Chapter 04, Task 01

Weather Data Collection

```
# code previous to this was building the datafrme that weve seen many times before
import sqlite3
# dataframes can be written directly to a databse which can be accessed/stored in
# different ways than a typical dataframe
with sqlite3.connect('data/weather.db') as connection:
    df.to_sql('weather', connection, index=False, if_exists='replace')
```

```
# making a new request and getting the below data
stations = pd.DataFrame(response.json()['results'])[['id', 'name', 'latitude', 'longitude',
stations.to_csv('data/weather_stations.csv', index=False)
# then merging it into the same file as a different table (stations instead of weather)
with sqlite3.connect('data/weather.db') as connection:
    stations.to_sql(
        'stations', connection, index=False, if_exists='replace'
    )
```

Querying and MErging

```
# getting info about snow  days in the nyc weather csv
weather = pd.read_csv('data/nyc_weather_2018.csv')
snow_data = weather.query('datatype == "SNOW" and value > 0 and station.str.contains("US1NY"
snow_data.head()
```

```
# the same way of doing that, but using the sql table instead of pd dataframe
with sqlite3.connect('data/weather.db') as connection:
    snow_data_from_db = pd.read_sql(
        'SELECT * FROM weather WHERE datatype == "SNOW" AND value > 0 and station LIKE "%US1
        connection
    )
```

```
# mergeing 'station_info' and 'weather'
# By default, `merge()` performs an inner join
inner_join = weather.merge(station_info, left_on='station', right_on='id')
inner_join.sample(5, random_state=0)
# station and id columns are the same after this merge. resolve by:
weather.merge(station_info.rename(dict(id='station'), axis=1), on='station').sample(5, rando

# because the data is not exactlly 1:1, some gets lost on an inner join. try left or right
left_join = station_info.merge(weather, left_on='id', right_on='station', how='left')
right_join = weather.merge(station_info, left_on='station', right_on='id', how='right')

right_join[right_join.datatype.isna()].head()
```

Data meanings:

```
- `PRCP`: precipitation in millimeters
- `SNOW`: snowfall in millimeters
- `SNWD`: snow depth in millimeters
- `TMAX`: maximum daily temperature in Celsius
- `TMIN`: minimum daily temperature in Celsius
- `TOBS`: temperature at time of observation in Celsius
- `WESF`: water equivalent of snow in millimeters
```

```
# drop dups and drop a column
dirty_data = pd.read_csv(
    'data/dirty_data.csv', index_col='date'
).drop_duplicates().drop(columns='SNWD')

# drop some extra columns
valid_station = dirty_data.query('station != "?"').drop(columns=['WESF', 'station'])
station_with_wesf = dirty_data.query('station == "?"').drop(columns=['station', 'TOBS', 'TM

# merge once cleaned up a bit
valid_station.merge(
    station_with_wesf, how='left', left_index=True, right_index=True
).query('WESF > 0').head()
```

Dataframe Operations

```
Data meanings:
- `AWND`: average wind speed
- `PRCP`: precipitation in millimeters
- `SNOW`: snowfall in millimeters
- `SNWD`: snow depth in millimeters
- `TMAX`: maximum daily temperature in Celsius
- `TMIN`: minimum daily temperature in Celsius
```

```
weather = pd.read_csv('data/nyc_weather_2018.csv', parse_dates=['date'])
fb = pd.read_csv('data/fb_2018.csv', index_col='date', parse_dates=True)

fb.assign(
    abs_z_score_volume=lambda x: \
        x.volume.sub(x.volume.mean()).div(x.volume.std()).abs()
).query('abs_z_score_volume > 3')
# make zscore column and query by zscore > 3

# in 2018 the stock price never had a low above 215
(fb > 215).any()
# OHLC were all had atleast one day at 215 or less
(fb > 215).all()

# split volume into 3 equal range groups
```

```
volume_binned = pd.cut(fb.volume, bins=3, labels=['low', 'med', 'high'])
volume_binned.value_counts()    # show counts of each
fb[volume_binned == 'high'].sort_values('volume', ascending=False)  # show highs

# not worrying about data viz until that chapter

fb.apply(   # compare vectorize to applymap
    lambda x: np.vectorize(lambda y: len(str(np.ceil(y))))(x)
).astype('int64').equals(
    # add function to each element in the 'fb' df without using np.vectorize
    fb.applymap(lambda x: len(str(np.ceil(x))))
)

# find rolling 3 day mean, and display the first 7 rows and the first 6 columns of the resu
central_park_weather.loc['2018-10'].rolling('3D').mean().head(7).iloc[:,:6]

# using pipe to compare to result of rolling
fb.pipe(pd.DataFrame.rolling, '20D').mean().equals(fb.rolling('20D').mean())
```

Aggregations

```
fb = pd.read_csv('data/fb_2018.csv', index_col='date', parse_dates=True).assign(
    trading_volume=lambda x: pd.cut(x.volume, bins=3, labels=['low', 'med', 'high']))

weather = pd.read_csv('data/weather_by_station.csv', index_col='date', parse_dates=True)
# add 2 digits after the decimal
pd.set_option('display.float_format', lambda x: '%.2f' % x)

# group by trading_volume and agg the close to find min maxx mean
fb.groupby('trading_volume')['close'].agg(['min', 'max', 'mean'])

# toll up datetimes in index to find quarter total of precipitation per year
weather.query('datatype == "PRCP"').groupby(
    ['station_name', pd.Grouper(freq='Q')]
).sum().unstack().sample(5, random_state=1)

weather.groupby('station_name').filter( # station names with "NY US" in them
    lambda x: x.name.endswith('NY US')
).query('datatype == "SNOW"').groupby('station_name').sum().squeeze() # aggregate and make
# trading_volume column with low med high columns
fb.pivot_table(columns='trading_volume')
# low med high rows with trading_volume as index
fb.pivot_table(index='trading_volume')

pd.crosstab(
```

```
    index=fb.trading_volume,
    columns=fb.index.month,
    colnames=['month'],
    normalize='columns'
)
```

Time Series

```
fb = pd.read_csv('data/fb_2018.csv', index_col='date', parse_dates=True).assign(
    trading_volume=lambda x: pd.cut(x.volume, bins=3, labels=['low', 'med', 'high'])
)

# time based filtering
fb['2018-10-11':'2018-10-15']
# returns True
fb.loc['2018-q1'].equals(fb['2018-01':'2018-03'])

fb_reindexed = fb.reindex(pd.date_range('2018-01-01', '2018-12-31', freq='D'))
# returns True
fb_reindexed.first('1D').isna().squeeze().all()
# returns Timestamp('2018-01-02 00:00:00', freq='D')
fb_reindexed.loc['2018-Q1'].first_valid_index()

# per minute
stock_data_per_minute = pd.read_csv(
    'data/fb_week_of_may_20_per_minute.csv', index_col='date', parse_dates=True,
    date_parser=lambda x: pd.to_datetime(x, format='%Y-%m-%d %H-%M')
)
# groups it per day
stock_data_per_minute.groupby(pd.Grouper(freq='1D')).agg({
    'open': 'first',
    'high': 'max',
    'low': 'min',
    'close': 'last',
    'volume': 'sum'
})
# @ set time per day in set
stock_data_per_minute.at_time('9:30')

# returns True
(
    fb.drop(columns='trading_volume')
    - fb.drop(columns='trading_volume').shift()
).equals(
    fb.drop(columns='trading_volume').diff()
)
```

```
# find the difference between day to day
fb.drop(columns='trading_volume').diff().head()

with sqlite3.connect('data/stocks.db') as connection:
    fb_prices = pd.read_sql(
        'SELECT * FROM fb_prices', connection,
        index_col='date', parse_dates=['date']
    )
    aapl_prices = pd.read_sql(
        'SELECT * FROM aapl_prices', connection,
        index_col='date', parse_dates=['date']
    )

# FB is by minute and AAPL is by second
fb_prices.index.second.unique()
aapl_prices.index.second.unique()
```