# James Cross - Assignment 03 Task 01

## Setup

## 1 - Wide vs Long

**wide**
A single column representing a group of variables that each have columns
Kind of like having a folder to have extra files in
This makes it easier to read and display
Makes it harder or slower to operate with

**long**
Each variable is represented Kind of like having all files in the same directory without organization
More difficult to read, easier to operate with

```
# .plot() is used to create line graph of the data points
wide_df.plot(
    x='date', y=['TMAX', 'TMIN', 'TOBS'], figsize=(15, 5),
    title='Temperature in NYC in October 2018'
).set_ylabel('Temperature in Celsius')
# plt.show() uses matlab to display it
plt.show()
```

```
import seaborn as sns
# seaborn is better than pandas for displaying long format data
sns.set(rc={'figure.figsize': (15, 5)}, style='white')

ax = sns.lineplot(
    data=long_df, x='date', y='value', hue='datatype'
)
ax.set_ylabel('Temperature in Celsius')
ax.set_title('Temperature in NYC in October 2018')
plt.show()
```

```
# we learn more about seaborn in chap_06
sns.set(
    rc={'figure.figsize': (20, 10)}, style='white', font_scale=2
)

g = sns.FacetGrid(long_df, col='datatype', height=10)
g = g.map(plt.plot, 'date', 'value')
g.set_titles(size=25)
g.set_xticklabels(rotation=45)
plt.show()
```

## 2 - Cleaning Data

```
# some websites make API that is better way of accessing data
# mostly either free or freemium
# altnerative to webscraping which is slow and resource heavy for both parties
response = make_request('datasets', {'startdate': '2018-10-01'})

response.status_code
response.ok

# API results is given (mostly/always?) as a JSON which is easy to navigate
payload = response.json()
payload.keys()

# various methods of getting data out of the JSON
payload['metadata']
payload['results'][0].keys()
[(data['id'], data['name']) for data in payload['results']]

# get data category id
response = make_request(
    'datacategories', payload={'datasetid': 'GHCND'}
)
response.status_code

response.json()['results']

# get data type id
response = make_request(
    'datatypes',
    payload={
        'datacategoryid': 'TEMP',
        'limit': 100
    }
)
response.status_code

[(datatype['id'], datatype['name']) for datatype in response.json()['results']][-5:] # look
```

I liked the way the book showed that data in the real world can be messy, and how something
that is supposed to be there isn't always. Getting the mindset of thinking on the fly is useful, like swapping the location from Central Park to LaGuardia.

## 3 - Cleaning Data

```
df.rename(
    columns={
        'value': 'temp_C',
        'attributes': 'flags'
    }, inplace=True
    # inplace is useful for making changes in the main dataframe instead of making a new one
)

df.loc[:,'date'] = pd.to_datetime(df.date)
# data isn't always in the ideal format when gotten from somewhere else
# changing dates from objects to datetime lets it be observed in better ways
df.date.describe(datetime_is_numeric=True)
# like changing the timezome
pd.date_range(start='2018-10-25', periods=2, freq='D').tz_localize('EST')

df = df.assign(
    # i dont like how this was done. they have for example 64.94 round to 64 after using as
    # it would be better to do lambda x: round(x.temp_F).astype(int)
    date=lambda x: pd.to_datetime(x.date),
    temp_C_whole=lambda x: x.temp_C.astype('int'),
    temp_F=lambda x: (x.temp_C * 9/5) + 32,
    temp_F_whole=lambda x: x.temp_F.astype('int')
)

df[df.datatype == 'TMAX'].sort_values(by=['temp_C', 'date'], ascending=[False, True]).head(1
# adding sorting rules to values is useful for handling ties and easier selection of preferr
# in this example, the max temp in C, then the date it happened

df[df.datatype == 'TAVG'].nlargest(n=10, columns='temp_C')
# alternative method of same goal, just depends on usecase

df.nsmallest(n=5, columns=['temp_C', 'date'])
# or the inverse

# Forward fill ro fill up empty data
sp.reindex(bitcoin.index, method='ffill').head(10)\
    .assign(day_of_week=lambda x: x.index.day_name())

import numpy as np

sp_reindexed = sp.reindex(bitcoin.index).assign(
    volume=lambda x: x.volume.fillna(0), # put 0 when market is closed
    close=lambda x: x.close.fillna(method='ffill'), # carry this forward
    # take the closing price if these aren't available
    open=lambda x: np.where(x.open.isnull(), x.close, x.open),  # Use 'close' price if 'open
    high=lambda x: np.where(x.high.isnull(), x.close, x.high),  # Use 'close' price if 'high
```

```
    low=lambda x: np.where(x.low.isnull(), x.close, x.low)  # Use 'close' price if 'low' is
)

# Add a new column for the day of the week and show first 10 rows
sp_reindexed.head(10).assign(
    day_of_week=lambda x: x.index.day_name()  # Add 'day_of_week' column based on index valu
)
```

## 4 - Reshaping Data

```
import pandas as pd

long_df = pd.read_csv(
    'data/long_data.csv', usecols=['date', 'datatype', 'value']
).rename(
    columns={'value': 'temp_C'} # rename 'vallue' to 'temp_C'
).assign(
    date=lambda x: pd.to_datetime(x.date),  # new column date, but in datetime object
    temp_F=lambda x: (x.temp_C * 9/5) + 32  # new collumn converting temp_C to temp_F
)

long_df.set_index('date').head(6).T # index to date and transpose (swap row and column)

pivoted_df = long_df.pivot( #pivot around the date using temp_C
    index='date', columns='datatype', values='temp_C'
)

# create a new df with multiple indexes, then confirm there is multiple
multi_index_df = long_df.set_index(['date', 'datatype'])
multi_index_df.head().index

extra_data = long_df.append([{
    'datatype': 'TAVG',     # make a new row TAVG with the below values
    'date': '2018-10-01',
    'temp_C': 10,
    'temp_F': 50
}]).set_index(['date', 'datatype']).sort_index()

extra_data.unstack().head() # everywhere else has NaN for TAVG because it does not exist

extra_data.unstack(fill_value=-40).head() # the data can be filled in however

melted_df = wide_df.melt(   # convert wide to long format
    id_vars='date',         # use date as column identifier
    value_vars=['TMAX', 'TMIN', 'TOBS'],    # what to melt
    value_name='temp_C',    # new column for the melted values
    var_name='measurement'  # new column for melted variable names
```

```
)

stacked_series = wide_df.stack()    # kind of melt but not as good. Series

stacked_df = stacked_series.to_frame('values') # same thing but as a df

stacked_df.index.set_names(['date', 'datatype'], inplace=True)
# set name for the date datatype instead of defaults (which the datatype one was empty)
# inplace = add to the 'main' df instead of making a new one
```

## 5 - Handling Data Issues

```
# troubleshooting steps if data is looking weird
df.head()
df.describe()
df.info()

contain_nulls = df[
    df.SNOW.isna() | df.SNWD.isna() | df.TOBS.isna()
    | df.WESF.isna() | df.inclement_weather.isna()
]
contain_nulls.shape[0]  # there is a lot of empty data points

# Does not work:
df[df.inclement_weather == 'NaN'].shape[0]
df[df.inclement_weather == np.nan].shape[0]
# Proper method of finding nulls
df[df.inclement_weather.isna()].shape[0]

# or find infinite
df[df.SNWD.isin([-np.inf, np.inf])].shape[0]

# 1. make the date a datetime
df.date = pd.to_datetime(df.date)

# 2. save this information for later
station_qm_wesf = df[df.station == '?'].drop_duplicates('date').set_index('date').WESF

# 3. sort ? to the bottom
df.sort_values('station', ascending=False, inplace=True)

# 4. drop duplicates based on the date column keeping the first occurrence
# which will be the valid station if it has data
df_deduped = df.drop_duplicates('date')

# 5. remove the station column because we are done with it
```

```
df_deduped = df_deduped.drop(columns='station').set_index('date').sort_index()

# 6. take valid station's WESF and fall back on station ? if it is null
df_deduped = df_deduped.assign(
    WESF=lambda x: x.WESF.combine_first(station_qm_wesf)
)

df_deduped.shape

# drop any row with a null value
df_deduped.dropna().shape
# only drop a row if ALL values are null
df_deduped.dropna(how='all').shape
# fill null values
df_deduped.loc[:,'WESF'].fillna(0, inplace=True)

# use lambda to replace mistaken values with NaN
df_deduped = df_deduped.assign(
    TMAX=lambda x: x.TMAX.replace(5505, np.nan),
    TMIN=lambda x: x.TMIN.replace(-40, np.nan),
)

df_deduped.assign( # forward fill (with previous values)
    TMAX=lambda x: x.TMAX.fillna(method='ffill'),
    TMIN=lambda x: x.TMIN.fillna(method='ffill')
).head()

df_deduped.assign(  # replace NaN with 0 in SNWD value
    SNWD=lambda x: np.nan_to_num(x.SNWD)
).head()

df_deduped\
    .reindex(pd.date_range('2018-01-01', '2018-12-31', freq='D'))\
    .apply(lambda x: x.interpolate())\
    .head(10)   # with missing data, average the previous and next values to make new one
```