In [1]:
```
# Import all required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from collections import Counter
import re
import warnings
warnings.filterwarnings('ignore')

# Set up plotting style
plt.style.use('default')
sns.set_palette("viridis")
%matplotlib inline

print("✅ Libraries imported successfully!")
```

```
✅ Libraries imported successfully!
```

In [2]:
```
# Load your dataset
print("📖 Loading data...")
try:
    df = pd.read_csv("C:\\Users\\pc\\Downloads\\metadata.csv", low_memory=False)
    print("✅ Data loaded successfully!")
    print(f"📊 Shape: {df.shape[0]} rows, {df.shape[1]} columns")
except Exception as e:
    print(f"❌ Error loading data: {e}")
```

```
📖 Loading data...
✅ Data loaded successfully!
📊 Shape: 1056660 rows, 19 columns
```

In [3]:
```
# Quick overview of the data
print("🔍 QUICK DATA OVERVIEW")
print("=" * 50)

# Display first few rows
print("First 5 rows:")
display(df.head())

# Basic info
print("\nDataset info:")
df.info()

# Check columns
print(f"\nNumber of columns: {len(df.columns)}")
print("Column names:")
for i, col in enumerate(df.columns, 1):
    print(f"  {i:2d}. {col}")
```

```
🔍 QUICK DATA OVERVIEW
==================================================
First 5 rows:
```

	cord_uid	sha	source_x	title	
0	ug7v899j	d1aafb70c066a2068b02786f8929fd9c900897fb	PMC	Clinical features of culture-proven Mycoplasma...	10.1186/14 2334-
1	02tnwd4m	6b0567729c2143a66d737eb0a2f63f2dce2e5a7d	PMC	Nitric oxide: a pro-inflammatory mediator in l...	10.1186/r
2	ejv2xln0	06ced00a5fc04215949aa72528f2eeaae1d58927	PMC	Surfactant protein-D and pulmonary host defense	10.1186/r
3	2b73a28n	348055649b6b8cf2b9a376498df9bf41f7123605	PMC	Role of endothelin-1 in lung disease	10.1186/r
4	9785vg6d	5f48792a5fa08bed9f56016f4981ae2ca6031b32	PMC	Gene expression in epithelial cells in respons...	10.1186/r

```
Dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1056660 entries, 0 to 1056659
Data columns (total 19 columns):
 #   Column            Non-Null Count    Dtype  
---  ------            --------------    -----  
 0   cord_uid          1056660 non-null  object 
 1   sha               373766 non-null   object 
 2   source_x          1056660 non-null  object 
 3   title             1056157 non-null  object 
 4   doi               656780 non-null   object 
 5   pmcid             389571 non-null   object 
 6   pubmed_id         498932 non-null   object 
 7   license           1056660 non-null  object 
 8   abstract          821116 non-null   object 
 9   publish_time      1054846 non-null  object 
 10  authors           1032791 non-null  object 
 11  journal           969338 non-null   object 
 12  mag_id            0 non-null        float64
 13  who_covidence_id  482935 non-null   object 
 14  arxiv_id          14249 non-null    object 
 15  pdf_json_files    373766 non-null   object 
 16  pmc_json_files    315742 non-null   object 
 17  url               686934 non-null   object 
 18  s2_id             976468 non-null   float64
dtypes: float64(2), object(17)
memory usage: 153.2+ MB

Number of columns: 19
Column names:
   1. cord_uid
   2. sha
   3. source_x
   4. title
   5. doi
   6. pmcid
   7. pubmed_id
   8. license
   9. abstract
  10. publish_time
  11. authors
  12. journal
  13. mag_id
  14. who_covidence_id
  15. arxiv_id
  16. pdf_json_files
  17. pmc_json_files
  18. url
  19. s2_id
```

In [4]:

```
print("🔍 DETAILED DATA EXPLORATION")
print("=" * 50)

# Check data types
print("Data types:")
print(df.dtypes)
```

```
# Missing values analysis
print("\n❌ Missing values analysis:")
missing_data = df.isnull().sum().sort_values(ascending=False)
missing_percentage = (missing_data / len(df)) * 100

missing_df = pd.DataFrame({
    'Missing_Count': missing_data,
    'Missing_Percentage': missing_percentage
})
print(missing_df.head(15))  # Show top 15 columns with most missing values

# Basic statistics for numerical columns
print("\n📊 Basic statistics:")
print(df.describe())
```

```
🔍 DETAILED DATA EXPLORATION
==================================================
Data types:
cord_uid             object
sha                  object
source_x             object
title                object
doi                  object
pmcid                object
pubmed_id            object
license              object
abstract             object
publish_time         object
authors              object
journal              object
mag_id              float64
who_covidence_id     object
arxiv_id             object
pdf_json_files       object
pmc_json_files       object
url                  object
s2_id               float64
dtype: object

❌ Missing values analysis:
                  Missing_Count  Missing_Percentage
mag_id                  1056660          100.000000
arxiv_id                1042411           98.651506
pmc_json_files           740918           70.118865
pdf_json_files           682894           64.627600
sha                      682894           64.627600
pmcid                    667089           63.131849
who_covidence_id         573725           54.296084
pubmed_id                557728           52.782163
doi                      399880           37.843772
url                      369726           34.990063
abstract                 235544           22.291371
journal                   87322            8.263964
s2_id                     80192            7.589196
authors                   23869            2.258910
publish_time               1814            0.171673

📊 Basic statistics:
       mag_id         s2_id
count     0.0  9.764680e+05
mean      NaN  2.175871e+08
std       NaN  5.312281e+07
min       NaN  9.600000e+01
25%       NaN  2.211411e+08
50%       NaN  2.320829e+08
75%       NaN  2.373948e+08
max       NaN  2.491936e+08
```

In [5]:
```
print("🧹 DATA CLEANING")
print("=" * 50)
```

```
# Create a clean copy
df_clean = df.copy()

# Keep important columns (adjust based on what's available in your dataset)
important_cols = ['title', 'abstract', 'publish_time', 'journal', 'authors', 'sourc
available_cols = [col for col in important_cols if col in df_clean.columns]
df_clean = df_clean[available_cols]

print(f"📄 Keeping columns: {available_cols}")

# Remove rows without titles (essential for analysis)
initial_count = len(df_clean)
df_clean = df_clean.dropna(subset=['title'])
removed_count = initial_count - len(df_clean)
print(f"📝 Removed {removed_count} rows without titles")

# Convert dates
print("⏰ Converting dates...")
df_clean['publish_time'] = pd.to_datetime(df_clean['publish_time'], errors='coerce'

# Extract year
df_clean['year'] = df_clean['publish_time'].dt.year

# Word counts for abstracts
print("📊 Counting words...")
df_clean['abstract_word_count'] = df_clean['abstract'].apply(
    lambda x: len(str(x).split()) if pd.notnull(x) else 0
)

print(f"✅ Cleaned data shape: {df_clean.shape}")
print(f"📅 Year range: {df_clean['year'].min()} - {df_clean['year'].max()}")

# Show sample of cleaned data
print("\n👀 Sample of cleaned data:")
display(df_clean.head(3))
```

```
🧹 DATA CLEANING
==================================================
📄 Keeping columns: ['title', 'abstract', 'publish_time', 'journal', 'authors', 'so
urce_x', 'doi']
📝 Removed 503 rows without titles
⏰ Converting dates...
📊 Counting words...
✅ Cleaned data shape: (1056157, 9)
📅 Year range: 1856.0 - 2024.0

👀 Sample of cleaned data:
```

	title	abstract	publish_time	journal	authors	source_x	doi	yea
0	Clinical features of culture-proven Mycoplasma...	OBJECTIVE: This retrospective chart review des...	2001-07-04	BMC Infect Dis	Madani, Tariq A; Al-Ghamdi, Aisha A	PMC	10.1186/1471-2334-1-6	2001.
1	Nitric oxide: a pro-inflammatory mediator in l...	Inflammatory diseases of the respiratory tract...	2000-08-15	Respir Res	Vliet, Albert van der; Eiserich, Jason P; Cros...	PMC	10.1186/rr14	2000.
2	Surfactant protein-D and pulmonary host defense	Surfactant protein-D (SP-D) participates in th...	2000-08-25	Respir Res	Crouch, Erika C	PMC	10.1186/rr19	2000.

In [6]:
```
print("📊 BASIC ANALYSIS")
print("=" * 50)

# Yearly counts
yearly_counts = df_clean['year'].value_counts().sort_index()
print("📈 Publications by year:")
for year, count in yearly_counts.items():
    print(f"  {year}: {count:>6,} papers")

# Top journals (if journal column exists)
if 'journal' in df_clean.columns:
    top_journals = df_clean['journal'].value_counts().head(10)
    print("\n🏆 Top 10 journals:")
    for journal, count in top_journals.items():
        print(f"  {journal}: {count:>6,} papers")
else:
    print("\n❌ Journal column not available")

# Top words in titles
def get_top_words(text_series, n=20):
    all_text = ' '.join(text_series.dropna().astype(str))
    words = re.findall(r'\b[a-zA-Z]{3,}\b', all_text.lower())
    stop_words = {'the', 'and', 'of', 'in', 'to', 'for', 'with', 'on', 'as', 'by',
    words = [word for word in words if word not in stop_words]
    return Counter(words).most_common(n)

top_words = get_top_words(df_clean['title'])
print(f"\n🔤 Top 20 words in titles:")
for i, (word, count) in enumerate(top_words, 1):
    print(f"  {i:2d}. {word:15} : {count:>6,}")

# Abstract statistics
print(f"\n📝 Abstract statistics:")
```

```
print(f"  Average word count: {df_clean['abstract_word_count'].mean():.1f}")
print(f"  Maximum word count: {df_clean['abstract_word_count'].max()}")
print(f"  Minimum word count: {df_clean['abstract_word_count'].min()}")
print(f"  Papers with abstracts: {(df_clean['abstract'].notna().sum()):>6,}")
```

📊 BASIC ANALYSIS
==
📈 Publications by year:
1856.0: 3 papers
1857.0: 1 papers
1860.0: 2 papers
1864.0: 1 papers
1876.0: 1 papers
1878.0: 1 papers
1879.0: 1 papers
1900.0: 1 papers
1902.0: 1 papers
1903.0: 1 papers
1955.0: 1 papers
1957.0: 2 papers
1961.0: 1 papers
1962.0: 1 papers
1963.0: 1 papers
1964.0: 2 papers
1965.0: 1 papers
1967.0: 1 papers
1968.0: 1 papers
1969.0: 3 papers
1970.0: 10 papers
1971.0: 8 papers
1972.0: 13 papers
1973.0: 11 papers
1974.0: 14 papers
1975.0: 20 papers
1976.0: 17 papers
1977.0: 24 papers
1978.0: 26 papers
1979.0: 19 papers
1980.0: 31 papers
1981.0: 42 papers
1982.0: 39 papers
1983.0: 38 papers
1984.0: 68 papers
1985.0: 73 papers
1986.0: 82 papers
1987.0: 89 papers
1988.0: 94 papers
1989.0: 108 papers
1990.0: 116 papers
1991.0: 124 papers
1992.0: 156 papers
1993.0: 103 papers
1994.0: 104 papers
1995.0: 101 papers
1996.0: 101 papers
1997.0: 112 papers
1998.0: 152 papers
1999.0: 182 papers
2000.0: 212 papers
2001.0: 200 papers
2002.0: 703 papers

```
2003.0:    835 papers
2004.0:  1,632 papers
2005.0:  1,553 papers
2006.0:  1,744 papers
2007.0:  1,682 papers
2008.0:  2,190 papers
2009.0:  2,542 papers
2010.0:  2,223 papers
2011.0:  2,337 papers
2012.0:  2,471 papers
2013.0:  2,948 papers
2014.0:  3,207 papers
2015.0:  3,461 papers
2016.0:  3,944 papers
2017.0:  3,691 papers
2018.0:  3,982 papers
2019.0:  5,629 papers
2020.0: 164,537 papers
2021.0: 219,335 papers
2022.0: 85,265 papers
2023.0:      1 papers
2024.0:      1 papers
```

🏆 Top 10 journals:

```
PLoS One:  9,953 papers
bioRxiv:  8,961 papers
Int J Environ Res Public Health:  8,201 papers
BMJ:  6,928 papers
Sci Rep:  5,935 papers
Cureus:  4,212 papers
Reactions Weekly:  3,891 papers
Front Psychol:  3,541 papers
BMJ Open:  3,515 papers
Front Immunol:  3,442 papers
```

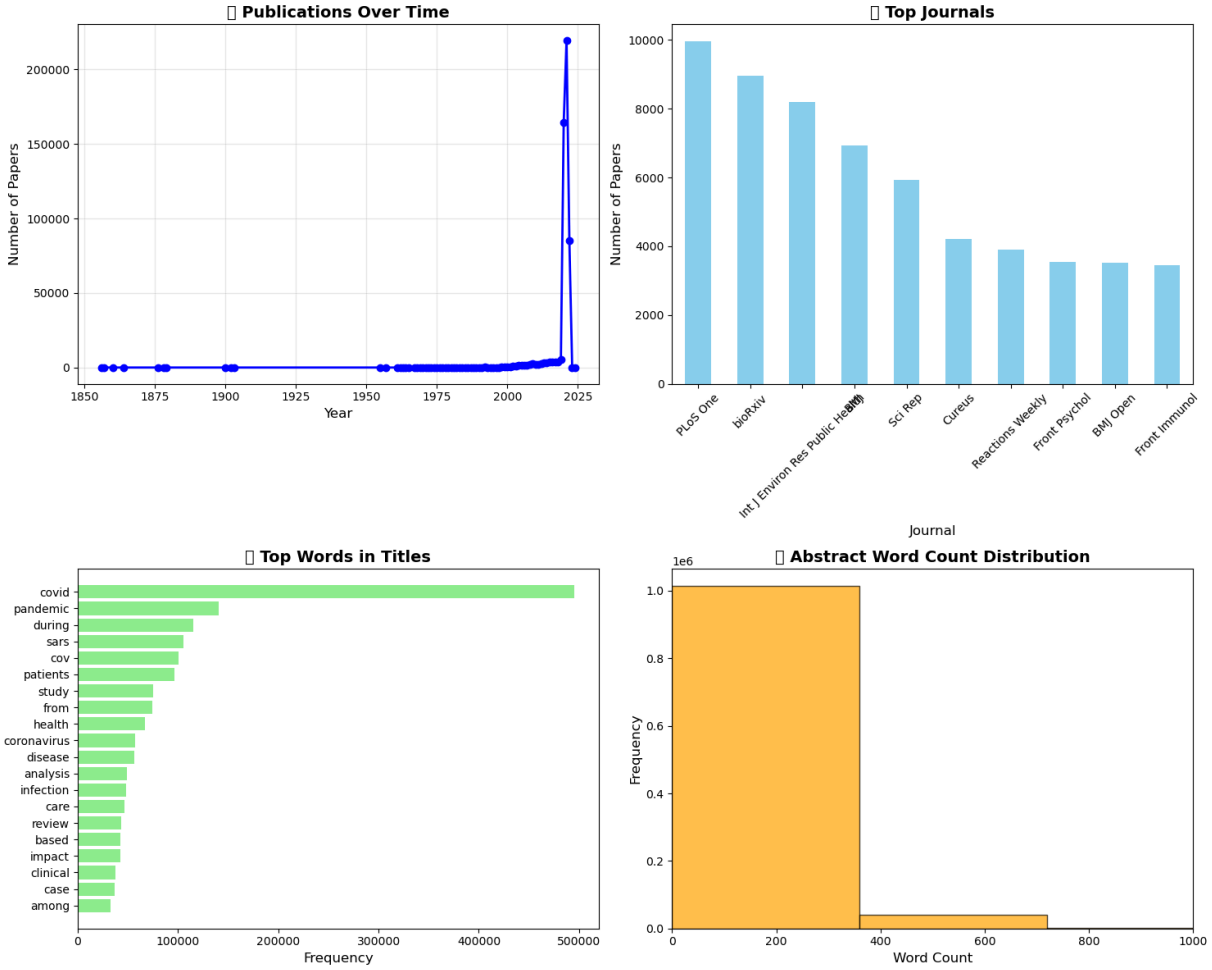
🔤 Top 20 words in titles:

```
 1. covid           : 494,829
 2. pandemic        : 140,316
 3. during          : 115,644
 4. sars            : 105,827
 5. cov             : 100,508
 6. patients        : 96,714
 7. study           : 75,469
 8. from            : 74,208
 9. health          : 66,883
10. coronavirus     : 57,631
11. disease         : 56,448
12. analysis        : 49,067
13. infection       : 48,050
14. care            : 46,372
15. review          : 43,466
16. based           : 42,292
17. impact          : 42,202
18. clinical        : 37,298
19. case            : 36,758
20. among           : 32,350
```

```
📝 Abstract statistics:
  Average word count: 164.7
  Maximum word count: 18000
  Minimum word count: 0
  Papers with abstracts: 821,005
```

In [7]:
```python
print("📈 VISUALIZATION 1: Publication Trends")
print("=" * 50)

# Create subplots
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

# Plot 1: Time series
axes[0, 0].plot(yearly_counts.index, yearly_counts.values, marker='o', linewidth=2,
axes[0, 0].set_title('📈 Publications Over Time', fontsize=14, fontweight='bold')
axes[0, 0].set_xlabel('Year', fontsize=12)
axes[0, 0].set_ylabel('Number of Papers', fontsize=12)
axes[0, 0].grid(True, alpha=0.3)

# Plot 2: Top journals (if available)
if 'journal' in df_clean.columns:
    top_journals = df_clean['journal'].value_counts().head(10)
    top_journals.plot(kind='bar', ax=axes[0, 1], color='skyblue')
    axes[0, 1].set_title('🏆 Top Journals', fontsize=14, fontweight='bold')
    axes[0, 1].set_xlabel('Journal', fontsize=12)
    axes[0, 1].set_ylabel('Number of Papers', fontsize=12)
    axes[0, 1].tick_params(axis='x', rotation=45)
else:
    axes[0, 1].text(0.5, 0.5, 'Journal data not available', ha='center', va='center
    axes[0, 1].set_title('Journal Data Not Available')

# Plot 3: Word frequency
words, counts = zip(*top_words)
axes[1, 0].barh(words[::-1], counts[::-1], color='lightgreen')
axes[1, 0].set_title('🔤 Top Words in Titles', fontsize=14, fontweight='bold')
axes[1, 0].set_xlabel('Frequency', fontsize=12)

# Plot 4: Abstract word count distribution
axes[1, 1].hist(df_clean['abstract_word_count'], bins=50, alpha=0.7, color='orange'
axes[1, 1].set_title('📝 Abstract Word Count Distribution', fontsize=14, fontweight
axes[1, 1].set_xlabel('Word Count', fontsize=12)
axes[1, 1].set_ylabel('Frequency', fontsize=12)
axes[1, 1].set_xlim(0, 1000)

plt.tight_layout()
plt.show()

# Save the figure
plt.savefig('analysis_results.png', dpi=300, bbox_inches='tight')
print("✅ Saved analysis_results.png")
```

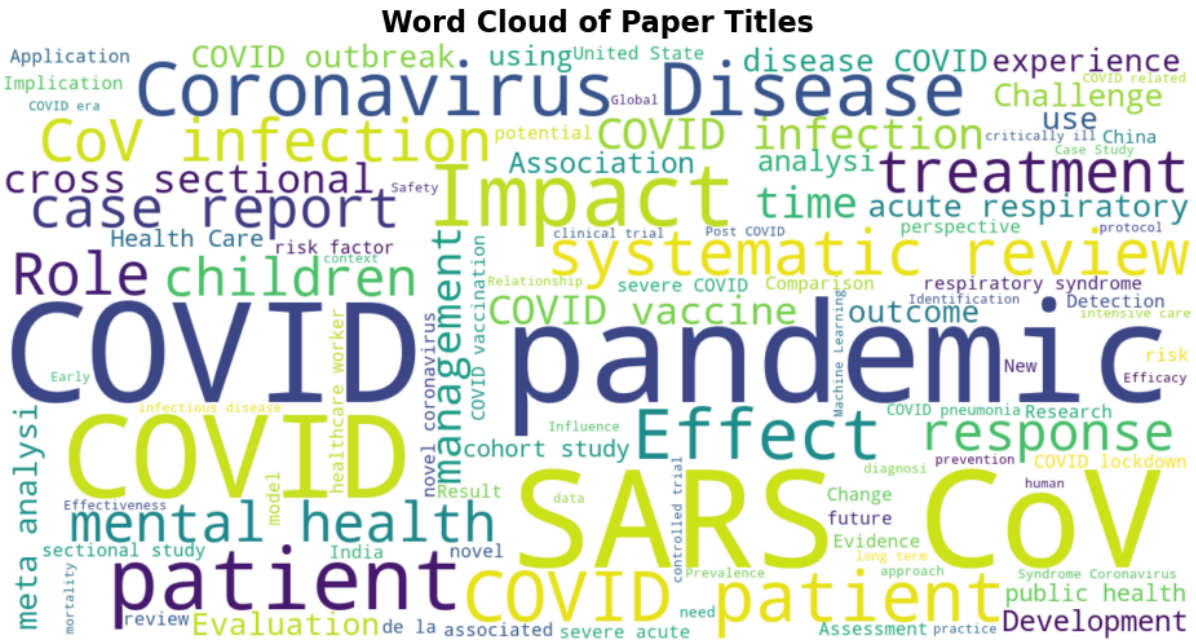
```
📈 VISUALIZATION 1: Publication Trends
==================================================
```



✅ Saved analysis_results.png

In [8]:
```
print("☁ VISUALIZATION 2: Word Cloud")
print("=" * 50)

# Generate word cloud from titles
text = ' '.join(df_clean['title'].dropna().astype(str))
wordcloud = WordCloud(width=1200, height=600, background_color='white',
                      max_words=100, colormap='viridis').generate(text)

# Display word cloud
plt.figure(figsize=(12, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Paper Titles', fontsize=16, fontweight='bold')
plt.show()

# Save the word cloud
plt.savefig('wordcloud.png', dpi=300, bbox_inches='tight')
print("✅ Saved wordcloud.png")
```

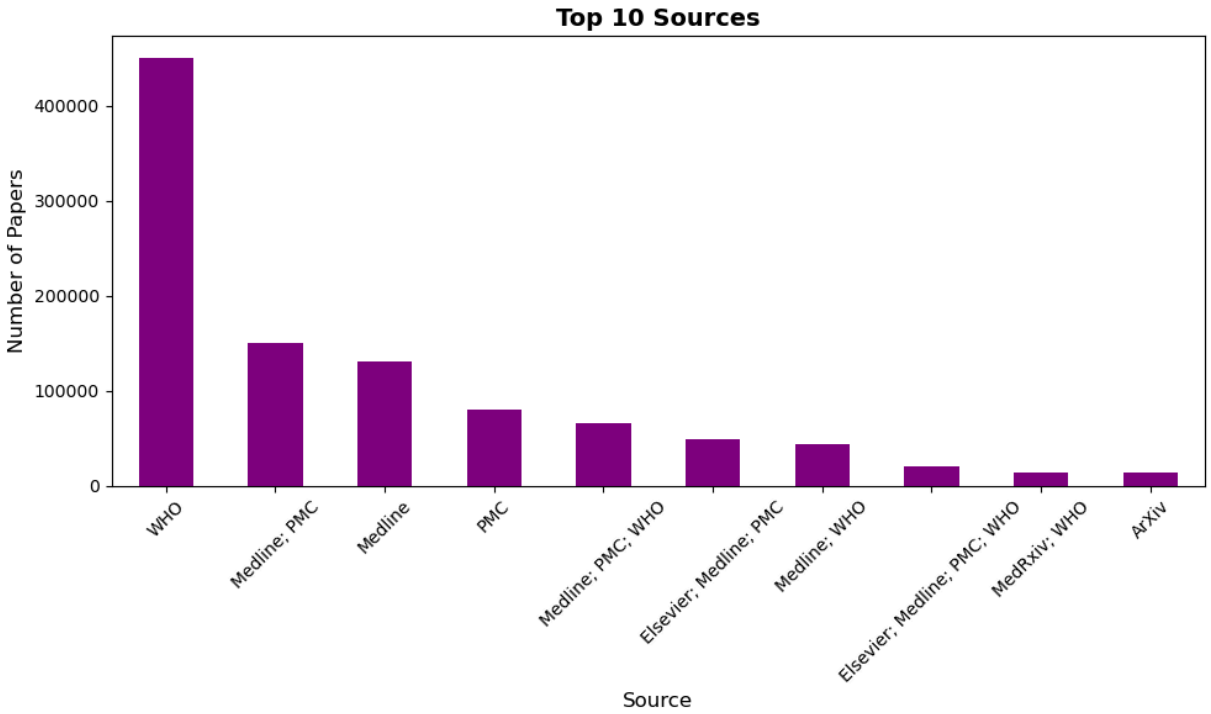
```
☁ VISUALIZATION 2: Word Cloud
==================================================
<Figure size 640x480 with 0 Axes>
```



✅ Saved wordcloud.png

In [9]:
```
print("📊 ADVANCED ANALYSIS: Source Analysis")
print("=" * 50)

# Check if source_x column exists
if 'source_x' in df_clean.columns:
    source_counts = df_clean['source_x'].value_counts().head(10)

    plt.figure(figsize=(10, 6))
    source_counts.plot(kind='bar', color='purple')
    plt.title('Top 10 Sources', fontsize=14, fontweight='bold')
    plt.xlabel('Source', fontsize=12)
    plt.ylabel('Number of Papers', fontsize=12)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    print("Top sources:")
    for source, count in source_counts.items():
        print(f"  {source}: {count:>6,} papers")
else:
    print("❌ Source column not available")
```

```
📊 ADVANCED ANALYSIS: Source Analysis
==================================================
<Figure size 640x480 with 0 Axes>
```

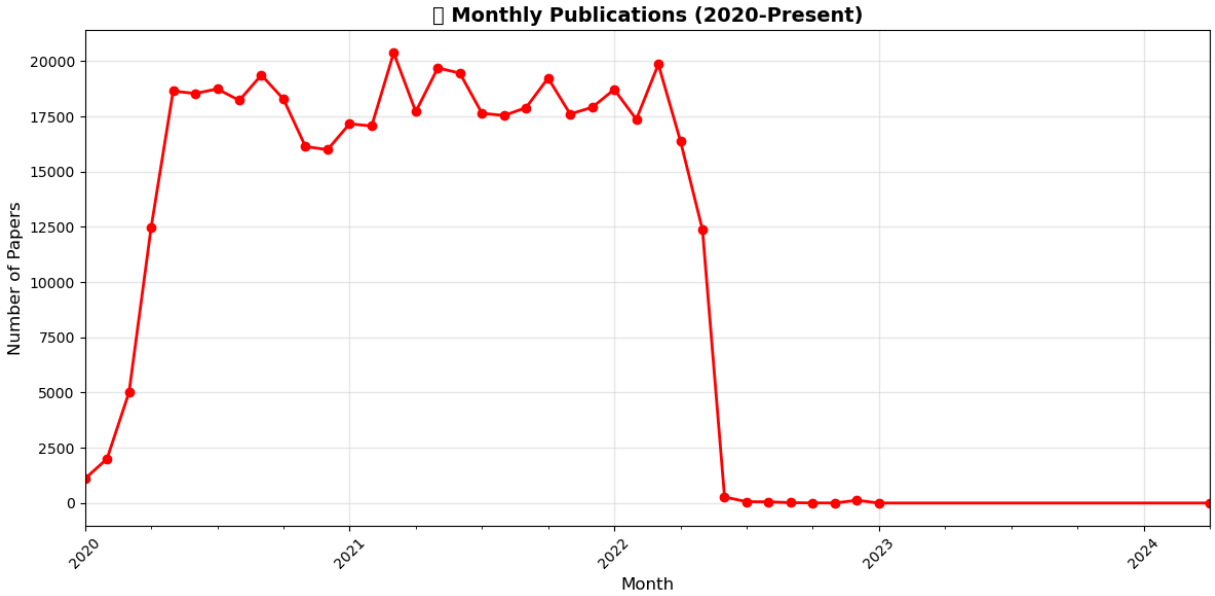



```
Top sources:
  WHO: 450,459 papers
  Medline; PMC: 150,592 papers
  Medline: 131,110 papers
  PMC: 80,189 papers
  Medline; PMC; WHO: 65,968 papers
  Elsevier; Medline; PMC: 49,314 papers
  Medline; WHO: 43,855 papers
  Elsevier; Medline; PMC; WHO: 19,793 papers
  MedRxiv; WHO: 13,756 papers
  ArXiv: 13,176 papers
```

In [10]:
```python
print("⏰ TIME-BASED ANALYSIS")
print("=" * 50)

# Analyze recent trends (2020 onwards)
recent_data = df_clean[df_clean['year'] >= 2020].copy()

if not recent_data.empty:
    recent_data['month_year'] = recent_data['publish_time'].dt.to_period('M')
    monthly_counts = recent_data['month_year'].value_counts().sort_index()

    plt.figure(figsize=(12, 6))
    monthly_counts.plot(kind='line', marker='o', linewidth=2, color='red')
    plt.title('📈 Monthly Publications (2020-Present)', fontsize=14, fontweight='bo
    plt.xlabel('Month', fontsize=12)
    plt.ylabel('Number of Papers', fontsize=12)
    plt.grid(True, alpha=0.3)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    print("Monthly publication statistics (2020-Present):")
    print(f"Total months: {len(monthly_counts)}")
```

```
    print(f"Average per month: {monthly_counts.mean():.1f}")
    print(f"Maximum in a month: {monthly_counts.max()}")
    print(f"Minimum in a month: {monthly_counts.min()}")
else:
    print("❌ No recent data available (2020+)")
```

⏰ TIME-BASED ANALYSIS
==



Monthly publication statistics (2020-Present):
Total months: 38
Average per month: 12345.8
Maximum in a month: 20381
Minimum in a month: 1

In [11]:
```
print("🔍 INTERACTIVE EXPLORATION")
print("=" * 50)

# Function to explore data by year
def explore_by_year(selected_year):
    year_data = df_clean[df_clean['year'] == selected_year]

    print(f"📊 Year {selected_year}: {len(year_data)} papers")

    if len(year_data) > 0:
        # Top words for selected year
        year_top_words = get_top_words(year_data['title'], n=10)
        print("\n🔤 Top words this year:")
        for word, count in year_top_words:
            print(f"  {word}: {count}")

        # Journal distribution
        if 'journal' in year_data.columns:
            top_journals_year = year_data['journal'].value_counts().head(5)
            if len(top_journals_year) > 0:
                print("\n🏆 Top journals this year:")
                for journal, count in top_journals_year.items():
                    print(f"  {journal}: {count}")
```

```
        # Show sample papers
        print(f"\n📋 Sample papers from {selected_year}:")
        for i, (idx, row) in enumerate(year_data.head(3).iterrows(), 1):
            print(f"{i}. {row['title'][:100]}...")
            if pd.notnull(row.get('journal')):
                print(f"   Journal: {row['journal']}")
            print()

# Get available years for dropdown
available_years = sorted(df_clean['year'].dropna().unique())
print("Available years:", available_years)

# You can manually call the function with different years
print("\nTry exploring different years:")
print("Example: explore_by_year(2020)")
```

```
🔍 INTERACTIVE EXPLORATION
==================================================
Available years: [1856.0, 1857.0, 1860.0, 1864.0, 1876.0, 1878.0, 1879.0, 1900.0, 19
02.0, 1903.0, 1955.0, 1957.0, 1961.0, 1962.0, 1963.0, 1964.0, 1965.0, 1967.0, 1968.
0, 1969.0, 1970.0, 1971.0, 1972.0, 1973.0, 1974.0, 1975.0, 1976.0, 1977.0, 1978.0, 1
979.0, 1980.0, 1981.0, 1982.0, 1983.0, 1984.0, 1985.0, 1986.0, 1987.0, 1988.0, 1989.
0, 1990.0, 1991.0, 1992.0, 1993.0, 1994.0, 1995.0, 1996.0, 1997.0, 1998.0, 1999.0, 2
000.0, 2001.0, 2002.0, 2003.0, 2004.0, 2005.0, 2006.0, 2007.0, 2008.0, 2009.0, 2010.
0, 2011.0, 2012.0, 2013.0, 2014.0, 2015.0, 2016.0, 2017.0, 2018.0, 2019.0, 2020.0, 2
021.0, 2022.0, 2023.0, 2024.0]

Try exploring different years:
Example: explore_by_year(2020)
```

In [12]:

```
print("💾 SAVING RESULTS AND SUMMARY")
print("=" * 50)

# Save cleaned data
df_clean.to_csv('cleaned_cord19_metadata.csv', index=False)
print("✅ Saved cleaned_cord19_metadata.csv")

# Save summary statistics
summary = {
    'total_papers': len(df_clean),
    'years_covered': f"{df_clean['year'].min()} - {df_clean['year'].max()}",
    'papers_with_abstracts': df_clean['abstract'].notna().sum(),
    'unique_journals': df_clean['journal'].nunique() if 'journal' in df_clean.colum
    'average_abstract_length': df_clean['abstract_word_count'].mean()
}

print("📊 SUMMARY STATISTICS:")
for key, value in summary.items():
    print(f"  {key.replace('_', ' ').title()}: {value}")

print("\n🎉 Analysis complete! Files created:")
print("   - analysis_results.png")
print("   - wordcloud.png")
print("   - cleaned_cord19_metadata.csv")
```

```
💾 SAVING RESULTS AND SUMMARY
==================================================
✅ Saved cleaned_cord19_metadata.csv
📊 SUMMARY STATISTICS:
  Total Papers: 1056157
  Years Covered: 1856.0 - 2024.0
  Papers With Abstracts: 821005
  Unique Journals: 54993
  Average Abstract Length: 164.6692025901452

🎉 Analysis complete! Files created:
   - analysis_results.png
   - wordcloud.png
   - cleaned_cord19_metadata.csv
```

In [13]:

```python
# Cell 1: Create the app.py file using Python code
with open('app.py', 'w', encoding='utf-8') as f:
    f.write('''
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt

# Simple CORD-19 Explorer App
st.title("📊 CORD-19 Research Explorer")
st.write("Exploring COVID-19 research papers made easy!")

# Load your data
try:
    df = pd.read_csv("C:\\\\Users\\\\pc\\\\Downloads\\\\metadata.csv")
    st.success("✅ Data loaded successfully!")

    # Show basic information
    st.subheader("📈 Dataset Overview")
    col1, col2, col3 = st.columns(3)
    with col1:
        st.metric("Total Papers", f"{len(df):,}")
    with col2:
        st.metric("Columns", len(df.columns))
    with col3:
        st.metric("File Size", f"{(df.memory_usage().sum() / 1024 / 1024):.1f} MB")

    # Show sample data
    st.subheader("📋 Sample Data (First 10 rows)")
    st.dataframe(df.head(10))

    # Simple analysis - publications by year
    st.subheader("📅 Publications by Year")

    # Extract year from publish_time
    df['publish_time'] = pd.to_datetime(df['publish_time'], errors='coerce')
    df['year'] = df['publish_time'].dt.year

    # Count publications by year
    yearly_counts = df['year'].value_counts().sort_index()

    # Create bar chart
```

```
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.bar(yearly_counts.index.astype(str), yearly_counts.values, color='skyblue')
    ax.set_xlabel('Year')
    ax.set_ylabel('Number of Papers')
    ax.set_title('Research Publications Over Time')
    plt.xticks(rotation=45)
    st.pyplot(fig)

except Exception as e:
    st.error(f"❌ Error loading data: {str(e)}")
''')

print("✅ app.py file created successfully!")
```

```
✅ app.py file created successfully!
```

In [ ]:
```
# Run the Streamlit app
!streamlit run app.py
```

In [ ]:

In [ ]: