CC - AI

Programación para la Computación Científica - IA

Análisis de Datos (Primera Sesión)

Pandas

Universidad Sergio Arboleda
Prof. John Corredor

Agenda

- Desarrollo de una rutina de análisis de datos
- Reducir la memoria cambiando los tipos de datos
- Seleccionando el más pequeño de los más grandes
- Seleccionando el más grande de cada grupo mediante la clasificación

- Desarrollo de una rutina de análisis de datos

Es importante considerar los pasos que usted, como analista, toma cuando se encuentra por primera vez con un "conjunto de datos" después de importarlo al espacio de trabajo como un DataFrame.

Se plantean las preguntas:
¿Existe un conjunto de tareas que suele llevar a cabo para examinar primero los datos?

¿Conoce todos los tipos de datos posibles?

Exploratory Data Analysis (EDA)

Análisis exploratorio de datos: es un término utilizado para abarcar todo el proceso de análisis de datos sin el uso formal de procedimientos de pruebas estadísticas. Gran parte de la EDA implica la visualización de las diferentes relaciones entre los datos para detectar patrones interesantes y desarrollar hipótesis.

```
import pandas as pd
import numpy as np
from IPython.display import display
pd.options.display.max_columns = 50
college = pd.read_csv('data/college.csv')

college.head()


college.shape()
```

```
with pd.option_context('display.max_rows', 8):
    display(college.describe(include=[np.number]).T)
```

Obtener estadísticas resumidas para las columnas numéricas y transponer el DataFrame para una salida más legible.

	count	mean	std	min	25%	50%	75%	max
HBCU	7164.0	0.014238	0.118478	0.0	0.0000	0.00000	0.000000	1.0
MENONLY	7164.0	0.009213	0.095546	0.0	0.0000	0.00000	0.000000	1.0
WOMENONLY	7164.0	0.005304	0.072642	0.0	0.0000	0.00000	0.000000	1.0
RELAFFIL	7535.0	0.190975	0.393096	0.0	0.0000	0.00000	0.000000	1.0
...	...	...	...	...	...	...	...	...
CURROPER	7535.0	0.923291	0.266146	0.0	1.0000	1.00000	1.000000	1.0
PCTPELL	6849.0	0.530643	0.225544	0.0	0.3578	0.52150	0.712900	1.0
PCTFLOAN	6849.0	0.522211	0.283616	0.0	0.3329	0.58330	0.745000	1.0
UG25ABV	6718.0	0.410021	0.228939	0.0	0.2415	0.40075	0.572275	1.0

college.info()

Enumere el tipo de datos de cada columna, el número de valores "non-missing" y el uso de la memoria.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7535 entries, 0 to 7534
Data columns (total 27 columns):
INSTNM                  7535 non-null object
CITY                    7535 non-null object
STABBR                  7535 non-null object
HBCU                    7164 non-null float64
...
PCTFLOAN                6849 non-null float64
UG25ABV                 6718 non-null float64
MD_EARN_WNE_P10         6413 non-null object
GRAD_DEBT_MDN_SUPP      7503 non-null object
dtypes: float64(20), int64(2), object(5)
memory usage: 1.6+ MB
```

```
college.describe(include=[np.object, pd.Categorical]).T
```

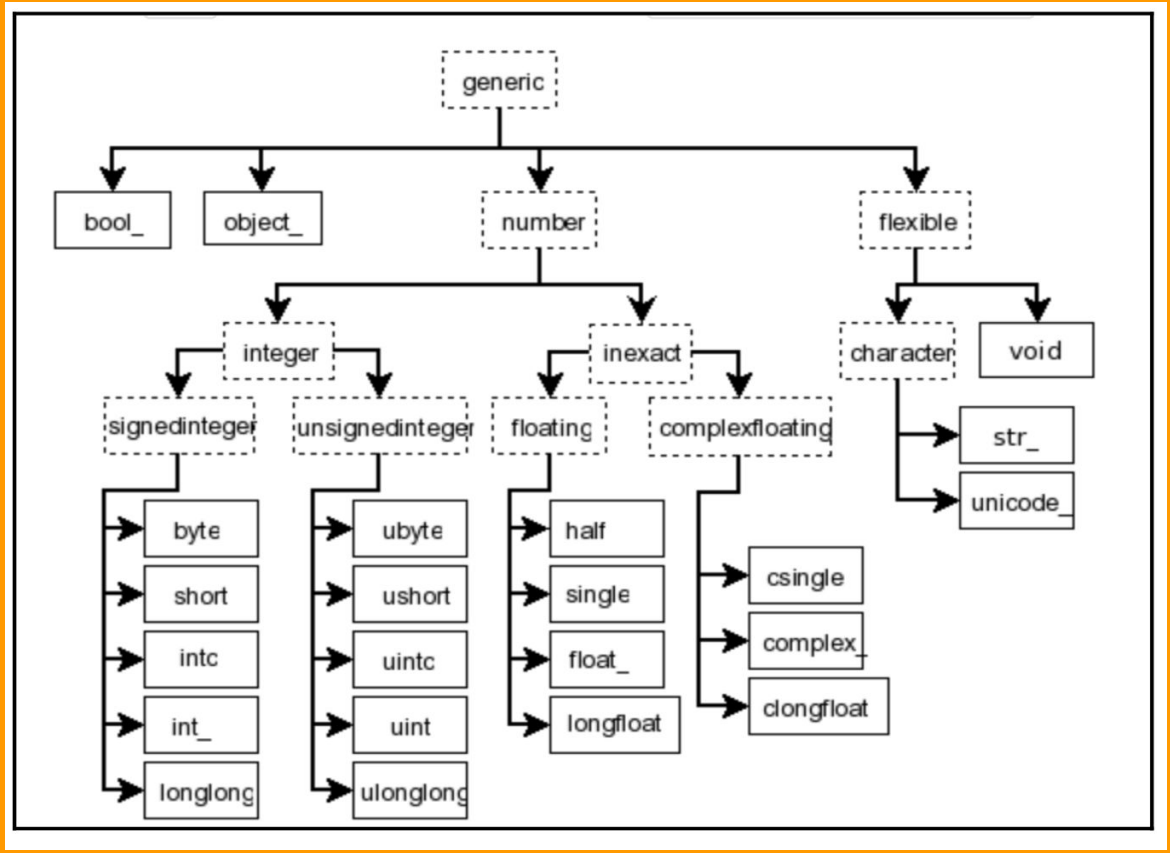
Obtener estadísticas resumidas para el objeto y las columnas categóricas

https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.Categorical.html

	count	unique	top	freq
INSTNM	7535	7535	Nunez Community College	1
CITY	7535	2514	New York	87
STABBR	7535	59	CA	773
MD_EARN_WNE_P10	6413	598	PrivacySuppressed	822
GRAD_DEBT_MDN_SUPP	7503	2038	PrivacySuppressed	1510

- Por defecto, "describe" da como resultado un resumen de todas las columnas numéricas (en su mayoría continuas) y deja caer silenciosamente cualquier columna categórica.
- Se puede utilizar np.number o el número de cadena para incluir tanto números enteros como flotantes en el resumen.
- Técnicamente, los tipos de datos forman parte de una jerarquía en la que el número reside por encima de los números enteros y los flotantes.

Jerarquía de Datos en Numpy



En términos generales, podemos clasificar los datos como continuos o categóricos.

Los datos continuos son siempre numéricos y normalmente pueden asumir un número infinito de posibilidades como la altura, el peso y el salario.

Los datos categóricos representan valores discretos que toman un número infinito de posibilidades como la etnia, el estatus de empleo y el color del coche.

Los datos categóricos pueden ser representados numéricamente o con caracteres.

```
with pd.option_context('display.max_rows', 5):
    display(college.describe(include=[np.number],
                             percentiles=[.01, .05, .10, .25, .5, .75, .9, .95, .99]).T)
```

Es posible especificar los cuantiles exactos devueltos por el método "describe" cuando se utiliza con columnas numéricas.

	count	mean	std	min	1%	5%	10%	25%	50%	75%	90%	95%	99%	max
HBCU	7164.0	0.014238	0.118478	0.0	0.0000	0.0000	0.0000	0.0000	0.00000	0.000000	0.00000	0.00000	1.000000	1.0
MENONLY	7164.0	0.009216	0.095520	0.0	0.0000	0.0000	0.0000	0.0000	0.00000	0.000000	0.00000	0.00000	0.000000	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
PCTFLOAN	6849.0	0.522211	0.283616	0.0	0.0000	0.0000	0.0000	0.3329	0.58330	0.745000	0.84752	0.89792	0.986368	1.0
UG25ABV	6718.0	0.410021	0.228939	0.0	0.0025	0.0374	0.0899	0.2415	0.40075	0.572275	0.72666	0.80000	0.917383	1.0

Diccionarios de Datos

- Una parte crucial del análisis de datos implica la creación y el mantenimiento de un diccionario de datos.
- Un diccionario de datos es una tabla de metadatos y notas en cada columna de datos.
- Uno de los principales propósitos de un diccionario de datos es explicar el significado de los nombres de las columnas.
- Los datos de **college** utilizan muchas abreviaturas que probablemente no sean familiares para un analista que los inspeccione por primera vez.

```
college_dd = pd.read_csv('data/college_data_dictionary.csv')
with pd.option_context('display.max_rows', 8):
    display(college_dd)
```

	column_name	description
0	INSTNM	Institution Name
1	CITY	City Location
2	STABBR	State Abbreviation
3	HBCU	Historically Black College or University
...	...	...
23	PCTFLOAN	Percent Students with federal loan
24	UG25ABV	Percent Students Older than 25
25	MD_EARN_WNE_P10	Median Earnings 10 years after enrollment
26	GRAD_DEBT_MDN_SUPP	Median debt of completers

27 rows × 2 columns

Es inmensamente útil para descifrar los nombres abreviados de las columnas. Los DataFrames no son en realidad el mejor lugar para almacenar diccionarios de datos. Una plataforma como Google Sheets con una fácil capacidad de editar valores y añadir columnas es una mejor opción. Mínimamente, una columna para llevar un registro de las notas sobre los datos debe ser incluida en un diccionario de datos. Un diccionario de datos es una de las primeras cosas que puede compartir como analista con los colaboradores.

Reducir la memoria cambiando los tipos de datos

Pandas no clasifica en general los datos como **continuos** o **categóricos**, pero tiene definiciones técnicas precisas para muchos tipos de datos distintos.

Después de leer el conjunto de datos, seleccionamos unas cuantas columnas de diferentes tipos de datos que mostrarán claramente cuánta memoria se puede guardar.

```
college = pd.read_csv('data/college.csv')
different_cols = ['RELAFFIL', 'SATMTMID', 'CURROPER', 'INSTNM', 'STABBR']
col2 = college.loc[:, different_cols]
col2.head()
```

	RELAFFIL	SATMTMID	CURROPER	INSTNM	STABBR
0	0	420.0	1	Alabama A & M University	AL
1	0	565.0	1	University of Alabama at Birmingham	AL
2	1	NaN	1	Amridge University	AL
3	0	590.0	1	University of Alabama in Huntsville	AL
4	0	430.0	1	Alabama State University	AL

col2.dtypes

```
RELAFFIL      int64
SATMTMID    float64
CURROPER      int64
INSTNM       object
STABBR       object
dtype: object
```

Inspeccionamos el tipo de datos

original_mem = col2.memory_usage(deep=True)
original_mem

```
Index           80
RELAFFIL     60280
SATMTMID     60280
CURROPER     60280
INSTNM      660240
STABBR      444565
dtype: int64
```

Con el método memory_usage, observamos el uso de la memoria de cada columna. (Bytes Consumidos)

```
col2['RELAFFIL'] = col2['RELAFFIL'].astype(np.int8)
col2.dtypes
```

```
RELAFFIL        int8
SATMTMID     float64
CURROPER       int64
INSTNM        object
STABBR        object
dtype: object
```

No es necesario utilizar 64 bits para la columna RELAFILL ya que sólo contiene 0/1. Convirtamos esta columna en un entero de 8 bits (1 byte) con el método "astype".

```
col2.select_dtypes(include=['object']).nunique()
```

```
INSTNM     7535
STABBR       59
dtype: int64
```

Para ahorrar aún más memoria, se considerara cambiar los tipos de datos de los objetos a categóricos si tienen una cardinalidad razonablemente baja (valores numéricos únicos). Comprobemos primero el número de valores únicos para ambas columnas de objetos.

```
col2['STABBR'] = col2['STABBR'].astype('category')
col2.dtypes
```

```
RELAFFIL         int8
SATMTMID      float64
CURROPER        int64
INSTNM         object
STABBR       category
dtype: object
```

La columna STABBR es una buena candidata para convertir a Categórica ya que menos del uno por ciento de sus valores son únicos.

```
new_mem = col2.memory_usage(deep=True)
new_mem
```

```
Index            80
RELAFFIL       7535
SATMTMID      60280
CURROPER      60280
INSTNM       660699
STABBR        13576
dtype: int64
```

Calculemos el uso de la memoria de nuevo

new_mem / original_mem

```
Index        1.000000
RELAFFIL     0.125000
SATMTMID     1.000000
CURROPER     1.000000
INSTNM       1.000695
STABBR       0.030538
dtype: float64
```

Finalmente se compara la memoria usada con la actualizada.

college['MENONLY'].dtype

college['MENONLY'].astype('int8')

No todas las columnas pueden ser coaccionadas a un tipo deseado.

Además, es posible sustituir los nombres de las cadenas en lugar de los objetos de Python al referirse a los tipos de datos. Por ejemplo, cuando se utiliza el parámetro include en el método "describe" de DataFrame, es posible pasar la lista del objeto formal Numpy/Pandas o su representación de cadena equivalente.

```
>>> college.describe(include=['int64', 'float64']).T

>>> college.describe(include=[np.int64, np.float64]).T

>>> college.describe(include=['int', 'float']).T

>>> college.describe(include=['number']).T
```

Por último, es posible ver la enorme diferencia de memoria entre el RangeIndex mínimo y el Int64Index, que almacena cada índice de fila en la memoria

```
college.index = pd.Int64Index(college.index)
college.index.memory_usage()
```

- Seleccionando el más pequeño de los más grandes

Esta receta puede ser utilizada para crear titulares de noticias interesantes, como que

- *De las 100 mejores universidades*, estas *5 tienen la matrícula más baja*
- *De las 50 mejores ciudades para vivir, estas 10 son las más asequibles.*
- *Entre otros.*

Durante un análisis, es posible que primero tenga que encontrar una agrupación de datos que contenga los valores n superiores en una sola columna y, a partir de este subconjunto, encontrar los valores m inferiores basados en una columna diferente.

```
movie = pd.read_csv('data/movie.csv')
movie2 = movie[['movie_title', 'imdb_score', 'budget']]
movie2.head()
```

	movie_title	imdb_score	budget
0	Avatar	7.9	237000000.0
1	Pirates of the Caribbean: At World's End	7.1	300000000.0
2	Spectre	6.8	245000000.0
3	The Dark Knight Rises	8.5	250000000.0
4	Star Wars: Episode VII - The Force Awakens	7.1	NaN

1.- Usamos el método “nlargest” para seleccionar las mejores 100 peliculas por imbd_score

```
movie2.nlargest(100, 'imdb_score').head()
```

2.- Se encadena el método "nsmallest" para devolver las cinco películas de menor presupuesto entre las 100 de mayor puntuación:

```
movie2.nlargest(100, 'imdb_score').nsmallest(5, 'budget')
```

- ## Seleccionando el más grande de cada grupo mediante la clasificación

Una de las operaciones más básicas y comunes que se realizan durante un análisis de datos es seleccionar las filas que contienen el mayor valor de alguna columna dentro de un grupo.

Por ejemplo, esto sería como encontrar la película de más alta calificación de cada año o la película de más alta recaudación por calificación de contenido.

Para llevar a cabo esta tarea, necesitamos clasificar los grupos así como la columna utilizada para clasificar cada miembro del grupo, y luego extraer el miembro más alto o cada grupo.

- Encontraremos la película de más alto rating de cada año.

1.- Seleccionamos para leer del DataFrame (movie.csv) las tres columnas que nos importan: **movie_title**, **title_year**, **imbd_score**

```
movie = pd.read_csv('data/movie.csv')
movie2 = movie[['movie_title', 'title_year', 'imdb_score']]
```

2.- Utiliza el método **sort_values** para ordenar el dataFrame por **title_year**. El comportamiento por defecto ordena desde el más pequeño al más grande. Use el parámetro "*ascending*" para invertir este comportamiento poniéndolo igual a "True"

```
movie2.sort_values('title_year', ascending=False).head()
```

3.- Fijate que solo se ordeno el año. Para ordenar multiples columnas al tiempo, usamos una lista.

```
movie3 = movie2.sort_values(['title_year','imdb_score'], ascending=False)
movie3.head()
```

4.- Ahora, usamos el método **drop_duplicates** para mantener sólo la primera fila de cada año.

```
movie_top_year = movie3.drop_duplicates(subset='title_year')
movie_top_year.head()
```

References

- Python Programming: An Introduction to Computer Science. John Zelle
- Big Data con Python. Rafael Caballero Enrique Martín Adrián Riesco
- Aprende Python en un Fin de Semana Alfredo Moreno Muñoz Sheila Córcoles Córcoles
- Learn Python Programming Fabrizio Romano
- Python Data Analytics Fabio Nelli
- Expert Python Programming Michael Jasworski Tarek Ziadé
- Statistical analysis of questionnaires: a unified approach based on R and Stata by Francesco Bartolucci. Boca Raton: CRC Press, 2016.
- Data visualisation: a handbook for data driven design by Andy Kirk. Los Angeles: Sage, 2016.
- Learning tableau: leverage the power of tableau 9.0 to design rich data visualizations and build fully interactive dashboards by Joshua N. Milligan. Mumbai: Packt Publishing, 2015.