COS30018 - Option B - Task 2: Data processing 1

Jiin Wen Tan 102846565

This function will allow you to deal with the NaN issue in the data:

```
#Deal with NaN issue with forward fill method
def handle_missing_values(df):
    if df.isnull().sum().sum() > 0:
        df.fillna(method='ffill', inplace=True)
    return df
```

With the function above to deal with NaN (Not a number) issue. In this case I had chosen forward fill method as it might be the best choice for now. However, there will be limitation when first row happened to be NaN there won't be a forward fill happening in that case.
`def handle_missing_values(df):` function is to check if there are any missing values in the dataframe and will fill missing values using the forward fill method.
With `if df.isnull().sum().sum() > 0:` to check total number missing value to be atleast greater than 0. First sum: gives a series with the number of missing values per column. Second sum: adds up the counts to give the total number of missing values in the entire dataframe.
Then `df.fillna(method='ffill', inplace=True)`to fill in the missing value with the forward fill method and will be modified directly in the dataframe

Splitting data into train and test sets:

```
from sklearn.model_selection import train_test_split

def custom_train_test_split(data, split_ratio=0.8, method='random'):
    train_data, test_data = train_test_split(data, train_size=split_ratio, random_state=42)

    return train_data, test_data

train, test = custom_train_test_split(data, split_ratio=0.8, method='random')
```

Parameters:
`data` Dataframe that was intended to be splited
`split_ratio=0.8` The proportion of the dataset was set to 80% training data and 20% testing data
`method='random'` Method was set to random by default indicates data will be split randomly

`train_data, test_data = train_test_split(data, train_size=split_ratio, random_state=42)`
By calling `train_test_split` function from scikit-learn to randomly splits dataset into training and test sets, and the training set to be decided with `train_size=split_ratio`, `random_state=42` is to ensure the split is repoducible after running multiple times with the same input data.

Store data into local machine:

```
def data_to_file(DATA_FILE):
    if os.path.exists(DATA_FILE):
        print(f"Loading data from {DATA_FILE}")
        data = pd.read_csv(DATA_FILE)
    else:
        print(f"Fetching data from {DATA_SOURCE} for {COMPANY}")
        data = yf.download(COMPANY, start=TRAIN_START, end=TRAIN_END, progress=False)
        print(f"Saving data to {DATA_FILE}")
        data.to_csv(DATA_FILE)
    return

if os.path.exists(DATA_FILE):
        print(f"Loading data from {DATA_FILE}")
        data = pd.read_csv(DATA_FILE)
```

With the above, if the file was found then will load and read data from the file in this case the file is named "TSLA_data.csv"

```
print(f"Fetching data from {DATA_SOURCE} for {COMPANY}")
        data = yf.download(COMPANY, start=TRAIN_START, end=TRAIN_END, progress=False)
        print(f"Saving data to {DATA_FILE}")
        data.to_csv(DATA_FILE)
```

If the file was not found, it will fetches data using `yf.download` method from `import yfinance as yf` from the range of start to end date and save it as "TSLA_data.csv