```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.ensemble import IsolationForest
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_sco
```

```
creditcardDF=pd.read_csv('/content/creditcard (1).csv')
creditcardDF.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	–1.359807	–0.072781	2.536347	1.378155	–0.338321	0.462388	0.239599	0.(
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	–0.082361	–0.078803	0.
2	1.0	–1.358354	–1.340163	1.773209	0.379780	–0.503198	1.800499	0.791461	0.
3	1.0	–0.966272	–0.185226	1.792993	–0.863291	–0.010309	1.247203	0.237609	0.
4	2.0	–1.158233	0.877737	1.548718	0.403034	–0.407193	0.095921	0.592941	–0.

5 rows × 31 columns

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

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

```
creditcardDF.shape
```

```
(284807, 31)
```

```
creditcardDF.isna().sum()
```

```
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'])
```

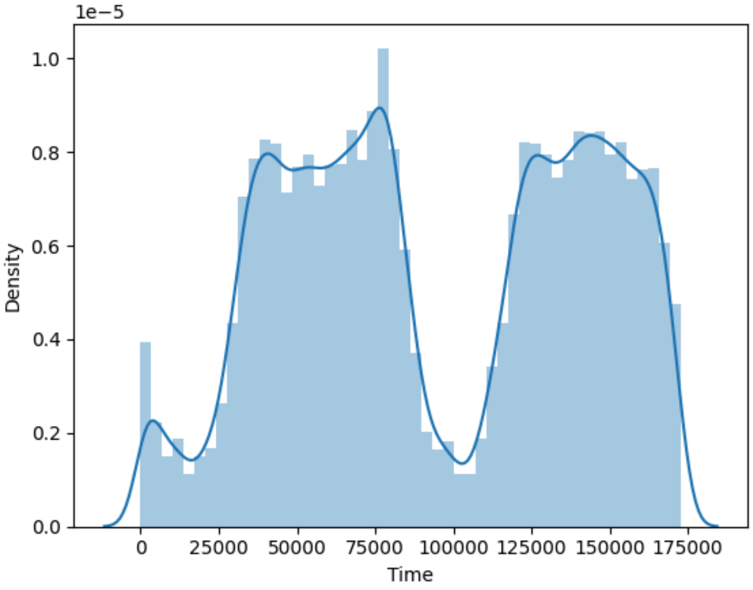
```
<ipython-input-8-16b1e7f6f4a2>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(creditcardDF['Time'])
<Axes: xlabel='Time', ylabel='Density'>
```



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

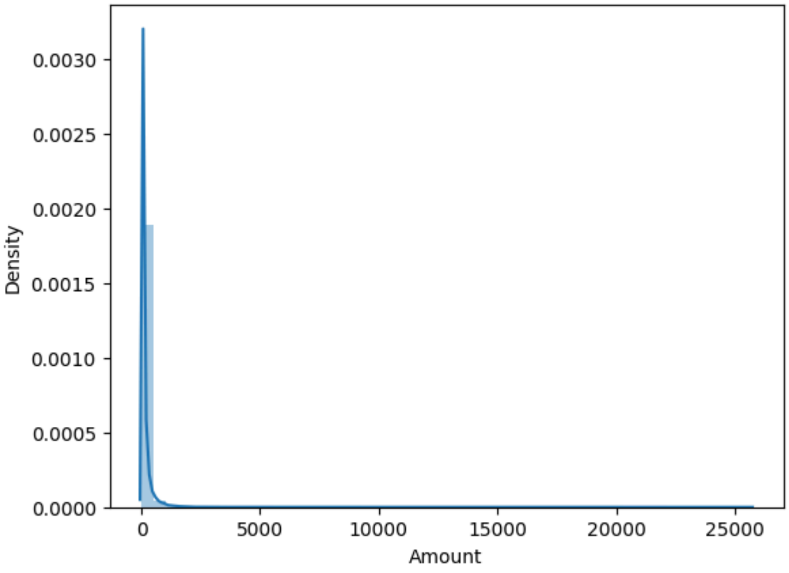
```
<ipython-input-9-8bcfe78dec34>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(creditcardDF['Amount'])
<Axes: xlabel='Amount', ylabel='Density'>
```



```
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)
anomaly_valid, anomaly_test = train_test_split(anomaly, test_size=0.5, random_state=0)
validation = pd.concat([normal_valid,anomaly_valid])
test = pd.concat([normal_test,anomaly_test])
print(validation.shape)
print(test.shape)
train = train.drop(columns = ['Class']).reset_index(drop=True)
print(train.shape)
```

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

```
featureNames = list(train.columns.values)
valFeatures = validation[featureNames].reset_index(drop= True)
testFeatures = test[featureNames].reset_index(drop= True)

valLabel = validation['Class']
testLabel = test['Class']
```

```
valFeatures.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7
0	0.248141	−0.248023	1.259502	−0.993999	−1.587788	1.913462	−0.630270	1.958852
1	0.247093	−1.614505	−0.970137	1.730517	−1.715497	−0.869271	−0.171355	1.216768
2	0.247657	1.106176	0.148096	0.424489	1.282916	−0.080275	0.146526	−0.007108
3	0.247637	−1.791995	1.102738	0.324217	1.082267	−0.303348	−1.050303	0.066270
4	0.248265	1.924286	0.324362	−0.734639	3.370481	0.783552	1.224944	−0.298881

5 rows × 30 columns

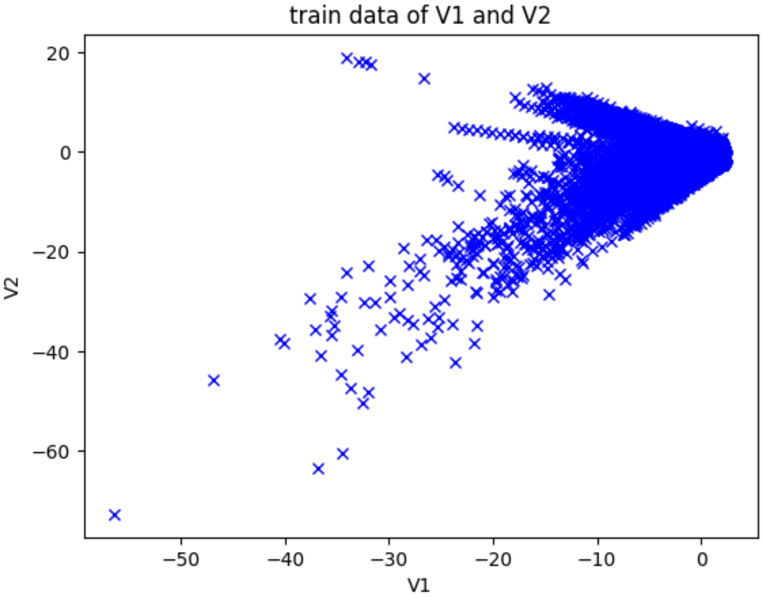
```
validation['Class'].value_counts()
```

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

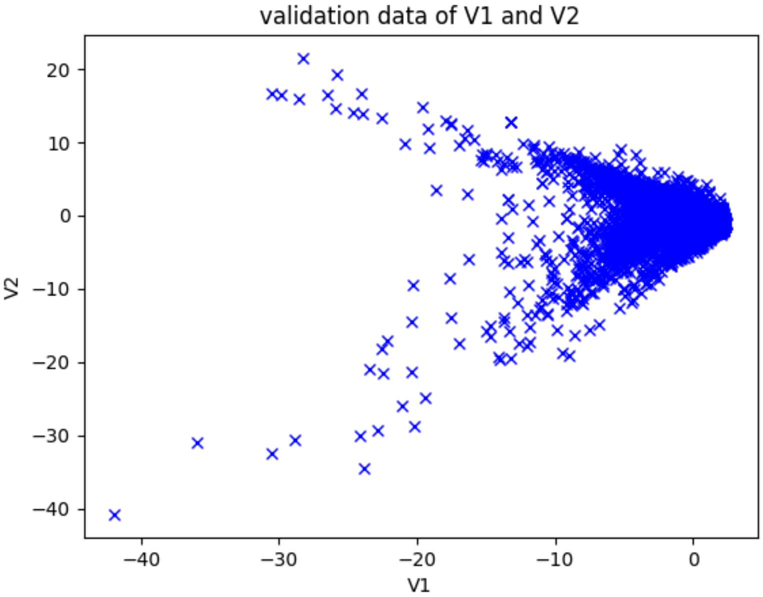
```
test['Class'].value_counts()
```

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

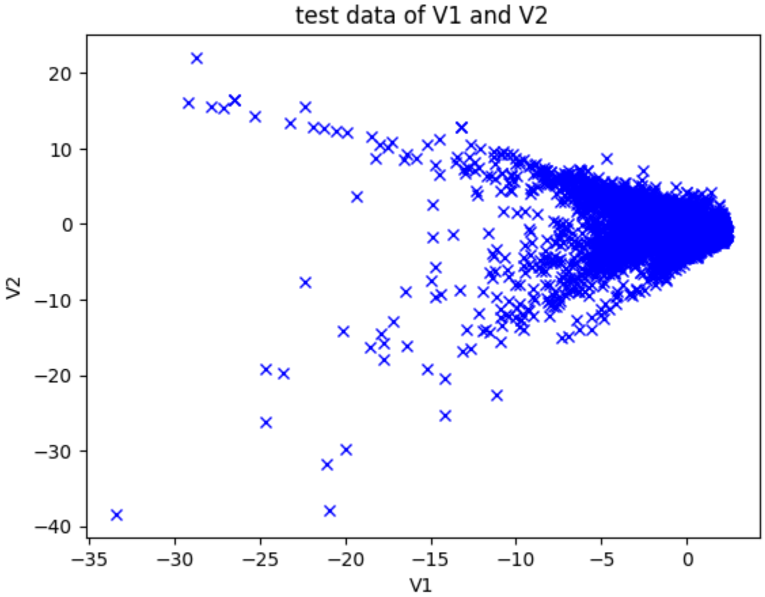
```
plt.figure()
plt.title("train data of V1 and V2")
plt.xlabel("V1")
plt.ylabel("V2")
plt.plot(train.iloc[:, 1],train.iloc[:,2],"bx")
plt.show()
```



```
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])
```

```
def estimateGaussian(dataset):
  mu = np.mean(dataset,axis = 0)
  sigma = np.cov(dataset.T)
  return mu,sigma
```

```
np.linspace(1,21,20,endpoint = False)
```

```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12., 13.,
       14., 15., 16., 17., 18., 19., 20.])
```

```
mu,sigma = estimateGaussian(train)
model = multivariate_normal(mean = mu,cov = sigma,allow_singular = True)
```

```
pdfVal = model.pdf(valFeatures)
print(max(pdfVal))
print(min(pdfVal))

p_val = model.logpdf(valFeatures)
print(max(p_val))
print(min(p_val))
```

```
1.4942008111133856e-07
0.0
-15.71650416155766
-7564.402462163995
```

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

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

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

```
[  -20.57568532   -24.01775294   -17.49568515 ... -5162.50816388
 -4533.73599154   -18.97091734]
[False False False ...  True  True False]
```

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

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

```
scores = []
p_val = model.logpdf(valFeatures)
thresholds = np.linspace(min(p_val),max(p_val),200)

for threshold in thresholds:
  y_pred = (p_val<threshold).astype(int)
  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/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, msg_start, len(result))
(200, 3)
```

```
print(scores)
```

```
 [0.84146341 0.33823529 0.48251748]
 [0.86585366 0.17530864 0.29158111]
 [1.         0.0085786  0.01701127]]
```

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

```
193
-243.31507575962132
```

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

```
0.00524675542006222
```

```
mu[1]
```

```
0.00524675542006222
```

```
print(mu)
```

```
Time      0.247504
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    0.196313
dtype: float64
```

```
y_test_pred_raw = model.logpdf(testFeatures)
y_pred_test = y_test_pred_raw < bestThreshold

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

```
0.7445544554455445
```

```
y_pred_test
```

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

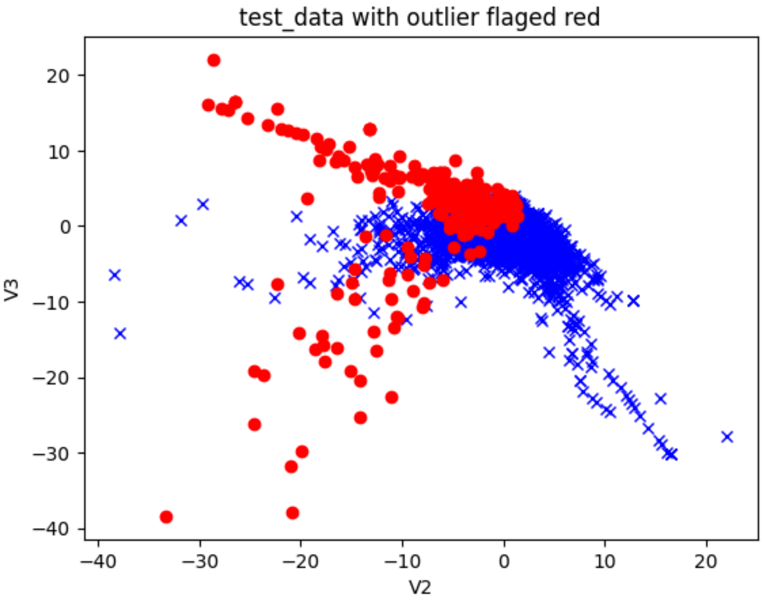
```
predoutliersTest = np.asarray(np.where(y_pred_test))
len(predoutliersTest[0])
```

```
259
```

```
predoutliersTest
```

```
array([[  248,   437,   605,  1007,  1353,  1451,  1462,  1546,  1988,
         2461,  3674,  3928,  4216,  4928,  5144,  5846,  5975,  6022,
         6682,  6706,  6858,  7017,  7138,  8452,  8611,  8677,  8936,
         8996,  9207,  9443,  9807,  9988, 10263, 10391, 10657, 11224,
        12205, 13539, 13935, 14050, 14573, 14579, 14869, 15740, 16061,
        16888, 17322, 17663, 19352, 19902, 20680, 20800, 21748, 22366,
        22552, 22859, 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]])
```

```
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()
```



```
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
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: : 0.9955017783666923
    roc_auc_score is: : 0.039987578407016486
    f1_score is: : 0.7445544554455445
    confusion_matrix is: 
            pred_0  pred_1
    true_0   28361      71
    true_1      58     188
    recall = 0.7642276422764228
    precision = 0.7258687258687259
```

```
IFModel = IsolationForest(random_state=0, contamination = 0.01, n_estimators = 200, max_features = 2).fit(train)
IFModel# (isolate normal and inormal)
```

```
    /usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but Isolat
      warnings.warn(
```

```
▾                  IsolationForest
IsolationForest(contamination=0.01, max_features=2, n_estimators=200,
                random_state=0)
```

```
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)#
```