```
#2. Forecast future stock prices using the ARIMA model and to optimize a stock portfolio
based on MVO (Mean-Variance Optimization)

# Perform the Augmented Dickey-Fuller test to check for stationarity in the time series.
def adf_test(series):
    result = adfuller(series, autolag='AIC')
    return result[1]

#Forecast future stock prices using the ARIMA model for each ticker in the dataset.
def forecast_arima(data, steps=183):
    print("The forecast is being calculated. Please wait.")
    forecasted_data = pd.DataFrame()
    arima_params = {}

    for ticker in data.columns:
        series = data[ticker].dropna()

        if adf_test(series) > 0.05:
            d = 1
        else:
            d = 0

        best_aic = float("inf")
        best_order = (0, d, 0)
        model = None

        try:
            p, q = 0, 0
            while True:
                try:
                    temp_model = auto_arima(series, start_p=p, start_q=q, max_p=p+1,
max_q=q+1, d=d, seasonal=False,
                                                stepwise=False, suppress_warnings=True,
error_action="ignore", trace=False)
                    temp_aic = temp_model.aic()

                    if temp_aic < best_aic:
                        best_aic = temp_aic
                        best_order = temp_model.order
                        model = temp_model
                        p, q = p + 1, q + 1
                    else:
                        break
                except Exception as e:
                    print(f"Failed to fit ARIMA model for {ticker} with p={p}, q={q}: {e}")
                    break

            forecast = model.predict(n_periods=steps)
            forecasted_data[ticker] = forecast
            arima_params[ticker] = best_order
        except Exception as e:
            print(f"Failed to fit ARIMA model for {ticker}: {e}")

    return forecasted_data, arima_params

# Calculate the daily returns of the stock data.
def calculate_daily_returns(data):
    returns = data.pct_change().dropna()
    return returns

#Calculate key statistics including: the mean returns, correlation matrix, and covariance
matrix from the daily returns.
def calculate_statistics(returns):
    mean_returns = returns.mean()
    corr_matrix = returns.corr()
    cov_matrix = returns.cov()
    return mean_returns, corr_matrix, cov_matrix
```

```
#Calculate the expected return and risk (standard deviation) of the portfolio.
def portfolio_performance(weights, mean_returns, cov_matrix):
    portfolio_return = np.dot(weights, mean_returns)
    portfolio_stddev = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    return portfolio_return, portfolio_stddev

#Objective function for portfolio optimization, balancing return and risk.
def objective_function(weights, mean_returns, cov_matrix, risk_preference):
    portfolio_return, portfolio_stddev = portfolio_performance(weights, mean_returns,
cov_matrix)
    return -(portfolio_return * risk_preference - portfolio_stddev * (1 - risk_preference))

# Constraint function to ensure that the sum of the portfolio weights equals 1
def check_sum(weights):
    return np.sum(weights) - 1

# Optimize the portfolio by finding the asset weights that maximize the user's utility.
def optimize_portfolio(mean_returns, cov_matrix, risk_preference):
    num_assets = len(mean_returns)
    args = (mean_returns, cov_matrix, risk_preference)

    constraints = {'type': 'eq', 'fun': check_sum}
    bounds = tuple((0, 1) for asset in range(num_assets))

    result = minimize(objective_function, num_assets * [1. / num_assets], args=args,
                      method='SLSQP', bounds=bounds, constraints=constraints)
    return result

# Integrate data downloading, ARIMA forecasting, and portfolio optimization to get the
optimal portfolio.
def get_optimal_portfolio(tickers, start_date, risk_preference):
    data = download_data_fillna(tickers, start_date="2023-01-03",
end_date=datetime.today()-timedelta(days=1))
    forecasted_data, arima_params = forecast_arima(data)
    combined_data = pd.concat([data, forecasted_data], axis=0)
    daily_returns = calculate_daily_returns(combined_data)
    mean_returns, corr_matrix, cov_matrix = calculate_statistics(daily_returns)

    optimal_portfolio = optimize_portfolio(mean_returns, cov_matrix,
risk_preference=risk_preference)
```