



Procesamiento de Datos con Python 2020

“Reinterpretación de procesos de BBDD usando Python”

Checkpoint

Alumno: Céspedes Jaén Abdiel

Experto: Ramón Arias José

Grupo: data - analysis - gdl - 20 - 06

Fecha de entrega: Sábado 23/01/2021



ÍNDICE

Resumen	3
Objetivos	3
Objetivo general	3
Objetivos particulares	3
Desarrollo	3
Post Work 1	3
<i>Identificación del problema</i>	3
<i>Investigación</i>	3
<i>Búsqueda de soluciones anteriores</i>	4
Post Work 2	4
<i>Serie de preguntas</i>	4
Post Work 3	4
<i>Recolección de datos</i>	4
Post Work 4	5
<i>Análisis</i>	5
Post Work 5	6
<i>Limpieza</i>	6
Post Work 6	7
<i>APIs</i>	7
Post Work 7	10
<i>Procesamiento de datos</i>	10
Post Work 8	15
<i>Presentación de la información</i>	15
<i>Presentación de exposición</i>	15
<i>Planes a futuro</i>	15
Respuestas	15
Conclusión	16
Bibliografía	16
Base de Datos Original	16
Base de Datos Utilizada	16



Resumen

En el presente trabajo se desglosa el procedimiento empleado por el alumno para dar respuesta a un conjunto de interrogantes realizadas con respecto a un conjunto de datos. La principal intención es emplear las diversas herramientas que ofrece el lenguaje de programación de Python para dar una solución justificada a las interrogantes planteadas en el proceso de análisis de datos.

Objetivos

Objetivo general

“Al finalizar el módulo serás capaz de utilizar Python para adquirir datos ya sea de archivos locales, APIs o Bases de Datos con el objetivo de limpiarlos, manejarlos, explorarlos para dejarlos de manera que queden en condiciones óptimas para su análisis y visualización.”

Objetivos particulares

- Poner en práctica y en evidencia los conocimientos adquiridos a lo largo del curso.
- Dar respuesta a las interrogantes señaladas a lo largo del proyecto.
- Justificar las respuestas a cada pregunta del proyecto mediante el uso de Python.

Desarrollo

Post Work 1

Identificación del problema

Se presentan problemas para la administración y relación de información entre las diversas BBDD de una empresa brasileña. Se solicita el uso de un lenguaje de programación para su estructuración, limpieza y estructuración con el fin de dar respuesta a posibles interrogantes.

Investigación

El presente trabajo fue realizado en base a una base de datos de una empresa brasileña de ecommerce, publicada por cortesía de Olist. Sus características permiten la visualización de su información a través de múltiples dimensiones: desde el estatus de la orden, el precio, el pago y el rendimiento del flete, la locación del



cliente, atributos del producto y reseñas hechas por los propios clientes. La base de datos incluye también una tabla enfocada a la geolocalización, sustentada gracias a códigos postales.

Olist fue el proveedor de la base de datos empleada, el cual es el mayor departamento de almacenes en los mercados de Brasil. Olist conecta pequeños negocios a lo largo de Brasil a canales con un contrato sencillo. Los comerciantes pueden vender sus productos en las tiendas de Olist y enviarlas directamente a los clientes usando los socios de logística de Olist.

Búsqueda de soluciones anteriores

Ya se realizó un estudio preliminar de las BBDD de la empresa brasileña por medio de SQL y por MongoDB. Se dieron respuesta a 4 interrogantes:

- ¿Qué clientes ya han sido satisfechos al haber registrado su estatus como “entregado”?
- ¿Cuántas veces ha aparecido el método de pago de tarjeta de crédito?
- ¿Cuántos son los pagos que están por encima del promedio de su propio valor?
- ¿Calcula la frecuencia de cada tipo de pago?

Post Work 2

Serie de preguntas

- ¿Qué clientes ya han sido satisfechos al haber registrado su estatus como “entregado”?
- ¿Cuántas veces ha aparecido el método de pago de tarjeta de crédito?
- ¿Cuántos son los pagos que están por encima del promedio de su propio valor?
- ¿Calcula la frecuencia de cada tipo de pago?

Post Work 3

Recolección de datos

Los datos pertenecen a una empresa brasileña y Olist, por medio de la plataforma de Kaggle, proporcionó todas y cada una de las BBDD en formato .csv. En el apartado de “bibliografía” se da crédito a la obtención de las bases de datos, así como una aclaración con respecto a su carga.



Post Work 4

Análisis

- ¿El conjunto de datos que tengo realmente me sirve para responder algunas de las preguntas que me planteé?
Sí, pues las preguntas se realizaron en base a la información proporcionada, pues las BBDD ya existían, no al revés.
- ¿Qué tamaño tiene mi conjunto de datos? ¿Serán datos suficientes?
Sí, es demasiado grande, por lo que la muestra obtenida fue seleccionada para optimizar en su momento el proceso de carga al servidor de MySQL Workbench.
- ¿Qué columnas tengo y qué información tengo en cada una de esas columnas?
Tengo diversas BBDD y representan los siguiente:
 - Clientes.
 - Localización.
 - Productos ordenados.
 - Productos pagados.
 - Reseñas de los productos.
 - Órdenes.
 - Productos.
 - Vendedores.
- Los nombres que tienen mis columnas, ¿son el nombre más apropiado?
Sí, al menos para mí me basta para identificar la información que traen consigo.
- ¿Qué tipos de datos tengo en cada columna? ¿Parecen ser el tipo correcto de datos? ¿O es un tipo de datos "incorrecto"?
Son el tipo de datos correctos, en el apartado de limpieza se demuestra esta afirmación.
- Si selecciono algunas filas al azar y las observo, ¿estoy obteniendo los datos que debería? ¿o es que hay datos que parecen estar "sucios" o "incorrectos"?
Sí obtengo los datos que debo obtener, pues las BBDD ya de por sí están muy bien estructuradas.



Post Work 5

Limpieza

1. Explora tu dataset con el fin de encontrar los NaNs que contiene. Piensa en la distribución de NaNs por columna y por fila.

```
[28] orders.isna().sum()
```

```
order_id            0
customer_id         0
order_status        0
order_purchase_timestamp  0
order_approved_at   1
order_delivered_carrier_date  7
order_delivered_customer_date 13
order_estimated_delivery_date  0
dtype: int64
```

```
[30] products.isna().sum()
```

```
product_id            0
product_category_name  9
product_name_length   9
product_description_length  9
product_photos_qty    9
product_weight_g      0
product_length_cm     0
product_height_cm     0
product_width_cm      0
dtype: int64
```

2. Piensa cuáles son los procedimientos que puedes aplicar a tus NaNs.

- Eliminar filas con NaN.
- Eliminar columnas con NaN.
- Sustituir NaN por un valor.

3. ¿Tenemos que eliminar las filas/columnas que tienen esos NaNs? ¿O podríamos rellenar esos NaNs con algún valor de manera que podamos retener esas filas/columnas?

```
orders_dropped = orders.dropna(axis=1, how='any')
orders_dropped.head()
```

	order_id	customer_id	order_status	order_purchase_timestamp	order_estimated_delivery_date
0	01e71f17355fc56620b0d3678f796c42	004a2b3e8b026284407c620f4f01f686	delivered	18/04/2018 22:19	18/05/2018 00:00
1	0286e258217733541e80588e29c208c1	010583f85dd55d32a6db1068f58ca6f1	delivered	08/10/2017 20:37	13/11/2017 00:00
2	0256530d814cde05587fcd8e1b992bc8	023d1d56f9187c91230a4e8b50ab6fb3	delivered	28/09/2017 23:57	26/10/2017 00:00
3	01ba9a2e5dcc1568b55695106d677b0c	0241bf6728f2c70f948e9a51484d1435	delivered	26/02/2017 22:28	20/03/2017 00:00
4	0231382a78f19752dabca6a86b618152	02cb822eec8b6309e7ce715ccea3390d	delivered	09/03/2017 21:17	28/03/2017 00:00

```
products_dropped = products.dropna(axis=1, how='any')
products_dropped.head()
```

	product_id	product_weight_g	product_length_cm	product_height_cm	product_width_cm
0	00ab8a8b9fe219511dc3f178c6d79698	2100	50	30	30
1	00ae7076313576f94d9107599d79a978	600	22	33	16
2	00b264091d1c8df03976c3f3b176b35c	250	16	16	16
3	00b4155166f994ba9da3cf001eb80505	1125	28	15	23
4	00b782074a5ec523bf31951b4935d572	250	16	5	11

4. Limpia tu dataset de manera que no quede ningún NaN.

```
[32] sellers.isna().sum()
```

```
seller_id          0  
seller_zip_code_prefix  0  
seller_city        0  
seller_state       0  
dtype: int64
```

5. Reindexa tu dataset si lo consideras necesario.

No es necesario. Para evitar confusión en un futuro, conservaré los índices que ya tienen por default, pues todos esos data frames fueron pensados para ser relacionados y, por lo tanto, tienen asignadas llaves primarias/foráneas.

6. Renombra tus columnas si lo consideras necesario.

No es necesario.

Post Work 6

APIs

1. Encuentra un API que quieras explorar. Puedes encontrar una lista enorme de APIs gratuitas aquí.

API usada: <https://api.nasa.gov/>

2. Crea una cuenta si es necesario.

Cuenta creada en clases pasadas.

3. Lee la documentación.

Documentación leída.

4. Realiza algunas peticiones de prueba para entender la estructura de los datos (si quieres explorar un poco, puedes intentar hacer peticiones usando este software).



```
[144] api_key = '23vKfPQPmWrjq0qrg2pq19cHO6C9QX0WERQH67R3'
      url = 'https://api.nasa.gov/neo/rest/v1/neo/browse/'
      payload = {
          'api_key': api_key
      }

[147] r = requests.get(url, params=payload)

[148] r.status_code

200

[150] json = r.json()

[151] json.keys()

dict_keys(['links', 'page', 'near_earth_objects'])

[152] json['links']

{'next': 'http://www.neowsapp.com/rest/v1/neo/browse?page=1&size=20&api_key=23vKfPQPmWrjq0qrg2pq19cHO6C9QX0WERQH67R3',
'self': 'http://www.neowsapp.com/rest/v1/neo/browse?page=0&size=20&api_key=23vKfPQPmWrjq0qrg2pq19cHO6C9QX0WERQH67R3'}

[153] json['page']

{'number': 0, 'size': 20, 'total_elements': 24927, 'total_pages': 1247}

[159] json['near_earth_objects']

[162] data = pd.json_normalize(json['near_earth_objects'])
      data.head()
```

5. Automatiza el proceso de realizar peticiones para obtener un dataset considerablemente grande.

```
[168] dict_datos = {}

      for i in range (0, 5):

          try:

              r = requests.get(url, params=payload)

              if r.status_code == 200:

                  json = r.json()

                  data = json['near_earth_objects']
                  dict_datos[i] = data

                  new_link = json['links']['next']

                  url = new_link

          except:

              continue

      #time.sleep(5)
```




6. Explora y limpia tu dataset.

```
[169] dict_datos
```

```
[176] api_df = pd.DataFrame.from_dict(data)
      api_df.head()
```

absolute_magnitude_h	estimated_diameter	is_potentially_hazardous_asteroid	close_approach_data	orbital_data	is_sentry_object	name_limited
18.8	{'kilometers': '(estimated_diameter_min': 0.46...	False	{'close_approach_date': '1904-10-12', 'close_...	{'orbit_id': '72', 'orbit_determination_date':...	False	NaN
16.0	{'kilometers': '(estimated_diameter_min': 1.67...	False	{'close_approach_date': '1901-06-29', 'close_...	{'orbit_id': '145', 'orbit_determination_date':...	False	NaN
18.1	{'kilometers': '(estimated_diameter_min': 0.63...	False	{'close_approach_date': '1900-08-26', 'close_...	{'orbit_id': '94', 'orbit_determination_date':...	False	NaN
19.5	{'kilometers': '(estimated_diameter_min': 0.33...	True	{'close_approach_date': '1900-11-17', 'close_...	{'orbit_id': '125', 'orbit_determination_date':...	False	NaN
19.0	{'kilometers': '(estimated_diameter_min': 0.42...	True	{'close_approach_date': '1904-05-01', 'close_...	{'orbit_id': '162', 'orbit_determination_date':...	False	NaN

```
[177] api_df.dtypes
```

```
links          object
id             object
neo_reference_id object
name           object
designation     object
nasa_jpl_url   object
absolute_magnitude_h float64
estimated_diameter object
is_potentially_hazardous_asteroid bool
close_approach_data object
orbital_data   object
is_sentry_object bool
name_limited   object
dtype: object
```

```
[178] api_df.isna().sum()
```

```
links          0
id             0
neo_reference_id 0
name           0
designation     0
nasa_jpl_url   0
absolute_magnitude_h 0
estimated_diameter 0
is_potentially_hazardous_asteroid 0
close_approach_data 0
orbital_data   0
is_sentry_object 0
name_limited   18
dtype: int64
```

```
[180] api_df = api_df.dropna(axis=1, how='any')
      api_df.isna().sum()
```

```
links          0
id             0
neo_reference_id 0
name           0
designation     0
nasa_jpl_url   0
absolute_magnitude_h 0
estimated_diameter 0
is_potentially_hazardous_asteroid 0
close_approach_data 0
orbital_data   0
is_sentry_object 0
dtype: int64
```

Post Work 7

Procesamiento de datos

1. Checa que todos tus datos tengan el tipo de dato correcto. Si no es así, usa casting para convertir tus datos al tipo de dato correcto (recuerda que tipos de datos como datetime 64 se guardan como strings cuando están en archivos .csv, así que tendrás que convertirlos al tipo de dato apropiado cada vez que importemos tu archivo).

```
[43] items.dtypes
```

```
id          int64
order_id    object
order_item_id  int64
product_id  object
seller_id   object
shipping_limit_date  object
price       float64
freight_value  float64
dtype: object
```

```
[51] items['shipping_limit_date'] = pd.to_datetime(items['shipping_limit_date'], unit = 'ns')
      items.dtypes
```

```
id          int64
order_id    object
order_item_id  int64
product_id  object
seller_id   object
shipping_limit_date  datetime64[ns]
price       float64
freight_value  float64
dtype: object
```



```
[45] orders_dropped.dtypes
```

```
order_id          object
customer_id       object
order_status      object
order_purchase_timestamp  object
order_estimated_delivery_date  object
dtype: object
```

```
[52] orders_dropped['order_estimated_delivery_date'] = pd.to_datetime(orders_dropped['order_estimated_delivery_date'], unit = 'ns')
orders_dropped.dtypes
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
"""Entry point for launching an IPython kernel.
```

```
order_id          object
customer_id       object
order_status      object
order_purchase_timestamp  object
order_estimated_delivery_date  datetime64[ns]
dtype: object
```

2. Si tienes columnas de texto, asegúrate de que todas tengan el formato correcto. Si no es así, utiliza las técnicas de manipulación de strings para darles el formato que necesitas.

Cuentan con el formato correcto.

3. Si consideras que alguna de tus columnas sería más clara si los datos tuvieran otro formato o representación usa map para transformar los datos de esa columna.

Tienen el formato correcto.

4. Si crees que es posible generar nuevas columnas útiles a partir de las columnas que ya tienes, usa apply para generar nuevos datos a partir de los que tienes y añadelos a tu dataset.

Para las preguntas no es necesario, pues en el caso de la tabla de pagos, esta abarca todo lo necesario para las futuras *queries*.

5. Con el fin de responder algunas de las preguntas que te planteaste acerca de tu dataset, usa filtros y sorting para crear nuevos subconjuntos y reordenamientos que sean más adecuados para responder tus preguntas. Primero comienza intentando responder las preguntas que te planteaste al principio, pero después puedes solamente explorar para ver si encuentras otras preguntas que no te habías planteado anteriormente.

- **¿Qué clientes ya han sido satisfechos al haber registrado su estatus como “entregado”?**



```
[71] join_customers_orders_dropped = pd.concat([customers, orders_dropped], axis=1)
join_customers_orders_dropped
```

	customer_id	customer_unique_id	customer_zip_code_prefix	customer_city	customer_state	ord
0	004a2b3e8b026284407c620f4f01f686	e90a1b194724309bbaa6228c398d1748	78795	pedra preta	MT	01e71f17355fc56620b0d3678f7
1	004d41e9bf012c669db1a257888d85a2	5e35d686648f9f99fd1f90161d3db93c	66087	belem	PA	0286e258217733541e80588e29c
2	004df18653e9438571e9d294776a5c5c	33e824e5fde35ad626278fba806c772	4623	sao paulo	SP	0256530d814cde05587fcd8e1b9
3	004ee20df425002ed78553e1f50caa3e	b6f424d6007b27912d37816a7d74ffaf	19560	indiana	SP	01ba9a2e5dcc1568b55695106d6
4	004f39eb181d53fa796fe82ce7a0cbd5	aa5824451a9437648ed4dcdac530566d	8775	mogi das cruzeiras	SP	0231382a78f19752dabca6a86b6
...
995	02cd8ee24434a2bb2574b61ba5f659a8	afb296a1b427a13ad961689550a56fa	42806	camacari	BA	
996	02d12adf94e256f7ff0056d447557027	2396ed0fb49c195da124f55b4c17ee35	4551	sao paulo	SP	
997	02d16f10703b6d4ef180f921028032a5	7f6a667d1d03cffe2d0f064b9b4d824d	5054	sao paulo	SP	
998	02d1b5b8831241174c6ef13efd35abbd	7c08375cd96553acf4ed275609fd22e7	45000	vitoria da conquista	BA	
999	02d22b2dd08fd5b4bb3ebac28f73d68c	91238ac9b555af75bab81ecb4a24ccca	6950	juquitiba	SP	

1000 rows x 10 columns

```
a = join_customers_orders_dropped['order_status'] == 'delivered'
a
```

```
0      True
1      True
2      True
3      True
4      True
...
995    False
996    False
997    False
998    False
999    False
Name: order_status, Length: 1000, dtype: bool
```

```
[90] a.sum()
```

487

- ¿Cuántas veces ha aparecido el método de pago de tarjeta de crédito?

```
[107] join_customers_payments = pd.concat([customers, payments], axis = 1)
join_customers_payments
```

customer_zip_code_prefix	customer_city	customer_state	payment_id	order_id	payment_sequential	payment_type	payment_installments
78795	pedra preta	MT	85284.0	00010242fe8c5a6d1ba2dd792cb16214	1.0	credit_card	2.0
66087	belem	PA	2500.0	00018f77f2f0320c557190d7a144bdd3	1.0	credit_card	3.0
4623	sao paulo	SP	12394.0	000229ec398224ef6ca0657da4fc703e	1.0	credit_card	5.0
19560	indiana	SP	32972.0	00024acbcdff0a6daa1e931b038114c75	1.0	credit_card	2.0
8775	mogi das cruzeiras	SP	98712.0	00042b26cf59d7ce69dfabb4e55b4fd9	1.0	credit_card	3.0
...
42806	camacari	BA	NaN	NaN	NaN	NaN	NaN
4551	sao paulo	SP	NaN	NaN	NaN	NaN	NaN
5054	sao paulo	SP	NaN	NaN	NaN	NaN	NaN
45000	vitoria da conquista	BA	NaN	NaN	NaN	NaN	NaN
6950	juquitiba	SP	NaN	NaN	NaN	NaN	NaN

```
[108] b = join_customers_payments['payment_type'] == 'credit_card'
      b
```

```
0      True
1      True
2      True
3      True
4      True
...
995    False
996    False
997    False
998    False
999    False
Name: payment_type, Length: 1000, dtype: bool
```

```
[109] b.sum()
```

```
366
```

- ¿Cuántos son los pagos que están por encima del promedio de su propio valor?

```
[111] media = np.mean(payments['payment_value'])
      media
```

```
141.84510000000003
```

```
[112] c = payments['payment_value'] > media
      c
```

```
0      False
1       True
2       True
3      False
4       True
...
495    False
496    False
497    False
498    False
499    False
Name: payment_value, Length: 500, dtype: bool
```

```
[113] c.sum()
```

```
166
```



- ¿Calcula la frecuencia de cada tipo de pago?

```
[139] payments_new = pd.read_csv('4_olist_order_payments_dataset.csv', index_col=0)
      payments_new.head()
```

	order_id	payment_sequential	payment_type	payment_installments	payment_value
payment_id					
85284	00010242fe8c5a6d1ba2dd792cb16214	1	credit_card	2	72.19
2500	00018f77f2f0320c557190d7a144bdd3	1	credit_card	3	259.83
12394	000229ec398224ef6ca0657da4fc703e	1	credit_card	5	216.87
32972	00024acbcd0a6daa1e931b038114c75	1	credit_card	2	25.78
98712	00042b26cf59d7ce69dfabb4e55b4fd9	1	credit_card	3	218.04

```
[141] c = payments_new.groupby(['payment_type']).count()
      c
```

	order_id	payment_sequential	payment_installments	payment_value
payment_type				
boleto	97	97	97	97
credit_card	366	366	366	366
debit_card	8	8	8	8
not_defined	1	1	1	1
voucher	28	28	28	28

```
▶ c.drop(columns=['payment_sequential', 'payment_installments', 'payment_value'])
```

	order_id
payment_type	
boleto	97
credit_card	366
debit_card	8
not_defined	1
voucher	28



Post Work 8

Presentación de la información

https://colab.research.google.com/drive/18MHze_KPV_qq0hccfRKF11Dfxn5Ob3EN?usp=sharing

Presentación de exposición

<https://docs.google.com/presentation/d/1vAx4euMk4JMdR6Z6k2uUsLaYp5zKaE4ViyAJbiYOPtw/edit?usp=sharing>

Planes a futuro

De ahora en adelante pienso buscar nuevos datos y de otra índole para replicar estos conocimientos, pero que, además, se presten a cumplir la tercera fase de la Ciencia de Datos: la predicción. Por lo cual, creo que el análisis de un fenómeno con suficiente evidencia recolectada por medio de datos me servirá para lograr este objetivo.

Respuestas

- ¿Qué clientes ya han sido satisfechos al haber registrado su estatus como “entregado”?
Un total de 487 de 500.
- ¿Cuántas veces ha aparecido el método de pago de tarjeta de crédito?
Ha aparecido 366 veces.
- ¿Cuántos son los pagos que están por encima del promedio de su propio valor?
De los 500 registros, se obtiene que 166 está por encima del promedio.
- ¿Calcula la frecuencia de cada tipo de pago?
Las frecuencias son:
 - Boleto = 97.
 - Tarjeta de crédito = 366.
 - Tarjeta de débito = 8.
 - No definido = 1.
 - Voucher = 28.



Conclusión

Gracias al lenguaje de programación de Python y, en particular, a su paquetería “Pandas” orientada al análisis de BBDD, es que se pudo dar tratamiento y presentación al conjunto de datos que a su vez ya había sido gestionados por primera vez con un SGBD como MySQL y MongoDB. Así pues, puedo asegurar que Python me ha brindado las herramientas necesarias para recolectar datos precisos y para gestionar un buen análisis con el propósito de dar solución a una problemática.

Bibliografía

Base de Datos Original

- kaggle. (s.f.). *Brazilian E-Commerce Public Dataset by Olist*. Recuperado el 27 de Octubre de 2020, de <https://www.kaggle.com/olistbr/brazilian-ecommerce>

Base de Datos Utilizada

La base de datos utilizada para la realización de este proyecto se redujo de 1000 registros/documentos para la primera tabla y a 500 para las demás porque en su momento este hecho facilitó su carga en el servidor de MySQL Workbench.