Basic approach to generate insights from data and use of ML

--- Author **Anmol Yadav**

--- Dataset from Kaggle(creditcard)

importing required libraries

In [1]:
```
import numpy as np
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import sys
import matplotlib.pyplot as plt
```

In [2]:
```
import seaborn as sns
import scipy
import sklearn
import statistics
import json
```

reading the data

In [3]:
```
cc_df = pd.read_csv("creditcard.csv")
```

exploring the data

In [4]:
```
cc_df.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	0.0
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

In [5]:
```
columns = list(cc_df.columns)
print("Columns list for this Dataset : \n {}".format(columns))
```

```
Columns list for this Dataset :
 ['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V1
7', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount', 'Class']
```

there are total of 31 columns with Column name "Class" is target variable here we can see that there are 28 variables Naming from V1 - V28 which looks like small values real number(we will explore in details further) We can see other 2 variables named : "Time" and "Amount"

In [6]:
```
print("Shape of the given dataset is : {} ".format(cc_df.shape))
```

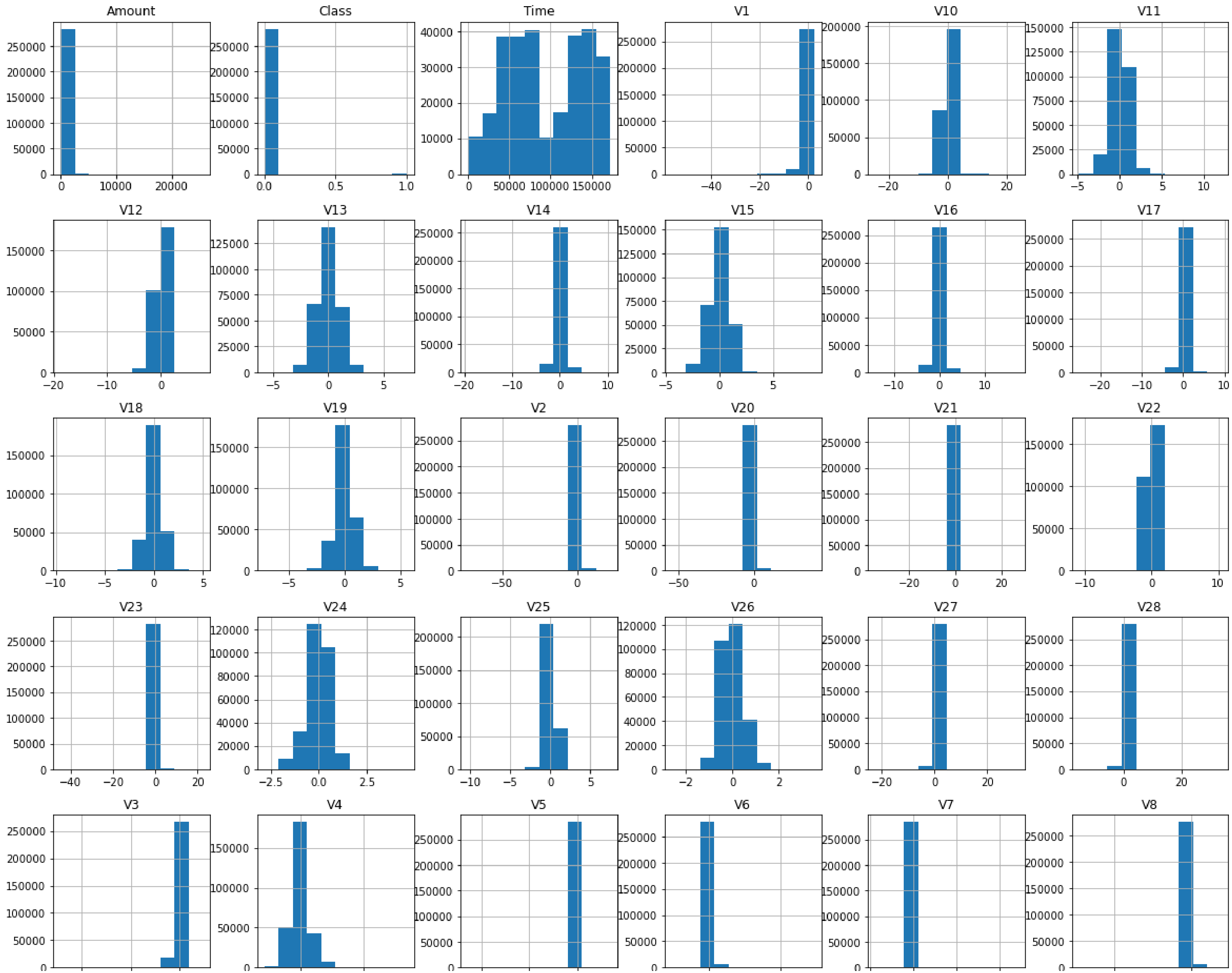
```
Shape of the given dataset is : (284807, 31)
```

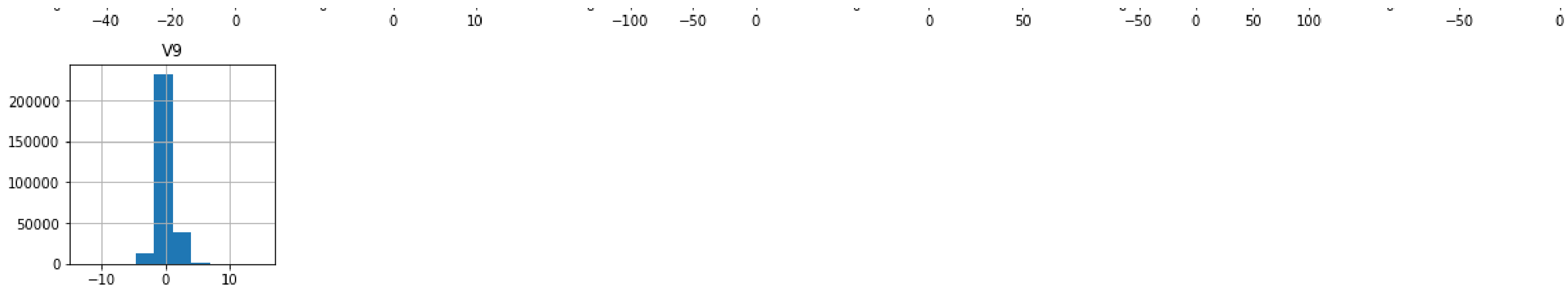
we can see that number of rows are 284807

now we will plot histograms for each variable

In [7]:
```
# plot the histogram of each parameter
cc_df.hist(figsize=(20, 20),bins = 10)
plt.show()
```

```
c:\users\anmol\appdata\local\programs\python\python36\lib\site-packages\pandas\plotting\_matplotlib\tools.py:298: Mat
plotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor releases later. Use ax.get_subplo
tspec().rowspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
c:\users\anmol\appdata\local\programs\python\python36\lib\site-packages\pandas\plotting\_matplotlib\tools.py:298: Mat
plotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor releases later. Use ax.get_subplo
tspec().colspan.start instead.
  layout[ax.rowNum, ax.colNum] = ax.get_visible()
c:\users\anmol\appdata\local\programs\python\python36\lib\site-packages\pandas\plotting\_matplotlib\tools.py:304: Mat
plotlibDeprecationWarning:
The rowNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor releases later. Use ax.get_subplo
tspec().rowspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
c:\users\anmol\appdata\local\programs\python\python36\lib\site-packages\pandas\plotting\_matplotlib\tools.py:304: Mat
plotlibDeprecationWarning:
The colNum attribute was deprecated in Matplotlib 3.2 and will be removed two minor releases later. Use ax.get_subplo
tspec().colspan.start instead.
  if not layout[ax.rowNum + 1, ax.colNum]:
```


Amount
0 10000 20000
Class
0.0 0.5 1.0
Time
0 50000 100000 150000
V1
−40 −20 0
V10
−20 0 20
V11
−5 0 5 10
V12
−20 −10 0
V13
−5 0 5
V14
−20 −10 0 10
V15
−5 0 5
V16
−10 0 10
V17
−20 −10 0 10
V18
−10 −5 0 5
V19
−5 0 5
V2
−50 0
V20
−50 0
V21
−20 0 20
V22
−10 0 10
V23
−40 −20 0 20
V24
−2.5 0.0 2.5
V25
−10 −5 0 5
V26
−2 0 2
V27
−20 0 20
V28
0 20
V3
V4
V5
V6
V7
V8



Important observations from the above plots:

1. Class has very few 1 values (Fraud cases)
2. Values of variables : name starting starting with V are very small in magnitude
3. we also notice that these values are very small that they centered around zero(0)

determining number of frauds cases

In [8]:
```
fraud_cases = cc_df[cc_df["Class"] ==1]
non_fraud_cases = cc_df[cc_df["Class"] == 0]
```

In [9]:
```
## % of fraud from total data

fraud_perc = round(len(fraud_cases)/len(cc_df)*100,2)
print("% of fraud cases from total data {} %".format(fraud_perc))
```

```
% of fraud cases from total data 0.17 %
```

In [10]:
```
### total fraud cases from dataset

print("Fraud Cases : {}".format(len(fraud_cases)))
print("Non-Fraud Cases : {}".format(len(non_fraud_cases)))
```

```
Fraud Cases : 492
Non-Fraud Cases : 284315
```

transaction amount difference between fraud cases and non-fraud cases

In [11]:
```
## average transcation amount for fraud cases
avg_amount_fraud = statistics.mean(fraud_cases["Amount"])
```

In [12]:
```
print("Average transcation amount for fraud cases  : {}".format(avg_amount_fraud))
```

```
Average transcation amount for fraud cases  : 122.21132113821139
```

In [13]:
```
## average transcation amount for non-fraud cases
avg_amount_non_fraud = statistics.mean(non_fraud_cases["Amount"])
```

In [14]:
```
print("Average transcation amount for non-fraud cases  : {}".format(avg_amount_non_fraud))
```

```
Average transcation amount for non-fraud cases  : 88.29102242231328
```

NOTE: very important observation from above statistics is that fraud cases transaction have higher revenue involved than normal transaction

We will figure out further how each independent variables are related to each other by Pearson correlation matrix

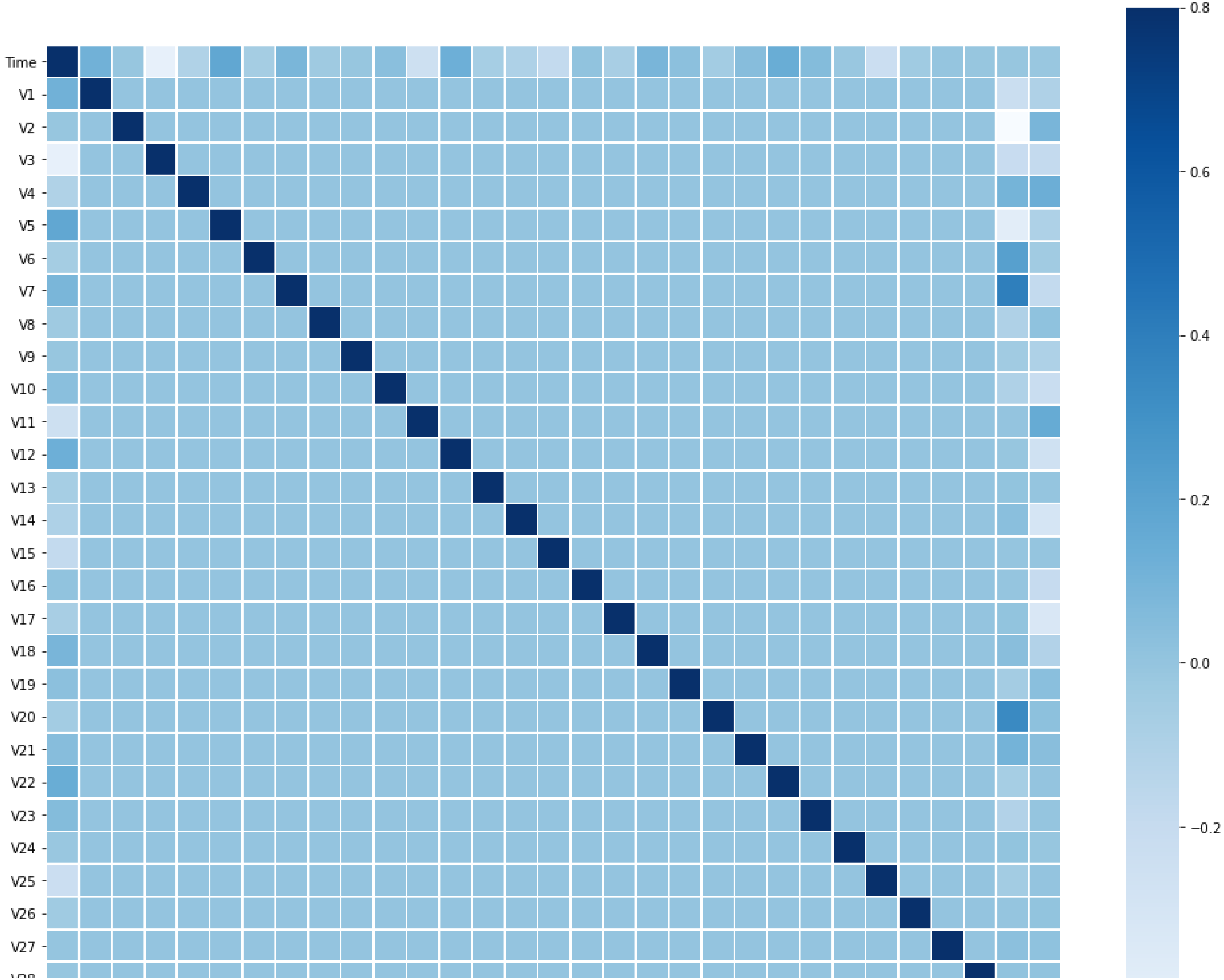
The Pearson correlation coefficient is a statistic that measures linear correlation between two variables X1 and X2. It has a value between +1 and −1. A value of +1 is total positive linear correlation, 0 is no linear correlation, and −1 is total negative linear correlation

NOTE: We will only talk about magnitude of correlation in following inferences

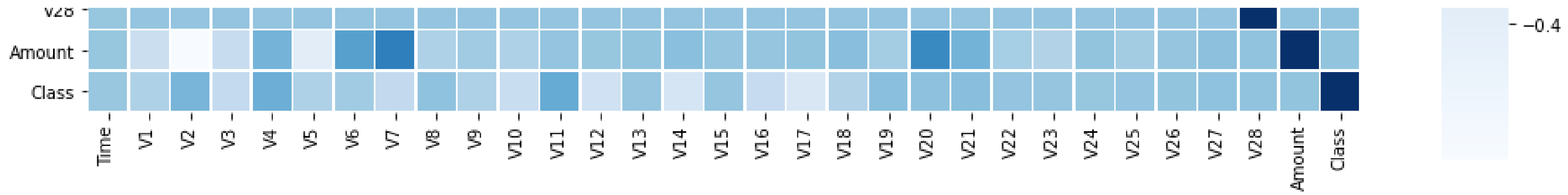
In [15]:

```
# Using Correlation
plt.figure(figsize=(17,15))
pear_corr_matrix = cc_df.corr()

sns.heatmap(pear_corr_matrix   ,  cmap=plt.cm.Blues, linewidths=.5 , vmax = .8, square = True)
plt.show()
```



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
0.8
0.6
0.4
0.2
0.0
−0.2



NOTE:

1. Each sqaure box is a correlation of correspoding X-axis variable to respective Y-axis variable
2. Light color means less magnitude of correlation which means variables are less related to each other
3. Darker color means high magnitude, which means high correlation between variables

Imppotant Observations :

1. As we see very large amount of light color square box, which means variables are hardly related to each other
2. Amount variable is bit related to few variables like V7, V20 , that too not to large extent

Machine Learning Technique

We will use this above split method in coming steps along with nested cross validation and hyper-parameter tuning

I am going to use **Decision Tree Classifier** for this dataset

following libraries are required for coming steps

In [16]:
```
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold
import itertools
import time
import sklearn.datasets
from skopt import BayesSearchCV as bayes_opt
from sklearn.tree import DecisionTreeClassifier as dt
```

1. Decision Tree parameters space , this will be used at the time of Hyper-parameter tuning
2. Range of values of these parameters are provided and Tuning method will tune between these ranges
3. Then best valued parameters are choosen for the classifier

In [17]:
```
### decision tree space
decision_tree_space = {'splitter':['best','random'],
          'max_depth': (1,10),
          'criterion':['gini','entropy'],
          'min_samples_split':(1,75)}
```

scoring dictionary stores evaluation metrics, on comparing values of these metrics best model is to be chosen from list of models options

In [18]:
```
### scoring dictionary
scoring_dic = {'AUC': 'roc_auc',
           'avg_pr': 'average_precision',
           'bal_accuracy':'balanced_accuracy',
           'accuracy':'accuracy',
           'f1_score':'f1'}
```

Now we will use nested cross validation to create different models for a single classifier bu using data folding technique inner cv function do following steps:

1. Hyper-parameter tuning using Stratified Cross Validation
2. Used as nested CV for outer CV

In [19]:
```
def inner_cv(X, y, n_folds, spaces, n_iter ,seed, scoring ):

        ## decision tree estimator
        estimator = dt(random_state = seed)

        ## this is inner cross validation , n_folds folds for train corresponding outer fold train data
        cv = StratifiedKFold(n_splits=n_folds,shuffle=True,random_state=seed+1)

        ## Bayesian optimization method is used with "n_iter" as number of iterations for hyper params tuning
        optimization = bayes_opt(cv = cv, estimator = estimator, n_iter = n_iter, n_jobs = -1,
                                 search_spaces=spaces, n_points=100, random_state = seed+2,
                                 error_score=-1, return_train_score=True, verbose=0,
                                 iid = True, refit = True, scoring = scoring)

        ## we fit the bayesian optimizer with train (X) and test (y) data
        model = optimization.fit(X=X,y=y)

        ## cross validation score for model by inner_cv is printed
        print("CV score: ",model.best_score_)

        return(model)
```

-> ounter_cv function acts as first step for nested cross-validation -> Here we :

1. define our model evaluation metrics
2. set seed for each inner and outer cv
3. folds data using Stratified K Fold method
4. split train and test data for each inner cv

-> There are n-number of models for N outer fold

-> For building each model, loop in run from 1-N

-> Evaluation metrics are stored after each loop (i.e. result of model from inner_cv)

In [20]:
```
def outer_cv(X,y, n_fold, spaces, seed, scoring):

    # StratifiedKFold helps making n_fold of given data
    cv = StratifiedKFold(n_splits = n_fold, shuffle=False,random_state = seed)

    # created empty list "models" in which we will append model after each inner_cv result
    models = []

    # dictioanry with keys as evaluation metrics, whose values will be appended after result of each inner_cv result(m
odel result)
    evaluation_metrics_dic = {
        'average_precision':[],
        'roc_auc':[],
        'balanced_accuracy':[],
        'f1':[],'accuracy':[]
    }

    # this list will have mean value of cross-validation score
    avg_cv_scores = []

    ## flag for printing model status currently trainig/predictiong
    loop = 1

    # changing random seed value after n_fold data split
    random_state = 1+seed
    for train_index, test_index in cv.split(X,y):

        X_train, X_test = X.iloc[train_index], X.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]

        print("Training Model-{}".format(loop))

        model = inner_cv(X=X_train, y=y_train, n_folds=4, n_iter=10, spaces = spaces, scoring = scoring,
                         seed = random_state)

        # changing random seed value after each inner cv output, so to have distinct seed value for each model
        random_state +=1

        # inner_cv results in creating decision tree classifier model, so "models" is appended
        models.append(model)
```

```python
        # appending model's cross-validation score for each model
        avg_cv_scores.append(model.best_score_)


        ## prediction
        print("Predicting Model-{}".format(loop))

        # model prediction on test data of each outer fold
        predictions = model.predict_proba(X_test)[:,1]

        ### comparing following evaluation metrics result of predicted "Class" with actual class
        ### and appending corresponding  metric value for this model to evaluation_metrics_dic dictionary


        ## accuracy
        accuracy_value = metrics.accuracy_score(y_true = y_test, y_pred=predictions>0.5)
        evaluation_metrics_dic['accuracy'].append(accuracy_value)

        ## balanced_accuracy
        balanced_accuracy_value = metrics.balanced_accuracy_score(y_true = y_test, y_pred=predictions>0.5)
        evaluation_metrics_dic['balanced_accuracy'].append(balanced_accuracy_value)

        ## f1
        f1_value = metrics.f1_score(y_true = y_test, y_pred=predictions>0.5)
        evaluation_metrics_dic['f1'].append(f1_value)

        ## roc_auc
        roc_auc_value = metrics.roc_auc_score(y_true = y_test, y_score=predictions)
        evaluation_metrics_dic['roc_auc'].append(roc_auc_value)

        ## average_precision
        average_precision_value = metrics.average_precision_score(y_true = y_test, y_score=predictions)
        evaluation_metrics_dic['average_precision'].append(average_precision_value)

        loop += 1

    return(models,evaluation_metrics_dic,avg_cv_scores)
```

In [21]:
```
## data without label(X)
X = cc_df.drop(['Time','Class'],axis=1)

## data with only label(y)
y = cc_df["Class"]
```

In [22]:
```
## time just before start of training ML models
t1 = time.time()
```

In [23]:
```
trained_models,evaluation_metrics_dic, avg_cv_scores = outer_cv(X=X, y=y, n_fold = 5,
                                                          spaces = decision_tree_space, seed = 42,
                                                          scoring = scoring_dic["f1_score"])
```

```
c:\users\anmol\appdata\local\programs\python\python36\lib\site-packages\sklearn\model_selection\_split.py:296: Future
Warning: Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You should le
ave random_state to its default (None), or set shuffle=True.
  FutureWarning

Training Model-1
CV score:  0.8081431923937207
Predicting Model-1
Training Model-2
CV score:  0.8075984260991482
Predicting Model-2
Training Model-3
CV score:  0.8343560983579774
Predicting Model-3
Training Model-4
CV score:  0.7968937822851166
Predicting Model-4
Training Model-5
CV score:  0.8381805379874928
Predicting Model-5
```

In [24]:
```
## time just after completion of training and prediction of ML models
t2 = time.time()
```

In [25]:
```
### we can know time taken to run 5 fold nested cv method for Decision Tree Classifier for this data

print("Time taken for training the DT models : {} seconds or {} minutes".format(t2-t1,(t2-t1)/60))
```

```
Time taken for training the DT models : 420.07944798469543 seconds or 7.001324133078257 minutes
```

In [26]:
```
print("Number of Decision Tree Models : {}".format(len(trained_models)))
```

```
Number of Decision Tree Models : 5
```

evaluation_metrics_dic is dictionary containg evaluation metrics for models

In [27]:
```
### printing evaluation_metrics_dic in beautify manner
print(json.dumps(evaluation_metrics_dic, sort_keys=True, indent=4))
```


```
{
    "accuracy": [
        0.9980162213405428,
        0.9995786664794073,
        0.9968926107336599,
        0.999350432752234,
        0.9991924299081828
    ],
    "average_precision": [
        0.3761252116275466,
        0.8268267288645288,
        0.514848071855028,
        0.6876511168251643,
        0.6715364070858009
    ],
    "balanced_accuracy": [
        0.9183389919809033,
        0.9090381507067226,
        0.7998069653269746,
        0.8468772240619864,
        0.8213318477242295
    ],
    "f1": [
        0.5949820788530467,
        0.8709677419354839,
        0.4,
        0.7861271676300577,
        0.7325581395348839
    ],
    "roc_auc": [
        0.8863239787566678,
        0.9273376716001973,
        0.8440279124153398,
        0.8821200041488906,
        0.8778787863561792
    ]
}
```

In [28]:
```
## mean value of f1 scores of all 5 models

"Mean F1 scores of 5 models : {}".format(np.mean(evaluation_metrics_dic["f1"]))
```

Out[28]: 'Mean F1 scores of 5 models : 0.6769270255906944'

In [29]:
```
## standard deviation of f1 scores of 5 models

"Standard deviation of F1 scores for 5 models : {}".format(np.std(evaluation_metrics_dic["f1"]))
```

Out[29]: 'Standard deviation of F1 scores for 5 models : 0.16496959917392814'

conclusion

f1 score : -.15 || 0.6919 || +.15

As we see, f1 score of varies too great extent as **standard deviation** is of **.15** this means for some part of data decision trees works fine but for other it's not good at all So this can be the case with unseen data that a classifier performs better for a part of data but fails to give correct prediction for new unseen data NOTE : So basically whole point of using 5 fold nested cross validation was to chech if my classifier is good for unseen data or not .. So it's just an example

we should be not sure about any ML algorithm at it's first result only.

In [30]:
```
##
for i in trained_models:print(i.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 2, 'splitter': 'random'}
{'criterion': 'entropy', 'max_depth': 5, 'min_samples_split': 15, 'splitter': 'best'}
{'criterion': 'entropy', 'max_depth': 8, 'min_samples_split': 11, 'splitter': 'best'}
{'criterion': 'gini', 'max_depth': 7, 'min_samples_split': 5, 'splitter': 'best'}
{'criterion': 'entropy', 'max_depth': 6, 'min_samples_split': 11, 'splitter': 'best'}
```

Above are results of best hyper-parameters selected for each models Hyperparameter tuning is effective way to find best fit values of hyper-parameters(out of given ranges values for each hyper params) for a given model