Processed Transformation Token

Uses EtherScan reference data for Token data enrichment

Input:
- Data Content: EtherScan Reference Data
- Data Type: Parquet
- Data Source: Preprocessed Layer

Output:
- Data Content: EtherScan Enriched Token Data
- Data Type: Parquet
- Data Destination: Processed Layer

** Notebook is simply for reference and only takes in a sample collection

```
from pyspark.sql.types import  StringType, MapType
from pyspark.sql.functions import udf
from Azure_configs import preprocessed_data_path, processed_data_path
from API_configs import etherscan_url, token_api_key
import datetime
import time
import requests
import sys
```

```
today=datetime.date.today().strftime('%m-%d-%y')
token_addresses={
    'Wrapped_eth':'0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2',
    'Tether':'0xdac17f958d2ee523a2206206994597c13d831ec7',
    'Usdc':'0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48'}
```

```
def get_token_balance(eth_address):
    new_dict={}
    for i in token_addresses.keys():
        param={'token_balance':{'module':'account',
                                'action':'tokenbalance',\
                                'contractaddress':token_addresses[i],\
                                'address':eth_address,\
                                'tag':'latest','apikey':token_api_key}}
        limit_exceeded=True
        while limit_exceeded==True:
            response=requests.get(etherscan_url,params=param['token_balance'])
            message=response.json()
            if message['result']=='Max rate limit reached':
                time.sleep(0.5)
                continue
            else:
                limit_exceeded=False
                new_dict[i]=message['result']

    return new_dict
```

```
def process_token_data(nft_name):

    EScan_parquet_path=f'{preprocessed_data_path}/{today}/EScan/NFT={nft_name}/'

    EScan_reference_DF=spark.read.parquet(EScan_parquet_path)

    token_udf=udf(lambda x : get_token_balance(x),MapType(StringType(),StringType()))

    token_balance_df=EScan_reference_DF.withColumn('Token_Balance',token_udf(EScan_reference_DF['owner_address']))
    token_balance_df.cache()
    token_balance_df.show(10,truncate=False)
    token_balance_df.write.mode('overwrite').parquet(f'{processed_data_path}{today}/Token_Balance/NFT={nft_name}/')

    return
```

```
process_token_data('CryptoPunks')
```

```
+-------------------------------------------+-----------------------------------------------------+
|owner_address                              |Token_Balance                                        |
+-------------------------------------------+-----------------------------------------------------+
|0x040da2c464933005b6d1ffeccec7fb4025dc9ddb|{Usdc -> 0, Tether -> 0, Wrapped_eth -> 0}           |
|0x04ae2f0bda04f1405991d91c2e8420d6148369ea|{Usdc -> 0, Tether -> 0, Wrapped_eth -> 0}           |
|0x06e63138f3241a420829bc125e6cb6bebf88c2c2|{Usdc -> 0, Tether -> 0, Wrapped_eth -> 0}           |
|0x0a8f4037729accb854da856431a87b8a264d8c40|{Usdc -> 0, Tether -> 0, Wrapped_eth -> 0}           |
|0x1251122f1d77fa46e1e576c4fd6dd56ab17812ff|{Usdc -> 0, Tether -> 0, Wrapped_eth -> 0}           |
|0x1db12c2a7c803567c9fd59a202504edf0e56ca78|{Usdc -> 0, Tether -> 0, Wrapped_eth -> 0}           |
|0x22eab1c78521596f9d6d73dd878778009c39317c|{Usdc -> 0, Tether -> 208952523762, Wrapped_eth -> 0}|
|0x266892ed0d40ea5c37f3e0239537999c13468311|{Usdc -> 0, Tether -> 0, Wrapped_eth -> 0}           |
|0x2e675eeae4747c248bfddbafaa3a8a2fdddaa44b|{Usdc -> 0, Tether -> 0, Wrapped_eth -> 0}           |
|0x34669322bdfca9e801ca334e7b0e6d69d1f87137|{Usdc -> 0, Tether -> 1542262769000, Wrapped_eth -> 0}|
+-------------------------------------------+-----------------------------------------------------+
only showing top 10 rows
```