location

January 13, 2024

1 Leveraging Postal Codes for Comprehensive Location Intelligence:

In this project, ZA postal codes are utilized as a vital resource to extract comprehensive location details, covering coordinates, specific area address, city, suburbs, municipality, and other pertinent information. The article serves as a guiding beacon in the realm of geographic data exploration, offering a systematic walkthrough from the initial stages of data import and cleansing to the intricate process of querying location details based on postal codes.

Importing Packages: The initial step involves importing the necessary Python packages, including geopy, pgeocode, pandas, and numpy. The code snippet showcases the importation process and an example of loading postal code data from a CSV file.

Data Cleansing: The article then delves into the crucial task of data cleansing. The code segment demonstrates how to handle missing values, check data types, and convert the data type for postal codes. A dedicated function is created to ensure uniformity in the format of postal codes.

Querying the Location: The heart of the article lies in querying location details based on postal codes. The code includes initializing geocoders, creating a function to obtain suburb, city, municipality, and province from coordinates, and applying these functions to the dataset. The resulting DataFrame offers a comprehensive snapshot of location information.

Save the Results: The final step involves saving the enriched dataset as an Excel file. The code snippet illustrates how to store the obtained location details along with the original data, ensuring a tangible output for further analysis.

This article provides a holistic guide for leveraging postal codes to unlock intricate location insights, empowering data analysts and enthusiasts alike to enhance their understanding of geographical data.

GitHup Repo. Link: https://github.com/Sbugzoh2/Location_Search.git

---

By: Mr S. Gumede (MSc. Statistics - UKZN)

1.0.1 Importing Packages:

[19]:
```
from geopy.geocoders import Nominatim
import pgeocode
import pandas as pd
import numpy  as np
```

1.0.2 Loading Data:

[20]:
```
data = pd.read_csv('pcodes.csv')

#Display first 5 rows
data.head()
```

[20]:

	POSTAL CODE
0	2
1	3
2	4
3	7300
4	2192

1.0.3 Data Cleansing:

Check for missing values:

[21]:
```
data['POSTAL CODE'].isna().sum()
```

[21]:
```
0
```

Check data type:

[22]:
```
data.dtypes
```

[22]:
```
POSTAL CODE    int64
dtype: object
```

Since the data type for postal code is an integer we need to convert this to a string or object:

[23]:
```
data['POSTAL CODE'] = data['POSTAL CODE'].astype(str)
```

Confirm that the data type has converted:

[24]:
```
data.dtypes
```

[24]:
```
POSTAL CODE    object
dtype: object
```

Create a function that will ensure that the postal code consist of 4 characters:

[25]:
```
# Function:
def convert_pcode(value):
    if len(value) == 1:
        return "000" + value
    elif len(value) == 2:
        return "00" + value
    elif len(value) == 3:
        return "0" + value
    else:
        return value # actual postal code will be returned as is only if its␣
 ↪len is equals to 4 characters

# Apply the above function to the "POSTAL CODE" column:
data["POSTAL CODE"] = data["POSTAL CODE"].apply(convert_pcode)

# Display the modified DataFrame
data.head()
```

[25]:

	POSTAL CODE
0	0002
1	0003
2	0004
3	7300
4	2192

1.0.4 Querying the Location:

[26]:
```
# The 'ZA' is the Country Code for South Africa and can be replaced by any␣
 ↪appropriate country code like US for United State:
country_code = 'ZA'

# Initialize the Nominatim instance for the specified country
nomi = pgeocode.Nominatim(country_code)

# Initialize the Nominatim geocoder
geolocator = Nominatim(user_agent="MyGeocodingApp/1.0")

# Convert postal codes column into a list:
postal_code_list = list(data['POSTAL CODE'])

# Create a new DataFrame with Postal Codes Coordinates
location = pd.DataFrame(nomi.query_postal_code(postal_code_list))

#Create a copy of the above DataFrame with only the relavant columns:
location_df = location[['postal_code', 'country_code', 'place_name',␣
 ↪'latitude', 'longitude']].copy()
```

```
# Function to get suburb, city, municipality and province based on coordinates
def get_location(row):
    try:
        if pd.notna(row['latitude']) and pd.notna(row['longitude']):
            location = geolocator.reverse((row['latitude'], row['longitude']),␣
 ↪exactly_one=True)
            address = location.raw['address']
            suburb = address.get('suburb', '')
            city = address.get('city', '')
            province = address.get('state', '')
            municipality = address.get('county', '')
            return pd.Series({'Suburb': suburb, 'City': city, 'Province':␣
 ↪province, 'Municipality': municipality})
        else:
            return pd.Series({'Suburb': np.nan, 'City': np.nan, 'Province': np.
 ↪nan, 'Municipality': np.nan})
    except Exception as e:
        print(f"Error: {e}")
        return pd.Series({'Suburb': np.nan, 'City': np.nan, 'Province': np.nan,␣
 ↪'Municipality': np.nan})

# Apply the function to each row of the DataFrame
result_df = location_df.apply(get_location, axis=1)

# Concatenate the original DataFrame with the results
final_df = pd.concat([location_df, result_df], axis=1)

# Display the final DataFrame
final_df.head()
```

[26]:

	postal_code	country_code	place_name	latitude	longitude	Suburb	City	Province	Municipality
0	0002	ZA	Pretoria	-25.70690	28.2294	Tshwane Ward 54	Pretoria	Gauteng	City of Tshwane Metropolitan Municipality
1	0003	ZA	Pretoria	-25.70690	28.2294	Tshwane Ward 54	Pretoria	Gauteng	City of Tshwane Metropolitan Municipality
2	0004	ZA	Pretoria	-25.70690	28.2294	Tshwane Ward 54	Pretoria	Gauteng	
3	7300	ZA	Malmesbury	-33.45000	18.7333	Swartland Ward 10	Swartland Local Municipality	Western Cape	
4	2192	ZA	Orange Grove, Johannesburg	-26.18335	28.0833	Bezuidenhoutsvallei	Johannesburg	Gauteng	

```
2       City of Tshwane Metropolitan Municipality
3                West Coast District Municipality
4  City of Johannesburg Metropolitan Municipality
```

1.0.5 Save the results as excel file:

[27]:
```
final_df.to_csv('results.xls', index=False)
```

[ ]:
```

```