Sobre el Merge y los Tests

Hay cuatro categorías:

- corrección,
- sugerencias,
- autoenredos,
- pedantería.

Correcciones

- En los *tests* `app_agent`, `app_resourcer` ya son instancias del `epic_py`.
  Si las pruebas son de bajo nivel, no se pueden usar.
  Si usamos `epic_py`, entonces cambiamos el enfoque de las pruebas para validarlo.

- `get_storage_client` termina con `print(...)` entonces no devuelve al cliente.
  Lo mejor es incluir `return b_service`.

- `isinstance(key_vault, KeyVault)` no funciona porque las *keyvaults* pueden generarse sin estar validadas. Se necesita hacer una petición del *keyvault*.
  ... ya vi que sigue en investigación.

- En particular los nombres de las funciones de *tests* tienen que ser muy específicas:

  - Nombre completo de `tbl` y que concuerde el plural (`tables`)
  - En `feather(s)` que concuerden los plurales
  - `if condicion: assert condicion` es redundante
  - Especificar las *especificaciones* (valga la redundancia) en `test_blob_(content|exist)`, por ejemplo `test_specs_content`.
    Además validar específicamente para `accounts`, `customers`, y tal vez `payments`.
  - También `test_feather_cols` se puede bajar a los casos expecíficos de `accounts` y `customers`.
    Con el seguimiento de que si falla te redirige al Excel correspondiente.

Sugerencias

- Las transformaciones `customers_i`, `customers_1`, `...` se deben empaquetar en funciones. Es para identificar lo que constituye el pre-procesamiento, post-procesamiento, etc.
  (La filosofía es que las tablas son los actores de la obra, y de alguna manera las columnas son accesorios que se manipulan aparte)

- `ENV+dbks_tables['client']` se define en `src.__init__`
  (El motivo es que los *notebooks* no queremos que dependan de `ENV`)

- En los resúmenes de las tablas:
  Lo principal es la comparativa entre los números de clientes y de cuentas.
  La longitud de las tablas es secundaria para cachar errores.
  Las gráficas son un *overkill*.

Necesitaremos investigar cómo condensar los resultados.
Tal vez por correo, tal vez en un canal de Slack, o algo del estilo que sea más accesible.

- Para imprimir enunciados, incluye un parámetro `verbose=False`:

```
def func(..., verbose=False):
  ...
    if verbose: print(variable)
  ...
```

- En `tests.get_usr` la esencia es el *token*, entonces de una vez validar ese.

- `pytest.fail(...); pass` es redundante.
El `pass` es un *placeholder* para las instrucciones que requieren código pero todavía no se escribe. Por ejemplo:

```
def func_por_desarrollar():
  pass
```

- Igual que en la carpeta de `notebooks`, `ENV` se debe quedar afuera de `tests`.

- `Test = Test()` se ve muy peligroso pues renombra la variable original generadora; mejor `este_test = Test()` donde la instanciación de la clase se entiende por el uso de mayúsculas.

Autoenredos

Tal vez lo había pensado de una forma, y después no me convenció.
O tal vez ya lo corregimos cuando hicimos *push* ayer.
En todo caso, perdón por la cconfusión.

- `dependencies.py -> dbks_dependencies.py`
(habíamos dicho que *scripts* en el directorio base no utilizan librerías, pero `dependencies.py` sí utiliza las de Databricks)

- Cambiar `token_userfile` de formato `yaml` -> `json` (`yaml` no es el mejor formato de configuraciones pequeñas, además `pyyaml` no es de las librerías estándar)
Implica hacer ajuste en `dependencies.py`

- Se queda `specs_local = true` en lo que se resuelve lo de los `blobs` en `prd`.
(ya me había emocionado de acceder a los *blobs*, pero falta llevarlos a `prd`)

Pedantería

Como te decía estos comentarios son una especie de "gusto culposo", pues me generan vergüencilla y a la vez apreciación por el ordenamiento 😳
En todo caso, yo te las digo y tú decides.

- Orden de librerías: `matplotlib` antes que `epic_py`

- `from module improt (obj1, obj2, ...)` Prefiero importar en horizontal.
- `get_blob(...) -> Dataframe` -> `get_blob_df`
  (confunde el nombre de la función, pues uno espera un *blob* y recibe un *dataframe*)
- Hay un estándar de los espacios en los parámetros 😬
  `funcion(arg1, arg2:str, arg3=None, arg4:str='')` o
  `funcion(arg1, arg2: str, arg3=None, arg4: str='')`
- A la par de los *tests* puedes ir investigando `pylint`, y también algunas guías de estilo de Python -por ejemplo esta.