Credit Card Fraud Detection Model

```
import matplotlib.pyplot as plt
import seaborn as sns, numpy as np
from sklearn.ensemble import IsolationForest
from numpy import genfromtxt
from scipy.stats import multivariate_normal
from sklearn.metrics import f1_score
import io
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, roc_auc_score, roc_curve, precision_recall_curve, auc, make_scorer, recall_score, accuracy_score, pr

```

```
creditcardDF = pd.read_csv('/content/drive/My Drive/fraudData/creditcard.csv')
creditcardDF.head()#all numerical
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V1
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.99139
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.48909
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.71729
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.50775
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.34585

```
#Distribution of the label column
creditcardDF['Class'].value_counts()
```

```
0    284315
1       492
Name: Class, dtype: int64
```

```
492 / (284315  + 492) #<0.2 percent
```

```
0.001727485630620034
```

```
creditcardDF.shape
```

```
(284807, 31)
```

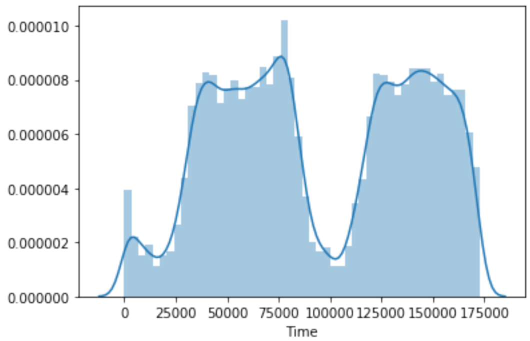
```
creditcardDF.isna().sum()#null checking
```

```
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
```

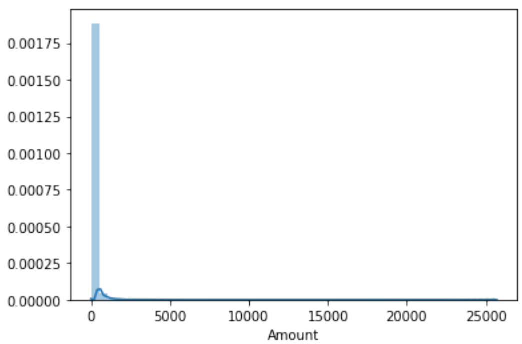
```


sns.distplot(creditcardDF['Time'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f248509b0>
```




```

sns.distplot(creditcardDF['Amount'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f24066a90>
```




```
creditcardDF['Amount'] = np.log(creditcardDF['Amount'] + 1)

creditcardDF['Time'] = np.log(creditcardDF['Time'] + 1)

normal = creditcardDF[creditcardDF['Class'] == 0]
anomaly = creditcardDF[creditcardDF['Class'] == 1]

train, small_normal = train_test_split(normal, test_size=0.2, random_state=0)
normal_valid, normal_test = train_test_split(small_normal, test_size=0.5, random_state=0)#good hd
anomaly_valid, anomaly_test = train_test_split(anomaly, test_size=0.5, random_state=0)#10 bad hd

validation = pd.concat([normal_valid, anomaly_valid])#include both good and bad, cross validation data in our text
test = pd.concat([normal_test, anomaly_test])#include both good and bad


print(validation.shape)
print(test.shape)

#reset_index
train = train.drop(columns = ['Class']).reset_index(drop= True)#no need of label in train data, drop it
print(train.shape)
```

```
(28677, 31)
(28678, 31)
(227452, 30)
```

```
featureNames = list(train.columns.values)#feature names only, no label
valFeatures = validation[featureNames].reset_index(drop= True)#feature df only, no label, will delete the current (sampled) index insteac
testFeatures = test[featureNames].reset_index(drop= True)#feature df only, no label

valLabel = validation['Class']#label df only
```

```
testLabel = test['Class']#label df only

```

```
valFeatures.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	
0	11.827043	-0.248023	1.259502	-0.993999	-1.587788	1.913462	-0.630270	1.958852	-0.659274	0.002373	0.718353	0.474332	0.297023	-0.
1	10.809566	-1.614505	-0.970137	1.730517	-1.715497	-0.869271	-0.171355	1.216768	-0.031314	0.992762	-2.191051	-1.019348	0.600947	0.
2	11.340380	1.106176	0.148096	0.424489	1.282916	-0.080275	0.146526	-0.007108	0.114953	-0.004731	0.033642	1.200830	1.342878	0.
3	11.321208	-1.791995	1.102738	0.324217	1.082267	-0.303348	-1.050303	0.066270	0.613586	-0.720545	-0.232754	-0.741686	0.317251	0.
4	11.956784	1.924286	0.324362	-0.734639	3.370481	0.783552	1.224944	-0.298881	0.291717	-0.790152	1.592072	-0.561561	-0.101690	-0.

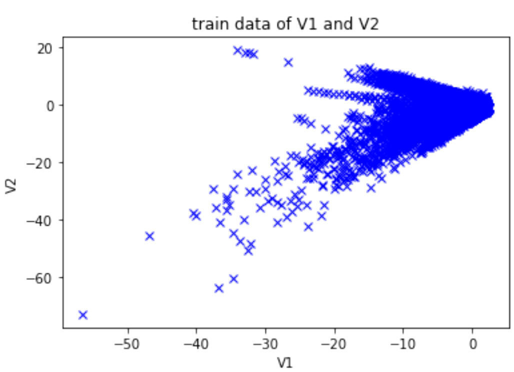
```
validation['Class'].value_counts()
```

```
0    28431
1      246
Name: Class, dtype: int64
```

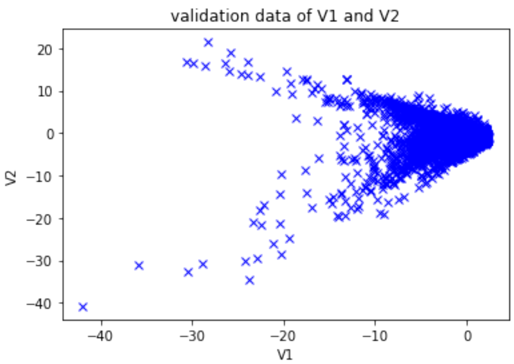
```
test['Class'].value_counts()



```

```
0    28432
1      246
Name: Class, dtype: int64
```

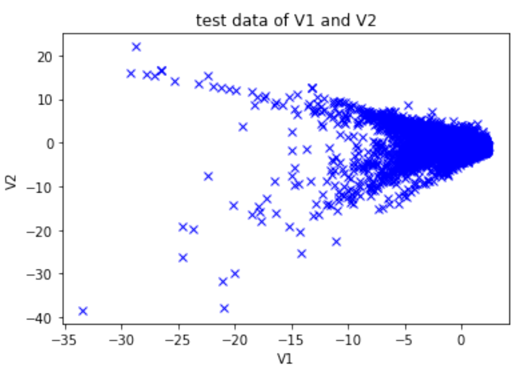
```
plt.figure()
plt.title("train data of V1 and V2")#may contain outliers
plt.xlabel("V1")
plt.ylabel("V2")
plt.plot(train.iloc[:, 1],train.iloc[:,2],"bx")
plt.show()



```




```
plt.figure()
plt.title("validation data of V1 and V2")
plt.xlabel("V1")
plt.ylabel("V2")
plt.plot(validation.iloc[:, 1],validation.iloc[:,2],"bx")
plt.show()
```




```
plt.figure()
plt.title("test data of V1 and V2")
plt.xlabel("V1")
plt.ylabel("V2")
plt.plot(test.iloc[:, 1],test.iloc[:,2],"bx")
plt.show()
```




```
# np.arange(1, 20, 2)
```

```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])
```

```
#find parameter for each col/feature in df for the Gaussian distribution
def estimateGaussian(dataset):
    mu = np.mean(dataset, axis=0)#vector
    sigma = np.cov(dataset.T)#matrix
    return mu, sigma

```

```

pdfVal = model.pdf(valFeatures)
print(max(pdfVal))#too small, can not differentiate
print(min(pdfVal))

p_val = model.logpdf(valFeatures)#Log of the pdf first, then apply to features, to change the magnitude of prob
print(max(p_val))#
print(min(p_val))
```

```
3.936022689247968e-12
0.0
-26.26085037221045
-7554.270217704667
```

```
p = model.logpdf(train)
print(p.shape)
print((p_val.shape))

# print(p)

```

```
(227452,)
(28677,)
```

```
print(p_val)
print(p_val < -500)
```

```
[  -31.28574735   -34.94205051   -27.79402451 ... -5175.93656039
  -4545.5057626    -29.7152192 ]
[False False False ...  True  True False]
```

```
[[1],[2],[3]]

ravel ->
[1,2,3]
```

```

#get the socre list
scores = []
p_val = model.logpdf(valFeatures)#Log of the pdf

# thresholds = np.linspace(-1000, -10,150)
thresholds = np.linspace(min(p_val), max(p_val),200)#generate all candidate threshold, epsilon

#find optimal threshold: bestThreshold
for threshold in thresholds:
  y_pred = (p_val < threshold).astype(int)# list of 0 and 1
  #calculate recall, precision and f1 for each (truth, pred) pair, corresponding to that threshold
  scores.append([recall_score(valLabel, y_pred),
                 precision_score(valLabel, y_pred),
                 f1_score(valLabel, y_pred, average = "binary")])

scores = np.array(scores)
maxIndex = scores[...,2].ravel().argmax()#maxIndex of the 3rd column (f1_score) #193, #.ravel return a flattened array
bestThreshold = thresholds[maxIndex]
print(scores.shape)#each row is a pair of (recall, precision, f1) corresponding to a threshold
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is ill-defined and bei
  _warn_prf(average, modifier, msg_start, len(result))
(200, 3)
```

```
print(scores)
```

```
 [0.00813008 1.         0.01612903]
 [0.01219512 1.         0.02409639]
 [0.01219512 1.         0.02409639]
 [0.01219512 1.         0.02409639]
 [0.01219512 1.         0.02409639]
```

```
 [0.1300813  1.         0.23021583]
 [0.1300813  1.         0.23021583]
 [0.1300813  1.         0.23021583]
 [0.1300813  1.         0.23021583]
 [0.1300813  1.         0.23021583]
 [0.1300813  1.         0.23021583]
 [0.13414634 1.         0.23655914]
 [0.15853659 1.         0.27368421]
 [0.17479675 1.         0.29757785]
 [0.17479675 1.         0.29757785]
 [0.17479675 1.         0.29757785]
 [0.17479675 1.         0.29757785]
 [0.17479675 1.         0.29757785]
 [0.17479675 1.         0.29757785]
 [0.17479675 1.         0.29757785]
 [0.17479675 1.         0.29757785]
 [0.17479675 1.         0.29757785]
 [0.17479675 1.         0.29757785]
 [0.17479675 1.         0.29757785]
```

```
print(maxIndex)
print(bestThreshold)
```

```
193
-253.2360071762041
```

```
np.mean(train.iloc[:,1])
```

```
0.005246755420062154
```

```
mu[1]
```

```
0.005246755420062154
```

```
print(mu)
# print(sigma)

```

```
Time      11.252384
V1         0.005247
V2        -0.005416
V3         0.010293
V4        -0.008144
V5         0.004281
V6         0.001813
V7         0.010354
V8        -0.001103
V9         0.006351
V10        0.009573
V11       -0.007736
V12        0.009943
V13        0.001084
V14        0.010816
V15        0.001082
V16        0.007216
V17        0.012364
V18        0.003412
V19       -0.001811
V20       -0.001092
V21       -0.001302
V22       -0.000354
V23        0.000209
V24        0.000288
V25        0.000375
V26        0.000457
V27       -0.000509
V28       -0.000119
Amount     3.152259
dtype: float64
```

```
#performance on test data
#prediction on test data
y_test_pred_raw = model.logpdf(testFeatures)
y_pred_test = y_test_pred_raw < bestThreshold

f1_score(testLabel, y_pred_test, average = "binary")
```

```
0.7401574803149606
```

```
y_pred_test
```

```
array([False, False, False, ...,  True, False,  True])
```

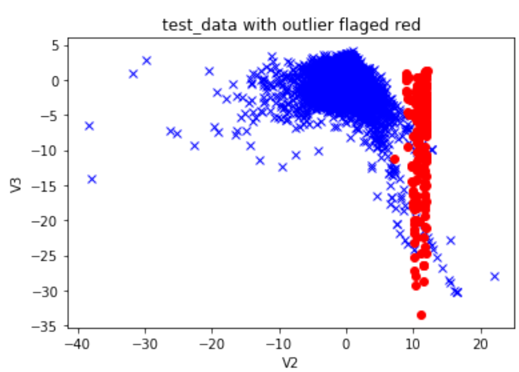
```
#index of predicted outliers in test data
predoutliersTest = np.asarray(np.where(y_pred_test))

len(predoutliersTest[0])
```

```
262
```

```
predoutliersTest #indexes
```

```
array([[  248,   437,   605,  1007,  1353,  1451,  1462,  1546,  1988,
         2461,  3674,  3928,  4216,  4928,  5144,  5846,  5975,  6022,
         6682,  6706,  6858,  7017,  7138,  8267,  8452,  8611,  8677,
         8936,  8996,  9207,  9443,  9807,  9988, 10263, 10391, 10657,
        11224, 12205, 13539, 13935, 14050, 14573, 14579, 14802, 14869,
        15740, 16061, 16888, 17322, 17663, 19352, 19902, 20680, 20800,
        21748, 22366, 22552, 22859, 23217, 23456, 23742, 24639, 24819,
        25654, 25678, 26035, 27282, 27293, 27314, 27587, 27723, 28117,
        28178, 28396, 28432, 28433, 28434, 28435, 28436, 28437, 28438,
        28440, 28443, 28444, 28445, 28446, 28447, 28449, 28450, 28453,
        28454, 28455, 28456, 28457, 28458, 28459, 28460, 28461, 28462,
        28463, 28464, 28465, 28466, 28467, 28468, 28469, 28470, 28471,
        28472, 28473, 28475, 28479, 28480, 28481, 28482, 28483, 28484,
        28486, 28487, 28490, 28492, 28493, 28494, 28496, 28497, 28498,
        28499, 28500, 28501, 28502, 28503, 28505, 28506, 28507, 28508,
        28510, 28511, 28512, 28513, 28517, 28521, 28523, 28525, 28526,
        28527, 28528, 28529, 28530, 28531, 28532, 28536, 28538, 28539,
        28540, 28542, 28543, 28544, 28546, 28547, 28549, 28550, 28551,
        28552, 28553, 28554, 28555, 28556, 28558, 28559, 28560, 28561,
        28562, 28564, 28565, 28566, 28567, 28568, 28570, 28572, 28574,
        28575, 28576, 28577, 28578, 28579, 28580, 28581, 28583, 28584,
        28585, 28586, 28588, 28589, 28591, 28592, 28594, 28596, 28598,
        28599, 28600, 28601, 28602, 28603, 28604, 28605, 28606, 28607,
        28609, 28610, 28612, 28615, 28617, 28618, 28619, 28620, 28621,
        28622, 28623, 28625, 28626, 28628, 28629, 28630, 28631, 28632,
        28633, 28636, 28637, 28638, 28639, 28640, 28641, 28642, 28643,
        28645, 28646, 28647, 28648, 28649, 28650, 28651, 28652, 28653,
        28654, 28656, 28657, 28658, 28659, 28660, 28661, 28662, 28663,
        28664, 28666, 28669, 28670, 28671, 28672, 28673, 28674, 28675,
        28677]])
```

```
#outliers identified on test data feature column V2 V3
plt.figure()
plt.title("test_data with outlier flaged red")
plt.xlabel("V2")
plt.ylabel("V3")
plt.plot(testFeatures.iloc[:, 2],testFeatures.iloc[:,3],"bx")
plt.plot(testFeatures.iloc[predoutliersTest[0],1],testFeatures.iloc[predoutliersTest[0],2],"ro")
plt.show()
```




```
# generate evaluation metrics
print("%s: %r" % ("accuracy_score is: ", accuracy_score(testLabel, y_pred_test)))
print("%s: %r" % ("roc_auc_score is: ", roc_auc_score(testLabel, y_test_pred_raw)))#correction: should be y_pred_test instead of y_test_p
print("%s: %r" % ("f1_score is: ", f1_score(testLabel, y_pred_test )))#string to int

print ("confusion_matrix is: ")
cm = confusion_matrix(testLabel, y_pred_test)
cmDF = pd.DataFrame(cm, columns=['pred_0', 'pred_1'], index=['true_0', 'true_1'])
print(cmDF)
```

```
print('recall =',float(cm[1,1])/(cm[1,0]+cm[1,1]))
print('precision =', float(cm[1,1])/(cm[1,1] + cm[0,1]))#1.0


```

```
accuracy_score is: : 0.9953971685612665
roc_auc_score is: : 0.03898289914947546
f1_score is: : 0.7401574803149606
confusion_matrix is:
        pred_0  pred_1
true_0   28358      74
true_1      58     188
recall = 0.7642276422764228
precision = 0.7175572519083969
```

```
# convert 1/-1 to 0/1 for f1 calculation
def convert(x):
  if x == 1:
    return 0
  else:
    return 1
pred = IFModel.predict(testFeatures) #1 for inliers, -1 for outliers.
pred2 = list(map(convert, pred))
# pred2
import collections

counter=collections.Counter(pred2)
print(counter)#

f1_score(testLabel, pred2, average = "binary")#0.48 when added parameters
```

```
Counter({0: 28288, 1: 390})
0.48113207547169806
```