Imputation methods:

1. Mean Imputation:
   * Suitable for: Stationary data, data with no strong trend or seasonality
   * Not suitable for: Data with strong trend, seasonality, or high volatility

def mean\_imputation(data):

return data.fillna(data.mean())

1. Median Imputation:
   * Suitable for: Stationary data, data with no strong trend or seasonality, data with outliers
   * Not suitable for: Data with strong trend, seasonality, or high volatility

def median\_imputation(data):

return data.fillna(data.median())

1. Mode Imputation:
   * Suitable for: Categorical data, data with no strong trend or seasonality
   * Not suitable for: Continuous data, data with strong trend, seasonality, or high volatility

def mode\_imputation(data):

return data.fillna(data.mode().iloc[0])

1. Forward-fill (Last Observation Carried Forward) Imputation:
   * Suitable for: Data with trend, seasonality, or cyclicity
   * Not suitable for: Highly volatile data

def forward\_fill\_imputation(data):

return data.ffill()

1. Backward-fill (Next Observation Carried Backward) Imputation:
   * Suitable for: Data with trend, seasonality, or cyclicity
   * Not suitable for: Highly volatile data

def backward\_fill\_imputation(data):

return data.bfill()

1. Linear Interpolation Imputation:
   * Suitable for: Data with trend, seasonality, or cyclicity
   * Not suitable for: Highly volatile data, non-linear trends

def linear\_interpolation\_imputation(data):

return data.interpolate(method='linear')

1. Polynomial or Spline Interpolation Imputation:
   * Suitable for: Data with non-linear trends, seasonality, or cyclicity
   * Not suitable for: Highly volatile data

def polynomial\_interpolation\_imputation(data, order=2):

return data.interpolate(method='polynomial', order=order)

def spline\_interpolation\_imputation(data, order=3):

return data.interpolate(method='spline', order=order)

1. Time Series Decomposition Imputation:
   * Suitable for: Data with strong trend, seasonality, or cyclicity
   * Not suitable for: Highly volatile data, non-stationary data

# Requires statsmodels library

import statsmodels.api as sm

def decomposition\_imputation(data, freq):

decomposition = sm.tsa.seasonal\_decompose(data,

import statsmodels.api as sm

def decomposition\_imputation(data, freq):

decomposition = sm.tsa.seasonal\_decompose(data, freq=freq, model='additive', extrapolate\_trend='freq')

trend = decomposition.trend

seasonal = decomposition.seasonal

residual = decomposition.resid

# Impute missing values in the residual component using mean imputation

residual\_imputed = residual.fillna(residual.mean())

# Recombine the components to obtain the imputed data

imputed\_data = trend + seasonal + residual\_imputed

return imputed\_data

1. Moving Average Imputation:
   * Suitable for: Data with trend, seasonality, or cyclicity
   * Not suitable for: Highly volatile data

def moving\_average\_imputation(data, window):

moving\_avg = data.rolling(window=window).mean()

return data.fillna(moving\_avg)

1. Exponential Smoothing Imputation:
   * Suitable for: Data with trend, seasonality, or cyclicity
   * Not suitable for: Highly volatile data

def exponential\_smoothing\_imputation(data, alpha):

return data.interpolate(method='akima')

1. K-Nearest Neighbors Imputation:
   * Suitable for: Data with complex patterns, non-linear trends, or seasonality
   * Not suitable for: Large datasets (due to computational complexity)

# Requires scikit-learn library

from sklearn.impute import KNNImputer

def knn\_imputation(data, n\_neighbors=5):

imputer = KNNImputer(n\_neighbors=n\_neighbors)

imputed\_data = imputer.fit\_transform(data.values.reshape(-1, 1))

return pd.Series(imputed\_data.ravel(), index=data.index)

1. Linear interpolation:
   * Suitable for: Trend, Cyclicity, Stationarity
   * Not suitable for: Volatility, Seasonality
2. Mean imputation:
   * Suitable for: Stationarity
   * Not suitable for: Trend, Seasonality, Volatility, Cyclicity
3. Median imputation:
   * Suitable for: Stationarity
   * Not suitable for: Trend, Seasonality, Volatility, Cyclicity
4. Mode imputation:
   * Suitable for: Categorical data, Stationarity
   * Not suitable for: Trend, Seasonality, Volatility, Cyclicity
5. Last Observation Carried Forward (LOCF) or Forward Fill:
   * Suitable for: Cyclicity, Stationarity
   * Not suitable for: Trend, Seasonality, Volatility
6. Next Observation Carried Backward (NOCB) or Backward Fill:
   * Suitable for: Cyclicity, Stationarity
   * Not suitable for: Trend, Seasonality, Volatility
7. Seasonal decomposition-based imputation:
   * Suitable for: Trend, Seasonality, Stationarity
   * Not suitable for: Volatility, Cyclicity
8. Moving average imputation:
   * Suitable for: Trend, Stationarity, Cyclicity
   * Not suitable for: Seasonality, Volatility
9. Exponential smoothing imputation:
   * Suitable for: Trend, Stationarity, Cyclicity
   * Not suitable

10. Time series regression imputation: - Suitable for: Trend, Seasonality, Cyclicity, Stationarity - Not suitable for: Volatility

1. State Space Models and Kalman Smoothing:
   * Suitable for: Trend, Seasonality, Cyclicity, Stationarity, Volatility
   * Not suitable for: N/A
2. Machine Learning-based imputation (e.g., K-Nearest Neighbors, Random Forest, or Neural Networks):
   * Suitable for: Trend, Seasonality, Cyclicity, Stationarity, Volatility
   * Not suitable for: N/A

Options:”

1. **Identify and separate missing days (market closed days):** Since the missing days are due to market closures, it's essential to identify and separate them from the rest of the data. You can create a separate time series for these missing days and fill them with NaN values.

import pandas as pd

def separate\_market\_closed\_days(data, closed\_days):

closed\_days\_data = pd.Series(index=closed\_days, data=np.nan)

return closed\_days\_data

1. **Impute missing values within trading days:** For missing values within trading days, you can use the Kalman Filter Imputation method, as it is suitable for high volatility data and can adapt to changes in the underlying state of the system.

imputed\_data = kalman\_filter\_imputation(data)

1. **Recombine the imputed data with the missing days data:** After imputing the missing values within trading days, you can recombine the imputed data with the missing days data to obtain a complete time series.

def recombine\_data(imputed\_data, closed\_days\_data):

combined\_data = pd.concat([imputed\_data, closed\_days\_data]).sort\_index()

return combined\_data

1. **Impute missing values for market closed days:** Since the missing values for market closed days are not random and are due to the market being closed, you can use a method like linear interpolation to fill in the missing values. This method will provide a reasonable estimate of the stock price during the closed days, considering the stock prices before and after the closed days.

def interpolate\_closed\_days(data):

return data.interpolate(method='linear')

Here's the complete process:

# Separate market closed days

closed\_days\_data = separate\_market\_closed\_days(data, closed\_days)

# Impute missing values within trading days

imputed\_data = kalman\_filter\_imputation(data)

# Recombine the imputed data with the missing days data

combined\_data = recombine\_data(imputed\_data, closed\_days\_data)

# Impute missing values for market closed days

final\_imputed\_data = interpolate\_closed\_days(combined\_data)

This approach should provide a reasonable imputation for the missing values in your stock price data, considering its volatility, non-normality, heteroskedasticity, and missing days.

Option 2

1. Forward-fill imputation for missing values within market hours: Since stock prices are often highly autocorrelated in short-term intervals, forward-filling the missing values within market hours would preserve the continuity of the data. This method would be appropriate for short periods of missing data within market hours, where the immediate previous value is a reasonable estimate for the missing value.
2. Forward-fill and back-fill imputation for Saturdays: As you mentioned, Saturdays are market closed days, and it's important to impute them based on the values from the surrounding Fridays and Sundays. First, forward-fill the missing values for Saturdays using the stock prices from the previous Friday. Then, back-fill any remaining missing values using the stock prices from the following Sunday. This approach maintains continuity in the data by considering the prices before and after the market closure.
3. Time series regression imputation for holidays and longer periods of missing data: For holidays and other longer periods of missing data, time series regression imputation can be a good choice. This method uses regression models, which can take into account trend, seasonality, and cyclicity to predict the missing values. You can fit a time series regression model using historical data, and use the model to impute the missing values during holidays or other longer periods of missing data.
4. Finally, evaluate the imputed data: After applying the above imputation methods, evaluate the imputed data by comparing summary statistics, distribution, and autocorrelation with the original data. This step ensures that the imputation methods have not significantly altered the overall characteristics of the time series data. You can also use visualization techniques such as histograms, box plots, and time series plots to visually inspect the imputed data.