Exp 5: Simple Anomaly Detection Example

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.neighbors import LocalOutlierFactor
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

```
df = pd.read_csv('creditcard.csv')
```

```
df.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	0.09
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27

5 rows × 31 columns

```
df.shape
```

```
(284807, 31)
```

```
df.describe()
```

```
                Time            V1            V2            V3            V4
```

```
data = df.sample(frac=0.3, random_state=1)
```

```
       mean    94813.859575    3.918649e-15    5.682686e-16    -8.761736e-15    2.811118e-15    -1.
```

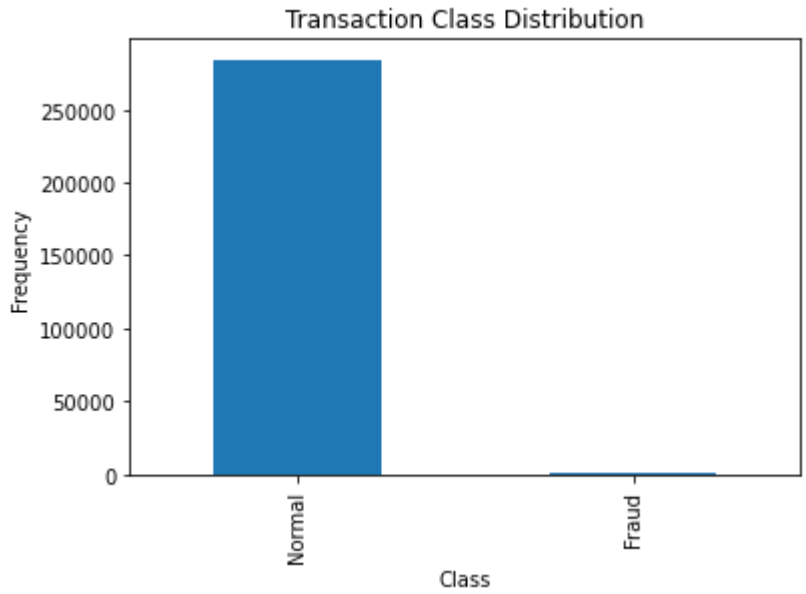
```
data.shape
```

```
(85442, 31)
```

```
numClasses = pd.value_counts(df['Class'], sort=True)
```

```
# Plot the Number of Transactions per class
numClasses.plot(kind='bar')
plt.title('Transaction Class Distribution')
plt.xticks(range(2), ['Normal', 'Fraud'])
plt.xlabel('Class')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```




```
fraud = df[df['Class']==1]
normal = df[df['Class']==0]
print(fraud.shape, normal.shape)
```

```
(492, 31) (284315, 31)
```

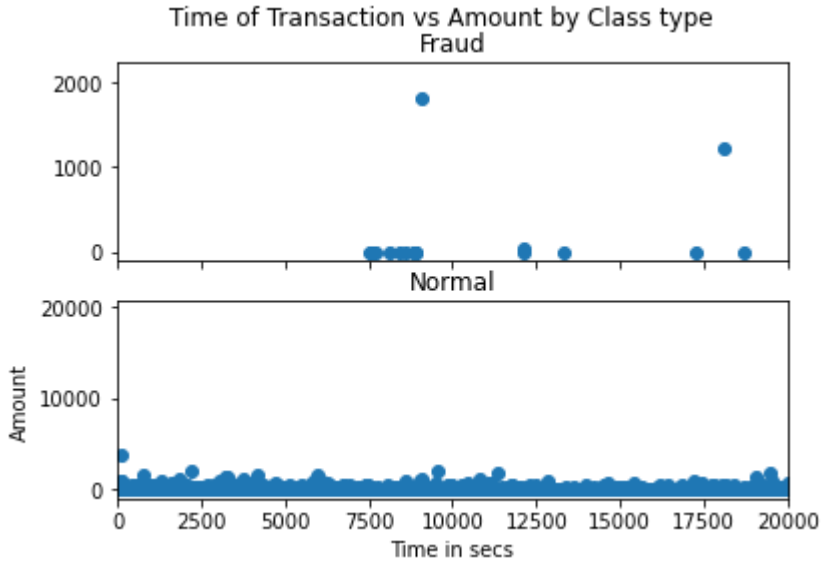
```
fraudFraction = data[data['Class']==1]
normalFraction = data[data['Class']==0]
print(fraudFraction.shape, normalFraction.shape)
```

```
(135, 31) (85307, 31)
```

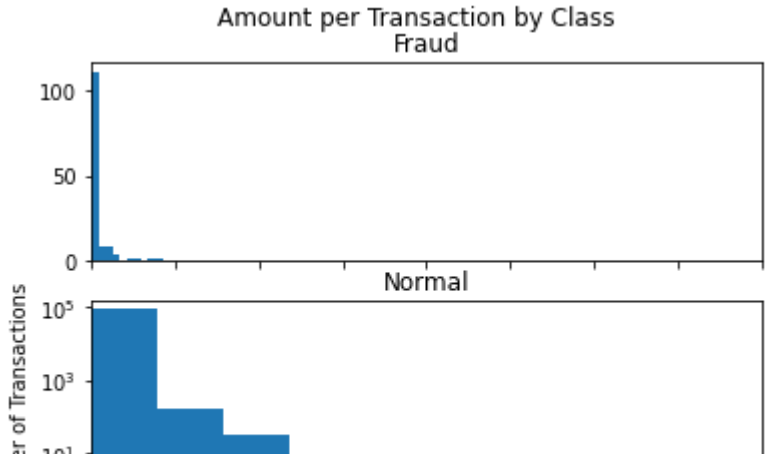
```
# Explore the Dataset and try to check whether any feature can convey...
# ...any information which can be used to classify better
```

```
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of Transaction vs Amount by Class type')

# First Plot
ax1.scatter(fraudFraction.Time, fraudFraction.Amount)
ax1.set_title('Fraud')
plt.ylabel('Amount')

# Second Plot
ax2.scatter(normalFraction.Time, normalFraction.Amount)
ax2.set_title('Normal')
plt.xlabel('Time in secs')
plt.ylabel('Amount')
plt.xlim((0, 20000))
```

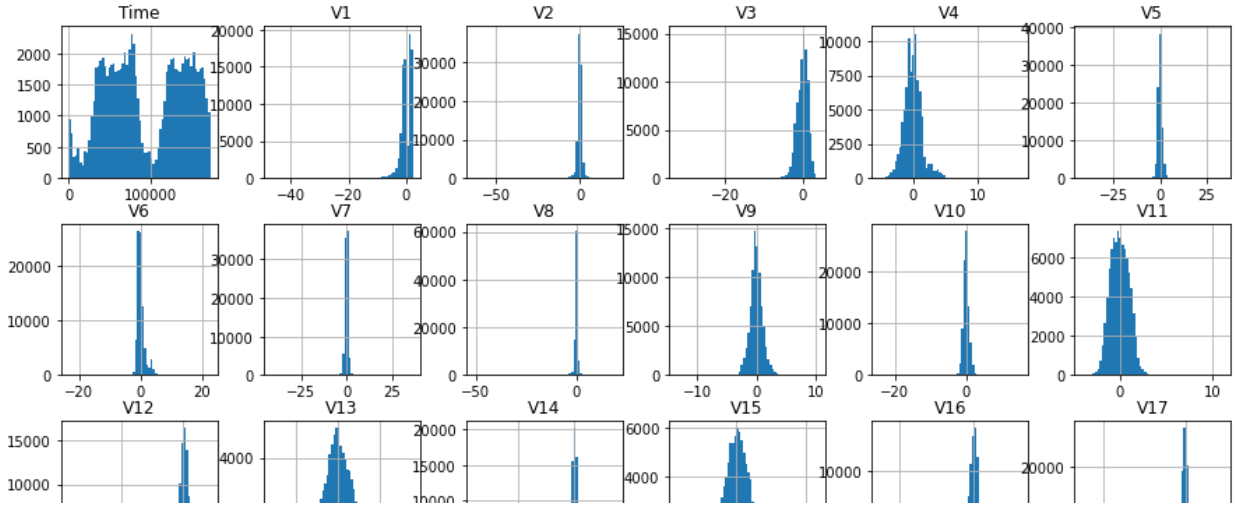
```
(0.0, 20000.0)
```




```
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Amount per Transaction by Class')
bins = 10


# First Plot
ax1.hist(fraudFraction.Amount, bins=bins)
ax1.set_title('Fraud')
plt.ylabel('Amount')

# Second Plot
ax2.hist(normalFraction.Amount, bins=bins)
ax2.set_title('Normal')
plt.xlabel('Amount($)')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))
plt.yscale('log')
```




```
# Detect the Anomaly each feature should be normally distributed...
# ...so that we can apply Unsupervised Anomaly Detection Algorithm


# Plot the Histogram of each Feature


data.hist(figsize=(15, 15), bins=64)
plt.show()
```




```
features = data.columns.to_list()
target = features[-1]
```

```
features
```

```
['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']
```

```
target
```

```
'Class'
```

```
# Split the Dataset into Train and Test
```

```
data.shape
data.shape[0] * 0.8
xTrain = data.iloc[: 68400, 1: -1]
xTrain.shape

yTrain = data.iloc[: 68400, -1]
yTrain.shape

xTest = data.iloc[68400: , 1: -1]
xTest.shape

yTest = data.iloc[68400: , -1]
yTest.shape
```

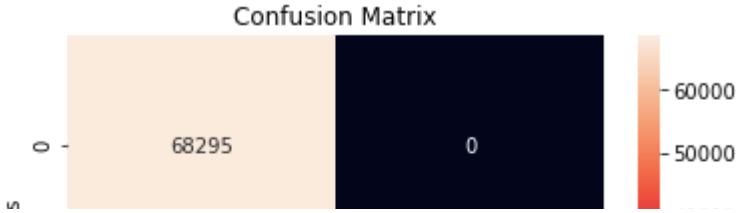
```
(17042,)
```

```
xTrain.shape
```

```
(68400, 29)
```

```
# Model
anomalyFraction = len(fraudFraction) / float(len(fraudFraction) + len(normalFractio

model = LocalOutlierFactor(contamination=anomalyFraction)


yTrainPred = model.fit_predict(xTrain)


yTrainPred[yTrainPred == 1] = 0
yTrainPred[yTrainPred == -1] = 1

yTestPred = model.fit_predict(xTest)
yTestPred[yTestPred == 1] = 0
yTestPred[yTestPred == -1] = 1


import seaborn as sns
cmTrain = confusion_matrix(yTrain, yTrainPred)
ax = plt.subplot()
sns.heatmap(cmTrain, annot=True, fmt='g', ax=ax)
ax.set_xlabel('Predicted Labels')
ax.set_ylabel('True Labels')
ax.set_title('Confusion Matrix')
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```




```
cmTrain = confusion_matrix(yTest, yTestPred)
ax = plt.subplot()
sns.heatmap(cmTrain, annot=True, fmt='g', ax=ax)
ax.set_xlabel('Predicted Labels')
ax.set_ylabel('True Labels')
ax.set_title('Confusion Matrix')
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```