```
"""
Experiment 1: Casting Lab
Course: MAT 010
Date: 9/8/22
Name: Michael Nigohosian

Effects of casting temperature and mold material on
cooling rate and grain structure of 99.99% aluminium.

Import data into a pandas dataframe,
and create plot using matplotlib.
"""

# importing 'libraries' gives you access to all of the 'methods' in the library
import pandas as pd  # dataframes
import numpy as np  # math for arrays (columns)
from scipy import stats  # regression line
import matplotlib.pyplot as plt  # plotting


# VARIABLES

# CREATE VARIABLES FOR FILEPATHS
#  "data/fname.xls" implies the fname.xls file is in a
#  subdirectory "data" within the current directory
# TODO: change filenames
filepath_ceramic = "exp1-casting/data/High_temp_ceramic.xls"
filepath_graphite = "exp1-casting/data/High_temp_graphite.xls"
filepath_steel = "exp1-casting/data/High_temp_steel.xls"

# CREATE VARIABLES FOR FREQUENCY OF MEASUREMENTS
# TODO: change values
freq_ceramic = 1 # Hz
freq_graphite = 5 # Hz
freq_steel = 5 # Hz

# CREATE VARIABLES FOR LINEAR FIT RANGE
# TODO: change values
range_ceramic = [15, 50]
range_graphite = [11, 17]
range_steel = [26, 34]

# COLUMNS
# TODO: add axis titles
x_axis = "Time (s)"
y_axis = "Temperature (C)"


def load_data(filepath, x, y, freq=1, header_line=0, view_data=False):
```

```
    """
    Function that takes a filepath and returns a pandas dataframe.
    Generates the df.index based on freq value in Hz.
    """

    # LOAD DATA from file into pandas dataframe
    #  'names' is a list of column names
    #  the header is not on the first row, so set the header parameter to the correct index
    #  hint: first row has index 0
    # TODO: change read_excel() to read_csv() if the data is not in an excel file
    df_loaded = pd.read_excel(filepath, names=[y], header=header_line)
    # print(df_loaded.columns)  # view column names on the terminal

    # delete the column of all NA values
    df_loaded.dropna(axis='columns', how='all', inplace=True)

    # generate the column of time values given the frequency
    # of the sensor data and add the column to the dataframe
    df_loaded[x] = np.arange(start=0, stop=len(df_loaded)/freq, step=(1/freq))  # create
new column for x axis

    # then tell pandas to use the new column as the index and delete the column
    df_loaded.set_index(x, inplace=True, drop=True)

    # # NOTE: setting the column as the 'index' allows us to use df.loc
    # print(df_loaded.loc[20])  # get value at index value 20 eg. at 20 seconds
    # print(df_loaded.iloc[20])  # get value at row index 20 eg. at 4 seconds if freq=5Hz

    if view_data:
        # VIEW DATA
        print(df_loaded.info(), "\n")  # view column names & types, number of rows, etc
        print(df_loaded.head(10), "\n")  # show the first 10 rows on the terminal

    return df_loaded

# CALL FUNCTION AND SAVE DATAFRAMES TO VARIABLES
# TODO: add view_data=True to each of the function calls
df_ceramic = load_data(filepath_ceramic, x=x_axis, y=y_axis, freq=freq_ceramic,
header_line=2, view_data=True)  # TODO: add parameters as needed
df_graphite = load_data(filepath_graphite, x=x_axis, y=y_axis, freq=freq_graphite,
header_line=2, view_data=True)  # TODO: add parameters as needed
df_steel = load_data(filepath_steel, x=x_axis, y=y_axis, freq=freq_steel, header_line=2,
view_data=True)  # TODO: add parameters as needed

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

def plot_data(df, x=x_axis, y=y_axis, r=[0,-1], info_scatter={}):
    """
    Function which takes a pandas dataframe and returns a matplotlib figure object.
```

```
df: pandas dataframe
x: x axis name
y: y axis name
r: range for the linear fit line
info_scatter: dictionary containing key word pairs to pass on to the scatter function

see this url for what parameters pyplot has available:
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.scatter.html
"""

# PLOT VARIABLES
# TODO: customize plot settings
PLOT_TEXTSIZE = 8

# FONT SETTINGS
plt.rc('font', size=PLOT_TEXTSIZE)  # controls default text sizes

# GET FIGURE AND AXES OBJECTS
fig, ax = plt.subplots()  # create new figure and axes



# CREATE A SLICE OF THE DATA
if (r[1] == -1):  # create slice from r[0] to the end of the data
    df_slice = df.loc[(df.index >= r[0])]
else:  # create slice between r[0] and r[1]
    df_slice = df.loc[(df.index >= r[0]) & (df.index <= r[1])]

x_slice = df_slice.index  # assumes plotting against the index
y_slice = df_slice[y]
# print(df_slice)



# GET LINEAR FIT REGRESSION LINE PARAMETERS ax+b
#  Use scipy.stats to do the linear regression on the dataframe slices.
#  Print equ to see all the stats available.
equ = stats.linregress(x_slice, y_slice)
a = equ.slope
b = equ.intercept
r2 = equ.rvalue**2  # r_squared = r_value^2

# Text for the annotation on the plot
# .3f means to show 3 decimal places
txt_equ = f"$y = {b:.3f} + {a:.3f}x$"
txt_r2 = f"$R^2 = {r2:.3f}$"
```

```
    txt_annotation = f"{txt_equ}\n{txt_r2}"

    # view regression information on terminal
    print(txt_equ)  # linear regression equation
    print(f"R-Squared is {r2:0.5f}")
    print(f"Cooling rate is {a:.1f}ºC/s\n")



    # PLOT THE DATA
    plt.scatter(df.index, df[y], **info_scatter)

    # PLOT LINEAR FIT
    plt.plot(x_slice, a*x_slice + b, label="_none_", c='black')  # TODO: customize fit line

    # ADD ANNOTATIONS TO PLOT: add linear fit equation to plot
    #  TODO: place the annotation close to the regression line
    plt.text(0, 0, txt_annotation, va='bottom', ha='left')


    # SET AXIS LABELS
    plt.xlabel(x)
    plt.ylabel(y)

    # SET AXIS LIMITS
    #  axis limits must be set after plot is made
    #  TODO: customize axis limits
    plt.xlim(left=0)  # x axis starts at 0
    plt.ylim(bottom=0)  # y axis starts at 0

    return fig


# CALL FUNCTION AND SHOW PLOTS
#  plot_data() defines 'df', 'x', 'y', and 'r' as parameters explicitly
#  info_scatter is passed on to the plt.scatter() function
#  for example, you can change the size of the points by adding 'info_scatter=dict(s=5)'

#  TODO: customize plot aesthetics. Change the shape or color of the markers.
fig1 = plot_data(df_ceramic, x=x_axis, y=y_axis, r=range_ceramic,
info_scatter=dict(s=5))
fig2 = plot_data(df_graphite, x=x_axis, y=y_axis, r=range_graphite)
fig3 = plot_data(df_steel, x=x_axis, y=y_axis, r=range_steel)


# TODO: customize saved image
# SAVE PLOTS TO filename.png in the plots/ folder
fig1.savefig("exp1-casting/plots/ceramic.png", dpi=150, bbox_inches="tight")
```

```
fig2.savefig("exp1-casting/plots/graphite.png", dpi=150, bbox_inches="tight")
fig3.savefig("exp1-casting/plots/steel.png", dpi=150, bbox_inches="tight")

# SHOW PLOTS in their own windows
#  plt.show() must be called after fig.savefig()
#  otherwise the figure saved will be blank
plt.show()
```