1. EDA: The best way to start any ML problem is by exploring the data. When working with time series data, some things to consider are:

%perform exploratory statistical analysis on the data. Data is likely

%distributed non parametrically so quartile tests for outliers will not be

%very effective. Results from EDA will hopefully inform how to deal with

%outliers/noise missing values. High frequency data.

1. Plotting: Start by plotting the raw time series data and also consider plotting the returns (percentage change) of the data over time. Other useful plots include histograms of the returns or boxplots to compare the distribution of returns across different time periods.

%Hurst exponent

%Seasonality

1. Decompose the Time Series: Use additive or multiplicative models to decompose the time series into trend, seasonal, and residual components.

%Trend

%Corellation between

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.seasonal import seasonal\_decompose

from statsmodels.graphics.tsaplots import plot\_acf, plot\_ccf

from statsmodels.tsa.arima.model import ARIMA

from scipy.stats import spearmanr

# Detrend and demean the time series

data\_detrended = np.array(pd.Series(data.Var2).detrend())

data\_demeaned = data\_detrended - np.mean(data\_detrended)

# Decompose time series into trend, seasonality, and residual components

result = seasonal\_decompose(data\_detrended, model='additive', period=12)

data\_trend = result.trend

data\_seasonal = result.seasonal

data\_residual = result.resid

# Analyze and model cleaned time series using autocorr, crosscorr, and ARIMA

plot\_acf(data\_residual, title='Autocorrelation of Residual Component')

plot\_ccf(data\_demeaned, data\_seasonal, title='Cross-Correlation between Detrended and Seasonal Components')

model = ARIMA(data.Var1, order=(1, 0, 1))

est = model.fit()

residuals = est.resid

plt.plot(residuals)

plt.title('Residuals of ARIMA Model')

# Spearman rank correlation test

rho, pval = spearmanr(x, y)

print(f'Spearman rank correlation test result: rho = {rho}, p = {pval}')

# Covariance

covariance = np.cov(x, y)

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Assume that data\_detrended and data\_seasonal are two pandas series

# Calculate cross-correlation using np.correlate with mode='full'

cross\_corr = np.correlate(data\_detrended, data\_seasonal, mode='full')

# Shift the cross-correlation vector to have zero at the center

lags = np.arange(-(len(data\_detrended)-1), len(data\_detrended))

cross\_corr\_shifted = np.roll(cross\_corr, len(data\_detrended)-1)

# Plot the cross-correlation using matplotlib

plt.plot(lags, cross\_corr\_shifted)

plt.title('Cross-Correlation between Detrended and Seasonal Components')

plt.xlabel('Lag')

plt.ylabel('Cross-correlation')

plt.show()

1. Seasonal Subseries Plots: Divide the time series into seasonal periods and create a subseries plot for each period to examine the patterns and trends within each season.
2. Detect Change Points or Structural Breaks: Use algorithms (e.g. PELT) or visual methods (e.g. CUSUM) to detect change points or structural breaks in the time series.
3. Spectral Analysis: Decompose the time series into its frequency components using techniques such as Fourier analysis or wavelet analysis to identify any periodicity or cycles.

%Periodogram

%Seasonality

1. Time Series Clustering: Group similar time series together based on their patterns and trends to identify potential similarities or differences between different stocks.
2. Test for Stationarity: Check whether the data is stationary by modeling the log returns and making them stationary (e.g. by taking the first difference). Use statistical tests (e.g. Augmented Dickey-Fuller test) or visual methods (e.g. rolling mean and standard deviation) to determine if the data is now stationary.

%Since time series stock price data, useful to use logrithmic

%transformation. The distribution of log returns can unlike linear returns

%easily be project to any horizon. Log returns typically have a symmetric

%distribution which makes modelling easier (stock prices are often assumed

%to be log normally distributed - log-returns follow a normal distribution)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import zscore

# Read data

price\_data = pd.read\_csv('path/to/price\_data.csv', parse\_dates=['Time'])

# Compute the logarithmic returns of the stock prices

log\_returns = pd.DataFrame({'Time': price\_data.Time,

'Var2': np.log(price\_data.Var2) - np.log(price\_data.Var2.shift(1)),

'Var3': np.log(price\_data.Var3) - np.log(price\_data.Var3.shift(1))}).dropna()

# Plot

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))

axs[0].plot(log\_returns.Time, log\_returns.Var2)

axs[0].set\_xlabel('Time')

axs[0].set\_ylabel('Value')

axs[0].set\_title('Log returns T1')

axs[1].plot(log\_returns.Time, log\_returns.Var3)

axs[1].set\_xlabel('Time')

axs[1].set\_ylabel('Value')

axs[1].set\_title('Log returns T2')

# Normalize the data using Z-Score normalization

log\_returns\_normalized = log\_returns.apply(lambda x: zscore(x, nan\_policy='omit'))

lt1 = log\_returns\_normalized.Var2

lt2 = log\_returns\_normalized.Var3

# Plot logarithmic return on top of original time series

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))

axs[0].plot(log\_returns\_normalized.Time, lt1)

axs[0].set\_xlabel('Time')

axs[0].set\_ylabel('Value')

axs[0].set\_title('Log returns T1')

axs[1].plot(log\_returns\_normalized.Time, lt2)

axs[1].set\_xlabel('Time')

axs[1].set\_ylabel('Value')

axs[1].set\_title('Log returns T2')

# Summary stats of logarithmic returns

stats = log\_returns\_normalized.describe()

print(stats)

# Verify normality of logarithmic returns. If non-normal then will have to use non-parametric tests

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))

axs[0].hist(lt1, bins=20)

axs[0].set\_xlabel('Value')

axs[0].set\_ylabel('Frequency')

axs[0].set\_title('T1')

axs[1].hist(lt2, bins=20)

axs[1].set\_xlabel('Value')

axs[1].set\_ylabel('Frequency')

axs[1].set\_title('T2')

% % Estimate an AR(2) model and find roots of characteristic equation

%test for normality.

% a low p-value indicates that the null hypothesis can be rejected and the

% data is not normally distributed.

% Apply Shapiro-Wilk test

% Apply Kolmogorov-Smirnov test

1. Descriptive Statistics: Calculate basic summary statistics such as mean, median, standard deviation, skewness, and kurtosis. Other useful measures include the maximum and minimum values, the range of the data, and the coefficient of variation (CV).

% Portmanteau test or Box-Pierce test

% White noise

% Calculate mean, variance, and covariance

%Summary statistic of original data

Outliers

Detect outliers:

Calculate log returns

Test for nomality if non normal, use modified zscore otherwise zscore

%%percent of missing values

%what is the maximum range of continuous data

%create a histogram of number of missing values

%interpolate missing values based on window size from nan histogram

% https://au.mathworks.com/help/matlab/matlab\_prog/clean-timetable-with-missing-duplicate-or-irregular-times.html

%Remove nan values from each and store each times series in separate

%varialbe

def find\_max\_missing\_range(series):

"""

Finds the maximum range of continuous missing data in a pandas Series object.

Parameters:

series (pandas.Series): A pandas Series object containing the data.

Returns:

max\_range (int): The maximum range of continuous missing data.

"""

# Check if the series has any missing values

if series.isnull().any():

# Find the start and end points of consecutive missing values

is\_missing = series.isnull()

start = is\_missing & ~is\_missing.shift(1)

end = is\_missing & ~is\_missing.shift(-1)

# Get the positions of the start and end points

start\_pos = series.index[start]

end\_pos = series.index[end]

# Calculate the lengths of each sequence

lengths = (end\_pos - start\_pos).dt.days

# Find the maximum range of continuous missing data

max\_range = lengths.max()

else:

# If there are no missing values, return 0

max\_range = 0

return max\_range

1. Box Plots: Use box plots to visualize the distribution of the time series and detect any outliers or extreme values.

Discuss the Need for Performing Correlation Analysis and Testing for Autocorrelation and Partial Autocorrelation: Emphasize the importance of understanding the relationship between variables and testing for autocorrelation and partial autocorrelation to identify potential AR and MA terms for a SARIMA model.

%Autocorellation

%Cyclicity

% White noise

1. Correlation analysis: In addition to calculating the Pearson correlation coefficient between the stock prices at different time points, it may also be useful to examine the cross-correlation function (CCF) between the stock prices and other variables that may be related, such as the prices of other stocks in the same industry, or macroeconomic indicators like interest rates or GDP growth.
2. Autocorrelation and partial autocorrelation: In addition to examining the ACF and PACF plots to identify potential AR and MA terms for a SARIMA model, it may also be useful to look at higher-order autocorrelations (e.g. ACF and PACF plots for lags > 12 for monthly data) to identify potential seasonality or longer-term dependencies in the data.

* Collect historical data of the stock prices over a period of time (e.g. daily, weekly, monthly).
* Check for missing values and outliers and handle them appropriately (e.g. impute, drop, replace).
* Transform the data into a stationary series if needed (e.g. differencing, log transformation).
* Discuss the need for handling missing values and outliers appropriately.
* Explain the need for transforming data into a stationary series if needed.
* 0. Clean dataremove outliers, wrong data, define outliers and remove/Intepolate

def impute\_missing\_values(series, window):

# check if the series has any missing values

if series.isnull().any():

# calculate the rolling mean with the specified window size

rolling\_mean = series.rolling(window=window, min\_periods=1).mean()

# fill the missing values with the rolling mean

imputed\_series = series.fillna(rolling\_mean)

else:

# if there are no missing values, return the original series

imputed\_series = series

return imputed\_series

1. Benchmarking
   1. To determine whether an MAE of 96.05 or an MAPE of 36% is good, you need to create a baseline by predicting the mean or using other domain-specific methods such as a rolling mean or predicting the last available data value. This baseline will serve as a benchmark to evaluate the performance of your models.
2. Model testing
   1. Cleaned data
      1. Linear regression
      2. Exponential smoothing
      3. ARIMA
      4. ~~GARCH~~
      5. ~~Mean reverting~~
      6. Vector auto regression
   2. Log returns of original data
      1. Random forest
      2. SVM
      3. Gaussian process
      4. LightGBM XGBoost
      5. LSTM

* Bayesian model selection: This involves using Bayesian methods to compare different models or variations of models based on their posterior probabilities, which can provide a more informative measure of model fit compared to traditional model selection techniques.
* Ensemble methods: This involves combining multiple models or variations of models to improve prediction accuracy and reduce overfitting, such as by using bagging, boosting, or stacking techniques.
* Granger causality tests: This involves testing for causal relationships between different stocks or macroeconomic variables that may influence the stock prices, such as by examining the lagged effects of one variable on another.
* Forecast evaluation: This involves comparing the actual forecasted values to the observed values and examining the accuracy and reliability of the forecasts, such as by using metrics such as mean absolute percentage error (MAPE) or symmetric mean absolute percentage error (SMAPE).

import numpy as np

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.arima.model import ARIMAResults

from statsmodels.tsa.arima.model import ARIMAParams

from statsmodels.tsa.arima.model import \_arma\_predict\_out\_of\_sample

from statsmodels.tsa.arima.model import \_arma\_predict\_in\_sample

from statsmodels.tsa.arima.model import \_arma\_predict

from arch import arch\_model

# Split data into training and testing sets

n = len(stock\_price)

train\_idx = np.random.choice(n, int(n\*0.8), replace=False)

test\_idx = np.setdiff1d(np.arange(n), train\_idx)

trainData = stock\_price[train\_idx]

testData = stock\_price[test\_idx]

# Train linear regression model with one lagged variable

mdl1 = ARIMA(trainData['Close'], order=(1,0,0)).fit()

# Calculate MSE of linear regression model on testing set

yhat1,\_,\_ = mdl1.forecast(len(testData))

mse1 = np.mean((yhat1 - testData['Close'])\*\*2)

# Calculate AIC of linear regression model

aic1 = mdl1.aic

# Train an ARIMA model on the training set with order (1,1,0)

trainDiff = np.diff(trainData['Close'])

mdl2 = ARIMA(trainDiff, order=(1,1,0)).fit()

# Specify a GARCH(1,1) variance model and an ARIMA(1,1,1) mean model

varMdl = arch\_model(trainDiff, mean='AR', lags=1, vol='GARCH', p=1, q=1)

meanMdl = ARIMA(trainDiff, order=(1,1,1), exog=varMdl.resid).fit()

# Forecast the conditional mean and variance for 10 steps ahead

yhat2,\_,\_ = meanMdl.forecast(steps=10, exog=varMdl.forecast(horizon=10).variance[-1, :][None,:])

vhat2 = varMdl.forecast(horizon=10).variance[-1, :]

# Plot the forecasted mean and variance with 95% confidence intervals

import matplotlib.pyplot as plt

fig, axs = plt.subplots(2,1, figsize=(10,8))

axs[0].plot(np.concatenate((trainData['Close'][-100:], testData['Close'])), 'b')

axs[0].plot(np.arange(len(trainData['Close']), len(trainData['Close'])+10), yhat2, 'r', linewidth=2)

axs[0].plot(np.arange(len(trainData['Close']), len(trainData['Close'])+10), yhat2+1.96\*np.sqrt(vhat2), 'k--')

axs[0].plot(np.arange(len(trainData['Close']), len(trainData['Close'])+10), yhat2-1.96\*np.sqrt(vhat2), 'k--')

axs[0].legend(['Historical', 'Forecast', '95% Interval'], loc='upper left')

axs[0].set\_title('Forecasted Conditional Mean')

axs[1].plot(np.arange(len(trainData['Close']), len(trainData['Close'])+10), vhat2, 'r', linewidth=2)

axs[1].plot(np.arange(len(trainData['Close']), len(trainData['Close'])+10), vhat2+1.96\*np.sqrt(vhat2), 'k--')

axs[1].

% Plot the forecasted mean and variance with 95% confidence intervals

figure;

subplot(2,1,1);

plot(y(end-100:end),'b');

hold on;

plot(length(y)+1:length(y)+10,yF,'r','LineWidth',2);

plot(length(y)+1:length(y)+10,yF+1.96\*sqrt(yMSE),'k--');

plot(length(y)+1:length(y)+10,yF-1.96\*sqrt(yMSE),'k--');

hold off;

legend('Historical','Forecast','95% Interval','Location','NorthWest');

title('Forecasted Conditional Mean');

subplot(2,1,2);

plot(vF,'r','LineWidth',2);

hold on;

plot(vF+1.96\*sqrt(vMSE),'k--');

plot(vF-1.96\*sqrt(vMSE),'k--');

hold off;

legend('Forecast','95% Interval','Location','NorthWest');

title('Forecasted Conditional Variance');

% Fit the model to the training set using maximum likelihood

estMdlTrain = estimate(meanMdl,ytrain);

% Compute the log-likelihood of the fitted model on the testing set

[logL] = infer(estMdlTrain,ytest);

% Perform a 5-fold cross-validation on the whole data set

cvlogL = cvpartition(logL,n,'KFold',5);

% Compute the mean log-likelihood for each fold

meanlogL = zeros(5,1);

for i = 1:5

meanlogL(i) = mean(logL(cvlogL.test(i)));

end

% Display the mean log-likelihood for each fold and their average

disp('Mean log-likelihood for each fold:');

disp(meanlogL);

disp('Average mean log-likelihood:');

disp(mean(meanlogL));

import pandas as pd

import numpy as np

from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error

import statsmodels.api as sm

from arch import arch\_model

from statsmodels.tsa.vector\_ar.var\_model import VAR

from sklearn.ensemble import RandomForestRegressor

from sklearn.gaussian\_process import GaussianProcessRegressor

from lightgbm import LGBMRegressor

from xgboost import XGBRegressor

# Load time series data

df = pd.read\_csv('data.csv', parse\_dates=['date'], index\_col='date')

# Split data into training and test sets

train = df.loc[:'2022-01-01']

test = df.loc['2022-01-02':]

# Calculate benchmark MAE and MAPE

mean\_baseline = train.mean()

last\_value\_baseline = train.iloc[-1]

rolling\_mean\_baseline = train.rolling(window=30).mean().iloc[-1]

mae\_mean = mean\_absolute\_error(test, [mean\_baseline] \* len(test))

mae\_last = mean\_absolute\_error(test, [last\_value\_baseline] \* len(test))

mae\_rolling = mean\_absolute\_error(test, [rolling\_mean\_baseline] \* len(test))

mape\_mean = mean\_absolute\_percentage\_error(test, [mean\_baseline] \* len(test))

mape\_last = mean\_absolute\_percentage\_error(test, [last\_value\_baseline] \* len(test))

mape\_rolling = mean\_absolute\_percentage\_error(test, [rolling\_mean\_baseline] \* len(test))

print('Benchmark MAE (mean):', mae\_mean)

print('Benchmark MAE (last value):', mae\_last)

print('Benchmark MAE (rolling mean):', mae\_rolling)

print('Benchmark MAPE (mean):', mape\_mean)

print('Benchmark MAPE (last value):', mape\_last)

print('Benchmark MAPE (rolling mean):', mape\_rolling)

# Fit and evaluate ARIMA model

arima = sm.tsa.ARIMA(train, order=(1,1,1)).fit()

arima\_preds = arima.forecast(len(test))

mae\_arima = mean\_absolute\_error(test, arima\_preds)

mape\_arima = mean\_absolute\_percentage\_error(test, arima\_preds)

print('ARIMA MAE:', mae\_arima)

print('ARIMA MAPE:', mape\_arima)

# Fit and evaluate GARCH model

garch = arch\_model(train, vol='Garch', p=1, o=0, q=1).fit(disp='off')

garch\_preds = garch.forecast(horizon=len(test), method='simulation').mean['h.1']

mae\_garch = mean\_absolute\_error(test, garch\_preds)

mape\_garch = mean\_absolute\_percentage\_error(test, garch\_preds)

print('GARCH MAE:', mae\_garch)

print('GARCH MAPE:', mape\_garch)

# Fit VAR model

model = VAR(train)

results = model.fit()

# Make predictions

lag\_order = results.k\_ar

preds = results.forecast(train.values[-lag\_order:], len(test))

# Evaluate predictions

mse = np.mean((preds - test.values)\*\*2)

mae = np.mean(np.abs(preds - test.values))

mape = np.mean(np.abs((preds - test.values) / test.values))

print("MSE: ", mse)

print("MAE: ", mae)

print("MAPE: ", mape)

# Fit random forest model

model = RandomForestRegressor()

model.fit(train.drop('target', axis=1), train['target'])

# Make predictions

preds = model.predict(test.drop('target', axis=1))

# Evaluate predictions

mse = mean\_squared\_error(test['target'], preds)

mae = mean\_absolute\_error(test['target'], preds)

mape = np.mean(np.abs((preds - test['target']) / test['target']))

print("MSE: ", mse)

print("MAE: ", mae)

print("MAPE: ", mape)

# Fit Gaussian Process model

kernel = ConstantKernel() \* RBF()

model = GaussianProcessRegressor(kernel=kernel)

model.fit(train.drop('target', axis=1), train['target'])

# Make predictions

preds, std = model.predict(test.drop('target', axis=1), return\_std=True)

# Evaluate predictions

mse = np.mean((preds - test['target'])\*\*2)

mae = np.mean(np.abs(preds - test['target']))

mape = np.mean(np.abs((preds - test['target']) / test['target']))

print("MSE: ", mse)

print("MAE: ", mae)

print("MAPE: ", mape)

# Fit LightGBM model

params = {'objective': 'regression'}

d\_train = lgb.Dataset(train.drop('target', axis=1), label=train['target'])

model = lgb.train(params, d\_train)

preds = model.predict(test.drop('target', axis=1))

# Evaluate predictions

mse = mean\_squared\_error(test['target'], preds)

mae = mean\_absolute\_error(test['target'], preds)

mape = np.mean(np.abs((pred

1. Model evaluation
   1. Cross validation
   2. MAE, RMSE, MAPE/WAPE, RMSSE, and MASE
   3. Hierarchical forecasting can also help to make your forecasts more coherent and accurate.
   * Checking for model assumptions and diagnostics (e.g. residuals analysis) and validating the model.
   * Checking for model assumptions and diagnostics such as normality, homoscedasticity and independence of residuals using statistical tests (e.g. Jarque-Bera test) or visual methods (e.g. QQ plot).
   * Validating the model using cross-validation techniques such as walk-forward validation or rolling window validation.

Model Selection:

* Select an appropriate time series model that can capture the patterns and dynamics of the data, such as ARIMA, VAR, or LSTM.
* Use Bayesian model selection techniques to compare different variations of the model based on their posterior probabilities and choose the one with the best fit.

1. Markov Chain Monte Carlo (MCMC): MCMC is a Bayesian method that can be used to estimate posterior distributions of model parameters. This method generates a sequence of samples from the posterior distribution of the model parameters, which can be used to calculate the posterior probabilities of different models. MCMC can handle complex models with many parameters and can also provide information about the uncertainty in the parameter estimates.
2. Bayes Factor: Bayes Factor is a statistical method that can be used to compare two or more models. It is the ratio of the marginal likelihoods of two models. The marginal likelihood is the probability of the observed data given the model parameters, integrated over the prior distribution of the parameters. The Bayes Factor provides a measure of the evidence in favor of one model over another. A Bayes Factor greater than one indicates evidence in favor of the model in the numerator, while a Bayes Factor less than one indicates evidence in favor of the model in the denominator.
3. Cross-Validation: Cross-validation is a method that can be used to estimate the predictive accuracy of different models. It involves partitioning the data into training and validation sets and fitting the models to the training set. The models are then used to make predictions on the validation set, and the prediction error is calculated. This process is repeated multiple times with different partitions of the data to obtain an estimate of the prediction error for each model. The model with the lowest prediction error is selected.
4. Hierarchical Bayesian Models: Hierarchical Bayesian models can be used to estimate the uncertainty in the model parameters and to account for parameter dependence across different levels of the model hierarchy. In the context of time series data, hierarchical Bayesian models can be used to estimate the parameters of the underlying dynamic process and to model the dependence between observations at different time points.
5. Bayesian Structural Time Series Models: Bayesian structural time series models are a class of models that can be used to decompose time series data into trend, seasonal, and irregular components. These models can be used to make short-term and long-term forecasts and can also be used to estimate the effects of external interventions on the time series data.
6. Forecast Evaluation:
   * Use the trained model to generate forecasts on the test set and compare the actual forecasted values to the observed values.
   * Evaluate the accuracy and reliability of the forecasts using metrics such as mean absolute percentage error (MAPE) or symmetric mean absolute percentage error (SMAPE).
   * Adjust the model and repeat the evaluation process until satisfactory performance is achieved.
7. Model selection and implement strategy
8. Plotting: Start by plotting the raw time series data and also consider plotting the returns (percentage change) of the data over time. Other useful plots include histograms of the returns or boxplots to compare the distribution of returns across different time periods.

%Hurst exponent

%Seasonality

1. Decompose the Time Series: Use additive or multiplicative models to decompose the time series into trend, seasonal, and residual components.

%Trend

%Corellation between

1. Seasonal Subseries Plots: Divide the time series into seasonal periods and create a subseries plot for each period to examine the patterns and trends within each season.
2. Detect Change Points or Structural Breaks: Use algorithms (e.g. PELT) or visual methods (e.g. CUSUM) to detect change points or structural breaks in the time series.
3. Spectral Analysis: Decompose the time series into its frequency components using techniques such as Fourier analysis or wavelet analysis to identify any periodicity or cycles.

%Periodogram

%Seasonality

1. Time Series Clustering: Group similar time series together based on their patterns and trends to identify potential similarities or differences between different stocks.
2. Test for Stationarity: Check whether the data is stationary by modeling the log returns and making them stationary (e.g. by taking the first difference). Use statistical tests (e.g. Augmented Dickey-Fuller test) or visual methods (e.g. rolling mean and standard deviation) to determine if the data is now stationary.

%Since time series stock price data, useful to use logrithmic

%transformation. The distribution of log returns can unlike linear returns

%easily be project to any horizon. Log returns typically have a symmetric

%distribution which makes modelling easier (stock prices are often assumed

%to be log normally distributed - log-returns follow a normal distribution)

% % Estimate an AR(2) model and find roots of characteristic equation

%test for normality.

% a low p-value indicates that the null hypothesis can be rejected and the

% data is not normally distributed.

% Apply Shapiro-Wilk test

% Apply Kolmogorov-Smirnov test

1. Descriptive Statistics: Calculate basic summary statistics such as mean, median, standard deviation, skewness, and kurtosis. Other useful measures include the maximum and minimum values, the range of the data, and the coefficient of variation (CV).

% Portmanteau test or Box-Pierce test

% White noise

% Calculate mean, variance, and covariance

%Summary statistic of original data

1. Box Plots: Use box plots to visualize the distribution of the time series and detect any outliers or extreme values.

Discuss the Need for Performing Correlation Analysis and Testing for Autocorrelation and Partial Autocorrelation: Emphasize the importance of understanding the relationship between variables and testing for autocorrelation and partial autocorrelation to identify potential AR and MA terms for a SARIMA model.

%Autocorellation

%Cyclicity

% White noise

1. Correlation analysis: In addition to calculating the Pearson correlation coefficient between the stock prices at different time points, it may also be useful to examine the cross-correlation function (CCF) between the stock prices and other variables that may be related, such as the prices of other stocks in the same industry, or macroeconomic indicators like interest rates or GDP growth.

Autocorrelation and partial autocorrelation: In addition to examining the ACF and PACF plots to identify potential AR and MA terms for a SARIMA model, it may also be useful to look at higher-order autocorrelations (e.g. ACF and PACF plots for lags > 12 for monthly data) to identify potential seasonality or longer-term dependencies