#Importing Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

import os

import re

DATA\_PATH = 'Raw\_Data'

#Present exact one example

#Load Data

rest\_example = pd.read\_csv(DATA\_PATH + "/" + "010.csv")

stress\_example = pd.read\_csv(DATA\_PATH + "/" + "011.csv")

#Calculate the magnitude

rest\_example["magnitude"] = np.sqrt(rest\_example.iloc[:,1]\*\*2+ rest\_example.iloc[:,2]\*\*2 + rest\_example.iloc[:,3]\*\*2)

stress\_example["magnitude"] = np.sqrt(stress\_example.iloc[:,1]\*\*2+ stress\_example.iloc[:,2]\*\*2 + stress\_example.iloc[:,3]\*\*2)

#slice between 3 and 10 seconds

raw\_vis\_rest = rest\_example[(rest\_example["Time (s)"] > 3) & (rest\_example["Time (s)"] <= 8)].copy()

raw\_vis\_stre = stress\_example[(stress\_example["Time (s)"] > 3) & (stress\_example["Time (s)"] <= 8)].copy()

rest\_example["Time (s)"]

##Visualize Pre Slicing

plt.title("Visualize Raw Data- Pre Slicing")

plt.xlabel("Time (s)")

plt.ylabel("Magnitude (m/s^2)")

plt.plot(rest\_example["Time (s)"], rest\_example["magnitude"], label="Rest", alpha = 0.8)

#plt.axhline(y=rest\_example["magnitude"].mean(), color="blue", linestyle="--", label="Rest Mean")

plt.plot(stress\_example["Time (s)"], stress\_example["magnitude"], label="Stress", alpha = 0.5)

#plt.axhline(y=stress\_example["magnitude"].mean(), color="orange", linestyle="--", label="Stress Mean")

plt.legend() # Show legend with labels

plt.show()

#Visulaize Post Slicing

plt.title("Visualize Raw Data- Post Slicing")

plt.xlabel("Time (s)")

plt.ylabel("Magnitude (m/s^2)")

plt.plot(raw\_vis\_rest["Time (s)"], raw\_vis\_rest["magnitude"], label="Rest", alpha = 0.8)

#plt.axhline(y=raw\_vis\_rest["magnitude"].mean(), color="blue", linestyle="--", label="Rest Mean")

plt.plot(raw\_vis\_stre["Time (s)"], raw\_vis\_stre["magnitude"], label="Stress", alpha = 0.5)

#plt.axhline(y=raw\_vis\_stre["magnitude"].mean(), color="orange", linestyle="--", label="Stress Mean")

plt.legend() # Show legend with labels

plt.show()

import matplotlib.pyplot as plt

# Calculate overall minimum and maximum values of magnitude

min\_magnitude = min(raw\_vis\_rest["magnitude"].min(), raw\_vis\_stre["magnitude"].min())

max\_magnitude = max(raw\_vis\_rest["magnitude"].max(), raw\_vis\_stre["magnitude"].max())

# Create subplots with two columns

fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Plot for Rest data

axes[0].plot(raw\_vis\_rest["Time (s)"], raw\_vis\_rest["magnitude"], label="Rest", color="blue", alpha=0.8)

axes[0].set\_title("Rest")

axes[0].set\_xlabel("Time (s)")

axes[0].set\_ylabel("Magnitude (m/s^2)")

axes[0].set\_ylim(min\_magnitude, max\_magnitude)

# Plot for Stress data

axes[1].plot(raw\_vis\_stre["Time (s)"], raw\_vis\_stre["magnitude"], label="Stress", color="orange", alpha=0.5)

axes[1].set\_title("Stress")

axes[1].set\_xlabel("Time (s)")

axes[1].set\_ylabel("Magnitude (m/s^2)")

axes[1].set\_ylim(min\_magnitude, max\_magnitude)

plt.tight\_layout()

plt.show()

#Create a stripplot, presenting all single data points of one examination

fig, axes = plt.subplots(1, 2, figsize=(12, 6))

sns.stripplot(ax=axes[0], data=raw\_vis\_rest["magnitude"], size=4)

sns.stripplot(ax=axes[1], data=raw\_vis\_stre["magnitude"], size=4)

axes[0].set\_xlabel('Rest Example')

axes[1].set\_xlabel('Stress Example')

axes[0].set\_ylabel('Values')

axes[1].set\_ylabel('Values')

axes[0].set\_title('Stripplot - Rest Example')

axes[1].set\_title('Stripplot - Stress Example')

plt.tight\_layout()

plt.show()

#Importing Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

import os

import re

DATA\_PATH = 'Raw\_Data'

final\_df = pd.read\_csv("0\_final\_df.csv", dtype={"id": str})

count\_gender = final\_df[final\_df["sex"] == "male"].count()

# Print the count of males

print(count\_gender)

# Assuming your DataFrame is named 'df'

descriptive\_stats = final\_df.describe()

# Print the descriptive statistics

print(descriptive\_stats)

import pandas as pd

import numpy as np

from scipy import stats

#import the data

df = pd.read\_csv("0\_final\_df.csv")

final\_df = pd.read\_csv("0\_final\_df.csv")

#get the values for rest

rest\_values = df[df["condition"] == "rest"]["standard deviation"].values

#get the values for stress

stress\_values = df[df["condition"] == "stress"]["standard deviation"].values

stats.shapiro(rest\_values) #rest

import numpy as np

from scipy import stats

# Data

stress = final\_df[final\_df["condition"] == "stress"]["standard deviation"].values

rest = final\_df[final\_df["condition"] == "rest"]["standard deviation"].values

# Paired two-sample t-test

t\_statistic, p\_value = stats.ttest\_rel(stress, rest)

# Effect size calculation

n = len(stress)

mean\_diff = np.mean(stress) - np.mean(rest)

pooled\_std = np.sqrt(((n - 1) \* np.var(stress) + (n - 1) \* np.var(rest)) / (2 \* (n - 1)))

effect\_size = mean\_diff / pooled\_std

print("Effect Size (Cohen's d):", effect\_size)

import statsmodels.stats.power as ssp

alpha = 0.05

beta = 0.2

effect\_size = 1.13

sample\_size = ssp.tt\_ind\_solve\_power(effect\_size=effect\_size,

nobs1 = None, ratio = 1, alpha=alpha, power=(1 - beta),

alternative='larger')

print("Sample Size:", round(sample\_size))

# Perform the Wilcoxon signed-rank test (one-tailed, alternative = 'greater')

statistic, p\_value = stats.wilcoxon(stress\_values, rest\_values, alternative='greater')

# Format the p-value

p\_value\_formatted = "< 0.001" if p\_value < 0.001 else "{:.3f}".format(p\_value)

# Print the test statistic and p-value

print("Test statistic:", statistic)

print("p-value:", p\_value\_formatted)

# Perform Fligner-Killeen test for equal variances of paired samples

statistic, p\_value = stats.fligner(rest\_values, stress\_values)

# Format the p-value

p\_value\_formatted = "< 0.001" if p\_value < 0.001 else "{:.3f}".format(p\_value)

# Print the test statistic and p-value

print("Test statistic:", statistic)

print("p-value:", p\_value\_formatted)

# Discrete Distribution

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import binom

# Parameters

n = 30 # Number of trials

p = 0.5 # Probability of success

# Generate binomial distribution

x = np.arange(0, n+1)

binom\_dist = binom.pmf(x, n, p)

# Plot the binomial distribution

plt.bar(x, binom\_dist)

plt.xlabel('Number of Successes')

plt.ylabel('Probability')

plt.title('Binomial Distribution (n=30, p=0.5)')

plt.show()

from scipy.stats import binom\_test

# Parameters

n = 30 # Number of trials

p = 0.5 # Probability of success

x = 12 # Number of successes

# Perform hypothesis test

p\_value = binom\_test(x, n, p, alternative='two-sided')

# Print the p-value

print("p-value:", p\_value)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

import os

from scipy.signal import find\_peaks

import re

DATA\_PATH = 'Raw\_Data'

#Load and show raw data

example = pd.read\_csv(DATA\_PATH + "/" + "140.csv")

example

# Create an empty DataFrame to store the final results

final\_df = pd.DataFrame()

# Get the list of filenames in the data directory

dir\_ = os.listdir(DATA\_PATH)

# Iterate over each file in the directory

for filename in dir\_:

# Extract the test person ID and condition from the filename

test\_person\_id = filename[:2]

condition = "rest" if filename[-5] == "0" else "stress"

# Read the data from the file

df = pd.read\_csv(DATA\_PATH + "/" + filename)

# Slice the data between 3 and 10 seconds

df = df[(df["Time (s)"] > 3) & (df["Time (s)"] <= 8)].copy()

# Calculate the magnitude

df["magnitude"] = np.sqrt(df.iloc[:, 1] \*\* 2 + df.iloc[:, 2] \*\* 2 + df.iloc[:, 3] \*\* 2)

# Calculate the sampling rate and minimum distance between peaks

time\_diff = df["Time (s)"].diff()

sampling\_rate = 1 / time\_diff.mean()

min\_distance = int(0.3 \* sampling\_rate)

# Find peaks in the magnitude signal

peaks, \_ = find\_peaks(df["magnitude"].values, distance=min\_distance)

# Filter peaks based on the time constraint and distance from mean

filtered\_peaks = [peaks[0]]

mean = df["magnitude"].mean()

std = df["magnitude"].std()

for i in range(1, len(peaks)):

time\_diff = df["Time (s)"].iloc[peaks[i]] - df["Time (s)"].iloc[peaks[i-1]]

magnitude\_diff = np.abs(df["magnitude"].iloc[peaks[i]] - mean)

if time\_diff >= 0.3 and magnitude\_diff >= 2 \* std:

filtered\_peaks.append(peaks[i])

# Count the number of peaks

num\_peaks = len(filtered\_peaks)

# Add the features to the final DataFrame

final\_df.loc[len(final\_df), "id"] = test\_person\_id

final\_df.loc[len(final\_df) - 1, "condition"] = condition

final\_df.loc[len(final\_df) - 1, "standard deviation"] = df.magnitude.std()

final\_df.loc[len(final\_df) - 1, "num\_peaks"] = num\_peaks

# Visualize the raw data with marked peaks (only for test person with id = "01")

if test\_person\_id == "01":

plt.title(f"Visualize Raw Data - Post Slicing (ID: {test\_person\_id}, Condition: {condition})")

plt.xlabel("Time (s)")

plt.ylabel("Magnitude (m/s^2)")

plt.plot(df["Time (s)"], df["magnitude"], label="Data", alpha=0.8)

plt.scatter(df["Time (s)"].iloc[filtered\_peaks], df["magnitude"].iloc[filtered\_peaks], c='red', marker='x', label="Peaks")

plt.legend()

plt.show()

# Calculate the "Calc. HR" column by multiplying the num\_peaks with 12

final\_df["Calc. HR"] = final\_df["num\_peaks"] \* 12

# Reset the index of the final DataFrame

final\_df.reset\_index(drop=True, inplace=True)

# Print the final DataFrame

final\_df

facts = {"01": {"age": 26, "sex": "male", "weight": 88},

"02": {"age": 24, "sex": "male", "weight": 80},

"03": {"age": 27, "sex": "female", "weight": 65},

"04": {"age": 21, "sex": "female", "weight": 76},

"05": {"age": 17, "sex": "male", "weight": 75},

"06": {"age": 23, "sex": "male", "weight": 90},

"07": {"age": 24, "sex": "female", "weight": 54},

"08": {"age": 27, "sex": "female", "weight": 53},

"09": {"age": 30, "sex": "male", "weight": 76},

"10": {"age": 19, "sex": "male", "weight": 84},

"11": {"age": 29, "sex": "female", "weight": 57},

"12": {"age": 18, "sex": "male", "weight": 80},

"13": {"age": 19, "sex": "female", "weight": 65},

"14": {"age": 35, "sex": "male", "weight": 82},

"15": {"age": 22, "sex": "male", "weight": 86},

"16": {"age": 27, "sex": "male", "weight": 91},

"17": {"age": 19, "sex": "female", "weight": 59},

"18": {"age": 33, "sex": "female", "weight": 66},

"19": {"age": 20, "sex": "male", "weight": 83},

"20": {"age": 27, "sex": "male", "weight": 89},

"21": {"age": 28, "sex": "female", "weight": 64},

"22": {"age": 29, "sex": "male", "weight": 84},

"23": {"age": 21, "sex": "female", "weight": 66},

"24": {"age": 28, "sex": "male", "weight": 78},

"25": {"age": 30, "sex": "male", "weight": 77},

"26": {"age": 22, "sex": "male", "weight": 92},

"27": {"age": 28, "sex": "female", "weight": 60},

"28": {"age": 18, "sex": "female", "weight": 68},

"29": {"age": 34, "sex": "male", "weight": 82},

"30": {"age": 30, "sex": "male", "weight": 77},

}

#Prepare a list of the collected features

age\_list = []

sex\_list = []

weight\_list = []

for i in facts:

proband = facts[i]

age\_list.append(proband["age"])

sex\_list.append(proband["sex"])

weight\_list.append(proband["weight"])

# Add the further facts to the final df

for i in facts:

#age

final\_df.loc[final\_df["id"] == i, "age"] = facts[i]["age"]

#sex

final\_df.loc[final\_df["id"] == i, "sex"] = facts[i]["sex"]

#weight

final\_df.loc[final\_df["id"] == i, "weight"] = facts[i]["weight"]

final\_df

# Export the dataframe to a csv file

final\_df = final\_df.sort\_values(by=['id'])

final\_df.to\_csv("0\_final\_df.csv", index = False)

final\_df

#Importing Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

import os

import re

DATA\_PATH = 'Raw\_Data'

#Prepare the data

final\_df = pd.read\_csv("0\_final\_df.csv", dtype={"id": str})

rest\_list\_std = []

stress\_list\_std = []

rest\_list\_HR = []

stress\_list\_HR = []

for i in range(len(final\_df)):

if final\_df.loc[i, "condition"] == "rest":

rest\_list\_std.append(final\_df.loc[i, "standard deviation"])

rest\_list\_HR.append(final\_df.loc[i, "Calc. HR"])

elif final\_df.loc[i, "condition"] == "stress":

stress\_list\_std.append(final\_df.loc[i, "standard deviation"])

stress\_list\_HR.append(final\_df.loc[i, "Calc. HR"])

#Creating Boxplots

data\_std = rest\_list\_std + stress\_list\_std

labels\_std = ['Rest'] \* len(rest\_list\_std) + ['Stress'] \* len(stress\_list\_std)

data\_HR = rest\_list\_HR + stress\_list\_HR

labels\_HR = ['Rest'] \* len(rest\_list\_HR) + ['Stress'] \* len(stress\_list\_HR)

plt.figure(figsize=(10, 6))

# Boxplot of standard deviation

plt.subplot(1, 2, 1)

sns.boxplot(x=labels\_std, y=data\_std, linewidth=1.5, color='lightgray')

sns.stripplot(x=labels\_std, y=data\_std, color='black', size=4, alpha=1)

plt.title('Standard Deviation of the Magnitude', fontsize=16)

plt.xlabel('Condition', fontsize=12)

plt.ylabel('Standard Deviation [m/s^2]', fontsize=12)

# Boxplot of heart rate

plt.subplot(1, 2, 2)

sns.boxplot(x=labels\_HR, y=data\_HR, linewidth=1.5, color='lightgray')

sns.stripplot(x=labels\_HR, y=data\_HR, color='black', size=4, alpha=1)

plt.title('Heart Rate', fontsize=16)

plt.xlabel('Condition', fontsize=12)

plt.ylabel('Heart Rate [bpm]', fontsize=12)

plt.tight\_layout()

plt.show()

# Calculate and print the correlation coefficient

correlation = pd.DataFrame({'Heart Rate': data\_HR, 'Standard Deviation': data\_std}).corr()

print('Correlation between Heart Rate and Standard Deviation:')

print(correlation.loc['Heart Rate', 'Standard Deviation'])

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# Set bigger font size for labels

plt.rcParams.update({'font.size': 14})

# Get the data for both groups and genders

stress\_male = final\_df[(final\_df["condition"] == "stress") & (final\_df["sex"] == "male")]["standard deviation"].values

stress\_female = final\_df[(final\_df["condition"] == "stress") & (final\_df["sex"] == "female")]["standard deviation"].values

rest\_male = final\_df[(final\_df["condition"] == "rest") & (final\_df["sex"] == "male")]["standard deviation"].values

rest\_female = final\_df[(final\_df["condition"] == "rest") & (final\_df["sex"] == "female")]["standard deviation"].values

# Determine the x-axis limits

x\_min = min(np.min(stress\_male), np.min(stress\_female), np.min(rest\_male), np.min(rest\_female))

x\_max = max(np.max(stress\_male), np.max(stress\_female), np.max(rest\_male), np.max(rest\_female))

# Calculate bin edges

bin\_edges = np.linspace(x\_min, x\_max, num=13)

# Create subplots

fig, axes = plt.subplots(2, 2, figsize=(16, 12))

xlabel = "Std of the Magnitude [m/s^2]"

# Plot for Stress group, Male

sns.histplot(stress\_male, bins=bin\_edges, ax=axes[0, 0])

axes[0, 0].set\_title("Stress Group - Male")

axes[0, 0].set\_xlabel(xlabel)

axes[0, 0].set\_ylabel("Frequency")

axes[0, 0].set\_xlim(x\_min, x\_max)

axes[0, 0].set\_ylim(0, 11)

# Plot for Stress group, Female

sns.histplot(stress\_female, bins=bin\_edges, ax=axes[1, 0])

axes[1, 0].set\_title("Stress Group - Female")

axes[1, 0].set\_xlabel(xlabel)

axes[1, 0].set\_ylabel("Frequency")

axes[1, 0].set\_xlim(x\_min, x\_max)

axes[1, 0].set\_ylim(0, 11)

# Plot for Rest group, Male

sns.histplot(rest\_male, bins=bin\_edges, ax=axes[0, 1])

axes[0, 1].set\_title("Rest Group - Male")

axes[0, 1].set\_xlabel(xlabel)

axes[0, 1].set\_ylabel("Frequency")

axes[0, 1].set\_xlim(x\_min, x\_max)

axes[0, 1].set\_ylim(0, 11)

# Plot for Rest group, Female

sns.histplot(rest\_female, bins=bin\_edges, ax=axes[1, 1])

axes[1, 1].set\_title("Rest Group - Female")

axes[1, 1].set\_xlabel(xlabel)

axes[1, 1].set\_ylabel("Frequency")

axes[1, 1].set\_xlim(x\_min, x\_max)

axes[1, 1].set\_ylim(0, 11)

# Adjust the spacing between subplots

plt.subplots\_adjust(wspace=0.3, hspace=0.3)

# Increase the font size of x-axis label

for ax in axes.flatten():

ax.xaxis.label.set\_fontsize(16)

# Display the plots

plt.tight\_layout()

plt.show()

#Histrogram for the standard devation

# Set bigger font size for labels

plt.rcParams.update({'font.size': 14})

# Get the data for both groups

stress = final\_df[final\_df["condition"] == "stress"]["standard deviation"].values

rest = final\_df[final\_df["condition"] == "rest"]["standard deviation"].values

combined\_data = np.concatenate((stress, rest))

# Determine the x-axis limits

x\_min = np.min(combined\_data)

x\_max = np.max(combined\_data)

bin\_edges1 = np.linspace(x\_min, x\_max, num=15)

# Create subplots

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Plot for Stress group

sns.histplot(stress, bins=bin\_edges1, ax=axes[0])

axes[0].set\_title("Stress group")

axes[0].set\_xlabel("Standard Deviation of the magnitude")

axes[0].set\_ylabel("Frequency")

axes[0].set\_ylim(0, 16)

axes[0].set\_xlim(x\_min, x\_max)

# Plot for Rest group

sns.histplot(rest, bins=bin\_edges1, ax=axes[1])

axes[1].set\_title("Rest group")

axes[1].set\_xlabel("Standard Deviation of the magnitude")

axes[1].set\_ylabel("Frequency")

axes[1].set\_ylim(0,16) # hard coded

axes[1].set\_xlim(x\_min, x\_max)

# Adjust the spacing between subplots

plt.subplots\_adjust(wspace=0.3)

# Display the plots

plt.show()

#Boxplots for the heart rate

# Set bigger font size for labels

plt.rcParams.update({'font.size': 14})

# Get the data for both groups

stress\_hr = final\_df[final\_df["condition"] == "stress"]["Calc. HR"].values

rest\_hr = final\_df[final\_df["condition"] == "rest"]["Calc. HR"].values

combined\_data\_hr = np.concatenate((stress\_hr, rest\_hr))

# Determine the x-axis limits

x\_min\_hr = np.min(combined\_data\_hr)

x\_max\_hr = np.max(combined\_data\_hr)

# Create subplots

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Plot for Stress group

sns.histplot(stress\_hr, bins=7, ax=axes[0])

axes[0].set\_title("Stress group")

axes[0].set\_xlabel("Heart Rate [bpm]")

axes[0].set\_ylabel("Frequency")

axes[0].set\_ylim(0,11) # hard coded

axes[0].set\_xlim(x\_min\_hr, x\_max\_hr)

# Plot for Rest group

sns.histplot(rest\_hr, bins=7, ax=axes[1])

axes[1].set\_title("Rest group")

axes[1].set\_xlabel("Heart Rate [bpm]")

axes[1].set\_ylabel("Frequency")

axes[1].set\_ylim(0,11) # hard coded

axes[1].set\_xlim(x\_min\_hr, x\_max\_hr)

# Adjust the spacing between subplots

plt.subplots\_adjust(wspace=0.3)

# Display the plots

plt.show()

#QQPlot

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

# Set bigger font size for labels

plt.rcParams.update({'font.size': 14})

# Get the data for both groups

stress = final\_df[final\_df["condition"] == "stress"]["standard deviation"].values

rest = final\_df[final\_df["condition"] == "rest"]["standard deviation"].values

# Create subplots

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Plot QQ plot for Stress group

stats.probplot(stress, plot=axes[0])

axes[0].set\_title("Stress group")

axes[0].set\_xlabel("Theoretical Quantiles")

axes[0].set\_ylabel("Ordered Values")

# Plot QQ plot for Rest group

stats.probplot(rest, plot=axes[1])

axes[1].set\_title("Rest group")

axes[1].set\_xlabel("Theoretical Quantiles")

axes[1].set\_ylabel("Ordered Values")

# Adjust the spacing between subplots

plt.subplots\_adjust(wspace=0.3)

# Display the plots

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

# Set bigger font size for labels

plt.rcParams.update({'font.size': 14})

# Get the data for both groups

stress\_hr = final\_df[final\_df["condition"] == "stress"]["Calc. HR"].values

rest\_hr = final\_df[final\_df["condition"] == "rest"]["Calc. HR"].values

# Create subplots

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Plot QQ plot for Stress group

stats.probplot(stress\_hr, plot=axes[0])

axes[0].set\_title("Stress group")

axes[0].set\_xlabel("Theoretical Quantiles")

axes[0].set\_ylabel("Ordered Values")

# Plot QQ plot for Rest group

stats.probplot(rest\_hr, plot=axes[1])

axes[1].set\_title("Rest group")

axes[1].set\_xlabel("Theoretical Quantiles")

axes[1].set\_ylabel("Ordered Values")

# Adjust the spacing between subplots

plt.subplots\_adjust(wspace=0.3)

# Display the plots

plt.show()

#Descriptive Statistics

numeric\_columns = ['standard deviation', 'num\_peaks', 'Calc. HR', 'age', 'weight']

# Calculate descriptive statistics

descriptive\_stats = final\_df[numeric\_columns].describe()

# Define the output file path

output\_file\_path = 'descriptive\_stats.txt'

# Export the descriptive statistics as a .txt file

descriptive\_stats.to\_csv(output\_file\_path, sep='\t', float\_format='%.2f')