Document

```
import numpy as np
```

- NumPy is a fundamental package for scientific computing with Python. It provides support for arrays, matrices, and many mathematical functions to operate on these data structures.
- np is a commonly used alias for NumPy, allowing for convenient access to its functions and methods.

```
import pandas as pd
```

- Pandas is a library used for data manipulation and analysis. It provides data structures like Series (one-dimensional) and DataFrame (two-dimensional) that are powerful and flexible for data handling.
- pd is the common alias used to import Pandas, making it easier to use its functions and methods.

```
import matplotlib.pyplot as plt
```

- Matplotlib is a plotting library for creating static, animated, and interactive visualizations in Python.
- pyplot is a module in Matplotlib that provides a MATLAB-like interface for plotting. plt is the alias commonly used to access this module.

```
import seaborn as sns
```

- Seaborn is a data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- sns is the alias used to import Seaborn, making it easy to access its functions and methods.

```
import sys
```

- The sys module provides access to some variables used or maintained by the Python interpreter and to functions that interact strongly with the interpreter.
- This can be useful for controlling the runtime environment, such as handling command-line arguments or exiting the program.

```
import scipy
```

- SciPy is a library used for scientific and technical computing. It builds on NumPy and provides a large number of higher-level functions that are useful for optimization, integration, interpolation, eigenvalue problems, algebraic equations, and more.
- SciPy does not have a standard alias, and is often used directly as scipy.

These imports collectively provide a robust toolkit for data science and analytics, enabling tasks ranging from numerical computation and data manipulation to sophisticated data visualization and scientific computing.

```
data = pd.read_csv('/content/creditcard.csv')
```

Load the dataset using pandas

```
print(data.columns)
```

Dataset exploring

```
data = data.sample(frac=0.1, random_state = 1)
print(data.shape)
print(data.describe())
```

Print the shape of the data

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

Plot histograms of each parameter

```
Fraud = data[data['Class'] == 1]
Valid = data[data['Class'] == 0]
```

Determine number of fraud cases in dataset

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

- len(Fraud)

This function call returns the number of elements (i.e., the length) in the Fraud dataset or list. In the context of data analysis, this might be a subset of data identified as fraudulent transactions.

- len(Valid)

Similarly, this returns the number of elements (i.e., the length) in the Valid dataset or list. This might represent a subset of data identified as valid (non-fraudulent) transactions.

- float(len(Valid))

The float() function converts the integer value returned by len(Valid) to a floating-point number. This ensures that the division operation will be performed with floating-point precision, which is important for getting an accurate decimal result.

- len(Fraud) / float(len(Valid))

This expression divides the number of fraudulent transactions by the number of valid transactions, yielding the ratio of fraudulent transactions to valid transactions.

- outlier_fraction

This variable stores the result of the division, representing the fraction of outliers (fraudulent transactions) relative to the valid transactions.

- print(outlier_fraction)

This prints the value of outlier_fraction to the console, allowing you to see the calculated fraction.

- Purpose

The purpose of this calculation is typically to determine the proportion of outliers (in this case, fraudulent transactions) within a dataset. This can be important in contexts such as fraud detection, where understanding the prevalence of fraudulent activities is crucial for designing and evaluating detection systems.

```
print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
```

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

- This expression is a way to filter the data DataFrame to include only the rows where the value in the Class column is 1.
- In the context of fraud detection, it is common to have a Class column where 1 indicates a fraudulent transaction and 0 indicates a valid transaction.

2. len(data[data['Class'] == 1])

- The len() function returns the number of rows in the filtered DataFrame where Class is 1. This effectively counts the number of fraudulent transactions.

3. 'Fraud Cases: {}'.format(len(data[data['Class'] == 1]))

- The format method is used to insert the count of fraudulent transactions into the string.
- {}.format(...) replaces {} with the provided argument.
- This creates a string that includes the number of fraud cases, such as 'Fraud Cases: 123'.

4. print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))

- This prints the formatted string to the console, displaying the count of fraudulent transactions.

5. data[data['Class'] == 0]

- Similar to the previous filtering, this expression filters the data DataFrame to include only the rows where the value in the Class column is 0.

6. len(data[data['Class'] == 0])

- This returns the number of rows in the filtered DataFrame where Class is 0, effectively counting the number of valid transactions.

7. 'Valid Transactions: {}'.format(len(data[data['Class'] == 0]))

- This uses the format method to insert the count of valid transactions into the string, creating a string like 'Valid Transactions: 987'.

8. print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))

- This prints the formatted string to the console, displaying the count of valid transactions.

Purpose

The purpose of these statements is to:
Provide a quick summary of the dataset by showing the number of fraudulent and valid transactions.
Allow you to understand the distribution of fraud cases in the dataset, which is crucial for tasks such as data analysis, model training, and evaluation.

```
corrmat = data.corr()
fig = plt.figure(figsize = (12, 9))
```

Correlation matrix

```
sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()
```

1. sns.heatmap:
- This function is from the Seaborn library, which is used to create heatmaps, a type of visualization that displays data in a matrix format where individual values are represented by colors.

2. corrmat:
- This is the data being passed to the heatmap function. Typically, corrmat would be a correlation matrix, which is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables.
- A correlation matrix is usually generated using the corr() method on a DataFrame, e.g., corrmat = data.corr().

3. vmax=0.8:
- This sets the maximum value for the colormap scale. In this case, vmax=0.8 means that the highest value of the colormap will correspond to a correlation coefficient of 0.8. This can help in making the heatmap more interpretable by focusing on a specific range of values.

4. square=True:
- This parameter ensures that each cell in the heatmap is square-shaped, making the heatmap easier to read and aesthetically pleasing.

5. plt.show():
- This function is from the Matplotlib library, which Seaborn is built on top of. plt.show() displays the plot that has been created.
- Without calling plt.show(), the heatmap may not be rendered, especially in scripts or certain interactive environments.

```
columns = data.columns.tolist()
```
Get all the columns from the dataFrame.

```
columns = [c for c in columns if c not in ["Class"]]
```
Filter the columns to remove data we do not want.

```
target = "Class"
```
Store the variable we'll be predicting on.

```
X = data[columns]
Y = data[target]
```

1. data:
- This refers to a DataFrame, likely created using the Pandas library, which holds your dataset. A DataFrame is a 2-dimensional labeled data structure with columns of potentially different types.

2. columns:
- This variable holds a list of column names that you want to include as features for your model. These columns represent the input variables (features) in your dataset.

3. target:
- This variable holds the name of the column in the DataFrame that you want to predict. It represents the output variable (label) in your dataset.

4. X = data[columns]:
- This statement selects the columns specified in the columns list from the data DataFrame and assigns them to the variable X.
- X now contains a DataFrame with only the feature columns needed for modeling.

5. Y = data[target]:
- This statement selects the column specified by the target variable from the data DataFrame and assigns it to the variable Y.
- Y now contains a Series (or a DataFrame if target is a list of multiple columns) with the target variable that you want to predict.

```
print(X.shape)
print(Y.shape)
```
Print shapes.

```
from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
```

Importing the needed libraries.

```
state = 1
```

Define random states.

```
classifiers = {
    "Isolation Forest": IsolationForest(max_samples=len(X),
                                        contamination=outlier_fraction,
                                        random_state=state),
    "Local Outlier Factor": LocalOutlierFactor(
        n_neighbors=20,
        contamination=outlier_fraction)}

plt.figure(figsize=(9, 7))
n_outliers = len(Fraud)
for i, (clf_name, clf) in enumerate(classifiers.items()):

    # fit the data and tag outliers
    if clf_name == "Local Outlier Factor":
        y_pred = clf.fit_predict(X)
        scores_pred = clf.negative_outlier_factor_
    else:
        clf.fit(X)
        scores_pred = clf.decision_function(X)
        y_pred = clf.predict(X)

    # Reshape the prediction values to 0 for valid, 1 for fraud.
    y_pred[y_pred == 1] = 0
    y_pred[y_pred == -1] = 1

    n_errors = (y_pred != Y).sum()

    # Run classification metrics
    print('{}: {}'.format(clf_name, n_errors))
    print(accuracy_score(Y, y_pred))
    print(classification_report(Y, y_pred))
```

Define outlier detection tools to be compared