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

Pandas:

Use Case: Pandas is a powerful data manipulation and analysis library for Python. It provides data structures like Series and DataFrame, which are particularly useful for handling structured data such as spreadsheets or SQL tables. Pandas is widely used for tasks like data cleaning, exploration, and preparation.

```
import pandas as pd

# Reading a CSV file into a DataFrame
data = pd.read_csv('example.csv')

# Filtering data
filtered_data = data[data['column_name'] > 10]

# Aggregation
grouped_data = data.groupby('category_column')['numeric_column'].mean()

# Creating summary statistics
summary_stats = data.describe()
```

NumPy:

Use Case: NumPy is a numerical computing library in Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays. NumPy is essential for scientific computing and tasks involving numerical operations and linear algebra.

```
import numpy as np

# Creating arrays
array_a = np.array([1, 2, 3])
array_b = np.array([4, 5, 6])

# Performing mathematical operations
result = array_a + array_b

# Linear algebra
dot_product = np.dot(array_a, array_b)
```

Matplotlib:

Use Case: Matplotlib is a plotting library for Python. It allows you to create a wide variety of static, animated, and interactive plots. Matplotlib is commonly used for data visualization to explore and communicate insights from data.

```
import matplotlib.pyplot as plt

# Plotting a simple line chart
x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x, y)
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Simple Line Chart')
plt.show()
```

```
!pip install quandl
import quandl
```

Quandl is a platform that provides access to a vast collection of financial, economic, and alternative data sources. It allows users to retrieve and download datasets in a structured format, making it particularly useful for financial analysts

```
data['Open - Close'] = data['Open'] - data['Close']
data['High - Low'] = data['High'] - data['Last']
data = data.dropna()
```

**data['Open - Close'] = data['Open'] - data['Close']:**

Reason: The difference between the opening and closing prices can provide information about the price movement within a trading day. Traders and analysts often look at this difference, known as the daily price range, to gauge market volatility and potential trends.

**data['High - Low'] = data['High'] - data['Last']:**

Reason: The difference between the high and low prices can indicate the intraday volatility. Traders use this information to assess the range of price movements during a given time period.

**data = data.dropna():**

Reason: Many machine learning algorithms cannot handle missing values, so it's common practice to remove or fill in missing data before training a model. This line ensures that the dataset does not contain any missing values after creating the new features.

IMP part :

```
Y = np.where(data['Close'].shift(-1)>data['Close'], 1, -1)
```

1. data['Close'].shift(-1): This part of the code shifts the 'Close' prices one step backward. It essentially represents the closing prices of the next time period (e.g., the next day) compared to the current day.
2. data['Close'].shift(-1) > data['Close']: This creates a boolean Series where each element is True if the closing price of the next time period is greater than the current closing price, and False otherwise.
3. np.where(condition, x, y): This NumPy function is used to assign values based on a specified condition. In this case:

If the condition (data['Close'].shift(-1) > data['Close']) is True, then the corresponding element in Y is set to 1.

If the condition is False, then the corresponding element in Y is set to -1.

So, effectively, Y is a binary array where:

1 indicates an expected price increase in the next time period.

-1 indicates an expected price decrease in the next time period.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# To find the optimal value of K which is a hyperparameter
params = {'n_neighbors' : [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]}
knn = neighbors.KNeighborsClassifier()
model = GridSearchCV(knn, params, cv=5)

#fit the model
model.fit(X_train, y_train)

#Accuracy Score
accuracy_train = accuracy_score(y_train, model.predict(X_train))
accuracy_test = accuracy_score(y_test, model.predict(X_test))

print('Train_data Accuracy: %.2f' %accuracy_train)
print('Test_data Accuracy: %.2f' %accuracy_test)
```

**Importing Tools:**

Imports necessary tools (like KNeighborsClassifier for KNN, GridSearchCV for parameter tuning, and accuracy_score for evaluating the model).

**Setting Up the Model:**

Defines a range of values to try for a specific parameter (number of neighbours).

Initializes a KNN model.

**Tuning the Model with Grid Search:**

Uses GridSearchCV to try different values of the number of neighbors and find the one that works best. It does this by testing different combinations of the parameter values with cross-validation.

**Training the Model:**

Trains the KNN model on the training data, and during this process, it figures out the best value for the number of neighbors.

**Evaluating Model Performance:**

Measures how well the model performs on the training set and the test set using accuracy, which is a measure of how often the model makes correct predictions.

**Printing Results:**

Displays the accuracy of the model on both the training and test data.