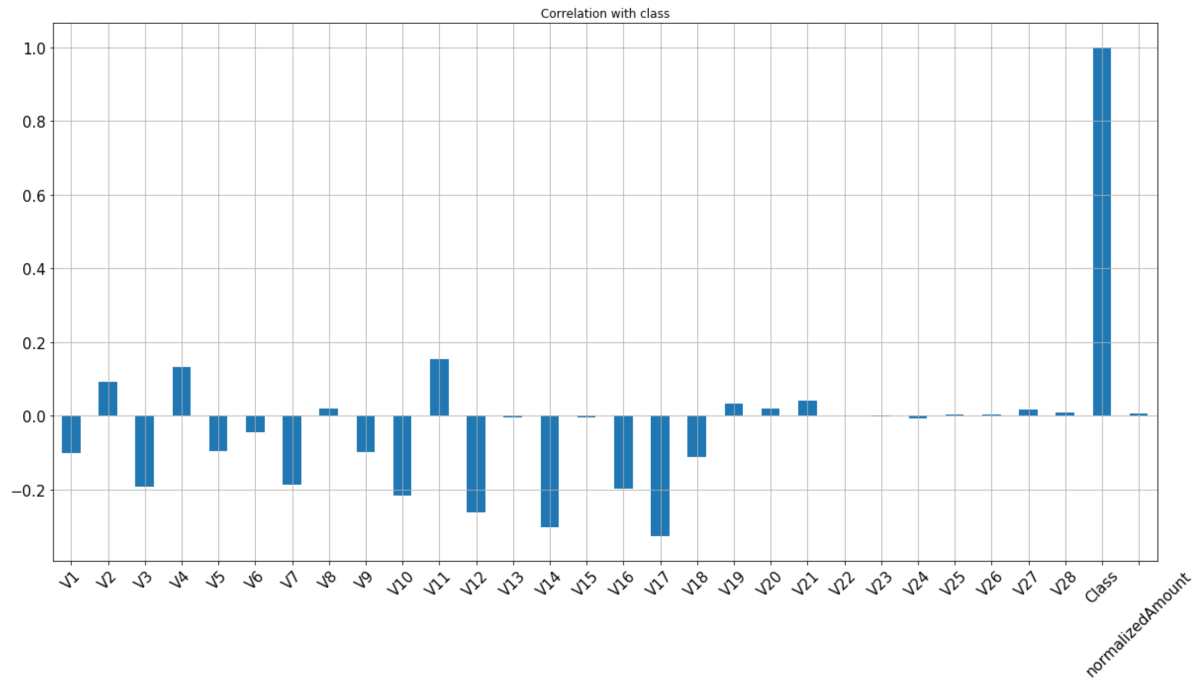
In [15]:
```
# sns.distplot(data['Class'])
sns.distplot(data['Class'],rug=True)
plt.show()
```



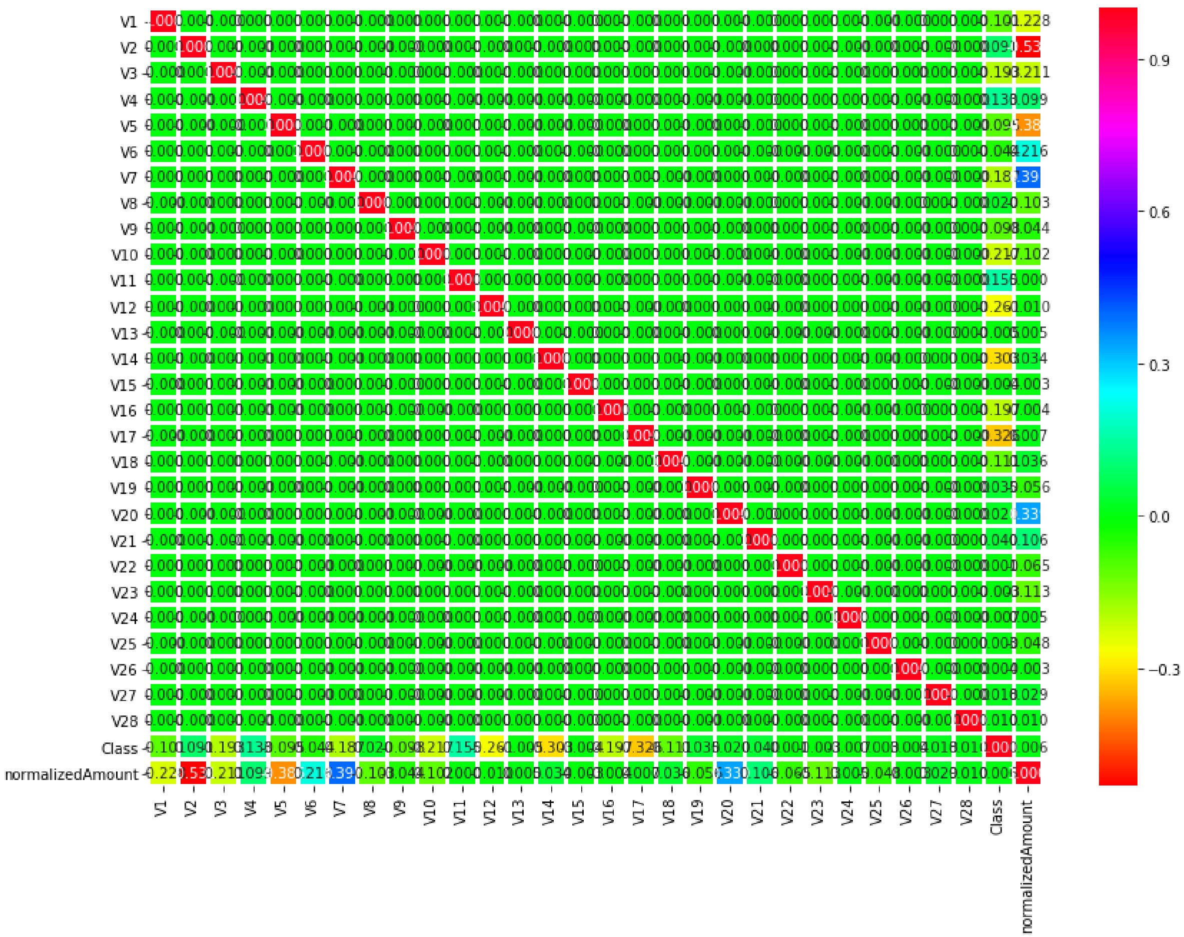
Correlation

In [16]:
```
data.corrwith(data.Class).plot.bar(figsize = (20, 10), title = "Correlation wi
th class", fontsize = 15,rot = 45, grid = True)
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x12f5328b448>



In [17]:
```python
plt.figure(figsize=(14,10))
sns.heatmap(data.corr(),annot=True,cmap='hsv',fmt='.3f',linewidths=2)
plt.show()
```



In [18]:
```python
# Compute the correlation matrix
corr = data.corr()
corr.head()
```

Out[18]:

	V1	V2	V3	V4	V5	V6	
V1	1.000000e+00	4.697350e-17	-1.424390e-15	1.755316e-17	6.391162e-17	2.398071e-16	1.99155
V2	4.697350e-17	1.000000e+00	2.512175e-16	-1.126388e-16	-2.039868e-16	5.024680e-16	3.96648
V3	-1.424390e-15	2.512175e-16	1.000000e+00	-3.416910e-16	-1.436514e-15	1.431581e-15	2.16857
V4	1.755316e-17	-1.126388e-16	-3.416910e-16	1.000000e+00	-1.940929e-15	-2.712659e-16	1.55633
V5	6.391162e-17	-2.039868e-16	-1.436514e-15	-1.940929e-15	1.000000e+00	7.926364e-16	-4.20985

5 rows × 30 columns

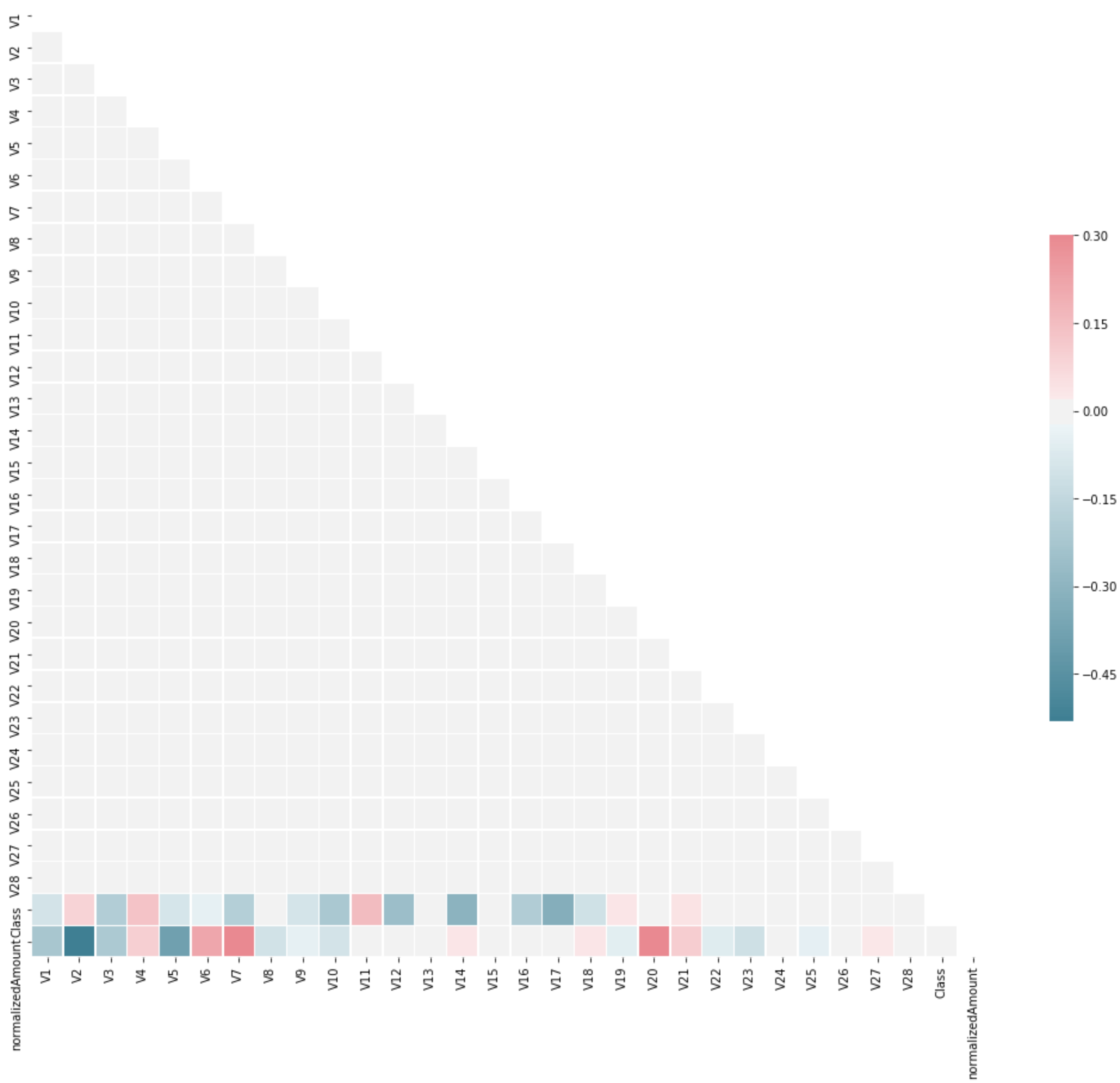
In [19]:
```python
# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(18, 15))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,square=True, linewid
ths=.5, cbar_kws={"shrink": .5})
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x12f67aa0e08>



Normalize the data

In [20]:
```
from sklearn.preprocessing import scale
scale_data=scale(data)
scale_data
```

Out[20]:
```
array([[-0.69424232, -0.04407492,  1.6727735 , ..., -0.06378115,
        -0.04159898,  0.24496426],
       [ 0.60849633,  0.16117592,  0.1097971 , ...,  0.04460752,
        -0.04159898, -0.34247454],
       [-0.69350046, -0.81157783,  1.16946849, ..., -0.18102083,
        -0.04159898,  1.16068593],
       ...,
       [ 0.98002374, -0.18243372, -2.14320514, ..., -0.0804672 ,
        -0.04159898, -0.0818393 ],
       [-0.12275539,  0.32125034,  0.46332013, ...,  0.31668678,
        -0.04159898, -0.31324853],
       [-0.27233093, -0.11489898,  0.46386564, ...,  0.04134999,
        -0.04159898,  0.51435531]])
```

In [21]:
```
np.exp(scale_data)
```

Out[21]:
```
array([[0.49945273, 0.95688226, 5.32692154, ..., 0.9382103 , 0.95925439,
        1.27757566],
       [1.83766607, 1.17489164, 1.1160516 , ..., 1.04561739, 0.95925439,
        0.7100112 ],
       [0.49982339, 0.44415671, 3.22028058, ..., 0.83441798, 0.95925439,
        3.19212208],
       ...,
       [2.66451949, 0.83323987, 0.11727835, ..., 0.92268517, 0.95925439,
        0.92142002],
       [0.88447999, 1.37885072, 1.58934205, ..., 1.37257258, 0.95925439,
        0.7310682 ],
       [0.76160218, 0.89145619, 1.5902093 , ..., 1.0422168 , 0.95925439,
        1.67255987]])
```

Divide the dataset into input and output

In [22]:
```
X = data.iloc[:,data.columns!= 'Class']
Y = data.iloc[:,data.columns== 'Class']
```

```
In [23]: # View the input data
X.head()
```

Out[23]:

	V1	V2	V3	V4	V5	V6	V7	V8	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.36378
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.25542
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.51465
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.38702
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.81773

5 rows × 29 columns

```
In [24]: # View the out put Data
Y.head()
```

Out[24]:

	Class
0	0
1	0
2	0
3	0
4	0

Split the data as train and test

```
In [25]: from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,Y,train_size=0.80,random_st
ate=0)
```

Build the Models

1. Decision Tree Classifier

```
In [26]: from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(random_state = 0,criterion = 'gini',  spli
tter='best', min_samples_leaf=1, min_samples_split=2)
```

In [27]:
```
# Fit the model
classifier.fit(x_train,y_train)
```

Out[27]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [28]:
```
# Predict the model
DT_pred = classifier.predict(x_test)
DT_pred
```

Out[28]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [29]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix,accuracy_score,f1_score,precision
_score,recall_score
print(confusion_matrix(DT_pred,y_test))
```

```
[[56839    26]
 [   22    75]]
```

In [30]:
```
# Accuracy Score
DT_acc = accuracy_score(DT_pred,y_test)
DT_acc
```

Out[30]:
```
0.9991573329588147
```

In [31]:
```
# Precision ,Recall,F1_score
DT_Prec = precision_score(DT_pred,y_test)
DT_rec = recall_score(DT_pred,y_test)
DT_f1 = f1_score(DT_pred,y_test)
```

In [32]:
```
# Store the results
results = pd.DataFrame([['Decision tree', DT_acc, DT_Prec, DT_rec, DT_f1]],col
umns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
```

In [33]:
```
# view the results
results
```

Out[33]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Decision tree	0.999157	0.742574	0.773196	0.757576

2. Random Forest Classifier

In [34]:
```
from sklearn.ensemble import RandomForestClassifier
classifier1 = RandomForestClassifier(random_state = 0, n_estimators = 100,crit
erion = 'entropy')
classifier1
```

Out[34]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entrop
y',
                       max_depth=None, max_features='auto', max_leaf_nodes=No
ne,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=
0,
                       warm_start=False)
```

In [35]:
```
# Fit the model
classifier1.fit(x_train,y_train)
```

```
C:\Users\sundara.rao.ext\Anaconda\lib\site-packages\ipykernel_launcher.py:2:
DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples,), for example using ravel().
```

Out[35]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entrop
y',
                       max_depth=None, max_features='auto', max_leaf_nodes=No
ne,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=
0,
                       warm_start=False)
```

In [36]:
```
# predict the model
RF_pred = classifier1.predict(x_test)
RF_pred
```

Out[36]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [37]:
```
# confusion_matrix
print(confusion_matrix(RF_pred,y_test))
```

```
[[56855    22]
 [    6    79]]
```

In [38]:
```
# Accuracy , precision,recall,f1_score
RF_acc = accuracy_score(RF_pred,y_test)
RF_prec = precision_score(RF_pred,y_test)
RF_rec = recall_score(RF_pred,y_test)
RF_f1= f1_score(RF_pred,y_test)
```

In [39]:
```
# Store the results
results1 = pd.DataFrame([['Random Forest (n=100)', RF_acc, RF_prec, RF_rec, RF
_f1]],columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
```

In [40]:
```
# view the results
results1
```

Out[40]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Random Forest (n=100)	0.999508	0.782178	0.929412	0.849462

3.Naive Bayes Classifier

In [41]:
```
from sklearn.naive_bayes import GaussianNB
classifier2 = GaussianNB()
```

In [42]:
```
# Fit the model
classifier2.fit(x_train,y_train)
```

```
C:\Users\sundara.rao.ext\Anaconda\lib\site-packages\sklearn\utils\validation.
py:724: DataConversionWarning: A column-vector y was passed when a 1d array w
as expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

Out[42]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [43]:
```
#Predict the model
NBC_pred = classifier2.predict(x_test)
NBC_pred
```

Out[43]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [44]:
```
# Confusion Matrix
print(confusion_matrix(NBC_pred,y_test))
```

```
[[55642    15]
 [ 1219    86]]
```

In [45]:
```
# Accuracy ,Precision,Recall,f1_core
NBC_acc = accuracy_score(NBC_pred,y_test)
NBC_prec = precision_score(NBC_pred,y_test)
NBC_rec = recall_score(NBC_pred,y_test)
NBC_f1= f1_score(NBC_pred,y_test)
```

In [46]:
```
# Store the results
results2 = pd.DataFrame([['Naive Bayes', NBC_acc, NBC_prec, NBC_rec, NBC_f1]],
columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
```

In [47]:
```
# view the result
results2
```

Out[47]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Naive Bayes	0.978336	0.851485	0.0659	0.122333

K-Nearest Neighbor

In [48]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier3 = KNeighborsClassifier(n_neighbors=5)
```

In [49]:
```
# Fit the model
classifier3.fit(x_train,y_train)
```

```
C:\Users\sundara.rao.ext\Anaconda\lib\site-packages\ipykernel_launcher.py:2:
DataConversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using ravel
().
```

Out[49]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [50]:
```
# Predict the model
KNN_pred = classifier3.predict(x_test)
KNN_pred
```

Out[50]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [51]:
```
# Confusion Matrix
print(confusion_matrix(KNN_pred,y_test))
```

```
[[56854    20]
 [    7    81]]
```

In [52]:
```
# Accuracy ,Precision,Recall,f1_core
KNN_acc = accuracy_score(KNN_pred,y_test)
KNN_prec = precision_score(KNN_pred,y_test)
KNN_rec = recall_score(KNN_pred,y_test)
KNN_f1= f1_score(KNN_pred,y_test)
```

In [53]:
```
# Store the results
results3 = pd.DataFrame([['K-Nearest Neighbor', KNN_acc, KNN_prec, KNN_rec, KN
N_f1]],columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
```

In [54]:
```
# view the result
results3
```

Out[54]:

	Model	Accuracy	Precision	Recall	F1 Score
0	K-Nearest Neighbor	0.999526	0.80198	0.920455	0.857143

5. Support Vector Machine (SVM)

In [55]:
```
from sklearn.svm import SVC
classifier4 = SVC(kernel='poly' ,random_state =0)
```

In [56]:
```
# fit the model
classifier4.fit(x_train,y_train)
```

```
C:\Users\sundara.rao.ext\Anaconda\lib\site-packages\sklearn\utils\validation.
py:724: DataConversionWarning: A column-vector y was passed when a 1d array w
as expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
C:\Users\sundara.rao.ext\Anaconda\lib\site-packages\sklearn\svm\base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 'scale'
in version 0.22 to account better for unscaled features. Set gamma explicitly
to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[56]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='poly', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
```

In [57]:
```
# Predict the model
SVM_pred = classifier4.predict(x_test)
```

In [58]:
```
#  Confusion Matrix
print(confusion_matrix(SVM_pred,y_test))
```

```
[[56849    22]
 [   12    79]]
```

In [59]:
```
# Accuracy ,Precision,Recall,f1_core
SVM_acc = accuracy_score(SVM_pred,y_test)
SVM_prec = precision_score(SVM_pred,y_test)
SVM_rec = recall_score(SVM_pred,y_test)
SVM_f1= f1_score(SVM_pred,y_test)
```

In [60]:
```
# Store the results
results4 = pd.DataFrame([['Support Vector Machine', SVM_acc, SVM_prec, SVM_rec
, SVM_f1]],columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
```

In [61]:
```
results4
```

Out[61]:

	Model	Accuracy	Precision	Recall	F1 Score
0	Support Vector Machine	0.999403	0.782178	0.868132	0.822917

Artificial Neural Networks

In [62]:
```
# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense

# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units =15 , kernel_initializer = 'uniform', activation =
'relu', input_dim = 29))

# Adding the second hidden layer
classifier.add(Dense(units = 15, kernel_initializer = 'uniform', activation =
'relu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation =
'sigmoid'))

# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
['accuracy'])
```

```
Using TensorFlow backend.
```

In [64]:
```
# Fitting the ANN to the Training set
classifier.fit(x_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
227845/227845 [==============================] - 14s 63us/step - loss: 0.0132
- accuracy: 0.9986
Epoch 2/100
227845/227845 [==============================] - 14s 61us/step - loss: 0.0030
- accuracy: 0.9994
Epoch 3/100
227845/227845 [==============================] - 14s 63us/step - loss: 0.0029
- accuracy: 0.9994
Epoch 4/100
227845/227845 [==============================] - 14s 63us/step - loss: 0.0027
- accuracy: 0.9994
Epoch 5/100
227845/227845 [==============================] - 14s 63us/step - loss: 0.0027
- accuracy: 0.9994
Epoch 6/100
227845/227845 [==============================] - 17s 74us/step - loss: 0.0026
- accuracy: 0.9994
Epoch 7/100
227845/227845 [==============================] - 15s 64us/step - loss: 0.0027
- accuracy: 0.9994
Epoch 8/100
227845/227845 [==============================] - 13s 58us/step - loss: 0.0025
- accuracy: 0.9994
Epoch 9/100
227845/227845 [==============================] - 19s 84us/step - loss: 0.0025
- accuracy: 0.9994
Epoch 10/100
227845/227845 [==============================] - 18s 77us/step - loss: 0.0025
- accuracy: 0.9995
Epoch 11/100
227845/227845 [==============================] - 16s 70us/step - loss: 0.0025
- accuracy: 0.9994
Epoch 12/100
227845/227845 [==============================] - 16s 69us/step - loss: 0.0023
- accuracy: 0.9995
Epoch 13/100
227845/227845 [==============================] - 16s 71us/step - loss: 0.0024
- accuracy: 0.9995
Epoch 14/100
227845/227845 [==============================] - 16s 71us/step - loss: 0.0023
- accuracy: 0.9995
Epoch 15/100
227845/227845 [==============================] - 17s 73us/step - loss: 0.0023
- accuracy: 0.9995
Epoch 16/100
227845/227845 [==============================] - 19s 83us/step - loss: 0.0022
- accuracy: 0.9995
Epoch 17/100
227845/227845 [==============================] - 17s 75us/step - loss: 0.0023
- accuracy: 0.9995
Epoch 18/100
227845/227845 [==============================] - 16s 72us/step - loss: 0.0023
- accuracy: 0.9995
Epoch 19/100
227845/227845 [==============================] - 22s 95us/step - loss: 0.0022
- accuracy: 0.9995
```

```
Epoch 20/100
227845/227845 [==============================] - 23s 100us/step - loss: 0.002
2 - accuracy: 0.9995
Epoch 21/100
227845/227845 [==============================] - 16s 70us/step - loss: 0.0021
- accuracy: 0.9995
Epoch 22/100
227845/227845 [==============================] - 15s 64us/step - loss: 0.0021
- accuracy: 0.9995
Epoch 23/100
227845/227845 [==============================] - 13s 56us/step - loss: 0.0020
- accuracy: 0.9995
Epoch 24/100
227845/227845 [==============================] - 14s 59us/step - loss: 0.0022
- accuracy: 0.9995
Epoch 25/100
227845/227845 [==============================] - 13s 57us/step - loss: 0.0020
- accuracy: 0.9996
Epoch 26/100
227845/227845 [==============================] - 13s 58us/step - loss: 0.0020
- accuracy: 0.9995
Epoch 27/100
227845/227845 [==============================] - 13s 59us/step - loss: 0.0020
- accuracy: 0.9995
Epoch 28/100
227845/227845 [==============================] - 13s 58us/step - loss: 0.0020
- accuracy: 0.9995
Epoch 29/100
227845/227845 [==============================] - 13s 57us/step - loss: 0.0019
- accuracy: 0.9995
Epoch 30/100
227845/227845 [==============================] - 14s 62us/step - loss: 0.0019
- accuracy: 0.9995
Epoch 38/100
227845/227845 [==============================] - 14s 62us/step - loss: 0.0018
- accuracy: 0.9995
Epoch 39/100
227845/227845 [==============================] - 14s 62us/step - loss: 0.0018
- accuracy: 0.9995
Epoch 40/100
227845/227845 [==============================] - 15s 65us/step - loss: 0.0017
- accuracy: 0.9996
Epoch 41/100
227845/227845 [==============================] - 14s 62us/step - loss: 0.0018
- accuracy: 0.9996
Epoch 42/100
227845/227845 [==============================] - 14s 60us/step - loss: 0.0017
- accuracy: 0.9996
Epoch 43/100
227845/227845 [==============================] - 14s 60us/step - loss: 0.0017
- accuracy: 0.9995
Epoch 44/100
227845/227845 [==============================] - 14s 60us/step - loss: 0.0017
- accuracy: 0.9996
Epoch 45/100
227845/227845 [==============================] - 13s 56us/step - loss: 0.0017
- accuracy: 0.9996
```

```
Epoch 46/100
227845/227845 [==============================] - 13s 56us/step - loss: 0.0018
- accuracy: 0.9995
Epoch 47/100
227845/227845 [==============================] - 13s 57us/step - loss: 0.0016
- accuracy: 0.9996
Epoch 48/100
227845/227845 [==============================] - 13s 58us/step - loss: 0.0016
- accuracy: 0.9995
Epoch 49/100
227845/227845 [==============================] - 13s 57us/step - loss: 0.0017
- accuracy: 0.9996
Epoch 50/100
227845/227845 [==============================] - 13s 59us/step - loss: 0.0016
- accuracy: 0.9996
Epoch 51/100
227845/227845 [==============================] - 16s 70us/step - loss: 0.0016
- accuracy: 0.9996
Epoch 52/100
227845/227845 [==============================] - 18s 80us/step - loss: 0.0016
- accuracy: 0.9996
Epoch 53/100
227845/227845 [==============================] - 16s 68us/step - loss: 0.0016
- accuracy: 0.9996
Epoch 54/100
227845/227845 [==============================] - 15s 68us/step - loss: 0.0016
- accuracy: 0.9995
Epoch 55/100
227845/227845 [==============================] - 18s 78us/step - loss: 0.0016
- accuracy: 0.9996
Epoch 56/100
227845/227845 [==============================] - 17s 73us/step - loss: 0.0016
- accuracy: 0.9996
Epoch 57/100
227845/227845 [==============================] - 17s 74us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 58/100
227845/227845 [==============================] - 16s 71us/step - loss: 0.0016
- accuracy: 0.9996
Epoch 59/100
227845/227845 [==============================] - 17s 75us/step - loss: 0.0016
- accuracy: 0.9996
Epoch 60/100
227845/227845 [==============================] - 17s 73us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 61/100
227845/227845 [==============================] - 16s 72us/step - loss: 0.0016
- accuracy: 0.9996
Epoch 62/100
227845/227845 [==============================] - 16s 72us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 63/100
227845/227845 [==============================] - 13s 59us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 64/100
227845/227845 [==============================] - 15s 66us/step - loss: 0.0014
- accuracy: 0.9996
```

```
Epoch 65/100
227845/227845 [==============================] - 16s 69us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 66/100
227845/227845 [==============================] - 15s 65us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 67/100
227845/227845 [==============================] - 15s 65us/step - loss: 0.0016
- accuracy: 0.9996
Epoch 68/100
227845/227845 [==============================] - 13s 56us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 69/100
227845/227845 [==============================] - 13s 59us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 70/100
227845/227845 [==============================] - 15s 66us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 71/100
227845/227845 [==============================] - 13s 56us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 72/100
227845/227845 [==============================] - 13s 56us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 73/100
227845/227845 [==============================] - 12s 53us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 74/100
227845/227845 [==============================] - 12s 51us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 75/100
227845/227845 [==============================] - 12s 53us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 76/100
227845/227845 [==============================] - 12s 54us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 77/100
227845/227845 [==============================] - 12s 54us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 78/100
227845/227845 [==============================] - 16s 68us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 79/100
227845/227845 [==============================] - 17s 74us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 80/100
227845/227845 [==============================] - 17s 76us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 81/100
227845/227845 [==============================] - 17s 75us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 82/100
227845/227845 [==============================] - 16s 68us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 83/100
227845/227845 [==============================] - 16s 72us/step - loss: 0.0014
- accuracy: 0.9996
```

```
Epoch 84/100
227845/227845 [==============================] - 16s 69us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 85/100
227845/227845 [==============================] - 16s 70us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 86/100
227845/227845 [==============================] - 16s 72us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 87/100
227845/227845 [==============================] - 17s 72us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 88/100
227845/227845 [==============================] - 16s 70us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 89/100
227845/227845 [==============================] - 16s 70us/step - loss: 0.0013
- accuracy: 0.9996
Epoch 90/100
227845/227845 [==============================] - 16s 71us/step - loss: 0.0013
- accuracy: 0.9996
Epoch 91/100
227845/227845 [==============================] - 16s 70us/step - loss: 0.0016
- accuracy: 0.9995
Epoch 92/100
227845/227845 [==============================] - 17s 74us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 93/100
227845/227845 [==============================] - 13s 55us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 94/100
227845/227845 [==============================] - 13s 57us/step - loss: 0.0015
- accuracy: 0.9996
Epoch 95/100
227845/227845 [==============================] - 13s 58us/step - loss: 0.0014
- accuracy: 0.9996
Epoch 96/100
227845/227845 [==============================] - 13s 58us/step - loss: 0.0013
- accuracy: 0.9996
Epoch 97/100
227845/227845 [==============================] - 13s 58us/step - loss: 0.0013
- accuracy: 0.9996
Epoch 98/100
227845/227845 [==============================] - 13s 57us/step - loss: 0.0013
- accuracy: 0.9996
Epoch 99/100
227845/227845 [==============================] - 12s 54us/step - loss: 0.0013
- accuracy: 0.9996
Epoch 100/100
227845/227845 [==============================] - 12s 52us/step - loss: 0.0013
- accuracy: 0.9996
```

```
Out[64]: <keras.callbacks.callbacks.History at 0x12f757b1e48>
```

In [66]:
```
# Predicting the Test set results
y_pred = classifier.predict(x_test)
y_pred = (y_pred > 0.5)
```

In [67]:
```
score = classifier.evaluate(x_test, y_test)
score
```

```
56962/56962 [==============================] - 2s 32us/step
```

Out[67]:
```
[0.004924485128219564, 0.9993504285812378]
```

In [69]:
```
# Making the Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[69]:
```
array([[56843,    18],
       [   19,    82]], dtype=int64)
```

In [70]:
```
#Let's see how our model performed
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

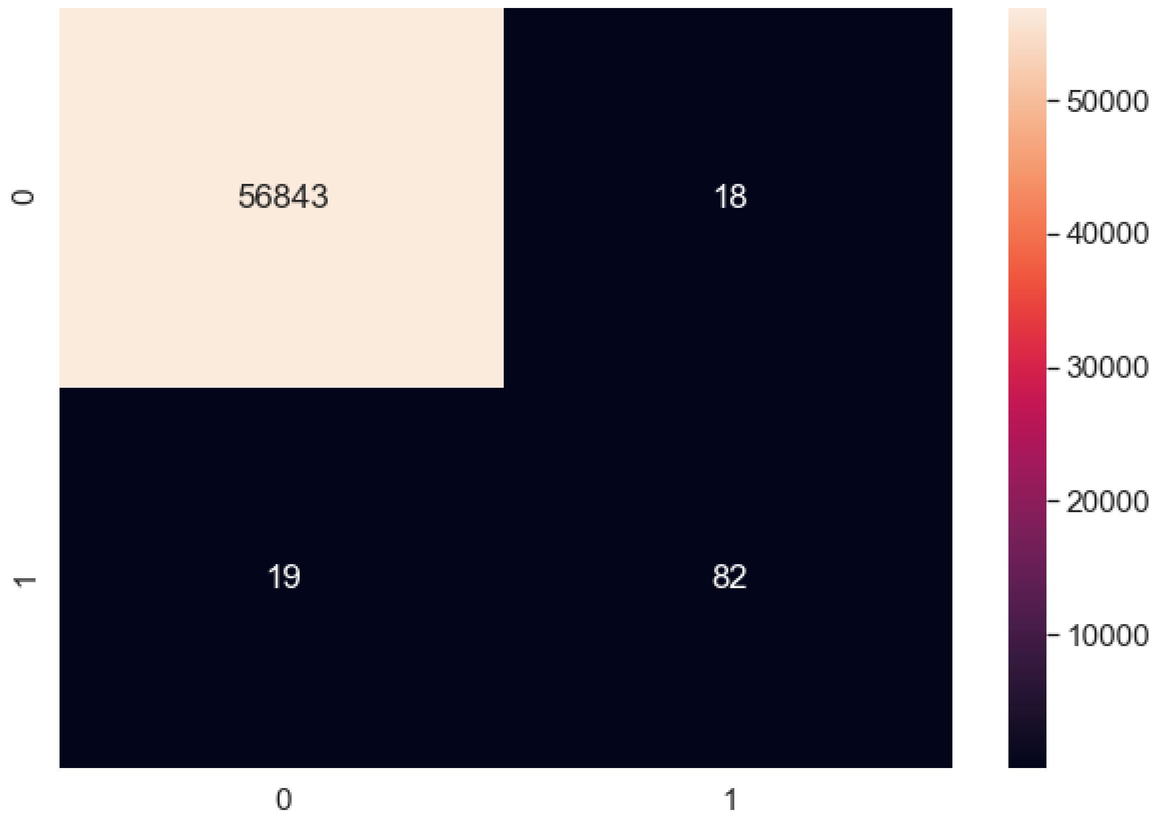
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.82      0.81      0.82       101

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

In [72]:
```
## EXTRA: Confusion Matrix
cm = confusion_matrix(y_test, y_pred) # rows = truth, cols = prediction
df_cm = pd.DataFrame(cm, index = (0, 1), columns = (0, 1))
plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)
sns.heatmap(df_cm, annot=True, fmt='g')
print("Test Data Accuracy: %0.4f" % accuracy_score(y_test, y_pred))
```

```
Test Data Accuracy: 0.9994
```



In [ ]: