5.1 - DATA WRANGLING with PANDAS

## **5.1.1 - Pandas Foundation**

**Chapter-1: Data Ingestion and Inspection**

|  |  |  |
| --- | --- | --- |
| Review of Pandas Data Frames | | |
| Indexes and columns | **type()** | type(df)  type(df.columns)  type(df.index) |
| **.shape** | df.shape |
| **.columns** | df.columns |
| **.index** | df.index |
| Slicing | **.iloc[row, column]** | df.iloc[5:, :] |
|  | **.head()** | df.head() |
| **.tail()** | df.tail() |
| **.info()** | df.info() |
| Broadcasting | **np.nan** | df.iloc[::3, -1] = np.nan  # Assigning scalar value to column slice broadcasts value to each row. |
| Building  DataFrames  from scratch | **pd.read\_csv(‘filename/path’)** | df = pd.read\_csv('datasets/users.csv', index\_col=0) |
| from dictionary | 1# df = pd.DataFrame(dictionay name)  2# zipped = list(zip(list\_labels, list\_cols))  data = dict(zipped)  df = pd.DataFrame(data) |
| Broadcasting |  | users['fees'] = 0 # Broadcasts to entire column |
|  | with a dict | data = {'height': heights, 'sex': 'M'}  results = pd.DataFrame(data) |
| Index and columns |  | results.columns = ['height (in)', 'sex']  results.index = ['A', 'B', 'C', 'D', 'E', 'F', 'G'] |
| Importing & exporting data | | |
| from CSV files | pd.read\_csv(filepath) | sunspots = pd.read\_csv(filepath) |
| Prob: no header | “**header**” keyword | sunspots = pd.read\_csv(filepath, header=None)  # integer atar |
| “**names**” keyword | col\_names = ['year', 'month', 'day', 'dec\_date', 'sunspots']  sunspots = pd.read\_csv(filepath, header=None, names=col\_names) |
| Prob: missing val. | “**na\_values**” keyword | sunspots = pd.read\_csv(filepath, header=None, names=col\_names, na\_values={'sunspots':[' -1']})  # “-1” gordugu yerde “NaN” atayacak |
| Prob: dates | “**parse\_dates**” keyword | sunspots = pd.read\_csv(filepath, …., parse\_dates=[[0, 1, 2]]) |
| Using dates as index |  | sunspots.index = sunspots['year\_month\_day']  sunspots.index.name = 'date' |
| Trimming redundant columns |  | cols = ['sunspots', 'definite']  sunspots = sunspots[cols] |
| Writing files | **.to\_csv()** | out\_csv = 'sunspots.csv'  sunspots.to\_csv(out\_csv) |
|  | **.to\_excel(** | out\_xlsx = 'sunspots.xlsx'  sunspots.to\_excel(out\_xlsx) |
| Ploting with pandas | | |
| Plotting arrays | **plt.plot()** | close\_arr = aapl['close'].values # array olusturma  plt.plot(close\_arr) |
| Ploting Series |  | close\_series = aapl['close']  plt.plot(close\_series) |
|  |  | close\_series.plot() # plots Series directly |
| Plo!ing DataFrames |  | aapl.plot() # plots all Series at once |
|  |  | plt.plot(aapl) # plots all columns at once |
| Fixing scales |  | plt.yscale('log') # logarithmic scale on vertical axis |
| Customizing plots | **plt.axis( )** | aapl['open'].plot(color='b', style='.-', legend=True)  plt.axis(('2001', '2002', 0, 100)) |
| Saving plots | **plt.savefig( )** | aapl.loc['2001':'2004',['open', 'close', 'high', 'low']].plot()  In [27]: plt.savefig('aapl.png')  In [28]: plt.savefig('aapl.jpg')  In [29]: plt.savefig('aapl.pdf') |

**Chapter-2: Exploratory Data Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Visual exploratory data analysis | | | |
| Line plot | **df.plot(x, y)** | iris.plot(x='sepal\_length', y='sepal\_width') |
| Scatter plot | **“kind=” keyw** | iris.plot(x, y, kind='scatter') |
| Box plot |  | iris.plot(y='sepal\_length’, kind='box') |
| Histogram |  | iris.plot(y='sepal\_length', kind='hist') |
| Customizing histogram | bins=  range=( )  normed=True/False  cumulative=T/F | iris.plot(y='sepal\_length', kind='hist', bins=30, range=(4,8), cumulative=True, normed=True) |
| DataFrame plot idioms | ● iris.plot(kind=‘hist’)  ● iris.plt.hist( )  ● iris.hist( ) |  |
| Statistical exploratory data analysis | | | |
| Summarizing | **.describe( )** | iris.describe() # summary statistics |
| counts | **.count( )** | iris['sepal\_length'].count() # Applied to Series  iris['sepal\_width'].count() # Applied to Series  iris[['petal\_length', 'petal\_width']].count() # Applied to DataFrame |
| Averages | **.mean( )** | iris['sepal\_length'].mean() # Applied to Series  iris.mean() # Applied to entire DataFrame |
| Std | **.std( )** | iris.std() |
| Median | **.median( )** | iris.median() |
|  | Medians & 0.5 quantiles | q = 0.5  iris.quantile(q) |
|  | Inter-quartile range (IQR) | q = [0.25, 0.75]  iris.quantile(q) |
| Ranges | **.min( )**  **.max( )** | iris.min()  iris.max() |
| Separating populations | | | |
| Describe column |  | iris['species'].describe()  count 150 count: # non-null entries  unique 3 unique: # distinct values  top setosa top: most frequent category  freq 50 freq: # occurrences of top |
| Unique & factors | **.unique( )** | iris['species'].unique()  array(['setosa', 'versicolor', 'virginica'], dtype=object) |
| Filtering by column | **.loc[ ]** | indices = iris['species'] == 'setosa'  setosa = iris.loc[indices,:] # extract new DataFrame |
| Checking Column | **.unique( )** | setosa['species'].unique()  array(['setosa'], dtype=object) |
| Checking indexes | **.head( )** |  |
| Visual EDA: all data | **.plot( )** | iris.plot(kind= 'hist', bins=50, range=(0,8), alpha=0.3)  plt.title('Entire iris data set')  plt.xlabel('[cm]')  plt.show() |
| Visual EDA: individual factors |  | setosa.plot(kind='hist', bins=50, range=(0,8), alpha=0.3) |
| Statistical EDA: describe() | **.describe( )** | describe\_all = iris.describe()  describe\_setosa = setosa.describe() |
| Computing errors |  | error\_setosa = 100 \* np.abs(describe\_setosa - describe\_all)  error\_setosa = error\_setosa/describe\_setosa |

**Chapter-3: Time Series in Pandas**

|  |  |  |
| --- | --- | --- |
| Indexing Time Series | | |
| ISO 8601 format |  | yyyy-mm-dd hh:mm:ss |
| Parse dates | “**parse\_dates=True**”  “**index\_col=**” | sales = pd.read\_csv('sales-feb-2015.csv', parse\_dates=True, index\_col= 'Date') |
| Selecting datetime  .loc[ ‘row’, ‘column’] | Single datetime | sales.loc['2015-02-19 11:00:00', 'Company'] |
| Whole day | sales.loc['2015-2-5'] |
| Whole month | sales.loc[‘2015-2’] |
| Whole year | sales.loc[‘2015’] |
| Alternative formats | sales.loc[‘February 5, 2015’]  sales.loc[‘2015-Feb-5’] |
| Slicing |  | sales.loc['2015-2-16':'2015-2-20'] |
| Convert strings to datetime | **pd.to\_datetime( )** | evening\_2\_11 = pd.to\_datetime(['2015-2-11 20:00', '2015-2-11 21:00', '2015-2-11 22:00', '2015-2-11 23:00']) |
| Reindexing DataFrame | **.reindex( )** | sales.reindex(evening\_2\_11) |
| Filling missing values | **method=’ffill’** | sales.reindex(evening\_2\_11, method='ffill') |
| **method='bfill'** | sales.reindex(evening\_2\_11, method='bfill') |
| Resampling time series data | | |
| Aggregating means | **.resample( ).mean( )** | daily\_mean = sales.resample('D').mean() |
| Verifying | **.loc[ ]** | sales.loc['2015-2-2', 'Units'].mean() |
| Method chaining | .resample( ).sum( ) | sales.resample('D').sum() |
|  | .resample().sum().max() | sales.resample('D').sum().max() |
| Resampling frequencies |  | Input, Description  ‘min’, ‘ T’ minute  ‘H’ hour  ‘D’ day  ‘B’ business day  ‘W’ week  ‘M’ month  ‘Q’ quarter  ‘A’ year |
| Multiplying frequencies | .resample('2W') | sales.loc[:,'Units'].resample('2W').sum() |
| Upsampling and filling |  | two\_days = sales.loc['2015-2-4': '2015-2-5', 'Units']  two\_days.resample('4H').ffill() |
| Manipulating time series data | | |
| String methods | **.str.upper( )** | sales['Company'].str.upper() |
| Substring matching | **.str.contains(‘….’)** | sales['Product'].str.contains('ware') |
| Boolean reduction |  | sales['Product'].str.contains('ware').sum() |
| Datetime methods | **.dt.hour** | sales['Date'].dt.hour |
| Set timezone | **.dt.tz\_localize(‘…..')** | central = sales['Date'].dt.tz\_localize('US/Central') |
| Convert timezone | **.dt.tz\_convert('…..')** | central.dt.tz\_convert('US/Eastern') |
| Method chaining |  | sales['Date'].dt.tz\_localize('US/Central').dt.tz\_convert('US/Eastern') |
| Upsample population | **.resample().first()** | population.resample('A').first() |
| Interpolate missing data | **.interpolate( )** | population.resample('A').first().interpolate('linear') |
| Time series visualization | | |
| One week | **.loc[ ]** | sp500.loc['2012-4-1':'2012-4-7', 'Close'].plot(title='S&P 500') |
| Plot styles | **style=** | sp500.loc['2012-4', 'Close'].plot(style='k.-', title='S&P500') |
| More plot styles |  | Color // Marker // Line  b: blue // o: circle // : do"ed  g: green //\*: star // –: dashed  r: red // s: square  c: cyan // +: plus |
| Area plot | **kind=area** | sp500['Close'].plot(kind='area', title='S&P 500') |
| Multiple columns | **.loc[ ]** | sp500.loc['2012', ['Close','Volume']].plot(title='S&P 500') |
| Subplots | **subplots=True** | In [21]: sp500.loc['2012', ['Close','Volume']].plot(subplots=True) |

Codes from exercises

|  |  |  |
| --- | --- | --- |
| Set the index | **.set\_index( )** | df.set\_index('Date', inplace=True) |
|  | **.split(',')** |  |
| Creating a datetime series | **pd.date\_range(start, periods, freq))** | i = pd.date\_range('2018-04-09', periods=4, freq='2D') |
|  | **pd.to\_numeric()** | converts a Series of values to floating-point values |
|  | **errors='coerce'** | you can force strings like 'M' to be interpreted as NaN. |
|  | **.corr()** | The Pearson correlation coefficient |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |