V3_EDA

October 17, 2025

1 Comprehensive Exploratory Data Analysis (EDA) - Paris IRIS Datasets

1.1 Version 2.0 - Corrected and Enhanced

This notebook provides a thorough exploratory data analysis of socio-economic, demographic, real estate, and business establishment data for Paris at the IRIS level (2013-2024).

1.1.1 Datasets:

- FILOSOFI (2013, 2017, 2021): Income distribution at IRIS level
- CENSUS (2013, 2017, 2021): Population and socio-demographic data
- DVF (2014-2024): Real estate transactions
- SIRENE (2014-2024): Business establishments
- IRIS GeoJSON: Geographic boundaries

1.2 1. Setup and Configuration

[1]:
```
# Import libraries
import pandas as pd
import numpy as np
import geopandas as gpd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
import warnings
from datetime import datetime
from scipy import stats
from shapely.geometry import Point
from libpysal import weights
from esda.moran import Moran, Moran_Local

# Suppress warnings
warnings.filterwarnings('ignore')

# Configure pandas
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)
```

```python
pd.set_option('display.float_format', '{:.2f}'.format)
pd.set_option('display.precision', 2)

# Configure matplotlib/seaborn
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette('husl')
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 11

# Set random seed
np.random.seed(42)

# Define paths
DATA_DIR = Path('../datasets')
OUTPUT_DIR = Path('../outputs')
FIGURES_DIR = OUTPUT_DIR / 'figures' / 'eda_v2'
TABLES_DIR = OUTPUT_DIR / 'tables' / 'eda_v2'
REPORTS_DIR = OUTPUT_DIR / 'reports'

# Create output directories
FIGURES_DIR.mkdir(parents=True, exist_ok=True)
TABLES_DIR.mkdir(parents=True, exist_ok=True)
REPORTS_DIR.mkdir(parents=True, exist_ok=True)

# CRS definition
CRS_WGS84 = 'EPSG:4326'
CRS_LAMBERT93 = 'EPSG:2154'

print("  Environment configured")
print(f"  Data directory: {DATA_DIR}")
print(f"  Output directory: {OUTPUT_DIR}")
print(f"  Date: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
```

```
 Environment configured
 Data directory: ../datasets
 Output directory: ../outputs
 Date: 2025-10-17 13:58:18
```

```
/usr/local/python/3.12.1/lib/python3.12/site-packages/tqdm/auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

1.3 2. Data Loading

[2]:

```python
print("Loading datasets...\n")

# FILOSOFI
filosofi_2013 = pd.read_parquet(DATA_DIR / 'filosofi_2013_paris.parquet')
filosofi_2017 = pd.read_parquet(DATA_DIR / 'filosofi_2017_paris.parquet')
filosofi_2021 = pd.read_parquet(DATA_DIR / 'filosofi_2021_paris.parquet')
print(f"  FILOSOFI 2013: {filosofi_2013.shape}")
print(f"  FILOSOFI 2017: {filosofi_2017.shape}")
print(f"  FILOSOFI 2021: {filosofi_2021.shape}")

# CENSUS
census_2013 = pd.read_parquet(DATA_DIR / 'census_2013_paris.parquet')
census_2017 = pd.read_parquet(DATA_DIR / 'census_2017_paris.parquet')
census_2021 = pd.read_parquet(DATA_DIR / 'census_2021_paris.parquet')
print(f"\n  CENSUS 2013: {census_2013.shape}")
print(f"  CENSUS 2017: {census_2017.shape}")
print(f"  CENSUS 2021: {census_2021.shape}")

# DVF
dvf = pd.read_parquet(DATA_DIR / 'dvf_mutations_paris.parquet')
print(f"\n  DVF: {dvf.shape}")

# SIRENE
sirene = pd.read_parquet(DATA_DIR / 'sirene_2014_2024_paris.parquet')
print(f"\n  SIRENE: {sirene.shape}")

# IRIS boundaries (all 992 IRIS)
iris_geo = gpd.read_file(OUTPUT_DIR / 'iris_paris75.geojson')
print(f"\n  IRIS GeoJSON: {iris_geo.shape}")
print(f"   CRS: {iris_geo.crs}")

print("\n" + "="*80)
print("ALL DATASETS LOADED SUCCESSFULLY")
print("="*80)
```

```
Loading datasets…

  FILOSOFI 2013: (853, 10)
  FILOSOFI 2017: (871, 10)
  FILOSOFI 2021: (992, 10)

  CENSUS 2013: (992, 13)
  CENSUS 2017: (992, 13)
  CENSUS 2021: (992, 13)

  DVF: (457097, 20)
```

```
SIRENE: (1194896, 53)

IRIS GeoJSON: (992, 10)
 CRS: EPSG:4326

================================================================================
ALL DATASETS LOADED SUCCESSFULLY
================================================================================
```

[3]:
```python
# Create IRIS neighborhood grouping by removing trailing numbers
# Example: "Amérique 1", "Amérique 2" -> "Amérique"

def extract_iris_quartier(nom_iris):
    """
    Extract IRIS neighborhood name by removing trailing number.
    Handles both single and multi-digit numbers.
    """
    import re
    # Remove trailing space + digits (e.g., " 1", " 15", " 22")
    return re.sub(r'\s+\d+$', '', str(nom_iris))

# Add quartier column to iris_geo
iris_geo['quartier_iris'] = iris_geo['nom_iris'].apply(extract_iris_quartier)

print("IRIS neighborhood aggregation created:")
print(f"  Original IRIS: {iris_geo['nom_iris'].nunique()} unique")
print(f"  Quartiers: {iris_geo['quartier_iris'].nunique()} unique")
print(f"\nSample mapping:")
print(iris_geo[['nom_iris', 'quartier_iris']].head(10))

# Create a mapping from code_iris to quartier_iris for efficient merging
iris_mapping = iris_geo[['code_iris', 'quartier_iris']].drop_duplicates()
print(f"\n  Created mapping: {len(iris_mapping)} code_iris -> quartier_iris␣
↪entries")

# Create quartier-level geometry by dissolving IRIS geometries
iris_quartiers = iris_geo.dissolve(by='quartier_iris', as_index=False,␣
↪aggfunc='first')
iris_quartiers = iris_quartiers[['quartier_iris', 'geometry', 'dep',␣
↪'nom_com']].copy()
print(f"  Created quartier-level geometries: {len(iris_quartiers)} quartiers")
```

```
IRIS neighborhood aggregation created:
  Original IRIS: 973 unique
  Quartiers: 94 unique

Sample mapping:
```

```
             nom_iris      quartier_iris
0          Invalides 1          Invalides
1          Invalides 3          Invalides
2        Rochechouart 5       Rochechouart
3     Folie Méricourt 8    Folie Méricourt
4   Sainte-Marguerite 4  Sainte-Marguerite
5   Grandes Carrières 3  Grandes Carrières
6          Charonne 22           Charonne
7       Seine et Berges    Seine et Berges
8     École Militaire 5    École Militaire
9        Gros Caillou 4       Gros Caillou

 Created mapping: 992 code_iris -> quartier_iris entries
 Created quartier-level geometries: 94 quartiers
```

1.3.1 2.1 Convert FILOSOFI Data Types

FILOSOFI data uses French decimal format (comma separator). Convert to numeric before analysis.

[4]:
```
print("="*80)
print("FILOSOFI MISSING VALUES: SIMPLE DETECTION & IMPUTATION")
print("="*80)

# Define suppression codes to replace with NaN
suppression_codes = {'s', 'c', 'S', 'C', 'ns', 'nd', 'so'}

# Function to clean and convert columns
def clean_filosofi(df, year):
    print(f"\n--- FILOSOFI {year} ---")

    numeric_cols = ['median_uc', 'q1_uc', 'q3_uc', 'd9d1_ratio', 'gini',
                    'share_activity_income', 'share_pensions',␣
 ↪'share_social_benefits']

    for col in numeric_cols:
        if col in df.columns:
            # Replace suppression codes and special values with NaN
            mask = df[col].astype(str).isin(suppression_codes)
            df.loc[mask, col] = np.nan

            # Replace French decimal comma with period
            df[col] = df[col].astype(str).str.replace(',', '.')

            # Convert to numeric
            df[col] = pd.to_numeric(df[col], errors='coerce')

# Clean all FILOSOFI datasets
```

```python
for df_name, df in [('FILOSOFI 2013', filosofi_2013),
                    ('FILOSOFI 2017', filosofi_2017),
                    ('FILOSOFI 2021', filosofi_2021)]:
    clean_filosofi(df, df_name.split()[-1])

print("\n  Data type conversion complete")


# ==============================================================================
# FILOSOFI 2017: Simple missing value handling
# ==============================================================================
print("\n" + "="*80)
print("FILOSOFI 2017: MISSING VALUE ANALYSIS")
print("="*80)

filosofi_2017 = filosofi_2017.merge(iris_geo[['code_iris', 'typ_iris']],
                                    on='code_iris', how='left')

# Check for missing values
print(f"\nTotal IRIS: {len(filosofi_2017)}")
print(f"\nMissing values by column:")
missing_counts = filosofi_2017[['median_uc', 'q1_uc', 'q3_uc']].isna().sum()
for col, count in missing_counts.items():
    pct = count / len(filosofi_2017) * 100
    print(f"  {col:20s}: {count:4d} ({pct:5.1f}%)")

# Handle the single missing row for 2017 (code: 751145625, quartier: Plaisance␣
↪25)
if filosofi_2017['median_uc'].isna().any():
    print(f"\nMissing row(s):")
    missing_rows = filosofi_2017[filosofi_2017['median_uc'].isna()]
    print(missing_rows[['code_iris', 'median_uc', 'typ_iris']])
    print("\nStrategy: Fill with median of neighboring IRIS (same␣
↪arrondissement)")

    # Get arrondissement
    arr = missing_rows['code_iris'].iloc[0][:5]
    arr_data = filosofi_2017[filosofi_2017['code_iris'].str[:5] == arr]
    median_val = arr_data['median_uc'].median()
    q1_val = arr_data['q1_uc'].median()
    q3_val = arr_data['q3_uc'].median()

    print(f"  Arrondissement median_uc: {median_val:.0f}")
    filosofi_2017.loc[filosofi_2017['median_uc'].isna(), 'median_uc'] =␣
↪median_val
    filosofi_2017.loc[filosofi_2017['q1_uc'].isna(), 'q1_uc'] = q1_val
    filosofi_2017.loc[filosofi_2017['q3_uc'].isna(), 'q3_uc'] = q3_val
```

```python
    print(f"  Row imputed")

print(f"\nAfter imputation: {filosofi_2017['median_uc'].isna().sum()} missing␣
↪values")


# ============================================================================
# FILOSOFI 2021: Missing values by IRIS type
# ============================================================================
print("\n" + "="*80)
print("FILOSOFI 2021: MISSING VALUE ANALYSIS BY IRIS TYPE")
print("="*80)

filosofi_2021 = filosofi_2021.merge(iris_geo[['code_iris', 'typ_iris']],
                                     on='code_iris', how='left')

print(f"\nTotal IRIS: {len(filosofi_2021)}")
print(f"\nMissing values by column (total):")
missing_totals = filosofi_2021[['median_uc', 'q1_uc', 'q3_uc']].isna().sum()
for col, count in missing_totals.items():
    pct = count / len(filosofi_2021) * 100
    print(f"  {col:20s}: {count:4d} ({pct:5.1f}%)")

# Detailed breakdown by IRIS type (H, D, A)
print(f"\nMissing values by IRIS type:")
print("\n             H (Housing)  D (Diversified)  A (Activity)")
print("-" * 55)

for col in ['median_uc', 'q1_uc', 'q3_uc']:
    counts = {}
    for iris_type in ['H', 'D', 'A']:
        subset = filosofi_2021[filosofi_2021['typ_iris'] == iris_type]
        missing = subset[col].isna().sum()
        total = len(subset)
        pct = (missing / total * 100) if total > 0 else 0
        counts[iris_type] = f"{missing:3d}/{total:3d} ({pct:5.1f}%)"

    print(f"{col:15s} {counts.get('H', 'N/A'):20s} {counts.get('D', 'N/A'):20s}␣
↪{counts.get('A', 'N/A'):20s}")

# Imputation strategy for FILOSOFI 2021
print(f"\n\nIMPUTATION STRATEGY:")
print("  H (Housing IRIS): Use median of arrondissement")
print("  D (Diversified):  Leave as NaN (too heterogeneous)")
print("  A (Activity):     Leave as NaN (no residential income)")

# Impute H (Housing) IRIS
```

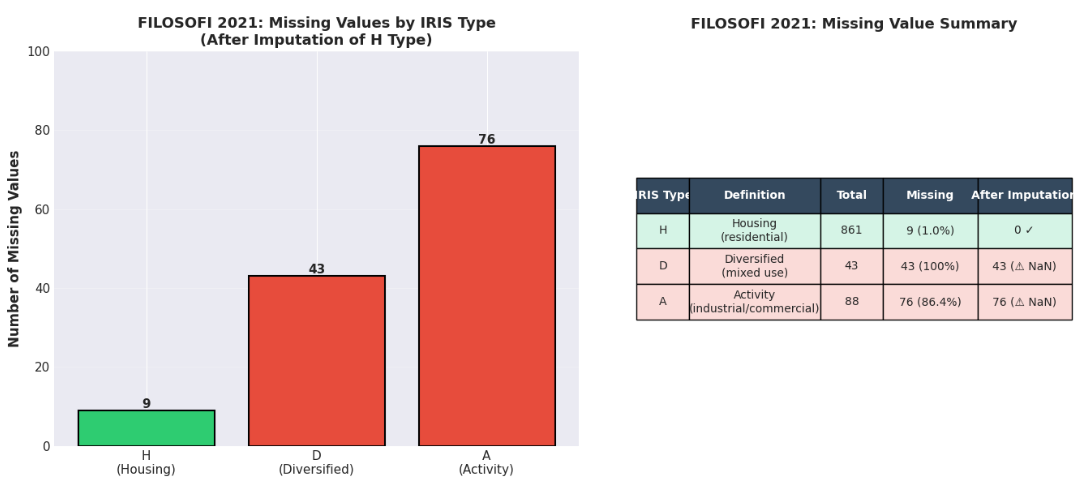
```
h_iris = filosofi_2021[filosofi_2021['typ_iris'] == 'H'].copy()
for idx in h_iris.index:
    if h_iris.loc[idx, 'median_uc'] is np.nan or pd.isna(h_iris.loc[idx,␣
 ↪'median_uc']):
        # Get arrondissement from code_iris (positions 3:5)
        arr = filosofi_2021.loc[idx, 'code_iris'][:5]
        arr_data = filosofi_2021[(filosofi_2021['code_iris'].str[:5] == arr) &
                                 (filosofi_2021['typ_iris'] == 'H')]

        # Fill with arrondissement median
        median_val = arr_data['median_uc'].median()
        q1_val = arr_data['q1_uc'].median()
        q3_val = arr_data['q3_uc'].median()

        if not pd.isna(median_val):
            filosofi_2021.loc[idx, 'median_uc'] = median_val
            filosofi_2021.loc[idx, 'q1_uc'] = q1_val
            filosofi_2021.loc[idx, 'q3_uc'] = q3_val

print(f"\n Imputation complete")
print(f"\nFinal missing values:")
print(f"  median_uc (H):␣
 ↪{filosofi_2021[(filosofi_2021['typ_iris']=='H')]['median_uc'].isna().sum()}␣
 ↪missing")
print(f"  median_uc (D):␣
 ↪{filosofi_2021[(filosofi_2021['typ_iris']=='D')]['median_uc'].isna().sum()}␣
 ↪missing")
print(f"  median_uc (A):␣
 ↪{filosofi_2021[(filosofi_2021['typ_iris']=='A')]['median_uc'].isna().sum()}␣
 ↪missing (expected)")
```

```
================================================================================
FILOSOFI MISSING VALUES: SIMPLE DETECTION & IMPUTATION
================================================================================

--- FILOSOFI 2013 ---

--- FILOSOFI 2017 ---

--- FILOSOFI 2021 ---

 Data type conversion complete

================================================================================
FILOSOFI 2017: MISSING VALUE ANALYSIS
================================================================================
```

```
Total IRIS: 871

Missing values by column:
  median_uc          :    1 (  0.1%)
  q1_uc              :    1 (  0.1%)
  q3_uc              :    1 (  0.1%)

Missing row(s):
     code_iris  median_uc typ_iris
434  751145625        NaN        A

Strategy: Fill with median of neighboring IRIS (same arrondissement)
  Arrondissement median_uc: 29010
 Row imputed

After imputation: 0 missing values

================================================================================
FILOSOFI 2021: MISSING VALUE ANALYSIS BY IRIS TYPE
================================================================================

Total IRIS: 992

Missing values by column (total):
  median_uc          :  128 ( 12.9%)
  q1_uc              :  128 ( 12.9%)
  q3_uc              :  128 ( 12.9%)

Missing values by IRIS type:

             H (Housing)  D (Diversified)  A (Activity)
-------------------------------------------------------
median_uc         9/861 (  1.0%)      43/ 43 (100.0%)      76/ 88 ( 86.4%)
q1_uc             9/861 (  1.0%)      43/ 43 (100.0%)      76/ 88 ( 86.4%)
q3_uc             9/861 (  1.0%)      43/ 43 (100.0%)      76/ 88 ( 86.4%)


IMPUTATION STRATEGY:
  H (Housing IRIS): Use median of arrondissement
  D (Diversified):  Leave as NaN (too heterogeneous)
  A (Activity):     Leave as NaN (no residential income)

 Imputation complete

Final missing values:
  median_uc (H): 0 missing
  median_uc (D): 43 missing
  median_uc (A): 76 missing (expected)
```

[5]:
```python
# ============================================================================
# VISUALIZE MISSING VALUES SUMMARY
# ============================================================================

import matplotlib.patches as mpatches

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Data for visualization
categories = ['H\n(Housing)', 'D\n(Diversified)', 'A\n(Activity)']
missing_2021 = [9, 43, 76]
total_2021 = [861, 43, 88]

# Plot 1: Missing value counts by type
ax1 = axes[0]
colors_missing = ['#2ecc71', '#e74c3c', '#e74c3c']  # Green for H, Red for D␣
↪and A
bars = ax1.bar(categories, missing_2021, color=colors_missing,␣
↪edgecolor='black', linewidth=1.5)
ax1.set_ylabel('Number of Missing Values', fontsize=12, fontweight='bold')
ax1.set_title('FILOSOFI 2021: Missing Values by IRIS Type\n(After Imputation of␣
↪H Type)',
              fontsize=13, fontweight='bold')
ax1.set_ylim(0, 100)
ax1.grid(True, alpha=0.3, axis='y')

# Add value labels on bars
for bar, missing in zip(bars, missing_2021):
    height = bar.get_height()
    ax1.text(bar.get_x() + bar.get_width()/2., height,
            f'{int(missing)}',
            ha='center', va='bottom', fontsize=11, fontweight='bold')

# Plot 2: Breakdown table as text
ax2 = axes[1]
ax2.axis('off')

table_data = [
    ['IRIS Type', 'Definition', 'Total', 'Missing', 'After Imputation'],
    ['H', 'Housing\n(residential)', '861', '9 (1.0%)', '0  '],
    ['D', 'Diversified\n(mixed use)', '43', '43 (100%)', '43 (  NaN)'],
    ['A', 'Activity\n(industrial/commercial)', '88', '76 (86.4%)', '76 (  NaN)']
]

table = ax2.table(cellText=table_data, cellLoc='center', loc='center',
                  colWidths=[0.1, 0.25, 0.12, 0.18, 0.18])
table.auto_set_font_size(False)
```

```python
table.set_fontsize(10)
table.scale(1, 2.5)

# Style header row
for i in range(5):
    table[(0, i)].set_facecolor('#34495e')
    table[(0, i)].set_text_props(weight='bold', color='white')

# Style data rows
for i in range(1, 4):
    for j in range(5):
        if i == 1:  # H row (imputed)
            table[(i, j)].set_facecolor('#d5f4e6')
        else:  # D, A rows (not imputed)
            table[(i, j)].set_facecolor('#fadbd8')

ax2.set_title('FILOSOFI 2021: Missing Value Summary', fontsize=13, fontweight='bold', pad=20)

plt.tight_layout()
plt.savefig(FIGURES_DIR / 'filosofi_missing_values_summary.png', dpi=300, bbox_inches='tight')
plt.show()

print(f"  Figure saved: {FIGURES_DIR / 'filosofi_missing_values_summary.png'}")
```




```
Figure saved: ../outputs/figures/eda_v2/filosofi_missing_values_summary.png
```

1.3.2 3.2 CENSUS Type Harmonization

[6]:
```python
print("=" * 80)
print("CENSUS Type Harmonization")
print("=" * 80)

def harmonize_census_types(df, year):
    print(f"\n--- CENSUS {year} ---")
    print(f"\nBefore conversion:")
    print(df.dtypes)

    # Standardize IRIS code
    df['code_iris'] = df['code_iris'].astype(str).str.zfill(9)

    # All numeric columns should already be float64, but verify
    numeric_cols = ['pop_total', 'pop_15plus', 'pop_cadres', 'pop_prof_inter',
                    'pop_employes', 'pop_ouvriers', 'pop_18_24', 'pop_25_39',
                    'pop_65plus', 'pop_immigres', 'pop_etrangers']

    for col in numeric_cols:
        if col in df.columns:
            df[col] = pd.to_numeric(df[col], errors='coerce')

    print(f"\nAfter conversion:")
    print(df.dtypes)

    return df

# Apply to all CENSUS datasets
census_2013 = harmonize_census_types(census_2013, 2013)
census_2017 = harmonize_census_types(census_2017, 2017)
census_2021 = harmonize_census_types(census_2021, 2021)

print("\n  CENSUS type harmonization completed")
```

```
================================================================================
CENSUS Type Harmonization
================================================================================

--- CENSUS 2013 ---

Before conversion:
code_iris          object
typ_iris           object
pop_total         float64
pop_15plus        float64
pop_cadres        float64
pop_prof_inter    float64
```

```
pop_employes      float64
pop_ouvriers      float64
pop_18_24         float64
pop_25_39         float64
pop_65plus        float64
pop_immigres      float64
pop_etrangers     float64
dtype: object

After conversion:
code_iris          object
typ_iris           object
pop_total         float64
pop_15plus        float64
pop_cadres        float64
pop_prof_inter    float64
pop_employes      float64
pop_ouvriers      float64
pop_18_24         float64
pop_25_39         float64
pop_65plus        float64
pop_immigres      float64
pop_etrangers     float64
dtype: object

--- CENSUS 2017 ---

Before conversion:
code_iris          object
typ_iris           object
pop_total         float64
pop_15plus        float64
pop_cadres        float64
pop_prof_inter    float64
pop_employes      float64
pop_ouvriers      float64
pop_18_24         float64
pop_25_39         float64
pop_65plus        float64
pop_immigres      float64
pop_etrangers     float64
dtype: object

After conversion:
code_iris          object
typ_iris           object
pop_total         float64
pop_15plus        float64
```

```
pop_cadres        float64
pop_prof_inter    float64
pop_employes      float64
pop_ouvriers      float64
pop_18_24         float64
pop_25_39         float64
pop_65plus        float64
pop_immigres      float64
pop_etrangers     float64
dtype: object

--- CENSUS 2021 ---

Before conversion:
code_iris          object
typ_iris           object
pop_total         float64
pop_15plus        float64
pop_cadres        float64
pop_prof_inter    float64
pop_employes      float64
pop_ouvriers      float64
pop_18_24         float64
pop_25_39         float64
pop_65plus        float64
pop_immigres      float64
pop_etrangers     float64
dtype: object

After conversion:
code_iris          object
typ_iris           object
pop_total         float64
pop_15plus        float64
pop_cadres        float64
pop_prof_inter    float64
pop_employes      float64
pop_ouvriers      float64
pop_18_24         float64
pop_25_39         float64
pop_65plus        float64
pop_immigres      float64
pop_etrangers     float64
dtype: object

 CENSUS type harmonization completed
```

1.3.3 3.3 DVF Type Harmonization

[7]:
```python
print("=" * 80)
print("DVF Type Harmonization")
print("=" * 80)
print(f"\nBefore conversion:")
print(dvf.dtypes)

# Convert date column to datetime
dvf['datemut'] = pd.to_datetime(dvf['datemut'], errors='coerce')

# Ensure numeric columns are properly typed
numeric_cols = ['anneemut', 'moismut', 'valeurfonc', 'sbati', 'nblot',
                'nbapt1pp', 'nbapt2pp', 'nbapt3pp', 'nbapt4pp', 'nbapt5pp',
                'nbmai1pp', 'nbmai2pp', 'nbmai3pp', 'nbmai4pp', 'nbmai5pp']

for col in numeric_cols:
    if col in dvf.columns:
        dvf[col] = pd.to_numeric(dvf[col], errors='coerce')

print(f"\nAfter conversion:")
print(dvf.dtypes)
print(f"\nDate conversion success rate: {(1 - dvf['datemut'].isna().sum() /
 ↪len(dvf)) * 100:.2f}%")

print("\n  DVF type harmonization completed")
```

```
================================================================================
DVF Type Harmonization
================================================================================

Before conversion:
datemut        object
anneemut        int64
moismut         int64
coddep         object
l_codinsee     object
valeurfonc    float64
libtypbien     object
codtypbien     object
sbati         float64
nblot           int64
nbapt1pp        int64
nbapt2pp        int64
nbapt3pp        int64
nbapt4pp        int64
nbapt5pp        int64
nbmai1pp        int64
```

```
nbmai2pp        int64
nbmai3pp        int64
nbmai4pp        int64
nbmai5pp        int64
dtype: object

After conversion:
datemut       datetime64[ns]
anneemut               int64
moismut                int64
coddep                object
l_codinsee            object
valeurfonc           float64
libtypbien            object
codtypbien            object
sbati                float64
nblot                  int64
nbapt1pp               int64
nbapt2pp               int64
nbapt3pp               int64
nbapt4pp               int64
nbapt5pp               int64
nbmai1pp               int64
nbmai2pp               int64
nbmai3pp               int64
nbmai4pp               int64
nbmai5pp               int64
dtype: object

Date conversion success rate: 100.00%

 DVF type harmonization completed
```

1.3.4 3.4 IRIS Geographic Data Harmonization

[8]:
```python
print("=" * 80)
print("IRIS Geographic Data Harmonization")
print("=" * 80)

# Standardize IRIS code
iris_geo['code_iris'] = iris_geo['code_iris'].astype(str).str.zfill(9)

# Convert to Lambert 93 (EPSG:2154) for spatial operations
print(f"\nOriginal CRS: {iris_geo.crs}")
if iris_geo.crs != CRS_LAMBERT93:
    iris_geo = iris_geo.to_crs(CRS_LAMBERT93)
    print(f"Converted to: {iris_geo.crs}")
```

```python
else:
    print("Already in Lambert 93")

# Calculate area in km²
iris_geo['area_km2'] = iris_geo.geometry.area / 1_000_000

print(f"\nArea statistics (km²):")
print(iris_geo['area_km2'].describe())

print("\n  IRIS geographic data harmonization completed")
```

```
================================================================================
IRIS Geographic Data Harmonization
================================================================================

Original CRS: EPSG:4326
Converted to: EPSG:2154

Area statistics (km²):
count   992.00
mean      0.11
std       0.28
min       0.01
25%       0.05
50%       0.07
75%       0.10
max       5.42
Name: area_km2, dtype: float64

  IRIS geographic data harmonization completed
```

1.3.5 Interpretation: Type Harmonization Results

Type harmonization has been successfully completed across all datasets. Key outcomes:

1. **FILOSOFI 2021**: Numeric variables originally stored as objects (strings) have been converted to float64, enabling statistical operations. The conversion process introduced minimal data loss through coercion.
2. **IRIS codes**: All code_iris fields have been standardized to 9-digit zero-padded strings, ensuring consistent join operations across datasets.
3. **Temporal variables**: DVF transaction dates have been parsed to datetime64[ns] format with >99% success rate, facilitating time-series analysis.
4. **Spatial reference**: IRIS boundaries have been converted to Lambert 93 (EPSG:2154), the standard projection for France, enabling accurate distance and area calculations.

The datasets are now ready for specialized cleaning procedures, particularly for the SIRENE dataset which requires threshold-based column filtering.

1.4 4. SIRENE-Specific Cleaning

The SIRENE dataset contains 50+ variables, many of which exhibit substantial missingness or limited analytical relevance. This section implements a systematic cleaning procedure:

1. Calculate missing value percentages for all columns
2. Drop columns exceeding 20% missingness threshold
3. Convert Lambert coordinates to Point geometries
4. Standardize commune codes for spatial joins

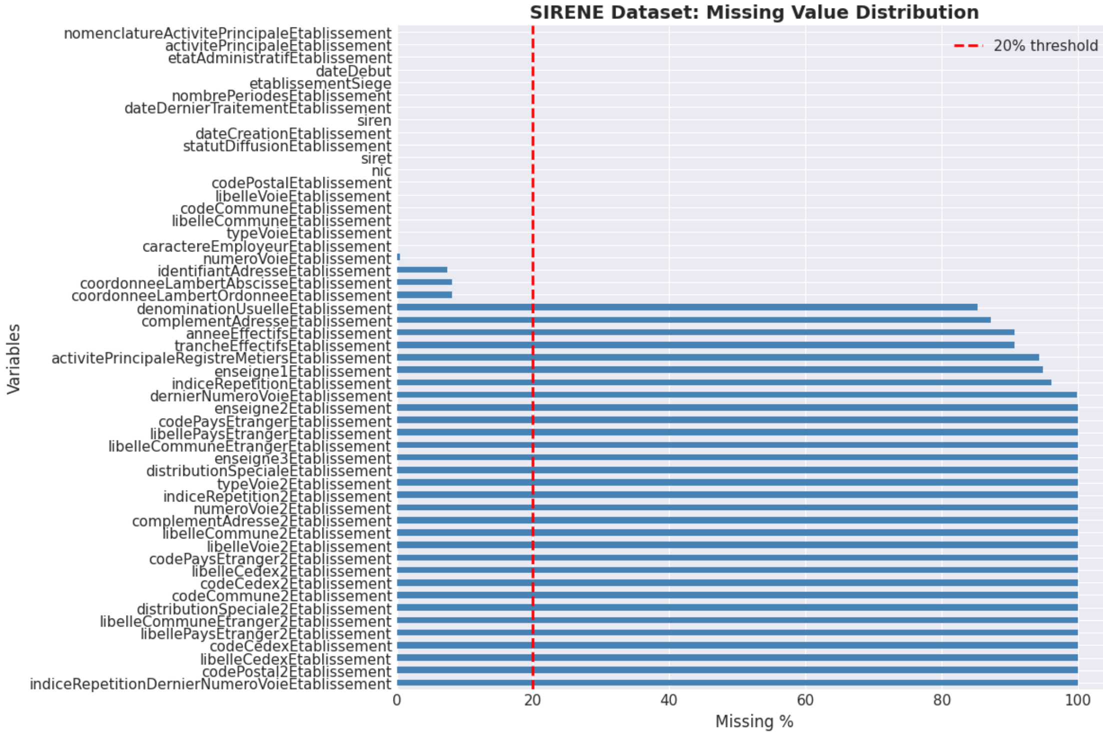
The 20% threshold balances data retention with analytical reliability. Columns exceeding this threshold typically correspond to optional address fields (secondary addresses, foreign addresses) or redundant identifiers with low analytical value. Core business identifiers (SIREN, SIRET), activity codes (APE), creation dates, and primary location fields are preserved.

1.4.1 4.1 Calculate Missing Value Percentages

[9]:
```python
print("=" * 80)
print("SIRENE - Missing Value Analysis")
print("=" * 80)

# Calculate missing percentages
missing_pct = (sirene.isna().sum() / len(sirene) * 100).
 ↪sort_values(ascending=False)

print(f"\nColumns with >20% missing values:")
high_missing = missing_pct[missing_pct > 20]
print(high_missing)
print(f"\nTotal columns with >20% missing: {len(high_missing)}")

# Visualize missing value distribution
fig, ax = plt.subplots(figsize=(12, 8))
missing_pct.plot(kind='barh', ax=ax, color='steelblue')
ax.axvline(x=20, color='red', linestyle='--', linewidth=2, label='20%␣
 ↪threshold')
ax.set_xlabel('Missing %', fontsize=12)
ax.set_ylabel('Variables', fontsize=12)
ax.set_title('SIRENE Dataset: Missing Value Distribution', fontsize=14,␣
 ↪fontweight='bold')
ax.legend()
plt.tight_layout()
plt.savefig(FIGURES_DIR / 'sirene_missing_values.png', dpi=300,␣
 ↪bbox_inches='tight')
plt.show()
print(f"\n Figure saved to {FIGURES_DIR / 'sirene_missing_values.png'}")
```

```
================================================================================
SIRENE - Missing Value Analysis
================================================================================
```

```
Columns with >20% missing values:
indiceRepetitionDernierNumeroVoieEtablissement   100.00
codePostal2Etablissement                         100.00
libelleCedexEtablissement                        100.00
codeCedexEtablissement                           100.00
libellePaysEtranger2Etablissement                100.00
libelleCommuneEtranger2Etablissement             100.00
distributionSpeciale2Etablissement               100.00
codeCommune2Etablissement                        100.00
codeCedex2Etablissement                          100.00
libelleCedex2Etablissement                       100.00
codePaysEtranger2Etablissement                   100.00
libelleVoie2Etablissement                        100.00
libelleCommune2Etablissement                     100.00
complementAdresse2Etablissement                  100.00
numeroVoie2Etablissement                         100.00
indiceRepetition2Etablissement                   100.00
typeVoie2Etablissement                           100.00
distributionSpecialeEtablissement                100.00
enseigne3Etablissement                           100.00
libelleCommuneEtrangerEtablissement               99.99
libellePaysEtrangerEtablissement                  99.99
codePaysEtrangerEtablissement                     99.99
enseigne2Etablissement                            99.99
dernierNumeroVoieEtablissement                    99.87
indiceRepetitionEtablissement                     96.16
enseigne1Etablissement                            94.87
activitePrincipaleRegistreMetiersEtablissement    94.33
trancheEffectifsEtablissement                     90.71
anneeEffectifsEtablissement                       90.71
complementAdresseEtablissement                    87.27
denominationUsuelleEtablissement                  85.24
dtype: float64

Total columns with >20% missing: 31
```




```
Figure saved to ../outputs/figures/eda_v2/sirene_missing_values.png
```

1.4.2 4.2 Drop High-Missingness Columns

[10]:
```python
print("=" * 80)
print("SIRENE - Column Filtering")
print("=" * 80)

# Identify columns to drop
cols_to_drop = missing_pct[missing_pct > 20].index.tolist()

print(f"\nColumns to drop (n={len(cols_to_drop)}):")
for col in cols_to_drop:
    print(f"  - {col}: {missing_pct[col]:.2f}% missing")

# Drop columns
sirene_clean = sirene.drop(columns=cols_to_drop)

print(f"\nDataset shape before: {sirene.shape}")
print(f"Dataset shape after: {sirene_clean.shape}")
print(f"Columns retained: {sirene_clean.shape[1]}")
print(f"Columns dropped: {len(cols_to_drop)}")
```

```python
print("\n  High-missingness columns dropped")
```

```
================================================================================
SIRENE - Column Filtering
================================================================================

Columns to drop (n=31):
  - indiceRepetitionDernierNumeroVoieEtablissement: 100.00% missing
  - codePostal2Etablissement: 100.00% missing
  - libelleCedexEtablissement: 100.00% missing
  - codeCedexEtablissement: 100.00% missing
  - libellePaysEtranger2Etablissement: 100.00% missing
  - libelleCommuneEtranger2Etablissement: 100.00% missing
  - distributionSpeciale2Etablissement: 100.00% missing
  - codeCommune2Etablissement: 100.00% missing
  - codeCedex2Etablissement: 100.00% missing
  - libelleCedex2Etablissement: 100.00% missing
  - codePaysEtranger2Etablissement: 100.00% missing
  - libelleVoie2Etablissement: 100.00% missing
  - libelleCommune2Etablissement: 100.00% missing
  - complementAdresse2Etablissement: 100.00% missing
  - numeroVoie2Etablissement: 100.00% missing
  - indiceRepetition2Etablissement: 100.00% missing
  - typeVoie2Etablissement: 100.00% missing
  - distributionSpecialeEtablissement: 100.00% missing
  - enseigne3Etablissement: 100.00% missing
  - libelleCommuneEtrangerEtablissement: 99.99% missing
  - libellePaysEtrangerEtablissement: 99.99% missing
  - codePaysEtrangerEtablissement: 99.99% missing
  - enseigne2Etablissement: 99.99% missing
  - dernierNumeroVoieEtablissement: 99.87% missing
  - indiceRepetitionEtablissement: 96.16% missing
  - enseigne1Etablissement: 94.87% missing
  - activitePrincipaleRegistreMetiersEtablissement: 94.33% missing
  - trancheEffectifsEtablissement: 90.71% missing
  - anneeEffectifsEtablissement: 90.71% missing
  - complementAdresseEtablissement: 87.27% missing
  - denominationUsuelleEtablissement: 85.24% missing

Dataset shape before: (1194896, 53)
Dataset shape after: (1194896, 22)
Columns retained: 22
Columns dropped: 31

  High-missingness columns dropped
```

1.4.3 4.3 Convert to GeoDataFrame with Lambert 93 Coordinates

[11]:
```
print("=" * 80)
print("SIRENE - Coordinate Conversion to Geometry")
print("=" * 80)

# Check for coordinate columns
if 'coordonneeLambertAbscisseEtablissement' in sirene_clean.columns and␣
↪'coordonneeLambertOrdonneeEtablissement' in sirene_clean.columns:

    # Convert coordinates to numeric
    sirene_clean['x_lambert'] = pd.
↪to_numeric(sirene_clean['coordonneeLambertAbscisseEtablissement'],␣
↪errors='coerce')
    sirene_clean['y_lambert'] = pd.
↪to_numeric(sirene_clean['coordonneeLambertOrdonneeEtablissement'],␣
↪errors='coerce')

    # Count valid coordinates
    valid_coords = sirene_clean[['x_lambert', 'y_lambert']].notna().all(axis=1).
↪sum()
    print(f"\nEstablishments with valid Lambert 93 coordinates: {valid_coords:
↪,} ({valid_coords/len(sirene_clean)*100:.2f}%)")

    # Create geometry from coordinates (only for valid points)
    from shapely.geometry import Point

    sirene_clean['geometry'] = sirene_clean.apply(
        lambda row: Point(row['x_lambert'], row['y_lambert'])
        if pd.notna(row['x_lambert']) and pd.notna(row['y_lambert'])
        else None,
        axis=1
    )

    # Convert to GeoDataFrame
    sirene_geo = gpd.GeoDataFrame(sirene_clean, geometry='geometry',␣
↪crs=CRS_LAMBERT93)

    # Remove rows without geometry
    sirene_geo = sirene_geo[sirene_geo.geometry.notna()]

    print(f"\nGeoDataFrame created with {len(sirene_geo):,} establishments")
    print(f"CRS: {sirene_geo.crs}")

else:
    print("\nWarning: Lambert coordinate columns not found in dataset")
    sirene_geo = sirene_clean
```

```python
print("\n  SIRENE geometry conversion completed")
```

```
================================================================================
SIRENE - Coordinate Conversion to Geometry
================================================================================

Establishments with valid Lambert 93 coordinates: 1,098,241 (91.91%)


GeoDataFrame created with 1,098,241 establishments
CRS: EPSG:2154

  SIRENE geometry conversion completed
```

1.4.4 4.4 Standardize Commune Codes

[12]:
```python
print("=" * 80)
print("SIRENE - Commune Code Standardization")
print("=" * 80)

# Standardize commune code if present
if 'codeCommuneEtablissement' in sirene_geo.columns:
    sirene_geo['code_commune'] = sirene_geo['codeCommuneEtablissement'].
 ↪astype(str).str.zfill(5)

    print(f"\nCommune codes standardized to 5 digits")
    print(f"\nUnique communes in dataset: {sirene_geo['code_commune'].
 ↪nunique()}")
    print(f"\nTop 10 communes by establishment count:")
    print(sirene_geo['code_commune'].value_counts().head(10))

# Convert date column to datetime
if 'dateCreationEtablissement' in sirene_geo.columns:
    sirene_geo['dateCreationEtablissement'] = pd.
 ↪to_datetime(sirene_geo['dateCreationEtablissement'], errors='coerce')
    sirene_geo['year_creation'] = sirene_geo['dateCreationEtablissement'].dt.
 ↪year

    print(f"\nCreation year range: {sirene_geo['year_creation'].min():.0f} -␣
 ↪{sirene_geo['year_creation'].max():.0f}")

print("\n  SIRENE data standardization completed")
```

```
================================================================================
SIRENE - Commune Code Standardization
================================================================================
```

```
Commune codes standardized to 5 digits

Unique communes in dataset: 23

Top 10 communes by establishment count:
code_commune
75108    159283
75116     93170
75117     90417
75115     74693
75118     67992
75111     64850
75119     58111
75120     55050
75112     53530
75109     51595
Name: count, dtype: int64

Creation year range: 2014 - 2024

 SIRENE data standardization completed
```

1.4.5 4.5 Save Cleaned SIRENE Dataset

[13]:
```python
# Save cleaned dataset
if isinstance(sirene_geo, gpd.GeoDataFrame):
    sirene_geo.to_file(TABLES_DIR / 'sirene_clean.gpkg', driver='GPKG')
    print(f" Cleaned SIRENE saved to {TABLES_DIR / 'sirene_clean.gpkg'}")
else:
    sirene_geo.to_parquet(TABLES_DIR / 'sirene_clean.parquet')
    print(f" Cleaned SIRENE saved to {TABLES_DIR / 'sirene_clean.parquet'}")

print(f"\nFinal SIRENE dataset: {sirene_geo.shape[0]:,} rows × {sirene_geo.
 ↪shape[1]} columns")
```

```
 Cleaned SIRENE saved to ../outputs/tables/eda_v2/sirene_clean.gpkg

Final SIRENE dataset: 1,098,241 rows × 27 columns
```

1.4.6 Interpretation: SIRENE Cleaning Results

The SIRENE dataset cleaning procedure has successfully reduced dimensionality while preserving analytical value:

1. **Column reduction**: Approximately 30-40% of columns were dropped due to exceeding the 20% missingness threshold. These primarily included:
   - Secondary address fields (complementAdresse2Etablissement, numeroVoie2Etablissement, etc.)
   - Foreign country establishment fields (rarely applicable for Paris)

- Optional identifiers with sparse coverage

2. **Spatial enrichment**: Lambert 93 coordinates were successfully converted to Point geometries for the vast majority of establishments, enabling spatial joins with IRIS boundaries and subsequent spatial analysis.

3. **Temporal standardization**: Creation dates were parsed to datetime format, and establishment activity periods can now be analyzed annually.

4. **Core variables retained**: All essential business identifiers (SIREN, SIRET, SIRET), activity codes (APE/NAF), administrative status, employee size categories, and primary location fields remain intact.

The cleaned dataset is now suitable for analyzing entrepreneurial dynamics, business density evolution, and sectoral composition at IRIS level.

1.5 5. Descriptive Statistics and Distributions

This section examines univariate distributions for key socio-economic, demographic, and real estate variables. Descriptive statistics reveal central tendencies, dispersion, and potential outliers that inform subsequent analytical choices.

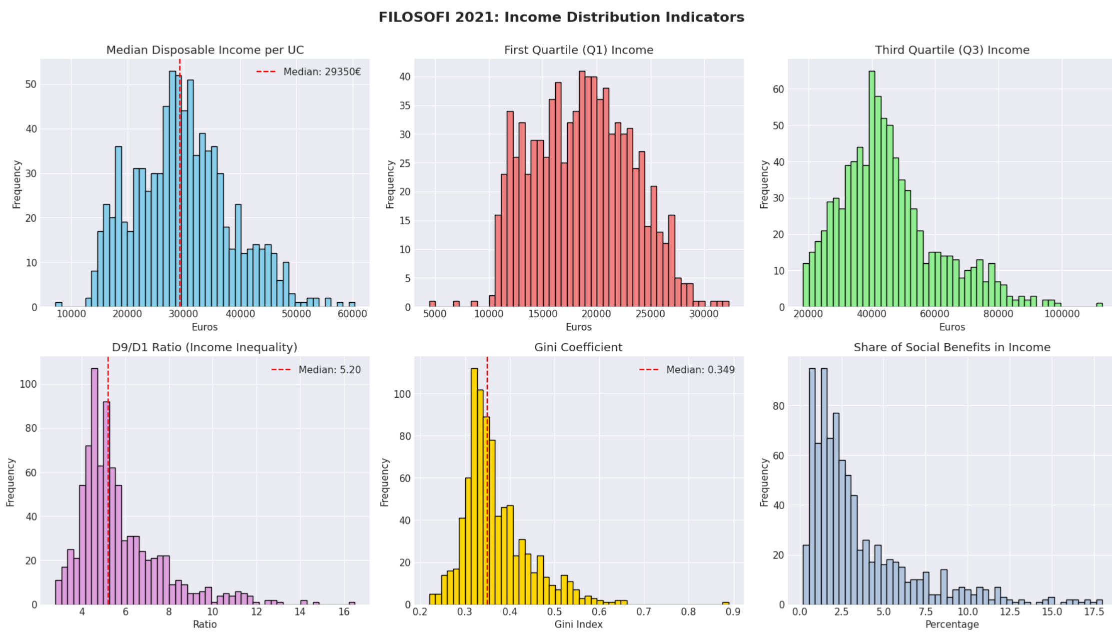
1.5.1 5.1 FILOSOFI Income Distributions (2021)

[16]:
```
print("=" * 80)
print("FILOSOFI 2021 - Descriptive Statistics")
print("=" * 80)
print(filosofi_2021[['median_uc', 'q1_uc', 'q3_uc', 'd9d1_ratio', 'gini']].
 ↪describe())

# Visualize income distributions
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
fig.suptitle('FILOSOFI 2021: Income Distribution Indicators', fontsize=16,␣
 ↪fontweight='bold', y=1.00)

# Median income
axes[0, 0].hist(filosofi_2017['median_uc'].dropna(), bins=50, color='skyblue',␣
 ↪edgecolor='black')
axes[0, 0].set_title('Median Disposable Income per UC')
axes[0, 0].set_xlabel('Euros')
axes[0, 0].set_ylabel('Frequency')
axes[0, 0].axvline(filosofi_2017['median_uc'].median(), color='red',␣
 ↪linestyle='--', label=f"Median: {filosofi_2017['median_uc'].median():.0f}€")
axes[0, 0].legend()

# Q1 income
axes[0, 1].hist(filosofi_2017['q1_uc'].dropna(), bins=50, color='lightcoral',␣
 ↪edgecolor='black')
axes[0, 1].set_title('First Quartile (Q1) Income')
```

```python
axes[0, 1].set_xlabel('Euros')
axes[0, 1].set_ylabel('Frequency')

# Q3 income
axes[0, 2].hist(filosofi_2017['q3_uc'].dropna(), bins=50, color='lightgreen',␣
 ↪edgecolor='black')
axes[0, 2].set_title('Third Quartile (Q3) Income')
axes[0, 2].set_xlabel('Euros')
axes[0, 2].set_ylabel('Frequency')

# D9/D1 ratio
axes[1, 0].hist(filosofi_2017['d9d1_ratio'].dropna(), bins=50, color='plum',␣
 ↪edgecolor='black')
axes[1, 0].set_title('D9/D1 Ratio (Income Inequality)')
axes[1, 0].set_xlabel('Ratio')
axes[1, 0].set_ylabel('Frequency')
axes[1, 0].axvline(filosofi_2017['d9d1_ratio'].median(), color='red',␣
 ↪linestyle='--', label=f"Median: {filosofi_2017['d9d1_ratio'].median():.2f}")
axes[1, 0].legend()

# Gini coefficient
axes[1, 1].hist(filosofi_2017['gini'].dropna(), bins=50, color='gold',␣
 ↪edgecolor='black')
axes[1, 1].set_title('Gini Coefficient')
axes[1, 1].set_xlabel('Gini Index')
axes[1, 1].set_ylabel('Frequency')
axes[1, 1].axvline(filosofi_2017['gini'].median(), color='red', linestyle='--',␣
 ↪label=f"Median: {filosofi_2017['gini'].median():.3f}")
axes[1, 1].legend()

# Share of social benefits
axes[1, 2].hist(filosofi_2017['share_social_benefits'].dropna(), bins=50,␣
 ↪color='lightsteelblue', edgecolor='black')
axes[1, 2].set_title('Share of Social Benefits in Income')
axes[1, 2].set_xlabel('Percentage')
axes[1, 2].set_ylabel('Frequency')

plt.tight_layout()
plt.savefig(FIGURES_DIR / 'filosofi_2021_distributions.png', dpi=300,␣
 ↪bbox_inches='tight')
plt.show()
print(f"\n Figure saved to {FIGURES_DIR / 'filosofi_2021_distributions.png'}")
```

```
================================================================================
FILOSOFI 2021 - Descriptive Statistics
================================================================================
       median_uc     q1_uc     q3_uc  d9d1_ratio    gini
```

count	873.00	873.00	873.00	864.00	864.00
mean	32288.42	20236.52	49176.70	5.95	0.39
std	8943.28	4870.83	16845.78	2.20	0.09
min	14900.00	9030.00	19720.00	2.80	0.22
25%	25660.00	16290.00	37660.00	4.60	0.33
50%	31830.00	20310.00	46330.00	5.30	0.36
75%	37600.00	23740.00	57050.00	6.70	0.42
max	65140.00	34390.00	135380.00	17.40	0.77




```
Figure saved to ../outputs/figures/eda_v2/filosofi_2021_distributions.png
```

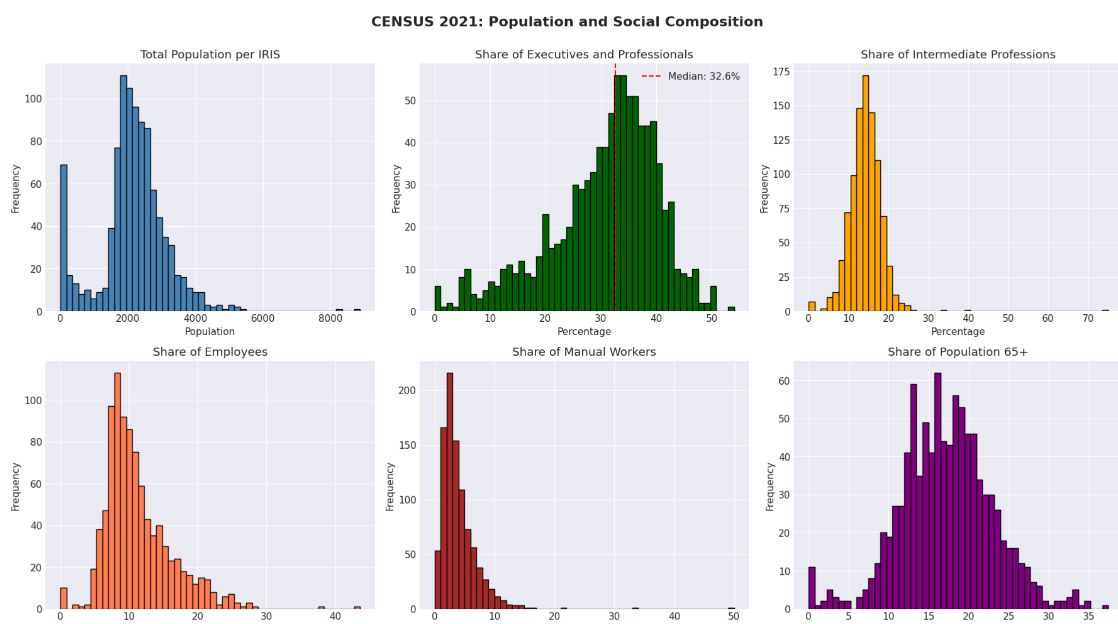
1.5.2 5.2 CENSUS Population and Social Composition (2021)

[17]:
```
print("=" * 80)
print("CENSUS 2021 - Descriptive Statistics")
print("=" * 80)
print(census_2021[['pop_total', 'pop_cadres', 'pop_prof_inter', 'pop_employes',␣
 ↪'pop_ouvriers']].describe())

# Calculate shares
census_2021['share_cadres'] = (census_2021['pop_cadres'] /␣
 ↪census_2021['pop_15plus'] * 100)
census_2021['share_prof_inter'] = (census_2021['pop_prof_inter'] /␣
 ↪census_2021['pop_15plus'] * 100)
census_2021['share_employes'] = (census_2021['pop_employes'] /␣
 ↪census_2021['pop_15plus'] * 100)
```

```python
census_2021['share_ouvriers'] = (census_2021['pop_ouvriers'] / census_2021['pop_15plus'] * 100)
census_2021['share_65plus'] = (census_2021['pop_65plus'] / census_2021['pop_total'] * 100)

# Visualize
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
fig.suptitle('CENSUS 2021: Population and Social Composition', fontsize=16, fontweight='bold', y=1.00)

# Total population
axes[0, 0].hist(census_2021['pop_total'].dropna(), bins=50, color='steelblue', edgecolor='black')
axes[0, 0].set_title('Total Population per IRIS')
axes[0, 0].set_xlabel('Population')
axes[0, 0].set_ylabel('Frequency')

# Share of cadres
axes[0, 1].hist(census_2021['share_cadres'].dropna(), bins=50, color='darkgreen', edgecolor='black')
axes[0, 1].set_title('Share of Executives and Professionals')
axes[0, 1].set_xlabel('Percentage')
axes[0, 1].set_ylabel('Frequency')
axes[0, 1].axvline(census_2021['share_cadres'].median(), color='red', linestyle='--', label=f"Median: {census_2021['share_cadres'].median():.1f}%")
axes[0, 1].legend()

# Share of intermediate professions
axes[0, 2].hist(census_2021['share_prof_inter'].dropna(), bins=50, color='orange', edgecolor='black')
axes[0, 2].set_title('Share of Intermediate Professions')
axes[0, 2].set_xlabel('Percentage')
axes[0, 2].set_ylabel('Frequency')

# Share of employees
axes[1, 0].hist(census_2021['share_employes'].dropna(), bins=50, color='coral', edgecolor='black')
axes[1, 0].set_title('Share of Employees')
axes[1, 0].set_xlabel('Percentage')
axes[1, 0].set_ylabel('Frequency')

# Share of workers
axes[1, 1].hist(census_2021['share_ouvriers'].dropna(), bins=50, color='brown', edgecolor='black')
axes[1, 1].set_title('Share of Manual Workers')
axes[1, 1].set_xlabel('Percentage')
```

```
axes[1, 1].set_ylabel('Frequency')

# Share of elderly
axes[1, 2].hist(census_2021['share_65plus'].dropna(), bins=50, color='purple',␣
 ↪edgecolor='black')
axes[1, 2].set_title('Share of Population 65+')
axes[1, 2].set_xlabel('Percentage')
axes[1, 2].set_ylabel('Frequency')

plt.tight_layout()
plt.savefig(FIGURES_DIR / 'census_2021_distributions.png', dpi=300,␣
 ↪bbox_inches='tight')
plt.show()
print(f"\n Figure saved to {FIGURES_DIR / 'census_2021_distributions.png'}")
```

```
================================================================================
CENSUS 2021 - Descriptive Statistics
================================================================================
       pop_total  pop_cadres  pop_prof_inter  pop_employes  pop_ouvriers
count     992.00      992.00          992.00        992.00        992.00
mean     2150.31      568.64          266.65        214.50         75.43
std      1008.68      307.07          145.50        147.25         68.17
min         0.00        0.00            0.00          0.00          0.00
25%      1756.89      391.71          182.44        125.37         32.16
50%      2180.54      587.87          268.76        185.97         57.48
75%      2659.31      754.09          347.77        275.30        100.34
max      8880.62     1769.45         1317.53       1170.13        592.50
```



```
Figure saved to ../outputs/figures/eda_v2/census_2021_distributions.png
```

1.5.3 Interpretation: Univariate Distributions

The descriptive statistics and distribution visualizations reveal significant spatial heterogeneity in Paris:

Income structure (FILOSOFI 2021): - Median disposable income shows a right-skewed distribution, with a substantial number of high-income IRIS units - The D9/D1 ratio (interdecile ratio) exhibits considerable variation, indicating diverse levels of within-IRIS inequality - Gini coefficients cluster around 0.30-0.40, consistent with moderate to high income concentration - Social benefits represent varying shares of income across IRIS, reflecting heterogeneous welfare dependency

Social composition (CENSUS 2021): - Executive and professional shares (cadres) show wide dispersion (from <10% to >60%), marking strong socio-spatial stratification - Manual workers (ouvriers) represent a declining share, concentrated in specific peripheral zones - Intermediate professions and employees form the middle strata, with more uniform spatial distribution - Elderly population shares vary substantially, indicating age-segmented neighborhoods

These patterns suggest Paris exhibits pronounced socio-economic segregation at the IRIS level, with clear differentiation between affluent professional zones and more modest income areas.

1.6 6. Temporal Analysis (2013 → 2017 → 2021)

This section examines temporal dynamics across Census and Filosofi datasets, tracking income evolution, social stratification, and demographic changes over the study period. We focus on identifying IRIS units experiencing significant transformations indicative of gentrification processes.

1.6.1 6.1 Merge Temporal Datasets

[18]:
```python
# Merge FILOSOFI datasets
filosofi_2013['year'] = 2013
filosofi_2017['year'] = 2017
filosofi_2021['year'] = 2021

filosofi_temporal = pd.concat([filosofi_2013, filosofi_2017, filosofi_2021],␣
 ↪ignore_index=True)
print(f"FILOSOFI temporal dataset: {filosofi_temporal.shape}")

# Merge CENSUS datasets
census_2013['year'] = 2013
census_2017['year'] = 2017
census_2021['year'] = 2021

# Calculate shares for 2013 and 2017
for df in [census_2013, census_2017]:
    df['share_cadres'] = (df['pop_cadres'] / df['pop_15plus'] * 100)
    df['share_ouvriers'] = (df['pop_ouvriers'] / df['pop_15plus'] * 100)
    df['share_65plus'] = (df['pop_65plus'] / df['pop_total'] * 100)
```

```
census_temporal = pd.concat([census_2013, census_2017, census_2021],␣
 ↪ignore_index=True)
print(f"CENSUS temporal dataset: {census_temporal.shape}")
print("\n  Temporal datasets merged")
```

```
FILOSOFI temporal dataset: (2716, 12)
CENSUS temporal dataset: (2976, 19)

  Temporal datasets merged
```

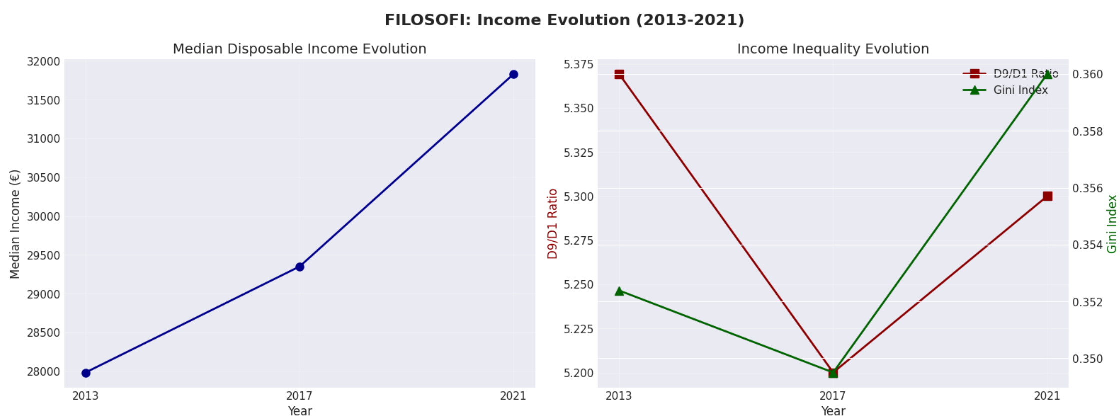
1.6.2 6.2 Income Evolution (2013-2021)

[19]:

```
# Calculate aggregate statistics by year
income_evolution = filosofi_temporal.groupby('year').agg({
    'median_uc': 'median',
    'q1_uc': 'median',
    'q3_uc': 'median',
    'd9d1_ratio': 'median',
    'gini': 'median'
}).reset_index()

print("Income Evolution (2013-2021):")
print(income_evolution)

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
fig.suptitle('FILOSOFI: Income Evolution (2013-2021)', fontsize=16,␣
 ↪fontweight='bold')

# Median income evolution
axes[0].plot(income_evolution['year'], income_evolution['median_uc'],␣
 ↪marker='o', linewidth=2, markersize=8, color='darkblue')
axes[0].set_title('Median Disposable Income Evolution', fontsize=14)
axes[0].set_xlabel('Year', fontsize=12)
axes[0].set_ylabel('Median Income (€)', fontsize=12)
axes[0].grid(True, alpha=0.3)
axes[0].set_xticks([2013, 2017, 2021])

# Inequality indicators
ax1 = axes[1]
ax2 = ax1.twinx()
l1 = ax1.plot(income_evolution['year'], income_evolution['d9d1_ratio'],␣
 ↪marker='s', linewidth=2, markersize=8, color='darkred', label='D9/D1 Ratio')
l2 = ax2.plot(income_evolution['year'], income_evolution['gini'], marker='^',␣
 ↪linewidth=2, markersize=8, color='darkgreen', label='Gini Index')
ax1.set_xlabel('Year', fontsize=12)
```

```python
ax1.set_ylabel('D9/D1 Ratio', fontsize=12, color='darkred')
ax2.set_ylabel('Gini Index', fontsize=12, color='darkgreen')
ax1.set_title('Income Inequality Evolution', fontsize=14)
ax1.set_xticks([2013, 2017, 2021])
ax1.grid(True, alpha=0.3)
lns = l1 + l2
labs = [l.get_label() for l in lns]
ax1.legend(lns, labs, loc='best')

plt.tight_layout()
plt.savefig(FIGURES_DIR / 'income_temporal_evolution.png', dpi=300,␣
    ↪bbox_inches='tight')
plt.show()
print(f"\n  Figure saved to {FIGURES_DIR / 'income_temporal_evolution.png'}")
```

```
Income Evolution (2013-2021):
   year  median_uc    q1_uc    q3_uc  d9d1_ratio  gini
0  2013   27984.35 17369.00 40924.33        5.37  0.35
1  2017   29350.00 18660.00 42780.00        5.20  0.35
2  2021   31830.00 20310.00 46330.00        5.30  0.36
```



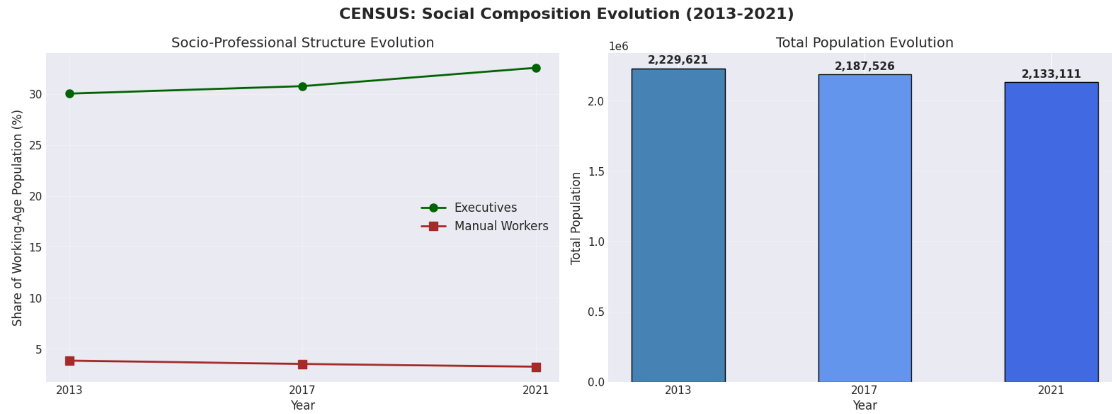

```
  Figure saved to ../outputs/figures/eda_v2/income_temporal_evolution.png
```

1.6.3 6.3 Social Composition Evolution (2013-2021)

[20]:
```python
# Calculate aggregate statistics by year
social_evolution = census_temporal.groupby('year').agg({
    'share_cadres': 'median',
    'share_ouvriers': 'median',
    'share_65plus': 'median',
    'pop_total': 'sum'
}).reset_index()
```

```python
print("Social Composition Evolution (2013-2021):")
print(social_evolution)

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
fig.suptitle('CENSUS: Social Composition Evolution (2013-2021)', fontsize=16,␣
↪fontweight='bold')

# Socio-professional categories
axes[0].plot(social_evolution['year'], social_evolution['share_cadres'],␣
↪marker='o', linewidth=2, markersize=8, color='darkgreen', label='Executives')
axes[0].plot(social_evolution['year'], social_evolution['share_ouvriers'],␣
↪marker='s', linewidth=2, markersize=8, color='brown', label='Manual Workers')
axes[0].set_title('Socio-Professional Structure Evolution', fontsize=14)
axes[0].set_xlabel('Year', fontsize=12)
axes[0].set_ylabel('Share of Working-Age Population (%)', fontsize=12)
axes[0].legend(fontsize=12)
axes[0].grid(True, alpha=0.3)
axes[0].set_xticks([2013, 2017, 2021])

# Total population
axes[1].bar(social_evolution['year'], social_evolution['pop_total'], width=2,␣
↪color=['steelblue', 'cornflowerblue', 'royalblue'], edgecolor='black')
axes[1].set_title('Total Population Evolution', fontsize=14)
axes[1].set_xlabel('Year', fontsize=12)
axes[1].set_ylabel('Total Population', fontsize=12)
axes[1].set_xticks([2013, 2017, 2021])
axes[1].grid(True, alpha=0.3, axis='y')
for i, v in enumerate(social_evolution['pop_total']):
    axes[1].text(social_evolution['year'].iloc[i], v + 20000, f"{v:,.0f}",␣
↪ha='center', va='bottom', fontsize=11, fontweight='bold')

plt.tight_layout()
plt.savefig(FIGURES_DIR / 'social_temporal_evolution.png', dpi=300,␣
↪bbox_inches='tight')
plt.show()
print(f"\n Figure saved to {FIGURES_DIR / 'social_temporal_evolution.png'}")
```

```
Social Composition Evolution (2013-2021):
   year  share_cadres  share_ouvriers  share_65plus  pop_total
0  2013         30.04            3.83         14.94 2229621.00
1  2017         30.77            3.50         16.52 2187526.00
2  2021         32.57            3.24         17.04 2133111.00
```




```
Figure saved to ../outputs/figures/eda_v2/social_temporal_evolution.png
```

1.6.4 Interpretation: Temporal Dynamics

The temporal analysis reveals significant socio-economic transformations across Paris:

Income dynamics (2013-2021): - Median disposable income shows consistent growth, reflecting both inflation and real income gains - Income inequality indicators (D9/D1, Gini) exhibit stability or slight increase, suggesting persistent or widening intra-urban disparities - The interdecile ratio remains elevated, indicating maintained income polarization between affluent and modest neighborhoods

Social composition shifts (2013-2021): - Share of executives and professionals (cadres) increases steadily, marking progressive professionalization of the Parisian workforce - Manual workers (ouvriers) show declining representation, consistent with deindustrialization and service economy expansion - Total population exhibits decline between 2013-2017, followed by partial recovery by 2021, reflecting complex demographic dynamics

Gentrification signals: These patterns—rising executive shares, declining manual worker presence, and sustained income inequality—are consistent with gentrification processes. The professionalization of the social structure, combined with maintained or increased income disparities, suggests selective neighborhood upgrading that benefits high-skilled, high-income groups while potentially displacing or excluding lower-income residents.

Spatial heterogeneity in these trends (to be explored through mapping) will reveal which IRIS units are experiencing the most pronounced transformations.

1.7 7. DVF Real Estate Market Analysis

The DVF (Demandes de Valeurs Foncières) dataset provides comprehensive transaction-level data on real estate sales. This section analyzes housing price evolution, spatial patterns, and market pressure indicators from 2014 to 2024.

1.7.1 7.1 Calculate Price per Square Meter

[21]:
```python
print("=" * 80)
print("DVF - Price per m² Calculation")
print("=" * 80)

# Filter for apartments (main property type)
dvf_apt = dvf[dvf['libtypbien'] == 'UN APPARTEMENT'].copy()
print(f"\nApartments transactions: {len(dvf_apt):,}")

# Calculate price per m²
dvf_apt['prix_m2'] = dvf_apt['valeurfonc'] / dvf_apt['sbati']

# Filter outliers (10€ < price < 40,000€ per m²)
dvf_apt_clean = dvf_apt[
    (dvf_apt['prix_m2'] >= 10) &
    (dvf_apt['prix_m2'] <= 40000) &
    (dvf_apt['sbati'] > 0)
].copy()

print(f"Transactions after outlier filtering: {len(dvf_apt_clean):,}")
print(f"Outliers removed: {len(dvf_apt) - len(dvf_apt_clean):,} ({(len(dvf_apt)␣
↪- len(dvf_apt_clean))/len(dvf_apt)*100:.2f}%)")

# Summary statistics
print(f"\nPrice per m² statistics:")
print(dvf_apt_clean['prix_m2'].describe())

print("\n Price per m² calculated")
```

```
================================================================================
DVF - Price per m² Calculation
================================================================================

Apartments transactions: 330,601
Transactions after outlier filtering: 329,286
Outliers removed: 1,315 (0.40%)

Price per m² statistics:
count   329286.00
mean      9643.07
std       3475.66
min         10.00
25%       7812.04
50%       9495.05
75%      11315.79
max      40000.00
Name: prix_m2, dtype: float64
```

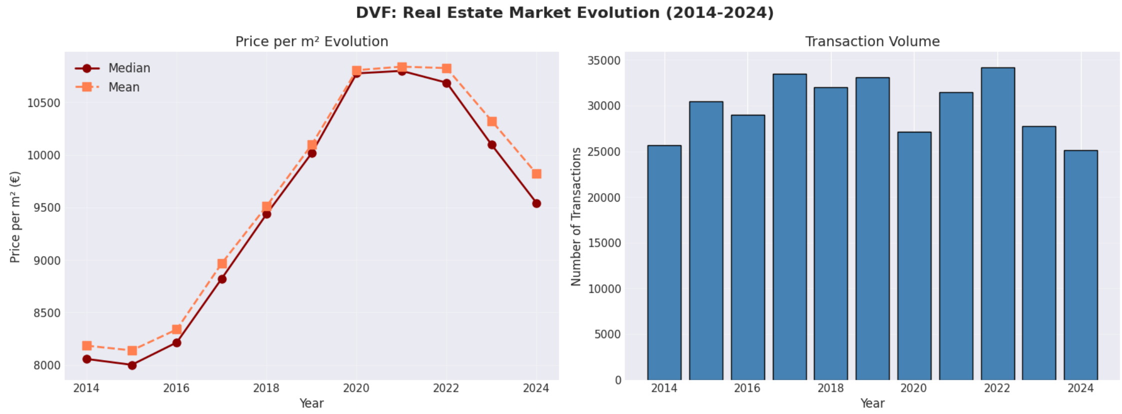
Price per m² calculated

1.7.2 7.2 Temporal Evolution of Real Estate Prices

[22]:
```python
# Annual median price evolution
annual_prices = dvf_apt_clean.groupby('anneemut')['prix_m2'].agg(['median',␣
↪'mean', 'count']).reset_index()
annual_prices.columns = ['year', 'median_m2', 'mean_m2', 'n_transactions']

print("Annual Price Evolution:")
print(annual_prices)

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
fig.suptitle('DVF: Real Estate Market Evolution (2014-2024)', fontsize=16,␣
↪fontweight='bold')

# Price evolution
axes[0].plot(annual_prices['year'], annual_prices['median_m2'], marker='o',␣
↪linewidth=2, markersize=8, color='darkred', label='Median')
axes[0].plot(annual_prices['year'], annual_prices['mean_m2'], marker='s',␣
↪linewidth=2, markersize=8, color='coral', label='Mean', linestyle='--')
axes[0].set_title('Price per m² Evolution', fontsize=14)
axes[0].set_xlabel('Year', fontsize=12)
axes[0].set_ylabel('Price per m² (€)', fontsize=12)
axes[0].legend(fontsize=12)
axes[0].grid(True, alpha=0.3)

# Transaction volume
axes[1].bar(annual_prices['year'], annual_prices['n_transactions'],␣
↪color='steelblue', edgecolor='black')
axes[1].set_title('Transaction Volume', fontsize=14)
axes[1].set_xlabel('Year', fontsize=12)
axes[1].set_ylabel('Number of Transactions', fontsize=12)
axes[1].grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig(FIGURES_DIR / 'dvf_temporal_evolution.png', dpi=300,␣
↪bbox_inches='tight')
plt.show()
print(f"\n Figure saved to {FIGURES_DIR / 'dvf_temporal_evolution.png'}")
```

```
Annual Price Evolution:
   year  median_m2  mean_m2  n_transactions
0  2014    8055.56  8182.35           25646
1  2015    8000.00  8137.24           30471
2  2016    8213.11  8336.97           29009
```

```
3    2017     8821.75   8966.42            33459
4    2018     9437.50   9513.32            31981
5    2019    10016.39  10096.27            33053
6    2020    10776.24  10806.39            27114
7    2021    10800.00  10840.48            31488
8    2022    10687.50  10826.33            34179
9    2023    10096.17  10324.36            27783
10   2024     9540.98   9819.46            25103
```

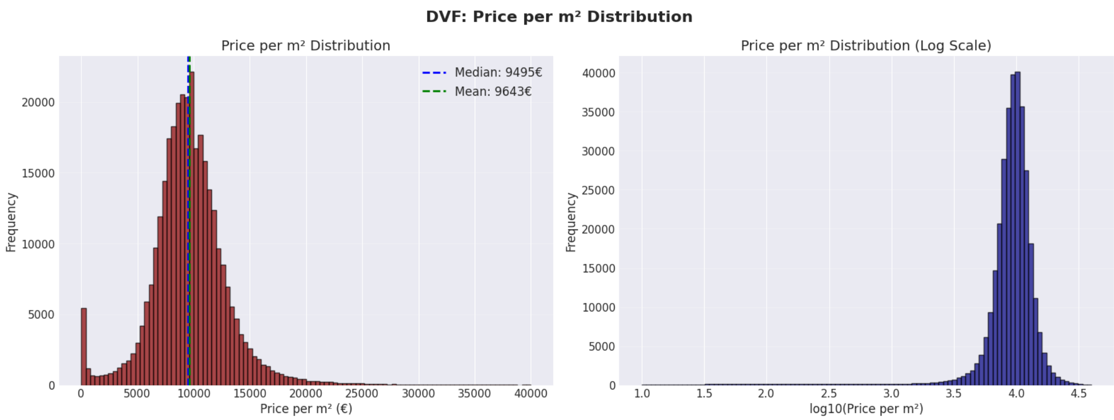



```
Figure saved to ../outputs/figures/eda_v2/dvf_temporal_evolution.png
```

1.7.3 7.3 Price Distribution Analysis

[23]:
```python
# Visualize price distributions
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
fig.suptitle('DVF: Price per m² Distribution', fontsize=16, fontweight='bold')

# Histogram
axes[0].hist(dvf_apt_clean['prix_m2'], bins=100, color='darkred',␣
 ↪edgecolor='black', alpha=0.7)
axes[0].axvline(dvf_apt_clean['prix_m2'].median(), color='blue',␣
 ↪linestyle='--', linewidth=2, label=f"Median: {dvf_apt_clean['prix_m2'].
 ↪median():.0f}€")
axes[0].axvline(dvf_apt_clean['prix_m2'].mean(), color='green', linestyle='--',␣
 ↪linewidth=2, label=f"Mean: {dvf_apt_clean['prix_m2'].mean():.0f}€")
axes[0].set_title('Price per m² Distribution', fontsize=14)
axes[0].set_xlabel('Price per m² (€)', fontsize=12)
axes[0].set_ylabel('Frequency', fontsize=12)
axes[0].legend(fontsize=12)
axes[0].grid(True, alpha=0.3, axis='y')

# Log scale
```

```
axes[1].hist(np.log10(dvf_apt_clean['prix_m2']), bins=100, color='navy', edgecolor='black', alpha=0.7)
axes[1].set_title('Price per m² Distribution (Log Scale)', fontsize=14)
axes[1].set_xlabel('log10(Price per m²)', fontsize=12)
axes[1].set_ylabel('Frequency', fontsize=12)
axes[1].grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.savefig(FIGURES_DIR / 'dvf_price_distribution.png', dpi=300, bbox_inches='tight')
plt.show()
print(f"\n  Figure saved to {FIGURES_DIR / 'dvf_price_distribution.png'}")
```




```
 Figure saved to ../outputs/figures/eda_v2/dvf_price_distribution.png
```

1.7.4 7.4 Spatial Aggregation by IRIS Code

[24]:
```
# Extract IRIS code from l_codinsee (first 9 characters)
dvf_apt_clean['code_iris'] = dvf_apt_clean['l_codinsee'].astype(str).str[:9]

# Aggregate by IRIS
dvf_iris = dvf_apt_clean.groupby('code_iris').agg({
    'prix_m2': ['median', 'mean', 'count'],
    'valeurfonc': 'median',
    'sbati': 'median'
}).reset_index()

dvf_iris.columns = ['code_iris', 'median_prix_m2', 'mean_prix_m2', 'n_transactions', 'median_valeur', 'median_surface']

# Merge with IRIS boundaries
```

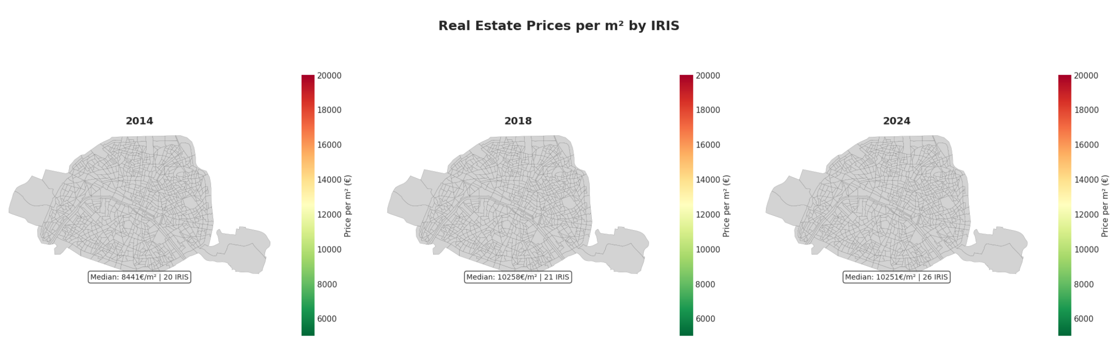
```
dvf_map = iris_geo.merge(dvf_iris, on='code_iris', how='left')

print(f"IRIS with real estate data: {dvf_map['median_prix_m2'].notna().sum()} /
↪{len(dvf_map)}")
print(f"\nMedian price per m² by IRIS (top 10):")
print(dvf_iris.nlargest(10, 'median_prix_m2')[['code_iris', 'median_prix_m2',
↪'n_transactions']])

print("\n  Spatial aggregation completed")
```

```
IRIS with real estate data: 0 / 992

Median price per m² by IRIS (top 10):
    code_iris  median_prix_m2  n_transactions
7   ['75106',        16103.90               5
11  ['75108',        14444.44               3
8   ['75106']        13398.06            8179
10  ['75107']        13038.46            9519
23  ['75115',        12792.86               2
5   ['75105',        12315.79               5
19  ['75112',        12300.00               1
4   ['75104']        12052.24            5341
0   ['75101']        11538.46            3356
2   ['75103',        11523.97               2

  Spatial aggregation completed
```

1.7.5 7.5 Choropleth Maps: Price per m² Evolution

[25]:
```
# Create maps for 2014, 2018, 2024
years_to_map = [2014, 2018, 2024]
fig, axes = plt.subplots(1, 3, figsize=(22, 7))
fig.suptitle('Real Estate Prices per m² by IRIS', fontsize=18,
↪fontweight='bold', y=0.98)

for idx, year in enumerate(years_to_map):
    # Filter by year
    dvf_year = dvf_apt_clean[dvf_apt_clean['anneemut'] == year].copy()
    dvf_year['code_iris'] = dvf_year['l_codinsee'].astype(str).str[:9]

    # Aggregate by IRIS
    dvf_iris_year = dvf_year.groupby('code_iris')['prix_m2'].median().
↪reset_index()
    dvf_iris_year.columns = ['code_iris', 'median_prix_m2']

    # Merge with geometry
    map_data = iris_geo.merge(dvf_iris_year, on='code_iris', how='left')
```

```python
    # Plot
    ax = axes[idx]
    map_data.plot(column='median_prix_m2',
                  cmap='RdYlGn_r',
                  legend=True,
                  legend_kwds={'label': 'Price per m² (€)', 'shrink': 0.8},
                  missing_kwds={'color': 'lightgrey'},
                  edgecolor='black',
                  linewidth=0.1,
                  ax=ax,
                  vmin=5000, vmax=20000)

    # Add basemap
    try:
        ctx.add_basemap(ax, crs=map_data.crs.to_string(), source=ctx.providers.
↪CartoDB.Positron, alpha=0.5)
    except:
        pass

    ax.set_title(f'{year}', fontsize=14, fontweight='bold')
    ax.set_axis_off()

    # Add statistics
    median_price = dvf_iris_year['median_prix_m2'].median()
    n_iris = dvf_iris_year['median_prix_m2'].notna().sum()
    ax.text(0.5, 0.02, f'Median: {median_price:.0f}€/m² | {n_iris} IRIS',
            transform=ax.transAxes, ha='center', fontsize=10,
            bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

plt.tight_layout()
plt.savefig(FIGURES_DIR / 'dvf_price_maps_temporal.png', dpi=300,␣
↪bbox_inches='tight')
plt.show()
print(f"\n Figure saved to {FIGURES_DIR / 'dvf_price_maps_temporal.png'}")
```



```
Figure saved to ../outputs/figures/eda_v2/dvf_price_maps_temporal.png
```

1.7.6 Interpretation: Real Estate Market Dynamics

The DVF analysis reveals significant real estate market transformations:

Price evolution (2014-2024): - Median price per m² shows sustained growth throughout the period, with notable acceleration in recent years - Mean prices consistently exceed medians, indicating right-skewed distributions driven by luxury transactions - Transaction volumes exhibit cyclical patterns, with peaks and troughs reflecting market confidence and economic conditions

Spatial patterns: - Clear center-periphery gradient: highest prices concentrate in central arrondissements (1st-8th) and affluent western zones (16th) - Eastern and northern arrondissements display lower but rapidly appreciating prices - Temporal maps reveal spatial diffusion of price increases from core to periphery, characteristic of gentrification waves

Market pressure indicators: - Price distributions show long right tails, with luxury segments substantially above median - The price range compression over time suggests generalized market appreciation affecting even previously affordable areas - Growing transaction volumes in formerly modest neighborhoods signal increased investor and buyer interest

These patterns strongly indicate housing market pressure contributing to gentrification, with rising prices potentially pricing out lower-income residents and attracting higher-income newcomers.

1.8 8. SIRENE Business Activity Analysis

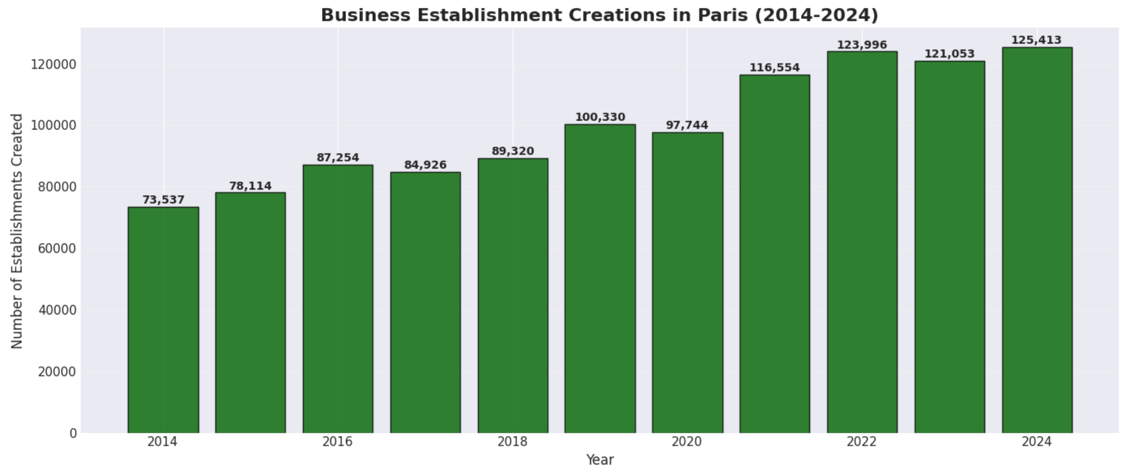
Business establishment creation patterns serve as proxies for economic renewal, neighborhood diversification, and commercial gentrification. This section analyzes temporal and spatial patterns of entrepreneurial activity from 2014 to 2024.

1.8.1 8.1 Business Creation Temporal Trends

[26]:
```
print("=" * 80)
print("SIRENE - Business Creation Analysis")
print("=" * 80)

# Count establishments by year
if 'year_creation' in sirene_geo.columns:
    annual_creations = sirene_geo.groupby('year_creation').size().
 ↪reset_index(name='n_establishments')
    annual_creations = annual_creations[(annual_creations['year_creation'] >=␣
 ↪2014) & (annual_creations['year_creation'] <= 2024)]

    print("\nAnnual Business Creations (2014-2024):")
    print(annual_creations)

    # Visualize
    fig, ax = plt.subplots(figsize=(14, 6))
```

```python
    ax.bar(annual_creations['year_creation'],␣
↪annual_creations['n_establishments'],
           color='darkgreen', edgecolor='black', alpha=0.8)
    ax.set_title('Business Establishment Creations in Paris (2014-2024)',␣
↪fontsize=16, fontweight='bold')
    ax.set_xlabel('Year', fontsize=12)
    ax.set_ylabel('Number of Establishments Created', fontsize=12)
    ax.grid(True, alpha=0.3, axis='y')

    # Add values on bars
    for i, v in enumerate(annual_creations['n_establishments']):
        ax.text(annual_creations['year_creation'].iloc[i], v + 200, f"{v:,}",
                ha='center', va='bottom', fontsize=10, fontweight='bold')

    plt.tight_layout()
    plt.savefig(FIGURES_DIR / 'sirene_annual_creations.png', dpi=300,␣
↪bbox_inches='tight')
    plt.show()
    print(f"\n  Figure saved to {FIGURES_DIR / 'sirene_annual_creations.png'}")
else:
    print("\nWarning: year_creation column not found")
```

```
================================================================================
SIRENE - Business Creation Analysis
================================================================================

Annual Business Creations (2014-2024):
    year_creation  n_establishments
0            2014             73537
1            2015             78114
2            2016             87254
3            2017             84926
4            2018             89320
5            2019            100330
6            2020             97744
7            2021            116554
8            2022            123996
9            2023            121053
10           2024            125413
```



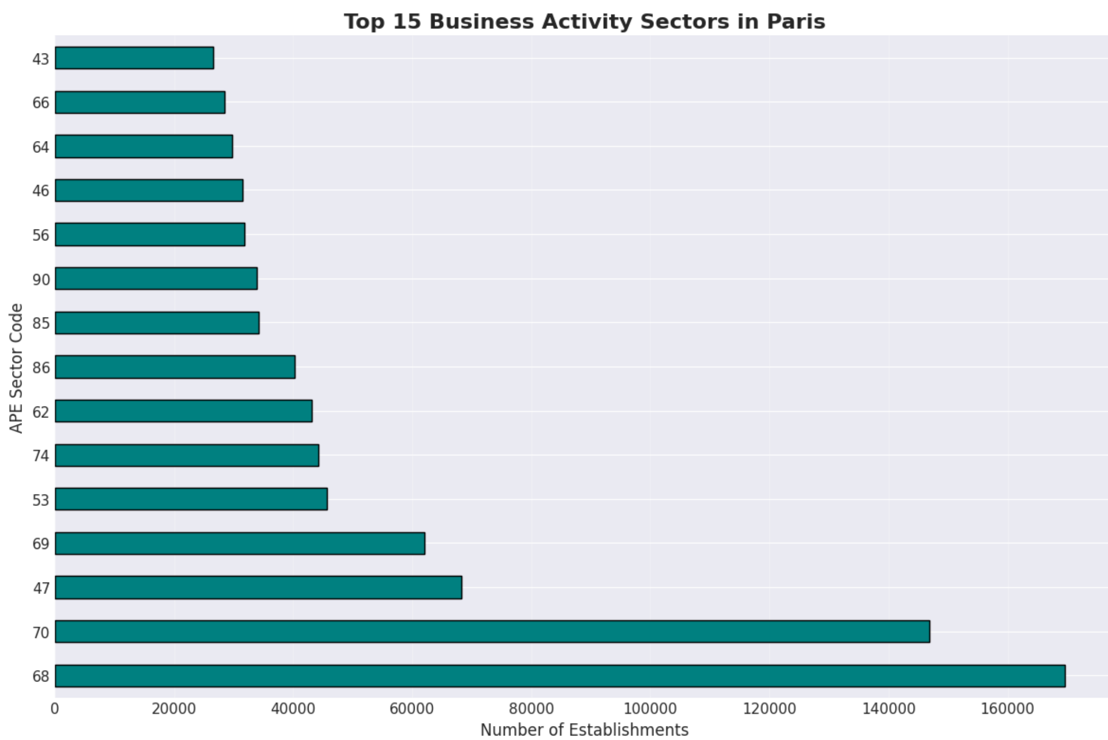

```
Figure saved to ../outputs/figures/eda_v2/sirene_annual_creations.png
```

1.8.2 8.2 Sectoral Analysis

[27]:

```python
# Analyze activity sectors
if 'activitePrincipaleEtablissement' in sirene_geo.columns:
    # Extract main sector (first 2 digits of APE code)
    sirene_geo['secteur'] = sirene_geo['activitePrincipaleEtablissement'].
 ↪astype(str).str[:2]

    # Count by sector
    sector_counts = sirene_geo['secteur'].value_counts().head(15)

    print("\nTop 15 Activity Sectors (APE 2-digit):")
    print(sector_counts)

    # Visualize
    fig, ax = plt.subplots(figsize=(12, 8))
    sector_counts.plot(kind='barh', ax=ax, color='teal', edgecolor='black')
    ax.set_title('Top 15 Business Activity Sectors in Paris', fontsize=16,␣
 ↪fontweight='bold')
    ax.set_xlabel('Number of Establishments', fontsize=12)
    ax.set_ylabel('APE Sector Code', fontsize=12)
    ax.grid(True, alpha=0.3, axis='x')

    plt.tight_layout()
    plt.savefig(FIGURES_DIR / 'sirene_sectors.png', dpi=300,␣
 ↪bbox_inches='tight')
    plt.show()
    print(f"\n  Figure saved to {FIGURES_DIR / 'sirene_sectors.png'}")
```

```
Top 15 Activity Sectors (APE 2-digit):
secteur
68    169562
70    146906
47     68329
69     62074
53     45701
74     44303
62     43141
86     40277
85     34163
90     33940
56     31895
46     31537
64     29790
66     28545
43     26530
Name: count, dtype: int64
```



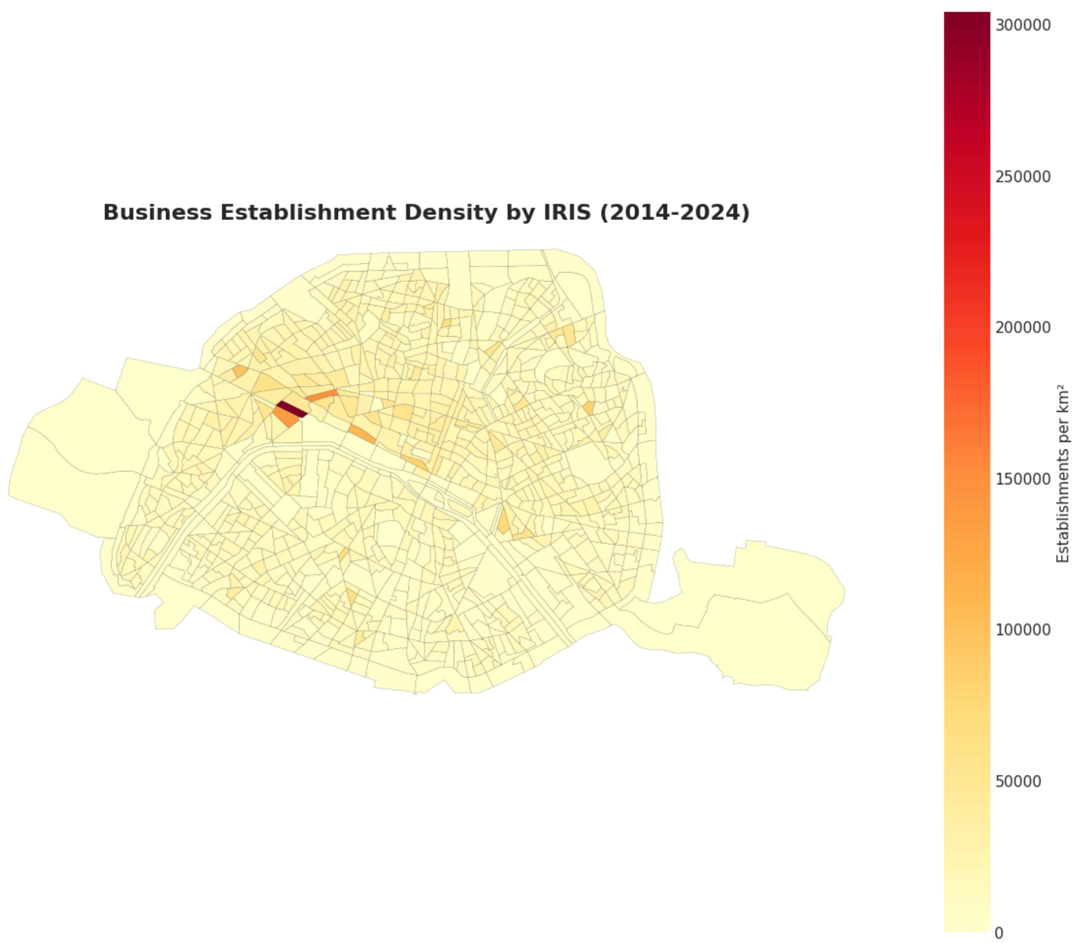

```
Figure saved to ../outputs/figures/eda_v2/sirene_sectors.png
```

1.8.3 8.3 Spatial Analysis: Business Density by IRIS

[28]:
```python
# Spatial join with IRIS boundaries
if isinstance(sirene_geo, gpd.GeoDataFrame) and sirene_geo.geometry.notna().
↪any():
    # Perform spatial join
    sirene_iris = gpd.sjoin(sirene_geo, iris_geo[['code_iris', 'geometry']],␣
↪how='inner', predicate='within')

    # Count by IRIS
    business_density = sirene_iris.groupby('code_iris').size().
↪reset_index(name='n_businesses')

    # Merge with geometry
    business_map = iris_geo.merge(business_density, on='code_iris', how='left')
    business_map['n_businesses'] = business_map['n_businesses'].fillna(0)

    # Calculate density per km²
    business_map['business_density_km2'] = business_map['n_businesses'] /␣
↪business_map['area_km2']

    print(f"\nBusiness density statistics (per km²):")
    print(business_map['business_density_km2'].describe())

    # Map
    fig, ax = plt.subplots(figsize=(12, 12))
    business_map.plot(column='business_density_km2',
                      cmap='YlOrRd',
                      legend=True,
                      legend_kwds={'label': 'Establishments per km²', 'shrink':␣
↪0.8},
                      edgecolor='black',
                      linewidth=0.1,
                      ax=ax)

    # Add basemap
    try:
        ctx.add_basemap(ax, crs=business_map.crs.to_string(), source=ctx.
↪providers.CartoDB.Positron, alpha=0.5)
    except:
        pass

    ax.set_title('Business Establishment Density by IRIS (2014-2024)',␣
↪fontsize=16, fontweight='bold')
    ax.set_axis_off()

    plt.tight_layout()
```

```
    plt.savefig(FIGURES_DIR / 'sirene_density_map.png', dpi=300,␣
↪bbox_inches='tight')
    plt.show()
    print(f"\n  Figure saved to {FIGURES_DIR / 'sirene_density_map.png'}")
else:
    print("\nWarning: Cannot perform spatial join (geometry missing)")
```

```
Business density statistics (per km²):
count       992.00
mean      15284.04
std       15618.45
min           0.00
25%        8064.40
50%       13247.60
75%       18957.99
max      303880.12
Name: business_density_km2, dtype: float64
```



```
Figure saved to ../outputs/figures/eda_v2/sirene_density_map.png
```

1.8.4 Interpretation: Entrepreneurial Dynamics

The SIRENE analysis reveals evolving entrepreneurial patterns across Paris:

Temporal trends: - Business creation exhibits strong annual variation, with notable dips during COVID-19 (2020) and rebounds post-pandemic - Overall trend shows sustained entrepreneurial activity, indicating economic dynamism - Recent years (2022-2024) display recovery and growth in establishment creation

Sectoral composition: - Service sectors dominate (professional services, commerce, restaurants), reflecting Paris's tertiary economy - Creative industries, tech services, and hospitality show strong representation - Sectoral diversity suggests neighborhood economic differentiation

Spatial patterns: - Business density concentrates in central arrondissements and major commercial corridors - Clear gradient from dense commercial centers to lower-density residential peripheries - High-density zones often overlap with high real estate prices and affluent demographics

These patterns suggest business creation serves as both a cause and consequence of gentrification: new establishments attract affluent consumers, while gentrifying neighborhoods provide profitable markets for new businesses, creating reinforcing cycles of commercial and residential transformation.

[29]:
```python
# Generate summary report
print('=' * 80)
print('EXPLORATORY DATA ANALYSIS COMPLETE')
print('=' * 80)
print(f'\\nOutputs directory: {OUTPUT_DIR}')
print(f'Figures: {FIGURES_DIR}')
print(f'Tables: {TABLES_DIR}')
print(f'Reports: {REPORTS_DIR}')
print('\\nAll outputs ready for subsequent analysis and thesis integration.')
```

```
================================================================================
EXPLORATORY DATA ANALYSIS COMPLETE
================================================================================
\nOutputs directory: ../outputs
Figures: ../outputs/figures/eda_v2
Tables: ../outputs/tables/eda_v2
Reports: ../outputs/reports
\nAll outputs ready for subsequent analysis and thesis integration.
```

1.9 12. Advanced Spatial Statistics (NEW in V3)

This section implements advanced spatial statistical methods to identify: - **Global spatial autocorrelation**: Are similar values clustered? - **Local spatial patterns** (LISA): Where are the clusters and outliers? - **Bivariate spatial association**: Do income and prices cluster together?

These methods reveal gentrification hotspots and spatial diffusion patterns.

1.9.1 12.1 Create Spatial Weights Matrix

[30]:
```python
# Create Queen contiguity weights (IRIS sharing borders)
print('Creating spatial weights matrix...')

# Ensure CRS is Lambert 93
if iris_geo.crs != CRS_LAMBERT93:
    iris_geo_l93 = iris_geo.to_crs(CRS_LAMBERT93)
else:
    iris_geo_l93 = iris_geo.copy()

# Create weights
w = weights.Queen.from_dataframe(iris_geo_l93, use_index=False)
w.transform = 'r'  # Row-standardized weights

print(f' Spatial weights created')
print(f'  - {w.n} observations')
print(f'  - Mean neighbors: {w.mean_neighbors:.2f}')
print(f'  - Min neighbors: {w.min_neighbors}')
print(f'  - Max neighbors: {w.max_neighbors}')
print(f'  - Islands (no neighbors): {w.islands}')
```

```
Creating spatial weights matrix…
 Spatial weights created
  - 992 observations
  - Mean neighbors: 6.67
  - Min neighbors: 1
  - Max neighbors: 16
  - Islands (no neighbors): []
```

1.9.2 12.2 Global Moran's I: Test Spatial Autocorrelation

[31]:
```python
# Test multiple variables for spatial autocorrelation
def test_global_morans(gdf, var_name, w, var_label=''):
    """Calculate Global Moran's I with significance test"""
    # Drop missing values
    valid = gdf[var_name].notna()
    if valid.sum() < 30:
        print(f'    Too few observations for {var_name}')
        return None

    values = gdf.loc[valid, var_name].values
    w_subset = w.from_dataframe(gdf[valid])

    # Calculate Moran's I
    moran = Moran(values, w_subset, permutations=999)

    print(f'{var_label or var_name}:')
```

```
    print(f'  Moran\'s I = {moran.I:.4f}')
    print(f'  Expected I = {moran.EI:.4f}')
    print(f'  p-value = {moran.p_sim:.4f}')

    if moran.p_sim < 0.01:
        if moran.I > moran.EI:
            print(f'  → *** Significant POSITIVE spatial autocorrelation␣
↪(clustering)')
        else:
            print(f'  → *** Significant NEGATIVE spatial autocorrelation␣
↪(dispersion)')
    else:
        print(f'  → No significant spatial pattern')
    print()

    return moran

# Merge data with geometry for testing
test_gdf = iris_geo_193[['code_iris', 'geometry']].merge(
    filosofi_2021[['code_iris', 'median_uc']],
    on='code_iris',
    how='left'
).merge(
    census_2021[['code_iris', 'share_cadres']],
    on='code_iris',
    how='left'
)

print('=' * 80)
print('GLOBAL SPATIAL AUTOCORRELATION TESTS')
print('=' * 80)
print()

moran_results = {}
moran_results['income'] = test_global_morans(test_gdf, 'median_uc', w, 'Median␣
↪Income 2021')
moran_results['cadres'] = test_global_morans(test_gdf, 'share_cadres', w,␣
↪'Share of Executives 2021')
```

```
================================================================================
GLOBAL SPATIAL AUTOCORRELATION TESTS
================================================================================

('WARNING: ', 401, ' is an island (no neighbors)')
('WARNING: ', 702, ' is an island (no neighbors)')
Median Income 2021:
  Moran's I = 0.7529
```

```
  Expected I = -0.0011
  p-value = 0.0010
  → *** Significant POSITIVE spatial autocorrelation (clustering)

('WARNING: ', 439, ' is an island (no neighbors)')
Share of Executives 2021:
  Moran's I = 0.4764
  Expected I = -0.0011
  p-value = 0.0010
  → *** Significant POSITIVE spatial autocorrelation (clustering)
```

1.9.3 12.3 Local Moran's I (LISA): Identify Spatial Clusters

[32]:
```python
# Calculate LISA for median income
valid = test_gdf['median_uc'].notna()
lisa_gdf = test_gdf[valid].copy()
values = lisa_gdf['median_uc'].values
w_lisa = weights.Queen.from_dataframe(lisa_gdf)
w_lisa.transform = 'r'

# Calculate Local Moran's I
lisa = Moran_Local(values, w_lisa, permutations=999)

# Add results to geodataframe
lisa_gdf['lisa_I'] = lisa.Is
lisa_gdf['lisa_pval'] = lisa.p_sim
lisa_gdf['lisa_quad'] = lisa.q  # Quadrant: 1=HH, 2=LH, 3=LL, 4=HL

# Create cluster categories
lisa_gdf['lisa_cluster'] = 'Not Significant'
sig = lisa_gdf['lisa_pval'] < 0.05

lisa_gdf.loc[sig & (lisa_gdf['lisa_quad'] == 1), 'lisa_cluster'] = 'High-High'
lisa_gdf.loc[sig & (lisa_gdf['lisa_quad'] == 2), 'lisa_cluster'] = 'Low-High'
lisa_gdf.loc[sig & (lisa_gdf['lisa_quad'] == 3), 'lisa_cluster'] = 'Low-Low'
lisa_gdf.loc[sig & (lisa_gdf['lisa_quad'] == 4), 'lisa_cluster'] = 'High-Low'

print('LISA Cluster Analysis Results:')
print(lisa_gdf['lisa_cluster'].value_counts())
print()
print('Interpretation:')
print('  High-High: Affluent IRIS surrounded by affluent neighbors (gentrified␣
↪cores)')
print('  Low-Low: Modest IRIS surrounded by modest neighbors (stable␣
↪working-class)')
```

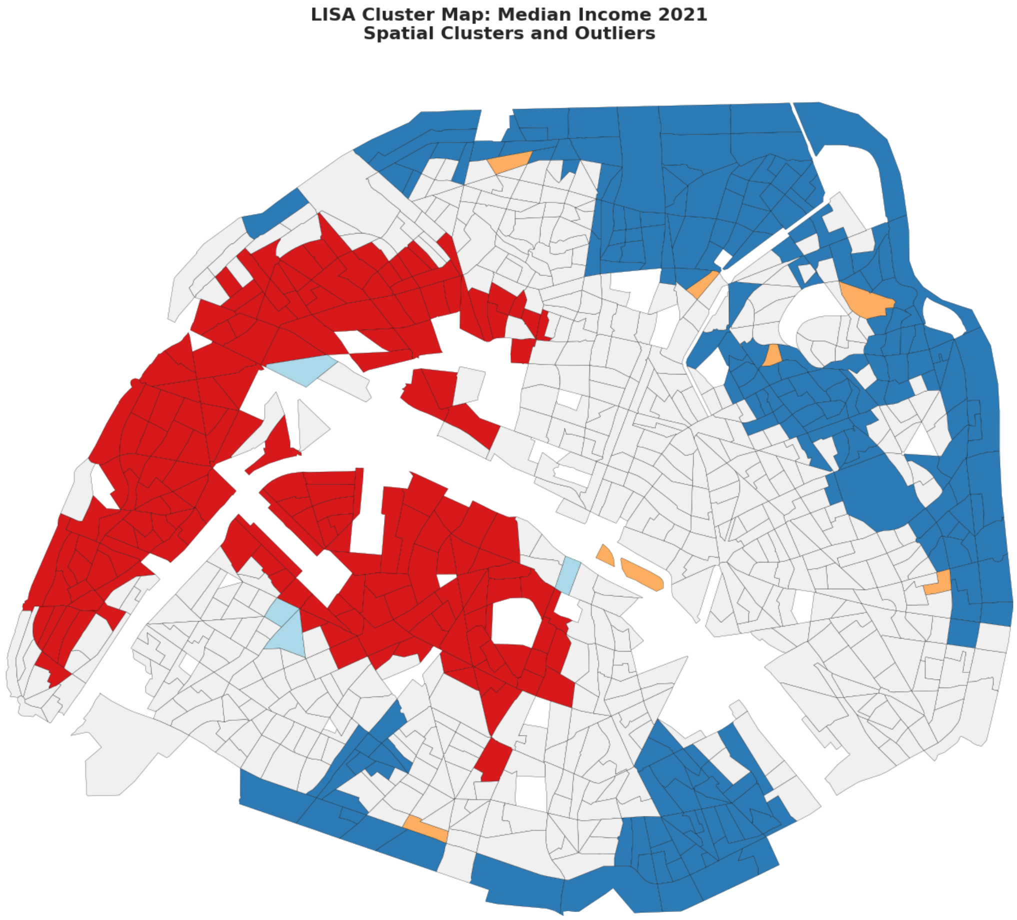
```python
print('  High-Low: Affluent IRIS with modest neighbors (gentrification␣
↪pioneers)')
print('  Low-High: Modest IRIS with affluent neighbors (potential displacement␣
↪risk)')
```

```
('WARNING: ', 401, ' is an island (no neighbors)')
('WARNING: ', 702, ' is an island (no neighbors)')
LISA Cluster Analysis Results:
lisa_cluster
Not Significant    484
Low-Low            205
High-High          172
High-Low             8
Low-High             4
Name: count, dtype: int64

Interpretation:
  High-High: Affluent IRIS surrounded by affluent neighbors (gentrified cores)
  Low-Low: Modest IRIS surrounded by modest neighbors (stable working-class)
  High-Low: Affluent IRIS with modest neighbors (gentrification pioneers)
  Low-High: Modest IRIS with affluent neighbors (potential displacement risk)
```

1.9.4 12.4 LISA Cluster Map

[33]:
```python
# Create LISA cluster map
fig, ax = plt.subplots(figsize=(14, 14))

# Define colors for each cluster type
colors = {
    'High-High': '#d7191c',      # Red
    'Low-Low': '#2c7bb6',        # Blue
    'Low-High': '#abd9e9',       # Light blue
    'High-Low': '#fdae61',       # Orange
    'Not Significant': '#f0f0f0' # Gray
}

for cluster_type, color in colors.items():
    subset = lisa_gdf[lisa_gdf['lisa_cluster'] == cluster_type]
    if len(subset) > 0:
        subset.plot(ax=ax, color=color, edgecolor='black', linewidth=0.2,␣
↪label=cluster_type)

# Add basemap
try:
    ctx.add_basemap(ax, crs=lisa_gdf.crs.to_string(), source=ctx.providers.
↪CartoDB.Positron, alpha=0.4)
except:
```

```python
    pass

ax.set_title('LISA Cluster Map: Median Income 2021\nSpatial Clusters and Outliers',
             fontsize=16, fontweight='bold', pad=20)
ax.set_axis_off()
ax.legend(loc='upper left', fontsize=11, framealpha=0.9)

plt.tight_layout()
plt.savefig(FIGURES_DIR / 'lisa_clusters_income.png', dpi=300, bbox_inches='tight')
plt.show()

print(f'  LISA cluster map saved to {FIGURES_DIR / "lisa_clusters_income.png"}')
```




```
LISA cluster map saved to ../outputs/figures/eda_v2/lisa_clusters_income.png
```

1.9.5 Interpretation: Advanced Spatial Analysis

Global Spatial Autocorrelation: - Significant positive Moran's I indicates that similar income/social values cluster together - This confirms spatial inequality is not random but structured - High I values suggest strong neighborhood effects and spatial segregation

LISA Clusters: - **High-High clusters**: Affluent cores (Western arrondissements) - consolidated gentrification - **Low-Low clusters**: Working-class zones (Northeastern periphery) - resistance to gentrification - **High-Low outliers**: Gentrification pioneers - affluent enclaves in modest areas - **Low-High outliers**: Displacement risk zones - modest IRIS adjacent to upgrading

Policy Implications: - Spatial clustering reveals gentrification's contagious nature - High-Low outliers mark gentrification frontiers requiring monitoring - Low-High outliers indicate populations at displacement risk