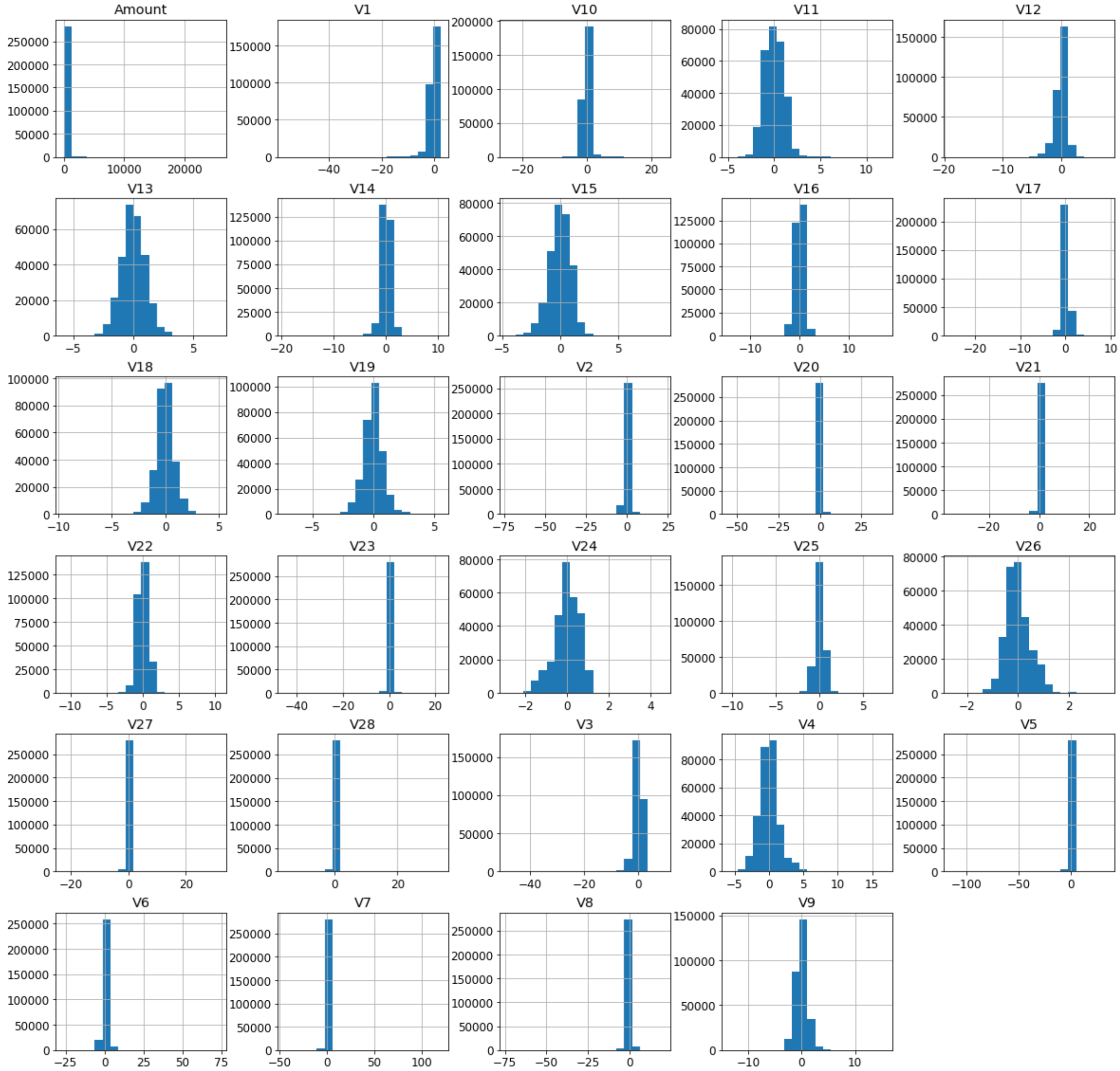
In [111]:

```
# -*- coding: utf-8 -*-
"""
Created on Sun May  5 19:20:05 2019
Author GRP 47
"""

# import the external libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

# Load the dataset from the csv file using pandas
data = pd.read_csv('creditcard.csv')
# data = data.sample(frac=0.1, random_state = 1) # use this for quick calculations
data.head()

# Only use the 'Amount' and 'V1', ..., 'V28' features
features = ['V%d' % number for number in range(1, 29)] + ['Amount']

# The target variable which we would like to predict, is the 'Class' variable (we use this to label our data)
target = 'Class'

# Creating an X variable (containing the features) and an y variable (containing only the target variable)
X = data[features]
y = data[target]



# Plot histograms of each parameter
X.hist(figsize = (20, 20), bins = 20)
plt.show()


# run preprocessing to reduce workload
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaler.fit(X)

# Plot histograms of each parameter
#X.hist(figsize = (20, 20), bins = 20)
#plt.show()


#Split the data set using 'train_test_split' function
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101, shuffle=True)




totalTrueFrauds = y_test.sum()
print("Total number of true frauds in test set: %d" % (totalTrueFrauds))
```

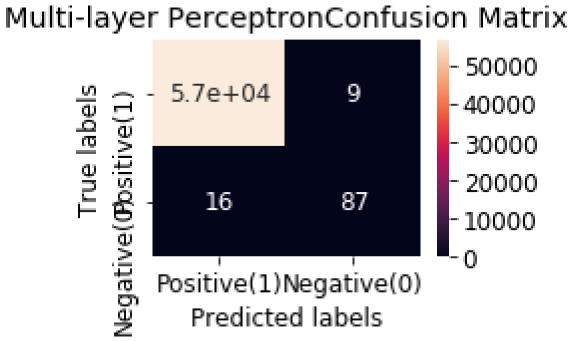


```
Total number of true frauds in test set: 103
```

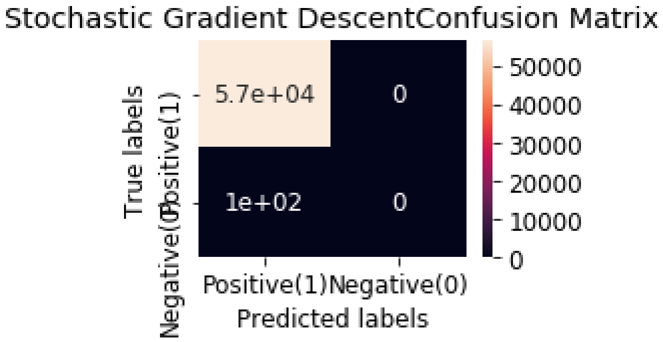
In [110]:

```
# Instantiate the model to an empty object
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import SGDClassifier

Fraud = y_test.sum()
Valid = len(y_test) - y_test.sum()
state = 1
outlier_fraction = Fraud/Valid


classifiers = {
#    "Isolation Forest": IsolationForest(max_samples=len(X_train),
#                                        contamination=outlier_fraction,
#                                        random_state=state, max_features = 1, n_estima
tors = 2),
#    "Local Outlier Factor": LocalOutlierFactor(n_neighbors=20,
#                                               contamination=outlier_fraction, novelty
=True),

    "Multi-layer Perceptron": MLPClassifier(activation='relu',
                                            alpha=0.0001,
                                            hidden_layer_sizes=(50, 50, 50),
                                            learning_rate='constant',
                                            solver='adam'),


    "Stochastic Gradient Descent": SGDClassifier(alpha=0.1,
                                                 epsilon=0.1,
                                                 l1_ratio=1e-05,
                                                 loss='log', #loss='squared_hinge',
                                                 penalty='l1'),

    "Logistic Regression": LogisticRegression(max_iter=100,
                                              solver='liblinear',
                                              tol=0.001)
    }
#activation='relu',alpha=0.0001,hidden_layer_sizes=(50, 50, 50),learning_rate='constan
t',solver='adam'

#activation='tanh',alpha=0.0001,hidden_layer_sizes=(50,50,50),learning_rate='constant',
solver='adam'),
# activation='relu',alpha=0.001,hidden_layer_sizes=(20, 50, 100, 100, 50, 20),learning_
rate='constant',solver='adam'

plt.rcParams.update({'font.size':12}) # 28

for i, (clf_name, clf) in enumerate(classifiers.items()):

    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    #used for saving the predicted table.
    if i == 0:
        y_prob0 = clf.predict_proba(X_test)
        y_pred0 = clf.predict(X_test)
    elif i == 1:
        y_prob1 = clf.predict_proba(X_test)
        y_pred1 = clf.predict(X_test)
    elif i == 2:
        y_prob2 = clf.predict_proba(X_test)
        y_pred2 = clf.predict(X_test)


    n_errors = (y_pred != y_test).sum()
    totalPredFrauds = y_pred.sum()

    # Run classification metrics
    from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
    print('{}: {}'.format(clf_name, n_errors))
    print(accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred))

    results = confusion_matrix(y_test, y_pred)
    print( 'Confusion Matrix :')
    print(results)
    plt.figure()

    fig, ax = plt.subplots(figsize=(3,2))

    #ax= plt.subplot()
    sns.heatmap(results, annot=True, ax = ax ); #annot=True to annotate cells

    # labels, title and ticks
    ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
    ax.set_title(clf_name + 'Confusion Matrix');
    ax.xaxis.set_ticklabels(['Positive(1)', 'Negative(0)']); ax.yaxis.set_ticklabels([
'Positive(1)', 'Negative(0)']);

    plt.show()

    print()
```

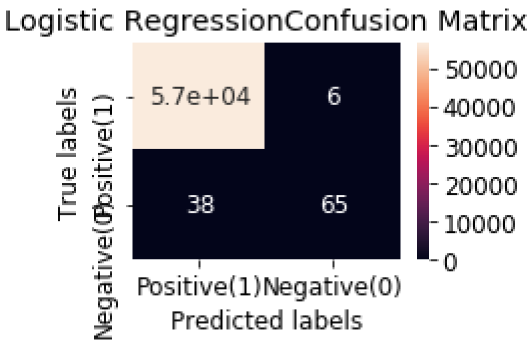
```
Multi-layer Perceptron: 25
0.9995611109160493
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56859
           1       0.91      0.84      0.87       103

   micro avg       1.00      1.00      1.00     56962
   macro avg       0.95      0.92      0.94     56962
weighted avg       1.00      1.00      1.00     56962

Confusion Matrix :
[[56850     9]
 [   16    87]]

<Figure size 432x288 with 0 Axes>
```



```
Stochastic Gradient Descent: 103
0.9981917769741231
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56859
           1       0.00      0.00      0.00       103

   micro avg       1.00      1.00      1.00     56962
   macro avg       0.50      0.50      0.50     56962
weighted avg       1.00      1.00      1.00     56962

Confusion Matrix :
[[56859     0]
 [  103     0]]

<Figure size 432x288 with 0 Axes>
```



```
Logistic Regression: 44
0.9992275552122467
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56859
           1       0.92      0.63      0.75       103

   micro avg       1.00      1.00      1.00     56962
   macro avg       0.96      0.82      0.87     56962
weighted avg       1.00      1.00      1.00     56962

Confusion Matrix :
[[56853     6]
 [   38    65]]

<Figure size 432x288 with 0 Axes>
```

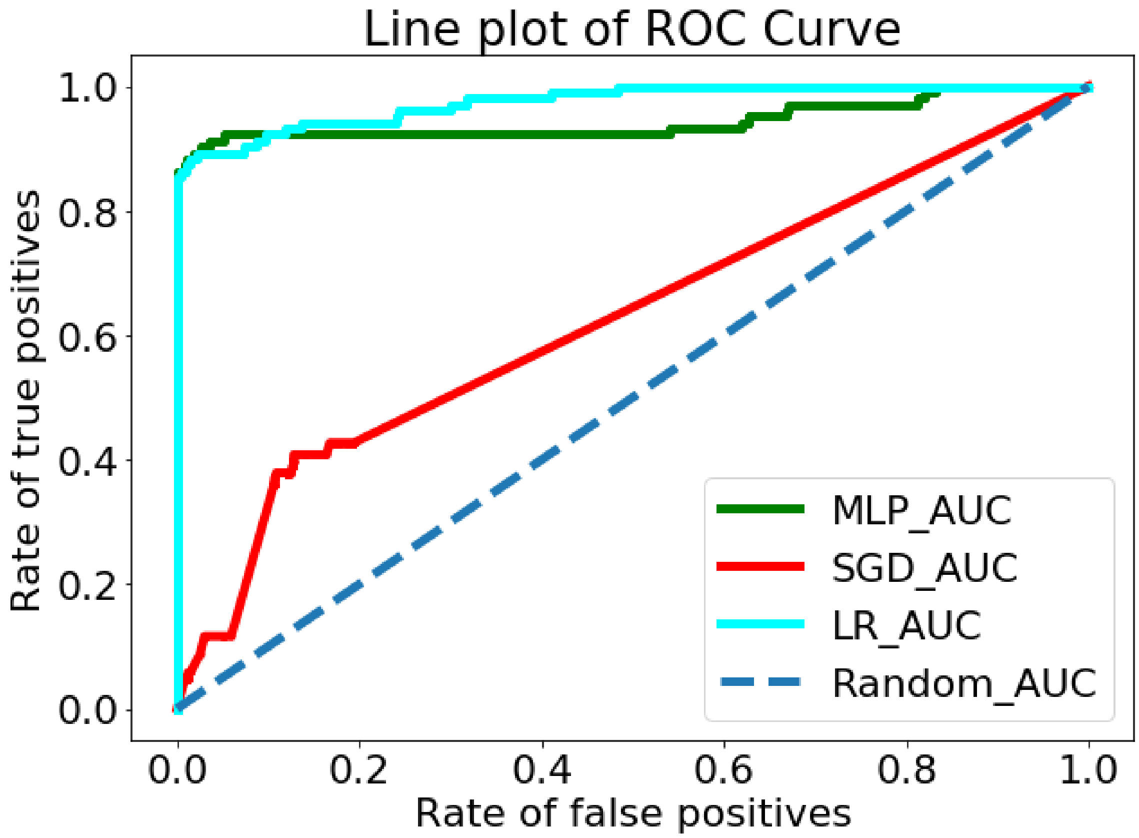


In [112]:

```
# roc curve and auc
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib.pyplot import figure

# generate 2 class dataset
#X, y = make_classification(n_samples=1000, n_classes=2, weights=[1,1], random_state=1)
# split into train/test sets
#trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)
# fit a model
#model = KNeighborsClassifier(n_neighbors=3)
#model.fit(trainX, trainy)
# predict probabilities
#probs = model.predict_proba(testX)
# keep probabilities for the positive outcome only

#setup plot parameters
plt.figure(figsize=(10,7)) # 25,15
plt.rcParams.update({'font.size':22}) # 28

## MLP
prob_MLP = y_prob0[:, 1]
# calculate AUC
auc0 = roc_auc_score(y_test, prob_MLP)
print('MLP_AUC: %.3f' % auc0)
# calculate roc curve
fpr0, tpr0, thresholds0 = roc_curve(y_test, prob_MLP)
# plot the roc curve for the model
plt.plot(fpr0, tpr0, color='green', marker='.', label='MLP_AUC', linewidth=5)

## Stochastic Gradient decent
prob_SGD = y_prob1[:, 1]
# calculate AUC
auc1 = roc_auc_score(y_test, prob_SGD)
print('SGD_AUC: %.3f' % auc1)
# calculate roc curve
fpr1, tpr1, thresholds1 = roc_curve(y_test, prob_SGD)
# plot the roc curve for the model
plt.plot(fpr1, tpr1, color='red', marker='.', label='SGD_AUC', linewidth=5)

## Logistic Regression
prob_LR = y_prob2[:, 1]
# calculate AUC
auc2 = roc_auc_score(y_test, prob_LR)
print('LR_AUC: %.3f' % auc2)
# calculate roc curve
fpr2, tpr2, thresholds2 = roc_curve(y_test, prob_LR)
# plot the roc curve for the model
plt.plot(fpr2, tpr2, color='cyan',marker='.', label='LR_AUC', linewidth=5)

# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--', label='Random_AUC', linewidth=5)
legend = pyplot.legend();
plt.title('Line plot of ROC Curve')
plt.xlabel('Rate of false positives')
plt.ylabel('Rate of true positives')
# show the plot
```

```
plt.show()
```

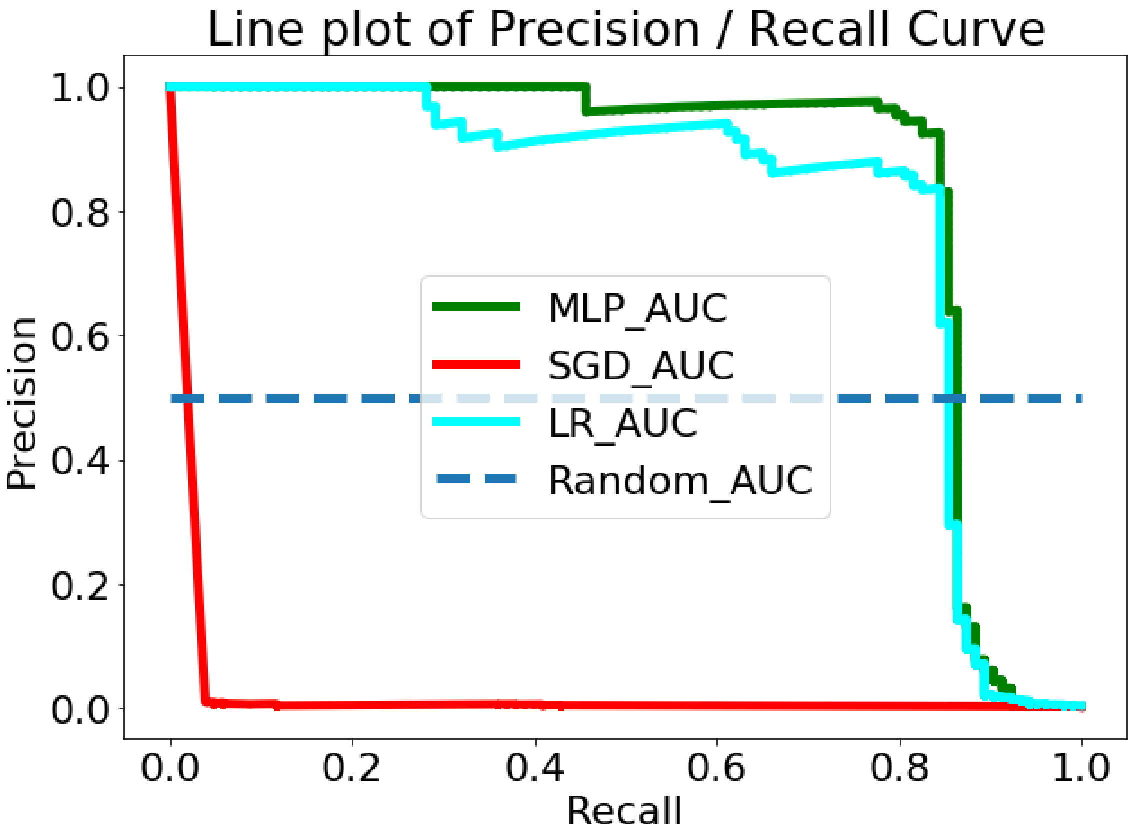
```
MLP_AUC: 0.944
SGD_AUC: 0.627
LR_AUC: 0.975
```



In [113]:

```
# precision-recall curve and f1
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
from sklearn.metrics import average_precision_score

plt.figure(figsize=(10,7)) # 25,15
plt.rcParams.update({'font.size':22}) # 28

prob_MLP = y_prob0[:, 1]
# get precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, prob_MLP)
# get F1 score
f1_MLP = f1_score(y_test, y_pred0)
# get precision-recall AUC
auc_MLP = auc(recall, precision)
# get average precision score
ap_MLP = average_precision_score(y_test, prob_MLP)
print('MLP Scores: F1=%.3f AUC=%.3f AP=%.3f' % (f1_MLP, auc_MLP, ap_MLP))
# plot precision-recall curve for the model
plt.plot(recall, precision, color='green',marker='.', label='MLP_AUC', linewidth=5)

prob_SGD = y_prob1[:, 1]
# get precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, prob_SGD)
# get F1 score
f1_SGD = f1_score(y_test, y_pred2)
# get precision-recall AUC
auc_SGD = auc(recall, precision)
# get average precision score
ap_SGD = average_precision_score(y_test, prob_SGD)
print('SGD Scores: F1=%.3f AUC=%.3f AP=%.3f' % (f1_SGD, auc_SGD, ap_SGD))
# plot precision-recall curve for the model
plt.plot(recall, precision, color='red',marker='.', label='SGD_AUC', linewidth=5)

prob_LR = y_prob2[:, 1]
# get precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_test, prob_LR)
# get F1 score
f1_LR = f1_score(y_test, y_pred2)
# get precision-recall AUC
auc_LR = auc(recall, precision)
# get average precision score
ap_LR = average_precision_score(y_test, prob_LR)
print('LR  Scores: F1=%.3f AUC=%.3f AP=%.3f' % (f1_LR, auc_LR, ap_LR))
# plot precision-recall curve for the model
plt.plot(recall, precision, color='cyan',marker='.', label='LR_AUC', linewidth=5)

# plot no skill
plt.plot([0, 1], [0.5, 0.5], linestyle='--', label='Random_AUC', linewidth=5 )
legend = pyplot.legend();
plt.title('Line plot of Precision / Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
# show the plot
plt.show()
```

```
MLP Scores: F1=0.874 AUC=0.850 AP=0.850
SGD Scores: F1=0.747 AUC=0.023 AP=0.004
LR  Scores: F1=0.747 AUC=0.803 AP=0.804
```



```
In [ ]:
```