Hold the Code

October 15, 2021

Table of Contents

[ ]:
```
df.columns
```

[ ]:
```
df.index
```

[ ]:
```
# Select by datatype
df.dtypes

# This include both int and float columns
df.select_dtypes(include='number').head()
```

```
# select just object columns
df.select_dtypes(include='object')

# select multiple data types
df.select_dtypes(include=['int', 'datetime', 'object'])

# exclude certain data types
df.select_dtypes(exclude='int')
```

1 Rename

[ ]:
```
import numpy as np
import pandas as pd
from numpy.random import randn
```

[ ]:
```
imm = pd.read_excel('/Users/jimcody/Documents/2021Python/intropython/data/
 ↪immunotherapy.xlsx',
                               sheet_name = 'Immuno1')
imm.head()
```

[ ]:
```
# Rename as part of read_csv
imm = pd.read_excel('/Users/jimcody/Documents/2021Python/intropython/data/
 ↪immunotherapy.xlsx',
                    names = ('a','b','c','d','e','f','g','h'),
                    sheet_name = 'Immuno1')
imm.head()
```

[ ]:
```
# Rename all columns using rename()
imm.rename(columns = {'a':'a1','b':'b1','c':'c1','d':'d1','e':'e1','f':'f1','g':
 ↪'g1','h':'h1'}, inplace = True)
imm.head()
```

[ ]:
```
# Rename some using rename()
imm.rename(columns = {'b1':'BB','e1':'EE'}, inplace = True)
imm.head()
```

[ ]:
```
# Rename using set_axis()   not used that often
imm.set_axis(['AA','BB','CC','DD','EE','FF','GG','HH'], axis = 'columns',␣
 ↪inplace = True)
imm.head()
```

[ ]:
```
# Rename using .columns()
imm.columns = ['AaA','BbB','CcC','DdD','EeE','FfF','GgG','HhH']
imm.head()
```

[ ]:

```
# Rename using str.replace
imm.columns = imm.columns.str.replace('CcC','CCC')
imm.head()
```

2 String functions

[ ]:

```
# Passing `str.lower` function to lowercase column names
df.rename(columns=str.lower).head()

#df_excel['gender'].str.title()
#df_excel['gender'].str.upper()
#df_excel['gender'].str.title()
```

[ ]:

```
# Custom function
def toUpperCase(string):
    return string.upper()

df.rename(columns=toUpperCase).head()
```

[33]:

```
# joining strings

name = 'Guido van Rossum'
year = 1991
f'Python was created by {name} and released in {year}'
```

[33]:

```
'Python was created by Guido van Rossum and released in 1991'
```

[34]:

```
# joining strings

name = 'Guido van Rossum'
year = 1991

"Python was created by {} and released in {}".format(name, year)
```

[34]:

```
'Python was created by Guido van Rossum and released in 1991'
```

[35]:

```
import datetime
now = datetime.datetime.now()
f'Today is {now:%B} {now:%-d}, {now:%Y}'
```

[35]:

```
'Today is October 15, 2021'
```

3 Dropping/Imputing missing data

[ ]:

```
# To drop rows if any NaN values are present
df.dropna(axis = 0)
```

[ ]:

```
# To drop columns if any NaN values are present
df.dropna(axis = 1)
```

[ ]:

```
# To drop columns in which more than 10% of values are missing
df.dropna(thresh=len(df)*0.9, axis=1)
```

[ ]:

```
# To replace all NaN values with a scalar
df.fillna(value=10)

# To replace NaN values with the values in the previous row.
df.fillna(axis=0, method='ffill')

# To replace NaN values with the values in the previous column.
df.fillna(axis=1, method='ffill')

# Replace with the values in the next row
df.fillna(axis=0, method='bfill')

# Replace with the values in the next column
df.fillna(axis=1, method='bfill')

# To replace NaN values with the mean
df['Age'].fillna(value=df['Age'].mean(), inplace=True)
```

[ ]:

```

```

[ ]:

```
  # Basic statistics for price
```

[ ]:

```
df['price'].describe()
df['price'].unique()
df['price'].nunique()
df['col2'].value_counts()
```

[ ]:

```

```

[ ]:

```
# Segregation of Numerical and Categorical Variables/Columns

cat_col = df.select_dtypes(include=['object']).columns
num_col = df.select_dtypes(exclude=['object']).columns
df_cat = df[cat_col]
df_num = df[num_col]
```

3.1 Applying functions

[ ]:
```
def times2(x):
    return x*2
```

[ ]:
```
df['col1'].apply(times2)
```

[ ]:
```
df['col3'].apply(len)
```

[ ]:
```
df['col1'].sum()
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```
# Transform Sex into binary values 0 and 1
sex = pd.Series( np.where( full.Sex == 'male' , 1 , 0 ) , name = 'Sex' )
```

[ ]:
```
# Extracting Car Company from the CarName as per direction in Problem

df['CarName'] = df['CarName'].str.split(' ',expand=True)
```

[ ]:
```
# Unique Car company

df['CarName'].unique()# Renaming the typo errors in Car Company names

df['CarName'] = df['CarName'].replace({'maxda': 'mazda', 'nissan': 'Nissan',␣
 ↪'porcshce': 'porsche', 'toyouta': 'toyota',
                            'vokswagen': 'volkswagen', 'vw': 'volkswagen'})
```

[ ]:
```
# Fill missing values of ChildPoverty with the average of ChildPoverty (mean)

df[ 'ChildPoverty' ] = df.ChildPoverty.fillna( df.ChildPoverty.mean() )
df_null = df.isna().mean().round(4) * 100

df_null.sort_values(ascending=False).head()
```

[ ]:
```
df['A'].fillna(value=df['A'].mean())
```

[ ]:
```
# Create dataset
imputed = pd.DataFrame()

# Fill missing values of Age with the average of Age (mean)
imputed[ 'Age' ] = full.Age.fillna( full.Age.mean() )

# Fill missing values of Fare with the average of Fare (mean)
imputed[ 'Fare' ] = full.Fare.fillna( full.Fare.mean() )
```

```
imputed.head()
```

[ ]:

```
cabin = pd.DataFrame()

# replacing missing cabins with U (for Uknown)
cabin[ 'Cabin' ] = full.Cabin.fillna( 'U' )

# mapping each Cabin value with the cabin letter
cabin[ 'Cabin' ] = cabin[ 'Cabin' ].map( lambda c : c[0] )

# dummy encoding ...
cabin = pd.get_dummies( cabin['Cabin'] , prefix = 'Cabin' )

cabin.head()
```

[ ]:

```
family = pd.DataFrame()

# introducing a new feature : the size of families (including the passenger)
family[ 'FamilySize' ] = full[ 'Parch' ] + full[ 'SibSp' ] + 1

# introducing other features based on the family size
family[ 'Family_Single' ] = family[ 'FamilySize' ].map( lambda s : 1 if s == 1␣
 ↪else 0 )
family[ 'Family_Small' ]  = family[ 'FamilySize' ].map( lambda s : 1 if 2 <= s␣
 ↪<= 4 else 0 )
family[ 'Family_Large' ]  = family[ 'FamilySize' ].map( lambda s : 1 if 5 <= s␣
 ↪else 0 )

family.head()
```

[ ]:

```
# Select which features/variables to include in the dataset from the list below:
# imputed , embarked , pclass , sex , family , cabin , ticket

full_X = pd.concat( [ imputed , embarked , cabin , sex ] , axis=1 )
full_X.head()
```

[ ]:

```
# Create a new dataframe

df2 = df[['TotalPopulation','ChildPoverty','MeanCommute',
          'MeanHealthCommute','Unemployment','Unemployment%_2019',
          'Median Household Income_2019',]]
```

[ ]:

```
# Create a new variable for every unique value of Embarked
embarked = pd.get_dummies( full.Embarked , prefix='Embarked' )
embarked.head()
```

[ ]:

```
# a function that extracts each prefix of the ticket, returns 'XXX'
# if no prefix (i.e the ticket is a digit)
def cleanTicket( ticket ):
    ticket = ticket.replace( '.' , '' )
    ticket = ticket.replace( '/' , '' )
    ticket = ticket.split()
    ticket = map( lambda t : t.strip() , ticket )
    ticket = list(filter( lambda t : not t.isdigit() , ticket ))
    if len( ticket ) > 0:
        return ticket[0]
    else:
        return 'XXX'

ticket = pd.DataFrame()

# Extracting dummy variables from tickets:
ticket[ 'Ticket' ] = full[ 'Ticket' ].map( cleanTicket )
ticket = pd.get_dummies( ticket[ 'Ticket' ] , prefix = 'Ticket' )

ticket.shape
ticket.head()
```

[ ]:

```
title = pd.DataFrame()
# we extract the title from each name
title[ 'Title' ] = full[ 'Name' ].map( lambda name: name.split
                                       ( ',' )[1].split( '.' )[0].strip() )

# a map of more aggregated titles
Title_Dictionary = {
                    "Capt":       "Officer",
                    "Col":        "Officer",
                    "Major":      "Officer",
                    "Jonkheer":   "Royalty",
                    "Don":        "Royalty",
                    "Sir" :       "Royalty",
                    "Dr":         "Officer",
                    "Rev":        "Officer",
                    "the Countess":"Royalty",
                    "Dona":       "Royalty",
                    "Mme":        "Mrs",
                    "Mlle":       "Miss",
                    "Ms":         "Mrs",
                    "Mr" :        "Mr",
                    "Mrs" :       "Mrs",
                    "Miss" :      "Miss",
                    "Master" :    "Master",
                    "Lady" :      "Royalty"
```

```
                  }

# we map each title
title[ 'Title' ] = title.Title.map( Title_Dictionary )
title = pd.get_dummies( title.Title )
#title = pd.concat( [ title , titles_dummies ] , axis = 1 )

title.head()
```

3.2 Sorting DF

3.2.1 Remember - sort needs inplace

[ ]:
```
import pandas as pd
import numpy as np
client_dictionary = {'name': ['Michael', 'Ana', 'Sean'],
                     'country': ['UK', 'UK', 'USA'],
                     'age': [10, 51, 13],
                     'latest date active': ['07-05-2019', '23-12-2019',␣
 ↪'03-04-2016']}
df = pd.DataFrame(client_dictionary)
df.head()
```

[ ]:
```
df.sort_values(by='name')
```

[ ]:
```
df.sort_values(by='name', ascending=False)
```

[ ]:
```
df.sort_values(by='age')
```

[ ]:
```
df.sort_values(by=['country','name'])
```

[ ]:
```
df['latest date active'] = df['latest date active'].astype('datetime64[ns]')
df.sort_values(by='latest date active')
```

3.3 Changing data types

[ ]:
```
data = {"Car": ["A", "B", "C","D", "E", "F", "G", "H"],\
     "Color": ["Red", "Yellow", "Black", "Green", "Black", "Red", "Black",␣
 ↪"Black"],\
     "Year": ["1990", "1980", "2003", "2000", "2001", "2004", "1999", "2020"],\
     "Rating": ["2.5", "1.5", "3.8", "9.7", "8.9", "3.2", "5.5", "6.9"],\
     "Service": ["30/05/2010", "31/08/1999", "28/12/2005", "29/06/2011",\
                 "30/12/2020", "31/07/2013", "28/11/2000", "25/12/2020"]}
df = pd.DataFrame(data)
df.head()
```

[ ]:

```
df1 = df.copy()
```

[ ]:

```
# single column
df1["Year"] = df1["Year"].astype("int64")
df1.head()
```

[ ]:

```
# Multiple columns

df1 = df1.astype({"Year": "complex", "Rating": "float64",\
                  "Car": 'int32'}, errors='ignore')
df1.info()
```

[ ]:

```
df1["Car"] = df1["Car"].astype("int64", errors='ignore')
df1.head()
```

[ ]:

```
df1 = df1.astype("int64", errors='ignore')
df1.head()
```

[ ]:

```
df2 = df.copy()
```

[ ]:

```
df2["RealDate"] = pd.to_datetime(df2["Service"])
df2
```

[ ]:

```
df2["Rating"]=pd.to_numeric(df2["Rating"])

df2.info()
```

[ ]:

```
df2[["Rating", "Year"]] = df2[["Rating",\
                               "Year"]].apply(pd.to_numeric)
df2.head()
```

3.3.1 convert_dtypes()

[ ]:

```
df3 = df.copy()
dfn = df3.convert_dtypes()
dfn.info()
```

4 Selecting columns based on conditions

[ ]:

```
df = pd.DataFrame(
 [
  (73, 15, 55, 33, 'foo'),
  (63, 64, 11, 11, 'bar'),
  (56, 72, 57, 55, 'foo'),
 ],
 columns=['A', 'B', 'C', 'D', 'E'],
)
```

[ ]:

```
# Selecting Rows
x = df.loc[df['B'] == 64]
y = df[df['B'] == 64]
z = df[df.B == 64]
print(x)
print(y)
print(z)
```

[ ]:

```
# In the context of Python, it is a common practise to name
# such boolean conditions as mask that we then pass to DataFrame
# when indexing it.

mask = df.B == 64
df[mask]

# loc[] is the way to go. This is simply because df[mask] will always
# dispatch to df.loc[mask] which means using loc directly will be
# slightly faster.
```

[ ]:

```
df.loc[df['B'] != 64]                 # scalar not equal
df.loc[df['B'] >= 64]                 # scalar >=
df.loc[df['E'].str.contains('oo')]    # contains a string
df.loc[df['B'].isin([64, 15])]        # column is an iterable
df.loc[~df['B'].isin([64, 15])]       # not in the list
```

[ ]:

```
# Multiple conditions
df.loc[(df['A'] >= 59) & (df['E'].isin(['foo', 'boo']))]
```

[ ]:

```
#Select from DataFrame using criteria from multiple columns
newdf = df[(df['col1']>2) & (df['col2']==444)]
```

5 Factorize

.factorize() is a Pandas method that helps you to quickly transform your data from **text to numbers**.

It makes the same work as map or replace , but you don’t need to write a dictionary.

[ ]:

```python

```

[ ]:

```python
import pandas as ps
import seaborn as sns
df = sns.load_dataset('taxis')
```

[ ]:

```python
# slice only some textual columns
df = df[['payment', 'pickup_borough', 'pickup_zone']]
df.payment.unique()
```

[ ]:

```python
# .map is fine if there are only a few values
df.payment.map({'cash':0, 'credit card':1})
```

[ ]:

```python
df.pickup_zone.nunique()
```

5.0.1 Factorize()

[ ]:

```python
df.pickup_zone.factorize()
```

[ ]:

```python
# New factorized column
# Each pickup zone is assigned a value
df['zone_code'] = df.pickup_zone.factorize()[0]
df
```

[ ]:

```python
df['pay_cd'] = df.payment.factorize()[0]
df
```

6 Binning

[ ]:

```python
# Convert continuous data to categorical data
import sys

df['ageGroup']=pd.cut(
    df['Age'],
    bins=[0, 13, 19, 61, sys.maxsize],
    labels=['<12', 'Teen', 'Adult', 'Older']
)

df['ageGroup'].head(8)
```

7 Stories

7.1 Story 1

https://towardsdatascience.com/a-complete-yet-simple-guide-to-move-from-excel-to-python-d664e5683039

```
[ ]: import pandas as pd
     import numpy as np
     df_excel = pd.read_csv('/Users/jimcody/Documents/2021Python/intropython/data/
      ↪StudentsPerformance.csv')
     df_excel
```

```
[ ]: df_excel['math score'].mean()
```

```
[ ]: # Math
     df_excel['average'] = (df_excel['math score']
                            + df_excel['reading score']
                            + df_excel['writing score'])/3
```

```
[ ]: df_excel['average2'] = df_excel.mean(axis=1)
```

```
[ ]: df_excel
```

```
[ ]: df_excel['gender'].value_counts()
```

```
[ ]: df_excel['pass/fail'] = np.where(df_excel['average'] > 70, 'Pass', 'Fail')
```

```
[ ]: conditions = [
         (df_excel['average']>=90),
         (df_excel['average']>=80) & (df_excel['average']<90),
         (df_excel['average']>=70) & (df_excel['average']<80),
         (df_excel['average']>=60) & (df_excel['average']<70),
         (df_excel['average']>=50) & (df_excel['average']<60),
         (df_excel['average']<50),
     ]
     values = ['A', 'B', 'C', 'D', 'E', 'F']
```

```
[ ]: df_excel['grades'] = np.select(conditions, values)
     df_excel
```

```
[ ]: df_female = df_excel[df_excel['gender'] == 'female']
```

```
[ ]: df_sumifs = df_excel[(df_excel['gender'] == 'female') & (df_excel['race/
      ↪ethnicity'] == 'group B')]
```

```
[ ]: df_sumifs = df_sumifs.assign(sumifs = df_sumifs['math score']
                                  + df_sumifs['reading score']
                                  + df_sumifs['writing score'])
```

```
[ ]: #df_excel['gender'].str.lower()
     #df_excel['gender'].str.upper()
     df_excel['gender'].str.title()
```

```
[ ]: df_excel['gender'] = df_excel['gender'].str.title()
```

[ ]:
```
df_excel
```

[ ]:
```
df_excel['race/ethnicity'].str.extract(r'([A-Z])')
```

7.2 Story 2

https://towardsdatascience.com/pandas-cheat-sheet-for-data-preprocessing-cd1bcd607426

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_boston
from sklearn import preprocessing
```

[ ]:
```
pd.set_option('display.max_rows', 50)
pd.set_option('display.max_columns', 50)
```

[ ]:
```
boston = load_boston()
df_X = pd.DataFrame(boston.data, columns=boston.feature_names)
df_y = pd.DataFrame(boston.target, columns=['target'])
```

[ ]:
```
df_X.columns
```

[ ]:
```
df_X.columns.tolist()
```

[ ]:
```
df_X[df_X.duplicated()]
```

[ ]:
```
df_X[df_X.duplicated(keep='last')]
```

[ ]:
```
df_X.T[df_X.T.duplicated(keep=False)].T
```

[ ]:
```
np.where(pd.isnull(df_X))
```

[ ]:
```
df_X.replace(np.nan, 0)
```

[ ]:
```
df_X.dtypes
```

[ ]:
```
col_miss = ['DIS', 'B']
for i_col in col_miss:
    for j in df_X[i_col].unique():
        try:
            float(j)
        except ValueError:
            print(j)
```

[ ]:
```
df_X.replace('1..7554', 1.7554)
df_X.replace('396.9.9', 396.99)
```

[ ]:
```
df_X[['DIS', 'B']] = df_X[['DIS', 'B']].astype(float)
```

[ ]:
```
df_X.nunique()
```

[ ]:
```
mm = preprocessing.MinMaxScaler()
df_float = df_X.loc[:, df_X.dtypes == 'float64']
df_scaled = pd.DataFrame(mm.fit_transform(df_float), index=df_float.index,␣
 ↪columns=df_float.columns)
duplicates = df_scaled.T[df_scaled.T.duplicated()]
duplicates.T
```

[ ]:
```
df_X.drop(['TEST', 'TEST2'], axis=1)
```

[ ]:
```
df_X.describe()
```

7.3 Story 3

https://betterprogramming.pub/3-pandas-functions-to-group-and-aggregate-data-9763a32583bb

[ ]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from sklearn.datasets import load_boston

boston = load_boston()
df = pd.DataFrame(data=boston.data,columns=boston.feature_names)
df['price']=boston.target
df.head()
```

[ ]:
```
df['RM_levels']=df['RM'].apply(lambda x: 'low' if x<5.8 else 'middle' if x<6.6␣
 ↪else 'high')
df['price_levels']=df['price'].apply(lambda x: 'low' if x<17 else 'middle' if␣
 ↪x<25 else 'high')
df[['RM','RM_levels','price','price_levels']].head()
```

[ ]:
```
df['RM_levels'] = pd.cut(df['RM'],bins=[3,5.8,6.6,9],right=False)
df['price_levels'] = pd.cut(df['price'],bins=[5,17,25,51],right=False)
df[['RM','RM_levels','price','price_levels']].head()
```

[ ]:
```
df[df.price_levels=='[17, 25)']
type(df['price_levels'][0])
```

[ ]:
```
l=[[3,5.8,6.6,9],[5,17,25,51]]
lbs1 = ['[{},{})'.format(l[0][i],l[0][i+1]) for i in range(len(l[0])-1)]
lbs2 = ['[{},{})'.format(l[1][i],l[1][i+1]) for i in range(len(l[1])-1)]

df['RM_levels'] = pd.cut(df['RM'],bins=[3,5.8,6.6,9],right=False,labels=lbs1)
df['price_levels'] = pd.
 ↪cut(df['price'],bins=[5,17,25,51],right=False,labels=lbs2)
print(type(df['price_levels'][0]))
```

```
[ ]: df['RM_levels'] = pd.cut(df['RM'],bins=[3,5.8,6.
      ↪6,9],labels=['low','middle','high'],right=False)
     df['price_levels'] = pd.
      ↪cut(df['price'],bins=[5,17,25,51],labels=['low','middle','high'],right=False)
```

7.3.1 Simple Aggregations

```
[ ]: df_price = df.groupby(by=['price_levels']).mean()
     df_price
```

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

```
[ ]: df['price_levels']=df['price_levels'].astype('category')
     df['price_levels'].cat.reorder_categories(['low','middle','high'], inplace=True)

     df['RM_levels']=df['RM_levels'].astype('category')
     df['RM_levels'].cat.reorder_categories(['low','middle','high'], inplace=True)
```

```
[ ]: df_price = df.groupby(by=['price_levels'],as_index=False).mean()
     df_price
```

```
[ ]: df_price = df.groupby(by=['price_levels'],as_index=False)[['price','RM']].
      ↪mean().sort_values(by='price_levels', ascending=True)
     df_price
```

```
[ ]: df_price = df.groupby(by=['price_levels'],as_index=False)[['CRIM','LSTAT']].
      ↪apply(lambda v: v.max()-v.min())
     df_price
```

7.3.2 Multiple Aggregations

```
[ ]: df_price = df.
      ↪groupby(by=['price_levels'],as_index=False)[['price','AGE','CRIM']].
      ↪agg(['mean','min','max','std'])
     df_price
```

```
[ ]: df_price = df.groupby(by=['price_levels'])[['price','AGE','CRIM']].
      ↪agg(['mean','min','max','std']).reset_index()
     df_price
```

```
[ ]: df_price.columns
```

```
[ ]: stats = ['mean','min','max','std']
     df_price.columns = ['price_levels']+['price_{}'.format(stat) for stat in␣
      ↪stats]+['age_{}'.format(stat) for stat in stats]+['age_{}'.format(stat) for␣
      ↪stat in stats]
     df_price
```

[ ]:

```
df_price = df.groupby(by=['price_levels','RM_levels'],as_index=False).
 ↪agg({'CRIM':['count','mean','max','min'],'price':['mean']})
df_price
```