```
  [Parameter(Mandatory = $false)]
    [String]
    $VenvDir,
    [Parameter(Mandatory = $false)]
    [String]
    $Prompt
)

<# Function declarations -----------------------------------------------
--- #>

<#
.Synopsis
Remove all shell session elements added by the Activate script, including
the
addition of the virtual environment's Python executable from the
beginning of
the PATH variable.

.Parameter NonDestructive
If present, do not remove this function from the global namespace for the
session.

#>
function global:deactivate ([switch]$NonDestructive) {
    # Revert to original values

    # The prior prompt:
    if (Test-Path -Path Function:_OLD_VIRTUAL_PROMPT) {
        Copy-Item -Path Function:_OLD_VIRTUAL_PROMPT -Destination
Function:prompt
        Remove-Item -Path Function:_OLD_VIRTUAL_PROMPT
    }

    # The prior PYTHONHOME:
    if (Test-Path -Path Env:_OLD_VIRTUAL_PYTHONHOME) {
        Copy-Item -Path Env:_OLD_VIRTUAL_PYTHONHOME -Destination
Env:PYTHONHOME
        Remove-Item -Path Env:_OLD_VIRTUAL_PYTHONHOME
    }

    # The prior PATH:
    if (Test-Path -Path Env:_OLD_VIRTUAL_PATH) {
        Copy-Item -Path Env:_OLD_VIRTUAL_PATH -Destination Env:PATH
        Remove-Item -Path Env:_OLD_VIRTUAL_PATH
    }

    # Just remove the VIRTUAL_ENV altogether:
    if (Test-Path -Path Env:VIRTUAL_ENV) {
        Remove-Item -Path env:VIRTUAL_ENV
    }

    # Just remove VIRTUAL_ENV_PROMPT altogether.
    if (Test-Path -Path Env:VIRTUAL_ENV_PROMPT) {
        Remove-Item -Path env:VIRTUAL_ENV_PROMPT
    }

    # Just remove the _PYTHON_VENV_PROMPT_PREFIX altogether:
    if (Get-Variable -Name "_PYTHON_VENV_PROMPT_PREFIX" -ErrorAction
SilentlyContinue) {
        Remove-Variable -Name _PYTHON_VENV_PROMPT_PREFIX -Scope Global -
Force
    }

    # Leave deactivate function in the global namespace if requested:
    if (-not $NonDestructive) {
        Remove-Item -Path function:deactivate
    }
}

<#
.Description
Get-PyVenvConfig parses the values from the pyvenv.cfg file located in
the
given folder, and returns them in a map.

For each line in the pyvenv.cfg file, if that line can be parsed into
exactly
two strings separated by `=` (with any amount of whitespace surrounding
the =)
then it is considered a `key = value` line. The left hand string is the
key,
the right hand is the value.

If the value starts with a `'` or a `"` then the first and last character
is
stripped from the value before being captured.

.Parameter ConfigDir
Path to the directory that contains the `pyvenv.cfg` file.
#>
function Get-PyVenvConfig(
    [String]
    $ConfigDir
) {
    Write-Verbose "Given ConfigDir=$ConfigDir, obtain values in
pyvenv.cfg"

    # Ensure the file exists, and issue a warning if it doesn't (but
still allow the function to continue).
    $pyvenvConfigPath = Join-Path -Resolve -Path $ConfigDir -ChildPath
'pyvenv.cfg' -ErrorAction Continue

    # An empty map will be returned if no config file is found.
    $pyvenvConfig = @{ }

    if ($pyvenvConfigPath) {
        Write-Verbose "File exists, parse `key = value` lines"
        $pyvenvConfigContent = Get-Content -Path $pyvenvConfigPath

        $pyvenvConfigContent | ForEach-Object {
            $keyval = $PSItem -split "\s*=\s*", 2
            if ($keyval[0] -and $keyval[1]) {
                $val = $keyval[1]

                # Remove extraneous quotations around a string value.
                if ("'""".Contains($val.Substring(0, 1))) {
                    $val = $val.Substring(1, $val.Length - 2)
                }

                $pyvenvConfig[$keyval[0]] = $val
                Write-Verbose "Adding Key: '$($keyval[0])'='$val'"
            }
        }
    }
    return $pyvenvConfig
}


<# Begin Activate script -------------------------------------------------
--- #>

# Determine the containing directory of this script
$VenvExecPath = Split-Path -Parent $MyInvocation.MyCommand.Definition
$VenvExecDir = Get-Item -Path $VenvExecPath

Write-Verbose "Activation script is located in path: '$VenvExecPath'"
Write-Verbose "VenvExecDir Fullname: '$($VenvExecDir.FullName)"
Write-Verbose "VenvExecDir Name: '$($VenvExecDir.Name)"

# Set values required in priority: CmdLine, ConfigFile, Default
# First, get the location of the virtual environment, it might not be
# VenvExecDir if specified on the command line.
if ($VenvDir) {
    Write-Verbose "VenvDir given as parameter, using '$VenvDir' to
determine values"
}
else {
    Write-Verbose "VenvDir not given as a parameter, using parent
directory name as VenvDir."
    $VenvDir = $VenvExecDir.Parent.FullName.TrimEnd("\\/")
    Write-Verbose "VenvDir=$VenvDir"
}

# Next, read the `pyvenv.cfg` file to determine any required value such
# as `prompt`.
$pyvenvCfg = Get-PyVenvConfig -ConfigDir $VenvDir

# Next, set the prompt from the command line, or the config file, or
# just use the name of the virtual environment folder.
if ($Prompt) {
    Write-Verbose "Prompt specified as argument, using '$Prompt'"
}
else {
    Write-Verbose "Prompt not specified as argument to script, checking
pyvenv.cfg value"
    if ($pyvenvCfg -and $pyvenvCfg['prompt']) {
        Write-Verbose "  Setting based on value in
pyvenv.cfg='$($pyvenvCfg['prompt'])'"
        $Prompt = $pyvenvCfg['prompt'];
    }
    else {
        Write-Verbose "  Setting prompt based on parent's directory's
name. (Is the directory name passed to venv module when creating the
virtual environment)"
        Write-Verbose "  Got leaf-name of $VenvDir='$(Split-Path -Path
$venvDir -Leaf)'"
        $Prompt = Split-Path -Path $venvDir -Leaf
    }
}

Write-Verbose "Prompt = '$Prompt'"
Write-Verbose "VenvDir='$VenvDir'"

# Deactivate any currently active virtual environment, but leave the
# deactivate function in place.
deactivate -nondestructive

# Now set the environment variable VIRTUAL_ENV, used by many tools to
determine
# that there is an activated venv.
$env:VIRTUAL_ENV = $VenvDir

if (-not $Env:VIRTUAL_ENV_DISABLE_PROMPT) {

    Write-Verbose "Setting prompt to '$Prompt'"

    # Set the prompt to include the env name
    # Make sure _OLD_VIRTUAL_PROMPT is global
    function global:_OLD_VIRTUAL_PROMPT { "" }
    Copy-Item -Path function:prompt -Destination
function:_OLD_VIRTUAL_PROMPT
    New-Variable -Name _PYTHON_VENV_PROMPT_PREFIX -Description "Python
virtual environment prompt prefix" -Scope Global -Option ReadOnly -
Visibility Public -Value $Prompt

    function global:prompt {
        Write-Host -NoNewline -ForegroundColor Green
"($_PYTHON_VENV_PROMPT_PREFIX) "
        _OLD_VIRTUAL_PROMPT
    }
    $env:VIRTUAL_ENV_PROMPT = $Prompt
}

# Clear PYTHONHOME
```

```
if (Test-Path -Path Env:PYTHONHOME) {
    Copy-Item -Path Env:PYTHONHOME -Destination
Env:_OLD_VIRTUAL_PYTHONHOME
    Remove-Item -Path Env:PYTHONHOME
}

# Add the venv to the PATH
Copy-Item -Path Env:PATH -Destination Env:_OLD_VIRTUAL_PATH
$Env:PATH = "$VenvExecDir$([System.IO.Path]::PathSeparator)$Env:PATH"
```

```
.Dt TTX 1 "FontTools Manual"
.Sh NAME
.Nm ttx
.Nd tool for manipulating TrueType and OpenType fonts
.Sh SYNOPSIS
.Nm
.Bk
.Op Ar option ...
.Ek
.Bk
.Ar file ...
.Ek
.Sh DESCRIPTION
.Nm
is a tool for manipulating TrueType and OpenType fonts.  It can convert
TrueType and OpenType fonts to and from an
.Tn XML Ns -based format called
.Tn TTX .
.Tn TTX
files have a
.Ql .ttx
extension.
.Pp
For each
.Ar file
argument it is given,
.Nm
detects whether it is a
.Ql .ttf ,
.Ql .otf
or
.Ql .ttx
file and acts accordingly: if it is a
.Ql .ttf
or
.Ql .otf
file, it generates a
.Ql .ttx
file; if it is a
.Ql .ttx
file, it generates a
.Ql .ttf
```

```
or
.Ql .otf
file.
.Pp
By default, every output file is created in the same directory as the
corresponding input file and with the same name except for the
extension, which is substituted appropriately.
.Nm
never overwrites existing files; if necessary, it appends a suffix to
the output file name before the extension, as in
.Pa Arial#1.ttf .
.Ss "General options"
.Bl -tag -width ".Fl t Ar table"
.It Fl h
Display usage information.
.It Fl d Ar dir
Write the output files to directory
.Ar dir
instead of writing every output file to the same directory as the
corresponding input file.
.It Fl o Ar file
Write the output to
.Ar file
instead of writing it to the same directory as the
corresponding input file.
.It Fl v
Be verbose.  Write more messages to the standard output describing what
is being done.
.It Fl a
Allow virtual glyphs ID's on compile or decompile.
.El
.Ss "Dump options"
The following options control the process of dumping font files
(TrueType or OpenType) to
.Tn TTX
files.
.Bl -tag -width ".Fl t Ar table"
.It Fl l
List table information.  Instead of dumping the font to a
.Tn TTX
file, display minimal information about each table.
.It Fl t Ar table
Dump table
.Ar table .
This option may be given multiple times to dump several tables at
once.  When not specified, all tables are dumped.
.It Fl x Ar table
Exclude table
.Ar table
from the list of tables to dump.  This option may be given multiple
times to exclude several tables from the dump.  The
.Fl t
and
.Fl x
```

```
options are mutually exclusive.
.It Fl s
Split tables.  Dump each table to a separate
.Tn TTX
file and write (under the name that would have been used for the output
file if the
.Fl s
option had not been given) one small
.Tn TTX
file containing references to the individual table dump files.  This
file can be used as input to
.Nm
as long as the referenced files can be found in the same directory.
.It Fl i
.\" XXX: I suppose OpenType programs (exist and) are also affected.
Don't disassemble TrueType instructions.  When this option is specified,
all TrueType programs (glyph programs, the font program and the
pre-program) are written to the
.Tn TTX
file as hexadecimal data instead of
assembly.  This saves some time and results in smaller
.Tn TTX
files.
.It Fl y Ar n
When decompiling a TrueType Collection (TTC) file,
decompile font number
.Ar n ,
starting from 0.
.El
.Ss "Compilation options"
The following options control the process of compiling
.Tn TTX
files into font files (TrueType or OpenType):
.Bl -tag -width ".Fl t Ar table"
.It Fl m Ar fontfile
Merge the input
.Tn TTX
file
.Ar file
with
.Ar fontfile .
No more than one
.Ar file
argument can be specified when this option is used.
.It Fl b
Don't recalculate glyph bounding boxes.  Use the values in the
.Tn TTX
file as is.
.El
.Sh "THE TTX FILE FORMAT"
You can find some information about the
.Tn TTX
file format in
.Pa documentation.html .
```

```
In particular, you will find in that file the list of tables understood
by
.Nm
and the relations between TrueType GlyphIDs and the glyph names used in
.Tn TTX
files.
.Sh EXAMPLES
In the following examples, all files are read from and written to the
current directory.  Additionally, the name given for the output file
assumes in every case that it did not exist before
.Nm
was invoked.
.Pp
Dump the TrueType font contained in
.Pa FreeSans.ttf
to
.Pa FreeSans.ttx :
.Pp
.Dl ttx FreeSans.ttf
.Pp
Compile
.Pa MyFont.ttx
into a TrueType or OpenType font file:
.Pp
.Dl ttx MyFont.ttx
.Pp
List the tables in
.Pa FreeSans.ttf
along with some information:
.Pp
.Dl ttx -l FreeSans.ttf
.Pp
Dump the
.Sq cmap
table from
.Pa FreeSans.ttf
to
.Pa FreeSans.ttx :
.Pp
.Dl ttx -t cmap FreeSans.ttf
.Sh NOTES
On MS\-Windows and MacOS,
.Nm
is available as a graphical application to which files can be dropped.
.Sh SEE ALSO
.Pa documentation.html
.Pp
.Xr fontforge 1 ,
.Xr ftinfo 1 ,
.Xr gfontview 1 ,
.Xr xmbdfed 1 ,
.Xr Font::TTF 3pm
.Sh AUTHORS
.Nm
```

```
was written by
.An -nosplit
.An "Just van Rossum" Aq just@letterror.com .
.Pp
This manual page was written by
.An "Florent Rougon" Aq f.rougon@free.fr
for the Debian GNU/Linux system based on the existing FontTools
documentation.  It may be freely used, modified and distributed without
restrictions.

@echo off

rem This file is UTF-8 encoded, so we need to update the current code
page while executing it
for /f "tokens=2 delims=:." %%a in ('"%SystemRoot%\System32\chcp.com"')
do (
    set _OLD_CODEPAGE=%%a
)
if defined _OLD_CODEPAGE (
    "%SystemRoot%\System32\chcp.com" 65001 > nul
)

set VIRTUAL_ENV=D:\MOON009 - Python - Stock Price Prediction\myenv

if not defined PROMPT set PROMPT=$P$G

if defined _OLD_VIRTUAL_PROMPT set PROMPT=%_OLD_VIRTUAL_PROMPT%
if defined _OLD_VIRTUAL_PYTHONHOME set
PYTHONHOME=%_OLD_VIRTUAL_PYTHONHOME%

set _OLD_VIRTUAL_PROMPT=%PROMPT%
set PROMPT=(myenv) %PROMPT%

if defined PYTHONHOME set _OLD_VIRTUAL_PYTHONHOME=%PYTHONHOME%
set PYTHONHOME=

if defined _OLD_VIRTUAL_PATH set PATH=%_OLD_VIRTUAL_PATH%
if not defined _OLD_VIRTUAL_PATH set _OLD_VIRTUAL_PATH=%PATH%

set PATH=%VIRTUAL_ENV%\Scripts;%PATH%
set VIRTUAL_ENV_PROMPT=(myenv)

:END
if defined _OLD_CODEPAGE (
    "%SystemRoot%\System32\chcp.com" %_OLD_CODEPAGE% > nul
    set _OLD_CODEPAGE=
)
@echo off

if defined _OLD_VIRTUAL_PROMPT (
    set "PROMPT=%_OLD_VIRTUAL_PROMPT%"
)
set _OLD_VIRTUAL_PROMPT=
if defined _OLD_VIRTUAL_PYTHONHOME (
    set "PYTHONHOME=%_OLD_VIRTUAL_PYTHONHOME%"
    set _OLD_VIRTUAL_PYTHONHOME=
)

if defined _OLD_VIRTUAL_PATH (
    set "PATH=%_OLD_VIRTUAL_PATH%"
)

set _OLD_VIRTUAL_PATH=

set VIRTUAL_ENV=
set VIRTUAL_ENV_PROMPT=

home = C:\Users\ADMIN\AppData\Local\Programs\Python\Python311
include-system-site-packages = false
version = 3.11.5
executable =
C:\Users\ADMIN\AppData\Local\Programs\Python\Python311\python.exe
command =
C:\Users\ADMIN\AppData\Local\Programs\Python\Python311\python.exe -m venv
D:\MOON009 - Python - Stock Price Prediction\myenv

__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

# PyInstaller
#  Usually these files are written by a python script from a template
#  before PyInstaller builds the exe, so as to inject date/other infos
into it.
```

```
*.manifest
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

# Unit test / coverage reports
htmlcov/
.tox/
.nox/
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
*.cover
*.py,cover
.hypothesis/
.pytest_cache/
cover/

# Translations
*.mo
*.pot

# Django stuff:
*.log
local_settings.py
db.sqlite3
db.sqlite3-journal

# Flask stuff:
instance/
.webassets-cache

# Scrapy stuff:
.scrapy

# Sphinx documentation
docs/_build/

# PyBuilder
.pybuilder/
target/

# Jupyter Notebook
.ipynb_checkpoints

# IPython
profile_default/
ipython_config.py

# pyenv
#   For a library or package, you might want to ignore these files since
the code is
#   intended to run in multiple environments; otherwise, check them in:
# .python-version

# pipenv
#   According to pypa/pipenv#598, it is recommended to include
Pipfile.lock in version control.
#   However, in case of collaboration, if having platform-specific
dependencies or dependencies
#   having no cross-platform support, pipenv may install dependencies
that don't work, or not
#   install all needed dependencies.
#Pipfile.lock

# poetry
#   Similar to Pipfile.lock, it is generally recommended to include
poetry.lock in version control.
#   This is especially recommended for binary packages to ensure
reproducibility, and is more
#   commonly ignored for libraries.
#   https://python-poetry.org/docs/basic-usage/#commit-your-poetrylock-
file-to-version-control
#poetry.lock

# pdm
#   Similar to Pipfile.lock, it is generally recommended to include
pdm.lock in version control.
#pdm.lock
#   pdm stores project-wide configurations in .pdm.toml, but it is
recommended to not include it
#   in version control.
#   https://pdm.fming.dev/#use-with-ide
.pdm.toml

# PEP 582; used by e.g. github.com/David-OConnor/pyflow and
github.com/pdm-project/pdm
__pypackages__/

# Celery stuff
celerybeat-schedule
celerybeat.pid

# SageMath parsed files
*.sage.py

# Environments
.env
.venv
env/
venv/
ENV/
env.bak/
venv.bak/
```

```
# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
#  JetBrains specific template is maintained in a separate
JetBrains.gitignore that can
#  be found at
https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
#  and can be added to the global gitignore or merged into this file.
For a more nuclear
#  option (not recommended) you can uncomment the following to ignore the
entire idea folder.
#.idea/

# Models
models/

import collections
import os
import joblib

import pandas as pd
import numpy as np
import yfinance as yf

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

from parameters import *
```

```
# ----------------------------------------------------------------------
----------------------------------------------------------------------
---

def processData(isStoredDataLocally=True, company="TSLA",
startDate="2015-01-01", endDate="2020-01-01", dataSource = "yahoo",
                    numOfPastDays=50, numOfFutureDays=1, lookupSteps=1,
featureColumns=["Open", "High", "Low", "Close"],
                    trainRatio=0.8, randomSplit=False, randomSeed=None,
                    isScaledData=True, featureRange=(0, 1),
isStoredScaler=True):

    #   Check if the stock data's folder is exist
    if (not os.path.isdir("stockdatas")):
        os.mkdir("stockdatas")

    # LOAD DATA
    dataFileName = "{}-company_{}-type_{}-startD_{}-endD".format(company,
"data", startDate, endDate)
    dataFilePath = "stockdatas/{}.csv".format(dataFileName)

    processedData = {}

    if (dataSource == "yahoo"):
        # Check if the data has been saved
        if (os.path.exists(dataFilePath)):
            data = pd.read_csv(dataFilePath, index_col=0,
parse_dates=True)
            print("Loaded data from the saved file")
        # If the data has not been saved
        else:
            # Fetch data from Yahoo Finance
            data = yf.download(company, start=startDate, end=endDate,
progress=False)

            # Handle NaN values in the data by forwarding fill missing
values
            data.ffill(inplace=True)

            # Save data
            if (isStoredDataLocally):
                data.to_csv(dataFilePath)

        processedData["Data"] = data

        # Add date as a column
        if ("Date" not in data.columns):
            data["Date"] = data.index

        # SCALE DATA
        if (isScaledData):
            (data, featureColumnScalers) = scaleData(data=data,
dataFileName=dataFileName, featureColumns=featureColumns,
featureRange=featureRange, isStoredScaler=isStoredScaler)
```

```
            processedData["ColumnScalers"] = featureColumnScalers

        # Add the target column (label) by shifting by <lookupStep>
        data["Future"] = data["Close"]

        # The last <lookupSteps> column(s) contains NaN in the "Future"
column, get them before handling NaN values
        lastSequence = np.array(data[featureColumns].tail(lookupSteps))

        # Handle NaN values in the data by forwarding fill missing values
        data.ffill(inplace=True)

        # Create a list of sequences of features and their corresponding
targets
        sequenceData = []
        sequences = collections.deque(maxlen=numOfPastDays)

        for (sequence, target) in zip(data[featureColumns +
["Date"]].values, data["Future"].values):
            sequences.append(sequence)

            if (len(sequences) == numOfPastDays):
                sequenceData.append([np.array(sequences), target])

        # Get the last sequence by appending the last <numOfPastDays>
sequence with <lookupSteps> sequence
        # E.g. If lookupSteps=a and numOfPastDays=b, lastSequence should
be of (a+b) length
        # This lastSequence will be used to predict future stock prices
that are not available in the dataset
        lastSequence = list([sequence[:len(featureColumns)] for sequence
in sequences]) + list(lastSequence)

        lastSequence = np.array(lastSequence).astype(np.float32)

        processedData["LastSequence"] = lastSequence

        # Construct the x (sequences) and y (targets)
        (x, y) = ([], [])
        for (sequence, target) in sequenceData:
            x.append(sequence)
            y.append(target)

        # Convert to Numpy arrays
        x = np.array(x)
        y = np.array(y)

        # Reshape the targets to a 2D array, each element is a 1D array
of <numOfFutureDays> value(s)
        if (numOfFutureDays > 0):
            # Calculate the number of rows in the new 2D array
            newYLength = len(y) + 1 - numOfFutureDays
            # Initialize the 2D array
            intermediateY = np.empty((newYLength, numOfFutureDays),
dtype=y.dtype)

            # Fill the new 2D array
            for i in range(newYLength):
                intermediateY[i] = y[i:(i + numOfFutureDays)]

            # Assign the new y array
            y = intermediateY
            # Modify the x to remove the element(s) that do not have the
corresponding element(s) in the new y array
            if (numOfFutureDays > 1):
                x = x[:(1 - numOfFutureDays)]

        # print("================X=================")
        # print(x)
        # print("================Y=================")
        # print(y)

        # SPLIT DATA
        (processedData["XTrain"], processedData["XTest"],
processedData["YTrain"], processedData["YTest"]) = splitData(x=x, y=y,

trainRatio=trainRatio, randomSplit=randomSplit, randomSeed=randomSeed)

        # Get the xTest dates
        dates = processedData["XTest"][:, -1, -1]
        # Retrieve test features dates from the original data
        processedData["TestData"] = processedData["Data"].loc[dates]
        # Remove dupliacate dates
        processedData["TestData"] =
processedData["TestData"][~processedData["TestData"].index.duplicated(kee
p="first")]
        # Remove dates from xTrain and xTest and convert to float32
        processedData["XTrain"] = processedData["XTrain"][:, :,
:len(featureColumns)].astype(np.float32)
        processedData["XTest"] = processedData["XTest"][:, :,
:len(featureColumns)].astype(np.float32)

        return processedData

def splitData(x, y, trainRatio=0.8, randomSplit=False, randomSeed=None):
    if (not 0 < trainRatio < 1):
        raise ValueError("train ratio should be between 0 and 1.")

    if (randomSplit):
        (xTrain, xTest, yTrain, yTest) = train_test_split(x, y,
test_size=(1 - trainRatio), shuffle=False)
    else:
        trainSamples = int(trainRatio * len(x))

        (xTrain, xTest, yTrain, yTest) = (x[:trainSamples],
x[trainSamples:], y[:trainSamples], y[trainSamples:])
```

```
    return (xTrain, xTest, yTrain, yTest)

def scaleData(data, dataFileName, featureColumns=["Open", "High", "Low",
"Close"], featureRange=(0, 1), isStoredScaler=True):
    # Check if the stock data's folder is exist
    if (not os.path.isdir("stockdatas")):
        os.mkdir("stockdatas")

    scaledData = data.copy()
    featureColumnScalers = {}

    for col in featureColumns:
        scalerFilePath = "stockdatas/{}_scaler_{}.pkl".format(col,
dataFileName)

        # Scale the data with the new scaler
        if (not os.path.exists(scalerFilePath)):
            # Scale data
            scaler = MinMaxScaler(feature_range=featureRange)

            data[col] = scaler.fit_transform(np.expand_dims(data[col],
axis=1))

            print("Scaled data.")

            # Save scaler
            if (isStoredScaler):
                joblib.dump(scaler, scalerFilePath)
                print("Saved scaler.")
        # Load scaler and scaled data
        else:
            scaler = joblib.load(scalerFilePath)
            print("Loaded scaler.")

            scaledData[col] = scaler.transform(np.expand_dims(data[col],
axis=1))
            print("Scaled data.")

        featureColumnScalers[col] = scaler

    return scaledData, featureColumnScalers
    import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from mplfinance.original_flavor import candlestick_ohlc

from dataProcessing import *

# -------------------------------------------------------------------------
-------------------------------------------------------------------------
---

def preProcessDiplayedData(data, tradingDaysNumber):
    # Resample data to specified trading days (take the average value of
<tradingDaysNumber> consecutive days)
    if (tradingDaysNumber > 1):
        resampledData =
data.resample("{}B".format(tradingDaysNumber)).agg({"Open": "mean",
"High": "mean", "Low": "mean", "Close": "mean"})
    else:
        resampledData = data.copy()

    # Reset index to convert dates to numerical values for candlestick
plotting
    resampledData.reset_index(inplace=True)

    print("================ INITIAL DATA ====================")
    print(data)
    print("=============== RESAMPLED DATA ===================")
    print(resampledData)

    return resampledData

def candlestickChartDisplay(data, tradingDaysNumber=1, title="Stock
Candlestick Chart", colorup="g", colordown="r"):
    # Check if the number of trading days is less than 1
    if (tradingDaysNumber < 1):
        raise ValueError("tradingDaysNumber must be equal or greater than
1.")
    else:
        resampledData = preProcessDiplayedData(data, tradingDaysNumber)

        # Convert dates from a datetime format to numerical values (for
plotting)
        resampledData["Date"] =
mdates.date2num(resampledData["Date"].tolist())

        # Create a new figure and axes for the candlestick chart,
specifying the initial size of the chart
        (fig, ax) = plt.subplots(figsize=(16, 9))

        # Plot the candlestick chart on these axes using the data from
the DataFrame.
        candlestick_ohlc(ax, resampledData.values, width=0.6,
colorup=colorup, colordown=colordown, alpha=0.8)

        # Format the date
        ax.xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m-%d"))
        fig.autofmt_xdate()

        # Set title and labels
        ax.set_title(title)
        ax.set_xlabel("Date")
        ax.set_ylabel("Price")
        ax.legend(["Open", "High", "Low", "Close"])

        # Set the rotation angle for the x-axis (dates)
        plt.xticks(rotation=45)

        # Adjust the layout of the plot to prevent elements from
overlapping and improve overall aesthetics
        plt.tight_layout()

def boxplotChartDisplay(data, tradingDaysNumber=1, title="Stock Boxplot
Chart"):

    # Check if the number of trading days is less than 1
    if (tradingDaysNumber < 1):
         raise ValueError("tradingDaysNumber must be equal or greater
than 1.")
    else:
        resampledData = preProcessDiplayedData(data, tradingDaysNumber)

        # print("============= PRETRANSPOSED DATA =================")
        # print(resampledData)

        # Transpose DataFrame so that columns become rows and rows become
columns, as the output of the method ax.boxplot for resampledData
        # seems not to be like the requirement - the x-axis would be
Open/High/Low/Close instead of Date
        #   - Set 'Date' column as index and transpose the DataFrame
        transposedData = resampledData.set_index("Date").transpose()

        # print("=============== TRANSPOSED DATA ==================")
        # print(transposedData)

        # Create a new figure and axes for the boxplot chart, specifying
the size of the figure
        (fig, ax) = plt.subplots(figsize=(16, 9))
        ax.boxplot(transposedData)

        # Set title and labels
        ax.set_title(title)
        ax.set_xlabel("Date")
        ax.set_ylabel("Price")
        ax.legend(["Open", "High", "Low", "Close"])

        # Get the "Date" values from the DataFrame index
        dates = transposedData.columns
        # Format the dates to %Y-%m-%d
        dates = [date.strftime("%Y-%m-%d") for date in dates]
        # Set the "Date" values as x-axis ticks and rotate them
        plt.xticks(range(1, len(dates) + 1), dates, rotation=45)

        # Adjust the layout of the plot to prevent elements from
overlapping and improve overall aesthetics
        plt.tight_layout()

def plotSingleFeature(predictionName, actualData, predictedData):
    plt.figure(figsize=(16, 9))
    plt.title(predictionName)
    plt.plot(actualData, label="Actual Prices", color="blue")
    plt.plot(predictedData, label="Predicted Prices", color="orange")
    plt.xlabel("Date")
    plt.ylabel("Price")
    plt.legend()
    plt.show()

    import itertools
import os
import joblib

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

from keras import layers, models, metrics
from pmdarima import auto_arima
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.arima.model import ARIMA, ARIMAResults
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

from parameters import *
from dataProcessing import *
from dataVisualizing import *

# -----------------------------------------------------------------------
-------------------------------------------------------------------------
---

def constructDLModel(featureColumns=["Open", "High", "Low", "Close"],
numOfPastDays=50, numOfFutureDays=1, layersNumber=2, layerSize=256,
layerName=layers.LSTM,
                     dropout=0.3, loss="mean_absolute_error",
optimizer="rmsprop", bidirectional=False):
    # Check if the layersNumber >= 1
    if (layersNumber < 1):
        raise ValueError("Number of layers should be equal to or more
than 1")

    # Initialize a Sequential model
    model = models.Sequential()

    for i in range(layersNumber):
        # First Layer
        if (i == 0):
            if (bidirectional):
                model.add(layers.Bidirectional(layerName(units=layerSize,
return_sequences=True), input_shape=(numOfPastDays,
len(featureColumns))))
            else:
                model.add(layerName(units=layerSize,
return_sequences=True, input_shape=(numOfPastDays, len(featureColumns))))
        # Last layer
        elif (i == layersNumber - 1):
            if (bidirectional):
                model.add(layers.Bidirectional(layerName(units=layerSize,
return_sequences=False)))
            else:
                model.add(layerName(units=layerSize,
return_sequences=False))
        # Other layers
        else:
            if (bidirectional):
                model.add(layers.Bidirectional(layerName(units=layerSize,
return_sequences=True)))
            else:
                model.add(layerName(units=layerSize,
return_sequences=True))

        # Add Dropout after each layer
        model.add(layers.Dropout(dropout))

    # Add Dense output layer (with <NUMBER_OF_FUTURE_DAYS> neuron (unit))
    model.add(layers.Dense(numOfFutureDays))

    # Compile the model (with loss function, evaluation metric
(mean_absolute_error), and optimizer)
    model.summary()
    model.compile(optimizer=optimizer, loss=loss,
metrics=[metrics.MeanAbsoluteError()])

    return model
    def trainAndTestMultivariateDLModel(processedData,
featureColumns=["Open", "High", "Low", "Close"], numOfPastDays=50,
numOfFutureDays=1, layersNumber=2, layerSize=256, layerName=layers.LSTM,
                                 dropout=0.3, loss="mean_absolute_error",
optimizer="rmsprop", bidirectional=False,
                                 epochs=50, batchSize=32):
    # Training
    #   Check if the model's folder is exist
    if (not os.path.isdir("models")):
        os.mkdir("models")

    modelFilePath = "models/{}-fCols_{}-pDays_{}-fdays_{}-layers_{}-
lSize_{}-lName_{}-do_{}-loss_{}-optimizer_{}biD_{}-epochs_{}-
bSize.keras".format(len(featureColumns), numOfPastDays, numOfFutureDays,
layersNumber,

layerSize, str(layerName).split('.')[-1].split("'")[0],

dropout, loss, optimizer, bidirectional, epochs, batchSize)

    if (not os.path.exists(modelFilePath)):
        # Create and train the Deep Learning model
        model = constructDLModel(featureColumns=featureColumns,
numOfPastDays=numOfPastDays, numOfFutureDays=numOfFutureDays,
layersNumber=layersNumber, layerSize=layerSize, layerName=layerName,
                                 dropout=dropout, loss=loss,
optimizer=optimizer, bidirectional=bidirectional)

        model.fit(x=processedData["XTrain"], y=processedData["YTrain"],
epochs=EPOCHS, batch_size=BATCH_SIZE,
validation_data=(processedData["XTest"], processedData["YTest"]))

        # Save the model
        model.save(modelFilePath)
        print("Trained model saved to {}".format(modelFilePath))
    else:
        # Load the model
        model = models.load_model(modelFilePath)
        print("Trained model loaded from {}".format(modelFilePath))

    # Testing
    #   Get actual and predicted data
    yActualData = processedData["YTest"]
    yPredictedData = model.predict(processedData["XTest"])

    #   Descale
    if (IS_SCALED_DATA):
        yActualData =
np.squeeze(processedData["ColumnScalers"]["Close"].inverse_transform(yAct
ualData))
        yPredictedData =
np.squeeze(processedData["ColumnScalers"]["Close"].inverse_transform(yPre
dictedData))
```