WEEK: 10

10. A Data Science Project –More Advanced

import pandas as pd

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

import seaborn as sns

The sns is short name use for seaborn python library. The heatmap especially uses to show 2D (two dimensional) data in graphical format.

import numpy as np

When you call the statement import numpy as np, you are shortening the phrase "numpy" to "np" to make your code easier to read. It also helps to avoid namespace issues. adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

import pandas as pd

import seaborn as sns

import numpy as np

data = pd.read_csv("C:/Users/Pradeep/Desktop/datasets/creditcard.csv")

data.head()

```
In [3]: data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601

5 rows × 31 columns

data.tail()

```
In [4]: data.tail()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797

5 rows × 31 columns

This is when Python loc() function comes into the picture. The loc() function helps us to retrieve data values from a dataset at an ease.

fraud = data.loc[data['Class'] == 1]

normal = data.loc[data['Class'] == 0]

fraud

```
In [6]: fraud
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
541	406.0	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	-2.537387	1.391657	-2.770089	...	0.517232	-0.035049	-0.465211	0.320198
623	472.0	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	0.325574	-0.067794	-0.270953	...	0.661696	0.435477	1.375966	-0.293803
4920	4462.0	-2.303350	1.759247	-0.359745	2.330243	-0.821628	-0.075788	0.562320	-0.399147	-0.238253	...	-0.294166	-0.932391	0.172726	-0.087330
6108	6986.0	-4.397974	1.358367	-2.592844	2.679787	-1.128131	-1.706536	-3.496197	-0.248778	-0.247768	...	0.573574	0.176968	-0.436207	-0.053502
6329	7519.0	1.234235	3.019740	-4.304597	4.732795	3.624201	-1.357746	1.713445	-0.496358	-1.282858	...	-0.379068	-0.704181	-0.656805	-1.632653
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	0.778584	-0.319189	0.639419	-0.294885
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.370612	0.028234	-0.145640	-0.081049
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.751826	0.834108	0.190944	0.032070
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.583276	-0.269209	-0.456108	-0.183659
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.164350	-0.295135	-0.072173	-0.450261

492 rows × 31 columns

fraud.count()

```
In [7]: fraud.count()
Out[7]: Time      492
        V1        492
        V2        492
        V3        492
        V4        492
        V5        492
        V6        492
        V7        492
        V8        492
        V9        492
        V10       492
        V11       492
        V12       492
        V13       492
        V14       492
        V15       492
        V16       492
        V17       492
        V18       492
        V19       492
        V20       492
        V21       492
        V22       492
        V23       492
        V24       492
        V25       492
        V26       492
        V27       492
        V28       492
```

fraud.sum()

```
In [8]: fraud.sum()
```

```
Out[8]: Time      3.972743e+07
        V1       -2.347799e+03
        V2        1.782899e+03
        V3       -3.460374e+03
        V4        2.234678e+03
        V5       -1.550403e+03
        V6       -6.876865e+02
        V7       -2.739816e+03
        V8        2.807529e+02
        V9       -1.269912e+03
        V10      -2.793026e+03
        V11       1.869685e+03
        V12      -3.079621e+03
        V13      -5.379224e+01
        V14      -3.430088e+03
        V15      -4.572094e+01
        V16      -2.036853e+03
        V17      -3.279592e+03
        V18      -1.105184e+03
        V19       3.348844e+02
        V20       1.831811e+02
        V21       3.510855e+02
        V22       6.912050e+00
        V23      -1.983152e+01
        V24      -5.172411e+01
        V25       2.039285e+01
        V26       2.541088e+01
```

len(fraud)

len(normal)

```
In [9]: len(fraud)
```

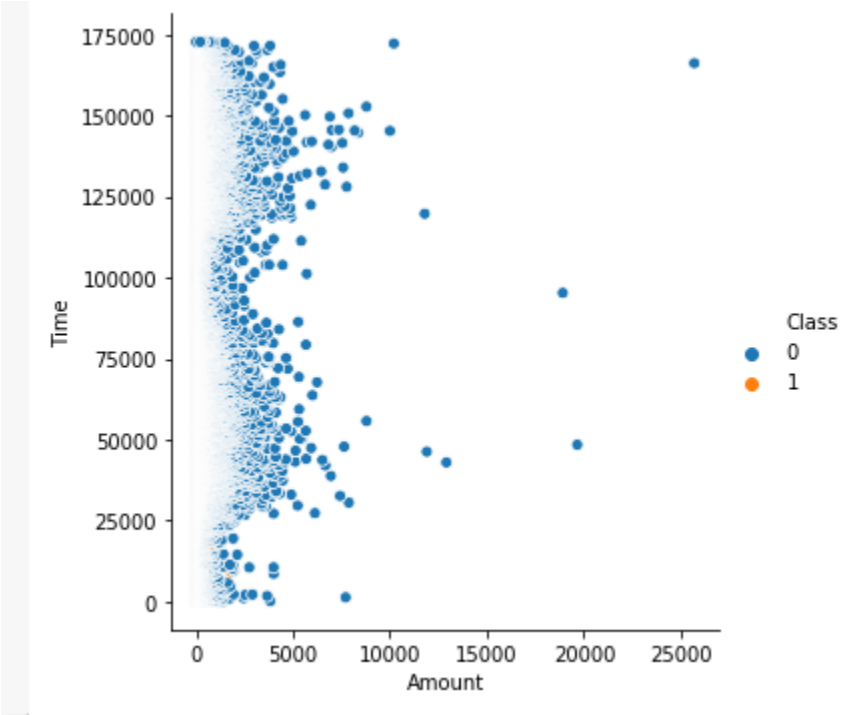
```
Out[9]: 492
```

```
In [10]: len(normal)
```

```
Out[10]: 284315
```

relplot () function has **arguments x, y and data parameters** to specify values to be plotted on XAxis, YAxis and the data it should use, respectively. We use the kind parameter to specify that it should use a scatter plot. Actually, by default, it is set to scatter.

sns.relplot(x= 'Amount',y="Time",hue="Class", data=data)



Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon **NumPy, SciPy** and **Matplotlib**.

Train-Test split

To know the performance of a model, we should test it on **unseen data**. For that purpose, we partition dataset into training set (around 70 to 90% of the data) and test set (10 to 30%). In sklearn, we use train_test_split function from sklearn.model_selection.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=1)
```

- stratify option tells sklearn to split the dataset into test and training set in such a fashion that the ratio of class labels in the variable specified (y in this case) is constant. If there 40% 'yes'

and 60% 'no' in y, then in both y_train and y_test, this ratio will be same. This is helpful in achieving fair split when data is imbalanced.

- test_size option helps to determine the size of test set (0.2=20%)
- Further there is shuffle option (by default shuffle=True) which shuffles the data before splitting.

from sklearn import linear_model

from sklearn.model_selection import train_test_split

Python iloc() function enables us to select a particular cell of the dataset, that is, it helps us select a value that belongs to a particular row or column from a set of values of a data frame or dataset.

X = data.iloc[:,:-1]

Y = data['Class']

X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size= 0.35)

clf () in Matplotlib Python clf () is a **method that is in pyplot module** which is in matplotlib library. clf () method is used to clear the entire plot and figure.it also clears subplots. it leaves empty space for re-usage of other plots.

clf = linear_model.LogisticRegression(C=1e5)

clf.fit(X_train, Y_train)

```
In [8]: clf.fit(X_train, Y_train)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will b
e changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)

Out[8]: LogisticRegression(C=100000.0, class_weight=None, dual=False,
                           fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                           max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                           random_state=None, solver='warn', tol=0.0001, verbose=0,
                           warm_start=False)
```

Y_pred = clf.predict(X_test)

Y =np.array(Y_test)

confusion_matrix

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one (in unsupervised learning it is usually called a matching matrix). Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class, or vice versa

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

print(accuracy_score(Y_test, Y_pred))

print(confusion_matrix(Y_test, Y_pred))

print(classification_report(Y_test, Y_pred))

```
In [14]: print(accuracy_score(Y_test, Y_pred))
         0.998876438309441

In [15]: print(confusion_matrix(Y_test, Y_pred))
         [[99465    45]
          [   67   106]]

In [18]: print(classification_report(Y_test, Y_pred))
                       precision    recall  f1-score   support

                    0       1.00      1.00      1.00     99510
                    1       0.70      0.61      0.65       173

             accuracy                           1.00     99683
            macro avg       0.85      0.81      0.83     99683
         weighted avg       1.00      1.00      1.00     99683
```

```
In [10]: from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

In [14]: print(accuracy_score(Y_test, Y_pred))
         0.998876438309441

In [15]: print(confusion_matrix(Y_test, Y_pred))
         [[99465    45]
          [   67   106]]

In [18]: print(classification_report(Y_test, Y_pred))
                       precision    recall  f1-score   support

                    0       1.00      1.00      1.00     99510
                    1       0.70      0.61      0.65       173

             accuracy                           1.00     99683
            macro avg       0.85      0.81      0.83     99683
         weighted avg       1.00      1.00      1.00     99683
```