Alex Wolf
5/6/17

Steps for Data Cleaning Pandas

Import Libraries

- import pandas as pd
- Import glob
- `import matplotlib.pyplot as plt`
- Import re

Read/Save Csv

- df= pd.read_csv('filename.csv', index_col=None)
- pd.to_csv('fileName.csv')

Inspect/Fix Preliminary Errors

- print(df.head())
  - Show DF from top 10 entries
- print(df.info())
  - Show datatypes
  - Show amount of null values
- *** make sure data matches DataTypes
- print(df.types)
- Column name consistency
  - print(df.columns)
  - df.columns= [' column list ',]
- Delete duplicates
  - df=df.drop_duplicates()

Check DataTypes to Make sure good in each columns

1. print(df.info())
   a. Shows dataTypes
2. Analyze it

. Numeric
i.df.value_counts(dropna=false)
a. Categorical
.Df.info or df.describe

Check to make sure Data Uniform

- 1. See types
  - df.types
  - **df.COLNAME.unique()**
    - Check for all types of values in a column
- 2. If should be string or Categorical
  - df['colName']=df['colName'].astype(TYPE)

        - Possible types
            - str
            - ‘category’
                - Make dataFrame Smaller in Memory
                - Use cases for future
    - String
    - Create new columns for data with more values in a column
        - For strings
            - df[’NewColName]='df.COLNAME.str[indexStart: IndexEnd]
    - String indexing
        - 1. df[‘listCol'] = df.CONAMEL.str.split(‘delimeter’)
        - 2. df[’newColName’]= df.listCol.str.get(INDEX of Split)
            - Make new column
    - String Conversion
        - string=string.replace(’toReplece’, ‘replacement')
- Numeric
    - From String
        - pd.to_numeric(df[‘COLNAME’], errors= ‘coerce’)
        - Invalid columns would be set as NaN
        - Error spit if can’t convert
- Functions Applying
    - Lambda functions
        - `df.apply(lambda x: x ** 2)`
    - Apply
        - Function Want 2 params
            - 1. The row
            - 2, the pattern to match for strings
        - df.apply(FUNCTION,)
        - df.apply(df, axis=1/0)
            - Axis1 means rows
            - Axis0 means columns (default)
        - May want to return NaN

        - Example

```
# Define recode_sex()
def recode_sex(sex_value):

    # Return 1 if sex_value is 'Male'
    if sex_value == 'Male':
        return 1

    # Return 0 if sex_value is 'Female'
    elif sex_value == 'Female':
        return 0

    # Return np.nan
    else:
```

```
        return np.nan

    # Apply the function to the sex column
    tips['sex_recode'] = tips.sex.apply(recode_sex)
```

Fill Missing Values

- checkNanCount
  - df.info()
- Check Range
  - df.describe()
- Drop missin values
  - df.dropna(axis=0/1)
  - 0 for rows check
  - 1 for call check
- Which to impute by
  - Mean if data balanced
  - Median if high amount
  - If a lot of noise can use a *Multiple IMputation*
- df[‘col1’, ‘col2’, ...] = df[‘col1’, ‘col2’, ...].fillna(df[‘col1’, ‘col2’, ...].mean())

Concatenate/Merge DataFrames

- Concatenate
  - Pd.concat([df1. Df2, …], axis=0, ignore_index=true)
    - ***Second param makes continuous index labels
    - Change Axis to 1 if the rows are the name and different collumns
- Merge
  - Handles concatenation for different ordering of Values
  - pd.merge(left=DF1, right=DF2, on=BLanK, left_on=‘’, right_on=‘’)
    - left= DF!
    - right=DF2
    - on= specifics the key if the Same
      - None if not the same
    - Left_on= colName of DF1 to merge
    - Right_on= colName of DF2 to merge
  - types
    - 1 tot 1
      - Same as above
    - Mant to 1/ 1 to Many

        - Same as above
    - Many to Many

```
pd.merge(df_new, df_n, left_on='subject_id', right_on='subject_id')
```

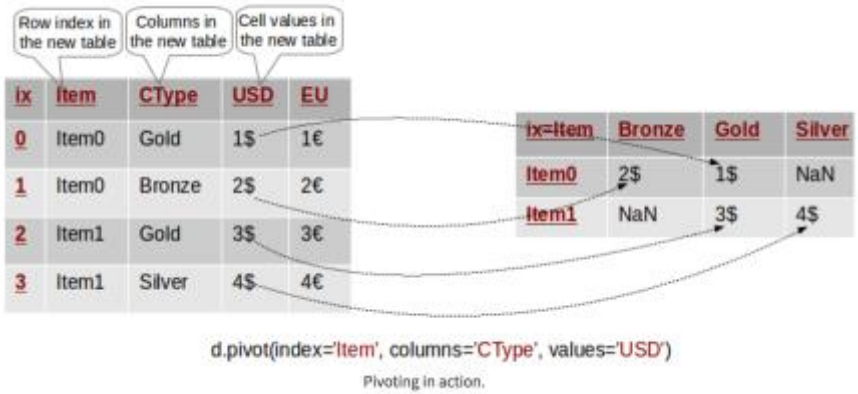
	subject_id	first_name	last_name	test_id
0	1	Alex	Anderson	51
1	2	Amy	Ackerman	15
2	3	Allen	Ali	15
3	4	Alice	Aoni	61
4	4	Billy	Bonder	61
5	5	Ayoung	Atiches	16
6	5	Brian	Black	16
7	7	Bryce	Brice	14
8	8	Betty	Btisan	15

-
    - df1-SubjectID [1:4] and df2 rest
    -

<u>Check for Tidy Data</u>

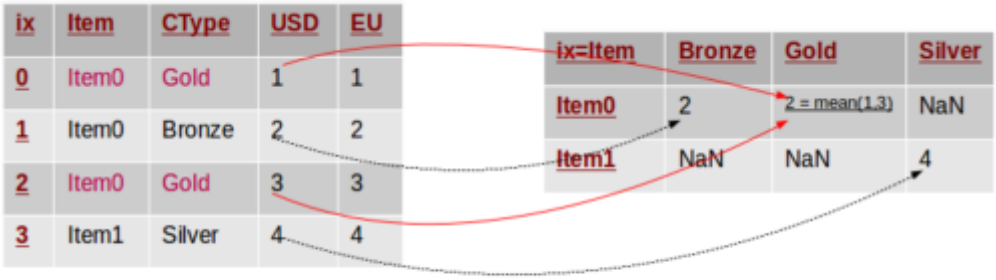
- Principles
    - Columns represent separate variables
        - Two columns don't mean the same thing
        - No two separate columns for ONE- categorical and then value
    - Rows Represent individual observations
    - Observational units form tables
- Fix using
    - 1. pd.melt()
        - Columns to rowData
        - Params
            - Id_vars- ['COLNAME'. …,]
                - Columns not to melt
            - Value_vars['COLNAME'. …,]
                - Columns to melt to rows
            - If not specified all columns not in id_vars will be melted into a one column(row)

- 2. Pivot
  - Opposite of melt
  - *** shuould do when Duplicate of some columes
    - print(Df[[‘col1’, col2]].value_counts>1)
  - Current idex should be consecutive integeres
  - rows to column Data
    - It value in it will be in new column called value
  - df.pivot(index='', columns=‘’, values=‘’)
    - index=which columns to fix during pivot
    - columns= columns to pivot into new columns
    - values= vars to fill in for column with pivot
  - May not always work
    - CAN’t HAVE DUPLICATE Examples



Pivoting in action.

  -
  - *** if values left blank everything else considered a value
-
-
-
- 3. Pivot Table
  - Does this if values merging have the same vale for index+column



-
-

Assert to check for correctness

- df=df.drop_duplicates()
- Check data types
  - assert gapminder.country.dtypes == np.object
  - assert gapminder.year.dtypes == np.int64
- Check for value in a certain range
  - assert (df >= 0).all().all()
- Check for instance count
  - assert df['ColName'].value_counts()[0] == DESIRED COUNT
- Check for no more missing Data
  - assert Df.notnull().all().all()
    - Have two alls if DF and not series
- Drop duplicates
  - df=df.drop_duplicates()