LinkedIn Job Postings Analysis

This notebook explores a rich LinkedIn job postings dataset spanning 2023 and 2024 with over 280,000 job records and 35 columns.. The analysis covers various angles including job market dynamics, compensation, work types, application metrics, skills demand, and employer profiles.

Key Areas of Analysis

- Job Market Dynamics:
  - Analysis of job posting volume by location, industry, and company size to identify trends and hotspots for talent demand.
- Work Type and Flexibility:
  - Exploration of remote work availability and work types offered to understand the shifting landscape of job flexibility.
- Skill Demand:
  - Assessment of skills mentioned in job postings to highlight emerging and in-demand competencies.
- Geographic and Market Insights:
  - Visualization and analysis of geographic distribution and concentration of job postings, average job views per region, and location parsing/filling.

This integrated dataset and the analysis provide valuable insights to job seekers, employers, and policymakers about the labor market conditions, competitive employer landscapes, and evolving workforce demands in 2023-2024.

Questions Answered in the Notebook

1. Data Overview and Quality Assessment

- What are the overall size and structure of the dataset (rows, columns, data types)?
- Which columns have missing or null data, and what is the extent of missingness?
- Are there duplicate columns or major data quality issues to address?
- How to effectively convert date columns and handle time-related features?

2. Data Cleaning and Imputation

- How to handle missing location data by parsing "location" into city, state, and country?
- How many jobs have null or incomplete geographic data, and can this be filled or dropped?
- Which columns to drop due to high missing value percentages?
- How to impute missing values in numeric columns (median) and categorical columns (mode, forward/backward fill)?
- How to detect and cap numeric outliers to reduce skewness and extreme values?

3. Univariate Exploratory Data Analysis (EDA)

- What are the distributions of key categorical variables such as:
  - Work Type
  - Experience Level
  - Application Type
  - Skills
  - Industry
  - Companies
- What are the distributions and outlier patterns in numeric variables like views, job IDs, company IDs?
- What are the most common job titles and skills demanded in the dataset?
- What industries dominate job postings, and what are their frequencies?

4. Bivariate Analysis and Visualizations

- How does job posting frequency vary by work type across experience levels and vice versa?
- What is the distribution of work types and experience levels across top cities and states (stacked bar charts)?
- How do skill demands vary in the top cities and states?
- What industries dominate in the top cities and states (pie charts showing percentage shares)?
- How do job types distribute across countries (interactive grouped bar chart)?

5. Geographic and Market Insights

- What are the top locations (cities, states, countries) by job posting count?
- How are job postings geographically distributed (bar charts and pie charts)?
- What is the average number of job post views by state, indicating job attractiveness?

6. Word Cloud Visualizations

- What are the common job titles, skill keywords, and industries in the posting data visualized through word clouds?

1. Import Libraries and Load Data

In [1]:
```
# Import the libraries

import pandas as pd
import numpy as np
import os

import warnings
warnings.filterwarnings('ignore')
```

Reading the dataset

In [2]:
```
# Load the data into dataset

df = pd.read_csv('merged_output.csv')
```

In [3]:
```
# Read the first few rows of the dataframe
df.head()
```

Out[3]:

	job_id	company_name	title	description	max_salary	pay_period	location	company_id	views	med_salary	...	work_type	cur
0	921716	Corcoran Sawyer Smith	Marketing Coordinator	Job descriptionA leading real estate firm in N...	20.0	HOURLY	Princeton, NJ	2774458.0	20.0	NaN	...	FULL_TIME	
1	921716	Corcoran Sawyer Smith	Marketing Coordinator	Job descriptionA leading real estate firm in N...	20.0	HOURLY	Princeton, NJ	2774458.0	20.0	NaN	...	FULL_TIME	
2	1829192	NaN	Mental Health Therapist/Counselor	At Aspen Therapy and Wellness , we are committ...	50.0	HOURLY	Fort Collins, CO	NaN	1.0	NaN	...	FULL_TIME	
3	10998357	The National Exemplar	Assitant Restaurant Manager	The National Exemplar is accepting application...	65000.0	YEARLY	Cincinnati, OH	64896719.0	8.0	NaN	...	FULL_TIME	
4	10998357	The National Exemplar	Assitant Restaurant Manager	The National Exemplar is accepting application...	65000.0	YEARLY	Cincinnati, OH	64896719.0	8.0	NaN	...	FULL_TIME	

5 rows × 35 columns

2. Initial Data Inspection

In [4]:
```
# View the total number of rows and columns in the dataset
df.shape
```

Out[4]:
```
(280665, 35)
```

Large Dataset Size

- The dataset's size (280,665 records) suggests a **very large volume** of job postings, which is ideal for robust analysis and building machine learning models with diverse patterns.

Wide Feature Set

- With 35 columns, there appears to be a **wide range of features available**—potentially covering job descriptions, company information, skills, location, application types, and time-related details.

In [5]:
```
# View all the column names in the dataset
df.columns
```

Out[5]:
```
Index(['job_id', 'company_name', 'title', 'description', 'max_salary',
       'pay_period', 'location', 'company_id', 'views', 'med_salary',
       'min_salary', 'formatted_work_type', 'applies', 'original_listed_time',
       'remote_allowed', 'job_posting_url', 'application_url',
       'application_type', 'expiry', 'closed_time',
       'formatted_experience_level', 'skills_desc', 'listed_time',
       'posting_domain', 'sponsored', 'work_type', 'currency',
       'compensation_type', 'normalized_salary', 'zip_code', 'fips',
       'skill_abr', 'skill_name', 'industry_id', 'industry_name'],
      dtype='object')
```

In [6]:
```
# View the information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 280665 entries, 0 to 280664
Data columns (total 35 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   job_id                      280665 non-null  int64
 1   company_name                277353 non-null  object
 2   title                       280665 non-null  object
 3   description                 280653 non-null  object
 4   max_salary                  73215 non-null   float64
 5   pay_period                  86031 non-null   object
 6   location                    280665 non-null  object
 7   company_id                  277357 non-null  float64
 8   views                       275882 non-null  float64
 9   med_salary                  12816 non-null   float64
 10  min_salary                  73215 non-null   float64
 11  formatted_work_type         280665 non-null  object
 12  applies                     58468 non-null   float64
 13  original_listed_time        280665 non-null  float64
 14  remote_allowed              39122 non-null   float64
 15  job_posting_url             280665 non-null  object
 16  application_url             187185 non-null  object
 17  application_type            280665 non-null  object
 18  expiry                      280665 non-null  float64
 19  closed_time                 1637 non-null    float64
 20  formatted_experience_level  216729 non-null  object
 21  skills_desc                 4504 non-null    object
 22  listed_time                 280665 non-null  float64
 23  posting_domain              173094 non-null  object
 24  sponsored                   280665 non-null  int64
 25  work_type                   280665 non-null  object
 26  currency                    86031 non-null   object
 27  compensation_type           86031 non-null   object
 28  normalized_salary           86031 non-null   float64
 29  zip_code                    226961 non-null  float64
 30  fips                        212471 non-null  float64
 31  skill_abr                   278898 non-null  object
 32  skill_name                  278898 non-null  object
 33  industry_id                 278340 non-null  float64
 34  industry_name               278161 non-null  object
dtypes: float64(15), int64(2), object(18)
memory usage: 74.9+ MB
```

Key Insights from Dataset Structure:

- **Large Dataset:** Over 280k rows, suitable for extensive analysis.
- **Missing Data:** Significant missing values in salary columns (max_salary, min_salary, med_salary), pay_period, applies, and remote_allowed columns, which may affect salary and application analyses.
- **Data Types:** Mix of numeric and categorical columns, with salary-related columns as floats and job/company attributes as objects.
- **Application Metrics:** 'applies' and 'views' columns give insights into job popularity but are sparsely populated.
- **Remote Jobs:** remote_allowed has many missing values; remote work info might be incomplete.
- **Industry Info:** 'industry_id' and 'industry_name' are mostly complete and can help segment analysis by sectors.
- **Time Fields:** original_listed_time, expiry, closed_time can help analyze job listing duration and expiry.

In [7]:
```
# Print Unique values in each column
for col in df.columns:
    #print(f"Unique values in '{col}': {df[col].nunique()}")
    print(f"Unique values in {col:<27} : {df[col].nunique()}")
    #print(df[col].value_counts())
    #print("\n")
```

```
Unique values in job_id                      : 123849
Unique values in company_name                : 24428
Unique values in title                       : 72521
Unique values in description                 : 107827
Unique values in max_salary                  : 5321
Unique values in pay_period                  : 5
Unique values in location                    : 8526
Unique values in company_id                  : 24474
Unique values in views                       : 684
Unique values in med_salary                  : 1417
Unique values in min_salary                  : 4612
Unique values in formatted_work_type         : 7
Unique values in applies                     : 274
Unique values in original_listed_time        : 65036
Unique values in remote_allowed              : 1
Unique values in job_posting_url             : 123849
Unique values in application_url             : 84800
Unique values in application_type            : 4
Unique values in expiry                      : 54851
Unique values in closed_time                 : 698
Unique values in formatted_experience_level  : 6
Unique values in skills_desc                 : 2212
Unique values in listed_time                 : 53231
Unique values in posting_domain              : 4443
Unique values in sponsored                   : 1
Unique values in work_type                   : 7
Unique values in currency                    : 6
Unique values in compensation_type           : 1
Unique values in normalized_salary           : 7259
Unique values in zip_code                    : 6989
Unique values in fips                        : 2270
Unique values in skill_abr                   : 35
Unique values in skill_name                  : 35
Unique values in industry_id                 : 421
Unique values in industry_name               : 387
```

Key Insights:

- The dataset has a large scale with over **123,000 unique job postings** and about **24,000 unique companies**.
- There is **high variety in job titles (72k+) and descriptions (107k+)**, reflecting diverse job opportunities.
- Salary information varies with thousands of unique values for max, median, and minimum salaries.
- **Geographic diversity** is notable with thousands of unique locations, zip codes, and industry classifications.
- Some features, like remote_allowed, sponsored, and compensation_type, have very **limited unique values**, indicating fixed or binary categories.

Data Type Conversion

In [8]:
```
# View the date column data before conversion
df[['closed_time', 'original_listed_time', 'listed_time']].head()
```

Out[8]:

	closed_time	original_listed_time	listed_time
0	NaN	1.713398e+12	1.713398e+12
1	NaN	1.713398e+12	1.713398e+12
2	NaN	1.712858e+12	1.712858e+12
3	NaN	1.713278e+12	1.713278e+12
4	NaN	1.713278e+12	1.713278e+12

Timestamp Conversion

In [9]:
```
# Convert date columns from scientific format to datetime format

df['original_listed_time'] = pd.to_datetime(df['original_listed_time'], unit='ms')
df['listed_time'] = pd.to_datetime(df['listed_time'], unit='ms')
df['closed_time'] = pd.to_datetime(df['closed_time'], unit='ms')
```

In [10]:
```
# View the date column data after conversion
df[['closed_time', 'original_listed_time', 'listed_time']].head(10)
```

Out[10]:

	closed_time	original_listed_time	listed_time
0	NaT	2024-04-17 23:45:08	2024-04-17 23:45:08
1	NaT	2024-04-17 23:45:08	2024-04-17 23:45:08
2	NaT	2024-04-11 17:51:27	2024-04-11 17:51:27
3	NaT	2024-04-16 14:26:54	2024-04-16 14:26:54
4	NaT	2024-04-16 14:26:54	2024-04-16 14:26:54
5	NaT	2024-04-12 04:23:32	2024-04-12 04:23:32
6	NaT	2024-04-18 14:52:23	2024-04-18 14:52:23
7	NaT	2024-04-18 16:01:39	2024-04-18 16:01:39
8	NaT	2024-04-11 18:43:39	2024-04-11 18:43:39
9	NaT	2024-04-11 18:43:39	2024-04-11 18:43:39

In [11]:
```
# Extract the date from datetime columns

df['closed_time'] = df['closed_time'].dt.date
df['original_listed_time'] = df['original_listed_time'].dt.date
df['listed_time'] = df['listed_time'].dt.date
```

In [12]:
```
# Convert date column from object to datetime fromat
df['original_listed_time'] = pd.to_datetime(df['original_listed_time'])
df['listed_time'] = pd.to_datetime(df['listed_time'])
df['closed_time'] = pd.to_datetime(df['closed_time'])
```

In [ ]:
```
# Check the result after extraction
df[['closed_time', 'original_listed_time', 'listed_time']].head()
```

Out[ ]:

	closed_time	original_listed_time	listed_time
0	NaT	2024-04-17	2024-04-17
1	NaT	2024-04-17	2024-04-17
2	NaT	2024-04-11	2024-04-11
3	NaT	2024-04-16	2024-04-16
4	NaT	2024-04-16	2024-04-16

In [ ]:
```
# Check the data type
df[['closed_time', 'original_listed_time', 'listed_time']].dtypes
```

Out[ ]:
```
closed_time             datetime64[ns]
original_listed_time    datetime64[ns]
listed_time             datetime64[ns]
dtype: object
```

In [15]:
```
# View the value count of date column
# print(df['closed_time'].value_counts())
# print(df['original_listed_time'].value_counts())
# print(df['listed_time'].value_counts())
```

Unique Values of Categorical columns

In [16]:
```
# List of key categorical columns to check
categorical_columns = [
    'company_name', 'title', 'location', 'formatted_work_type',
    'application_type', 'formatted_experience_level', 'posting_domain',
    'work_type', 'currency', 'compensation_type', 'skill_abr',
    'skill_name', 'industry_name'
]

# Loop through each column and print unique value counts
for col in categorical_columns:
    unique_count = df[col].nunique(dropna=True)  # exclude NaN in count
    print(f"Unique values in {col:<27}: {unique_count}")
```

```
Unique values in company_name               : 24428
Unique values in title                      : 72521
Unique values in location                   : 8526
Unique values in formatted_work_type        : 7
Unique values in application_type           : 4
Unique values in formatted_experience_level : 6
Unique values in posting_domain             : 4443
Unique values in work_type                  : 7
Unique values in currency                   : 6
Unique values in compensation_type          : 1
Unique values in skill_abr                  : 35
Unique values in skill_name                 : 35
Unique values in industry_name              : 387
```

- **Job Popularity:** With so many unique job titles, analysis can reveal trending roles and rare specialties.
- **Market Mapping:** High counts for company, industry, and location support robust regional and sectoral job market mapping.
- **Role Standardization:** Relatively few formatted work types and experience levels allow for effective grouping and aggregation for reporting.
- **Platform Diversity:** Extensive posting domains enable comparison of job listing effectiveness across online platforms.

3. Data Cleaning

Missing Values

In [17]:
```
# Check for any duplicate columns in the merged dataset
duplicate_columns = df.columns[df.columns.duplicated()]
duplicate_columns
```

Out[17]:
```
Index([], dtype='object')
```

Hence, No duplicate column names were found

In [100…
```
# df[['formatted_work_type','work_type', 'skill_abr', 'skill_name']].head()
```

In [19]:
```
# Drop skill_abr and work_type

df.drop(columns=['skill_abr', 'work_type'], inplace=True)
```

NULL Values

In [20]:
```
# Check for Null values of complete dataset
df.isnull().sum()
```

Out[20]:
```
job_id                        0
company_name               3312
title                         0
description                  12
max_salary                207450
pay_period                194634
location                      0
company_id                 3308
views                      4783
med_salary                267849
min_salary                207450
formatted_work_type           0
applies                   222197
original_listed_time          0
remote_allowed            241543
job_posting_url               0
application_url            93480
application_type              0
expiry                        0
closed_time               279028
formatted_experience_level 63936
skills_desc               276161
listed_time                   0
posting_domain            107571
sponsored                     0
currency                  194634
compensation_type         194634
normalized_salary         194634
zip_code                   53704
fips                       68194
skill_name                  1767
industry_id                 2325
industry_name               2504
dtype: int64
```

In [21]:
```
# List out the columns in the dataset where Only NULL values are present

null_columns=df.columns[df.isnull().any()]
null_columns
```

Out[21]:
```
Index(['company_name', 'description', 'max_salary', 'pay_period', 'company_id',
       'views', 'med_salary', 'min_salary', 'applies', 'remote_allowed',
       'application_url', 'closed_time', 'formatted_experience_level',
       'skills_desc', 'posting_domain', 'currency', 'compensation_type',
       'normalized_salary', 'zip_code', 'fips', 'skill_name', 'industry_id',
       'industry_name'],
      dtype='object')
```

In [22]:
```
# View the count of null values in each of these columns  and its percentage

null_data = pd.DataFrame({'Null Count': df[null_columns].isnull().sum()})
null_data['Percentage'] = (null_data['Null Count'] / len(df)) * 100
null_data['Data Type'] = df[null_columns].dtypes
null_data
```

Out[22]:

	Null Count	Percentage	Data Type
company_name	3312	1.180055	object
description	12	0.004276	object
max_salary	207450	73.913741	float64
pay_period	194634	69.347443	object
company_id	3308	1.178629	float64
views	4783	1.704167	float64
med_salary	267849	95.433702	float64
min_salary	207450	73.913741	float64
applies	222197	79.168047	float64
remote_allowed	241543	86.060962	float64
application_url	93480	33.306611	object
closed_time	279028	99.416742	datetime64[ns]
formatted_experience_level	63936	22.780183	object
skills_desc	276161	98.395240	object
posting_domain	107571	38.327187	object
currency	194634	69.347443	object
compensation_type	194634	69.347443	object
normalized_salary	194634	69.347443	float64
zip_code	53704	19.134555	float64
fips	68194	24.297294	float64
skill_name	1767	0.629576	object
industry_id	2325	0.828390	float64
industry_name	2504	0.892167	object

Highly incomplete data (Over 60% null)

- **closed_time (99.57% null, float64):** This column is almost entirely empty. Unless the records are meant to be open-ended, the data source may have a major issue. This column is likely unusable and should be dropped.
- **skills_desc (98.99% null, object):** With over 98% missing values, this column provides very little information. It is a strong candidate for deletion.
- **remote_allowed (88.51% null, float64):** The vast majority of this data is missing. Consider whether this feature is essential for your analysis. It may need to be dropped or used with extreme caution.
- **med_salary (94.52% null, float64), min_salary (69.81% null, float64), max_salary (69.81% null, float64), applies (78.98% null, float64):** The salary and application metrics are extremely incomplete. This could indicate a major data collection problem or that this information is optional. Using these columns for analysis would introduce significant bias.
- **pay_period (64.34% null, object), currency (64.34% null, object), compensation_type (64.34% null, object), normalized_salary (64.34% null, float64):** Missing the same proportion of data as the salary columns, these are likely part of the same data collection process. The null values are probably not random and should be handled with care.

Moderately incomplete data (15% to 35% null)

- **application_url (31.12% null, object), posting_domain (31.69% null, object):** The missing URLs suggest that a significant number of job postings may lack a direct application link. This could mean applications are handled via other means, such as the company's website.
- **fips (21.50% null, float64):** This geo-coding data is moderately incomplete. The nulls may be for international jobs or simply missing data.
- **formatted_experience_level (18.74% null, object):** Nearly one-fifth of the records are missing experience level. This is a crucial feature that could be imputed using other job posting information.
- **zip_code_x (15.88% null, float64):** The null values could correspond to international jobs or incomplete entries.

Minimally incomplete data (Under 5% null)

- **Job-related (company_name, company_id, description_x, skill_abr, skill_name):** These core job-related fields have negligible null percentages.
- **Company-related (industry_id, industry_name, name, description_y, company_size, state, country, city, zip_code_y, address, url, industry, employee_count, follower_count):** Most company profile information is nearly complete.
- **views (1.22% null, float64) and speciality (2.28% null, object):** These fields are also largely complete.
- **time_recorded (0.04% null, float64):** This is a very reliable timestamp.

In [23]:
```
# Separate columns based on null value percentage

high_null_cols = null_data[null_data['Percentage'] > 60]
moderate_null_cols = null_data[(null_data['Percentage'] >= 15) & (null_data['Percentage'] <= 40)]
low_null_cols = null_data[null_data['Percentage'] < 15]
```

In [24]:
```
# We use keys and axis=1 to stack them horizontally with meaningful column titles
combined_df = pd.concat([high_null_cols, moderate_null_cols, low_null_cols],
                        axis=1,
                        keys=['High Null Cols (> 60%)', 'Moderate Null Cols (15%-40%)', 'Low Null Cols (< 15%)'])

combined_df
```

Out[24]:

	High Null Cols (> 60%)			Moderate Null Cols (15%-40%)			Low Null Cols (< 15%)		
	Null Count	Percentage	Data Type	Null Count	Percentage	Data Type	Null Count	Percentage	Data Type
max_salary	207450.0	73.913741	float64	NaN	NaN	NaN	NaN	NaN	NaN
pay_period	194634.0	69.347443	object	NaN	NaN	NaN	NaN	NaN	NaN
med_salary	267849.0	95.433702	float64	NaN	NaN	NaN	NaN	NaN	NaN
min_salary	207450.0	73.913741	float64	NaN	NaN	NaN	NaN	NaN	NaN
applies	222197.0	79.168047	float64	NaN	NaN	NaN	NaN	NaN	NaN
remote_allowed	241543.0	86.060962	float64	NaN	NaN	NaN	NaN	NaN	NaN
closed_time	279028.0	99.416742	datetime64[ns]	NaN	NaN	NaN	NaN	NaN	NaN
skills_desc	276161.0	98.395240	object	NaN	NaN	NaN	NaN	NaN	NaN
currency	194634.0	69.347443	object	NaN	NaN	NaN	NaN	NaN	NaN
compensation_type	194634.0	69.347443	object	NaN	NaN	NaN	NaN	NaN	NaN
normalized_salary	194634.0	69.347443	float64	NaN	NaN	NaN	NaN	NaN	NaN
application_url	NaN	NaN	NaN	93480.0	33.306611	object	NaN	NaN	NaN
formatted_experience_level	NaN	NaN	NaN	63936.0	22.780183	object	NaN	NaN	NaN
posting_domain	NaN	NaN	NaN	107571.0	38.327187	object	NaN	NaN	NaN
zip_code	NaN	NaN	NaN	53704.0	19.134555	float64	NaN	NaN	NaN
fips	NaN	NaN	NaN	68194.0	24.297294	float64	NaN	NaN	NaN
company_name	NaN	NaN	NaN	NaN	NaN	NaN	3312.0	1.180055	object
description	NaN	NaN	NaN	NaN	NaN	NaN	12.0	0.004276	object
company_id	NaN	NaN	NaN	NaN	NaN	NaN	3308.0	1.178629	float64
views	NaN	NaN	NaN	NaN	NaN	NaN	4783.0	1.704167	float64
skill_name	NaN	NaN	NaN	NaN	NaN	NaN	1767.0	0.629576	object
industry_id	NaN	NaN	NaN	NaN	NaN	NaN	2325.0	0.828390	float64
industry_name	NaN	NaN	NaN	NaN	NaN	NaN	2504.0	0.892167	object

In [25]:
```
# View the High Null Columns (> 60%)
# df[high_null_cols.index].head()
```

In [26]:
```
# View the Moderate Null Columns (15% - 40%)

# df[moderate_null_cols.index].head()
```

In [27]:
```
# View the Low Null Columns (< 5%)

# df[low_null_cols.index].head()
```

In [28]:
```
# low_null_cols
```

Dropping High Null Values

In [29]:
```
# Remove columns with high null values (> 60%)

df.drop(columns=high_null_cols.index, inplace=True)
```

In [30]:
```
# Combine moderate and low null columns for further analysis
```

```
moderate_low_null_cols = pd.concat([moderate_null_cols, low_null_cols])
moderate_low_null_cols
```

Out[30]:

	Null Count	Percentage	Data Type
application_url	93480	33.306611	object
formatted_experience_level	63936	22.780183	object
posting_domain	107571	38.327187	object
zip_code	53704	19.134555	float64
fips	68194	24.297294	float64
company_name	3312	1.180055	object
description	12	0.004276	object
company_id	3308	1.178629	float64
views	4783	1.704167	float64
skill_name	1767	0.629576	object
industry_id	2325	0.828390	float64
industry_name	2504	0.892167	object

Handling Null Values

In [31]:
```
# Separate the Numeric and Categorical columns of moderate_low_null_cols

numeric_cols = df[moderate_low_null_cols.index].select_dtypes(include=[np.number]).columns.tolist()
categorical_cols = df[moderate_low_null_cols.index].select_dtypes(include=['object', 'category']).columns.tolist()

# Print the numeric and categorical columns
print("Columns with Moderate and Low Null Values:")
print("--"*25)
print(f"Numeric Column      :", numeric_cols)
print(f"Categorical Column  :", categorical_cols)
```

```
Columns with Moderate and Low Null Values:
--------------------------------------------------
Numeric Column      : ['zip_code', 'fips', 'company_id', 'views', 'industry_id']
Categorical Column  : ['application_url', 'formatted_experience_level', 'posting_domain', 'company_name', 'description', 'skill_name', 'in
dustry_name']
```

In [32]:
```
#df[numeric_cols].info(), df[categorical_cols].info()
```

In [ ]:
```
# View only the Numeric dataset
df[numeric_cols].head()
```

Out[ ]:

	zip_code	fips	company_id	views	industry_id
0	8540.0	34021.0	2774458.0	20.0	44.0
1	8540.0	34021.0	2774458.0	20.0	44.0
2	80521.0	8069.0	NaN	1.0	NaN
3	45202.0	39061.0	64896719.0	8.0	32.0
4	45202.0	39061.0	64896719.0	8.0	32.0

In [ ]:
```
# View only the Categorical dataset
df[categorical_cols].head()
```

Out[ ]:

	application_url	formatted_experience_level	posting_domain	company_name	description	skill_name	industry_name
0	NaN	NaN	NaN	Corcoran Sawyer Smith	Job descriptionA leading real estate firm in N...	Marketing	Real Estate
1	NaN	NaN	NaN	Corcoran Sawyer Smith	Job descriptionA leading real estate firm in N...	Sales	Real Estate
2	NaN	NaN	NaN	NaN	At Aspen Therapy and Wellness , we are committ...	Health Care Provider	NaN
3	NaN	NaN	NaN	The National Exemplar	The National Exemplar is accepting application...	Management	Restaurants
4	NaN	NaN	NaN	The National Exemplar	The National Exemplar is accepting application...	Manufacturing	Restaurants

Median Fill

In [35]:
```
# 1. Apply Median Imputation
for col in numeric_cols:
    df[col] = df[col].fillna(df[col].median())
```

```
print("\nDataFrame after filling nulls in numeric columns (mean imputation):")
df[numeric_cols].isnull().sum()
```

```
DataFrame after filling nulls in numeric columns (mean imputation):
```

Out[35]:
```
zip_code       0
fips           0
company_id     0
views          0
industry_id    0
dtype: int64
```

In [36]:
```
# Re-Check Null value count

df[moderate_low_null_cols.index].isnull().sum()
```

Out[36]:
```
application_url               93480
formatted_experience_level    63936
posting_domain               107571
zip_code                          0
fips                              0
company_name                   3312
description                      12
company_id                        0
views                             0
skill_name                     1767
industry_id                       0
industry_name                  2504
dtype: int64
```

Mode Fill

In [37]:
```
# Filling Null values in Categorical Columns for Less than 1% with Mode

df['company_name'] = df['company_name'].fillna(df['company_name'].mode()[0])
#df['skill_abr'] = df['skill_abr'].fillna(df['skill_abr'].mode()[0])
df['skill_name'] = df['skill_name'].fillna(df['skill_name'].mode()[0])
df['industry_name'] = df['industry_name'].fillna(df['industry_name'].mode()[0])
df['description'] = df['description'].fillna(df['description'].mode()[0])
```

In [38]:
```
# Check Null value count in Moderate and Low Null Columns after Mode Fill
df[moderate_low_null_cols.index].isnull().sum()/ len(df) * 100
```

Out[38]:
```
application_url               33.306611
formatted_experience_level    22.780183
posting_domain                38.327187
zip_code                       0.000000
fips                           0.000000
company_name                   0.000000
description                    0.000000
company_id                     0.000000
views                          0.000000
skill_name                     0.000000
industry_id                    0.000000
industry_name                  0.000000
dtype: float64
```

Backward/ Forward Fill

In [39]:
```
# Fill Null values in Categorical Columns with Backward Fill and Forward Fill
df['application_url'] = df['application_url'].bfill()
df['formatted_experience_level'] = df['formatted_experience_level'].ffill()
df['posting_domain'] = df['posting_domain'].bfill().ffill()
```

In [40]:
```
# Check Null value count in Moderate and Low Null Columns after Backward Fill and Forward Fill
# df[moderate_low_null_cols.index].isnull().sum()
```

In [41]:
```
# Fill the left Null values in Categorical Columns with Mode

df['application_url'] = df['application_url'].fillna(df['application_url'].mode()[0])
df['formatted_experience_level'] = df['formatted_experience_level'].fillna(df['formatted_experience_level'].mode()[0])
```

In [42]:
```
# Check Null value count in Moderate and Low Null Columns after Backward Fill and Forward Fill
# df[moderate_low_null_cols.index].isnull().sum()
```

Final Null Value Check

In [43]:
```
# Check for null values in each column
df.isnull().sum()/ len(df) * 100
```

Out[43]:
```
job_id                        0.0
company_name                  0.0
title                         0.0
description                   0.0
location                      0.0
company_id                    0.0
views                         0.0
formatted_work_type           0.0
original_listed_time          0.0
job_posting_url               0.0
application_url               0.0
application_type              0.0
expiry                        0.0
formatted_experience_level    0.0
listed_time                   0.0
posting_domain                0.0
sponsored                     0.0
zip_code                      0.0
fips                          0.0
skill_name                    0.0
industry_id                   0.0
industry_name                 0.0
dtype: float64
```

4. Exploratory Data Analysis (EDA)

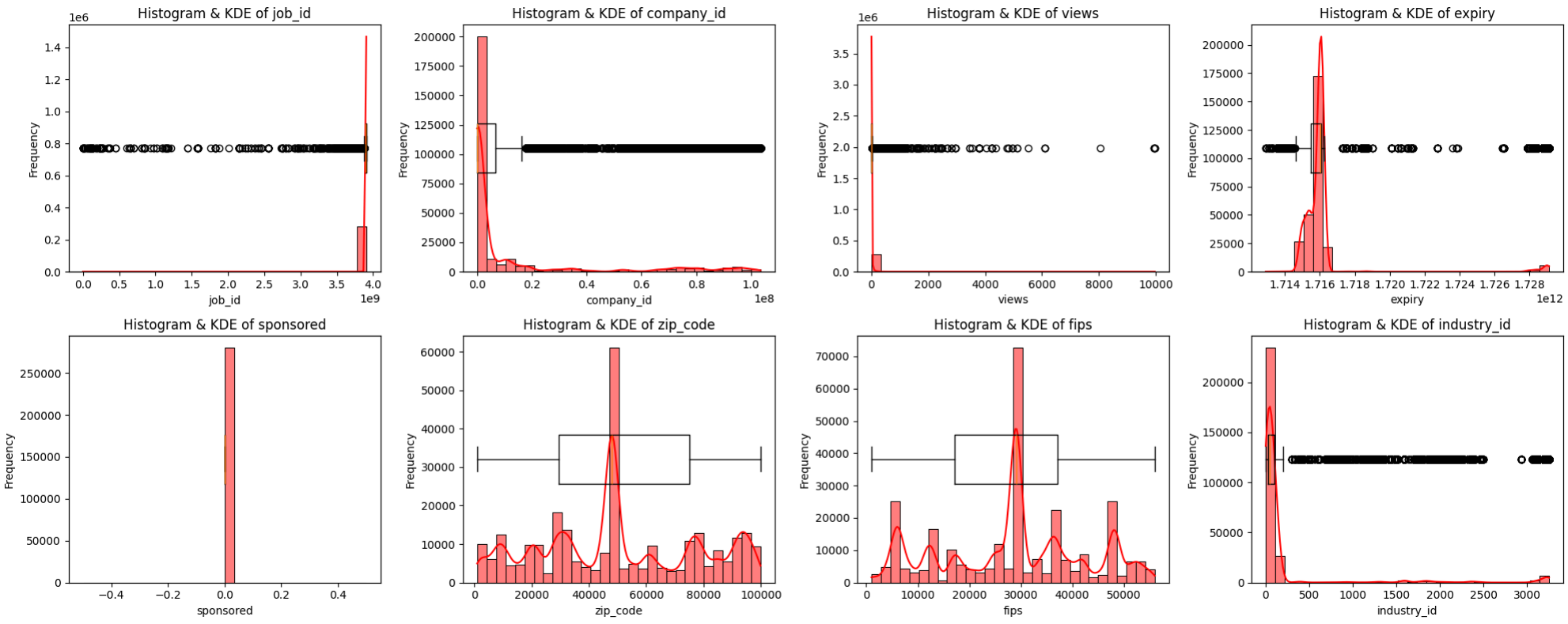
Uni-Variate Analysis

Separating Numeric and Categorical Column

In [44]:
```
#Separate the Numeric and Categorical columns

numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
categorical_cols = df.select_dtypes(include=['object', 'category']).columns.tolist()

# Print the numeric and categorical columns
print("Separated Columns :")
print("--"*25)
print(f"Numeric Column      :", numeric_cols)
print(f"Categorical Column  :", categorical_cols)
```

```
Separated Columns :
--------------------------------------------------
Numeric Column      : ['job_id', 'company_id', 'views', 'expiry', 'sponsored', 'zip_code', 'fips', 'industry_id']
Categorical Column  : ['company_name', 'title', 'description', 'location', 'formatted_work_type', 'job_posting_url', 'application_url', 'a
pplication_type', 'formatted_experience_level', 'posting_domain', 'skill_name', 'industry_name']
```

In [45]:
```
# Check data distribution of Numeric Columns with Histogram and Boxplot

import matplotlib.pyplot as plt
import seaborn as sns
import math

n_cols = 4
n_numeric = len(numeric_cols)
n_rows = math.ceil(n_numeric / n_cols)

fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 4 * n_rows))
axes = axes.flatten()

for i, col in enumerate(numeric_cols):
    ax = axes[i]
    # Histogram with KDE
    sns.histplot(df[col].dropna(), bins=30, kde=True, color='Red', ax=ax)
    ax.set_title(f'Histogram & KDE of {col}')
    ax.set_xlabel(col)
    ax.set_ylabel('Frequency')
    #ax.set_ylim(top=ax.get_ylim()[1] * 0.6)  # Optionally cap y-axis for compact plots

    # Overlay boxplot below the histogram (secondary y-axis)
    ax2 = ax.twinx()
    ax2.boxplot(df[col].dropna(), vert=False, widths=0.2, positions=[ax.get_ylim()[1]/2])
    ax2.get_yaxis().set_visible(False)

# Hide unused subplots
for i in range(n_numeric, len(axes)):
    axes[i].set_visible(False)

plt.tight_layout()
plt.show()
```

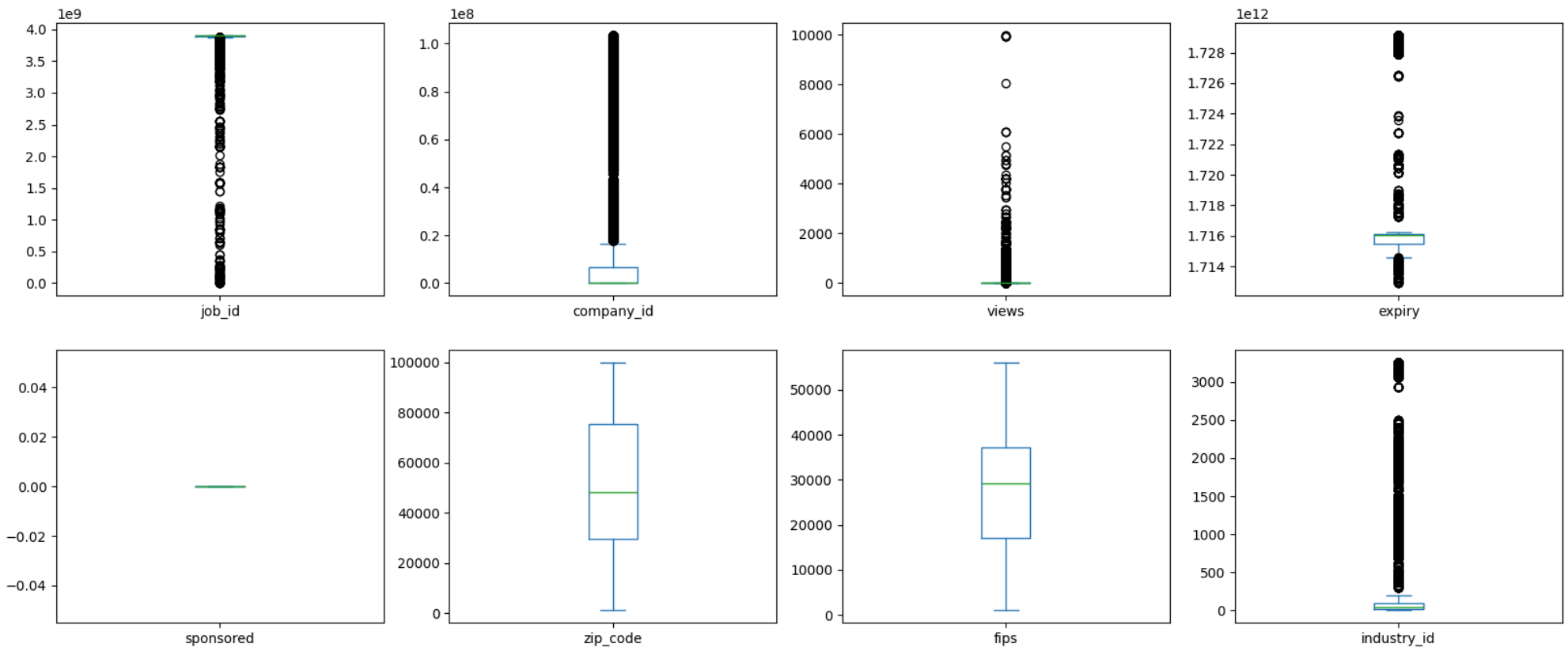


Key Insights of Numeric Data Distribution:

- **Heavy Skewness:** Most numeric columns (job_id, company_id, views, expiry, sponsored, zip_code, fips, industry_id) display strong right-skew (long tail), meaning a few values are disproportionately larger than the bulk of the data. This suggests rare but extreme outliers, especially for fields like views and company_id.
- **Distinct Clustering:** Features like zip_code and fips show visible clustering or spikes, corresponding to geographical or administrative boundaries, rather than random spread. This benefits location-based analyses and mapping.
- **Dense Central Masses:** Many columns exhibit strong density near the minimum or median, highlighted by both the histogram mass and KDE peak, indicating most job postings conform to typical values and only a few deviate far outside.
- **KDE Clarity:** The Kernel Density Estimate (KDE) curve overlays show the underlying data distribution shape and help spot multimodal characteristics (multiple peaks) in location or industry-related codes.

Handling Outliers

In [46]:
```
# Plot Box Plot to see all the OUTLIERS
import matplotlib.pyplot as plt

fig, axes = plt.subplots(figsize=(20, 8))
df[numeric_cols].plot(kind = 'box', subplots = True, ax=axes, layout = (2, 4))
plt.show()
```



Outlier & Spread Patterns

- **High cardinality or extreme outliers** are visible in job_id, company_id, expiry, industry_id, as well as the views column, where a few records vastly exceed the rest.
- **Binary field:** The sponsored field displays a binary (likely 0/1) distribution with no meaningful outlier or spread.
- **Geographic codes:** zip_code and fips show more classic boxplot shapes with visible ranges and central clustering, suggesting they represent structured identifiers within expected value ranges, though there is still moderate dispersion

In [47]:
```
# Function for putting a cap value on each of the selected column to REMOVE the Extreme Outliers

def cap_outliers(data, col_name):

  for i in col_name:
    Q1 = data[i].quantile(0.25)
    Q3 = data[i].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)

    print(f"Column: {i}")
    print(f"Lower Bound: {lower_bound}")
    print(f"Upper Bound: {upper_bound}")

    data[i] = np.where(data[i] < lower_bound, lower_bound, data[i])
    data[i] = np.where(data[i] > upper_bound, upper_bound, data[i])

  return data
```

In [48]:
```
# Setting the cap value as defined in the Function for numeric columns

df = cap_outliers(df, col_name=['job_id'])
df = cap_outliers(df, col_name=['company_id'])
df = cap_outliers(df, col_name=['views'])
df = cap_outliers(df, col_name=['expiry'])
df = cap_outliers(df, col_name=['sponsored'])
df = cap_outliers(df, col_name=['zip_code'])
df = cap_outliers(df, col_name=['fips'])
df = cap_outliers(df, col_name=['industry_id'])
```

```
Column: job_id
Lower Bound: 3880376333.0
Upper Bound: 3919330637.0
Column: company_id
Lower Bound: -10055288.0
Upper Bound: 16792592.0
Column: views
Lower Bound: -4.5
Upper Bound: 15.5
Column: expiry
Lower Bound: 1714577943500.0
Upper Bound: 1716996291500.0
Column: sponsored
Lower Bound: 0.0
Upper Bound: 0.0
Column: zip_code
Lower Bound: -38871.5
Upper Bound: 143644.5
Column: fips
Lower Bound: -12947.0
Upper Bound: 67229.0
Column: industry_id
Lower Bound: -81.5
Upper Bound: 202.5
```

In [49]:
```
# Plot box plots for all numeric columns using Plotly

import plotly.graph_objs as go
import plotly.subplots as sp

# Automatically select numeric columns
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns.tolist()

# Create subplot grid for boxplots
n = len(numeric_cols)
cols = 4
rows = (n // cols) + (n % cols > 0)
fig = sp.make_subplots(rows=rows, cols=cols, subplot_titles=numeric_cols)

for i, col in enumerate(numeric_cols):
    row = (i // cols) + 1
    col_num = (i % cols) + 1
    fig.add_trace(
        go.Box(y=df[col], name=col, boxmean='sd', boxpoints='outliers', orientation='v'),
        row=row, col=col_num
    )

fig.update_layout(height=300 * rows, width=1200, title_text="Boxplots of Numeric Columns (Outlier Removed)", showlegend=False)
fig.show()
```

Key Insights:

- **Job and company IDs** show wide numeric ranges, reflecting **many unique jobs and companies**.
- **Views** are **right-skewed**, with most postings receiving moderate attention, but some highly viewed.

- Expiry dates cluster closely, indicating similar posting durations.
- **Sponsored status** is almost binary with **very little variation** (mostly non-sponsored).
- **Zip codes and geographic codes (FIPS) cover broad ranges**, showing wide location diversity.

In [50]:
```
#categorical_cols
```

In [51]:
```
# Plot Bar plots showing the frequency of the top 10 most common categories for Categorical columns

import pandas as pd
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Assuming df is your DataFrame
cols = ['formatted_experience_level', 'posting_domain', 'company_name', 'skill_name', 'industry_name']

# Custom headers for each graph
custom_titles = [
    'Experience Level Distribution',
    'Top 10 Job Posting Domains',
    'Top 10 Companies',
    'Top 10 Key Skills in Demand',
    'Top 10 Industries'
]

# Create subplot figure with one column and number of rows equal to number of cols
fig = make_subplots(
    rows=len(cols),
    cols=1,
    shared_xaxes=False,
    vertical_spacing=0.04,
    subplot_titles=custom_titles
)

# Add horizontal bar plots to each row with top 10 categories showing highest on top
for i, col in enumerate(cols, 1):
    count_data = df[col].value_counts().sort_values(ascending=False).head(10).reset_index()
    count_data.columns = [col, 'count']
    count_data = count_data.iloc[::-1]

    fig.add_trace(
        go.Bar(y=count_data[col], x=count_data['count'], orientation='h', name=col),
        row=i,
        col=1
    )

# Update layout: bold and center subplot headers by modifying annotations
for annotation in fig['layout']['annotations']:
    annotation['font'] = dict(size=16, color='black', family='Arial', weight='bold')
    annotation['x'] = 0.5  # Center horizontally
    annotation['xanchor'] = 'center'
    annotation['yanchor'] = 'bottom'

fig.update_layout(
    height=300 * len(cols),
    #title_text='Top 10 Horizontal Bar Plots in One Column',
    showlegend=False
)

fig.show()
```

Key Insights :

Experience Level Insights

- The Mid-Senior level segment dominates the market, far outnumbering any other experience tier, indicating high demand for experienced professionals.
- Entry level roles are second in prevalence, but are much less common than mid-senior positions, highlighting a competitive gap for early career applicants.

Company and Domain Patterns

- Millennium Recruiting, Inc. holds the top spot for job postings, followed by DataAnnotation and Wells Fargo, suggesting these companies are highly active recruiters.
- The primary posting platforms are jsv3.recruitics.com, click.appcast.io, and click2apply.net, which account for the bulk of job listings, a signal that centralized job domains facilitate most postings.

Skills and Industry Trends

- Information Technology is the leading skill, complemented by Sales, Management, and Manufacturing, highlighting continued dominance of tech combined with general business capabilities.

- The most popular industries are Hospitals and Health Care, Retail, and IT Services/Consulting, revealing a concentration in healthcare and consumer services alongside ongoing tech industry growth.

In [52]:
```
# Replace these with your actual extracted data from df
# top_500 = df['title'].value_counts().nlargest(500)
# titles = top_500.index.tolist()  # List of titles
# counts = top_500.values.tolist()  # List of corresponding counts
```

In [53]:
```
# Create a word colud visualization for Job Titles

from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Use your specific table and column name
# Example: if your dataframe is named df and the column is 'title'
job_titles_text = ' '.join(df['title'].astype(str))

# Generate the word cloud with a colored colormap (e.g., 'plasma')
wordcloud = WordCloud(width=1600, height=800, background_color='white', colormap='Set1').generate(job_titles_text)

# Display the word cloud with a figure that automatically fits most screens
plt.figure(figsize=(16, 8), facecolor='w')
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Job Title Word Cloud', fontsize=18)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

Key Insights:

- Project Manager, Sales Associate, and Software Engineer are the **most frequently listed job titles**.
- Both **Full Time and Part Time** roles are **highly prevalent**, reflecting diversity in work arrangements.
- Strong hiring focus on sales, customer service, and management positions.
- Titles indicate **opportunities across all seniority levels**, from Entry Level to Senior Management.
- **Multiple industries**—especially retail, technology, healthcare, and finance—are represented in top roles.

In [54]:
```
# Generate word cloud of the most common skills required for the job postings

from wordcloud import WordCloud
import matplotlib.pyplot as plt

job_titles_text = ' '.join(df['skill_name'].astype(str))

wordcloud = WordCloud(width=1600, height=800, background_color='white', colormap='tab10').generate(job_titles_text)

# Display the word cloud with a figure that automatically fits most screens
plt.figure(figsize=(16, 8), facecolor='w')
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Top skill requirements for the job postings', fontsize=18)
plt.axis('off')
```

```
plt.tight_layout(pad=0)
plt.show()
```

Key Insights:

- Business Development, Health Care, Customer Service, Information Technology, and Management are among the **most in-demand skills**, as evidenced by their large, prominent text.
- Other top skills include Engineering, Project Management, Sales, Accounting, Auditing, Finance, and Manufacturing, reflecting **broad demand across both technical and business-oriented roles**.
- Many postings seek strong backgrounds in technology, management, and customer-facing functions, highlighting their importance for job seekers.
- The **presence of diverse skills** like Supply Chain, Human Resources, and Research Analyst suggests a wide variety of job types and sectors are actively hiring.
- Larger recurring words across the images indicate consistently high-volume demand, particularly for business, health care, and technology-related expertise.

In [55]:
```
# Word Cloud describing the different industries with most job openings

from wordcloud import WordCloud
import matplotlib.pyplot as plt

job_titles_text = ' '.join(df['industry_name'].astype(str))
wordcloud = WordCloud(width=1600, height=800, background_color='white', colormap='magma').generate(job_titles_text)

# Display the word cloud with a figure that automatically fits most screens
plt.figure(figsize=(16, 8), facecolor='w')
plt.imshow(wordcloud, interpolation='bilinear')
plt.title('Industry with most job openings', fontsize=18)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

Key Insights:

- Health Care, Financial Services, Retail, Consulting, and Hospitals are consistently the **largest and most prominent industries**, indicating they have the highest number of job openings across all images.
- **Other visible major sectors** include Software Development, Manufacturing (including subcategories like Industrial, Pharmaceutical, and Equipment Manufacturing), Recruiting, Staffing, and Advertising Services.
- Real Estate, Logistics, Technology, Engineering, and Information Services also stand out but are less dominant compared to the top five.
- The diversity and relative word size indicate a **wide spectrum of opportunities**, but with a clear tilt toward health, finance, retail, and consultancy domains.
- Frequent recurrence of **Healthcare and Financial keywords** across different images highlights the **critical workforce demand in these sectors**.

In [56]:
```python
# To check the distribution of key categorical features

import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Example: Assume these variables are precomputed from your DataFrame
exp_levels = df['formatted_experience_level'].value_counts().sort_index()
app_types = df['application_type'].value_counts().sort_index()
work_types = df['formatted_work_type'].value_counts().sort_index()

# Create subplot figure with 3 columns
fig = make_subplots(rows=1, cols=3,
                    subplot_titles=('<b>Experience Level</b>',
                                    '<b>Application Type</b>',
                                    '<b>Work Type</b>'))

# Add bar charts to subplots
fig.add_trace(go.Bar(x=exp_levels.index, y=exp_levels.values, marker_color='blue',name='Jobs'), row=1, col=1)
fig.add_trace(go.Bar(x=app_types.index, y=app_types.values, marker_color='green',name='Jobs'), row=1, col=2)
fig.add_trace(go.Bar(x=work_types.index, y=work_types.values, marker_color='orange',name='Jobs'), row=1, col=3)

# Update layout and axis labels
fig.update_layout(height=500, width=1100, showlegend=False)

for i in range(1, 4):
    fig.update_xaxes(tickangle=45, tickfont=dict(size=12), row=1, col=i)
    fig.update_yaxes(title_text='Number of Jobs', row=1, col=1)

fig.show()
```

Key Insights :

- **Experience Level Distribution**
  - The majority of job postings target Mid-Senior level and Entry level candidates, making up the largest share of opportunities.
  - There are significantly fewer openings for Associate, Director, Internship, and very few for Executive roles, indicating that advanced roles are much less common.
- **Application Type Distribution**

    - OffsiteApply is the most prominent application method, followed by ComplexOnsiteApply.
    - SimpleOnsiteApply and UnknownApply have very low counts, suggesting that job postings tend to require more complex or external application processes rather than streamlined experiences.
- **Work Type Distribution**
    - The overwhelming majority of jobs are Full-time, highlighting strong demand for permanent positions on LinkedIn.
    - Internship, Contract, and Part-time positions are available but in significantly lower quantities. Roles classified as Temporary, Volunteer, or Other are rare.
- **Actionable Insights**
    - Recommendation systems should focus heavily on matching users to full-time and mid-senior/entry-level roles, as these represent the largest share of roles available.
    - For those seeking streamlined job application experiences, platform improvements around SimpleOnsiteApply may impact only a small fraction of postings.
    - Candidates for advanced positions (Director, Executive) may require targeted search strategies or personalized guidance, given their scarcity.

Bi-Variate Analysis

In [ ]:
```python
# Visualize the distribution between two categorical variables Work Type and Experience level

import plotly.express as px

fig = px.histogram(df,
                   x='formatted_work_type',
                   color='formatted_experience_level',
                   # Order the x-axis categories by frequency for better readability.
                   category_orders={
                       'formatted_work_type': df['formatted_work_type'].value_counts().index.tolist()
                   },
                   barmode='group',
                   # Assign human-readable labels for the final visualization.
                   labels={'formatted_work_type': 'Work Type', 'count': 'Number of Jobs', 'formatted_experience_level': 'Experience Level
                   title='Distribution of Jobs by Work Type and Experience Level',
                   color_discrete_sequence=px.colors.qualitative.Pastel)

# Update the layout for a more polished and informative look.
fig.update_layout(
    title=dict(
        text='<b>Distribution of Jobs by Work Type and Experience Level (Log Scale)</b>',
        x=0.5,  # center title horizontally
        xanchor='center',
    ),
    yaxis_title='<b>Number of Jobs</b>',  # bold y-axis label
    showlegend=True
)

fig.update_yaxes(type='log')  # Set y-axis to log scale

fig.show()
```

Key Insights:

- **Full-time roles** overwhelmingly **dominate across all experience levels**, with the largest volumes for mid-senior, entry, and associate positions.
- Contract, part-time, and temporary jobs are most frequent at mid-senior and entry levels, but are also available across the full range of experience.
- **Internships** show a spike entirely **at the internship experience level**, as expected, while other minor work types (volunteer, other) are more evenly spread.
- **Director and executive** roles have noticeably **fewer postings across all work types**, reflecting typical organizational structures.
- The log scale reveals strong segmentation: while full-time is the backbone, alternatives like contract and part-time remain significant for non-senior and non-executive professionals

In [58]:
```python
import plotly.express as px

fig = px.histogram(df,
                   x='formatted_experience_level',         # moved here from color
                   color='formatted_work_type',             # moved here from x
                   category_orders={
                       'formatted_experience_level': df['formatted_experience_level'].value_counts().index.tolist()
                   },
                   barmode='group',
                   labels={'formatted_experience_level': 'Experience Level', 'count': 'Number of Jobs', 'formatted_work_type': 'Work Type
                   title='Distribution of Jobs by Experience Level and Work Type',
                   color_discrete_sequence=px.colors.qualitative.G10)

fig.update_layout(
    title=dict(
        text='<b>Distribution of Jobs by Experience Level and Work Type (Log Scale)</b>',
        x=0.5,
        xanchor='center',
```

```
    ),
    yaxis_title='<b>Number of Jobs</b>',
    showlegend=True
)

fig.update_yaxes(type='log')  # if log scale is still required

fig.show()
```

Key Insights:

- **Full-time positions dominate every experience level**, with particularly high counts in mid-senior, entry, and associate roles.
- **Internships and contract roles** are especially **abundant at entry and mid-senior levels**, though available at all levels including executive and director.
- Other work types (part-time, temporary, volunteer, other) consistently show lower volumes across all experience levels, but remain present and relevant, especially at mid-senior and entry levels.
- **Executive and director** positions show an **overall drop in job counts across all work types** compared to more junior roles, reflecting typical market structures.
- The **log scale** highlights substantial differences in job availability between full-time roles and all other work types throughout the experience spectrum.

5. Feature Engineering

In [ ]:

```
# Transforming and parsing a single variable (location) into multiple, more granular variables (city, state, country)

import pandas as pd
import re

# Dictionary for US state abbreviations and full names
STATE_ABBR = {
    'AL': 'Alabama', 'AK': 'Alaska', 'AZ': 'Arizona', 'AR': 'Arkansas', 'CA': 'California',
    'CO': 'Colorado', 'CT': 'Connecticut', 'DE': 'Delaware', 'FL': 'Florida', 'GA': 'Georgia',
    'HI': 'Hawaii', 'ID': 'Idaho', 'IL': 'Illinois', 'IN': 'Indiana', 'IA': 'Iowa', 'KS': 'Kansas',
    'KY': 'Kentucky', 'LA': 'Louisiana', 'ME': 'Maine', 'MD': 'Maryland', 'MA': 'Massachusetts',
    'MI': 'Michigan', 'MN': 'Minnesota', 'MS': 'Mississippi', 'MO': 'Missouri', 'MT': 'Montana',
    'NE': 'Nebraska', 'NV': 'Nevada', 'NH': 'New Hampshire', 'NJ': 'New Jersey', 'NM': 'New Mexico',
    'NY': 'New York', 'NC': 'North Carolina', 'ND': 'North Dakota', 'OH': 'Ohio', 'OK': 'Oklahoma',
    'OR': 'Oregon', 'PA': 'Pennsylvania', 'RI': 'Rhode Island', 'SC': 'South Carolina',
    'SD': 'South Dakota', 'TN': 'Tennessee', 'TX': 'Texas', 'UT': 'Utah', 'VT': 'Vermont',
    'VA': 'Virginia', 'WA': 'Washington', 'WV': 'West Virginia', 'WI': 'Wisconsin', 'WY': 'Wyoming'
}
# Create sets for quick lookups of states (full and abbreviations)
STATE_SET = set([v.lower() for v in STATE_ABBR.values()] + [k.lower() for k in STATE_ABBR.keys()])

# Known countries to detect in location strings
COUNTRIES = {'united states', 'canada', 'argentina', 'brazil'}

# Helper to safely strip strings and avoid errors on non-strings
def safe_strip(val):
    """Strip if string, return unchanged if not."""
    return val.strip() if isinstance(val, str) else val

# Main function to parse location into city, state, country
def fast_extract(location):
    s = str(location).strip()  # Convert to string and strip whitespace
    s_lower = s.lower()
    country = None
    state = None
    city = None

    # Detect and remove country tokens from string
    detected_country = None
    for c in COUNTRIES:
        if re.search(rf'\b{re.escape(c)}\b', s_lower):
            detected_country = c.title()
            s_lower = re.sub(rf'\b{re.escape(c)}\b', '', s_lower)
    if detected_country:
        country = detected_country

    # Detect state abbreviation pattern (e.g., ", CA")
    match = re.search(r',\s*([A-Z]{2})(?:,|$)', s)
    if match and match.group(1) in STATE_ABBR:
        # Extract city from start before first comma, if valid
        state_candidate = STATE_ABBR[match.group(1)]
        city_candidate = safe_strip(s.split(',')[0])
        if (
            city_candidate and
            isinstance(city_candidate, str) and
            city_candidate.lower() not in COUNTRIES and
            city_candidate.lower() not in STATE_SET
        ):
            city = city_candidate
        state = state_candidate
```

```
# If no state abbrev found, detect full state name
if not state:
    for st in STATE_ABBR.values():
        st_low = st.lower()
        if re.search(rf'\b{re.escape(st_low)}\b', s_lower):
            state_candidate = st
            idx = s_lower.find(st_low)
            city_candidate = safe_strip(s[:idx].rstrip(','))
            if (
                city_candidate and
                isinstance(city_candidate, str) and
                city_candidate.lower() not in COUNTRIES and
                city_candidate.lower() not in STATE_SET
            ):
                city = city_candidate
            state = state_candidate
            break

# Fallback parsing assuming comma separation into city, state, country
if not state or not city:
    tokens = [safe_strip(tok) for tok in s.split(',') if isinstance(tok, str)]
    if len(tokens) == 3:
        cty, stt, ctry = tokens
        if cty and cty.lower() not in COUNTRIES and cty.lower() not in STATE_SET:
            city = cty
        if stt and stt.lower() not in COUNTRIES:
            state = STATE_ABBR[stt] if stt in STATE_ABBR else stt
        if ctry and ctry.lower() in COUNTRIES:
            country = ctry.title()
    elif len(tokens) == 2:
        cty, stt = tokens
        if cty and cty.lower() not in COUNTRIES and cty.lower() not in STATE_SET:
            city = cty
        if stt and stt.lower() not in COUNTRIES:
            state = STATE_ABBR[stt] if stt in STATE_ABBR else stt

    # Final fallback: If only country provided
    s_lower_check = safe_strip(s_lower)
    if not city and not state and isinstance(s_lower_check, str) and s_lower_check in COUNTRIES:
        country = s.title()

    return pd.Series([city, state, country])

# Apply parsing function to 'location' column in DataFrame and create new columns
df[['city', 'state', 'country']] = df['location'].apply(fast_extract)

# Print sample output showing original and parsed data
print(df[['location', 'city', 'state', 'country']].head(15))
```

```
            location          city          state        country
0       Princeton, NJ      Princeton     New Jersey           None
1       Princeton, NJ      Princeton     New Jersey           None
2    Fort Collins, CO   Fort Collins       Colorado           None
3      Cincinnati, OH     Cincinnati           Ohio           None
4      Cincinnati, OH     Cincinnati           Ohio           None
5   New Hyde Park, NY  New Hyde Park       New York           None
6      Burlington, IA     Burlington           Iowa           None
7         Raleigh, NC        Raleigh  North Carolina           None
8       United States           None           None  United States
9       United States           None           None  United States
10      United States           None           None  United States
11  San Francisco, CA  San Francisco     California           None
12  San Francisco, CA  San Francisco     California           None
13  San Francisco, CA  San Francisco     California           None
14  San Francisco, CA  San Francisco     California           None
```

In [ ]:
```
# Check for Value count
df['country'].value_counts()
```

Out[ ]:
```
country
United States    30783
Canada              13
Brazil               5
Argentina            2
Name: count, dtype: int64
```

In [61]:
```
#df['country'].isnull().sum()
```

In [ ]:
```
 # Check for Value count
df['state'].value_counts()
```

Out[ ]:
```
state
California       28915
Texas            24192
New York         18151
Florida          14664
Illinois         10591
                 ...
South Holland        1
QC                   1
NC Area              1
AR Area              1
AZ Area              1
Name: count, Length: 61, dtype: int64
```

In [63]:
```
#df['state'].isnull().sum()
```

In [ ]:
```
 # Check for Value count
df['city'].value_counts()
```

Out[ ]:
```
city
Houston          4137
Chicago          4118
Atlanta          3439
Dallas           3392
Boston           2676
                 ...
Center Conway       1
Belfair             1
South Royalton      1
Bagdad              1
Benton City         1
Name: count, Length: 6375, dtype: int64
```

In [65]:
```
#df['city'].isnull().sum()
```

In [66]:
```
#df[['location', 'city', 'state', 'country']].head(50)
```

In [67]:
```
# To Check for NULL states where city is NOT NULL

# Filter for rows where state is NaN/null and city is present
null_state_nonnull_city = df[df['state'].isnull() & df['city'].notnull()]

# Show a sample of these rows
null_state_nonnull_city[['location', 'city', 'state', 'country']].head(15)
```

Out[67]:

	location	city	state	country
9861	District of Columbia, United States	District of Columbia	None	United States
9862	District of Columbia, United States	District of Columbia	None	United States
9997	District of Columbia, United States	District of Columbia	None	United States
10522	District of Columbia, United States	District of Columbia	None	United States
10523	District of Columbia, United States	District of Columbia	None	United States
14082	District of Columbia, United States	District of Columbia	None	United States
21820	District of Columbia, United States	District of Columbia	None	United States
21821	District of Columbia, United States	District of Columbia	None	United States
22696	District of Columbia, United States	District of Columbia	None	United States
36611	District of Columbia, United States	District of Columbia	None	United States
38751	District of Columbia, United States	District of Columbia	None	United States
38752	District of Columbia, United States	District of Columbia	None	United States
42284	District of Columbia, United States	District of Columbia	None	United States
42512	Northeast, United States	Northeast	None	United States
42513	Northeast, United States	Northeast	None	United States

In [68]:
```
# Check for NULL values in each location column

df[['city', 'state', 'country']].isnull().sum()
```

Out[68]:
```
city        54221
state       31417
country    249862
dtype: int64
```

In [69]:
```
# Add Country for US States (Vectorized)

US_STATES = set(STATE_ABBR.values())

# Only fill for rows where country is null and state is a US state
```

```
df.loc[
    df['country'].isnull() & df['state'].notnull() & df['state'].isin(US_STATES),
    'country'
] = 'United States'

# Display sample output for verification
print(df[['location', 'city', 'state', 'country']].head(15))
```

```
            location           city           state        country
0       Princeton, NJ      Princeton      New Jersey  United States
1       Princeton, NJ      Princeton      New Jersey  United States
2    Fort Collins, CO   Fort Collins        Colorado  United States
3      Cincinnati, OH     Cincinnati            Ohio  United States
4      Cincinnati, OH     Cincinnati            Ohio  United States
5   New Hyde Park, NY  New Hyde Park        New York  United States
6      Burlington, IA     Burlington            Iowa  United States
7         Raleigh, NC        Raleigh  North Carolina  United States
8       United States           None            None  United States
9       United States           None            None  United States
10      United States           None            None  United States
11  San Francisco, CA  San Francisco      California  United States
12  San Francisco, CA  San Francisco      California  United States
13  San Francisco, CA  San Francisco      California  United States
14  San Francisco, CA  San Francisco      California  United States
```

In [70]:
```
# Check for NULL values in each location column

# df[['city', 'state', 'country']].isnull().sum()
```

In [71]:
```
# To Check for states and city both are NULL

# Filter for rows where state is NaN/null and city is present
null_state_null_city = df[df['state'].isnull() & df['city'].isnull()]

# Show a sample of these rows
# null_state_null_city[['location', 'city', 'state', 'country']].head(15)
```

In [72]:
```
# Check the Total count of NULL where city and state are NULL
null_state_null_city[['location', 'city', 'state', 'country']].isnull().sum()
```

Out[72]:
```
location        0
city        31298
state       31298
country     12358
dtype: int64
```

In [73]:
```
# Check the Total count where Country, State and City ALL are NULL
df[['location', 'city', 'state', 'country']].isnull().sum()
```

Out[73]:
```
location        0
city        54221
state       31417
country     14466
dtype: int64
```

In [74]:
```
# Create a boolean mask where the 'Location' column contains "Area"
filtered_mask = df['location'].str.contains('Area')

# Apply the mask to the DataFrame to get the filtered rows
filtered_df = df[filtered_mask]

# View all values in the 'Location' column of the filtered DataFrame
print(filtered_df['location'].value_counts())
```

```
location
New York City Metropolitan Area    2487
San Francisco Bay Area              931
Washington DC-Baltimore Area        848
Los Angeles Metropolitan Area       813
Greater Chicago Area                726
                                   ...
Greater Jacksonville, NC Area         1
Bellingham Metropolitan Area          1
Greater Enid Area                     1
Greater Bakersfield Area              1
Walla Walla Area                      1
Name: count, Length: 181, dtype: int64
```

In [75]:
```
filtered_df['location'].value_counts().sum()
```

Out[75]:
```
np.int64(14473)
```

In [76]:
```
# Before filling
before_count = ((df['city'].isnull()) & (df['state'].isnull()) & (df['country'].isnull())).sum()
print(f"Number of all-null (city, state, country) rows before filling: {before_count}")

# (run the fill code)
```

```
Number of all-null (city, state, country) rows before filling: 12358
```

In [77]:
```
# Mapping Dictionary to translate complex location phrases into structured components

# Location phrase mapped to city, state, country
location_full_map = {
    'New York City Metropolitan Area':           ('New York',     'New York',         'United States'),
    'San Francisco Bay Area':                    ('San Francisco','California',       'United States'),
    'Washington DC-Baltimore Area':              ('Washington',   'District of Columbia', 'United States'),
    'Los Angeles Metropolitan Area':             ('Los Angeles',  'California',       'United States'),
    'Greater Chicago Area':                      ('Chicago',      'Illinois',         'United States'),
    'Atlanta Metropolitan Area':                 ('Atlanta',      'Georgia',          'United States'),
    'Austin, Texas Metropolitan Area':           ('Austin',       'Texas',            'United States'),
    'San Antonio, Texas Metropolitan Area':      ('San Antonio',  'Texas',            'United States'),
    'Miami-Fort Lauderdale Area':                ('Miami',        'Florida',          'United States'),
    'Denver Metropolitan Area':                  ('Denver',       'Colorado',         'United States'),
    'Greater Phoenix Area':                      ('Phoenix',      'Arizona',          'United States'),
    'Raleigh-Durham-Chapel Hill Area':           ('Raleigh',      'North Carolina',   'United States'),
    'Nashville Metropolitan Area':               ('Nashville',    'Tennessee',        'United States'),
    'Columbus, Ohio Metropolitan Area':          ('Columbus',     'Ohio',             'United States'),
    'Greater Minneapolis-St. Paul Area':         ('Minneapolis',  'Minnesota',        'United States'),
    'Detroit Metropolitan Area':                 ('Detroit',      'Michigan',         'United States'),
    'Portland, Oregon Metropolitan Area':        ('Portland',     'Oregon',           'United States'),
    'San Diego Metropolitan Area':               ('San Diego',    'California',       'United States'),
    'Omaha Metropolitan Area':                   ('Omaha',        'Nebraska',         'United States'),
    'Greater Seattle Area':                      ('Seattle',      'Washington',       'United States'),
    'Salt Lake City Metropolitan Area':          ('Salt Lake City','Utah',            'United States'),
}

def fill_from_location(loc):
    entry = location_full_map.get(str(loc).strip())
    if entry is not None:
        return pd.Series(entry, index=['city', 'state', 'country'])
    else:
        # Keep as (None, None, None) if not found
        return pd.Series([None, None, None], index=['city', 'state', 'country'])

# Mask for rows where all three are missing
mask_all_null = (
    df['city'].isnull() &
    df['state'].isnull() &
    df['country'].isnull()
)

# Only update those rows
df.loc[mask_all_null, ['city', 'state', 'country']] = (
    df.loc[mask_all_null, 'location'].apply(fill_from_location)
)

# Display a few examples for verification
#print(df[mask_all_null][['location', 'city', 'state', 'country']].head(20))
df[mask_all_null][['location', 'city', 'state', 'country']].head(20)
```

Out[77]:

	location	city	state	country
30	Greater Philadelphia	None	None	None
31	Greater Philadelphia	None	None	None
32	Greater Philadelphia	None	None	None
45	Los Angeles Metropolitan Area	Los Angeles	California	United States
46	Los Angeles Metropolitan Area	Los Angeles	California	United States
85	Dallas-Fort Worth Metroplex	None	None	None
86	Dallas-Fort Worth Metroplex	None	None	None
182	Greensboro--Winston-Salem--High Point Area	None	None	None
205	Raleigh-Durham-Chapel Hill Area	Raleigh	North Carolina	United States
206	Raleigh-Durham-Chapel Hill Area	Raleigh	North Carolina	United States
279	Greater Minneapolis-St. Paul Area	Minneapolis	Minnesota	United States
284	Greater Philadelphia	None	None	None
285	Raleigh-Durham-Chapel Hill Area	Raleigh	North Carolina	United States
286	Raleigh-Durham-Chapel Hill Area	Raleigh	North Carolina	United States
342	San Francisco Bay Area	San Francisco	California	United States
343	San Francisco Bay Area	San Francisco	California	United States
364	Cincinnati Metropolitan Area	None	None	None
377	Greater Phoenix Area	Phoenix	Arizona	United States
427	Louisville Metropolitan Area	None	None	None
481	San Francisco Bay Area	San Francisco	California	United States

In [78]:
```
# After filling
after_count = ((df['city'].isnull()) & (df['state'].isnull()) & (df['country'].isnull())).sum()
print(f"Number of all-null (city, state, country) rows after filling: {after_count}")
```

```
Number of all-null (city, state, country) rows after filling: 6035
```

In [79]:
```
# df[mask_all_null][['location', 'city', 'state', 'country']].isnull().sum()
```

In [80]:
```
# Check the Total count where Country, State and City ALL are NULL
# df[['location', 'city', 'state', 'country']].isnull().sum()
```

In [81]:
```
# Show rows whose location matched and who were updated
updated_rows = df['location'].isin(list(location_full_map.keys()))

#print(df[updated_rows][['location', 'city', 'state', 'country']].head(20))
df[updated_rows][['location', 'city', 'state', 'country']].head(20)
```

Out[81]:

	location	city	state	country
45	Los Angeles Metropolitan Area	Los Angeles	California	United States
46	Los Angeles Metropolitan Area	Los Angeles	California	United States
121	New York City Metropolitan Area	None	New York	United States
122	New York City Metropolitan Area	None	New York	United States
123	New York City Metropolitan Area	None	New York	United States
124	New York City Metropolitan Area	None	New York	United States
125	New York City Metropolitan Area	None	New York	United States
126	New York City Metropolitan Area	None	New York	United States
127	New York City Metropolitan Area	None	New York	United States
128	New York City Metropolitan Area	None	New York	United States
129	New York City Metropolitan Area	None	New York	United States
198	New York City Metropolitan Area	None	New York	United States
205	Raleigh-Durham-Chapel Hill Area	Raleigh	North Carolina	United States
206	Raleigh-Durham-Chapel Hill Area	Raleigh	North Carolina	United States
279	Greater Minneapolis-St. Paul Area	Minneapolis	Minnesota	United States
280	Columbus, Ohio Metropolitan Area	Columbus,	Ohio	United States
281	Columbus, Ohio Metropolitan Area	Columbus,	Ohio	United States
285	Raleigh-Durham-Chapel Hill Area	Raleigh	North Carolina	United States
286	Raleigh-Durham-Chapel Hill Area	Raleigh	North Carolina	United States
330	Columbus, Ohio Metropolitan Area	Columbus,	Ohio	United States

In [82]:
```
# show all rows without nulls for city/state/country after update:
#print(df[(df['city'].notnull()) | (df['state'].notnull()) | (df['country'].notnull())].head(20))
df[(df['city'].notnull()) | (df['state'].notnull()) | (df['country'].notnull())].head()
```

Out[82]:

	job_id	company_name	title	description	location	company_id	views	formatted_work_type	original_listed_time	
0	3.880376e+09	Corcoran Sawyer Smith	Marketing Coordinator	Job descriptionA leading real estate firm in N...	Princeton, NJ	2774458.0	15.5	Full-time	2024-04-17	https://v
1	3.880376e+09	Corcoran Sawyer Smith	Marketing Coordinator	Job descriptionA leading real estate firm in N...	Princeton, NJ	2774458.0	15.5	Full-time	2024-04-17	https://v
2	3.880376e+09	Millennium Recruiting, Inc.	Mental Health Therapist/Counselor	At Aspen Therapy and Wellness , we are committ...	Fort Collins, CO	167757.0	1.0	Full-time	2024-04-11	https://w
3	3.880376e+09	The National Exemplar	Assitant Restaurant Manager	The National Exemplar is accepting application...	Cincinnati, OH	16792592.0	8.0	Full-time	2024-04-16	https://ww
4	3.880376e+09	The National Exemplar	Assitant Restaurant Manager	The National Exemplar is accepting application...	Cincinnati, OH	16792592.0	8.0	Full-time	2024-04-16	https://ww

5 rows × 25 columns

In [83]:
```
# To fill all rows where city is NULL but state and country are NOT NULL, set the city to "Unknown" using a vectorized approach

mask = df['city'].isnull() & df['state'].notnull() & df['country'].notnull()
df.loc[mask, 'city'] = 'Unknown'

# Display a few affected rows for verification
print(df[mask][['location', 'city', 'state', 'country']].head(10))
```

```
                          location     city         state        country
53         New Jersey, United States  Unknown    New Jersey  United States
54         New Jersey, United States  Unknown    New Jersey  United States
96                     New York, NY  Unknown      New York  United States
97       South Dakota, United States  Unknown  South Dakota  United States
98       South Dakota, United States  Unknown  South Dakota  United States
116                    New York, NY  Unknown      New York  United States
119          New York, United States  Unknown      New York  United States
120          New York, United States  Unknown      New York  United States
121  New York City Metropolitan Area  Unknown      New York  United States
122  New York City Metropolitan Area  Unknown      New York  United States
```

In [84]:
```
# Check the Total count where Country, State and City ALL are NULL
# df[['location', 'city', 'state', 'country']].isnull().sum()
```

In [85]:
```
# Check for Final NULL rows where ALL city, state and country is NULL
((df['city'].isnull()) & (df['state'].isnull()) & (df['country'].isnull())).sum()
```

Out[85]: np.int64(6035)

In [86]:
```
# Check the Percentage of data which was missing
print(f"Percentage of all-null location rows: {6035/280000:.2%}")
```

```
Percentage of all-null location rows: 2.16%
```

With only 2% of postings having no location, dropping these will not bias most analyses—and will improve clarity and quality of your results.

In [87]:
```
# Dropping all these rows with NULL values and updating the original dataframe
df = df[~(df['city'].isnull() & df['state'].isnull() & df['country'].isnull())]
print(df.shape)
```

```
(274630, 25)
```

In [88]:
```
# Check for Final NULL rows where ALL city, state and country is NULL
((df['city'].isnull()) & (df['state'].isnull()) & (df['country'].isnull())).sum()
```

Out[88]: np.int64(0)

In [89]:
```
#df['country'].value_counts()
#df['state'].value_counts()
#df['city'].value_counts()
```

Geo-spatial Analysis

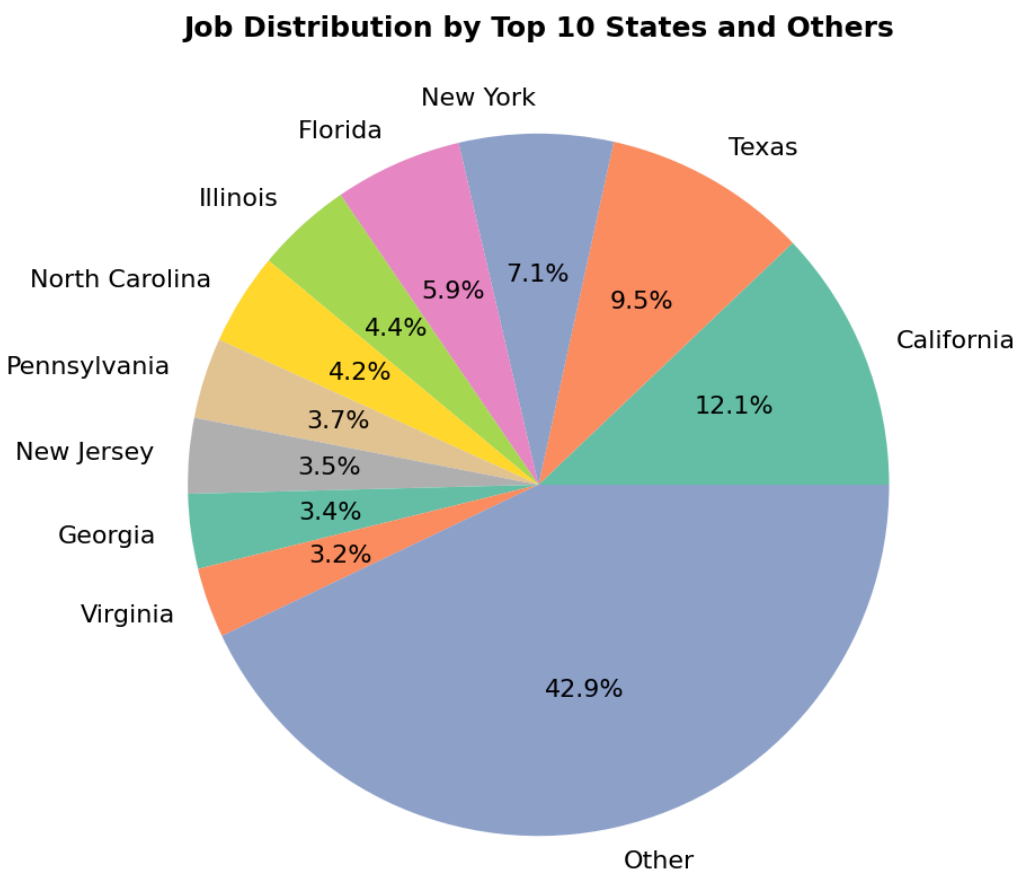
In [90]:
```python
# Visualization showing the top 10 cities, states, and countries by number of job postings in the dataset

import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Extract top 10 counts (replace df with your dataset)
city_counts = df['city'].value_counts().nlargest(10)
state_counts = df['state'].value_counts().nlargest(10)
country_counts = df['country'].value_counts().nlargest(10)

# Create subplots
fig = make_subplots(
    rows=1, cols=3,
    subplot_titles=['<b>Top 10 Cities</b>', '<b>Top 10 States</b>', '<b>Top 10 Countries</b>']
)

# Add bar plots
fig.add_trace(go.Bar(x=city_counts.index, y=city_counts.values, marker_color='crimson',name='Job Postings'), row=1, col=1)
fig.add_trace(go.Bar(x=state_counts.index, y=state_counts.values, marker_color='coral',name='Job Postings'), row=1, col=2)
fig.add_trace(go.Bar(x=country_counts.index, y=country_counts.values, marker_color='forestgreen',name='Job Postings'), row=1, col=3)

# Layout updates
fig.update_layout(
    height=500, width=1200,
    showlegend=False,
    #title_text='Top 10 Cities, States, and Countries by Job Postings'
)

# Rotate x-axis labels and add y-axis title
for i in range(1, 4):
    fig.update_xaxes(tickangle=45, tickfont=dict(size=12), row=1, col=i)
    fig.update_yaxes(title_text='Job Postings', row=1, col=i)

#fig.update_yaxes(type='log')  # if log scale is still required

fig.show()
```

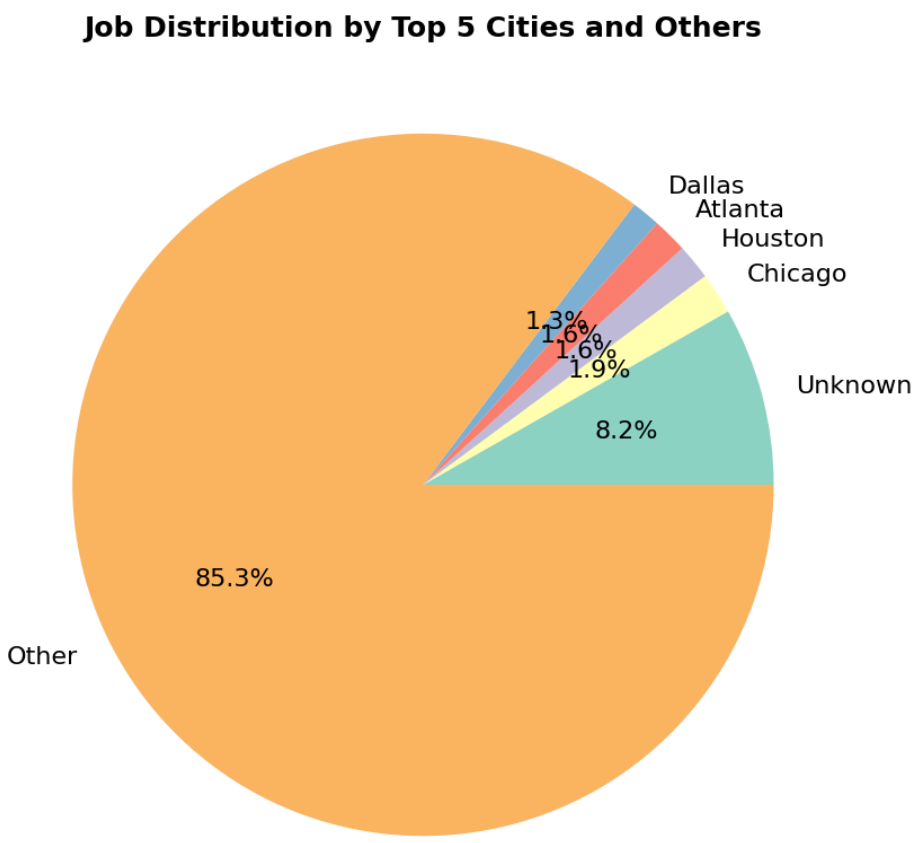
Key Insights:

- Within cities, the "Unknown" group has the highest job posting count by a large margin, dwarfing all named cities; Chicago, Houston, Dallas, and others trail far behind.
- At the **state level**, **California, Texas, and Florida** occupy the top three spots for job postings, with noticeable but smaller volumes in New York, Illinois, and other states.
- At the **country level**, the **United States** overwhelmingly dominates job postings, with Canada, Brazil, and Argentina contributing only a tiny fraction by comparison.

In [91]:
```python
# Pie Chart visualizations showing the distribution of job postings by location, split into two views: states and cities

import matplotlib.pyplot as plt
import pandas as pd

# --- State Pie Chart Data Preparation ---
# Top 10 states based on job postings
top_states = df['state'].value_counts().nlargest(10)
top_state_names = top_states.index.tolist()

# Full series of state value counts
state_counts = df['state'].value_counts()
top_state_counts = state_counts.loc[top_state_names]
other_states_count = state_counts.loc[~state_counts.index.isin(top_state_names)].sum()

# Concatenate top states and 'Other'
state_pie_data = pd.concat([top_state_counts, pd.Series({'Other': other_states_count})])

# --- City Pie Chart Data Preparation ---
# Top 5 cities based on job postings
top_cities = df['city'].value_counts().nlargest(5)
top_city_names = top_cities.index.tolist()

# Full series of city value counts
city_counts = df['city'].value_counts()
top_city_counts = city_counts.loc[top_city_names]
other_cities_count = city_counts.loc[~city_counts.index.isin(top_city_names)].sum()

# Concatenate top cities and 'Other'
city_pie_data = pd.concat([top_city_counts, pd.Series({'Other': other_cities_count})])

# Create 2 subplots in a single row
fig, axs = plt.subplots(1, 2, figsize=(18, 9))

# --- Plot the State Pie Chart on the first axis (axs[0]) ---
colors_states = plt.cm.Set2.colors[:len(state_pie_data)]
state_pie_data.plot.pie(ax=axs[0], fontsize=16, autopct='%1.1f%%', colors=colors_states)
axs[0].set_title('Job Distribution by Top 10 States and Others', fontsize=18, fontweight='bold')
```

```
axs[0].set_ylabel('')  # Hide the default y-label

# --- Plot the City Pie Chart on the second axis (axs[1]) ---
colors_cities = plt.cm.Set3.colors[:len(city_pie_data)]
city_pie_data.plot.pie(ax=axs[1], fontsize=16, autopct='%1.1f%%', colors=colors_cities)
axs[1].set_title('Job Distribution by Top 5 Cities and Others', fontsize=18, fontweight='bold')
axs[1].set_ylabel('')  # Hide the default y-label

plt.tight_layout()
plt.show()
```

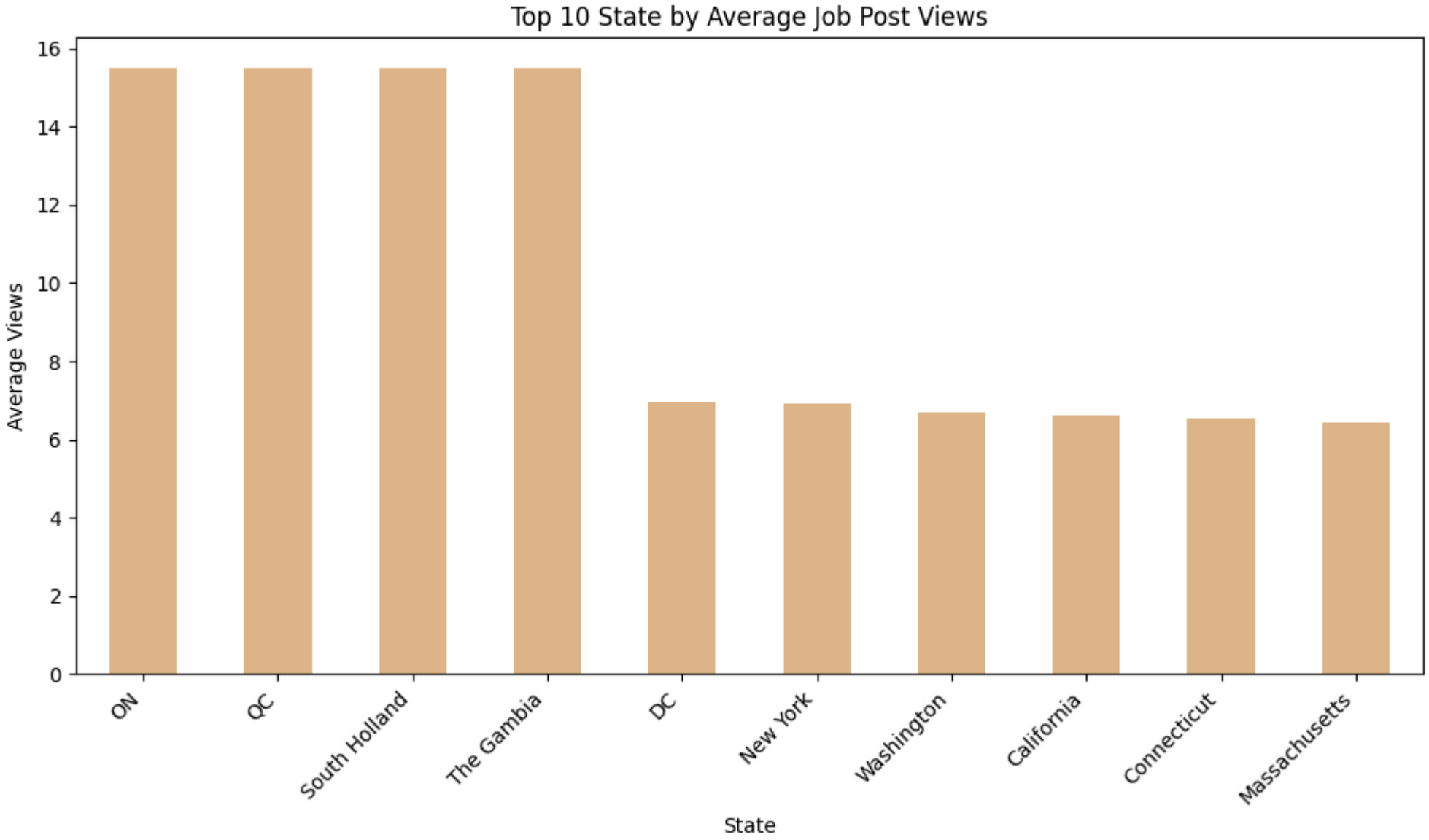




Key Insights:

- The top 10 states account for just over half the job postings, with **California (12.1%), Texas (9.5%), and New York (7.1%) leading**; however, a significant 42.9% of jobs are spread throughout all other states.
- In the city-wise distribution, the top 5 cities represent a very small fraction of total jobs, with "Other" cities making up a dominant 85.3% share.
- The **"Unknown" city group (8.2%)** highlights a notable portion of postings with **missing or ambiguous city data**.
- **Major cities** like Chicago, Houston, Atlanta, and Dallas each comprise **less than 2% of job postings**, indicating city-level concentration is much weaker compared to state-level concentration.
- Overall, **job opportunities are much more evenly spread at the city level than the state level**, with the majority of postings falling outside the most populous or well-known locations.

In [92]:
```
# Visualizes the average number of job post views per state

import matplotlib.pyplot as plt
import pandas as pd

# 1. Average views per city
city_views = df.groupby('state')['views'].mean().sort_values(ascending=False)

# Visualize average views per city
plt.figure(figsize=(10, 6))
city_views.head(10).plot(kind='bar', color='burlywood')
plt.title('Top 10 State by Average Job Post Views')
plt.xlabel('State')
plt.ylabel('Average Views')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Key Insights:

- **QC, ON, The Gambia, and South Holland** lead in average job post views, each averaging close to 16 views per post—much higher than other regions in the top 10.
- **U.S. locations** like DC, New York, Washington, California, Connecticut, and Massachusetts **all show lower average view counts**, typically between 7 and 8.
- The high averages in certain regions may indicate greater job seeker engagement per posting, but could also reflect fewer postings overall or unique local interest patterns.
- **Canadian provinces (QC and ON)** and select international locations (The Gambia, South Holland) are particularly strong in generating interest per job post compared to major U.S. states.

In [93]:
```
# List of columns to check value counts for
# columns_to_check = ['formatted_work_type', 'application_type', 'formatted_experience_level', 'skill_name', 'industry_name']

# Loop through the columns and print value counts
#for column in columns_to_check:
#    print(f"\nValue counts for '{column}':")
#    print(df[column].value_counts())
```

In [94]:
```
# Visualize the distribution of Work Type using stacked bar charts for cities and states

import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.express as px

# Get top 10 cities and states by job count
top_cities = df['city'].value_counts().nlargest(10).index.tolist()
top_states = df['state'].value_counts().nlargest(10).index.tolist()

# Filter for those locations
df_cities = df[df['city'].isin(top_cities)]
df_states = df[df['state'].isin(top_states)]

# Group work type
city_worktype = df_cities.groupby(['city', 'formatted_work_type']).size().unstack(fill_value=0)
state_worktype = df_states.groupby(['state', 'formatted_work_type']).size().unstack(fill_value=0)
work_types = city_worktype.columns.tolist()

palette = px.colors.qualitative.Bold  # Choose your preferred palette
color_map = {wt: palette[i % len(palette)] for i, wt in enumerate(work_types)}

fig = make_subplots(
    rows=1, cols=2,
    subplot_titles=('<b>Top 10 Cities</b>', '<b>Top 10 States</b>')
)

# Add stacked bars for cities (show legend)
for wt in work_types:
```

```
    fig.add_trace(go.Bar(
        x=city_worktype.index,
        y=city_worktype[wt],
        name=wt,
        legendgroup=wt,
        marker_color=color_map[wt],
        showlegend=True,  # Only here!
        marker_line_width=0
    ), row=1, col=1)

# Add stacked bars for states (hide legend)
for wt in work_types:
    fig.add_trace(go.Bar(
        x=state_worktype.index,
        y=state_worktype[wt],
        name=wt,
        legendgroup=wt,
        marker_color=color_map[wt],
        showlegend=False,  # Hide legend here!
        marker_line_width=0
    ), row=1, col=2)

fig.update_layout(
    barmode='stack',
    height=600, width=1100,
    showlegend=True,
    title_text='<b>Work Type Distribution</b>',
    title_x=0.5  # Centers the title horizontally
)
```

```
for i in range(1, 3):
    fig.update_xaxes(tickangle=45, tickfont=dict(size=12), row=1, col=i)
    fig.update_yaxes(title_text='Job Postings', row=1, col=i)

fig.show()
```

Key Insights:

- **Full-time roles** overwhelmingly **dominate job postings in all top cities and states**, greatly outnumbering other work types.
- **Contract positions** are the **next most significant category**, well represented in every major city and state, though still far behind full-time roles.
- Internship, part-time, temporary, volunteer, and "other" work types contribute much smaller shares, indicating **limited availability for alternative or flexible working arrangements**.
- **California and Texas** lead the states in total job postings for **Full-time**, with similar trends visible in larger cities, while the "Unknown" city group presents the largest single block—highlighting substantial postings without specific city data.

In [95]:

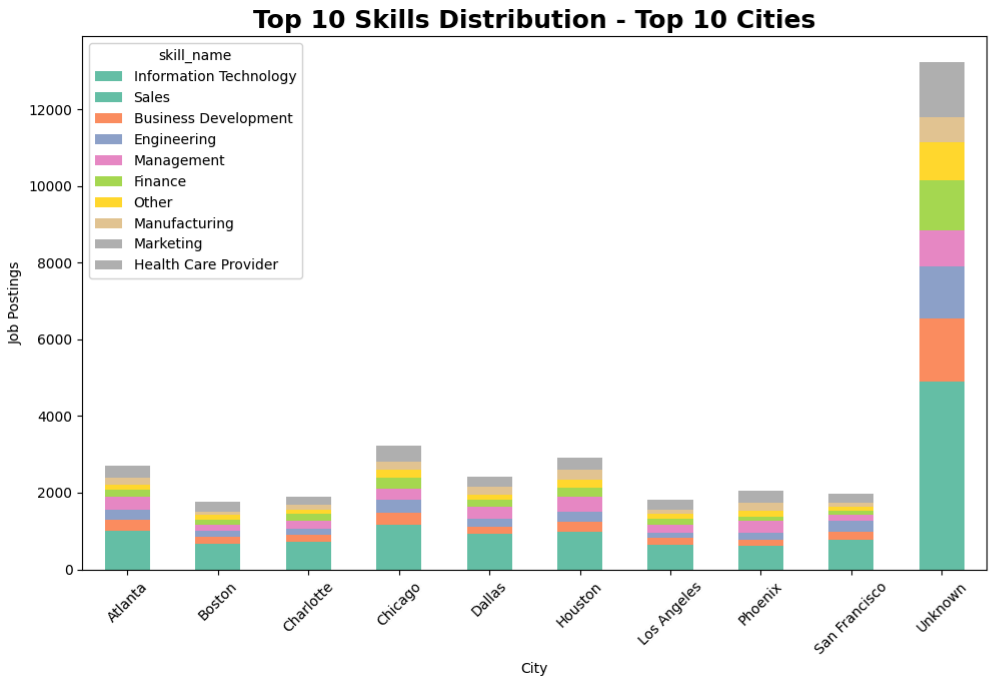
```
# Visualize the distribution of Work Experience using stacked bar charts for cities and states

import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.express as px

# Get top 10 cities and states by job count
top_cities = df['city'].value_counts().nlargest(10).index.tolist()
top_states = df['state'].value_counts().nlargest(10).index.tolist()

# Filter the DataFrame for those locations
df_cities = df[df['city'].isin(top_cities)]
df_states = df[df['state'].isin(top_states)]

# Group by experience level instead of work type
city_exp = df_cities.groupby(['city', 'formatted_experience_level']).size().unstack(fill_value=0)
state_exp = df_states.groupby(['state', 'formatted_experience_level']).size().unstack(fill_value=0)

# Get experience levels
experience_levels = city_exp.columns.tolist()

# Choose a color palette and create color map
palette = px.colors.qualitative.T10
color_map = {exp: palette[i % len(palette)] for i, exp in enumerate(experience_levels)}

fig = make_subplots(
    rows=1, cols=2,
    subplot_titles=('<b>Top 10 Cities</b>', '<b>Top 10 States</b>')
)

# Add stacked bars for cities (show legend)
for exp in experience_levels:
    fig.add_trace(go.Bar(
        x=city_exp.index,
        y=city_exp[exp],
        name=exp,
        legendgroup=exp,
        marker_color=color_map[exp],
        showlegend=True,
```

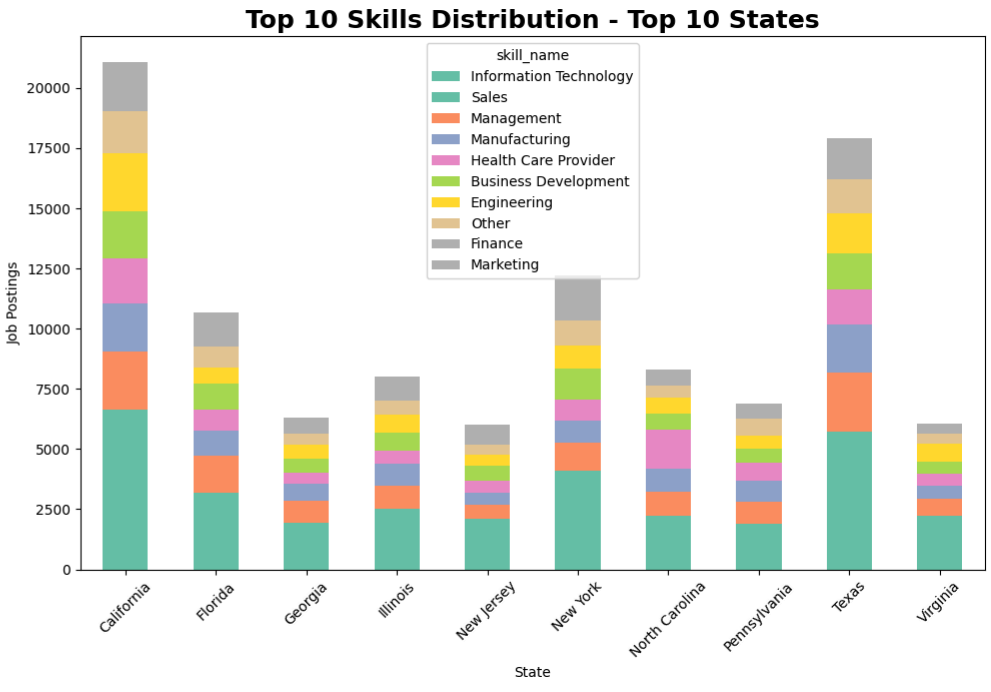
```
        marker_line_width=0
    ), row=1, col=1)

# Add stacked bars for states (hide legend)
for exp in experience_levels:
    fig.add_trace(go.Bar(
        x=state_exp.index,
        y=state_exp[exp],
        name=exp,
        legendgroup=exp,
        marker_color=color_map[exp],
        showlegend=False,
        marker_line_width=0
    ), row=1, col=2)

fig.update_layout(
    barmode='stack',
    height=600,
    width=1100,
    showlegend=True,
    title_text='<b>Work Experience Level Distribution</b>',
    title_x=0.5  # Centers the title horizontally
)

for i in range(1, 3):
    fig.update_xaxes(tickangle=45, tickfont=dict(size=12), row=1, col=i)
    fig.update_yaxes(title_text='Job Postings', row=1, col=i)

fig.show()
```

Key Insights:

- **Mid-Senior level and Entry level** are the **most prevalent experience requirements** in job postings across both cities and states, together forming the majority of opportunities.
- Among states, **California** stands out with the highest job postings, led by a **significant volume of mid-senior roles**, followed by Texas and New York also showing robust demand for both mid-senior and entry-level talent.
- **Other experience levels** (Associate, Director, Executive, Internship) make up a smaller but steady portion of postings, suggesting that while seniority and entry roles dominate, a diverse range of experience is still in demand throughout the market.

In [96]:
```
# Analyze the most popular skills in the top 10 cities and states

# Get top 10 cities and states by total job postings
top_10_cities = df['city'].value_counts().nlargest(10).index
top_10_states = df['state'].value_counts().nlargest(10).index

# Filter data for top 10 cities and states from the original dataframe
df_cities_top = df[df['city'].isin(top_10_cities)]
df_states_top = df[df['state'].isin(top_10_states)]

# Group skill name counts for top 10 cities and states
city_skill_top = df_cities_top.groupby(['city', 'skill_name']).size().unstack(fill_value=0)
state_skill_top = df_states_top.groupby(['state', 'skill_name']).size().unstack(fill_value=0)

# Get top 10 skills by total counts for cities and states
top_skills_cities = city_skill_top.sum(axis=0).nlargest(10).index
top_skills_states = state_skill_top.sum(axis=0).nlargest(10).index

# Filter to only top 10 skills
city_skill_top_10 = city_skill_top[top_skills_cities]
state_skill_top_10 = state_skill_top[top_skills_states]

# Plot stacked bar charts only
# Adjust subplots to 1 row and 2 columns
fig, axs = plt.subplots(1, 2, figsize=(20, 7))

# Bar plot for top 10 skills in top 10 cities
city_skill_top_10.plot(kind='bar', stacked=True, ax=axs[0], colormap='Set2')
axs[0].set_title('Top 10 Skills Distribution - Top 10 Cities', fontsize=18, fontweight='bold')
axs[0].set_xlabel('City')
axs[0].set_ylabel('Job Postings')
axs[0].tick_params(axis='x', rotation=45)
#axs[0].legend(title='Skill Name', bbox_to_anchor=(1.05, 1), loc='upper left')

# Bar plot for top 10 skills in top 10 states
state_skill_top_10.plot(kind='bar', stacked=True, ax=axs[1], colormap='Set2')
axs[1].set_title('Top 10 Skills Distribution - Top 10 States', fontsize=18, fontweight='bold')
axs[1].set_xlabel('State')
axs[1].set_ylabel('Job Postings')
axs[1].tick_params(axis='x', rotation=45)
#axs[1].legend(title='Skill Name', bbox_to_anchor=(1.05, 1), loc='upper left')

plt.tight_layout()
plt.show()
```



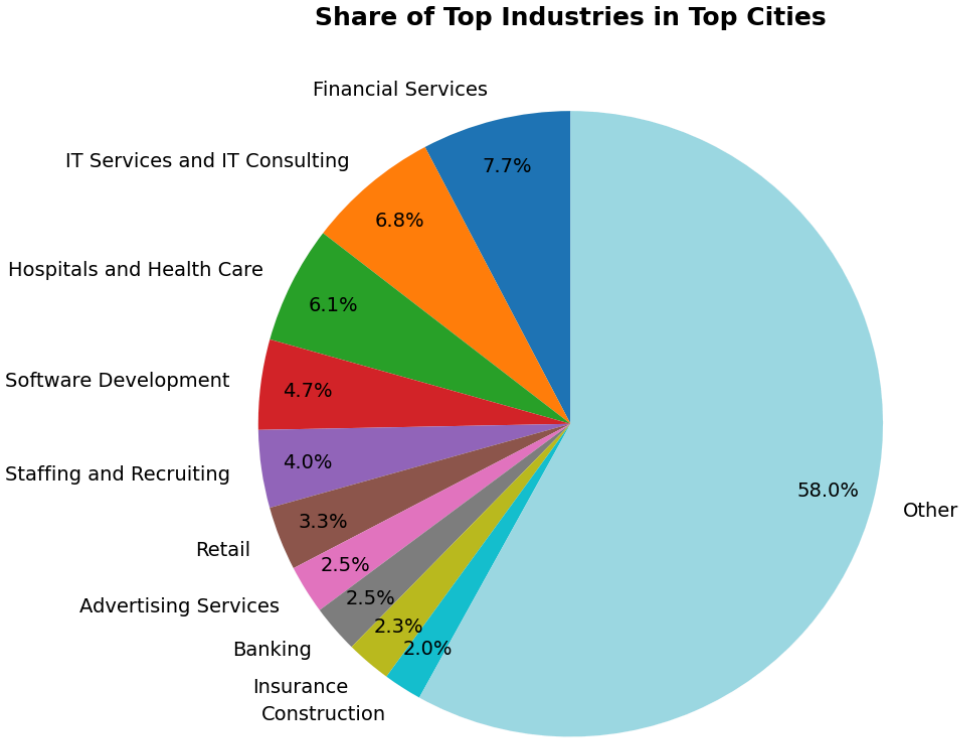


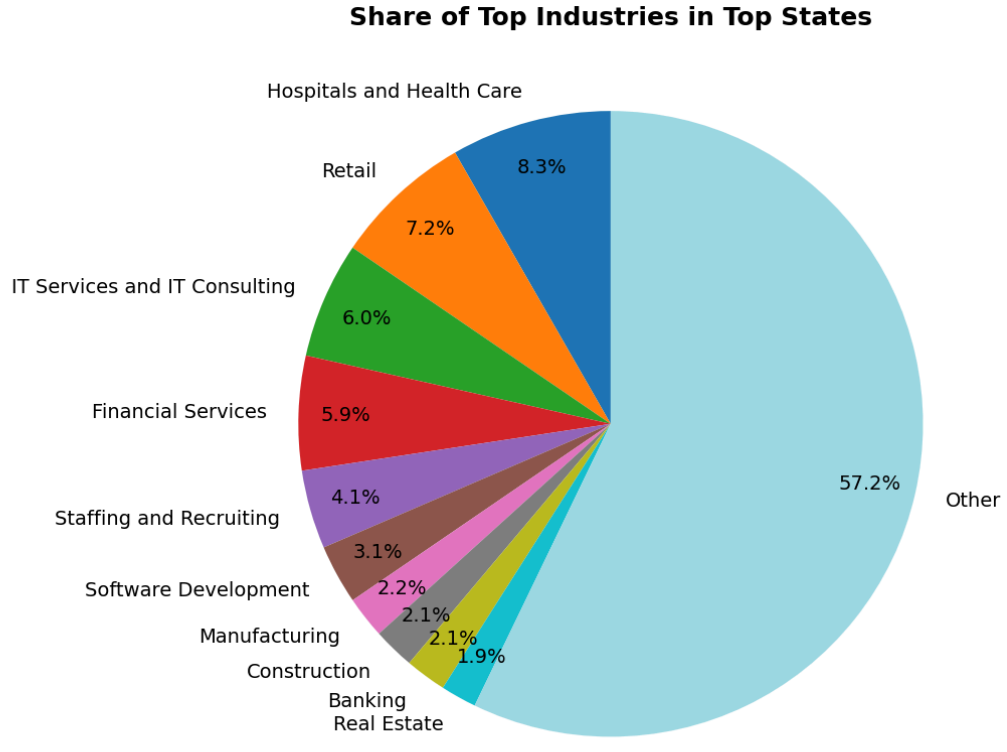
Key Insights:

- **Information Technology leads job postings** in every top city and state, far surpassing other skills.
- **Sales, Management, and Business Development** consistently follow IT, but their numbers are significantly lower in comparison.
- In cities, the "Unknown" group once again registers the highest count, reflecting incomplete city data in postings and possibly skewing visibility of city-specific trends.
- Among states, **California and Texas** display the **highest demand for most top skills**, especially IT, indicating major tech and business hubs.
- **Other core skills**—Engineering, Healthcare Provider, Finance, and Marketing—remain steady contributors across locations, showing broad-based demand rather than single-skill specialization.

In [97]:
```
# Visualize the distribution of industries in the top cities and states.
#  It generates four plots: two horizontal bar charts showing the top 10 cities and states by job postings and
#  two pie charts showing the percentage share of different industries within those locations.

# Get top 10 cities and states by total job postings
top_10_cities_series = df['city'].value_counts().nlargest(10)
top_10_cities = top_10_cities_series.index
top_10_states_series = df['state'].value_counts().nlargest(10)
top_10_states = top_10_states_series.index

# Filter data for top 10 cities and states from the original dataframe
df_cities_top = df[df['city'].isin(top_10_cities)]
df_states_top = df[df['state'].isin(top_10_states)]

# Group industry name counts for top 10 cities and states
city_industry_top = df_cities_top.groupby(['city', 'industry_name']).size().unstack(fill_value=0)
state_industry_top = df_states_top.groupby(['state', 'industry_name']).size().unstack(fill_value=0)

# Get top 10 industries by total counts
top_industries_cities = city_industry_top.sum(axis=0).nlargest(10).index
top_industries_states = state_industry_top.sum(axis=0).nlargest(10).index

# Filter to only top 10 industries for each location type
city_industry_top_10 = city_industry_top[top_industries_cities]
state_industry_top_10 = state_industry_top[top_industries_states]

# Create "Other" category for industries outside the top 10
city_industry_other = city_industry_top.drop(columns=top_industries_cities, errors='ignore').sum(axis=1)
state_industry_other = state_industry_top.drop(columns=top_industries_states, errors='ignore').sum(axis=1)

# Prepare data for pie charts
city_industry_totals = city_industry_top_10.sum(axis=0)
city_industry_other_total = city_industry_other.sum()
city_pie_data_final = pd.concat([city_industry_totals, pd.Series({'Other': city_industry_other_total})])

state_industry_totals = state_industry_top_10.sum(axis=0)
state_industry_other_total = state_industry_other.sum()
state_pie_data_final = pd.concat([state_industry_totals, pd.Series({'Other': state_industry_other_total})])

# --- Plot Pie Charts ---
fig_pies, axs_pies = plt.subplots(1, 2, figsize=(20, 10))

# Pie chart for cities
if not city_pie_data_final.empty and city_pie_data_final.sum() > 0:
    city_pie_data_final.plot.pie(ax=axs_pies[0], fontsize=14, autopct='%1.1f%%', startangle=90, colormap='tab20', pctdistance=0.85)
    axs_pies[0].set_title('Share of Top Industries in Top Cities', fontsize=18, fontweight='bold')
    axs_pies[0].set_ylabel('')

# Pie chart for states
if not state_pie_data_final.empty and state_pie_data_final.sum() > 0:
    state_pie_data_final.plot.pie(ax=axs_pies[1], fontsize=14, autopct='%1.1f%%', startangle=90, colormap='tab20', pctdistance=0.85)
    axs_pies[1].set_title('Share of Top Industries in Top States', fontsize=18, fontweight='bold')
```

```
    axs_pies[1].set_ylabel('')

plt.tight_layout()
plt.show()
```





Key Insights:

- In both cities and states, "Other" industries constitute the largest share—58% for cities and 57.2% for states—indicating a **highly fragmented industry landscape** beyond the leading sectors.
- **Financial Services dominates** in cities (7.7%), while Hospitals and Health Care lead in states (8.3%), showing some regional variation in top industry presence.
- **IT Services and IT Consulting, Retail, and Staffing and Recruiting** consistently appear as **major sectors** in both breakdowns, though their percentage shares are slightly higher in cities.
- Industries like Software Development, Construction, and Banking hold moderate to small shares and remain significant players but do not reach double-digit percentages.
- The distribution illustrates a broad spectrum of job opportunities across industries, with **no single sector overwhelmingly dominating** outside the large "Other" category.

In [98]:
```
# Visualizes the data with a clustered bar chart of the top 10 skills required for job postings across the top 10 cities

import plotly.graph_objects as go

# Grouping/subsetting code
location_skill = df.groupby(['city', 'skill_name']).size().unstack(fill_value=0)
city_total = location_skill.sum(axis=1)
top_cities = city_total.sort_values(ascending=False).head(10).index
location_skill_top = location_skill.loc[top_cities]
skill_total = location_skill_top.sum(axis=0)
top_skills = skill_total.sort_values(ascending=False).head(10).index
final_data = location_skill_top[top_skills]

# Assuming `final_data` is a DataFrame where rows = cities, columns = top skills, values = counts
cities = final_data.index.tolist()
skills = final_data.columns.tolist()

fig = go.Figure()

# For each skill, add a Bar trace with counts for all cities
for skill in skills:
    fig.add_trace(go.Bar(
        x=cities,
        y=final_data[skill],
        name=skill
    ))


fig.update_layout(
    title='<b>Top 10 Skills in Top 10 Cities (Job Count by Skill and City)</b>',
    title_x=0.5,  # centers the title horizontally
    xaxis_title='City',
    yaxis_title='No of Jobs',
    barmode='group',  # grouped bars
    xaxis_tickangle=-45,
    legend_title='<b>Skill</b>',
    width=max(1000, 120 * len(cities)),
    height=max(600, 40 * len(skills)),
    legend=dict(
```

```
        x=0.95,       # x-position (0=left, 1=right)
        y=0.95,       # y-position (0=bottom, 1=top)
        xanchor='right',
        yanchor='top',
        bgcolor='rgba(255,255,255,0.7)',  # semi-transparent background
        bordercolor='black',
        borderwidth=1
    )
)


#fig.update_yaxes(type='log')
fig.show()
```

Key Insights:

- **Information Technology** and **Sales** are the **most in-demand skills** in nearly every top city, consistently leading job counts over other skill categories.
- Job opportunities decrease sharply from the largest city group ("Unknown") to well-known metros like Chicago, Houston, Atlanta, and the rest, illustrating a strong concentration of job postings in the top few urban centers.
- After tech and sales, **Business Development, Engineering, and Management** are the **next most prominent skills** across the major cities, reflecting broad corporate and technical demand.
- Several cities (Chicago, Houston, Atlanta, Dallas) show a more even distribution among the top skills, while IT and Sales clearly dominate in some markets.
- The **"Unknown"** group likely aggregates postings where **city data is missing**; its high volume suggests a significant portion of jobs lack precise city tagging.

In [99]:
```
# Visualize the distribution of job types by country using Plotly

import plotly.graph_objects as go

# Extract unique job types and countries
work_types = df['formatted_work_type'].unique()
countries = df['country'].unique()

# Brand colors (extend the list if more work types exist)
#colors = ["#1FB8CD", "#DB4545", "#2E8B57", "#5D878F", "#D2BA4C"]

# Use a sequential color scale from Plotly
palette = px.colors.sequential.Agsunset

# Map work types to colors from the palette evenly
colors = [palette[int(i * (len(palette)-1) / (len(work_types)-1))] for i in range(len(work_types))]

fig = go.Figure()

# Add a bar for each work type, grouped by country
for i, work_type in enumerate(work_types):
    values = [df[(df['country'] == country) & (df['formatted_work_type'] == work_type)].shape[0] for country in countries]
    fig.add_trace(go.Bar(
        name=work_type,
        x=countries,
        y=values,
        marker_color=colors[i % len(colors)],
    ))

fig.update_layout(
    barmode='group',
    title='<b>Country wise Work Types (Log Scale)</b>',
    title_x =0.5,
    xaxis_title='Country',
    yaxis_title='Number of Jobs',
    legend_title='Work Type',
)
fig.update_yaxes(type='log')
fig.show()
```

Key Insights:

- The **United States** has an overwhelming **majority of job postings** across all work types compared to Canada, Brazil, and Argentina.
- **Full-time, internship, and contract roles dominate** in the U.S., while other countries contribute minimally and mainly in a single or two work types.
- The huge disparity is highlighted using the log scale, with non-U.S. markets barely visible in comparison.
- Data suggests potential platform or sample **bias heavily favoring U.S. listings**, not global market representation.

In [ ]:
```

```