aula05_funcoes

February 28, 2021

1 5.0. Aula 05 - Funções e Estrutura de Código

[28]:
```
import pandas as pd

# load dataset
data = pd.read_csv( 'kc_house_data.csv' )
```

[29]:
```
# data dimension
print( 'Number of Rows:{}'.format( data.shape[0] ) )
print( 'Number of Columns {}'.format( data.shape[1] ) )
```

```
Number of Rows:21613
Number of Columns 21
```

[30]:
```
# data types
data.dtypes
```

[30]:
```
id                int64
date             object
price           float64
bedrooms          int64
bathrooms       float64
sqft_living       int64
sqft_lot          int64
floors          float64
waterfront        int64
view              int64
condition         int64
grade             int64
sqft_above        int64
sqft_basement     int64
yr_built          int64
yr_renovated      int64
zipcode           int64
lat             float64
long            float64
sqft_living15     int64
sqft_lot15        int64
```

```
dtype: object
```

[31]:
```
# convert object to date
data['date'] = pd.to_datetime( data['date'] )
data.dtypes
```

[31]:
```
id                        int64
date             datetime64[ns]
price                   float64
bedrooms                  int64
bathrooms               float64
sqft_living               int64
sqft_lot                  int64
floors                  float64
waterfront                int64
view                      int64
condition                 int64
grade                     int64
sqft_above                int64
sqft_basement             int64
yr_built                  int64
yr_renovated              int64
zipcode                   int64
lat                     float64
long                    float64
sqft_living15             int64
sqft_lot15                int64
dtype: object
```

[32]:
```
# descriptive statistics
num_attributes = data.select_dtypes( include=['int64', 'float64'] )
```

[33]:
```
import numpy as np

# central tendency
media = pd.DataFrame( num_attributes.apply( np.mean ) )
mediana = pd.DataFrame( num_attributes.apply( np.median ) )
std = pd.DataFrame( num_attributes.apply( np.std ) )

# dispersion
std = pd.DataFrame( num_attributes.apply( np.std ) )
max_ = pd.DataFrame( num_attributes.apply( np.max ) )
min_ = pd.DataFrame( num_attributes.apply( np.min ) )

df1 = pd.concat( [max_, min_, media, mediana, std ], axis=1 ).reset_index()
df1.columns = ['attributes', 'mean', 'median', 'std', 'max', 'min']
df1
```

[33]:

	attributes	mean	median	std	max	min
0	id	9.900000e+09	1.000102e+06	4.580302e+09	3.904930e+09	2.876499e+09
1	price	7.700000e+06	7.500000e+04	5.400881e+05	4.500000e+05	3.671187e+05
2	bedrooms	3.300000e+01	0.000000e+00	3.370842e+00	3.000000e+00	9.300403e-01
3	bathrooms	8.000000e+00	0.000000e+00	2.114757e+00	2.250000e+00	7.701453e-01
4	sqft_living	1.354000e+04	2.900000e+02	2.079900e+03	1.910000e+03	9.184196e+02
5	sqft_lot	1.651359e+06	5.200000e+02	1.510697e+04	7.618000e+03	4.141955e+04
6	floors	3.500000e+00	1.000000e+00	1.494309e+00	1.500000e+00	5.399764e-01
7	waterfront	1.000000e+00	0.000000e+00	7.541757e-03	0.000000e+00	8.651520e-02
8	view	4.000000e+00	0.000000e+00	2.343034e-01	0.000000e+00	7.662998e-01
9	condition	5.000000e+00	1.000000e+00	3.409430e+00	3.000000e+00	6.507280e-01
10	grade	1.300000e+01	1.000000e+00	7.656873e+00	7.000000e+00	1.175432e+00
11	sqft_above	9.410000e+03	2.900000e+02	1.788391e+03	1.560000e+03	8.280718e+02
12	sqft_basement	4.820000e+03	0.000000e+00	2.915090e+02	0.000000e+00	4.425648e+02
13	yr_built	2.015000e+03	1.900000e+03	1.971005e+03	1.975000e+03	2.937273e+01
14	yr_renovated	2.015000e+03	0.000000e+00	8.440226e+01	0.000000e+00	4.016699e+02
15	zipcode	9.819900e+04	9.800100e+04	9.807794e+04	9.806500e+04	5.350379e+01
16	lat	4.777760e+01	4.715590e+01	4.756005e+01	4.757180e+01	1.385605e-01
17	long	-1.213150e+02	-1.225190e+02	-1.222139e+02	-1.222300e+02	1.408251e-01
18	sqft_living15	6.210000e+03	3.990000e+02	1.986552e+03	1.840000e+03	6.853754e+02
19	sqft_lot15	8.712000e+05	6.510000e+02	1.276846e+04	7.620000e+03	2.730355e+04

[39]:
```
# data dimension
print( 'Number of Rows:{}'.format( df1.shape[0] ) )
print( 'Number of Columns {}'.format( df1.shape[1] ) )
```

```
Number of Rows:20
Number of Columns 6
```

[40]:
```
#Crie uma nova coluna chamada: "house_age"
#- Se o valor da coluna "date" for maior que 2014-01-01 => 'new_house'
#- Se o valor da coluna "date" for menor que 2014-01-01 => 'old_house'
data['house_age'] = 'NA'
data.loc[data['date'] > pd.to_datetime( '2014-01-01' ), 'house_age'] =␣
 ↪'new_house'
data.loc[data['date'] < pd.to_datetime( '2014-01-01' ), 'house_age'] =␣
 ↪'old_house'

# data dimension
print( 'Number of Rows:{}'.format( data.shape[0] ) )
print( 'Number of Columns {}'.format( data.shape[1] ) )
```

```
Number of Rows:20
Number of Columns 29
```

[41]:
```
# Crie uma nova coluna chamada: "dormitory_type"
#- Se o valor da coluna "bedrooms" for igual à 1 => 'studio'
#- Se o valor da coluna "bedrooms" for igual a 2 => 'apartament'
#- Se o valor da coluna "bedrooms" for maior que 2 => 'house'
data['dormitory_type'] = 'NA'
for i in range( len( data ) ):
    if data.loc[i, 'bedrooms'] == 1:
        data.loc[i, 'dormitory_type'] = 'studio'

    elif data.loc[i, 'bedrooms'] == 2:
        data.loc[i, 'dormitory_type'] = 'apartment'

    else:
        data.loc[i, 'dormitory_type'] = 'house'

# data dimension
print( 'Number of Rows:{}'.format( data.shape[0] ) )
print( 'Number of Columns {}'.format( data.shape[1] ) )
```

```
Number of Rows:20
Number of Columns 29
```

[42]:
```
# Exemplo da Aplicação 01: Definir os níveis de preços
# 0 até 321.950           = Level 0
# Entre 321.950 e 450.000 = Level 1
# Entre 450.000 e 645.000 = Level 2
# Acima de 645.000        = Level 3

# define level of prices
```

```
for i in range( len( data ) ):
    if data.loc[i, 'price'] <= 321950:
        data.loc[i, 'level'] = 0

    elif ( data.loc[i,'price'] > 321950 ) & ( data.loc[i,'price'] <= 450000 ):
        data.loc[i, 'level'] = 1

    elif ( data.loc[i,'price'] > 450000 ) & ( data.loc[i,'price'] <= 645000 ):
        data.loc[i, 'level'] = 2

    else:
        data.loc[i, 'level'] = 3

# data dimension
print( 'Number of Rows:{}'.format( data.shape[0] ) )
print( 'Number of Columns {}'.format( data.shape[1] ) )
```

```
Number of Rows:20
Number of Columns 29
```

[43]:
```
from geopy.geocoders import Nominatim

# initializa API
geolocator = Nominatim( user_agent='geopiExercises' )

# only 10 rows
data = data.head(20)

# Create empty rows
data.loc[:, 'road'] = 'NA'
data.loc[:, 'house_number'] = 'NA'
data.loc[:, 'city'] = 'NA'
data.loc[:, 'county'] = 'NA'
data.loc[:, 'state'] = 'NA'

for i in range( len( data ) ):
    print( 'Loop: {}/{}'.format( i, len( data ) ) )
    # make request
    query = str( data.loc[i, 'lat'] ) + ',' + str( data.loc[i, 'long'] )
    response = geolocator.reverse( query )

    # parse data
    if 'house_number' in response.raw['address']:
        data.loc[i, 'house_number'] = response.raw['address']['house_number']

    if 'road' in response.raw['address']:
        data.loc[i, 'road'] = response.raw['address']['road']
```

```
    if 'city' in response.raw['address']:
        data.loc[i, 'city'] = response.raw['address']['city']

    if 'county' in response.raw['address']:
        data.loc[i, 'county'] = response.raw['address']['county']

    if 'state' in response.raw['address']:
        data.loc[i, 'state'] = response.raw['address']['state']


# data dimension
print( 'Number of Rows:{}'.format( data.shape[0] ) )
print( 'Number of Columns {}'.format( data.shape[1] ) )
```

```
Loop: 0/20
Loop: 1/20
Loop: 2/20
Loop: 3/20
Loop: 4/20
Loop: 5/20
Loop: 6/20
Loop: 7/20
Loop: 8/20
Loop: 9/20
Loop: 10/20
Loop: 11/20
Loop: 12/20
Loop: 13/20
Loop: 14/20
Loop: 15/20
Loop: 16/20
Loop: 17/20
Loop: 18/20
Loop: 19/20
Number of Rows:20
Number of Columns 29
```

[37]:
```
import plotly.express as px

# map
houses = data[['id', 'lat', 'long', 'price', 'level']].copy()
fig = px.scatter_mapbox( houses,
                         lat="lat",
                         lon="long",
                         color="level",
                         size="price",
```

```
                        color_continuous_scale=px.colors.cyclical.IceFire,
                        size_max=15,
                        zoom=10)

fig.update_layout(mapbox_style="open-street-map")
fig.update_layout(height=600, margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

1.1 5.2. Refazendo o código seguindo o ETL

[1]:
```
# ==================
# Bibliotecas
# ==================
import pandas as pd
import numpy as np
from geopy.geocoders import Nominatim
import plotly.express as px

# ==================
# Functions
# ==================
def show_dimensions( data ):
    print( 'Number of Rows:{}'.format( data.shape[0] ) )
    print( 'Number of Columns {}'.format( data.shape[1] ) )

    return None

def descriptive_statistics( data ):
    # descriptive statistics
    num_attributes = data.select_dtypes( include=['int64', 'float64'] )

    # central tendency
    media = pd.DataFrame( num_attributes.apply( np.mean ) )
    mediana = pd.DataFrame( num_attributes.apply( np.median ) )
    std = pd.DataFrame( num_attributes.apply( np.std ) )

    # dispersion
    std = pd.DataFrame( num_attributes.apply( np.std ) )
    max_ = pd.DataFrame( num_attributes.apply( np.max ) )
    min_ = pd.DataFrame( num_attributes.apply( np.min ) )

    df1 = pd.concat( [max_, min_, media, mediana, std ], axis=1 ).reset_index()
    df1.columns = ['attributes', 'mean', 'median', 'std', 'max', 'min']

    return df1
```

```
def create_new_attributes( data, geodata=False ):
    #Crie uma nova coluna chamada: "house_age"
    data.loc[data['date'] > pd.to_datetime( '2014-01-01' ), 'house_age'] =␣
↪'new_house'
    data.loc[data['date'] < pd.to_datetime( '2014-01-01' ), 'house_age'] =␣
↪'old_house'


    # Crie uma nova coluna chamada: "dormitory_type"
    for i in range( len( data ) ):
        if data.loc[i, 'bedrooms'] == 1:
            data.loc[i, 'dormitory_type'] = 'studio'

        elif data.loc[i, 'bedrooms'] == 2:
            data.loc[i, 'dormitory_type'] = 'apartment'

        else:
            data.loc[i, 'dormitory_type'] = 'house'

    # define level of prices
    for i in range( len( data ) ):
        if data.loc[i, 'price'] <= 321950:
            data.loc[i, 'level'] = 0

        elif ( data.loc[i,'price'] > 321950 ) & ( data.loc[i,'price'] <= 450000␣
↪):
            data.loc[i, 'level'] = 1

        elif ( data.loc[i,'price'] > 450000 ) & ( data.loc[i,'price'] <= 645000␣
↪):
            data.loc[i, 'level'] = 2

        else:
            data.loc[i, 'level'] = 3

    if geodata == True:
        # initializa API
        geolocator = Nominatim( user_agent='geopiExercises' )

        # only 10 rows
        data = data.head(20)

        # Create empty rows
        data.loc[:, 'road'] = 'NA'
        data.loc[:, 'house_number'] = 'NA'
        data.loc[:, 'city'] = 'NA'
        data.loc[:, 'county'] = 'NA'
```

```
        data.loc[:, 'state'] = 'NA'

        for i in range( len( data ) ):
            print( 'Loop: {}/{}'.format( i, len( data ) ) )
            # make request
            query = str( data.loc[i, 'lat'] ) + ',' + str( data.loc[i, 'long'] )
            response = geolocator.reverse( query )

            # parse data
            if 'house_number' in response.raw['address']:
                data.loc[i, 'house_number'] = response.
↪raw['address']['house_number']

            if 'road' in response.raw['address']:
                data.loc[i, 'road'] = response.raw['address']['road']

            if 'city' in response.raw['address']:
                data.loc[i, 'city'] = response.raw['address']['city']

            if 'county' in response.raw['address']:
                data.loc[i, 'county'] = response.raw['address']['county']

            if 'state' in response.raw['address']:
                data.loc[i, 'state'] = response.raw['address']['state']

    return data


def draw_map( data ):
    # map
    houses = data[['id', 'lat', 'long', 'price', 'level']].copy()
    fig = px.scatter_mapbox( houses,
                             lat="lat",
                             lon="long",
                             color="level",
                             size="price",
                             color_continuous_scale=px.colors.cyclical.IceFire,
                             size_max=15,
                             zoom=10)

    fig.update_layout(mapbox_style="open-street-map")
    fig.update_layout(height=600, margin={"r":0,"t":0,"l":0,"b":0})

    return fig


if __name__ == "__main__":
```

```
    # ==================
    # Extraction
    # ==================
    # load dataset
    data = pd.read_csv( 'kc_house_data.csv' )
    show_dimensions( data )

    # ==================
    # Transformation
    # ==================
    # convert object to date
    data['date'] = pd.to_datetime( data['date'] )

    # make descriptive analysis
    descriptive_statistics( data )

    # create new attributes
    create_new_attributes( data, geodata=False )

    # ==================
    # Load
    # ==================
    # draw a map
    fig = draw_map( data )
    fig.show()
```

```
/Users/meigarom.lopes/.pyenv/versions/3.8.0/envs/pythondozeroaods/lib/python3.8/
site-packages/pandas/compat/__init__.py:97: UserWarning: Could not import the
lzma module. Your installed Python is incomplete. Attempting to use lzma
compression will result in a RuntimeError.
  warnings.warn(msg)

Number of Rows:21613
Number of Columns 21
```

[ ]:

[ ]: