cs520_museum Projects Branches Notebook Dataset Caveats Settings

On branch **Untitled Branch**

[1] `LOAD DATASET metobjects AS csv FROM met-objects-subset (2).csv @ artifact file 7`

Console Timing Datasets Charts

metobjects (**50203** rows)

Views

	Object_Number (string)	Object_ID (int)	Department (string)	AccessionYear (date)	Object_Name (string)	Title (strir
0	2017.95	739852	Islamic Art	2017-00-01	Illuminated Manuscript	Bound Manuscript with Prayers in Praise of Imam 'Ali
1	63.664.5	375137	Drawings and Prints	1963-00-01	Print	Forest Park
2	10.130.1609	570794	Egyptian Art	1910-00-01	Cylinder seal	Cylinder seal
3	AY834 .A46 1813	679638	The Libraries	2009-00-01		Almanach impérial, pour l'année M. DCCC. XIII présente a
4	95.27.2	436631	European Paintings	1895-00-01	Painting	Self-Portrait
5	63.350.201.23.39	409698	Drawings and Prints	1963-00-01	Print	Rose-breasted Wood Robin, from the Song Birds of the Wo
6	33.43.237	267646	Photographs	1933-00-01	Photograph	Children - Nude
7	1982.1064.2	380581	Drawings and Prints	1982-00-01	Architectural Model	House X, model
8	42.115.3	69458	Asian Art	1942-00-01	Fragment	
9	1970.255.4	108776	Costume Institute	1970-00-01	Chemise	Chemise
10	66.543.19	392586	Drawings and Prints	1966-00-01	Print	De Arte Pictoria
11	1970.565.601	857142	Drawings and Prints		Book	Blue Beard : Pantomime Toy Books
12	63.350.215.172.1697	404343	Drawings and Prints	1963-00-01	Baseball card, photograph	Goodfellow, Catcher, Detroit Wolverines, from the Old Judg
13	34.100.178	10030	The American Wing	1934-00-01	Ornament	Ornament
14	1981.1055.49	719716	Drawings and Prints	1981-00-01	Drawing Ornament & Architecture	Textile Design with White Cups, Plates and Red Roses
15	63.350.204.89.52	425791	Drawings and Prints	1963-00-01	Print	Actress posing with hands beneath chin (Die-Cut), from the
16	28.31.15	349863	Drawings and Prints	1928-00-01	Print	Apples
17	53.601.16(10)	728464	Drawings and Prints	1953-00-01		God Sending the Archangel Gabriel to the Virgin
18	2009.300.2181	156978	Costume Institute	2009-00-01	Toque	Toque
19	2021.386.47	856086	Photographs	2021-00-01	Photograph	Untitled

[2]

```
SELECT * FROM metobjects WHERE Object_ID is not null;
```

Console Timing Datasets Charts

metobjects (**50000** rows)

Views

	Object_Number (string)	Object_ID (int)	Department (string)	AccessionYear (date)	Object_Name (string)	Title (strir
0	2017.95	739852	Islamic Art	2017-00-01	Illuminated Manuscript	Bound Manuscript with Prayers in Praise of Imam 'Ali
1	63.664.5	375137	Drawings and Prints	1963-00-01	Print	Forest Park
2	10.130.1609	570794	Egyptian Art	1910-00-01	Cylinder seal	Cylinder seal
3	AY834 .A46 1813	679638	The Libraries	2009-00-01		Almanach impérial, pour l'année M. DCCC. XIII présente a
4	95.27.2	436631	European Paintings	1895-00-01	Painting	Self-Portrait
5	63.350.201.23.39	409698	Drawings and Prints	1963-00-01	Print	Rose-breasted Wood Robin, from the Song Birds of the Wo
6	33.43.237	267646	Photographs	1933-00-01	Photograph	Children - Nude
7	1982.1064.2	380581	Drawings and Prints	1982-00-01	Architectural Model	House X, model
8	42.115.3	69458	Asian Art	1942-00-01	Fragment	
9	1970.255.4	108776	Costume Institute	1970-00-01	Chemise	Chemise
10	66.543.19	392586	Drawings and Prints	1966-00-01	Print	De Arte Pictoria
11	1970.565.601	857142	Drawings and Prints		Book	Blue Beard : Pantomime Toy Books
12	63.350.215.172.1697	404343	Drawings and Prints	1963-00-01	Baseball card, photograph	Goodfellow, Catcher, Detroit Wolverines, from the Old Judg
13	34.100.178	10030	The American Wing	1934-00-01	Ornament	Ornament

cs520_museum Projects Branches Notebook Dataset Caveats Settings

15	63.350.204.89.52	425791	Drawings and Prints	1963-00-01	Print	Actress posing with hands beneath chin (Die-Cut), from the
16	28.31.15	349863	Drawings and Prints	1928-00-01	Print	Apples
17	53.601.16(10)	728464	Drawings and Prints	1953-00-01		God Sending the Archangel Gabriel to the Virgin
18	2009.300.2181	156978	Costume Institute	2009-00-01	Toque	Toque
19	2021.386.47	856086	Photographs	2021-00-01	Photograph	Untitled

```
[3] LOAD DATASET cmoa AS csv FROM cmoa-artists.csv @ artifact file 8
```

Console Timing Datasets Charts

cmoa (**5962** rows)

Views

	artist_id (string)	party_type (string)	full_name (string)	cited_name (string)	nationality (string)	birth
0	cmoa:parties/2cb30976-c292-4060-9dea-ba0e9504dda7	Person	Robert Mapplethorpe	Mapplethorpe, Robert	American	1947
1	cmoa:parties/b5cb7ad2-50ca-4c5b-8a80-74202ddfb9c2	Person	Jo Baer	Baer, Jo	American	1929
2	cmoa:parties/4c7afe88-ebfe-4e3b-9010-9ddcb1e11674	Person	Franz Ackermann	Ackermann, Franz	German	1963
3	cmoa:parties/b21e4f72-77fc-4c7a-8343-74c67e9c4f3b	Person	unknown Chinese	unknown Chinese	Chinese	
4	cmoa:parties/299f2201-e5b4-4bb0-a6f6-8aec4026769c	Person	Cesare Casati	Casati, Cesare	Italian	1936
5	cmoa:parties/09d3ffbd-fc13-4893-aa78-a3aff5782d6f	Person	Emanuele Ponzio	Ponzio, Emanuele	Italian	1923
6	cmoa:parties/6d212f14-4e5f-4cee-aff8-4cdc6c737a1c	Person	Augustus Welby Northmore Pugin	Pugin, Augustus Welby Northmore	British	1812
7	cmoa:parties/59773f34-4ffc-43fe-8dce-f792fb4b730c	Organization	John Hardman & Co.	Hardman, John & Co.	Scottish	1838
8	cmoa:parties/04f2c765-1fc4-4444-8fdd-9bfa295fa2e9	Person	Alfred Ivory	Ivory, Alfred	British	1856
9	cmoa:parties/56810fc5-62d4-46b7-835e-2934376935aa	Person	Christopher Dresser	Dresser, Christopher	British	1834
10	cmoa:parties/8af6b50e-dd91-4501-8658-6449b8b6602a	Organization	Watcombe Pottery	Watcombe Pottery		1867
11	cmoa:parties/bd1c6390-bb5e-44b1-ae28-348ca2f18c4b	Organization	W. Brownfield & Sons	Brownfield, W. & Sons		1871
12	cmoa:parties/99ed2f03-37e2-4bed-a34b-f621fe8274c7	Organization	Old Hall Earthenware Company Limited	Old Hall Earthenware Company Limited	British	1861
13	cmoa:parties/13f822c9-e255-41b2-8d4c-75109ebcee33	Organization	Ault Pottery	Ault Pottery		1887
14	cmoa:parties/bca38816-6f7e-423d-b070-e42e9cd3143d	Person	Eugène Atget	Atget, Eugène	French	1857
15	cmoa:parties/03896989-0756-4eac-9894-2908dbb8a7be	Person	unknown American	unknown American	American	
16	cmoa:parties/e23614d5-9efb-4d40-aeac-ab3bda89e324	Person	Charles Aldridge	Aldridge, Charles	English	
17	cmoa:parties/d837065b-683c-41ba-b1dd-b32fec0f1f22	Person	Henry Green	Green, Henry	English	
18	cmoa:parties/119dfe94-1f6b-46d0-802d-0420119fafaf	Person	unknown English	unknown English	English	
19	cmoa:parties/2cc5bf0b-1297-42c2-93a1-cf2acfd9b5da	Person	Alex Katz	Katz, Alex	American	1927

```
[4] LOAD DATASET met AS csv FROM met-artists.csv @ artifact file 9
```

Console Timing Datasets Charts

met (**9521** rows)

Views

	Artist_Display_Name (string)	Artist_Display_Bio (string)	Artist_Alpha_Sort (string)	Artist_Nationality (string)
0	Paulding Farnham	1859–1927	Farnham, Paulding	
1	Tiffany & Co.	1837–present	Tiffany & Co.	
2	Louis Comfort Tiffany	American, New York 1848–1933 New York	Tiffany, Louis Comfort	American
3	Tiffany Glass and Decorating Company	American, 1892–1902	Tiffany Glass and Decorating Company	
4	John Henry Belter	American, born Hilter, Germany 1804-1863 New York	Belter, John Henry	
5	Tiffany Studios	1902–32	Tiffany Studios	
6	Louis Henry Sullivan	American, Boston, Massachusetts 1856–1924 Chicago, Illinois	Sullivan, Louis Henry	American
7	Dankmar Adler	American, 1844–1900	Adler, Dankmar	American
8	Samuel Hamlin	1746–1801	Hamlin, Samuel	
9	Thomas Danforth Boardman	1784–1873	Boardman, Thomas Danforth	

cs520_museum Projects Branches Notebook Dataset Caveats Settings

11	Robert Sanderson Sr.	ca. 1608–1693	Sanderson, Robert	
12	Thomas Seymour	1771–1848	Seymour, Thomas	
13	John Doggett	1780–1857	Doggett, John	
14	John Ritto Penniman	American, Milford, Massachusetts 1782–1841 Baltimore, Maryland	Penniman, John Ritto	American
15	Samuel McIntire	1757–1811	McIntire, Samuel	
16	Isaac Broome	1835–1922	Broome, Isaac	
17	Johann Heinrich Otto	ca. 1733–ca. 1800	Otto, Johann Heinrich	
18	William P. Jervis	American (born England), Stoke-on-Trent 1851–1925 Sparta, New Jersey	Jervis, William P.	American
19	Ceramic Art Company, Trenton, New Jersey	American, 1889–1896	Ceramic Art Company	

[5] `LOAD DATASET tate AS csv FROM tate-artists.csv @ artifact file 16`

Console Timing Datasets Charts

tate (**3532** rows)

Views

	id (short)	name (string)	gender (string)	dates (string)	yearOfBirth (date)	yearOfDeath (date)	placeOfBirth (string)	placeOfDeath (string)	
0	10093	Abakanowicz, Magdalena	Female	born 1930	1930-00-01		Polska		http://w
1	0	Abbey, Edwin Austin	Male	1852–1911	1852-00-01	1911-00-01	Philadelphia, United States	London, United Kingdom	http://w
2	2756	Abbott, Berenice	Female	1898–1991	1898-00-01	1991-00-01	Springfield, United States	Monson, United States	http://w
3	1	Abbott, Lemuel Francis	Male	1760–1803	1760-00-01	1803-00-01	Leicestershire, United Kingdom	London, United Kingdom	http://w
4	622	Abrahams, Ivor	Male	born 1935	1935-00-01		Wigan, United Kingdom		http://w
5	2606	Absalon	Male	1964–1993	1964-00-01	1993-00-01	Tel Aviv-Yafo, Yisra'el	Paris, France	http://w
6	9550	Abts, Tomma	Female	born 1967	1967-00-01		Kiel, Deutschland		http://w
7	623	Acconci, Vito	Male	born 1940	1940-00-01		New York, United States		http://w
8	624	Ackling, Roger	Male	1947–2014	1947-00-01	2014-00-01	Isleworth, United Kingdom		http://w
9	625	Ackroyd, Norman	Male	born 1938	1938-00-01		Leeds, United Kingdom		http://w
10	2411	Adam, Robert	Male	1728–1792	1728-00-01	1792-00-01	Kirkcaldy, United Kingdom	London, United Kingdom	http://w
11	626	Adams, Harry William	Male	1868–1947	1868-00-01	1947-00-01	Worcester, United Kingdom	Worcester, United Kingdom	http://w
12	627	Adams, Norman	Male	1927–2005	1927-00-01	2005-00-01	London, United Kingdom		http://w
13	628	Adams, Robert	Male	1917–1984	1917-00-01	1984-00-01	Northampton, United Kingdom		http://w
14	629	Adeney, Bernard	Male	1878–1966	1878-00-01	1966-00-01	London, United Kingdom		http://w
15	630	Adler, Jankel	Male	1895–1949	1895-00-01	1949-00-01	Tuszyn, Polska	Aldbourne, United Kingdom	http://w
16	2608	Adshead, Mary	Female	1904–1995	1904-00-01	1995-00-01	London, United Kingdom	Hampstead, United Kingdom	http://w
17	631	Adzak, Roy	Male	1927–1987	1927-00-01	1987-00-01	Reading, United Kingdom	Paris, France	http://w
18	632	Afro	Male	1912–1976	1912-00-01	1976-00-01	Udine, Italia	Zürich, Schweiz	http://w
19	633	Agar, Eileen	Female	1899–1991	1899-00-01	1991-00-01	Ayacucho, Argentina	London, United Kingdom	http://w

[6] `LOAD DATASET nga AS csv FROM nga-artists.csv @ artifact file 17`

Console Timing Datasets Charts

nga (**15778** rows)

Views

	constituentid (int)	ulanid (int)	preferreddisplayname (string)	forwarddisplayname (string)	lastname (string)	displaydate (string)	artistofngaobject (I
0	11		Baldung, Hans	Hans Baldung	Baldung		true
1	13		Anonymous Artist	Anonymous Artist	Anonymous Artist		true
2	22	500012143	Abbott, Lemuel Francis	Lemuel Francis Abbott	Abbott	British, c. 1755/1761 - 1802	true
3	25	500026325	Pippin, Horace	Horace Pippin	Pippin	American, 1888 - 1946	true
4	27	500009698	Aelst, Willem van	Willem van Aelst	Aelst	Dutch, 1627 - 1683	true
5	32	500018917	Heda, Willem Claesz	Willem Claesz Heda	Heda	Dutch, 1594 - 1680	true

cs520_museum Projects Branches Notebook Dataset Caveats Settings

7	53	500001550	Alexander, Francis	Francis Alexander	Alexander	American, 1800 - 1880	true
8	61		Allen, Luther	Luther Allen	Allen	American, 1780 - 1821	true
9	72	500031250	Altdorfer, Albrecht	Albrecht Altdorfer	Altdorfer	German, 1480 or before - 1538	true
10	80	500018732	Ames, Ezra	Ezra Ames	Ames	American, 1768 - 1836	true
11	85	500018722	Ames, Joseph Alexander	Joseph Alexander Ames	Ames	American, 1816 - 1872	true
12	91	500024770	Andrea di Bartolo	Andrea di Bartolo	Andrea di Bartolo	Sienese, active from 1389 - died 1428	true
13	99	500029319	Angelico, Fra	Fra Angelico	Angelico	Florentine, c. 1395 - 1455	true
14	103	500014404	Anselmi, Michelangelo	Michelangelo Anselmi	Anselmi	Parmese, 1491/1492 - 1554/1556	true
15	119	500007324	Antonello da Messina	Antonello da Messina	Antonello da Messina	Sicilian, c. 1430 - 1479	true
16	121	500022951	Aspertini, Amico	Amico Aspertini	Aspertini	Ferrarese-Bolognese, 1474/1475 - 1552	true
17	122	500016578	Audubon, John James	John James Audubon	Audubon	American, 1785 - 1851	true
18	128	500029919	Audubon, John Woodhouse	John Woodhouse Audubon	Audubon	American, 1812 - 1862	true
19	129	500030575	Avercamp, Hendrick	Hendrick Avercamp	Avercamp	Dutch, 1585 - 1634	true

[7] `LOAD DATASET moma AS csv FROM moma-artist.csv @ artifact file 18`

Console Timing Datasets Charts

moma (**15243** rows)

Views

	ConstituentID (int)	DisplayName (string)	ArtistBio (string)	Nationality (string)	Gender (string)	BeginDate (short)	EndDate (short)	Wiki_QID (stri
0	1	Robert Arneson	American, 1930–1992	American	Male	1930	1992	
1	2	Doroteo Arnaiz	Spanish, born 1936	Spanish	Male	1936	0	
2	3	Bill Arnold	American, born 1941	American	Male	1941	0	
3	4	Charles Arnoldi	American, born 1946	American	Male	1946	0	Q1063584
4	5	Per Arnoldi	Danish, born 1941	Danish	Male	1941	0	
5	6	Danilo Aroldi	Italian, born 1925	Italian	Male	1925	0	
6	7	Bill Aron	American, born 1941	American	Male	1941	0	
7	9	David Aronson	American, born Lithuania 1923	American	Male	1923	0	Q5230870
8	10	Irene Aronson	American, born Germany 1918	American	Female	1918	0	Q19748568
9	11	Jean (Hans) Arp	French, born Germany (Alsace). 1886–1966	French	Male	1886	1966	Q153739
10	12	Jüri Arrak	Estonian, born 1936	Estonian	Male	1936	0	
11	13	J. Arrelano Fischer	Mexican, 1911–1995	Mexican	Male	1911	1995	
12	15	Folke Arstrom	Swedish, 1907–1997	Swedish	Male	1907	1997	
13	16	Cristobal Arteche	Spanish, 1900–1964	Spanish	Male	1900	1964	
14	18	Artko				0	0	
15	19	Richard Artschwager	American, 1923–2013	American	Male	1923	2013	Q568262
16	21	Ruth Asawa	American, 1926–2013	American	Female	1926	2013	Q7382874
17	22	Isidora Aschheim	Israeli	Israeli	Female	0	0	
18	23	Charles Robert Ashbee	British, 1863–1942	British	Male	1863	1942	Q614071
19	24	Donald Ashcraft	American, born 1927	American	Male	1927	0	

[8]

```
from datetime import datetime

def is_valid_date(ds, col):
    for row in ds.rows:
        try:
            # Attempt to convert the string to a date
            print(datetime.strptime(row.get_value(col), '%Y-%m-%d'))
        except ValueError:
            # If conversion fails, return False
            return False
```

cs520_museum Projects Branches Notebook Dataset Caveats Settings

Console Timing Datasets Charts

[9]

```
import pandas as pd
# Use the pandas shape to get the
# dimensions of the dataframe as a (row, columns)
# pair. Scan the nulls dataframe by attribute
# and row, computing nulls_count and null_ratio.
# Construct a dictionary with the attribute_name
# as key and the pair # (null_count, null_ratio)

def get_null_info(dataframe):
    nulls = dataframe.isnull();
    # isnull generates a dataframe where each cell is
    # True only if the corresponding value of the cell is null
    # True only if the corresponding value cell is null
    null_info = {}
    for attr in nulls.columns:
        null_count = 0
        for val in nulls[attr]:
            if val: null_count += 1
        null_info[attr] = (null_count,
                           null_count/ float(nulls.shape[0]))
    return (null_info)

vizierdb.export_module(get_null_info);
```

Console Timing Datasets Charts

[10]

```
import jellyfish as jf
def similarity_score(x, y):
    # compute normalized score per distance-metric
    l = jf.levenshtein_distance(x, y)/ float(max(len(x), len(y)))
    j = 1 - jf.jaro_distance(x, y);
    w = 1 - jf.jaro_winkler(x, y);
    # take the mean of the scores
    s_s = (l + j + w ) /3;
    return s_s;

vizierdb.export_module(similarity_score);
```

Console Timing Datasets Charts

[11]

```
def clean_medium_columns(df_meta, column_name_medium):
    # Display the number of unique values before cleaning
    unique_values_before = df_meta[column_name_medium].nunique()
    # Clean the Medium column and replace the original column
    df_meta = df_meta.assign(Medium=df_meta[column_name_medium].str.strip().str.upper())
    # Display the number of unique values after cleaning
    unique_values_after = df_meta[column_name_medium].nunique()

def clean_problematic_accession_years(df, column_accession_year):
    import pandas as pd
    # Assuming df is your DataFrame
    # Identify and handle problematic values first if needed
```

cs520_museum Projects Branches Notebook Dataset Caveats Settings

```
    # Extract the year using pandas string methods
    df[column_accession_year] = df[column_accession_year].str.extract(r'(\d{4})', expand=False)
    # Convert the extracted values to numeric (optional, if needed)
    df[column_accession_year] = pd.to_numeric(df[column_accession_year], errors='coerce')
    # Drop the original column
    df = df.drop(columns=[column_accession_year])
    # Rename the new column
    df = df.rename(columns={column_accession_year: column_accession_year})

def clean_problematic_dates(df, begin_date, end_date, alt_date):
    import pandas as pd
    import numpy as np
    import re
    # Create a boolean series to identify rows where Object End Date is before Object Begin Date
    problematic_dates = df[end_date] < df[begin_date]
    # Extract the begin date when it contains a dash and "B.C."
    def extract_bc_date(x):
        match = re.search(r'\(?\-?(\d+)\s*(?:B\.C\.?)?\)?', str(x))
        return match.group(1) if match else np.nan
    problematic_begin_dates = df.loc[problematic_dates]
    problematic_begin_dates = df.loc[problematic_dates, alt_date].apply(extract_bc_date)
    print(problematic_begin_dates);
    # Replace the values in the Object Begin Date column with extracted dates
    df.loc[problematic_dates, begin_date] = problematic_begin_dates.astype(float)
    df[begin_date] = pd.to_numeric(df[begin_date], errors='coerce')
    df[end_date] = pd.to_numeric(df[end_date], errors='coerce')

vizierdb.export_module(clean_medium_columns);
vizierdb.export_module(clean_problematic_accession_years);
vizierdb.export_module(clean_problematic_dates);
```

Console Timing Datasets Charts

[12]

```
vizierdb.get_module("clean_medium_columns");
vizierdb.get_module("clean_problematic_accession_years");
vizierdb.get_module("clean_problematic_dates");

import pandas as pd
import numpy as np
df_tate = vizierdb.get_data_frame('tate');
df_moma = vizierdb.get_data_frame('moma');
df_meta = vizierdb.get_data_frame('metobjects');
df_cmoa = vizierdb.get_data_frame('cmoa');
df_met = vizierdb.get_data_frame('met');
df_nga = vizierdb.get_data_frame('nga');

# Do cleaning operations on Medium column
column_name_medium = {'meta': 'Medium', 'moma' : 'Medium', 'cmoa': 'medium', 'tate': 'medium'}
#clean_medium_columns(df_meta, column_name_medium['meta']);
#clean_medium_columns(df_moma, column_name_medium['moma']);
#clean_medium_columns(df_cmoa, column_name_medium['cmoa']);
#clean_medium_columns(df_tate, column_name_medium['tate']);


# Do cleaning operation on Problematic Accession Years
column_name_accession_years = {'meta': 'AccessionYear'}
clean_problematic_accession_years(df_meta, column_name_accession_years['meta']);

# Do cleaning operation on Problematic Begin Dates and End Dates
column_name_begin_dates = {'meta': 'Object_Begin_Date', 'moma' : 'BeginDate', 'cmoa': 'creation_date_earliest'}
column_name_end_dates = {'meta': 'Object_End_Date', 'moma' : 'EndDate', 'cmoa': 'creation_date_latest'}
column_name_alt_dates = {'meta': 'Object_Date', 'moma' : 'Date', 'cmoa': 'creation_date'}
clean_problematic_dates(df_meta, column_name_begin_dates['meta'], column_name_end_dates['meta'], column_name_alt_dates['me

# Do cleaning operation on Problematic Birth dates
column_name_begin_dates = {'moma' : 'BeginDate', 'cmoa': 'birth_date'}
```

cs520_museum Projects Branches Notebook Dataset Caveats Settings

```
clean_problematic_dates(df_moma, column_name_begin_dates['moma'], column_name_end_dates['moma'], column_name_alt_dates['mo
clean_problematic_dates(df_cmoa, column_name_begin_dates['cmoa'], column_name_end_dates['cmoa'], column_name_alt_dates['cm
```

Console Timing Datasets Charts

```
1690        8
5927      664
7370        8
7416      664
8248       16
13184     664
17923     664
21938    1545
23666       9
31822       9
41595     664
45328    1861
46482     664
46716     664
47597     664
Name: Object_Date, dtype: object

1        1936
2        1941
3        1946
4        1941
5        1925
         ...
15232    1984
15233    1987
15234    1917
15238    1953
15240    1971
Name: BeginDate, Length: 6446, dtype: object

3555    0699
4800    1936
5584    1890
Name: birth_date, dtype: object
```

[13]

```
vizierdb.get_module("clean_medium_columns");
vizierdb.get_module("clean_problematic_accession_years");
vizierdb.get_module("clean_problematic_dates");

import pandas as pd
import numpy as np
df_tate = vizierdb.get_data_frame('tate');
df_moma = vizierdb.get_data_frame('moma');
df_meta = vizierdb.get_data_frame('metobjects');
df_cmoa = vizierdb.get_data_frame('cmoa');
df_met = vizierdb.get_data_frame('met');
df_nga = vizierdb.get_data_frame('nga');


# Do cleaning operations on Medium column
column_name_medium = {'meta': 'Medium', 'moma' : 'Medium', 'cmoa': 'medium', 'tate': 'medium'}
#clean_medium_columns(df_meta, column_name_medium['meta']);
#clean_medium_columns(df_moma, column_name_medium['moma']);
#clean_medium_columns(df_cmoa, column_name_medium['cmoa']);
#clean_medium_columns(df_tate, column_name_medium['tate']);


# Do cleaning operation on Problematic Accession Years
column_name_accession_years = {'meta': 'AccessionYear'}
clean_problematic_accession_years(df_meta, column_name_accession_years['meta']);

# Do cleaning operation on Problematic Begin Dates and End Dates
```

```
column_name_alt_dates = {'meta': 'Object_Date', 'moma' : 'Date', 'cmoa': 'creation_date'}
clean_problematic_dates(df_meta, column_name_begin_dates['meta'], column_name_end_dates['meta'], column_name_alt_dates['me

# Do cleaning operation on Problematic Birth dates
column_name_begin_dates = {'moma' : 'BeginDate', 'cmoa': 'birth_date'}
column_name_end_dates = {'moma' : 'EndDate', 'cmoa': 'death_date'}
column_name_alt_dates = {'moma' : 'BeginDate', 'cmoa': 'birth_date'}
clean_problematic_dates(df_moma, column_name_begin_dates['moma'], column_name_end_dates['moma'], column_name_alt_dates['mc
clean_problematic_dates(df_cmoa, column_name_begin_dates['cmoa'], column_name_end_dates['cmoa'], column_name_alt_dates['cm


# Do cleaning operations
#column_name_mediums = {'meta': 'Medium'}
#clean_medium_columns(df_meta, column_name_mediums);
#clean_problematic_accession_years(df_meta);
#clean_problematic_dates(df_meta);


# Merge the 'Met' and 'MetObjects' DataFrames based on the artist names
# Use 'how='left'' to perform a left join, preserving all rows from the 'Met' DataFrame
merged_df = pd.merge(df_meta, df_met,
                     how='left', on='Artist_Display_Name')

data = merged_df;


data['Object_Begin_Date'] = pd.to_datetime(data['Object_Begin_Date'], errors='coerce')
data['Object_End_Date'] = pd.to_datetime(data['Object_End_Date'], errors='coerce')
# Find the oldest objects based on 'Object Begin Date' and 'Object End Date'
# Find the oldest objects based on 'Object Begin Date' and 'Object End Date'
oldest_object = data.loc[data['Object_Begin_Date'].idxmin()]

# Get the 'Object Date', 'Object Begin Date', and 'Object End Date' of the oldest object
oldest_object_date = oldest_object['Object_Date']
oldest_object_begin_date = oldest_object['Object_Begin_Date']
oldest_object_end_date = oldest_object['Object_End_Date']

# Print the oldest object information and date range
print("Oldest Object Date:", oldest_object_date)
print("Oldest Object Begin Date:", oldest_object_begin_date)
print("Oldest Object End Date:", oldest_object_end_date)

# Initialize a set to store unique non-anonymous artist names
unique_non_anonymous_artist_names = set()

# Iterate through the 'Artist Display Name' column and add unique non-anonymous names to the set
for artists in data['Artist_Display_Name']:
    if pd.notnull(artists):
        # Split the string into individual artist names using the '|' delimiter
        artist_names = artists.split('|')
        # Check if the artist name is not anonymous and not unknown, then add it to the set
        non_anonymous_names = [artist.strip() for artist in artist_names if artist.strip() and not artist.strip().startswi
        unique_non_anonymous_artist_names.update(non_anonymous_names)

# Get the number of unique non-anonymous artist names
num_unique_non_anonymous_artists = len(unique_non_anonymous_artist_names)

# Print the number of unique non-anonymous artist names
print("Number of unique non-anonymous artist names:", num_unique_non_anonymous_artists)

from collections import Counter

# Function to clean and extract artist names
def extract_artist_names(artists):
    if pd.notnull(artists):
        # Split the string into individual artist names using the '|' delimiter
        artist_names = artists.split('|')
        # Clean and return the artist names (removing empty strings, whitespaces, and anonymous names)
        return [artist.strip() for artist in artist_names if artist.strip() and not artist.strip().startswith(('Anonymous'
    else:
        return []
```

```
all_artist_names = [name for sublist in data['Artist_Display_Name'].apply(extract_artist_names) for name in sublist]

# Count the occurrences of each artist name
artist_name_counts = Counter(all_artist_names)

# Find the most frequent artist name
most_frequent_artist_name = artist_name_counts.most_common(1)[0]

# Print the most frequent artist name and its count
print("Most Frequent Artist Name:", most_frequent_artist_name[0])
print("Count:", most_frequent_artist_name[1])

# Initialize variables to store oldest object information
oldest_accession_year = float('inf')  # Set to positive infinity initially
oldest_object_titles = []

# Iterate through the data to find the oldest object(s)
for index, row in data.iterrows():
    accession_year = row['AccessionYear']
    try:
        # Try converting the accession year to an integer (ignore non-four-digit entries)
        accession_year = int(accession_year)
        # Check if the accession year is a valid four-digit number
        if 1000 <= accession_year <= 9999:
            # Compare accession years to find the oldest object(s)
            if accession_year < oldest_accession_year:
                oldest_accession_year = accession_year
                oldest_object_titles = [row['Title']]
            elif accession_year == oldest_accession_year:
                oldest_object_titles.append(row['Title'])
    except:
        # Handle invalid accession years (non-integer values) if any
        pass

# Print the titles of the oldest object(s)
print("Oldest Object(s) Title(s):")
for title in oldest_object_titles:
    print(title)

# Initialize empty lists for each artist-related column
artist_roles = []
artist_display_names = []
artist_begin_dates = []
artist_end_dates = []
artist_gender = []
artist_nationality = []
object_ids = []

# Iterate over the rows and columns to extract artist information
for index, row in data.iterrows():
    object_id = row['Object_ID']

    # Split the columns into lists
    display_names = str(row['Artist_Display_Name']).split('|')
    begin_dates = str(row['Artist_Begin_Date']).split('|')
    end_dates = str(row['Artist_End_Date']).split('|')
    gender = str(row['Artist_Gender']).split('|')
    nationality = str(row['Artist_Nationality']).split('|')

    # Append the corresponding elements to the lists
    for display_name, begin_date, end_date, gender, nationality in zip(display_names, begin_dates, end_dates, gender, nati
        artist_display_names.append(display_name.strip())
        artist_begin_dates.append(begin_date.strip())
        artist_end_dates.append(end_date.strip())
        artist_gender.append(gender.strip())
        artist_nationality.append(nationality.strip())
        object_ids.append(object_id)

# Create a new DataFrame from the lists
artists_df = pd.DataFrame({
    'Object ID': object_ids,
```

```
    'Artist End Date': artist_end_dates,
    'Artist Gender': artist_gender,
    'Artist Nationality': artist_nationality
})

# Print the new DataFrame with artist information
final_data = artists_df
# Additional mapping between nationalities and countries
# Extended mapping between nationalities and countries
nationality_to_country_mapping = {
    'American': 'United States',
    'British': 'United Kingdom',
    'French': 'France',
    'German': 'Germany',
    'Italian': 'Italy',
    'Japanese': 'Japan',
    'Chinese': 'China',
    'Indian': 'India',
    'Australian': 'Australia',
    'Canadian': 'Canada',
    'Brazilian': 'Brazil',
    'Russian': 'Russia',
    'South African': 'South Africa',
    'South Korean': 'South Korea',
    'Mexican': 'Mexico',
    'Nigerian': 'Nigeria',
    'Argentinian': 'Argentina',
    'Egyptian': 'Egypt',
    'Thai': 'Thailand',
    'Spanish': 'Spain',
    'Swedish': 'Sweden',
    'Netherlands': 'Netherlands',
    'Turkish': 'Turkey',
    'Switzerland': 'Switzerland',
    'Norwegian': 'Norway',
    'Danish': 'Denmark',
    'Ireland': 'Ireland',
    'Kenyan': 'Kenya',
    'Nigerien': 'Niger',
    'Moroccan': 'Morocco',
    'Ethiopian': 'Ethiopia',
    'Ghanaian': 'Ghana',
    'Ugandan': 'Uganda',
    'Senegalese': 'Senegal',
    'Cameroonian': 'Cameroon',
    'Tanzanian': 'Tanzania',
    'Rwandan': 'Rwanda',
    'Sudanese': 'Sudan',
    'Ivorian': 'Ivory Coast',
    'Chinese': 'China',
    'Japanese': 'Japan',
    'Indian': 'India',
    'Indonesian': 'Indonesia',
    'South Korean': 'South Korea',
    'Vietnamese': 'Vietnam',
    'Filipino': 'Philippines',
    'Thai': 'Thailand',
    'Malaysian': 'Malaysia',
    'Singaporean': 'Singapore',
    'Bangladeshi': 'Bangladesh',
    'Pakistani': 'Pakistan',
    'Sri Lankan': 'Sri Lanka',
    'Nepali': 'Nepal',
    'Mongolian': 'Mongolia',
    'Kazakhstani': 'Kazakhstan',
    # Add more mappings as needed
}

# Create a new column 'Country' based on the nationality-to-country mapping
final_data['Country'] = final_data['Artist Nationality'].map(nationality_to_country_mapping)
```

```
# Extended manual latitude and longitude for more countries (replace with actual values)
country_coordinates = {
    'United States': {'Latitude': 37.7749, 'Longitude': -122.4194},
    'United Kingdom': {'Latitude': 51.509865, 'Longitude': -0.118092},
    'France': {'Latitude': 48.8566, 'Longitude': 2.3522},
    'Germany': {'Latitude': 51.1657, 'Longitude': 10.4515},
    'Italy': {'Latitude': 41.9028, 'Longitude': 12.4964},
    'Japan': {'Latitude': 35.6895, 'Longitude': 139.6917},
    'China': {'Latitude': 39.9042, 'Longitude': 116.4074},
    'India': {'Latitude': 20.5937, 'Longitude': 78.9629},
    'Australia': {'Latitude': -25.2744, 'Longitude': 133.7751},
    'Canada': {'Latitude': 56.1304, 'Longitude': -106.3468},
    'Brazil': {'Latitude': -14.235, 'Longitude': -51.9253},
    'Russia': {'Latitude': 61.524, 'Longitude': 105.3188},
    'South Africa': {'Latitude': -30.5595, 'Longitude': 22.9375},
    'South Korea': {'Latitude': 35.9078, 'Longitude': 127.7669},
    'Mexico': {'Latitude': 23.6345, 'Longitude': -102.5528},
    'Nigeria': {'Latitude': 9.0820, 'Longitude': 8.6753},
    'Argentina': {'Latitude': -38.4161, 'Longitude': -63.6167},
    'Egypt': {'Latitude': 26.8206, 'Longitude': 30.8025},
    'Thailand': {'Latitude': 15.8700, 'Longitude': 100.9925},
    'Spain': {'Latitude': 40.4637, 'Longitude': -3.7492},
    'Sweden': {'Latitude': 60.1282, 'Longitude': 18.6435},
    'Netherlands': {'Latitude': 52.3676, 'Longitude': 4.9041},
    'Turkey': {'Latitude': 38.9637, 'Longitude': 35.2433},
    'Switzerland': {'Latitude': 46.8182, 'Longitude': 8.2275},
    'Norway': {'Latitude': 60.4720, 'Longitude': 8.4689},
    'Denmark': {'Latitude': 56.2639, 'Longitude': 9.5018},
    'Ireland': {'Latitude': 53.1424, 'Longitude': -7.6921},
    'Kenya': {'Latitude': 1.2921, 'Longitude': 36.8219},
    'Niger': {'Latitude': 17.6078, 'Longitude': 8.0817},
    'Morocco': {'Latitude': 31.7917, 'Longitude': -7.0926},
    'Ethiopia': {'Latitude': 9.1450, 'Longitude': 40.4897},
    'Ghana': {'Latitude': 7.2500, 'Longitude': -2.3333},
    'Uganda': {'Latitude': 1.3733, 'Longitude': 32.2903},
    'Senegal': {'Latitude': 14.6928, 'Longitude': -17.4467},
    'Cameroon': {'Latitude': 7.3697, 'Longitude': 12.3547},
    'Tanzania': {'Latitude': -6.369028, 'Longitude': 34.888822},
    'Rwanda': {'Latitude': -1.9403, 'Longitude': 29.8739},
    'Sudan': {'Latitude': 12.8628, 'Longitude': 30.2176},
    'Ivory Coast': {'Latitude': 7.5400, 'Longitude': -5.5471},
    'China': {'Latitude': 35.8617, 'Longitude': 104.1954},
    'Japan': {'Latitude': 36.2048, 'Longitude': 138.2529},
    'India': {'Latitude': 20.5937, 'Longitude': 78.9629},
    'Indonesia': {'Latitude': -0.7893, 'Longitude': 113.9213},
    'South Korea': {'Latitude': 35.9078, 'Longitude': 127.7669},
    'Vietnam': {'Latitude': 14.0583, 'Longitude': 108.2772},
    'Philippines': {'Latitude': 12.8797, 'Longitude': 121.7740},
    'Thailand': {'Latitude': 15.8700, 'Longitude': 100.9925},
    'Malaysia': {'Latitude': 4.2105, 'Longitude': 101.9758},
    'Singapore': {'Latitude': 1.3521, 'Longitude': 103.8198},
    'Bangladesh': {'Latitude': 23.6850, 'Longitude': 90.3563},
    'Pakistan': {'Latitude': 30.3753, 'Longitude': 69.3451},
    'Sri Lanka': {'Latitude': 7.8731, 'Longitude': 80.7718},
    'Nepal': {'Latitude': 28.3949, 'Longitude': 84.1240},
    'Mongolia': {'Latitude': 46.8625, 'Longitude': 103.8467},
    'Kazakhstan': {'Latitude': 48.0196, 'Longitude': 66.9237},
    # Add more mappings as needed
}

# Merge coordinates with the final_data DataFrame
final_data = pd.merge(final_data, pd.DataFrame(country_coordinates).T, left_on='Country', right_index=True, how='left')

from bokeh.plotting import figure, show
from bokeh.io import output_notebook
from bokeh.models import ColumnDataSource
from bokeh.palettes import Category20_20

# Create a Bokeh figure
p = figure(title="Nationalities Map",
```

```
# Create a ColumnDataSource from the DataFrame
source = ColumnDataSource(final_data)

# Add circle glyphs for each country
p.circle(x='Longitude', y='Latitude', size=10, color=Category20_20[0], alpha=0.6, source=source)
p.text(x='Longitude', y='Latitude', text='Country', text_font_size='10pt', source=source)

# Show the plot
show(p)
```

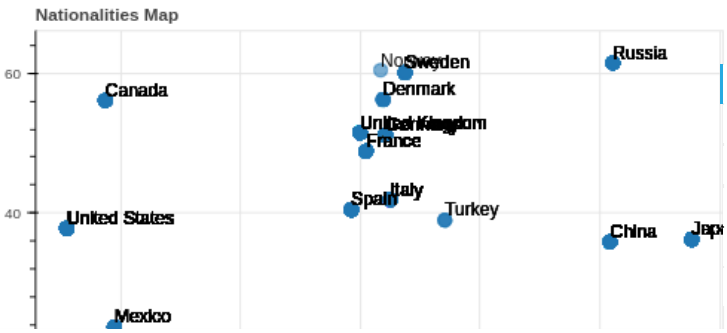
Console Timing Datasets Charts

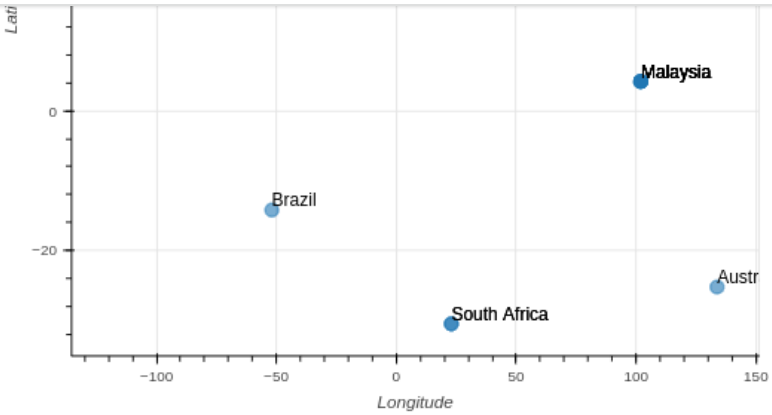
```
1690        8
5927      664
7370        8
7416      664
8248       16
13184     664
17923     664
21938    1545
23666       9
31822       9
41595     664
45328    1861
46482     664
46716     664
47597     664
Name: Object_Date, dtype: object

1        1936
2        1941
3        1946
4        1941
5        1925
         ...
15232    1984
15233    1987
15234    1917
15238    1953
15240    1971
Name: BeginDate, Length: 6446, dtype: object

3555    0699
4800    1936
5584    1890
Name: birth_date, dtype: object

Oldest Object Date: ca. 400,000–240,000 B.C.
Oldest Object Begin Date: 1969-12-31 23:59:59.999600
Oldest Object End Date: 1969-12-31 23:59:59.999760
Number of unique non-anonymous artist names: 13957
Most Frequent Artist Name: Walker Evans
Count: 773
Oldest Object(s) Title(s):
Marble sarcophagus with garlands
```





[14]

```
vizierdb.get_module("get_null_info");
from collections import Counter
import numpy as np
import pandas as pd
pd.options.mode.chained_assignment = None  # default='warn'
# Use the pandas shape to get the
# dimensions of the dataframe as a (row, columns)
# pair. Scan the nulls dataframe by attribute
# and row, computing nulls_count and null_ratio.
# Construct a dictionary with the attribute_name
# as key and the pair # (null_count, null_ratio)


metobjects = vizierdb.get_data_frame('metobjects');
cmoa = vizierdb.get_data_frame('cmoa');
met = vizierdb.get_data_frame('met');
tate = vizierdb.get_data_frame('tate');
nga = vizierdb.get_data_frame('nga');
moma = vizierdb.get_data_frame('moma');


import re
import pandas as pd

# Function to apply transformation rules to a DataFrame column
def apply_t_rules(table, attr, rules):
    cardinality = table.shape[0]
    for i in range(0, cardinality):
        if pd.notna(table[attr].iloc[i]):
            # Convert the column value to string before applying rules
            value_str = str(table[attr].iloc[i])
            for rule in rules:
                # Apply regular expression substitution
                value_str = re.sub(rule[0], rule[1], value_str, 0, re.MULTILINE)
                # Assign the modified value back to the DataFrame
                table[attr].iloc[i] = value_str
    return table

# Function to clean and transform date columns in the 'Met' dataset
def clean_transform_met(met_df):
    # Define transformation rules for date columns
    Date_t_rules = []
    # Remove YYYY-MM-DD format and convert to YYYY
    LHS = r"(\d{4})-\d{2}-\d{2}"
    RHS = r"\1"
    Date_t_rules.append((LHS, RHS))

    # Replace 9999 with NaN to indicate the artist is still alive
    LHS = r"9999"
    RHS = "NaN"
    Date_t_rules.append((LHS, RHS))

    # Apply rules to the 'Artist Begin Date' column
```

```
    # Apply rules to the 'Artist End Date' column
    met_df = apply_t_rules(met_df, 'Artist_End_Date', Date_t_rules)

    return met_df


# Function to clean and transform date columns in the 'Tate' dataset
def clean_transform_tate(tate_df):
    # Define transformation rules for date columns
    Date_t_rules = []
    LHS = r"(\d{4})-\d{2}-\d{2}"
    RHS = r"\1"
    Date_t_rules.append((LHS, RHS))

    # Apply rules to the 'Artist_Begin_Date' column (renamed to 'yearOfBirth')
    tate_df = apply_t_rules(tate_df, 'yearOfBirth', Date_t_rules)
    # Apply rules to the 'Artist_End_Date' column (renamed to 'yearOfDeath')
    tate_df = apply_t_rules(tate_df, 'yearOfDeath', Date_t_rules)
    return tate_df

# Function to clean and transform date columns in the 'CMOA' dataset
def clean_transform_cmoa(cmoa_df):
    # Define transformation rules for date columns
    Date_t_rules = []
    # Extract the year from the date and convert to numeric value
    LHS = r"(\d{4})-\d{2}-\d{2}"
    RHS = r"\1"
    Date_t_rules.append((LHS, RHS))

    # Apply rules to the 'birth_date' column
    cmoa_df = apply_t_rules(cmoa_df, 'birth_date', Date_t_rules)

    # Apply rules to the 'death_date' column
    cmoa_df = apply_t_rules(cmoa_df, 'death_date', Date_t_rules)

    return cmoa_df
# Clean and transform date columns in the 'Met' dataset
met = clean_transform_met(met)
# Clean and transform date columns in the 'CMOA' dataset
cmoa = clean_transform_cmoa(cmoa)
# Clean and transform date columns in the 'Tate' dataset
tate = clean_transform_tate(tate)

import pandas as pd
pd.options.mode.chained_assignment = None  # default='warn'
#define relational-algebraic operators
def Rename(input, paired_list):
        return input.rename(index = str, columns = paired_list)
def Project(input, proj_list):
        return input[proj_list]
def Union(left, right):
        output = pd.DataFrame(columns=list(left.columns))
        return output._append(left)._append(right);
def Natural_Join(left, right):
    output = pd.DataFrame(columns = list(left.columns)
                              + list(right.columns))
    print(output)
    output = pd.merge(left, right)
    return output

N12 = Natural_Join(met, metobjects);
vizierdb.save_data_frame('natural_joined_before', N12);

Agcy1 = cmoa
Agcy1['ulan_id'] = np.nan
Agcy1['source'] = 'cmoa'
Agcy1['gender'] = np.nan

Agcy2 = met
Agcy2['cited_name'] = np.nan
Agcy2['source'] = 'met'
```

```
Agcy3 = tate
Agcy3['ulan_id'] = np.nan
Agcy3['cited_name'] = np.nan
Agcy3['source'] = 'tate'
Agcy3['nationality'] = np.nan


Agcy4 = nga
Agcy4['ulan_id'] = np.nan
Agcy4['cited_name'] = np.nan
Agcy4['source'] = 'nga'
Agcy4['nationality'] = np.nan
Agcy4['gender'] = np.nan

Agcy5 = moma
Agcy5['source'] = 'moma'
Agcy5['cited_name'] = np.nan

# given source-to-target matches
Agcy1_keys = {'full_name': 'full_name',
              'cited_name': 'cited_name',
              'birth_date': 'birth_year',
              'death_date' : 'death_year',
              'nationality' : 'nationality',
              'ulan_id': 'ulan_id',
              'source': 'source',
              'gender': 'gender'}
Agcy2_keys = {'Artist_Display_Name': 'full_name',
              'cited_name': 'cited_name',
              'Artist_Nationality': 'nationality',
              'Artist_Begin_Date': 'birth_year',
              'Artist_End_Date': 'death_year',
              'Artist_ULAN_URL': 'ulan_id',
              'Artist_Gender': 'gender'}
Agcy3_keys = {'name': 'full_name',
              'cited_name': 'cited_name',
              'yearOfBirth': 'birth_year',
              'yearOfDeath' : 'death_year',
              'nationality' : 'nationality',
              'ulan_id': 'ulan_id',
              'gender': 'gender'}
Agcy4_keys = {'forwarddisplayname': 'full_name',
              'cited_name': 'cited_name',
              'beginyear': 'birth_year',
              'endyear' : 'death_year',
              'nationality' : 'nationality',
              'ulanid': 'ulan_id',
              'gender': 'gender'}
Agcy5_keys = {'DisplayName': 'full_name',
              'cited_name': 'cited_name',
              'BeginDate': 'birth_year',
              'EndDate' : 'death_year',
              'Nationality' : 'nationality',
              'ULAN': 'ulan_id',
              'Gender': 'gender'}

Target_keys = {'postcode' : 'po',
            'crimedecile': 'cr'}
N4 = Agcy1
N3 = Rename(Project(N4, Agcy1_keys.keys()), Agcy1_keys)
N6 = Agcy2
N5 = Rename(Project(N6, Agcy2_keys.keys()), Agcy2_keys)
N8 = Agcy3
N7 = Rename(Project(N8, Agcy3_keys.keys()), Agcy3_keys)
N10 = Agcy4
N9 = Rename(Project(N10, Agcy4_keys.keys()), Agcy4_keys)
N12 = Agcy5
N11 = Rename(Project(N12, Agcy5_keys.keys()), Agcy5_keys)

print(N3.head())
print(N5.head())
```

```
print(N11.head())

Agencies = []
Agencies.append(N3)
Agencies.append(N5)
Agencies.append(N7)
Agencies.append(N9)
Agencies.append(N11)

Agcy1 = N3
Agcy1['gender'] = Agcy1['gender'].apply(lambda x:str(x).lower())

Agcy2 = N5
Agcy2['gender'] = Agcy3['gender'].apply(lambda x:str(x).lower())

Agcy3 = N7
Agcy3['gender'] = Agcy3['gender'].apply(lambda x:str(x).lower())

Agcy4 = N9
Agcy4['gender'] = Agcy4['gender'].apply(lambda x:str(x).lower())

Agcy5 = N11
Agcy5['gender'] = Agcy5['gender'].apply(lambda x:str(x).lower())

# Add 'source' column for each dataset

Agcy1 = Agcy1.assign(source='cmoa')
Agcy2 = Agcy2.assign(source='met')
Agcy3 = Agcy3.assign(source='tate')
Agcy4 = Agcy4.assign(source='nga')
Agcy5 = Agcy5.assign(source='moma')

import pandas as pd
#FD A->B is represented as ['A','B']
def apply_fd_based_repair(df, fd_list, threshold):
    for fd in fd_list:
        u_vals = df[fd[0]].unique()
        for u_val in u_vals:
            #given a unique value of the determinant
            #find all values for its dependent
            corr_vals = list(df.loc[df[fd[0]] == u_val, fd[1]])
            pop_size = len(corr_vals)
            #find the most frequent dependent value
            # Find the maximum count
            corr_vals = Counter(corr_vals)
            max_count = max(corr_vals.values())
            # Find all elements with the maximum count (modes)
            modes = [k for k, v in corr_vals.items() if v == max_count]
            #compute the frequency of the mode
            mode_freq = max_count/float(pop_size)
            #if the mode frequency is less than 1 and above the threshold then repair
            if mode_freq < 1 and mode_freq > threshold:
                df.loc[df[fd[0]] == u_val, fd[1]] = modes[0]
                return df

Agencies = []
Agencies.append(Agcy1)
Agencies.append(Agcy2)
Agencies.append(Agcy3)
Agencies.append(Agcy4)
Agencies.append(Agcy5)


for Agency in Agencies:
    apply_fd_based_repair(Agency, [['full_name','nationality']],  0.7);
    apply_fd_based_repair(Agency, [['full_name','birth_year']],  0.7);
    apply_fd_based_repair(Agency, [['full_name','death_year']],  0.7);

# Drop unnecessary columns
Agcy1 = Agcy1[['full_name', 'cited_name', 'birth_year', 'death_year', 'nationality', 'ulan_id', 'source', 'gender']]
Agcy2 = Agcy2[['full_name', 'cited_name', 'birth_year', 'death_year', 'nationality', 'ulan_id', 'source', 'gender']]
```

cs520_museum Projects Branches Notebook Dataset Caveats Settings

```
Agcy5 = Agcy5[['full_name', 'cited_name', 'birth_year', 'death_year', 'nationality', 'ulan_id', 'source', 'gender']]


vizierdb.save_data_frame('cmoa_repaired', Agcy1)
vizierdb.save_data_frame('met_repaired', Agcy2)
vizierdb.save_data_frame('tate_repaired', Agcy3)
vizierdb.save_data_frame('nga_repaired', Agcy4)
vizierdb.save_data_frame('moma_repaired', Agcy5)
```

Console Timing Datasets Charts

```
Empty DataFrame
Columns: [Artist_Display_Name, Artist_Display_Bio, Artist_Alpha_Sort, Artist_Nationality, Artist_Begin_Date, Artist_End_D
Index: []

full_name            cited_name birth_year  ... ulan_id source  gender
0  Robert Mapplethorpe  Mapplethorpe, Robert       1947  ...     NaN   cmoa     NaN
1              Jo Baer              Baer, Jo       1929  ...     NaN   cmoa     NaN
2      Franz Ackermann      Ackermann, Franz       1963  ...     NaN   cmoa     NaN
3      unknown Chinese       unknown Chinese       None  ...     NaN   cmoa     NaN
4        Cesare Casati        Casati, Cesare       1936  ...     NaN   cmoa     NaN

[5 rows x 8 columns]

full_name  ...  gender
0                     Paulding Farnham  ...    None
1                        Tiffany & Co.  ...    None
2                Louis Comfort Tiffany  ...    None
3  Tiffany Glass and Decorating Company  ...    None
4                    John Henry Belter  ...    None

[5 rows x 7 columns]

full_name  cited_name birth_year  ... nationality  ulan_id  gender
0  Abakanowicz, Magdalena         NaN       1930  ...         NaN      NaN  Female
1     Abbey, Edwin Austin         NaN       1852  ...         NaN      NaN    Male
2        Abbott, Berenice         NaN       1898  ...         NaN      NaN  Female
3  Abbott, Lemuel Francis         NaN       1760  ...         NaN      NaN    Male
4          Abrahams, Ivor         NaN       1935  ...         NaN      NaN    Male

[5 rows x 7 columns]

full_name  cited_name  ...      ulan_id  gender
0           Hans Baldung         NaN  ...          NaN     NaN
1       Anonymous Artist         NaN  ...          NaN     NaN
2  Lemuel Francis Abbott         NaN  ...  500012143.0     NaN
3          Horace Pippin         NaN  ...  500026325.0     NaN
4       Willem van Aelst         NaN  ...  500009698.0     NaN

[5 rows x 7 columns]

full_name  cited_name  birth_year  ...  nationality      ulan_id  gender
0   Robert Arneson         NaN        1930  ...     American          NaN    Male
1   Doroteo Arnaiz         NaN        1936  ...      Spanish          NaN    Male
2      Bill Arnold         NaN        1941  ...     American          NaN    Male
3  Charles Arnoldi         NaN        1946  ...     American  500027998.0    Male
4      Per Arnoldi         NaN        1941  ...       Danish          NaN    Male

[5 rows x 7 columns]
```

[15]

```
select * from met_repaired;
```

Console Timing Datasets Charts

	full_name (string)	cited_name (real)	birth_year (string)	death_year (string)	nationality (string)	ulan_id (string)	source (s
0	Paulding Farnham		1859	1927		http://vocab.getty.edu/page/ulan/500336597	met
1	Tiffany & Co.		1837	NaN		http://vocab.getty.edu/page/ulan/500330306	met
2	Louis Comfort Tiffany		1848	1933	American	http://vocab.getty.edu/page/ulan/500030415	met
3	Tiffany Glass and Decorating Company		1892	1902		http://vocab.getty.edu/page/ulan/500331512	met
4	John Henry Belter		1804	1863		http://vocab.getty.edu/page/ulan/500001413	met
5	Tiffany Studios		1902	1932		http://vocab.getty.edu/page/ulan/500331813	met
6	Louis Henry Sullivan		1856	1924	American	http://vocab.getty.edu/page/ulan/500013453	met
7	Dankmar Adler		1844	1900	American	http://vocab.getty.edu/page/ulan/500000517	met
8	Samuel Hamlin		1746	1801		http://vocab.getty.edu/page/ulan/500056845	met
9	Thomas Danforth Boardman		1784	1873		http://vocab.getty.edu/page/ulan/500334231	met
10	John Hull		1624	1683		http://vocab.getty.edu/page/ulan/500098953	met
11	Robert Sanderson Sr.		1605	1693		http://vocab.getty.edu/page/ulan/500106131	met
12	Thomas Seymour		1771	1848		http://vocab.getty.edu/page/ulan/500023428	met
13	John Doggett		1780	1857		http://vocab.getty.edu/page/ulan/500064036	met
14	John Ritto Penniman		1782	1841	American	http://vocab.getty.edu/page/ulan/500018826	met
15	Samuel McIntire		1757	1811		http://vocab.getty.edu/page/ulan/500023295	met
16	Isaac Broome		1835	1922		http://vocab.getty.edu/page/ulan/500314986	met
17	Johann Heinrich Otto		1730	1800		http://vocab.getty.edu/page/ulan/500331480	met
18	William P. Jervis		1851	1925	American	http://vocab.getty.edu/page/ulan/500111370	met
19	Ceramic Art Company, Trenton, New Jersey		1889	1896		http://vocab.getty.edu/page/ulan/500356006	met

[16]

```
import pandas as pd
import fuzzymatcher
import numpy as np
import dedupe
import nltk
from scipy.sparse import csr_matrix
import itertools
from nltk.tokenize import word_tokenize

df_cmoa = vizierdb.get_data_frame('cmoa_repaired')
df_met = vizierdb.get_data_frame('met_repaired')
df_tate = vizierdb.get_data_frame('tate_repaired')
df_nga = vizierdb.get_data_frame('nga_repaired')
df_moma = vizierdb.get_data_frame('moma_repaired')

# Extract ULAN identifiers from the 'Met' dataset
df_met['ulan_id'] = df_met['ulan_id'].str.extract(r'\/ulan\/(\d+)').astype('float64')

df = pd.concat([df_cmoa, df_met, df_tate, df_nga, df_moma], ignore_index = True)

from collections import defaultdict

def standard_blocking(field_values: pd.Series) -> dict[str, list]:

    blocks = defaultdict(list)
    for idx, key in enumerate(field_values):
        if key is not None:
            blocks[key].append(idx)

    return blocks

sb_artist = standard_blocking(df.full_name)


import itertools
from scipy.sparse import csr_matrix
```

```
    return [
        pair
        for indices in blocks.values()
        for pair in list(itertools.combinations(indices, 2))
    ]


def get_adjacency_matrix_from_pairs(
    pairs: list[list], matrix_shape: tuple[int, int]
) -> csr_matrix:

    idx1 = [pair[0] for pair in pairs]
    idx2 = [pair[1] for pair in pairs]
    ones = np.ones(len(idx1))

    return csr_matrix(
        (ones, (idx1, idx2)), shape=matrix_shape, dtype=np.int8
    )

adj_matrix_list = []
for blocks in [sb_artist]:
    pairs = get_pairs_from_blocks(blocks)
    adj_matrix_list.append(
        get_adjacency_matrix_from_pairs(pairs, (len(df), len(df)))
    )

def get_pairs_from_adj_matrix(adjacency_matrix: csr_matrix) -> np.ndarray:
    return np.array(adjacency_matrix.nonzero()).T


def get_union_of_adj_matrices(adj_matrix_list: list) -> csr_matrix:

    adj_matrix = csr_matrix(adj_matrix_list[0].shape)
    for matrix in adj_matrix_list:
        adj_matrix += matrix

    return adj_matrix

adj_matrix_union = get_union_of_adj_matrices(adj_matrix_list)
sb_pairs = get_pairs_from_adj_matrix(adj_matrix_union)


def sorted_neighborhood(
    df: pd.DataFrame, keys: list, window_size: int = 30
) -> np.ndarray:

    sorted_indices = (
        df[keys].dropna(how="all").sort_values(keys).index.tolist()
    )
    pairs = []
    for window_end in range(1, len(sorted_indices)):
        window_start = max(0, window_end - window_size)
        for i in range(window_start, window_end):
            pairs.append([sorted_indices[i], sorted_indices[window_end]])

    return np.array(pairs)

columns = ['full_name', 'ulan_id', 'nationality']
sn_pairs = sorted_neighborhood(df, columns)

block = pd.DataFrame(
    [
        ["Standard Blocking", f"{len(sb_pairs):,}"],
    ],
    columns=["Blocking Approach", "N of Candidate Pairs"]
)

from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
```

```
) -> dict[str, np.ndarray]:
    """
    Measuring similarity by field. It is either cosine similarity
    (if sim_type == 'fuzzy') or exact match 0/1 (if sim_type == 'exact').
    Attribute's similarity scores are stored in field_score dictionary
    with the field name as key.
    """

    field_scores = {}

    for field, sim_type in field_config.items():
        if sim_type == "fuzzy":
            field_scores[field] = cosine_similarities(
                df[field].fillna(""), pairs
            )
        else:
            field_scores[field] = exact_matches(df[field], pairs)

    return field_scores

def cosine_similarities(
    field_values: pd.Series, pairs: np.ndarray
) -> np.ndarray:
    """
    Computing cosine similarities on pairs
    """

    token_matrix_1, token_matrix_2 = get_token_matrix_pair(
        field_values, pairs
    )
    cos_sim = cosine_similarities_on_pair_matrices(
        token_matrix_1, token_matrix_2
    )

    return cos_sim

def get_token_matrix_pair(
    field_values: pd.Series, pairs: np.ndarray,
) -> tuple[csr_matrix, csr_matrix]:
    """
    Converting pairs into matrices of token counts (matrix of records
    by tokens filled with token counts).
    """

    all_idx = np.unique(pairs)
    vectorizer = CountVectorizer(analyzer="char", ngram_range=(3, 3))
    vectorizer.fit(field_values.loc[all_idx])
    token_matrix_1 = vectorizer.transform(field_values.loc[pairs[:, 0]])
    token_matrix_2 = vectorizer.transform(field_values.loc[pairs[:, 1]])

    return token_matrix_1, token_matrix_2

def cosine_similarities_on_pair_matrices(
    token_matrix_1: csr_matrix, token_matrix_2: csr_matrix
) -> np.ndarray:
    """
    Computing cosine similarities on pair of token count matrices.
    It normalizes each record (axis=1) first, then computes dot product
    for each pair of records.
    """

    token_matrix_1 = normalize(token_matrix_1, axis=1)
    token_matrix_2 = normalize(token_matrix_2, axis=1)
    cos_sim = np.asarray(
        token_matrix_1.multiply(token_matrix_2).sum(axis=1)
    ).flatten()

    return cos_sim

def exact_matches(
    field_values: pd.Series, pairs: np.ndarray
```

```
    Performing exact matches on pairs
    """

    arr1 = field_values.loc[pairs[:, 0]].values
    arr2 = field_values.loc[pairs[:, 1]].values

    return ((arr1 == arr2) & (~pd.isna(arr1)) & (~pd.isna(arr2))).astype(int)


field_config = {
    # <field>: <sim_type>
    "full_name": "fuzzy",

}
field_scores_sb = get_field_similarity_scores(df, sb_pairs, field_config)


print(field_scores_sb);


def calc_overall_scores(field_scores: dict[str, np.ndarray]) -> np.ndarray:
    return np.array(list(field_scores.values())).mean(axis=0)

def find_matches(scores: np.ndarray, threshold: float) -> np.ndarray:
    return scores >= threshold


scores_sb = calc_overall_scores(field_scores_sb)
is_matched_sb = find_matches(scores_sb, threshold=0.64)

field_scores_sn = get_field_similarity_scores(df, sn_pairs, field_config)
scores_sn = calc_overall_scores(field_scores_sn)
is_matched_sn = find_matches(scores_sn, threshold=0.64)
print(field_scores_sn)


matched_pairs = sb_pairs[is_matched_sb]
matched_scores = scores_sb[is_matched_sb]


matched_pairs = sn_pairs[is_matched_sn]
matched_scores = scores_sn[is_matched_sn]



from scipy.sparse.csgraph import connected_components

def connected_components_from_pairs(
    pairs: np.ndarray, dim: int
) -> np.ndarray:

    adjacency_matrix = get_adjacency_matrix_from_pairs(pairs, (dim, dim))
    _, clusters = connected_components(
        csgraph=adjacency_matrix, directed=False, return_labels=True
    )

    return clusters
cc_clusters = connected_components_from_pairs(matched_pairs, len(df))

print(cc_clusters);

def sort_pairs(pairs: np.ndarray, scores: np.ndarray) -> np.ndarray:
    sorted_ids = (-1 * scores).argsort()
    return pairs[sorted_ids]

pairs_sored = sort_pairs(matched_pairs, matched_scores)

print(pairs_sored);


def get_center_cluster_pairs(pairs, dim):
```

```
    cluster_centers:
        list tracking cluster center for each record.
        indices of the list correspond to the original df indices
        and the values represent assigned cluster centers' indices
    center_cluster_pairs:
        list of pairs of indices representing center-child pairs
    merge_cluster_pairs:
        list of pairs of merged nodes' indices
    """
    cluster_centers = [None] * dim
    center_cluster_pairs = []
    merge_cluster_pairs = []

    for idx1, idx2 in pairs:

        if (
            cluster_centers[idx1] is None
            or cluster_centers[idx1] == idx1
            or cluster_centers[idx2] is None
            or cluster_centers[idx2] == idx2
        ):
            # if both aren't child, those nodes are merged
            merge_cluster_pairs.append([idx1, idx2])

        if cluster_centers[idx1] is None and cluster_centers[idx2] is None:
            # if both weren't seen before, idx1 becomes center and idx2 gets child
            cluster_centers[idx1] = idx1
            cluster_centers[idx2] = idx1
            center_cluster_pairs.append([idx1, idx2])
        elif cluster_centers[idx2] is None:
            if cluster_centers[idx1] == idx1:
                # if idx1 is center, idx2 is assigned to that cluster
                cluster_centers[idx2] = idx1
                center_cluster_pairs.append([idx1, idx2])
            else:
                # if idx1 is not center, idx2 becomes new center
                cluster_centers[idx2] = idx2
        elif cluster_centers[idx1] is None:
            if cluster_centers[idx2] == idx2:
                # if idx2 is center, idx1 is assigned to that cluster
                cluster_centers[idx1] = idx2
                center_cluster_pairs.append([idx1, idx2])
            else:
                # if idx2 is not center, idx1 becomes new center
                cluster_centers[idx1] = idx1

    return center_cluster_pairs, merge_cluster_pairs

from sklearn.metrics.cluster import rand_score, adjusted_rand_score
from IPython.display import display

def get_stats(labels, clusters):

    stats = []
    stats.append(f"{rand_score(labels, clusters):.3f}")
    stats.append(f"{adjusted_rand_score(labels, clusters):.3f}")
    clus_dist = pd.Series(clusters).value_counts()
    stats.append(f"{len(clus_dist):,}")
    stats.append(f"{clus_dist.mean():.3f}")
    stats.append(f"{clus_dist.min():,}")
    stats.append(f"{clus_dist.max():,}")

    return stats


def compare_clusters(cluster_list, cluster_names, labels):

    stats_dict = {}
    for clusters, name in zip(cluster_list, cluster_names):
        stats = get_stats(labels, clusters)
        stats_dict[name] = stats
```

cs520_museum Projects Branches Notebook Dataset Caveats Settings

```
        pd.DataFrame(
            stats_dict,
            index=[
                "Rand Index",
                "Adjusted Rand Index",
                "Cluster Count",
                "Mean Cluster Size",
                "Min Cluster Size",
                "Max Cluster Size",
            ],
        )
    )
center_cluster_pairs, merge_cluster_pairs = get_center_cluster_pairs(pairs_sored, len(df))
ct_clusters = connected_components_from_pairs(center_cluster_pairs, len(df))
mc_clusters = connected_components_from_pairs(merge_cluster_pairs, len(df))

print(mc_clusters)
print(ct_clusters)
print(len(mc_clusters));
print(len(ct_clusters));
clus_dist = pd.Series(mc_clusters).value_counts()
print(len(clus_dist));
print(clus_dist.mean());
print(clus_dist.min());
print(clus_dist.max());



df['cluster_id'] = ct_clusters;
df = df.astype(str)

vizierdb.save_data_frame('clustered', df);
```

Console Timing Datasets Charts

```
{'full_name': array([1., 1., 1., ..., 1., 1., 1.])}
{'full_name': array([0.20044593, 0.0521286 , 0.05572782, ..., 0.        , 0.        ,
       0.        ])}

[    0     1     2 ... 35524 35525 35526]
[[ 2386 49169]
 [ 7584 41879]
 [ 7584 30567]
 ...
 [ 3143 10057]
 [ 8524 22877]
 [22877 10057]]

[    0     1     2 ... 35548 35549 35550]
[    0     1     2 ... 39042 39043 39044]
50036
50036
35551
1.4074428286124159
1
105
```

[17]

```
select * from clustered order by cluster_id asc;
```

Console Timing Datasets Charts

temporary_dataset (**50036** rows)

Views

1	Robert Mapplethorpe	nan	1946.0	1989.0	nan	500090430.0	nga	nan	5011
2	Robert Mapplethorpe	nan	1946	1989	American	500090430.0	moma	male	3343
3	Jo Baer	Baer, Jo	1929	None	American	nan	cmoa	nan	1
4	Jo Baer	nan	1929.0	nan	nan	500009657.0	nga	nan	6952
5	Watcombe Pottery	Watcombe Pottery	1867	None	None	nan	cmoa	nan	10
6	Robert Lekawa	Lekawa, Robert	1933	None	American	nan	cmoa	nan	100
7	Augustus Charles Pugin	Pugin, A. C.	1769	1832	French	nan	cmoa	nan	1004
8	Augustus Charles Pugin	nan	1762.0	1832.0	nan	nan	nga	nan	3816
9	Briton Riviere	nan	1840	1920	British	500020920.0	met	nan	4947
10	John Montgomery Flagg	nan	1877	1960	American	500003051.0	met	nan	4948
11	John Montgomery Grove	nan	1847.0	1947.0	nan	nan	nga	nan	11689
12	Sir Bernard Partridge	nan	1861	1945	British	500019341.0	met	nan	4950
13	Harrison Fisher	nan	1877	1934	American	500032777.0	met	nan	4951
14	Harrison Fisher	nan	1875	1934	American	nan	moma	male	13909
15	Leonard Raven-Hill	nan	1867	1942	British	500003628.0	met	nan	4952
16	Wilhelm Hunt Diederich	nan	1884	1953	American, born Hungary	500019593.0	met	nan	4953
17	John Simon	nan	1675	1725	None	500051859.0	met	nan	4954
18	John Simon	nan	1675.0	1751.0	nan	500051859.0	nga	nan	9444
19	Leonard Knyff	nan	1650	1722	Dutch	500011895.0	met	nan	4955

[18]

```
import pandas as pd
# Assuming you have a final_merged_data dataframe with artist_id, name, birth_date, death_date, and nationality columns
final_merged_data = vizierdb.get_data_frame('clustered')

# Convert date columns to datetime objects
final_merged_data['birth_year'] = pd.to_numeric(final_merged_data['birth_year'], errors='coerce')
final_merged_data['death_year'] = pd.to_numeric(final_merged_data['death_year'], errors='coerce')

print(final_merged_data['birth_year'])

def mode_without_empty_and_nan(x):
    # Filter out empty or NaN values before calculating the mode
    valid_values = x.dropna()
    return valid_values.mode().iloc[0] if not valid_values.empty else np.nan


# Define a function to choose nationality based on domain knowledge
def choose_nationality(nationalities):
    # Remove empty strings and NaN values
    nationalities = nationalities.dropna().replace('', pd.NA)

    # Add additional transformations based on domain knowledge
    # Example 1: If 'American' appears in any dataset, prioritize it over other nationalities
    if 'American' in nationalities.values:
        return 'American'

    # Example 2: If 'British' appears in any dataset, prioritize it over other nationalities
    if 'British' in nationalities.values:
        return 'British'

    # Return the most common nationality if no specific rules apply
    return nationalities.mode().iloc[0]

# Group by artist_id and aggregate information
final_dataset = final_merged_data.groupby('cluster_id').agg({
    'full_name': mode_without_empty_and_nan,  # Choose the most common name
    'birth_year': 'median',  # Use the median birth date
    'death_year': 'median',  # Use the median death date
    'nationality': lambda x: choose_nationality(x),  # Custom function for handling nationality,
    'gender': mode_without_empty_and_nan
}).reset_index()
```

cs520_museum Projects Branches Notebook Dataset Caveats Settings

```
# Handle any additional transformations based on domain or statistical knowledge
# Example: You can add conditions to choose certain values over others based on specific criteria
vizierdb.save_data_frame('aggregated', final_merged_data);
```

Console Timing Datasets Charts

```
0        1947.0
1        1929.0
2        1963.0
3           NaN
4        1936.0
          ...
50031    1953.0
50032    1942.0
50033    1971.0
50034       0.0
50035    1919.0
Name: birth_year, Length: 50036, dtype: float64
```

[19]

```
select * from aggregated;
```

Console Timing Datasets Charts

temporary_dataset (**50036** rows)

Views

	full_name (string)	cited_name (string)	birth_year (real)	death_year (real)	nationality (string)	ulan_id (string)	source (string)	gender
0	Robert Mapplethorpe	Mapplethorpe, Robert	1947	1989	American	nan	cmoa	nan
1	Jo Baer	Baer, Jo	1929		American	nan	cmoa	nan
2	Franz Ackermann	Ackermann, Franz	1963		German	nan	cmoa	nan
3	unknown Chinese	unknown Chinese			Chinese	nan	cmoa	nan
4	Cesare Casati	Casati, Cesare	1936		Italian	nan	cmoa	nan
5	Emanuele Ponzio	Ponzio, Emanuele	1923		Italian	nan	cmoa	nan
6	Augustus Welby Northmore Pugin	Pugin, Augustus Welby Northmore	1812	1852	British	nan	cmoa	nan
7	John Hardman & Co.	Hardman, John & Co.	1838		Scottish	nan	cmoa	nan
8	Alfred Ivory	Ivory, Alfred	1856	1882	British	nan	cmoa	nan
9	Christopher Dresser	Dresser, Christopher	1834	1904	British	nan	cmoa	nan
10	Watcombe Pottery	Watcombe Pottery	1867		None	nan	cmoa	nan
11	W. Brownfield & Sons	Brownfield, W. & Sons	1871	1891	None	nan	cmoa	nan
12	Old Hall Earthenware Company Limited	Old Hall Earthenware Company Limited	1861	1886	British	nan	cmoa	nan
13	Ault Pottery	Ault Pottery	1887		None	nan	cmoa	nan
14	Eugène Atget	Atget, Eugène	1857	1927	French	nan	cmoa	nan
15	unknown American	unknown American			American	nan	cmoa	nan
16	Charles Aldridge	Aldridge, Charles			English	nan	cmoa	nan
17	Henry Green	Green, Henry			English	nan	cmoa	nan
18	unknown English	unknown English			English	nan	cmoa	nan
19	Alex Katz	Katz, Alex	1927		American	nan	cmoa	nan

[20]

```
from __future__ import print_function
from six.moves import zip, range
import pandas as pd
import jellyfish
from collections import OrderedDict
```

```
df_nsf_awards.head()

df_ucpay = vizierdb.get_data_frame('aggregated');

# Look at what the UC data contains
df_ucpay.head()

# List of columns to keep
selected_columns = ['full_name','cited_name', 'birth_year', 'death_year', 'nationality', 'gender']

# Subset data frame to only include those columns
df_ucpay = df_ucpay[selected_columns]

# Get all of the values in the "name" column in the df_ucpay dataframe
names_ucpay = df_ucpay.full_name.values
print(names_ucpay)


# Take the first name from the UC dataset
test_name = names_ucpay[0]

# Make UC names lower case
test_name = test_name.lower()

# Split UC names on comma
test_name = test_name.split(',')

print(test_name)

def split_names(name):
    """
    Splits names fields into first, middle and last names
    and return lower case values.

    Parameters
    ----------
    name: str
        e.g. SHAPIRO, JORDAN ISAAC

    Returns
    -------
    (first, middle, last): str
        e.g. (jordan, isaac, shapiro)
    """

    # make the name lower case
    name=name.lower()

    # split the name into two fields at the comma
    ls_name = name.split(' ')
    first_name = ''
    middle_name = ''
    last_name = ''
    if(len(ls_name) > 1):
        # Since the last name came first, we save the first entry as the last name
        # and second entry as the first name
        last_name = ls_name[-1]
        first_name = ls_name[0];
        middle_name = ' ';

    #split by space to get the first and middle name
    if len(ls_name) > 2:
        first_name = ls_name[0]
        middle_name = ls_name[1]
        last_name = ls_name[2];
    elif len(ls_name) == 2:
        first_name = ls_name[0]
        middle_name = ''
        last_name = ls_name[1]
    return str(first_name.strip()), str(middle_name.strip()), str(last_name.strip())
```

```
ls_first, ls_middle, ls_last = zip(*ls_cleaned_names)


# Put colums in the UC dataset for first, middle, and last name
df_ucpay['first'] = ls_first
df_ucpay['middle'] = ls_middle
df_ucpay['last'] = ls_last

df_nsf_awards.dropna(subset=['Artist_Display_Name'], inplace=True)
names_nsf_awards = df_nsf_awards.Artist_Display_Name.values
ls_cleaned_names = [split_names(name) for name in names_nsf_awards]

ls_first, ls_middle, ls_last = zip(*ls_cleaned_names)

df_nsf_awards['FirstName'] = ls_first
df_nsf_awards['LastName'] = ls_last
df_nsf_awards['first'] = [str(name.lower()) for name in df_nsf_awards['FirstName'].values]
df_nsf_awards['last'] = [str(name.lower()) for name in df_nsf_awards['LastName'].values]

class StringComparators():
    """
    Test various string comparators
    """

    def test_levenshtein_distance():
        assert jellyfish.levenshtein_distance('John', 'John') == 0
        assert jellyfish.levenshtein_distance('Jon', "John") == 2
        assert jellyfish.levenshtein_distance('Joseph', 'Joesph') == 1

    def test_damerau_levenshtein():
        assert jellyfish.damerau_levenshtein_distance('Joseph', 'Joesph') == 1

    def test_jaro_winklear():
        assert (np.isclose(jellyfish.jaro_winkler('Joseph', 'Joesph'), 0.955555))
        assert (np.isclose(jellyfish.jaro_winkler('Chris', 'Christoper'), 0.9))

# Get all of the unique names from NSF and UC
nsf_firstnames = set( df_nsf_awards['first'].values )

# grab the uc_names
uc_firstnames = df_ucpay['first'].values



# Comparison of records
testname = str(uc_firstnames[0])



# we should document this better and uc_names an argument
def get_matching_first_name(testname, NUM_NAMES=10):
    """
    get top 10 first names that match
    """
    dict_name_pair = {}
    for name in uc_firstnames:
        name = str(name)
        dist = jellyfish.jaro_winkler_similarity(testname,name)
        dict_name_pair[name] = dist

    orddict_dict_name_pair = OrderedDict(
                                sorted(dict_name_pair.items(), key=lambda x: x[1]))

    ls_sorted_name = list(orddict_dict_name_pair.keys())


    return ls_sorted_name[-NUM_NAMES:][::-1]


print(testname, get_matching_first_name(testname))
```

```
    testname = str(nm)
    print(testname, get_matching_first_name(testname))


dict_nsf_awards = df_nsf_awards.to_dict(orient='index')



def create_rule_mask(nsf_first_name,
                     nsf_last_name,
                     df_ucpay,
                     first_name_thresh=0.90,
                     last_name_thresh=0.90):
    """
    Returns a boolean array of records to match based on a
    fixed threshold.

    Parameters
    ----------
    (nsf_first_name, nsf_last_name): str
        first and last name in the NSF dataset

    df_ucpay: DataFrame
        DataFrame of the UC directory

    (first_name_thresh,last_name_thresh): int


    Returns
    -------
    jaro_mask: ls[bool]
        boolean list of records to match
    """
    compare_first = lambda x: jellyfish.jaro_winkler_similarity(nsf_first_name,x)
    compare_last = lambda x: jellyfish.jaro_winkler_similarity(nsf_last_name,x)

    jaro_first = df_ucpay['first'].map(compare_first)
    jaro_last = df_ucpay['last'].map(compare_last)

    jaro_mask = (jaro_first > first_name_thresh) & (jaro_last > last_name_thresh)

    return jaro_mask


def match_records(dict_nsf_awards, df_ucpay, f_create_rule_mask):
    """
    match records from the nsf and uc datasets based on the fields 'first' and 'last' name

    Parameters
    ---------
    dict_nsf_awards: dict
        dictionary of nsf awards
    df_ucpay: DataFrame
        DataFrame of UC employees
    create_rule_mask: function
        Function that takes a first name, last name and df_ucpay
        and returns a Boolean array of whether or not to match
        records

    Returns
    -------
    df_linked_data: DataFrame
    """

    df_linked_data = pd.DataFrame()
    for key in dict_nsf_awards.keys():
        dict_test_row = dict_nsf_awards[key]

        nsf_first_name = dict_test_row['first']
        nsf_last_name = dict_test_row['last']
```

cs520_museum Projects Branches Notebook Dataset Caveats Settings

```
        df_matches = df_ucpay[jaro_mask]
        for row in df_matches.iterrows():
            dict_test_row['full_name'] = row[1]['full_name']
            dict_test_row['nationality'] = row[1]['nationality']
            dict_test_row['birth_year'] = row[1]['birth_year']
            dict_test_row['death_year'] = row[1]['death_year']
            dict_test_row['gender'] = row[1]['gender']
            df_dictionary = pd.DataFrame([dict_test_row])
            df_linked_data = pd.concat([df_linked_data, df_dictionary], ignore_index=True)

    return df_linked_data


df_linked_data = match_records(dict_nsf_awards, df_ucpay, create_rule_mask )

# replace np nan or none with empty string
df_linked_data.nationality = df_linked_data.nationality.fillna('')
import numpy as np
df_linked_data['nationality'] = df_linked_data['nationality'].replace(np.nan, '', regex=True)
vizierdb.save_data_frame('df_linked_data', df_linked_data);
```

Console Timing Datasets Charts

```
['Robert Mapplethorpe' 'Jo Baer' 'Franz Ackermann' ... 'Arnt Jensen'
 'After Sophie Taeuber-Arp' 'William Theophilius Brown']

['robert mapplethorpe']
robert ['robert', 'roberta', 'roberts', 'roberto', 'robbert', 'robertus', 'roberts,', 'robertson,', 'h-robert', 'norbert'
robert ['robert', 'roberta', 'roberts', 'roberto', 'robbert', 'robertus', 'roberts,', 'robertson,', 'h-robert', 'norbert'
jo ['jo', 'joy', 'jos', 'joe', 'jon', 'joão', 'johs', 'jock', 'joop', 'jodi']
franz ['franz', 'frantz', 'fran', 'frans', 'frank', 'franziska', 'fra', 'franciszka', 'franciszek', 'franjo']
unknown ['unknown', 'unknown,', 'unkn', 'unno', 'unkoku', 'un', 'ung-no', 'ninon', 'nuno', 'union']
cesare ['cesare', 'cesar', 'clear', 'carme', 'carle', 'césar', 'clare', 'case', 'earle', 'cestmir']
emanuele ['emanuele', 'emanuel', 'emanuel,', 'emmanuel', 'manuel', 'manuel,', 'emmanuil', 'manu', 'emmanuel-joseph-raphae
augustus ['augustus', 'august', 'auguste-louis', 'augusto', 'augusta', 'auguste', 'augustin', 'auguste-jules-marie', 'aug
john ['john', 'john,', 'johan', 'johns,', 'johnny', 'jon', '"john', 'johann', 'johnson', 'johnnie']
alfred ['alfred', 'alfredo', 'alfred-andre', 'alfred-henri', 'alfred-louis', 'alfred-ernest', 'fred', 'alfred-nicolas', '
christopher ['christopher', 'christophe', 'christoph', 'christer', 'christoffer', 'christofle', 'christof', 'christoffel'
watcombe ['watcombe', 'whitcombe,', 'atco', 'watanabe', 'tom', 'wobbe', 'newcomb', 'balcomb', 'walton', 'walter']
w. ['w.', 'w.r.', 'w.c.', 'w.w.', 'w.j.', 'w.t.', 'w.h.', 'w.h.l.', 'z.', 'q.']
old ['old', 'gold', 'oldřich', 'oldrich', 'oldenburg,', 'soldi,', 'nolde,', 'holden', 'oli', 'ole']
ault ['ault', 'al', 'paulette', 'raul', 'walt', 'saul', 'paul', 'gaultney,', 'alton', 'austin']
eugène ['eugène', 'eugêne', 'eugen', 'eugène-louis', 'eugène-andré', 'eugène-joseph', 'eugène-samuel', 'eugène-emanuel',
unknown ['unknown', 'unknown,', 'unkn', 'unno', 'unkoku', 'un', 'ung-no', 'ninon', 'nuno', 'union']
charles ['charles', 'charles,', '"charles', 'charlie', 'charlee', 'charline', 'charlene', 'charles-ange', 'charles-paul',
henry ['henry', 'henryk', 'henry,', 'henri', 'henny', 'henri,', 'henrik', 'herry', 'henry-louis', 'henry-martin']
unknown ['unknown', 'unknown,', 'unkn', 'unno', 'unkoku', 'un', 'ung-no', 'ninon', 'nuno', 'union']
alex ['alex', 'alexa', 'alex.', 'alexei', 'alexey', 'alexej', 'alexis', 'lex', 'alexina', 'alexandr']
clive ['clive', 'cleve', 'clover', 'claire', 'carlie', 'clarice', 'oliver', 'cliff', 'claudine', 'collier']
heal ['heal', 'health', 'hal', 'head', 'hell', 'haël', 'ha', 'hall', 'hale', 'heath']
henry ['henry', 'henryk', 'henry,', 'henri', 'henny', 'henri,', 'henrik', 'herry', 'henry-louis', 'henry-martin']
andré ['andré', 'andrés', 'andrée', 'andr.', 'andre', 'andro', 'andrás', 'andres', 'andrej', 'andre,']
vassily ['vassily', 'vasily', 'wassily', 'vasili', 'vasilii', 'vasyl', 'vasilios', 'vally', 'vasarely,', "vasyl'"]
```

[21]

```
import pandas as pd
import numpy as np
df_tate = vizierdb.get_data_frame('df_linked_data');
print(get_null_info(df_tate));
```

Console Timing Datasets Charts

```
{'Object_Number': (0, 0.0), 'Object_ID': (0, 0.0), 'Department': (0, 0.0), 'AccessionYear': (168, 0.004882443546746491),
```

cs520_museum Projects Branches Notebook Dataset Caveats Settings

Console Timing Datasets Charts

temporary_dataset (**34409** rows)

Views

	Object_Number (string)	Object_ID (long)	Department (string)	AccessionYear (date)	Object_Name (string)	Title (string)
0	63.664.5	375137	Drawings and Prints	1963-00-01	Print	Forest Park
1	63.664.5	375137	Drawings and Prints	1963-00-01	Print	Forest Park
2	63.664.5	375137	Drawings and Prints	1963-00-01	Print	Forest Park
3	95.27.2	436631	European Paintings	1895-00-01	Painting	Self-Portrait
4	95.27.2	436631	European Paintings	1895-00-01	Painting	Self-Portrait
5	63.350.201.23.39	409698	Drawings and Prints	1963-00-01	Print	Rose-breasted Wood Robin, from the Song Birds of the World series
6	33.43.237	267646	Photographs	1933-00-01	Photograph	Children - Nude
7	33.43.237	267646	Photographs	1933-00-01	Photograph	Children - Nude
8	33.43.237	267646	Photographs	1933-00-01	Photograph	Children - Nude
9	66.543.19	392586	Drawings and Prints	1966-00-01	Print	De Arte Pictoria
10	66.543.19	392586	Drawings and Prints	1966-00-01	Print	De Arte Pictoria
11	28.31.15	349863	Drawings and Prints	1928-00-01	Print	Apples
12	28.31.15	349863	Drawings and Prints	1928-00-01	Print	Apples
13	2009.300.2181	156978	Costume Institute	2009-00-01	Toque	Toque
14	2021.386.47	856086	Photographs	2021-00-01	Photograph	Untitled
15	67.803.15	363833	Drawings and Prints	1967-00-01	Drawing	Floral Tailpiece Designs, The Duchess of Malfi and The White Devil
16	67.803.15	363833	Drawings and Prints	1967-00-01	Drawing	Floral Tailpiece Designs, The Duchess of Malfi and The White Devil
17	51.551	350651	Drawings and Prints	1951-00-01	Book	The Multiple Badge. Catalogue No. 137B
18	51.551	350651	Drawings and Prints	1951-00-01	Book	The Multiple Badge. Catalogue No. 137B
19	1994.258.206	275633	Photographs	1994-00-01	Negative	[Plantation House, Louisiana]

[23]
```
CREATE LENS ON df_linked_data IMPUTE MISSING VALUES ON COLUMN 17
```

Console Timing Datasets Charts

```
Created Impute Missing Values Lens on df_linked_data
```

[24]
```
select * from df_linked_data;
```

Console Timing Datasets Charts

temporary_dataset (**34409** rows)

Views

	Object_Number (string)	Object_ID (long)	Department (string)	AccessionYear (date)	Object_Name (string)	Title (string)
0	63.664.5	375137	Drawings and Prints	1963-00-01	Print	Forest Park
1	63.664.5	375137	Drawings and Prints	1963-00-01	Print	Forest Park
2	63.664.5	375137	Drawings and Prints	1963-00-01	Print	Forest Park
3	95.27.2	436631	European Paintings	1895-00-01	Painting	Self-Portrait
4	95.27.2	436631	European Paintings	1895-00-01	Painting	Self-Portrait
5	63.350.201.23.39	409698	Drawings and Prints	1963-00-01	Print	Rose-breasted Wood Robin, from the Song Birds of the World series
6	33.43.237	267646	Photographs	1933-00-01	Photograph	Children - Nude
7	33.43.237	267646	Photographs	1933-00-01	Photograph	Children - Nude
8	33.43.237	267646	Photographs	1933-00-01	Photograph	Children - Nude
9	66.543.19	392586	Drawings and Prints	1966-00-01	Print	De Arte Pictoria

cs520_museum Projects Branches Notebook Dataset Caveats Settings

11	28.31.15	349863	Drawings and Prints	1928-00-01	Print	Apples
12	28.31.15	349863	Drawings and Prints	1928-00-01	Print	Apples
13	2009.300.2181	156978	Costume Institute	2009-00-01	Toque	Toque
14	2021.386.47	856086	Photographs	2021-00-01	Photograph	Untitled
15	67.803.15	363833	Drawings and Prints	1967-00-01	Drawing	Floral Tailpiece Designs, The Duchess of Malfi and The White Devil
16	67.803.15	363833	Drawings and Prints	1967-00-01	Drawing	Floral Tailpiece Designs, The Duchess of Malfi and The White Devil
17	51.551	350651	Drawings and Prints	1951-00-01	Book	The Multiple Badge. Catalogue No. 137B
18	51.551	350651	Drawings and Prints	1951-00-01	Book	The Multiple Badge. Catalogue No. 137B
19	1994.258.206	275633	Photographs	1994-00-01	Negative	[Plantation House, Louisiana]

[25]

```
select SUM(CASE when gender='male' then 1 else 0 end) male_artists , SUM(CASE when gender='female' then 1 else 0 end) fema
```

Console Timing Datasets Charts

ex_data_view (**10** rows)

Views

	male_artists (long)	female_artists (long)	nationality (string)
0	2605	239	American
1	567	8	French
2	261	52	British
3	159	6	German
4	125	1	Japanese
5	63	0	Mexican
6	40	2	Dutch
7	29	3	None
8	28	2	Italian
9	26	0	Spanish

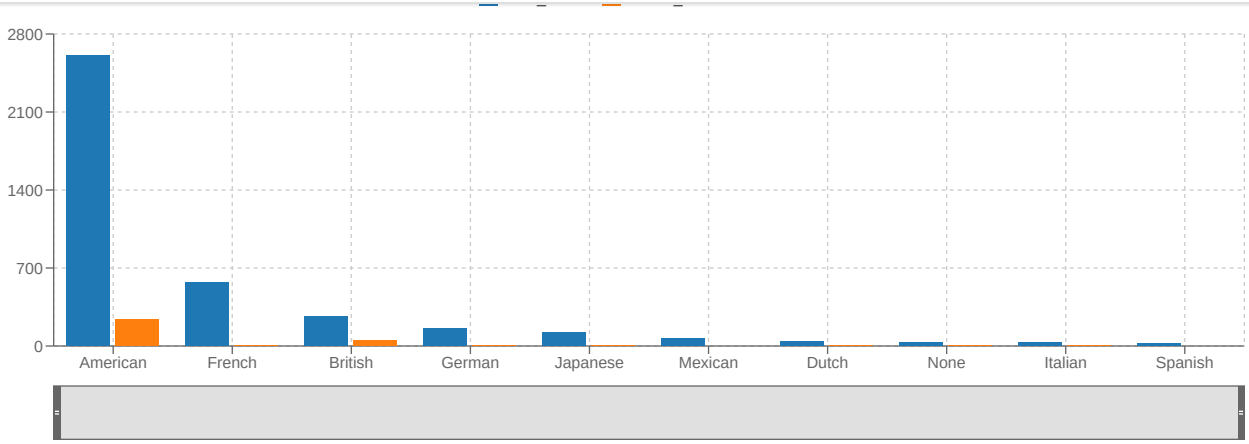
[26] CREATE Bar Chart null FOR "ex_data_view"

Console Timing Datasets Charts

ex_data_view_Visualized

Charts Bar Chart Grouped

cs520_museum Projects Branches Notebook Dataset Caveats Settings



Download Chart

Connected to vizier @ http://localhost:5000/vizier-db/api/v1/