Pandas

`df[ ['last', 'email'] ]` : we are passing a list in dataframe, it will dislplay the two rows named those

`df.corrwith(df1, axis =0 )` cross correlation between two data frames having same columns

we can covert a dictionary to data frame:

`df = pd.DataFrame(dict_name)` where the keys will become rows and their lists will become rows

`df.columns` to view the columns we have

`df.iloc[0]` will give us the first row, it gives by location of index

`df.iloc[[0,1]]` selecting multiple rows, we are passing a list of rows thats why double bracket

`df.iloc[[0,1], 2 ]` we can get values for 2nd column of the two rows using this

`df.loc[[0,1], 'email']` or `df.loc[ [ 0,1], [ 'email', 'last_name']]` similar to iloc but here we can pass column name

`df['col_name'].value_counts()` will give the count of unique values in that column

How to Set, Reset, and Use Indexes

`df.set_index('col_name', implace = True)` :implace = True is used to apply it permanently on dataframe

`df.reset_index(implace = True)` : will reset index to 0, 1...

`df.sort_index(ascending =False, implace =True)` sorting index in descending order

Using Conditionals to Filter Rows and Columns

`df[(df['last] == 'doe') & (df['first']== 'john')]` will give rows and columns where last ==doe is true

if we use -ve then it will give the opposite set

()brackets are important for conditions

`df[df['Col2'].isnull()]`, where our column data is null

`df_mg['long_signal'] = np.where(df_mg['price'] > 1.02*df_mg['prev_price'], 1, 0)`

conditionals with specified value, when true =1, when false =0

`high = (df['something']>1000)`

`df.isnull().sum()` , null value count in every column

`df.loc[ high, ['country', 'state']]` we can also do this, filter specific rows based on condition

```
cols_with_missing = [col for col in X_train.columns if X_train[col].isnull().any()]
```

Filtering out columns having NAN, we can use this with `df.drop()` to drop as required

```
object_cols = [col for col in X_train.columns if X_train[col].dtype == "object"]
```

Columns having categorical values

Updating Rows and Columns - Modifying Data Within DataFrames

`df.columns = ['first_name', 'last_name', 'email']` will update the columns to this

`df.columns = [ x.upper() for x in df.columns]` we can convert the above columns to uppercase

`df.columns = df.columns.str.replace('_', ' ' )` replace _ in the column name to space

```
df.rename(columns = {'fist_name': 'fisrt' , 'last_name' : 'last', implace = True)
```

`df.loc[2] = [ a list ]` : we can pass a list to make a row this way

we can also use condition like above to update rows and column

`df['col_name']. str.lower()` to lowercase

`df['email']. apply(len)` this apply function applies a particular function to whole df

`df['email'] = df['email']. apply(lambda x:x.lower())` lambda function applying to df

`df.applymap(len)` : it will apply the function to each individual value of dataframe

`df['first'].map({'corey' :'chris' , 'jane':'mary'})` Map function changes the values according to the keys and the other values in the dataframe becomes NAN , so we can use replace() function instead if we do not want that

Add/Remove Rows and Columns From DataFrames

```
df['full_name']  = df['first'] + ' ' + df['last']
```

`df.drop(columns = [ 'first',  'last'], implace = True)` dropping a list of columns, implace = True to apply the changes

`df['full_name'].str.split(' ', expand =True)` will split the full name into two columns

`df.append({'first': 'Tony'}, ignore_index = True)` will append a new row with fist column vale tony and others rows value as NAN

`df.drop(df[df['last'] == 'doe'].index, inplace =True)` conditionally dropping rows

pass them to df.replace(), specifying each char and it's replacement:

```
df[cols] = df[cols].replace({'\$': '', ',': ''}, regex=True)
```

`sf_permits.dropna(axis=1)` Dropping columns having one or more NAN, `axis =0` for dropping rows

Sorting Data

`df.sort_values(by='last', ascending =False)` sorting value by a particular column in descending order, we can use implace = True to make change to original dataframe

`df.sort_values(by=['last', 'first'] ascending = [False, True])`

`df.sort_index()`

Grouping and Aggregating, concating

`df.median()` to get median of whole dataframe columns

`df.describe()`

`df['social_media'].value_counts(normalize = True)` this will get the value count of every input , we can use normalize = True to get the % of every input

`country_grp  =` `df.groupby(['country'])` will group the data by country , now we can accesss data for every country without using conditions

`country_grp['coverted_comp']. agg(['mean', 'median'])` aggregate function will give the value of the inputed function for every country for that column

`country_grp['coverted_comp']. agg(['mean', 'median']).loc['unitedstates']` will get the mean, median values of converted comp for united states

we can also use apply map and other functions to country_grp

`python_df = pd.concat([country_respondents, country_uses_python], axis = 'columns', sort = False)` this will concatinated the two dataframes inputed, we gave axis as columns coz default is row

Casting Datatypes

`df['age']= df['age'].astype(int)`

`df.astype(float)` if we want to change datatype of every column inside df

`DataFrame.select_dtypes( include=None , exclude=None)`

To Date_time

`df['Date'] = pd.to_datetime(df['Date'], format = '%Yy%mm%dd')` give format parameter according to your data

`df.loc[0, 'Date'].day_name()` will give name of that date on 0th row of date column

`df['Date'].dt.day_name()`

`df['Date'].max() - df['Date'].min()` will give no. of days between whole dataframe

`df['high'].resample('D').max()` will resample the hourly data into daily with max() as a value for the day , we can use another function as well

`df.resample('w').agg({'close':'mean', 'high':'max'})` weekly resampling with using aggregate function

`datetime.today().year` to get the year of today we can apply `.year` to any datetime object, similar function is available for `day` and `month`

Reading/Writing from different formats

`pd.read_csv()`

`df.to_csv('filename.csv', index = False, header =None )`

`pd.read_excel()`

`df.to_excel('filename.xlsx')`

`pd.read_json('filename.json', orient = 'records', line =True`)

`df.to_json('filename.json', orient = 'recors', line =True )`