Data Preparation using Python

1. Import libraries

2. Import data

- Read from csv
- Connect to a database

3. Metadata and data types

- Metadata
- Modify data types

4. Duplicate detection

- Find duplicates
- Remove duplicates

5. Dataframe manipulation

- Columns and rows
- Aggregation, filtering and sorting
- Combine dataframes

6. Outlier detection

- Tukey’s test for extreme values
- Kernel density estimation

7. Variable generation and manipulation

- Generate new variables
- Bucket variables
- Encode categorical variables
- Generate dummy variables

8. Prepararation of data for modeling

- Draw samples and split dataset
- Reshape data for modeling

https://medium.com/@Pichitchai.Pim (https://medium.com/@Pichitchai.Pim)

Import libraries

In [ ]:
```
import pandas as pd
import numpy as np
import psycopg2 as ps
from IPython.display import HTML
%matplotlib inline # show plots inline in Jupyter Notebook
```

Import data

Read from csv

In [ ]:
```
df = pd.read_csv('PATH', encoding = 'utf-8')
len(df) # N rows imported (add for future reference)
```

In [ ]:
```
#If you want dates to be recognized as dates, add this argument:
pd.read_csv('PATH', parse_dates=True)
```

Read from Excel

In [ ]:
```
df = pd.read_excel('PATH', encodeing = 'utf-8')
len(df)
```

Connect to a database

In [ ]:
```
# Establish database connection
con=ps.connect(dbname= 'DBNAME', host='HOST',
port= 'PORT', user= 'USER', password= 'PASSWORD')
cur = con.cursor()

# Query raw data
cur.execute("""SELECT STATEMENT""")
data = cur.fetchall()
len(data) # Nrows imported, DATE (add for future reference)

# Close connection
cur.close()

# Parse the tables into a Pandas dataframe (specify column names)
data = pd.DataFrame(list(data), columns=('COL1', 'COL2', 'CO3'))
data.head()
```

Metadata and data types

Metadata

Runtime info for single cells: %%time magic gets you the time spent on cell execution.

Info on data types:

In [ ]:
```
df.info()
```

Summary of missing values:

In [ ]:
```
df.isnull().sum()
```

Replace missings by 0 for numeric variables (if appropriate):

In [ ]:
```
for i in range(0, 3):
    df.iloc[:,i].fillna(value=0, inplace=True)
#Specify the range of columns - here: columns 1-4.
```

Modify data types

Cast to other data types:

In [ ]:
```
for i in range(2, 2):
    df.iloc[:, [i]] = df.iloc[:, [i]].apply(np.int64)
#Specify the range of columns (here: columns 2-3) and the data type (can also
 be String, etc.).
```

Convert a microseconds integer timestamp to datetime:

In [ ]:
```
df.my_ts = pd.to_datetime(df.my_ts)
```

Duplicate detection

Find duplicates

Count unique values of one column:

In [ ]:
```
df.groupby('my_category').my_user_id.nunique()
```

Show duplicate values for one column:

In [ ]:
```
pd.concat(i for _, i in df.groupby('my_user_id') if len(i) > 1)

#In case you want to show duplicates for a combination of columns,
#replace 'my_user_id' by the list of columns to be considered (e.g. ['my_user_
id', 'my_country']).
```

Remove duplicates

In [ ]:
```
df = df.drop_duplicates(subset='my_user_id', keep=False)

#Change 'keep' parameter to 'first'/'last' if applicable.
```

Dataframe manipulation

Columns and rows

Drop one or more column(s):

In [ ]:
```
df = df.drop(['col1', 'col2'], axis=1)
```

Delete a column:

In [ ]:
```
del df['COL1']
```

Drop one or more rows:

In [ ]:
```
df = df.drop(['Index1', 'Index2'])
```

Remove all rows that do not fulfill a condition:

In [ ]:
```
df = df[df.COL1 < x]
```

Replace values:

In [ ]:
```
df[COL1].replace({"VALUE1": 0, "VALUE2": 1})
```

Rename one column:

In [ ]:
```
df = df.rename(columns = {'OLDNAME': 'NEWNAME'})
```

Rename all columns:

In [ ]:
```
df.columns = ['COL1', 'COL2', 'COL3']
```

Change the order of columns:

In [ ]:
```
# Show the list of columns to then copy and rearrange
cols = list(df.columns.values)
# Rearrange
df = df[['COL2', 'COL3', 'COL1']]
```

Transpose a dataframe:

In [ ]:
```
data = data.transpose(as_index = False)
```

Pivot a dataframe:

In [ ]:
```
df2 = df.pivot_table('CATEGORY_COUNT', 'INDEX_VARIABLE', 'CATEGORY')
```

Aggregation, filtering and sorting

Aggregate by grouping (equivalent to SQL "GROUP BY"):

In [ ]:
```
df.groupby(by = ['COL1', 'COL2'], as_index = False, sort = False).COL3.sum()
```

Filter a dataframe:

One condition:

In [ ]:
```
df[df.COL1 > 1]
```

Multiple conditions:

In [ ]:
```
df[(df['colCOL11'] >= 1) & (df['COL2'] <=1 )]
```

Sort dataframe:

In [ ]:
```
df.sort_values(by = ['COL1', 'COL2'], ascending = (0, 1))
```

Add a column containing the sum over different columns:

In [ ]:
```
helplist = ['COL1', 'COL2', 'COL3']
df['total'] = df[helplist].sum(axis = 1)
```

Combine dataframes

Join dataframes (equivalent to SQL JOIN operations):

In [ ]:
```
# left join
df = pd.merge(df1, df2, how = 'left', left_on = ['my_user_id'])

# Right join
df = pd.merge(df1, df2, how = 'right', left_on = ['my_user_id'])

#inner join
df = pd.merge(df1, df2, how = 'inner', left_on = ['my_user_id','my_department_
id'])


# outer join
df = pd.merge(df1, df2, how = 'outer', left_on = ['my_user_id','my_department_
id'])
```

Add rows of another dataframe to your dataframe:

In [ ]:
```
df = pd.concat([df1, df2], ignore_index=True)
```

In [ ]:
```
df = df1.append(df2, ignore_index=True)
```

Outlier detection

Kernel density estimation

In [ ]:
```
from statsmodels.nonparametric.kde import KDEUnivariate

# Define outlier function
def find_outliers_kde(x):
    x_scaled = scale(list(map(float, x)))
    kde = KDEUnivariate(x_scaled)
    kde.fit(bw = "scott", fft = True)
    pred = kde.evaluate(x_scaled)

    n = sum(pred < 0.05)
    outlier_ind = np.asarray(pred).argsort()[:n]
    outlier_value = np.asarray(x)[outlier_ind]

    return outlier_ind, outlier_value

# Print outlier values
for x in range(1, 7): # Modify to select numeric columns
    kde_indices, kde_values = find_outliers_kde(data.ix[:, x])
    print(list(data[[x]]), np.sort(kde_values))
```

Tukey's test for extreme values

In [ ]:
```
# Define function using 1.5x interquartile range deviations from quartile 1/3
 as floor/ceiling
def find_outliers_tukey(x):
    q1 = np.percentile(x, 25)
    q3 = np.percentile(x, 75)
    iqr = q3-q1
    floor = q1 - 1.5 * iqr
    ceiling = q3 + 1.5 * iqr
    outlier_indices = list(x.index[(x < floor)|(x > ceiling)])
    outlier_values = list(x[outlier_indices])
    return outlier_indices, outlier_values

# Print outliers for each numeric variable
for x in range(1, 7): # Modify to select numeric columns
    tukey_indices, tukey_values = find_outliers_tukey(data.ix[:, x])
    print(list(data[[x]]), np.sort(tukey_values))
```

Variable generation and manipulation

Generate new variables

In [ ]:
```
df['NEWCOL'] = 0
```

In [ ]:
```
df['NEWCOL'] = df.COL1/df.COL2
```

In [ ]:
```
def mins_to_secs(x):
    x = df.time_spent_minutes/60

df['time_spent_seconds'] = df.time_spent_minutes.map(mins_to_secs)
```

In [ ]:
```
# Bucket low frequency categories as "Other"
def repl(x):
    if x == 'US': return 'US'
    elif x == 'BR': return 'BR'
    elif x == 'ES': return 'ES'
    else: return 'Other'

df['COUNTRY'] = df['COUNTRY'].apply(repl)
print(df['COUNTRY'].value_counts().sort_values(ascending=False))
```

Encode categorical variables

Encode a boolean variable by casting to integer:

In [ ]:
```
df['BOOL'] = (df.COL1=="ABC").astype(int)
dta.head()
```

Encode manually by mapping a dictionnary:

In [ ]:
```
dic = {'Yes': 1, 'No': 2}
df['VAR'] = df['VAR'].map(dic)
```

Generate dummy variables

In [ ]:
```
dummy = pd.get_dummies(dta1['country'], prefix='ct').astype(int)
dummy.head()
```

Preparation of data for modeling

Draw samples and split dataset

Draw a random sample from a dataset:

In [ ]:
```
data2 = data1.sample(1000)
```

Split test and training data:

In [ ]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, rando
m_state=0)
```

Reshape data for modeling

Reshape dataframe to array (input for Scikit-Learn or Scipy models):

In [ ]:
```
array = df.values
```

In [ ]:
```
X = array[:,1:5]
Y = array[:,0]
```

Flatten dataframe into a 1-dimensional array:

In [ ]:
```
Y = np.ravel(Y)
```