Intro to Data Science - Applying Function to Data

March 25, 2020

[1]:

```
# This Python 3 environment comes with many helpful analytics libraries␣
 ↪installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/
 ↪docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
 ↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.
```

```
/kaggle/input/california-housing-prices/housing.csv
```

This is the post about **Introduction to Data Scinece**, I am going to write about* handling tabular/dataframe* data in python3.

0.0.1 Importing Data

[2]:

```
import pandas as pd
df = pd.read_csv('/kaggle/input/california-housing-prices/housing.csv')
display(df.head())
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0
```

```
   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
```

During data analysis, we need to use our data to perform some calculations and generate some new data or output from it. Pandas makes it very easy to apply user-defined operations, in Python terminology, on individual data items, rows, and columns of a dataframe.

Pandas has an **apply** function which applies the provided function to the data. One of the reasons for the success of pandas is how fast the apply function performs.

In the Dataset, the field **median_income** has values which are written in tens of thousands of dollars. During analysis, we might want to convert this to Dollars. Let's see how we can do that with the apply function.

[3]:

```python
def convert(n):
    return n * 10000

converted = df['median_income'].apply(convert)
display(converted.head())

# update value
df['median_income'] = converted
display(df.head())
```

```
0    83252.0
1    83014.0
2    72574.0
3    56431.0
4    38462.0
Name: median_income, dtype: float64
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0        83252.0            452600.0        NEAR BAY
1      2401.0      1138.0        83014.0            358500.0        NEAR BAY
2       496.0       177.0        72574.0            352100.0        NEAR BAY
3       558.0       219.0        56431.0            341300.0        NEAR BAY
4       565.0       259.0        38462.0            342200.0        NEAR BAY
```

0.1 ### Converting numerical values to categories

During analysis, sometimes we want to classify our data into separate classes based on some criteria. For instance, we might want to separate these housing blocks into three distinct categories based on the median income of the households i.e.

- High-incomes
- Moderate-incomes
- Low-incomes

[4]:
```
def category(n):
    value = n / 10000
    if value > 10:
        return 'high-income'
    elif value > 2 and value < 10:
        return 'moderate-income'
    else:
        return 'low-income'

categories = df['median_income'].apply(category)
df['income-category'] = categories
display(df.head())

print(df['income-category'].value_counts())
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity  \
0       322.0       126.0        83252.0            452600.0        NEAR BAY
1      2401.0      1138.0        83014.0            358500.0        NEAR BAY
2       496.0       177.0        72574.0            352100.0        NEAR BAY
3       558.0       219.0        56431.0            341300.0        NEAR BAY
4       565.0       259.0        38462.0            342200.0        NEAR BAY

   income-category
0  moderate-income
1  moderate-income
2  moderate-income
3  moderate-income
4  moderate-income
```

```
moderate-income    17874
low-income          2458
high-income          308
Name: income-category, dtype: int64
```