In [1]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import matplotlib.dates as mdates
import scipy as sp
import seaborn as sns
from scipy import signal
from statsmodels.tsa.stattools import adfuller
```

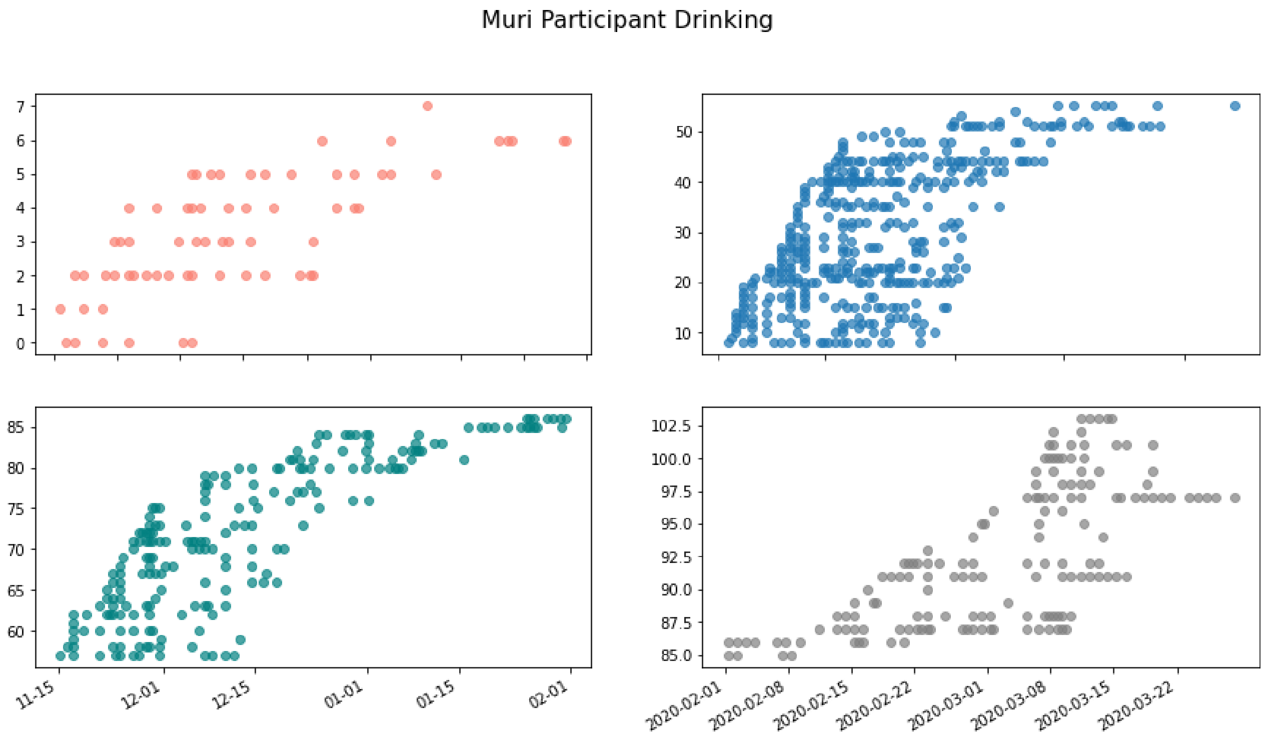
Drinking and Mood Autocorrelation, Stationarity and Cross Correlation.

In [2]:

```
ema = pd.read_csv("/Users/djpassey/Data/Muri/SHINE_EMA_Round1_19May2020.csv", pa
ema['Notification.Time'] = pd.to_datetime(ema['Notification.Time'])
ema['Num_Alcohol'] = ema.Num_Beer + ema.Num_Wine + ema.Num_Liquor
```

In [3]:

```
plt.rcParams["figure.figsize"] = [15, 8]

drinks = ema[ema.HadAlcohol == 1]
unique_ids = drinks.sort_values("Notification.Time")["ID"].unique()
idmapping = {unique_ids[i]:i for i in range(len(unique_ids))}

start_date = drinks["Notification.Time"].min()
end_date = drinks["Notification.Time"].max()
assert np.all(drinks["Notification.Time"] >= start_date)
assert np.all(drinks["Notification.Time"] <= end_date)

cohort1_end = "2019-05-01"
cohort2_start = "2019-05-01"
cohort2_end = "2019-08-01"
cohort3_start = "2019-10-01"
cohort3_end = "2020-02-01"
cohort4_start = "2020-02-01"
cohort4_end = end_date

pid = drinks.ID
times = drinks["Notification.Time"]

plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%M-%d'))
#plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=7))

plt.subplot(2, 2, 1)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%M-%d'))
mask = (times > start_date) & (times < cohort1_end)
ids = [idmapping[id] for id in pid[mask]]
plt.scatter(times[mask], ids, c="salmon", alpha=0.7)
# plt.ylabel("Participant ID")
plt.tick_params(labelbottom=True)

plt.subplot(2, 2, 2)
mask = (times >= cohort2_start) & (times < cohort2_end)
ids = [idmapping[id] for id in pid[mask]]
plt.scatter(times[mask], ids, alpha=0.7)
plt.tick_params(labelbottom=True)

plt.subplot(2, 2, 3)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%m-%d'))
mask = (times >= cohort3_start) & (times < cohort3_end)
ids = [idmapping[id] for id in pid[mask]]
plt.scatter(times[mask], ids, c="teal", alpha=0.7)

# plt.ylabel("Participant ID")
# plt.xlabel("Drink Time")

plt.subplot(2, 2, 4)
mask = (times >= cohort4_start) & (times < cohort4_end)
ids = [idmapping[id] for id in pid[mask]]
plt.scatter(times[mask], ids, c="gray", alpha=0.7)
# plt.xlabel("Drink Time")


plt.suptitle("Muri Participant Drinking", fontsize=16)
plt.gcf().autofmt_xdate()
#plt.tight_layout()
```



In [3]:

```
MORNING = ['FirstMorning', 'Morning']
EVENING = ['Evening']
PVAL = 0.01

def drink_sessions(df, prompt=None):
    if prompt is None:
        ds = df[df["Session.Name"].isin(MORNING+EVENING)]
    if prompt is "morning":
        ds = df[df["Session.Name"].isin(MORNING)]
    if prompt is "evening":
        ds = df[df["Session.Name"].isin(EVENING)]
    ds.fillna({"Num_Alcohol":0, "HadAlcohol":0})
    ds.sort_values("Notification.Time", inplace=True)
    return ds
```

```python
def drink_train(idnum, df, prompt=None):
    drink_notif = df[df.ID == idnum]
    # WARNING: Fills signals with zeros if they didn't respond
    drink_notif = drink_sessions(drink_notif, prompt=prompt)
    drink_notif.sort_values("Notification.Time", inplace=True)
    times = drink_notif["Notification.Time"]
    ndrinks = drink_notif["Num_Alcohol"].fillna(0)
    return times, ndrinks.values

def autocorr(y):
    """ Autocorrelation of a signal. Algorithm taken from:
        https://www.itl.nist.gov/div898/handbook/eda/section3/eda331.htm
    """
    mu = np.mean(y)
    N = len(y)
    auto = np.zeros(N)
    for h in range(N):
        auto[h] = np.sum((y[:N - h] - mu) * (y[h:] - mu)) / N
    var = np.sum((y - mu)**2)/N
    return auto/var

def fourier_transform(t, y):
    """ Take the fourier transform of a signal, rescale and provide a frequency
        for easy plotting
    """
    total_time = float(t[-1] - t[0])
    N = len(y)
    xf = np.arange(N)/ (total_time)
    yf = sp.fft.fft(y) / N
    # Take half of the dfft and multiply by 2 because it is a mirror image
    yf = 2*np.abs(yf[:N//2])
    xf = xf[:N//2]
    return xf, yf

def nans_to_lines(y):
    """ This function fills in nans in time series data by find valid points
    that bookend a given series of nans, then replacing the nans with a line bet
    It also trims leading/trailing nans.
    """
    m = len(y)
    clean_y = np.zeros(m)
    nani = 0
    a = 0
    b = 0
    found_nan = False
    # Count initial nans
    initnan = 0
    if np.isnan(y[0]):
        while np.isnan(y[initnan]):
            initnan += 1
    for i in range(initnan, m):
        if np.isnan(y[i]) and not found_nan:
            a = y[i-1]
            nani = i
            found_nan = True
        if ~np.isnan(y[i]) and found_nan:
            b = y[i]
            clean_y[i] = b
            clean_y[nani:i] = np.linspace(a, b, i - nani + 2)[1:-1]
            found_nan = False
        else:
            clean_y[i] = y[i]
    return clean_y[initnan:nani] # Remove training nans

COLORS = ["salmon", "teal", "grey", "green"]

def hist_templ(
    *data,
    xlab="Value",
    ylab="Frequency",
    title="Histogram",
    label=["Morning Prompt", "Evening Prompt"],
    bins=20,
    alpha=0.6
):
    for i, x in enumerate(data):
        plt.hist(x, bins=bins, color=COLORS[i], alpha=0.6, label=label[i])
    plt.legend()
    plt.ylabel(ylab)
    plt.xlabel(xlab)
    p = plt.title(title)
    return p

def lines_templ(
    xdata,
    *ydata,
    xlab="Value",
    ylab="Frequency",
    title="Lines",
    label=["Morning Prompt", "Evening Prompt"],
    alpha=0.6
):
    for i, y in enumerate(ydata):
        plt.plot(xdata, y, alpha=alpha, color=COLORS[i], label=label[i], lw=3)
    plt.legend()
    plt.ylabel(ylab)
    plt.xlabel(xlab)
    p = plt.title(title)
    return p
```

Extract Mood and Drinking Time Series

We separate by morning and evening prompt so that there are 24 hours between each datapoint

In [4]:

```
timeseries = {
    "id" : [],
    "drink.morning" : [],
    "drink.evening" : [],
    "mood.morning" : [],
    "mood.evening" : []
}

time_series_keys = ["drink.morning", "drink.evening", "mood.morning", "mood.even

# Separate morning and evening prompts
morn = drink_sessions(ema, prompt="morning")
eve = drink_sessions(ema, prompt="evening")
```

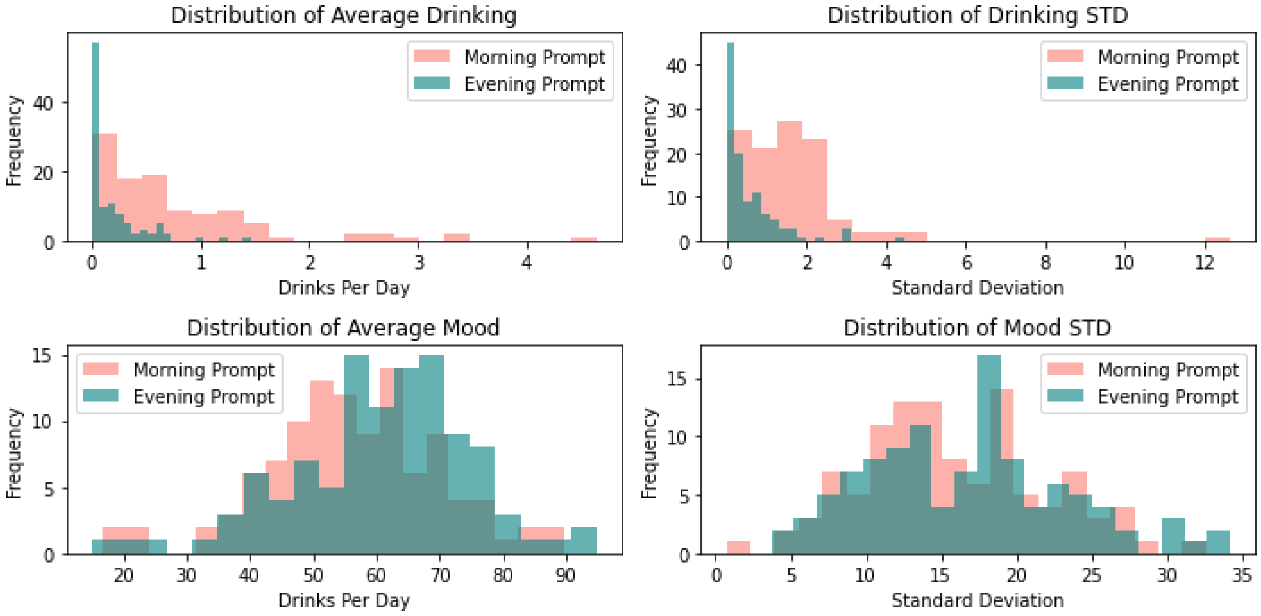
```
# Participants to exclude
EXCLUDE_ID = [52927]

# Drinking and Mood data
drinkmorn = tuple()
drinkeve = tuple()
moodmorn = tuple()
moodeve = tuple()
for idnum in ema.ID.unique():
    if idnum not in EXCLUDE_ID:
        # ID numbers
        timeseries["id"].append(idnum)
        morn_id = morn[morn.ID == idnum]
        eve_id = eve[eve.ID == idnum]
        # Number of drinks time series
        drinkmorn += (morn_id["Num_Alcohol"].fillna(0).values,)
        drinkeve += (eve_id["Num_Alcohol"].fillna(0).values,)
        # Positive mood time series
        morn_m = morn_id.PositiveMood
        moodmorn += (morn_m.fillna(np.mean(morn_m)).values,)
        eve_m = eve_id.PositiveMood
        moodeve += (eve_m.fillna(np.mean(eve_m)).values,)

timeseries["drink.morning"] = np.vstack(drinkmorn)
timeseries["drink.evening"] = np.vstack(drinkeve)
timeseries["mood.morning"] = np.vstack(moodmorn)
timeseries["mood.evening"] = np.vstack(moodeve)

for key in time_series_keys:
    Yf = tuple()
    for ts in timeseries[key]:
        x = np.arange(28)
        xf, yf = fourier_transform(x, ts)
        Yf += (yf,)
    timeseries[key + ".ft"] = np.vstack(Yf)
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:13: Setti
ngWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  del sys.path[0]
```

In [5]:

```
plt.rcParams["figure.figsize"] = [10, 5]

# Drinking Data

plt.subplot(2,2, 1) # Mean drinking
dm_mean = np.mean(timeseries["drink.morning"], axis=1)
de_mean = np.mean(timeseries["drink.evening"], axis=1)
hist_templ(dm_mean, de_mean, xlab="Drinks Per Day", title="Distribution of Avera

plt.subplot(2,2, 2) # Drinking STD
dm_mean = np.std(timeseries["drink.morning"], axis=1)
de_mean = np.std(timeseries["drink.evening"], axis=1)
hist_templ(dm_mean, de_mean, xlab="Standard Deviation", title="Distribution of D

# Mood Data
```

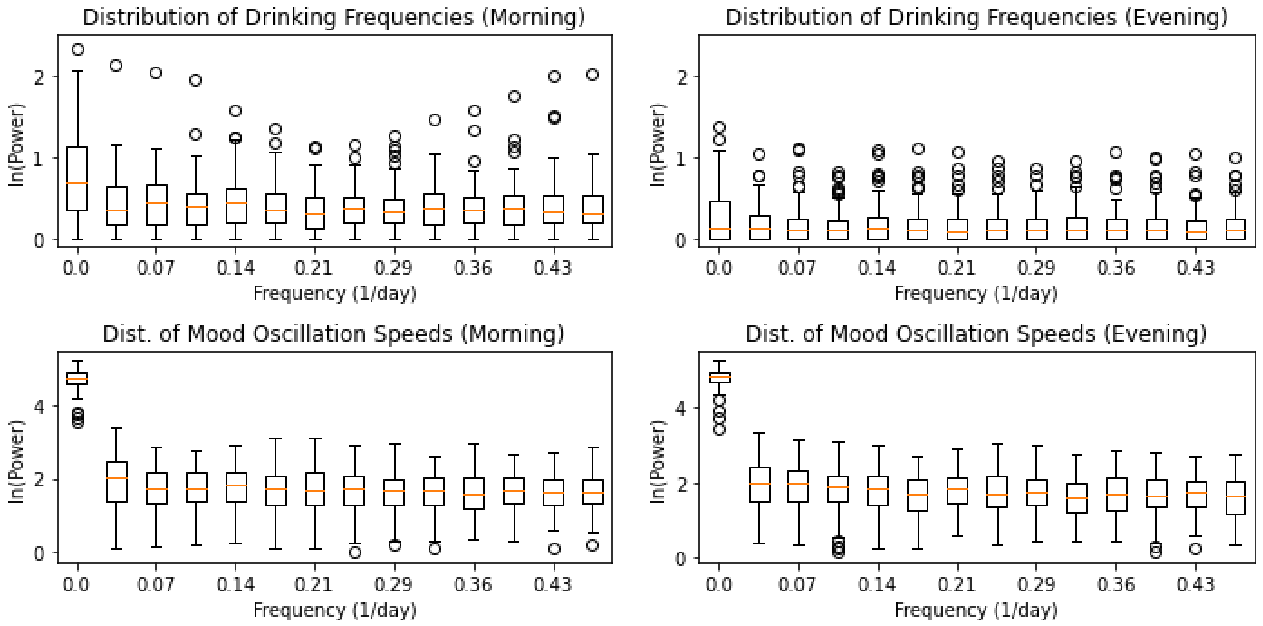
```
plt.subplot(2,2, 3) # Mean mood
dm_mean = np.mean(timeseries["mood.morning"], axis=1)
de_mean = np.mean(timeseries["mood.evening"], axis=1)
hist_templ(dm_mean, de_mean, xlab="Drinks Per Day", title="Distribution of Avera

plt.subplot(2,2, 4) # Mood std
dm_mean = np.std(timeseries["mood.morning"], axis=1)
de_mean = np.std(timeseries["mood.evening"], axis=1)
hist_templ(dm_mean, de_mean, xlab="Standard Deviation", title="Distribution of M

plt.tight_layout()
plt.show()
```



In [6]:

```
plt.rcParams["figure.figsize"] = [10, 5]

# Frequency axis (For fourier transform)
total_time = 28.0
N = 28
xf = np.arange(N)/ (total_time)
xf = xf[:N//2]

# Drinking Data

plt.subplot(2, 2, 1) # Fourier transform morning
plt.boxplot(np.log(timeseries["drink.morning.ft"] + 1))
plt.xticks(np.arange(1,15,2), np.round(xf[::2],2))
plt.title("Distribution of Drinking Frequencies (Morning)")
plt.ylabel("ln(Power)")
plt.ylim(-.1, 2.50)
plt.xlabel("Frequency (1/day)")

plt.subplot(2, 2, 2) # Fourier transform evening
plt.boxplot(np.log(timeseries["drink.evening.ft"] + 1))
plt.xticks(np.arange(1,15,2), np.round(xf[::2],2))
plt.title("Distribution of Drinking Frequencies (Evening)")
plt.ylabel("ln(Power)")
plt.ylim(-.1, 2.50)
plt.xlabel("Frequency (1/day)")
```

```
# Mood data

plt.subplot(2, 2, 3) # Fourier transform morning
plt.boxplot(np.log(timeseries["mood.morning.ft"] + 1))
plt.xticks(np.arange(1,15,2), np.round(xf[::2],2))
plt.title("Dist. of Mood Oscillation Speeds (Morning)")
plt.ylabel("ln(Power)")
#plt.ylim(-.1, 2.50)
plt.xlabel("Frequency (1/day)")

plt.subplot(2, 2, 4) # Fourier transform evening
plt.boxplot(np.log(timeseries["mood.evening.ft"] + 1))
plt.xticks(np.arange(1,15,2), np.round(xf[::2],2))
plt.title("Dist. of Mood Oscillation Speeds (Evening)")
plt.ylabel("ln(Power)")
#plt.ylim(-.1, 2.50)
plt.xlabel("Frequency (1/day)")

plt.tight_layout()
plt.show()
```



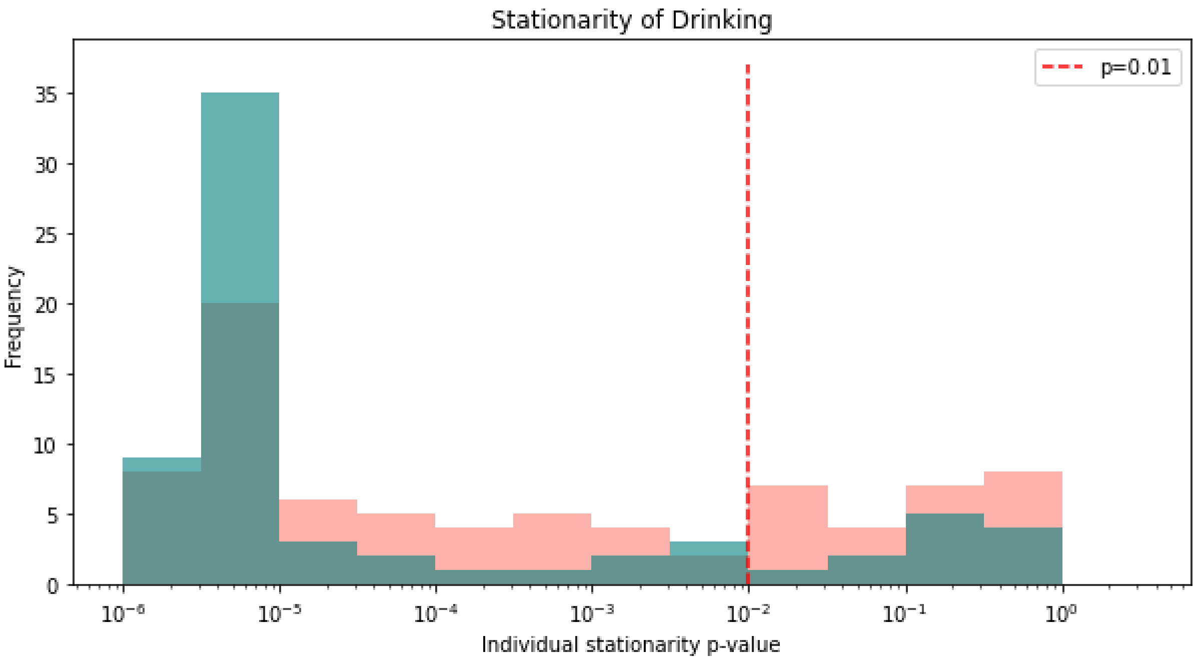
In [ ]:

```
for key in time_series_keys:
    ts = timeseries[key]
    pvals = [adfuller(x)[1] for x in ts]
    timeseries[key + ".adfuller"] = np.array(pvals)
```

Stationarity of Drinking

In [8]:

```
bins = 10**(np.arange(-6.0, 1, 0.5))
dmp = timeseries["drink.morning.adfuller"]
dep = timeseries["drink.evening.adfuller"]
plt.xscale('log')
plt.hist(dmp, bins=bins, color=COLORS[0], alpha=0.6)
plt.hist(dep, bins=bins, color=COLORS[1], alpha=0.6)
plt.plot(np.ones(38)*PVAL, np.arange(0,38), "--", c="r", label=f"p={PVAL}")
plt.xlabel("Individual stationarity p-value")
plt.ylabel("Frequency")
plt.title("Stationarity of Drinking")
plt.legend()
plt.show()

adf_sig = lambda x: np.sum(x < PVAL)
print(f"{adf_sig(dmp)} / 108 Participants show stationarity in drinking (Morning
print(f"{adf_sig(dep)} / 108 Participants show stationarity in drinking (Evening
```

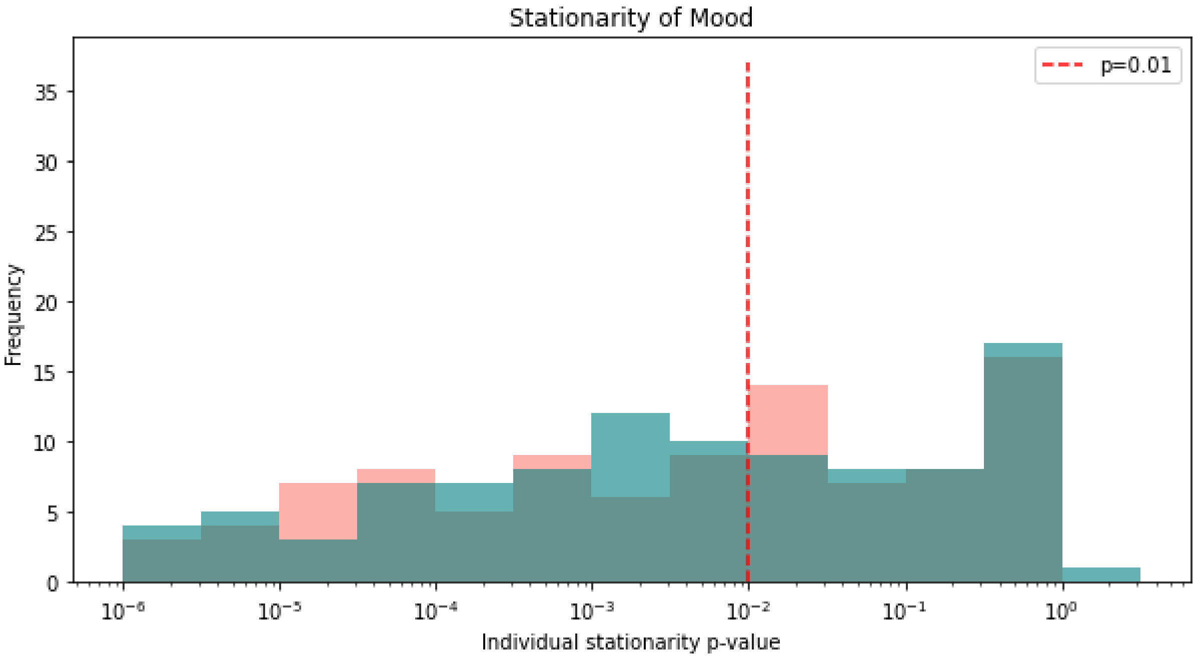


```
72 / 108 Participants show stationarity in drinking (Morning)
67 / 108 Participants show stationarity in drinking (Evening)
```

Stationarity of Mood

In [10]:

```
bins = 10**(np.arange(-6.0, 1, 0.5))
mmp = timeseries["mood.morning.adfuller"]
mep = timeseries["mood.evening.adfuller"]

plt.xscale('log')
plt.hist(mmp, bins=bins, color=COLORS[0], alpha=0.6)
plt.hist(mep, bins=bins, color=COLORS[1], alpha=0.6)
plt.title("Stationarity of Mood")
plt.xlabel("Individual stationarity p-value")
plt.ylabel("Frequency")
plt.plot(np.ones(38)*PVAL, np.arange(0,38), "--", c="r", label=f"p={PVAL}")
plt.legend()
plt.show()

print(f"{adf_sig(mmp)} / 108 Participants show stationarity in mood (Morning)")
print(f"{adf_sig(mep)} / 108 Participants show stationarity in mood (Evening)")
```



```
63 / 108 Participants show stationarity in mood (Morning)
65 / 108 Participants show stationarity in mood (Evening)
```

Autocorrelation of Drinking

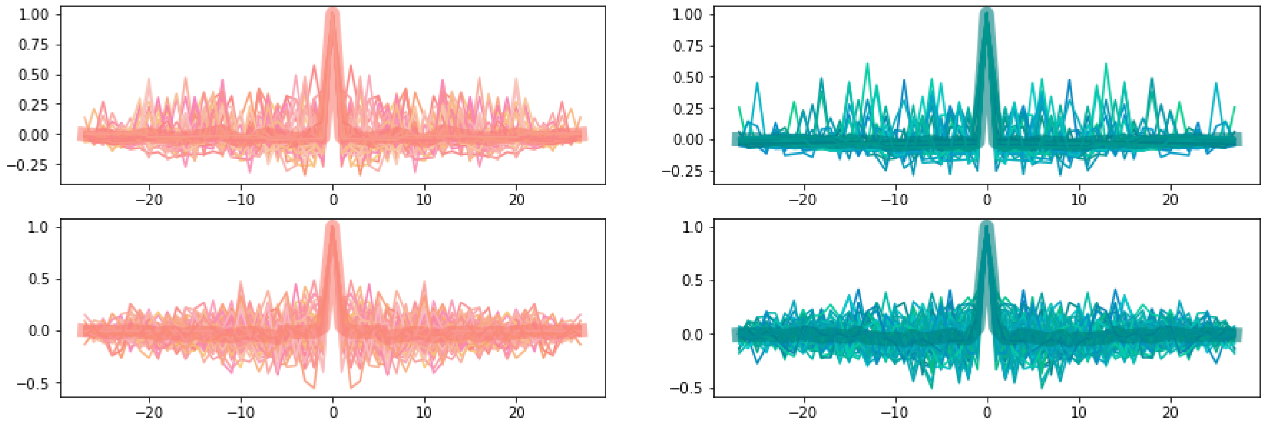
In [43]:

```
def crosscorr(x, y):
    w = x - np.mean(x)
    z = y - np.mean(y)
    cc = signal.correlate(w, z)
    cc /= np.max(np.abs(cc))
    return cc

def autocorr(x):
    return crosscorr(x, x)
```

In [47]:

```
plt.rcParams["figure.figsize"] = [15, 5]
lags = np.arange(-27, 28)

plt.subplot(2,2,1)
mu_mm_ac = np.zeros(55)
n = 0
salmon = np.array([250,131,117]) / 256
for x, p in zip(timeseries["drink.morning"], timeseries["drink.morning.adfuller"

    if p < 0.01:
        ac = autocorr(x)
        mu_mm_ac += ac
        n +=1
        jitter = np.random.rand(3) * 0.3
        jitter[0] = 0
        plt.plot(lags, ac, c=salmon + jitter)
plt.plot(lags, mu_mm_ac/n, lw=10, alpha=0.6, c="salmon")

plt.subplot(2,2,2)
mu_em_ac = np.zeros(55)
n = 0
```

```
teal = np.array([4,128,128]) / 256
for x, p in zip(timeseries["drink.evening"], timeseries["drink.evening.adfuller"

    if p < 0.01:
        ac = autocorr(x)
        mu_em_ac += ac
        n +=1
        jitter = np.random.rand(3) * 0.3
        jitter[0] = 0
        plt.plot(lags, ac, c=teal + jitter)
plt.plot(lags, mu_em_ac/n, lw=10, alpha=0.6, c="teal")

plt.subplot(2,2,3)
mu_mm_ac = np.zeros(55)
n = 0
salmon = np.array([250,131,117]) / 256
for x, p in zip(timeseries["mood.morning"], timeseries["mood.morning.adfuller"])

    if p < 0.01:
        ac = autocorr(x)
        mu_mm_ac += ac
        n +=1
        jitter = np.random.rand(3) * 0.3
        jitter[0] = 0
        plt.plot(lags, ac, c=salmon + jitter)
plt.plot(lags, mu_mm_ac/n, lw=10, alpha=0.6, c="salmon")

plt.subplot(2,2,4)
mu_em_ac = np.zeros(55)
n = 0
teal = np.array([4,128,128]) / 256
for x, p in zip(timeseries["mood.evening"], timeseries["mood.evening.adfuller"])

    if p < 0.01:
        ac = autocorr(x)
        mu_em_ac += ac
        n +=1
        jitter = np.random.rand(3) * 0.3
        jitter[0] = 0
        plt.plot(lags, ac, c=teal + jitter)
plt.plot(lags, mu_em_ac/n, lw=10, alpha=0.6, c="teal")

plt.show()
```



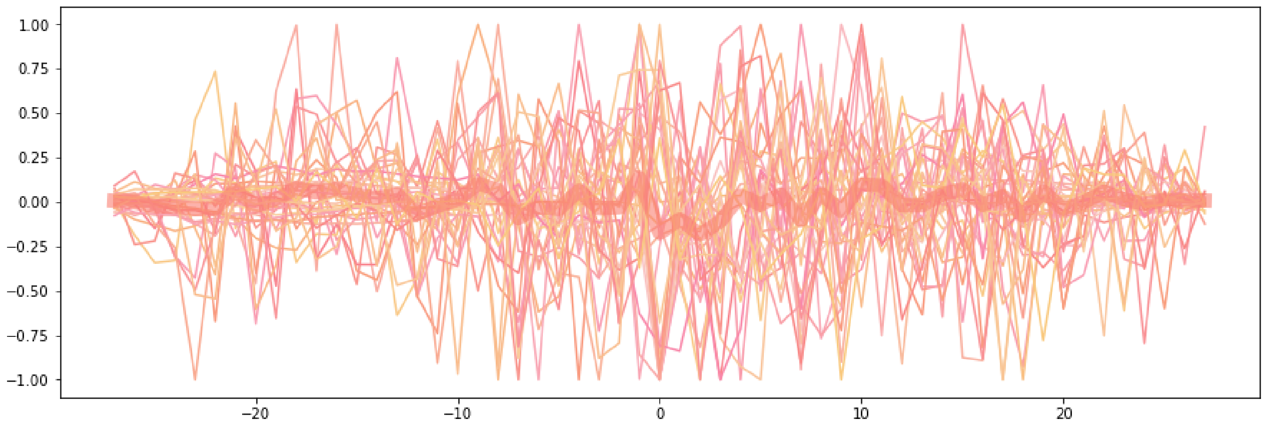
Cross Correlation of Mood and Drinking

In [49]:

```
mu_cc = np.zeros(55)
n = 0

for m, mp, d, dp in zip(timeseries["mood.morning"], timeseries["mood.morning.adf

    if (mp < 0.01) and (dp < 0.01):
        cc = crosscorr(m, d)
        mu_cc += cc
        n +=1
        jitter = np.random.rand(3) * 0.3
        jitter[0] = 0
        plt.plot(lags, cc, c=salmon + jitter)
plt.plot(lags, mu_cc/n, lw=10, alpha=0.6, c="salmon")
```

Out[49]:

```
[<matplotlib.lines.Line2D at 0x16759b0d0>]
```

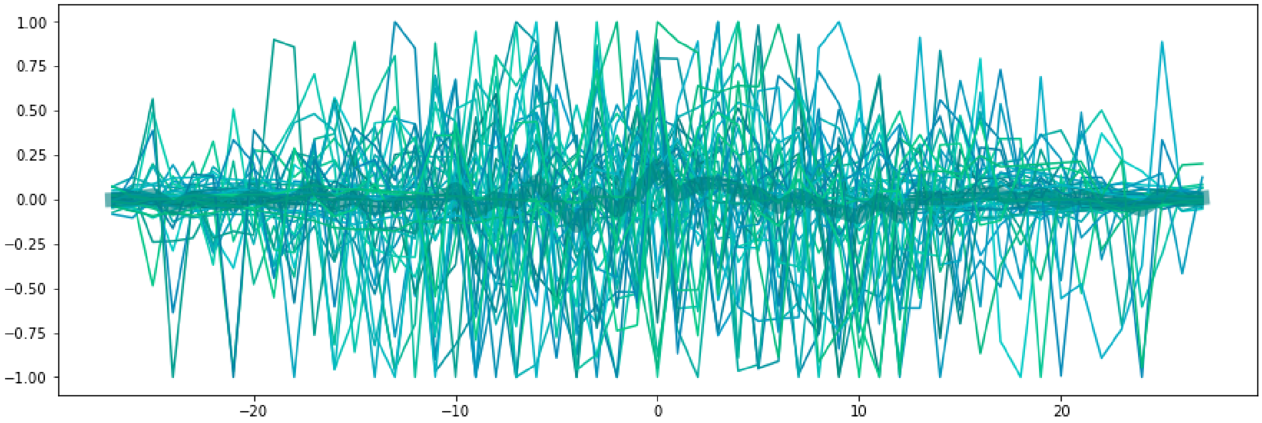


In [52]:

```
mu_cc = np.zeros(55)
n = 0

for m, mp, d, dp in zip(timeseries["mood.evening"], timeseries["mood.evening.adf

    if (mp < 0.01) and (dp < 0.01):
        cc = crosscorr(m, d)
        mu_cc += cc
        n +=1
        jitter = np.random.rand(3) * 0.3
        jitter[0] = 0
        plt.plot(lags, cc, c=teal + jitter)
plt.plot(lags, mu_cc/n, lw=10, alpha=0.6, c="teal")
```

Out[52]:

```
[<matplotlib.lines.Line2D at 0x16750ed50>]
```



```
In [ ]:
```