10/24/2023

Pandas :- (will complete this by 26th Stay on fast Track.

- Pandas is a library for data Analysis.
- Extremely powerfull table (DataFrame) system built of Numpy.
- fantastic documentation

What can we do with pandas:
- Tools for reading and writting data between many formats.
- Intelligently grab data based on indexing, logic, subsetting, and more.
- Handle missing data.
- Adjust and restructure data.

Section - Overview
- Series and DataFrame
- Condition filtering and usefull methods.
- Missing data
- Combining DataFrames
- Text method and time method
- Input and Output.

Series - part-one

- A series is a data structure in pandas that hold an array of information along with a named index.
- The named object differentiates this from a simple Numpy array.
- Formal definition : One-dimensional n-darray with axis label.

- Numpy array has index

Index	Data
0	1776
1	224
2	1460

Label Index :	Data
USA	1776
CANADA	1867
MEXICO	1821

Data is still numerically organized

Code #

```
import numpy as np
import pandas as pd

# help(pd.Series)

pd.Series(data=None,
          index=None,
          dtype=None,
          name=None,
          copy=False,
          fastpath=False,)

myindex = ['USA', "Canada", 'Mexico']
mydata = [1776, 1867, 1821]
myseries = pd.Series(data=mydata,
                     index=myindex)

myseries[0]
    out: 1776

myseries['USA']
    out: 1776
```

Series from Dictionary.

```
age = {'Sam':5, 'Frank':10,
       'Spike':7}
pd.Series(age)
```

out!
```
Sam     5
Frank   10
Spike   7
dtype: int64
```

Pandas - Series part 2

```
# Imaginary Sales for 1st and 2nd Quarters for GL
q1 = {'Japan': 80, 'China': 450, 'India': 200, 'USA': 250}
q2 = {'Brazil': 200, 'China': 500, 'India': 210, 'USA': 260}

sales_q1 = pd.Series(q1)
sales_q2 = pd.Series(q2)

sales_q2 = pd.Series
sales_q2
```

```
sales_q1.keys()
```

out: Index(['Japan', 'China', 'India'], dtype='object')

Broadcasting occurs in pandas Series:-

[1, 2] * 2 ⇒ [1, 2, 1, 2]

But in np Broadcasting occurs

```
np.array([1,2]) * 2
```

⇒ [2, 4]

then :-

Sales_q1 * 2

Japan	160
China	900
India	400
USA	500

Sales_q1/100

Sales_q1 + Sales_q2

Brazil	NaN
China	950.0
India	410.0
JAPAN	NaN
USA	510.0

Sales_q1.add(Sales_q2, fill_value = 0)

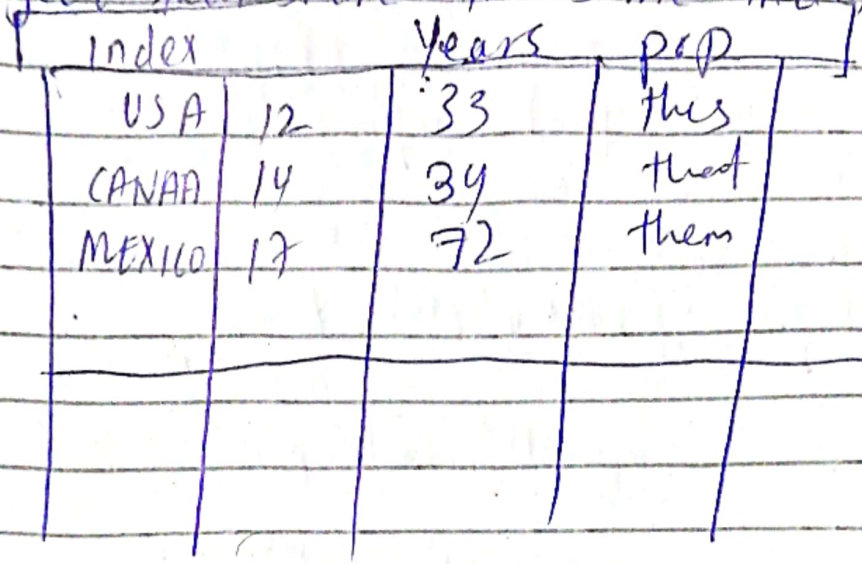
Convert to numerical to float in numerical operation

Pandas - DataFrame

A dataFrame is a table of columns and rows in pandas that we can easily restructure and filter.

Formal Definition :-

A group of pandas series object that share the same index.



This Series Covers first the "basics"

- Create a data frame
- Grab a column or multiple column
- Grab a row or multiple rows.
- Insert a new column or new rows.

Code #

```
import numpy as np
import pandas as pd
```

pd.

```
# np.random.seed(101)
mydata = np.random.randint(
                    0,101,(4,3))

mydata
    array [[ 1, 11, 8 ],
           [ 22, 10, 18],
           [ 95, 9 , 77],
           [ 40, 4 , 63],]

myindex = ['LA','NY','AZ','TX'],

mycoloums = ['Jan', Feb', 'Mar']

df = pd.DataFrame(data = mydata,
                  index = myindex,
                  coloums = mycolorms)

df.info();

# Where my python code available
pwd => full path

LS -> Report all the files in
Current directory
df = pd.read_csv('tips.csv')
```

pd.read_csv("C:\\pdsc. \\download\\tips.c

pwd and LS only in jupyter

part 2 - pandas Dataframe

df.columns - attributs

Index(["total"bill .....])

df.index RangeIndex(start=0, stop=244, step=1)

df.head() - First 5 rows
df.head() → Last 5 rows

df.info() basic info, ram info using

df.describe() calculate statistical values

df.describe().transponse()

part # 3 - pandas Dataframe

type(df['total_bill'])

pandas.core.series

```
mycols = ['total_bills', 'tips']
df[mycols]
```

```
df[['total', 'tips']]
```

```
# New column
(df['tip'] / df['total']) * 100.
```

```
df['tip_percentage'] = (df['tips'] / df['total']) * 100
```

```
df
```

```
np.round(df['total_bills'] / df['tips'], 2)
df.head()
```

```
# Removing columns
df.drop(label=None, axis=1)
```

axis=0 remove row
axis=1 remove column

```
df = df.drop(label=None, axis=1)
```

```
df.shape
(244, 11)
 0    1
axis
```

Range index

Part 4 Dataframe

df.index

df.set_index("payment_ID")

↓

Set index to column

df = df.set_index("payment_ID")
df.head();

df=df.reset_index() — permanent change.
df

df.iloc[0] → numeric.

df.loc['Sun2991'] → Label based

Slice for multple

df.iloc[0:4]

df.loc[['Sunday'],'Sun5260']]

df.drop('Sun2545', axis=0)
df

one_row = df.iloc[0]

df.append(one_row)

df = df.append(one_row)

Condition Filtering :-

Organizing data

Index	Year	POP	GDP
USA	1768	328	20.5
CANADA	1821	38	1.7
MEXICO	321	126	1.22

df["pop"] > 50

return

Series of boolean values

[true
  false]

df[df["pop"] > 50]

- Conditional Filtering :
  - filter by multiple conditions.
- Check against multiple possible values.

Code #

Single condition

```
df[df["total_bills"] > 40]
```

will give us rows only total_bill greater then 40.

```
df[df['sex'] == "Male"]
```

Multiple condition

AND & --- Both cond need true
OR | --- Either condition true

```
(df['total_bill'] > 30) & (df['sex'] == "Male")
```

and, or are used in python too compare two value

But in pandas &, and | are used

& am percent

```
df[(df['day'] == 'Sun') | (df['day'] == 'Sat')
option = ['Saturday', 'Sun']
df[
df[df['day'].isin(option)]
```

.isin

Usefull Methods

part one - Apply Method

- apply()

apply our custom function to each of the series - rows of

```
def lastfour(num):
    str(1234567534)[-4:]
    return str(num)[-4:]

df['last_4'] = df["CC Number"].apply(lastfour)

df["total_bill"].mean()
19.785

def yelp(price):
    if price < 10:
        return '$'
    elif price >= 10 and price < 30:
        return '$$'
    else:
        return '$$$'
df['yelp'] = df['total bill'].apply('yelp')
```

part #2

Apply on Multiple Coluums:-

Lambda function

```
def simple(num):
    return num*2
```

↓ to lamda

```
Lambda num: num*2
```

```
df['total_bill'].apply(lambda num: num*2)
```

apply with multiple colowns:-

```
def quality(total_bill, tip):
    if tip/total_bill > 0.25:
        return "Generous"
    else:
        'other'
quality(16.99, 1.01)
```

```
df[['total_bills', 'tips']].apply(lambda
    df: quality(df['total_bills'], df['tips']
    , axis=1)
```

Run alot faster:-

```
df['Quality'] = np.vectorize(quality)
    (df['total_bill'], df['tips'])
```

np.vectorize just a convenience

import timeit

Usefull methods part #3

Describing and Sorting methods

```
df = pd.read_csv('tips.csv')
df.describe()
df.sort_values('tip')
```

a defaut assending

```
df.sort_values(['tips', 'size'])
```

first it sort by tip then If they are same then it sort by size colum.

```
df['total_bill'].max()        .min()
df['total_bill'].idxmax()     .idxmin()
170
df.iloc[170]
```

df.corr() — correlation between numerical column

df['sex'].value_counts()
on column

df['day'].unique()
↳ unique values
df['day'].nunique()

df.head()

df['sex'].replace('female','F')
df['sex'].replace(['female','male'],['F','M'])

mymap = {'female':'F','Male':'M'}

df['sex'].map(mymap)
↳ lots of item

df.duplicated() — duplicates

simple-df.drop-duplicates()

df['total_bill'].between(10,20, inclusive=True)

df.nlargest(10, 'tips')

df.nsmallest(2, 'tips')

---

df.sample(5) — return random rows

df.sample(frac=0.1) will return 10% of the sample

Missing Data

- when reading in missing values, pandas will display them as NAN values.

- There are also newer version specific null pandas value such as pd.NAT to imply the value missing should be a timestamp.

Option for missing Data

- keep it
- Remove it
- Replace it

Keep the missing data

- pros
  - Easiest to data
  - Does not manipulate or change the true data

Cons:
- many methods does not support NAN
- Often there are reasonable guesses.

- Dropping or Removing the missing data
  - easy to do
  - can be based on Rule

- CONS:
  - potential to lose ~~also~~ a lot of data or usefull information -
  - limit trained data on Future prediction

- Removing or Dropping missing data -:
- Dropping a row
  - make sense when info is missing -

Filling the missing data :-
- pros:
  - potential to save alot of data for use in training a model -
- Cons:
  - Hardest to do and somewhat Arbitary -
  - potential to lead to false conclusion -

Missing Data part-two

np.nan — old version
pd.NA

pd.NaT → time stamp

np.nan == np.nan
False

np.nan is np.nan
True

```
df = pd.read_csv('movies_score.csv')
df.head()
```

df.isnull() — Show null value
df.notnull() → Show not values

```
df[df['pre_movie_score'].notnull()]
```

```
df['pre_movies_score'].isnull()]
```

```
df[(df['pre_movie_score'].isnull()) & (df['first_Name'].notnull())]
```

Keep the data
Fill Or DROP DATA
FILL DATA

```
df.dropna()                    # at least not null value
df.dropna(thresh=1)

df.dropna(axis=1)
df.dropna(subset=['last name'])
```

Fill the Data

```
df.fillna('New value!:')
df['pre-mones'].fillna(0)
df
df['pre-mone-score'].mean()
df['pre-mone-score'].fillna(df.mean)
```

Interpolation to fill missing values :-

```
airline_fix = {'first':10, 'business':np.nan, 'economy-plus':30}
ser = pd.Series(airline_fix)
ser
```

ser

ser.interpolate()

14. Groupby Operation

- A groupby(): operation allow us to examine data on a per category basis :

category	Datavalue
A	10
A	5
B	2
B	4
C	12
C	6

A	10
A	5

B	2
B	4

C	12
C	6

decide on Aggregate function
- sum()

category	Result
A	15
B	6
C	18

- mean()
- count()

- Note: In pandas calling .groupby() by itself creates a "Lazy" groupby object waiting to be evaluated by an aggregate method call-

```
df = pd.read("mpg.csv")

df['model_year'].value_counts()

df.groupby('model_year').mean()['mpg']

df.groupby(['model_year','cylinders'])
    .mean()
    .index()

df.groupby('model_year').describe()
    .transpose()

year_cyl = df.groupby(['model_year',
    'cylinders']).mean()

year_cyl.index.levels

year_cyl.index.names
    [[70, 82]]

year_cyl.loc[70]

year_cyl.loc[(70,4)]
```

Groupby multilevel indexes

Cross-section

```
year_cyl.xs(key=70, level='model_year')

year_cyl.loc[[70,80]]

year_cyl.xs(key=4, levels='cylinders')

df[df['cylinder'].isin([6,8])].
    groupby(['model_year','cylinders'])
    .mean()

year_cyl.swaplevel()   — swap levels

year_cycle.sort_index(level='model year',
                      assending=False)

df.agg(['std','mean'])['mpg']

df.agg({'mpg':['max','mean'],'weight':
        ['mean','std']}
```

16 - Combining Dataframe - Concatination

- Often the data you need exist in two seperate sources, fortunately, pandas make it easier to ~~concatenate~~ combine together using pd.Concat() call.

Concatination is simply "pasting" the two DataFrame together, by columns:

	Year	POP
USA	1776	328
CAN	1867	38
MEX	1821	126

⟷

	GDP	Per-
USA	2.65	75%
CAN	1.7	NAN
MEXICO	1.22	25%

	Year	POP	GDP	Percent
USA	1776	328	20.5	25%
LANADA	1867	38	1.7	NAN
MEXICO	1821	126	1.22	25%

Also can be performed on rows -

MEX
Ca

USA
Ind

Code #

dataF1 =

dataf2 =

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	A2	C2	D2
3	A3	A4	C3	D4

pd.concat([one, two], axis=1)

pd.concat([one, two], axis=0)

↓

One way to resolve it was to rename

two.columns = one.columns

↓

pd.concat([one, two], axis=0)

mydf = pd.concat

mydf.index = range(len(df))

Combining Data frame

Inner Merge -17

- merge()

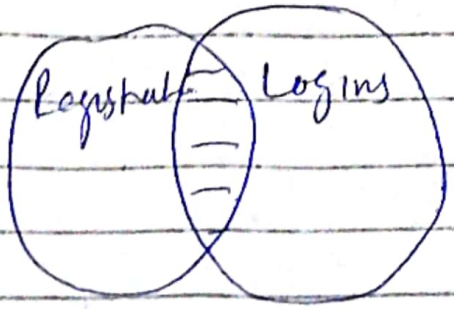
The merge method takes in a key argument labeled 'how'.

- There are 3 main ways of merging tables together using how parameter;
  - Inner
  - Outer
  - Left or Right

Registrations	
reg-id	name
1	Andrew
2	Bob
3	Charlie
4	Dan

Logins	
login_id	name
1	Xavier
2	Andrew
3	Yolanda
4	Bob
5	

on = "name".
how = "inner"



pd.merge(registration, logins, how='inner', on='name')

Result

reg-id	name	log-id
1	Andrew	2
2	Bob	4

registrations = pd.DataFrame([ ])

logins = pd.DataFrame( )

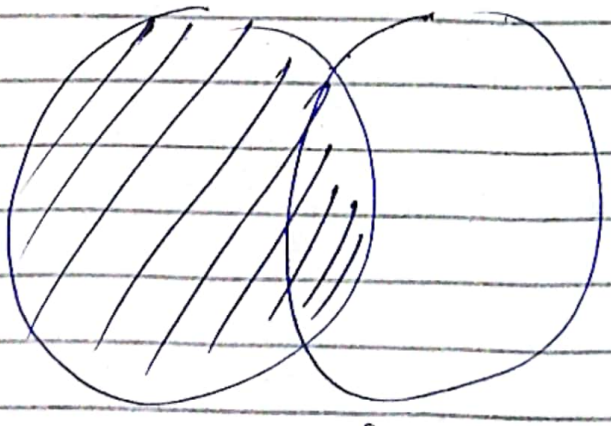
pd.merge(registrations, logins, how="inner", on="name")

	reg-id	name	log-id
0	1	Andrew	2
1	2	Bobo	4

Left - Right merge (18)

- Order matters in ~~def~~ pd.merge()

Left merge



every thing in left name column should shows

pd.merge(regist, logs, how="left", on='name

Result

reg-id	name	log-id
1	Andrew	2
2	Bob	4
3	charlie	Nan
4	David	Nan

Right –

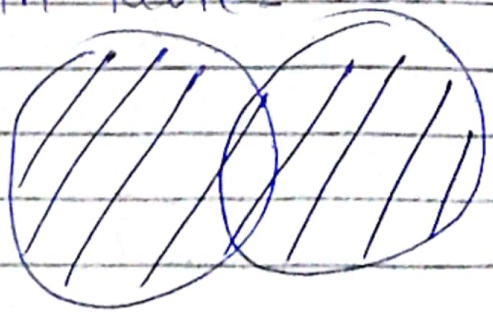
pd.merge(regist, logins, how='right', on='name')

Result

	name	
1	Andrew	2
		4
2	Bob	1
Nan	Xavier	
Nan	Yolanda	3

Outer-Merge (19)

Setting how = "outer" allow as to include everything present in both table.



All the values

order in outer will not matter-

```
pd.merge(logins, registration, how='outer',
                                on='name')
registration = registrations.set_index('name')
registrations

pd.merge(registration, logins,
                        left_index=True,
                        right_on='name',
registrations                   how='inner')

registrations.columns = ['reg-name', 'reg-id']
```

for different column names

```
pd.merge(registrations, logins, how="inner",
                        right_on="name",
                        left_on="reg_name")
result.drop('reg-name', axis=1)
```

for same columns

```
pd.merge(registrations, logins, how='inner',
                        on='name')
                        suffixes=('_reg', '_log')
```

20- Text Methods for string Data

- Often text data needs to be cleaned or manipulated for processing.
- While we can always use a custom apply() function for these task, pandas comes with many built-in string methods calls.
- Lets learn how to use them!

Code #

```
import pandas as pd
import numpy as np

name = 'jose'
email = 'jose@@mail.com'
email.split('@')

name = pd.Series(['andrew', 'bobo', 'claire',
                  'david', 5, 2])

names.str.upper()
.email.isdigit()


names.str.isdigit()
```

```
tech_finance = ['GOOG;APPL;AMZN;
    'JPM,BAC,GS']
len(tech_finance)

tickers = pd.Series(tech_finacee)
tickers.str.split(';')

tickers.str.split(';').str[0]

tickers.str.split(';', expand=True)

messy_names = pd.Series(['andrew  ',
                         "bO;bO",'  claire  ']

messy_names.str.replace(';',"").str.strip().
    str.capitalize()
```

21 - Time Methods in pandas

- Basic python have a datetime object containing date and time information.
- pandas allow us to easily extract information from a datetime object to use feature engineering -

.dt methods

Extract information from the timestamp, such as:
- Day of the week
- Weekend vs Weekday
-

```
from datetime import datetime

myyear = 2015
mymonth = 1
myday = 1
myhour = 2
mymin = 30
mysec = 15

mydate = datetime(myyear, mymonth, myday)
mydate

myser = pd.Series(['Nov 3, 1990',
                   '2000-01-01', None])
```

```
pd.to_datetime(myseo)
0    1990-11-03
1    2000-01-01
2    NaT
```

timeses[0].year

```
euro_date = '31-12-20000'
pd.to_datetime(euro_date)
```

```
pd.to_datetime(euro_date, dayfirst=True)
```

```
style_date = '12--Dec--2000'
pd.to_datetime(style_date, format
                    = '%d--%b--%Y')
```

```
custom_date = "12th of Dec 2000"
pd.to_datetime(custom_date)
```

```
sales = pd.read_csv('plate1.csv')
```

22 - Pandas Input and Output

- Note #

You need to know exactly directory location and correct files name -

- You may need passwords or permission for certain data inputs (e.g. a SQL database password).

Video lecture
- CSV files
- HTML Tables
- Excel files
- SQL Database

Code #

```
import pandas
pwd , ls

import OS
os.getcwd() ⟶ current path display

df= pd.read_csv('example.csv')

df = pd.read_csv('example.csv', index_col=0

df.to_csv('C:\\Users\\Maravalla\\
                newfiles.csv')
df.to_csv('newfiles.csv', index=False)
```

23 - pandas HTML Table

website display tabular information through the use of HTML tables tags.
- </table>

- pandas has the ability to automatically convert these HTML tables into a DataFrame.

- Note every table in a website is availble through HTML tables.

```
import pandas as pd
conda install lxml
pip install lXml

url = "https://....."
pd.read_html(url)
table = pd.read_html(url)
len(tables)
table[0]
tables[1]
world_topten = tables[0]
world_topten["World c...  ()]
world_topten
world_topten = world_topten.drop('#', axis=1)
```

world.topten = ['country', '2000', '2015', '2030', ['Shares

world_topten.to_html('sampletable_html',
index=False)

24- Excel Files

- pandas can read and write excel files -
- Important Note!
  - pandas can read and write in raw data. it is not able to read in macros, visualization, or formulas created inside of spreadsheet.

- pandas treats an Excel Workbook as a dictionary with the key being the sheet name and the value being the DataFrame representing the sheets itself

Note! Using pandas with excel requires aditional libraries.
excel requires libraries
- openpyxl → .xlsx
- xlrd .xls

Code #1

```
df = pd.read_excel('my_excel.xlsx',
                   sheet_name = 'first_sheet')
```

```
wb = pd.ExcelFile('my_excel_file.xlsx')
wb.sheet_names
['first_sheets']
```

```
excel_sheets: dict = pd.read_excel("...xlsx",
                                   sheet_name=None)
```

If set to None then Create a dictionary and saves everything to it

```
type(excel_sheet_dict)
excel_sheets.dicts.keys()
```

```
our_df.to_excel('example.xlsx', sheet_name
                = 'first_sheet', index=False)
```

25- Pandas Input and Output
SQL - DataBase

- pandas can read and write to various SQL engines through use of a drives and the sqlalchemy python library -
- So how this work?

Step 1:
- figure out what sequel engine you are going connecting to, for just a few examples :-
  - postgreSQL
  - MySQL
  - MS SQL Server

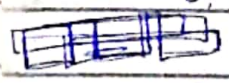
Step 2:
- Install appropriate python driver library (Most likely requires a Google Search);
  - postgreSQL - psycopg2
  - MySQL - pymsql
  - MS SQL Server - pyodbc

Step 3:
Use sqlalchemy library to connect to your SQL database with the driver:

Step 4:

- Use the Sqlalchemy driver connection with pandas read_sql method.
- pandas can read in entires tables as a DataFrame or actual parse a SQL query through the connection:
- SELECT * FROM table;

Using SQLite we can easily install database



pip install sqlalchemy

temp database inside ram

```
from sqlalchemy import create_engine
temp_db = create_engine('sqlite:///:memory:')

pd.DataFrame(data = np.random.randint
                (low=0, high=100,
                 size=(4,4)),
             columns = ['a','b','c',])

df.to_sql(name='new_table', con=temp_db)

pd.read_sql(sql='new_tables', con=temp_db)
```

new_df

index	a	b	c	d
0	93	77	80	
1	98	94	97	
2	63	80	63	

pd.read_sql_query('SELECT a, c FROM
result = newtable', con=temp_db)

26- Pivot Tables

- Allow you to reorganize data, refactoring cell based on columns and a new index.
- This is best shown visually....

- A dataFrame with repeated value can be pivoted for a reorganization and clarity-

df.pivot(index='foo', columns='bar', values='baz')

Code #

pd.read_csv('sales_funnel_CRM.csv')

licences = df[['company', 'product', 'licence']]

pd.pivot(data=licences, index='company',
columns='products', values='Licences')

Pd.pivot_table(df, index="company", aggfunc='sum',
values=['licence', 'sales price'])

pd.pivot_table(df, index=['Account Manager',
'Contact'], values=['Sales Price'],
aggfunc='sum'], fill_value=0)
columns=['product'],

Practice Exercise:-

1) How many rows are there?

2) Is there missing data? if so which column has the most missing data?

3) Drop the "company" column from the dataset?

4) What are the most common types of country codes in dataset

5) What is the name of person. who paid the highest ADR (Average daily rate)? How much was their ADR?

6) The adr is average daily rate for a person's stay at the hotel. What is the mean adr across all the hotel stays in the dataset?

7) What is the average number of (Mean) night for a stay across the entire data set? Feel free to round this to 2 decimal points.

8) What is the average total cost for a stay in dataset? Not average daily cost, but total stay cost. Feel free to round this to 2 decimal points.

9) What are the names and emails of people who made 5 "special request"?

10) What percentage of hostel stay were classified as "repeat guest"? (Do not based this on name of person but instead of is_repeated guest. column

11) What are the ~~most~~ top 5 common last name in the dataset?

(b) What are the names of the people who have booked the most number of children and babies for their stay? (Don't worry about cancelled just r)

13) What are top 3 most common area codes in phone number? (first 3 digit)

14) How many arrival took place between first and the 15th of the month: inclusive of 1 and 15: can you do it in one line of pandas code!

Hard Bonus Task

15) Create a table for counts for each day of week that people arrived. (Eg. 5000 arrival were on a Monday, 3000 were on Tuesday. etc)