Apriori Analysis

install libs

```
pip install apyori
```

```
Requirement already satisfied: apyori in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (1.1.2)
Note: you may need to restart the kernel to use updated packages.
```

import libs

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

import dataset

```
# from google.colab import files
# uploaded = files.upload()
# D14data1.csv

import os
os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
os.getcwd()
```

```
'C:\\Users\\surya\\Downloads\\PG-DBDA-Mar23\\Datasets'
```

```
dataset = pd.read_csv('D14data3.csv',header=None)
dataset.head()
```

	0	1	2	3	4	5	6	7	8	9	10
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
dataset.shape
```

```
(7501, 20)
```

4	mineral	milk	energy	whole	green	NaN	NaN	NaN	NaN	NaN	NaN

generate transactions

```
# converting dataframe into list of lists
transaction = []
for i in range(1, 7501):
    transaction.append([str(dataset.values[i, j]) for j in range(0, 20)])
```

```
# transaction[:3] #debug
```

Apriori Rules

```
from apyori import apriori
```

```
rules = apriori(transaction, min_support=0.0045, min_confidence=0.2, min_lift=3, min_length=2)
# defining MinSup, MinConf, MinLift and MinLength using transaction data to generate rules
rules
```

```
<generator object apriori at 0x000001F089928820>
```

```
# converting rules into list
results = list(rules)
results[:2]
```

```
[RelationRecord(items=frozenset({'light cream', 'chicken'}), support=0.004533333333333334, ordered_statistics=[OrderedStatistic(items_base=frozenset({'light
cream'}), items_add=frozenset({'chicken'}), confidence=0.2905982905982906, lift=4.843304843304844)]),
 RelationRecord(items=frozenset({'escalope', 'mushroom cream sauce'}), support=0.005733333333333333, ordered_statistics=
[OrderedStatistic(items_base=frozenset({'mushroom cream sauce'}), items_add=frozenset({'escalope'}), confidence=0.30069930069930073, lift=3.7903273197390845)])]
```

Rules' Combinations

```
for i in range(0, len(results)):
    print(results[i][0])
# printing combinations from rules
```

```
frozenset({'light cream', 'chicken'})
frozenset({'escalope', 'mushroom cream sauce'})
frozenset({'escalope', 'pasta'})
frozenset({'ground beef', 'herb & pepper'})
frozenset({'tomato sauce', 'ground beef'})
frozenset({'olive oil', 'whole wheat pasta'})
frozenset({'shrimp', 'pasta'})
frozenset({'nan', 'light cream', 'chicken'})
frozenset({'chocolate', 'shrimp', 'frozen vegetables'})
frozenset({'ground beef', 'spaghetti', 'cooking oil'})
frozenset({'nan', 'escalope', 'mushroom cream sauce'})
frozenset({'nan', 'escalope', 'pasta'})
frozenset({'frozen vegetables', 'ground beef', 'spaghetti'})
frozenset({'frozen vegetables', 'olive oil', 'milk'})
frozenset({'frozen vegetables', 'shrimp', 'mineral water'})
frozenset({'frozen vegetables', 'spaghetti', 'olive oil'})
frozenset({'frozen vegetables', 'shrimp', 'spaghetti'})
frozenset({'tomatoes', 'frozen vegetables', 'spaghetti'})
frozenset({'ground beef', 'spaghetti', 'grated cheese'})
frozenset({'ground beef', 'herb & pepper', 'mineral water'})
frozenset({'nan', 'ground beef', 'herb & pepper'})
frozenset({'spaghetti', 'ground beef', 'herb & pepper'})
frozenset({'ground beef', 'olive oil', 'milk'})
frozenset({'tomato sauce', 'nan', 'ground beef'})
frozenset({'ground beef', 'spaghetti', 'shrimp'})
frozenset({'spaghetti', 'olive oil', 'milk'})
frozenset({'olive oil', 'soup', 'mineral water'})
frozenset({'nan', 'olive oil', 'whole wheat pasta'})
frozenset({'nan', 'shrimp', 'pasta'})
frozenset({'pancakes', 'spaghetti', 'olive oil'})
frozenset({'chocolate', 'nan', 'shrimp', 'frozen vegetables'})
frozenset({'nan', 'ground beef', 'spaghetti', 'cooking oil'})
frozenset({'frozen vegetables', 'nan', 'ground beef', 'spaghetti'})
frozenset({'frozen vegetables', 'spaghetti', 'milk', 'mineral water'})
frozenset({'frozen vegetables', 'nan', 'olive oil', 'milk'})
frozenset({'frozen vegetables', 'nan', 'shrimp', 'mineral water'})
frozenset({'frozen vegetables', 'nan', 'spaghetti', 'olive oil'})
frozenset({'frozen vegetables', 'nan', 'shrimp', 'spaghetti'})
frozenset({'tomatoes', 'frozen vegetables', 'nan', 'spaghetti'})
frozenset({'nan', 'ground beef', 'spaghetti', 'grated cheese'})
frozenset({'nan', 'ground beef', 'herb & pepper', 'mineral water'})
frozenset({'spaghetti', 'nan', 'ground beef', 'herb & pepper'})
frozenset({'nan', 'ground beef', 'olive oil', 'milk'})
```

```
frozenset({'nan', 'ground beef', 'spaghetti', 'shrimp'})
frozenset({'nan', 'spaghetti', 'olive oil', 'milk'})
frozenset({'nan', 'olive oil', 'soup', 'mineral water'})
frozenset({'pancakes', 'nan', 'spaghetti', 'olive oil'})
frozenset({'milk', 'mineral water', 'spaghetti', 'frozen vegetables', 'nan'})
```

Rules, Support, Confidence and Lift ratio

```
# printing Rules, Support, Confidence and Lift ratio
for item in results:
    # first index of the inner list
    # Contains base item and add item
    pair = item[0]
    items = [x for x in pair]
    print("Rule: " + items[0] + " -> " + items[1])

    # second index of the inner list
    print("Support: " + str(item[1]))

    # third index of the list located at 0th
    # of the third index of the inner list

    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("=====================================")
```

```
Rule: light cream -> chicken
Support: 0.004533333333333334
Confidence: 0.2905982905982906
Lift: 4.843304843304844
=====================================
Rule: escalope -> mushroom cream sauce
Support: 0.005733333333333333
Confidence: 0.30069930069930073
Lift: 3.7903273197390845
=====================================
Rule: escalope -> pasta
Support: 0.005866666666666667
Confidence: 0.37288135593220345
Lift: 4.700185158809287
=====================================
Rule: ground beef -> herb & pepper
Support: 0.016
Confidence: 0.3234501347708895
Lift: 3.2915549671393096
=====================================
Rule: tomato sauce -> ground beef
```

```
Support: 0.005333333333333333
Confidence: 0.37735849056603776
Lift: 3.840147461662528
=====================================
Rule: olive oil -> whole wheat pasta
Support: 0.008
Confidence: 0.2714932126696833
Lift: 4.130221288078346
=====================================
Rule: shrimp -> pasta
Support: 0.005066666666666666
Confidence: 0.3220338983050848
Lift: 4.514493901473151
=====================================
Rule: nan -> light cream
Support: 0.004533333333333334
Confidence: 0.2905982905982906
Lift: 4.843304843304844
=====================================
Rule: chocolate -> shrimp
Support: 0.005333333333333333
Confidence: 0.23255813953488372
Lift: 3.260160834601174
=====================================
Rule: ground beef -> spaghetti
Support: 0.0048
Confidence: 0.5714285714285714
Lift: 3.281557646029315
=====================================
Rule: nan -> escalope
Support: 0.005733333333333333
Confidence: 0.30069930069930073
Lift: 3.7903273197390845
=====================================
Rule: nan -> escalope
Support: 0.005866666666666667
Confidence: 0.3728813559322034S
```

Anomaly Detection

Unsupervised Outlier Detection

- it uses two techniques
    1. `Local Outlier Factor (LOF)`
        - anamoly score of each sample
        - measures the `deviation of density` of a given sample w.r.t. to its neighbors

- it is local in that the anamoly score depends on how isolated the object is with respect to the surrounding neighborhood

2. `Isolation Forest Algorithm`
    - isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature
    - Since recursive partitioning can be represented by a tree structure, the number of splits required to isolate a sample is equivalent to the path length from the root node to the terminating node
    - this path length, averaged over a forest of such random trees, is a `measure of normality` and our `decision function`
    - Random partitioning produces noticeably shorter paths for anomalies, hence when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be `anomalies`

install libs

```
# pip install glemaitre imbalanced-learn imblearn
```

```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from imbalanced-learn) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from imbalanced-learn) (1.11.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from imbalanced-learn) (1.3.0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from imbalanced-learn) (1.3.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from imbalanced-learn) (3.2.0)
Note: you may need to restart the kernel to use updated packages.
```

```
pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from imblearn) (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from imbalanced-learn->imblearn) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from imbalanced-learn->imblearn) (1.11.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from imbalanced-learn->imblearn) (1.3
Requirement already satisfied: joblib>=1.1.1 in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from imbalanced-learn->imblearn) (1.3.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\surya\appdata\local\programs\python\python39\lib\site-packages (from imbalanced-learn->imblearn) (3.
Note: you may need to restart the kernel to use updated packages.
```

import libs

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
# from google.colab import files
# uploaded = files.upload()
# creditcard.csv

import os
os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
os.getcwd()
```

```
'C:\\Users\\surya\\Downloads\\PG-DBDA-Mar23\\Datasets'
```

```
dataset = pd.read_csv('creditcard.csv')
dataset.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

```
dataset.shape
```

```
(284807, 31)
```

```
dataset.describe()
```

	Time	V1	V2	V3	V4	V
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+0
mean	94813.859575	1.759061e-12	-8.251130e-13	-9.654937e-13	8.321385e-13	1.649999e-1
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+0
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+0
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-0
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-0

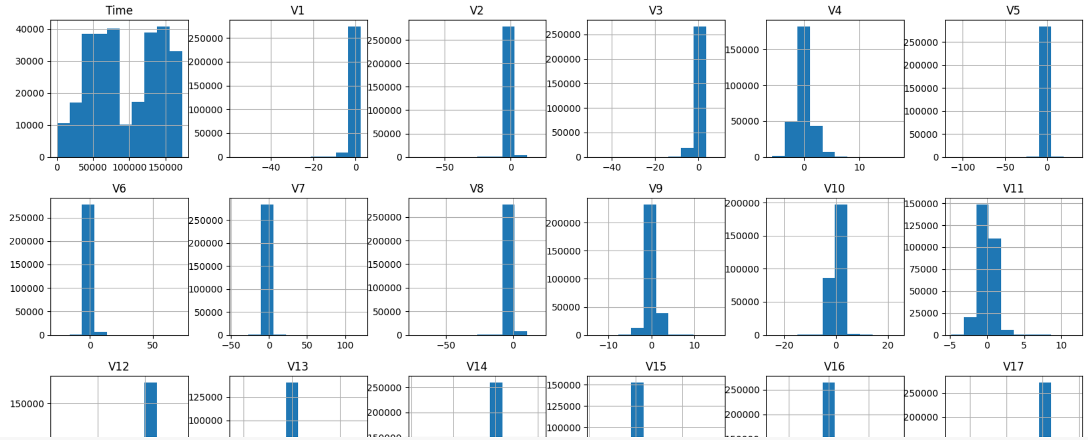
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
```

```
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

EDA

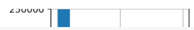
```
dataset.hist(figsize=(20, 20))
plt.show()
```




```
1 Fraud = dataset[dataset['Class'] == 1]
2 Fraud.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	
541	406.0	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.537387	1.3916
623	472.0	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.325574	-0.0677
4920	4462.0	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.562320	-0.3991
6108	6986.0	-4.397974	1.358367	-2.592844	2.679787	-1.128131	-1.706536	-3.496197	-0.2487
6329	7519.0	1.234235	3.019740	-4.304597	4.732795	3.624201	-1.357746	1.713445	-0.4963

5 rows × 31 columns



```
1 Fraud.shape
```

```
(492, 31)
```

```
1 Valid = dataset[dataset['Class'] == 0]
2 Valid.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102

```
Valid.shape
```

```
(284315, 31)
```

Outlier Fraction

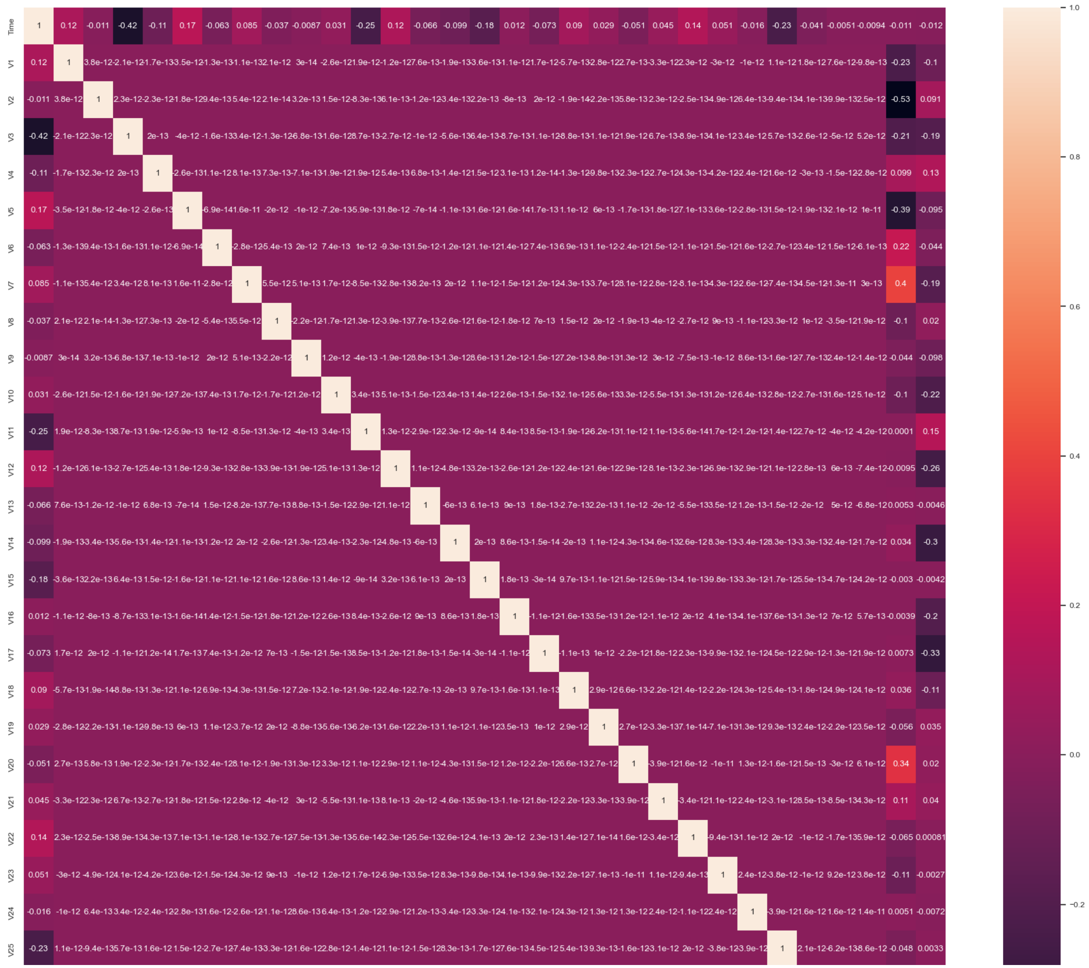
```
outlier_fraction = len(Fraud) / float(len(Valid))
outlier_fraction
```

```
0.0017304750013189597
```

Correlation

```
corrmat = dataset.corr()
fig = plt.figure(figsize=(23, 23))
sns.set(font_scale = 0.8)
sns.heatmap(corrmat, annot=True)
```

<Axes: >



identifying X & Y

```
1 x = dataset.iloc[ : , :-1].values
2 x[:2]
```

```
array([[ 0.00000000e+00, -1.35980713e+00, -7.27811730e-02,
         2.53634674e+00,  1.37815522e+00, -3.38320770e-01,
         4.62387778e-01,  2.39598554e-01,  9.86979010e-02,
         3.63786970e-01,  9.07941720e-02, -5.51599533e-01,
        -6.17800856e-01, -9.91389847e-01, -3.11169354e-01,
         1.46817697e+00, -4.70400525e-01,  2.07971242e-01,
         2.57905800e-02,  4.03992960e-01,  2.51412098e-01,
        -1.83067780e-02,  2.77837576e-01, -1.10473910e-01,
```

```
         6.69280750e-02,  1.28539358e-01, -1.89114844e-01,
         1.33558377e-01, -2.10530530e-02,  1.49620000e+02],
       [ 0.00000000e+00,  1.19185711e+00,  2.66150712e-01,
         1.66480113e-01,  4.48154078e-01,  6.00176490e-02,
        -8.23608090e-02, -7.88029830e-02,  8.51016550e-02,
        -2.55425128e-01, -1.66974414e-01,  1.61272666e+00,
         1.06523531e+00,  4.89095016e-01, -1.43772296e-01,
         6.35558093e-01,  4.63917041e-01, -1.14804663e-01,
        -1.83361270e-01, -1.45783041e-01, -6.90831350e-02,
        -2.25775248e-01, -6.38671953e-01,  1.01288021e-01,
        -3.39846476e-01,  1.67170404e-01,  1.25894532e-01,
        -8.98309900e-03,  1.47241690e-02,  2.69000000e+00]])
```

```
y = dataset.iloc[ : , -1].values
y[:2]
```

```
array([0, 0], dtype=int64)
```

```
x.shape
```

```
(284807, 30)
```

```
y.shape
```

```
(284807,)
```

splitting

```
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=0)
```

IsolationForest & LocalOutlierFactor

```
from sklearn.ensemble import IsolationForest
# Isolation Forest Algorithm

# The Isolation Forest isolates observation by randomly selecting a feature
# and then randomly selecting a split value between
# the maximum and minimum values of the selected features
```

```
from sklearn.neighbors import LocalOutlierFactor
# Unsupervised Outlier Detection using the Local Outlier Factor (LOF)
# to find neighbor
```

```
# define random states
state = 1
# define outlier detection tools to eb compared
classifier = {
    "Isolation Forest": IsolationForest(max_samples=len(x),
                                        contamination=outlier_fraction,
                                        random_state=1),
    "Local Outlier Factor": LocalOutlierFactor(n_neighbors=20,
                                               contamination=outlier_fraction)
}
```

```
n_outliers = len(Fraud)
```

```
for i, (clf_name, clf) in enumerate(classifier.items()):
    # fit the data and tag outliers
    if clf_name == "Local Outlier Factor":
        y_pred_anamoly = clf.fit_predict(x)
        scores_pred = clf.negative_outlier_factor_
    else:
        clf.fit(x)
        scores_pred = clf.decision_function(x)
        y_pred_anamoly = clf.predict(x)
```

```
# reshape the prediction values to 0 for valid, 1 for fraud
y_pred_anamoly[y_pred_anamoly == 1] = 0
y_pred_anamoly[y_pred_anamoly == -1] = 1
```

```
n_errors = (y_pred_anamoly != y).sum()
n_errors
```

```
935
```

Evaluation

accuracy_score

```
from sklearn.metrics import accuracy_score
```

```
accuracy_score(y, y_pred_anamoly)
```

```
0.9967170750718908
```

classification_report

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y, y_pred_anamoly))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    284315
           1       0.05      0.05      0.05       492

    accuracy                           1.00    284807
   macro avg       0.52      0.52      0.52    284807
weighted avg       1.00      1.00      1.00    284807
```

confusion_matrix

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(y, y_pred_anamoly)
cm
```

```
array([[283847,    468],
       [   467,     25]], dtype=int64)
```

```
TP = cm[0, 0]
TN = cm[1, 1]
FP = cm[0, 1]
FN = cm[1, 0]
print(TP)
print(TN)
print(FP)
print(FN)
```

```
283847
25
468
467
```

precision

```
precision = TP / float(TP + FP)
precision
```

```
0.9983539384133795
```

recall

```
recall = TP / float(TP + FN)
recall
```

```
0.9983574498617726
```

tpr

```
tpr = TP/float(TP+FN)
tpr
```

```
0.9983574498617726
```

fpr

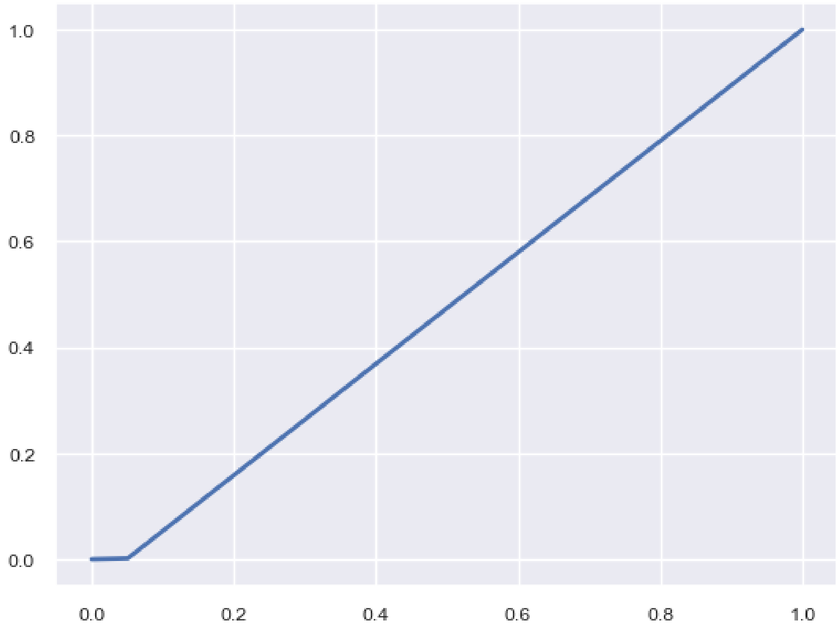
```
fpr = FP/float(FP + TN)
fpr
```

```
0.949290060851927
```

roc_curve

```
from sklearn.metrics import roc_curve
```

```
tpr, fpr, threshold = roc_curve(y, y_pred_anamoly)
plt.plot(fpr, tpr, linewidth=2)
plt.show()
```




```

```

Time Series Data Application

- set of data collected and arranged in accordance of time
- According to Croxton and Cowdon, "A time series consist of data arranged chronologically"
- used for non-stationary data, data which is constantly fluctuating over time or are affected by time
- helps to predict the future behavior of variable based on past experience
- Time Series can be decomposed into four components, each expressing a particular aspect of the movement of the values of the time series

    1. `Secular Trend`
        - describes movement along the trend
    2. `Seasonal Variations`

- represents seasonal changes

3. `Cyclical Fluctuations`
    - corresponds to periodical but not seasonal variations
4. `Irregular Variations`
    - other non-random sources of variations

- Two types of time series data
    1. `Metrics`
        - measurements gathered at regular intervals of time
    2. `Events`
        - measurements gathered at irregular intervals of time
- Three types of models for time series
    1. Moving Average (`MA`)
    2. Exponential Smoothing (`ES`)
    3. AutoRegressive Integrated Moving Average (`AR` / `ARIMA`)
- CHaracteristics of Time Series
    - Time Series Exhibits one or more of the following features
        1. Trends
        2. Seasonal Cycles
        3. Non-Seasonal Cycles
        4. Pulses and Steps
        5. Outliers

importing libs

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime
```

importing dataset

```
# from google.colab import files
# uploaded = files.upload()
# D14data1.csv

import os
os.chdir(r'C:\Users\surya\Downloads\PG-DBDA-Mar23\Datasets')
os.getcwd()
```

```
'C:\\Users\\surya\\Downloads\\PG-DBDA-Mar23\\Datasets'
```

```
dataset = pd.read_csv('D15data1.csv')
dataset.head()
```

	Date	Temp
0	01-01-1981	20.7
1	02-01-1981	17.9
2	03-01-1981	18.8
3	04-01-1981	14.6
4	05-01-1981	15.8

```
dataset.shape
```

```
(3650, 2)
```

```
dataset.describe()
```

	Temp
count	3650.000000
mean	11.177753
std	4.071837
min	0.000000
25%	8.300000
50%	11.000000
75%	14.000000
max	26.300000

```
dataset.info()
```

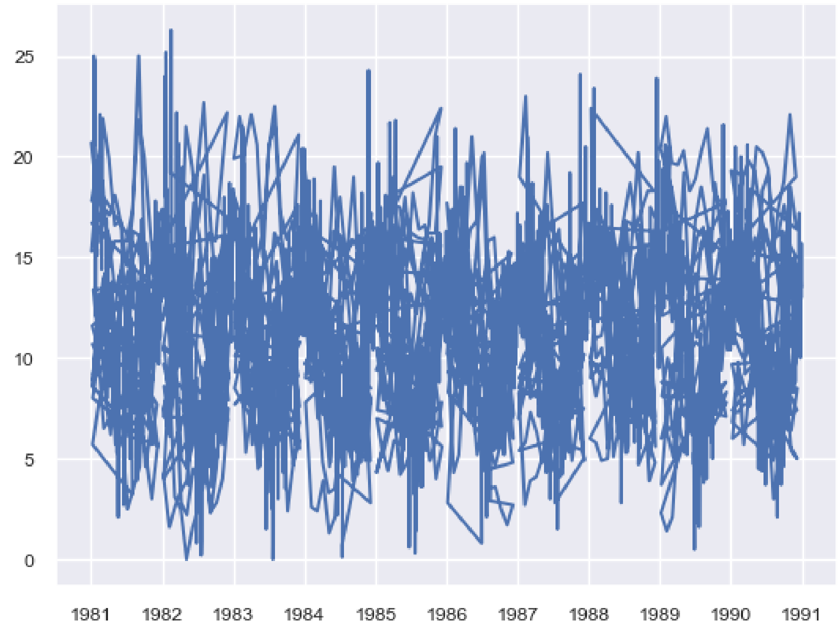
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3650 entries, 0 to 3649
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    3650 non-null   object 
 1   Temp    3650 non-null   float64
dtypes: float64(1), object(1)
memory usage: 57.2+ KB
```

converting string type to date type

```
dataset['Date'] = pd.to_datetime(dataset['Date'], format="mixed")
```

```
# date vs Temp plot
plt.plot(dataset['Date'], dataset['Temp'])
```

```
[<matplotlib.lines.Line2D at 0x1f0a8ea0790>]
```




```
dataset['M12'] = dataset['Temp'].rolling(12).mean()
```

```
dataset['Month'] = [i.month for i in dataset['Date']]
dataset.head()
```

	Date	Temp	M12	Month
0	1981-01-01	20.7	NaN	1
1	1981-02-01	17.9	NaN	2
2	1981-03-01	18.8	NaN	3
3	1981-04-01	14.6	NaN	4
4	1981-05-01	15.8	NaN	5

```
dataset['Year'] = [i.year for i in dataset['Date']]
dataset.head()
```

	Date	Temp	M12	Month	Year
0	1981-01-01	20.7	NaN	1	1981
1	1981-02-01	17.9	NaN	2	1981
2	1981-03-01	18.8	NaN	3	1981
3	1981-04-01	14.6	NaN	4	1981
4	1981-05-01	15.8	NaN	5	1981

```
dataset['Series'] = np.arange(1, len(dataset)+1)
dataset.head()
```

	Date	Temp	M12	Month	Year	Series
0	1981-01-01	20.7	NaN	1	1981	1
1	1981-02-01	17.9	NaN	2	1981	2
2	1981-03-01	18.8	NaN	3	1981	3
3	1981-04-01	14.6	NaN	4	1981	4
4	1981-05-01	15.8	NaN	5	1981	5

```
dataset.drop(['Date', 'M12'], axis=1, inplace=True)
dataset.head()
```

	Temp	Month	Year	Series
0	20.7	1	1981	1
1	17.9	2	1981	2
2	18.8	3	1981	3
3	14.6	4	1981	4
4	15.8	5	1981	5

```
dataset = dataset[['Series', 'Year', 'Month', 'Temp']]
```

Identify X & Y

```
x = dataset.iloc[ : , :-1].values
x[:5]
```

```
array([[   1, 1981,    1],
       [   2, 1981,    2],
       [   3, 1981,    3],
       [   4, 1981,    4],
       [   5, 1981,    5]], dtype=int64)
```

```
y = dataset.iloc[ : , -1].values
y[:5]
```

```
array([20.7, 17.9, 18.8, 14.6, 15.8])
```

Splitting

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=1/3, random_state=0)
```

```
x_train[:5]
```

```
array([[3562, 1990,    4],
       [1062, 1983,   11],
       [ 998, 1983,    9],
```

```
       [1556, 1985,    6],
       [2887, 1988,   11]], dtype=int64)
```

```
y_train[:5]
```

```
array([11. , 15.8,  7.6, 15.9, 11.8])
```

Modeling- Linear Regression

```
from sklearn.linear_model import LinearRegression
```

```
linreg_ts = LinearRegression()
```

Training- Linear Regression

```
linreg_ts.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

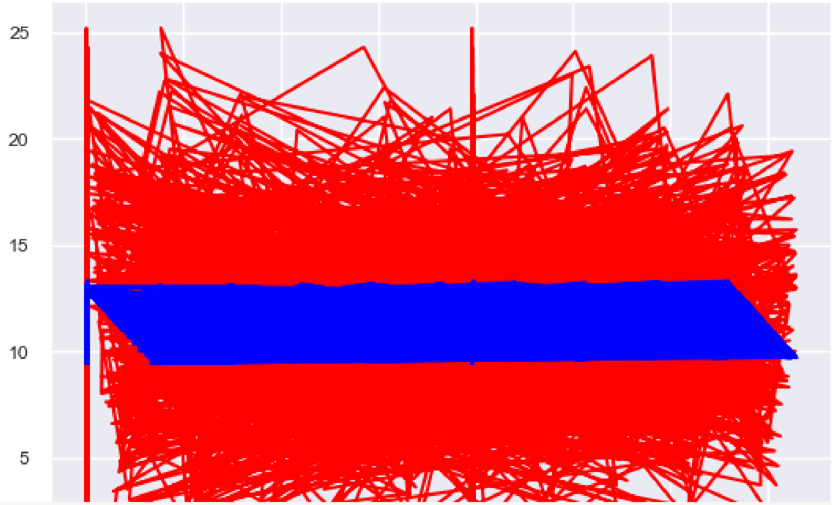
Prediction- Linear Regression

```
y_pred_ts = linreg_ts.predict(x_test)
y_pred_ts[:5]
```

```
array([10.94451163, 10.9524679 , 10.45613512, 10.45347194, 13.26754352])
```

```
plt.plot(x_train, y_train, color='red')
# Training datav plot: x_train vs y_train
plt.plot(x_train, linreg_ts.predict(x_train), color='blue')
# Training Prediction plot : x_train vs predict(y_train)
```

```
[<matplotlib.lines.Line2D at 0x1f08630bf40>,
 <matplotlib.lines.Line2D at 0x1f08630bf70>,
 <matplotlib.lines.Line2D at 0x1f08630bfa0>]
```




```
plt.plot(x_test, y_test, color='red')
# Testing Data Plot : x_test vs y_test
plt.plot(x_train, linreg_ts.predict(x_train), color='blue')
# Training Prediction plot : x_train vs predict(y_train)
# to check for overfitting
```

```
[<matplotlib.lines.Line2D at 0x1f0a976e880>,
 <matplotlib.lines.Line2D at 0x1f0a976e8b0>,
 <matplotlib.lines.Line2D at 0x1f08638f160>]
```