Project Fake Finder - Progress Report 1

May 13, 2025

Project Progress Report # 1

Team Members: John DeLeon, Darshan Joshi, Dae-Breona Armour

CS 6330-DS1 Data Science | Spring B 2025

Dr. Dogdu | Angelo State University

Department of Computer Science

1 Data Exploratory Data Analysis (EDA)

1.1 Environment Setup for EDA

Before we start our Exploratory Data Analysis, there are some additional packages we will need to install.

- kagglehub - Import our Kaggle dataset
- textstat - To gain additional insight with our text fields

[1]:
```
%%capture
!pip install kagglehub > /dev/null 2>&1
!pip install textstat
```

1.1.1 Purpose:

Additionally, we want to update some of the default pandas options and set basic logging configuration to get a better view of our dataset during our data exploration.

[2]:
```
import os
import logging
from typing import List
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path

# Set up logging for debug information:
logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

# Option 1: Set display options to show all rows:
```

```
pd.set_option('display.max_rows', None)

# Option 2: Set display options to show all columns:
pd.set_option('display.max_columns', None)

# Option 3: Set the width to show all content:
pd.set_option('display.width', None)

# Option 4: Set the max column width to avoid truncating cell contents:
pd.set_option('display.max_colwidth', None)
```

1.2 Data Wrangling

1.2.1 Step 1: Discovering

Discovery, also called data exploration, familiarizes the data scientist with source data in preparation for subsequent steps.

[25]:
```
import kagglehub
import os
import pandas as pd

# Download latest version of the dataset
path = kagglehub.dataset_download("rishantenis/
  ↪real-and-fake-jobs-posting-prediction")
csv_file = [f for f in os.listdir(path) if f.endswith('.csv')][0]
df = pd.read_csv(os.path.join(path,csv_file))
```

Purpose: Get the first 5 rows of our dataset to make sure it loaded correctly and that there are no issues. Since the text columns has a lot of data we view those separately in an additional code cell.

[4]:
```
df[
    ['job_id',
     'title',
     'location',
     'department',
     'salary_range',
     'telecommuting',
     'has_company_logo',
     'has_questions',
     'employment_type',
     'required_experience',
     'required_education',
     'industry',
     'function',
     'fraudulent'
     ]
```

```
    ].head()
```

[4]:
```
   job_id                                      title            location  \
0       1                          Marketing Intern    US, NY, New York
1       2  Customer Service - Cloud Video Production      NZ, , Auckland
2       3    Commissioning Machinery Assistant (CMA)       US, IA, Wever
3       4          Account Executive - Washington DC  US, DC, Washington
4       5                       Bill Review Manager  US, FL, Fort Worth

  department salary_range  telecommuting  has_company_logo  has_questions  \
0  Marketing          NaN              0                 1              0
1    Success          NaN              0                 1              0
2        NaN          NaN              0                 1              0
3      Sales          NaN              0                 1              0
4        NaN          NaN              0                 1              1

  employment_type required_experience required_education  \
0           Other          Internship                NaN
1       Full-time      Not Applicable                NaN
2             NaN                NaN                NaN
3       Full-time    Mid-Senior level  Bachelor's Degree
4       Full-time    Mid-Senior level  Bachelor's Degree

                   industry              function  fraudulent
0                       NaN             Marketing           0
1  Marketing and Advertising      Customer Service           0
2                       NaN                   NaN           0
3          Computer Software                 Sales           0
4     Hospital & Health Care  Health Care Provider           0
```

[5]:
```
df[
    ['job_id',
     'company_profile',
     'description',
     'requirements',
     'benefits'
     ]
    ].head(1)
```

[5]:
```
   job_id  \
0       1

company_profile  \
0  We're Food52, and we've created a groundbreaking and award-winning cooking
site. We support, connect, and celebrate home cooks, and give them everything
they need in one place.We have a top editorial, business, and engineering team.
We're focused on using technology to find new and better ways to connect people
```

```
around their specific food interests, and to offer them superb, highly curated
information about food and cooking. We attract the most talented home cooks and
contributors in the country; we also publish well-known professionals like Mario
Batali, Gwyneth Paltrow, and Danny Meyer. And we have partnerships with Whole
Foods Market and Random House.Food52 has been named the best food website by the
James Beard Foundation and IACP, and has been featured in the New York Times,
NPR, Pando Daily, TechCrunch, and on the Today Show.We're located in Chelsea, in
New York City.

                 description  \
0  Food52, a fast-growing, James Beard Award-winning online food community and
crowd-sourced and curated recipe hub, is currently interviewing full- and part-
time unpaid interns to work in a small team of editors, executives, and
developers in its New York City headquarters.Reproducing and/or repackaging
existing Food52 content for a number of partner sites, such as Huffington Post,
Yahoo, Buzzfeed, and more in their various content management systemsResearching
blogs and websites for the Provisions by Food52 Affiliate ProgramAssisting in
day-to-day affiliate program support, such as screening affiliates and assisting
in any affiliate inquiriesSupporting with PR &amp;amp; Events when neededHelping
with office administrative work, such as filing, mailing, and preparing for
meetingsWorking with developers to document bugs and suggest improvements to the
siteSupporting the marketing and executive staff

                                          requirements  \
0  Experience with content management systems a major plus (any blogging
counts!)Familiar with the Food52 editorial voice and aestheticLoves food,
appreciates the importance of home cooking and cooking with the
seasonsMeticulous editor, perfectionist, obsessive attention to detail, maddened
by typos and broken links, delighted by finding and fixing themCheerful under
pressureExcellent communication skillsA+ multi-tasker and juggler of
responsibilities big and smallInterested in and engaged with social media like
Twitter, Facebook, and PinterestLoves problem-solving and collaborating to drive
Food52 forwardThinks big picture but pitches in on the nitty gritty of running a
small company (dishes, shopping, administrative support)Comfortable with the
realities of working for a startup: being on call on evenings and weekends, and
working long hours

  benefits
0      NaN
```

Results: Our dataset we loaded seems to match the structure we expect from Kaggle and now we will keep doing further discovery.

Purpose: Get a high-level summary of our data's structure.

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17880 entries, 0 to 17879
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   job_id               17880 non-null  int64
 1   title                17880 non-null  object
 2   location             17534 non-null  object
 3   department           6333 non-null   object
 4   salary_range         2868 non-null   object
 5   company_profile      14572 non-null  object
 6   description          17879 non-null  object
 7   requirements         15184 non-null  object
 8   benefits             10668 non-null  object
 9   telecommuting        17880 non-null  int64
 10  has_company_logo     17880 non-null  int64
 11  has_questions        17880 non-null  int64
 12  employment_type      14409 non-null  object
 13  required_experience  10830 non-null  object
 14  required_education   9775 non-null   object
 15  industry             12977 non-null  object
 16  function             11425 non-null  object
 17  fraudulent           17880 non-null  int64
dtypes: int64(5), object(13)
memory usage: 2.5+ MB
```

Results: From using the "info()" function we understand that we have the 18 columns that we expect, the 17,880 rows we expect, and we understand at a high-level our data types, and how many "non-null" values we have for each column. Next let's dig deeper into the Null fields we have.

Purpose: From order of greatest to least, we want to see what percentage of a given fields values are Null, we also want to start thinking about which fields we can probably use for our Data Science Model. If a field has too many Nulls or if real-world job posting don't have that data then we may suggest to drop those fields.

[7]:
```python
def inspect_data(df: pd.DataFrame) -> pd.DataFrame:
    """
    Inspect the dataset for missing values, datatypes, and number of unique␣
 ↪values.

    Args:
        (Pandas DataFrame): DataFrame to inspect.

    Returns:
        Summary Pandas DataFrame with information per column.
    """
    try:
```

```
        summary = pd.DataFrame({
            'Data Type': df.dtypes,
            'Missing Values': df.isnull().sum(),
            'Missing %': round((df.isnull().mean() * 100), 2),
            'Unique Values': df.nunique(),
            # 'Sample Value': df.iloc[0]
        })
        summary = summary.sort_values(by='Missing %', ascending=False)
        logging.info("Data inspection completed.")
        return summary
    except Exception as e:
        logging.error(f"Error during data inspection: {e}")
        raise

# Run inspection
data_summary = inspect_data(df)

data_summary
```

```
INFO: Data inspection completed.
```

[7]:

	Data Type	Missing Values	Missing %	Unique Values
salary_range	object	15012	83.96	874
department	object	11547	64.58	1337
required_education	object	8105	45.33	13
benefits	object	7212	40.34	6203
required_experience	object	7050	39.43	7
function	object	6455	36.10	37
industry	object	4903	27.42	131
employment_type	object	3471	19.41	5
company_profile	object	3308	18.50	1709
requirements	object	2696	15.08	11965
location	object	346	1.94	3105
description	object	1	0.01	14801
title	object	0	0.00	11231
job_id	int64	0	0.00	17880
telecommuting	int64	0	0.00	2
has_questions	int64	0	0.00	2
has_company_logo	int64	0	0.00	2
fraudulent	int64	0	0.00	2

Results: Suggested that we Drop the following Columns later on: salary_range, department, required_education, benefits, required_experience, function, industry, location, job_id, has_questions, has_company_logo.

We reccommend droping the above fields because some of them aren't useful in identifying fraudulent jobs even if the columns didn't have so many nulls. For other columns, when looking at real world job posting, we don't always see this information and ideally we want the fields used in our

training data to match the real-world as closely as possible.

Purpose: Checking for Duplicate Rows of Data to see if we need to de-duplicate our dataset later on.

[8]:
```python
def check_duplicates(df, subset=None):
    """
    Check a DataFrame for duplicate rows and return statistics.

    Parameters:
    -----------
    df : pandas.DataFrame
        The DataFrame to check for duplicates
    subset : list, optional
        List of column names to consider for identifying duplicates.
        If None, use all columns.

    Returns:
    --------
    dict
        Dictionary containing:
        - 'has_duplicates': Boolean indicating if duplicates exist
        - 'duplicate_count': Number of duplicate rows
        - 'duplicate_rows': DataFrame containing only the duplicate rows
        - 'value_counts': Series showing counts for each duplicated row
    """
    # Get value counts for each unique row combination
    if subset is None:
        value_counts = df.value_counts()
    else:
        value_counts = df.value_counts(subset=subset)

    # Find rows with count > 1 (duplicates)
    duplicated_rows = value_counts[value_counts > 1]

    # Calculate total number of duplicate rows (excluding first occurrences)
    duplicate_count = sum(count - 1 for count in duplicated_rows)

    # Find the actual duplicate rows in the DataFrame
    if subset is None:
        mask = df.duplicated(keep=False)
    else:
        mask = df.duplicated(subset=subset, keep=False)

    duplicate_rows_df = df[mask].copy()

    # Return results as a dictionary
```

```
    result = {
        'has_duplicates': len(duplicated_rows) > 0,
        'duplicate_count': duplicate_count,
        'duplicate_rows': duplicate_rows_df,
        'value_counts': duplicated_rows
    }

    return result

# Check for dupes:
num_duplicates = check_duplicates(df)
print(f"Has duplicates: {num_duplicates['has_duplicates']}")
print(f"Number of duplicate rows: {num_duplicates['duplicate_count']}")
print("Duplicate rows:")
print(num_duplicates['duplicate_rows'])
```

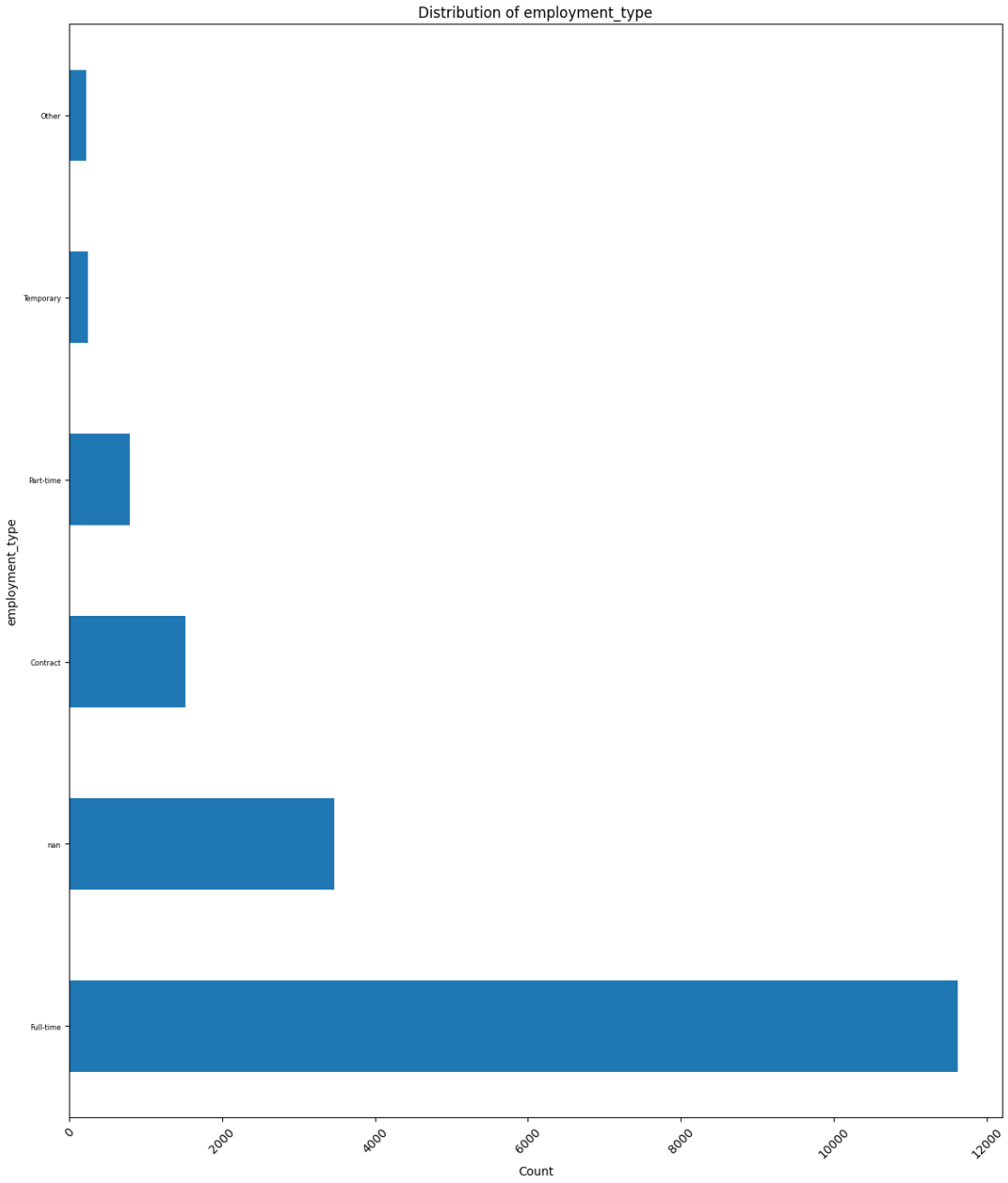
```
Has duplicates: False
Number of duplicate rows: 0
Duplicate rows:
Empty DataFrame
Columns: [job_id, title, location, department, salary_range, company_profile,
description, requirements, benefits, telecommuting, has_company_logo,
has_questions, employment_type, required_experience, required_education,
industry, function, fraudulent]
Index: []
```

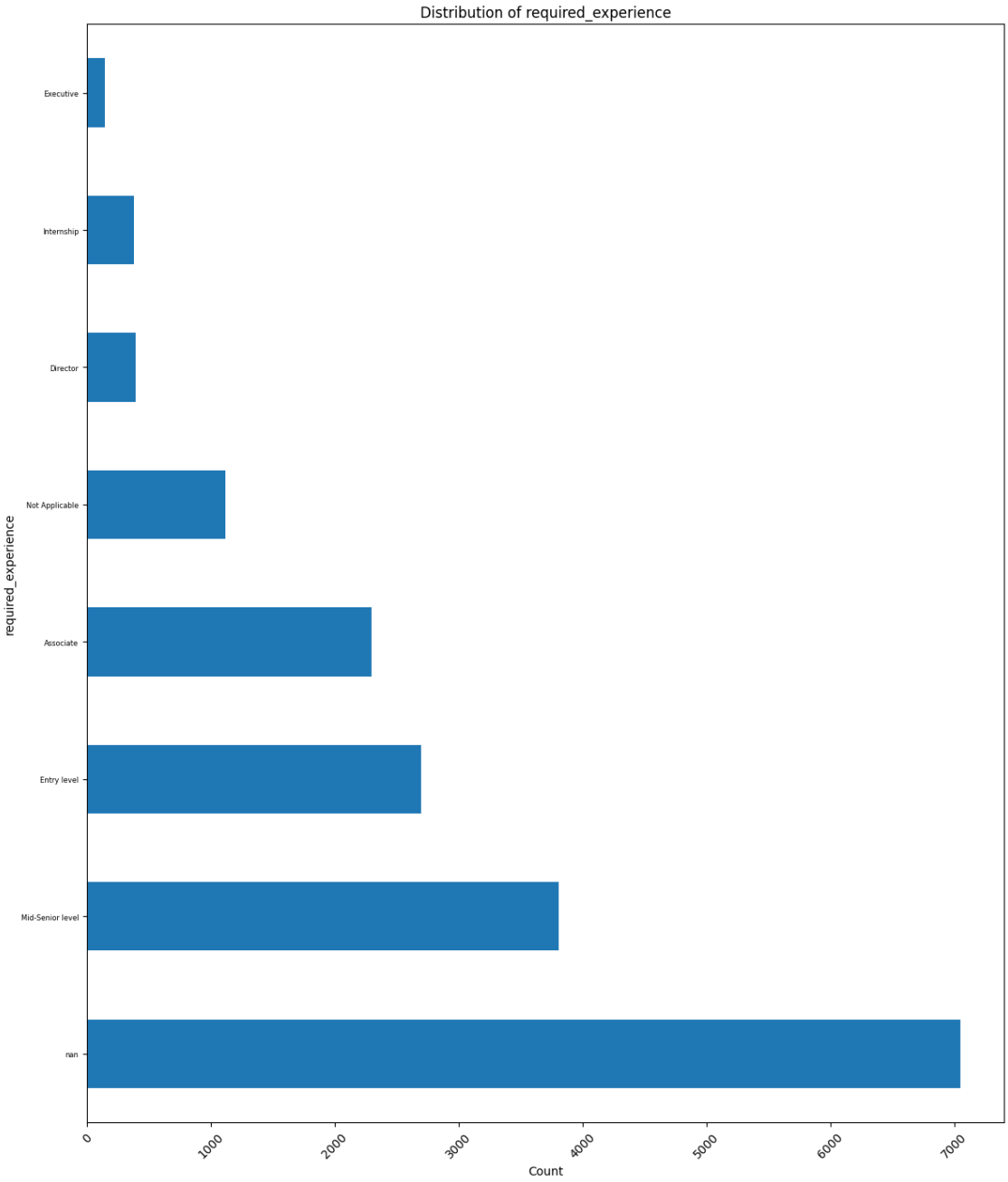
Results: Seems like we don't have any duplicate rows of data in our dataset so we won't need to do any "de-duplication" in our "Cleaning" step.

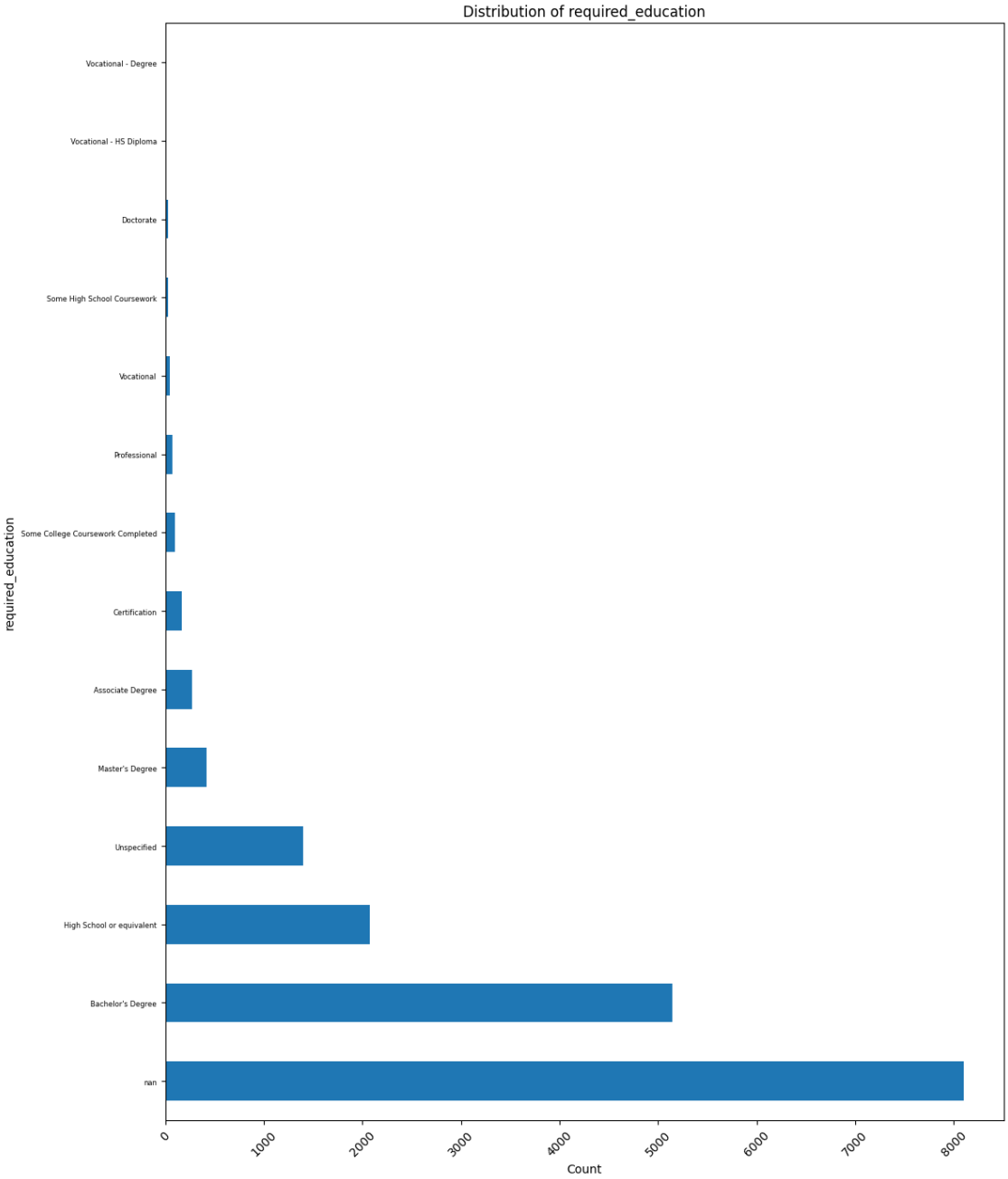
Purpose: The purpose of the below function is to visualize the distribution of values in categorical and binary columns from the dataset. For each specified column, it generates a horizontal bar chart that shows how frequently each category or value appears. This helps in understanding the balance, skewness, and patterns within categorical variables, which is critical for exploratory data analysis and for identifying potential data issues (such as class imbalance).

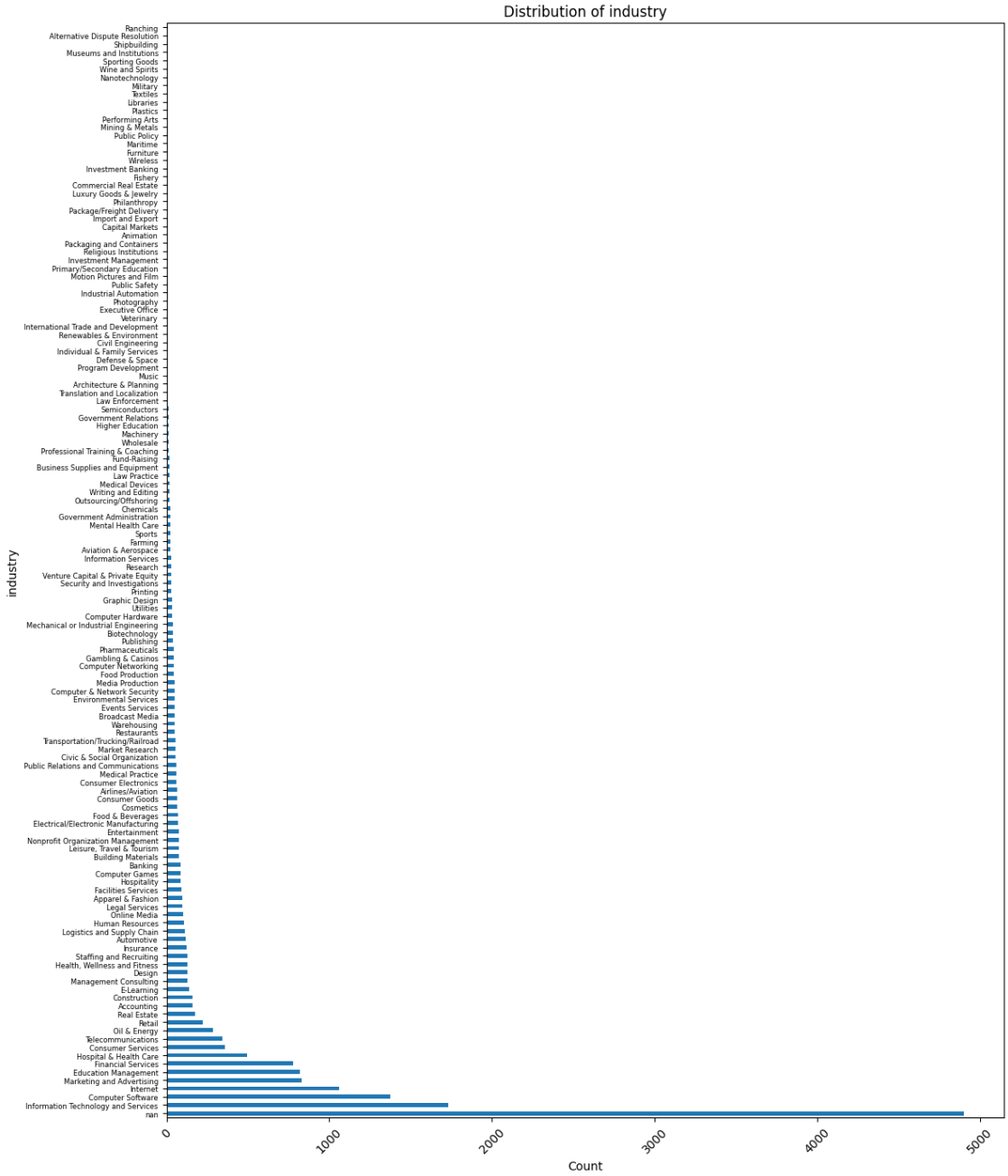
[9]:
```
def plot_categorical_distributions(df: pd.DataFrame, columns: list):
    """
    Plot bar charts for categorical columns to show value distributions.

    Args:
        df (pd.DataFrame): The dataset.
        columns (list): List of categorical & binary column names.
    """
    for col in columns:
        try:
            plt.figure(figsize=(12, 14))
            df[col].value_counts(dropna=False).plot(kind='barh')
```

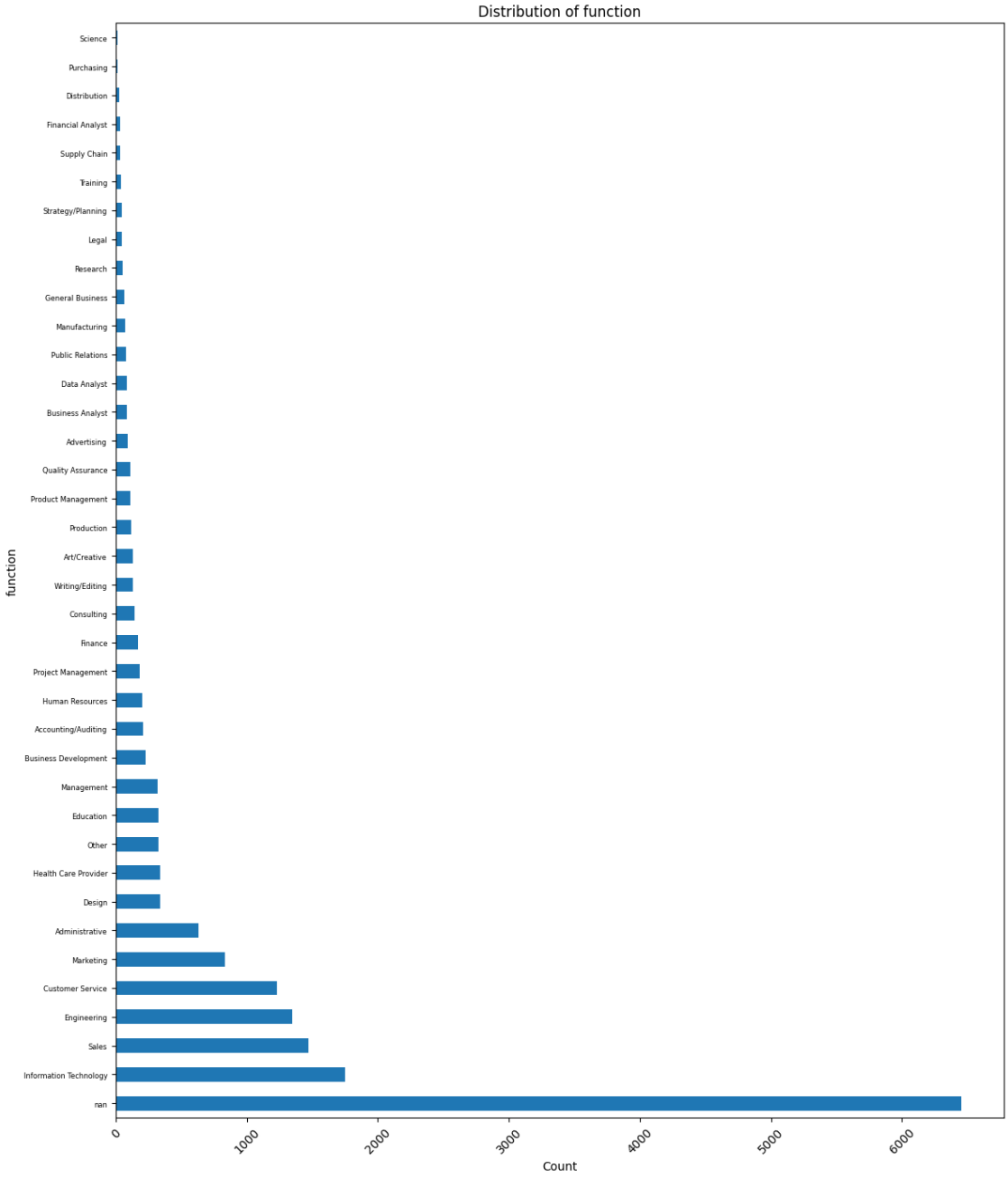
```
            plt.title(f'Distribution of {col}')
            plt.xlabel('Count')
            plt.ylabel(col)
            plt.xticks(rotation=45)
            plt.yticks(fontsize=6)
            plt.tight_layout()
            plt.show()
        except Exception as e:
            logging.error(f"Failed to plot distribution for {col}: {e}")

# Select categorical and binary columns to visualize
categorical_columns = [
    'employment_type', 'required_experience', 'required_education',
    'industry', 'function', 'telecommuting', 'has_company_logo',
    'has_questions', 'fraudulent'
]

# Plot value distributions
plot_categorical_distributions(df, categorical_columns)
```

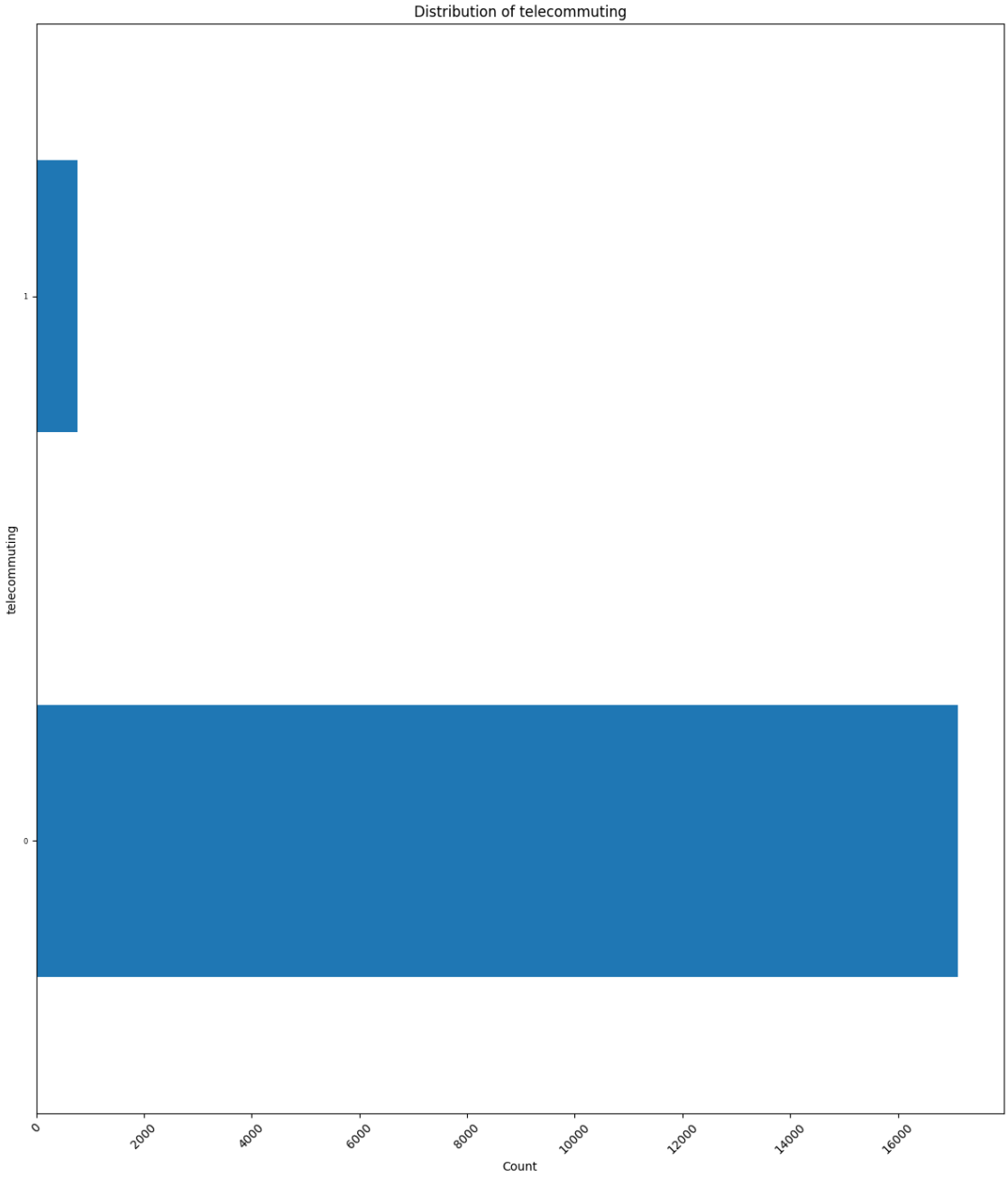

Distribution of employment_type
Other
Temporary
Part-time
Contract
nan
Full-time
employment_type
0
2000
4000
6000
8000
10000
12000
Count

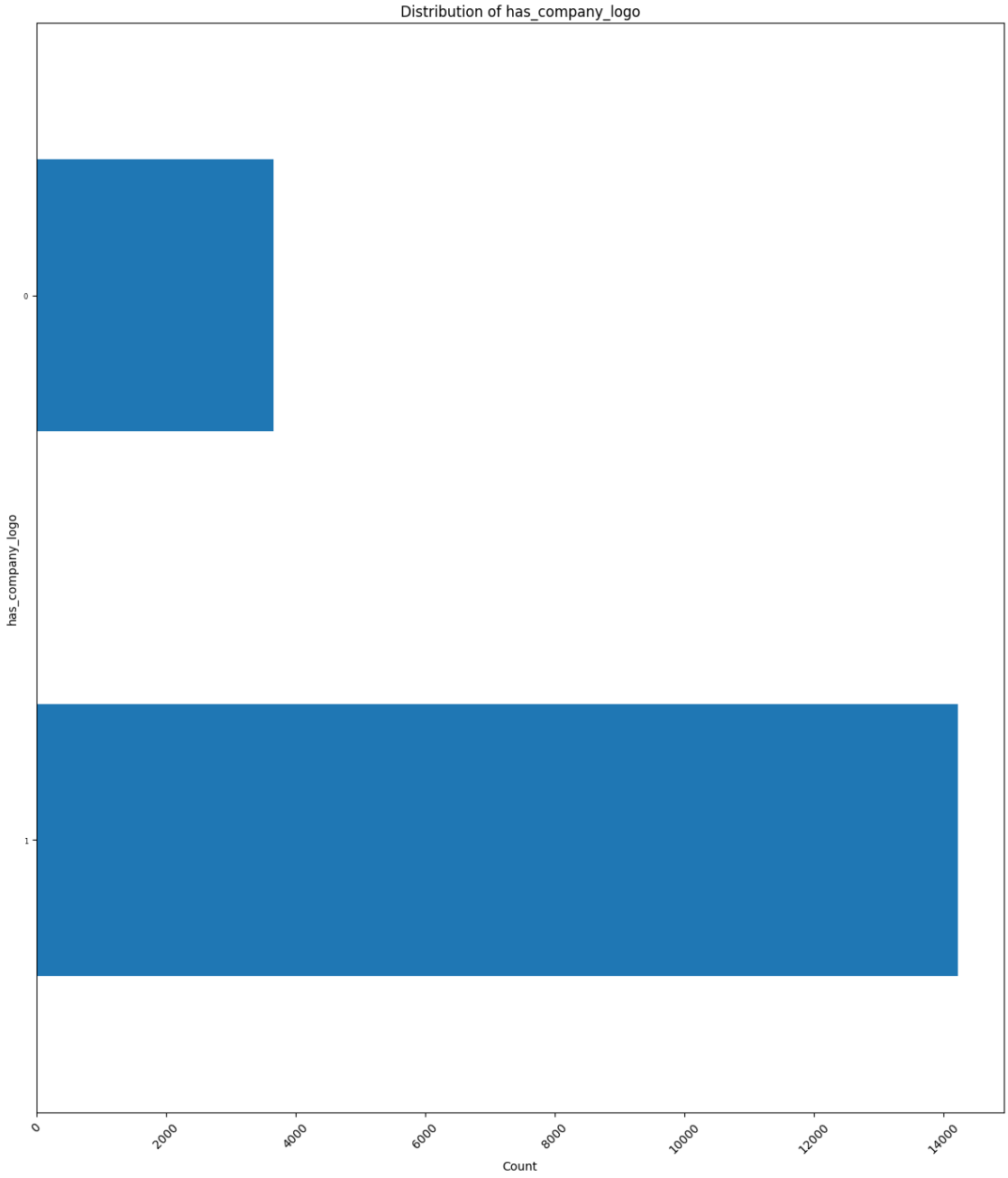

Distribution of required_experience
Executive
Internship
Director
Not Applicable
Associate
Entry level
Mid-Senior level
nan
required_experience
0
1000
2000
3000
4000
5000
6000
7000
Count

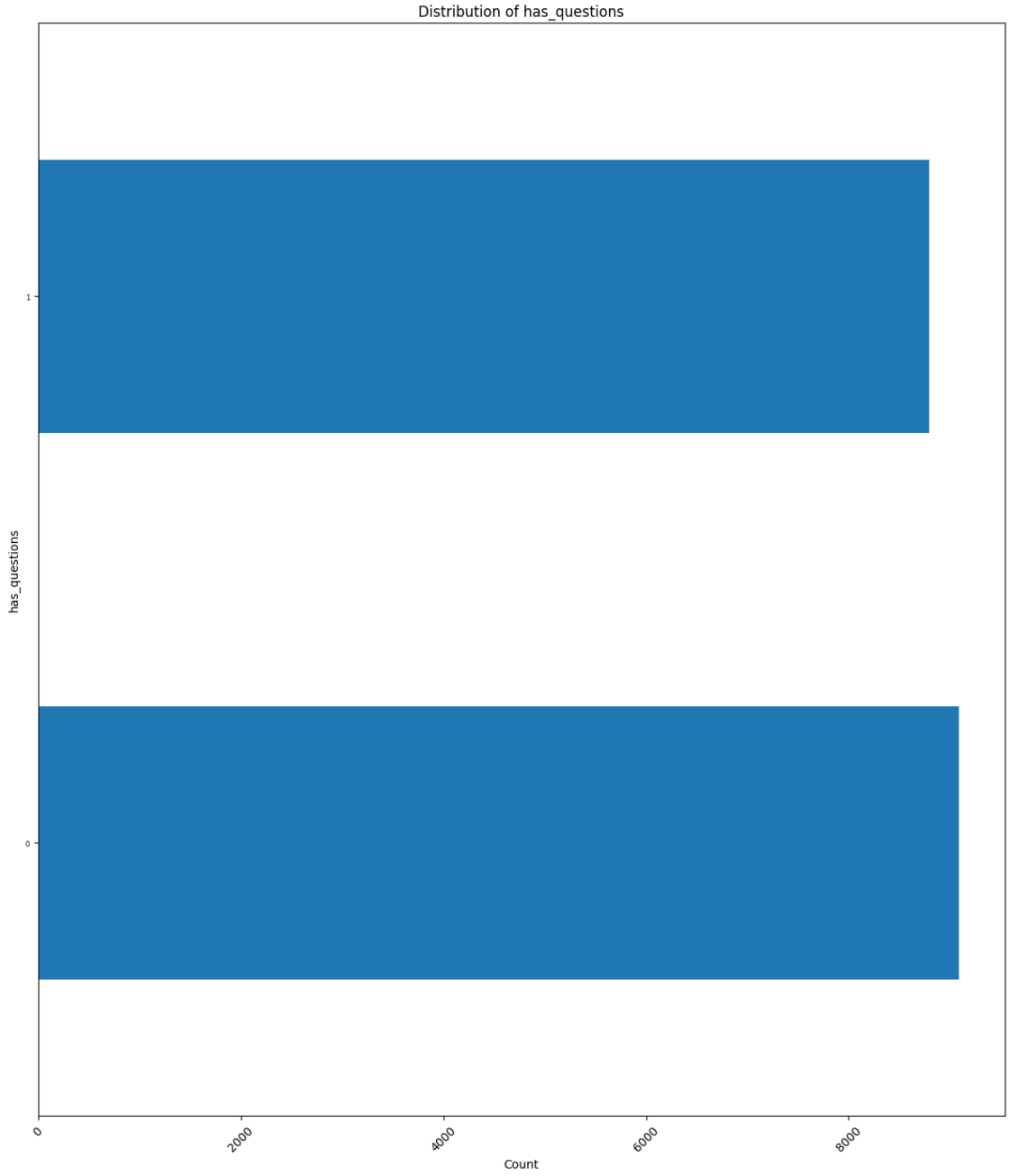

Distribution of required_education
Vocational - Degree
Vocational - HS Diploma
Doctorate
Some High School Coursework
Vocational
Professional
Some College Coursework Completed
Certification
Associate Degree
Master's Degree
Unspecified
High School or equivalent
Bachelor's Degree
nan
required_education
0
1000
2000
3000
4000
5000
6000
7000
8000
Count

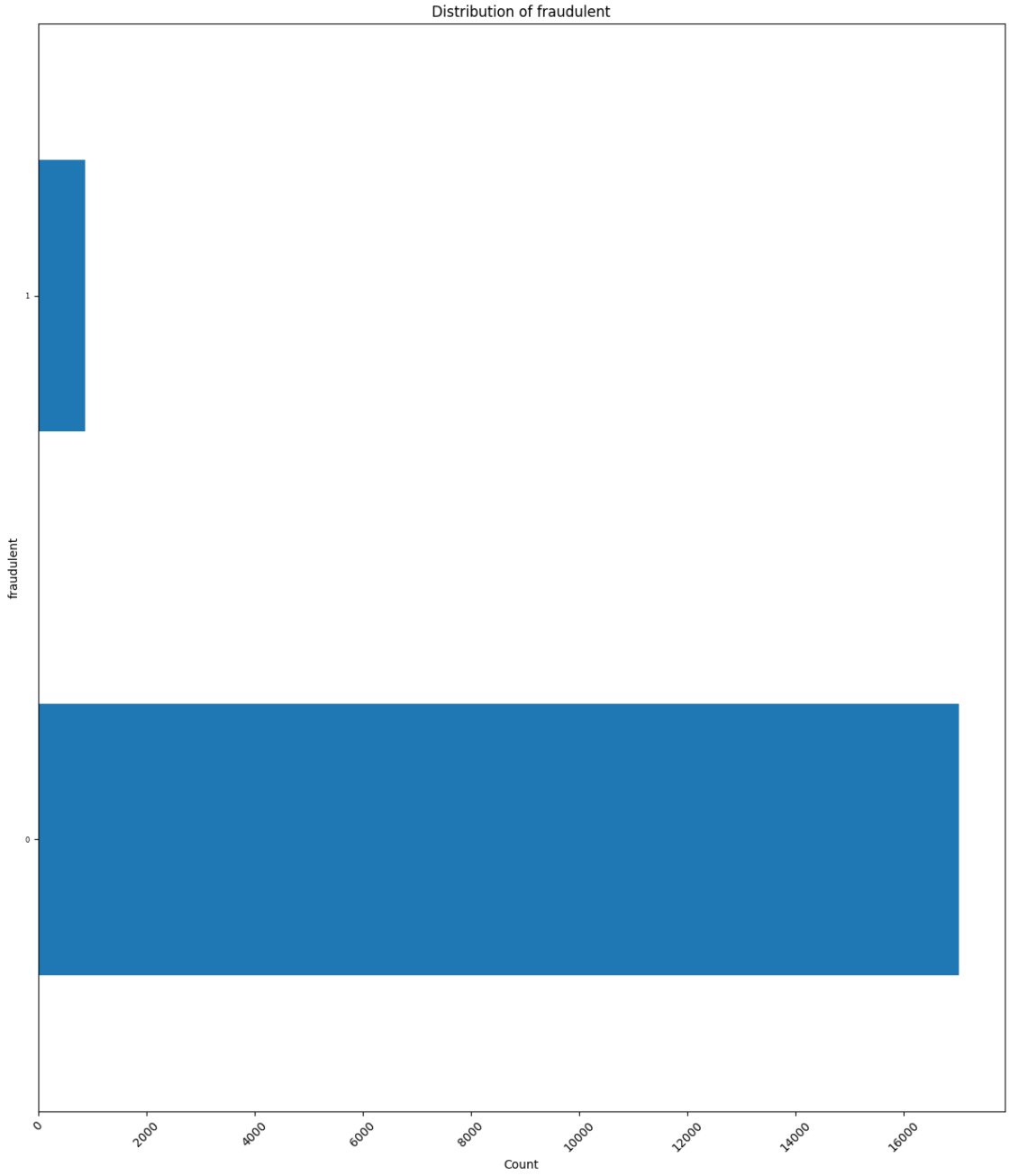

Distribution of industry
industry
Count
Ranching
Alternative Dispute Resolution
Shipbuilding
Museums and Institutions
Sporting Goods
Wine and Spirits
Nanotechnology
Military
Textiles
Libraries
Plastics
Performing Arts
Mining & Metals
Public Policy
Maritime
Furniture
Wireless
Investment Banking
Fishery
Commercial Real Estate
Luxury Goods & Jewelry
Philanthropy
Package/Freight Delivery
Import and Export
Capital Markets
Animation
Packaging and Containers
Religious Institutions
Investment Management
Primary/Secondary Education
Motion Pictures and Film
Public Safety
Industrial Automation
Photography
Executive Office
Veterinary
International Trade and Development
Renewables & Environment
Civil Engineering
Individual & Family Services
Defense & Space
Program Development
Music
Architecture & Planning
Translation and Localization
Law Enforcement
Semiconductors
Government Relations
Higher Education
Machinery
Wholesale
Professional Training & Coaching
Fund-Raising
Business Supplies and Equipment
Law Practice
Medical Devices
Writing and Editing
Outsourcing/Offshoring
Chemicals
Government Administration
Mental Health Care
Sports
Farming
Aviation & Aerospace
Information Services
Research
Venture Capital & Private Equity
Security and Investigations
Printing
Graphic Design
Utilities
Computer Hardware
Mechanical or Industrial Engineering
Biotechnology
Publishing
Pharmaceuticals
Gambling & Casinos
Computer Networking
Food Production
Media Production
Computer & Network Security
Environmental Services
Events Services
Broadcast Media
Warehousing
Restaurants
Transportation/Trucking/Railroad
Market Research
Civic & Social Organization
Public Relations and Communications
Medical Practice
Consumer Electronics
Airlines/Aviation
Consumer Goods
Cosmetics
Food & Beverages
Electrical/Electronic Manufacturing
Entertainment
Nonprofit Organization Management
Leisure, Travel & Tourism
Building Materials
Banking
Computer Games
Hospitality
Facilities Services
Apparel & Fashion
Legal Services
Online Media
Human Resources
Logistics and Supply Chain
Automotive
Insurance
Staffing and Recruiting
Health, Wellness and Fitness
Design
Management Consulting
E-Learning
Construction
Accounting
Real Estate
Retail
Oil & Energy
Telecommunications
Consumer Services
Hospital & Health Care
Financial Services
Education Management
Marketing and Advertising
Internet
Computer Software
Information Technology and Services
nan
0
1000
2000
3000
4000
5000


Distribution of function
Science
Purchasing
Distribution
Financial Analyst
Supply Chain
Training
Strategy/Planning
Legal
Research
General Business
Manufacturing
Public Relations
Data Analyst
Business Analyst
Advertising
Quality Assurance
Product Management
Production
Art/Creative
Writing/Editing
Consulting
Finance
Project Management
Human Resources
Accounting/Auditing
Business Development
Management
Education
Other
Health Care Provider
Design
Administrative
Marketing
Customer Service
Engineering
Sales
Information Technology
nan
function
0
1000
2000
3000
4000
5000
6000
Count


Distribution of telecommuting
telecommuting
1
0
0
2000
4000
6000
8000
10000
12000
14000
16000
Count


Distribution of has_company_logo
has_company_logo
0
1
0
2000
4000
6000
8000
10000
12000
14000
Count


Distribution of has_questions
has_questions
1
0
0
2000
4000
6000
8000
Count



Results: After visualizing our data, we understand that our dataset has class imbalance because we have many more real jobs than fraudulent jobs. We will have to keep this in mind as we build our data science model.

Target Variable: fraudulent

Imbalanced classes:

0 (Real jobs) dominate the dataset.

1 (Fake jobs) are a minority.

We'll need to address this imbalance during modeling (e.g., SMOTE, stratified sampling).

Purpose: This function is designed to visualize the correlation relationships between selected numerical columns in the dataset. It computes the pairwise correlation matrix and plots it as a heatmap, where the strength and direction of the relationships are represented by color intensity. This helps in identifying strong positive or negative correlations between variables, which can inform feature selection, detect multicollinearity, or guide further analysis.

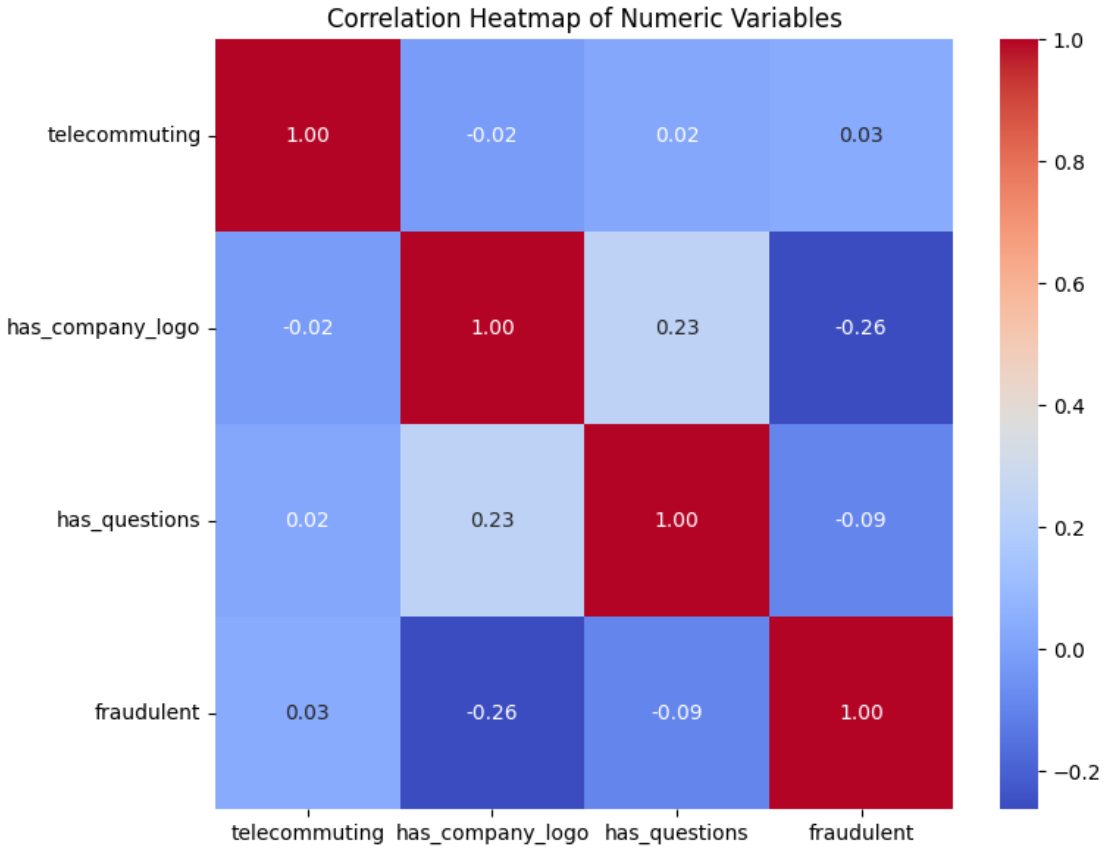
Correlation is a measure that shows the relationship between two variables. It ranges from -1 to +1:

Positive (+1) –> Strong positive correlation - as one goes up, the other also goes up.

Negative (-1) –> Strong negative correlation - as one goes up, the other goes down.

None (0) –> No correlation - they don't move together in any predictable way.

[10]:
```
import seaborn as sns

def plot_correlation_heatmap(df: pd.DataFrame, numeric_cols: list):
    """
    Plot a heatmap showing correlation between numerical columns.

    Args:
        Pandas Dataframe (pd.DataFrame): The dataset.
        numeric_cols (list): List of numerical column names.
    """
    try:
        corr_matrix = df[numeric_cols].corr()
        plt.figure(figsize=(8, 6))
        sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm',␣
 ↪cbar=True)
        plt.title('Correlation Heatmap of Numeric Variables')
        plt.tight_layout()
        plt.show()
    except Exception as e:
        logging.error(f"Error plotting correlation heatmap: {e}")
        raise

# Define numeric columns
numeric_columns = ['telecommuting', 'has_company_logo', 'has_questions',␣
 ↪'fraudulent']

# Plot correlation heatmap
plot_correlation_heatmap(df, numeric_columns)
```



Results:

1. Field: has_company_logo
   - Correlation with fraudulent: negative
   - Meaning: If a job has a company logo (has_company_logo = 1), it's less likely to be fake
   - So: Fake jobs often don't have logos → makes sense.
2. Field: has_questions
   - Correlation with fraudulent: positive
   - Meaning: If has_questions = 1 (i.e., the job does have screening questions), it's more likely to be fake
   - But this seems counterintuitive? You'd think real jobs use questions more?
3. Explanation?

The opposite is more likely true in most real-world cases:

Real jobs do include screening questions. Fake jobs tend to skip them, because they want to look easy/appealing.

So, if has_questions has a positive correlation with fraudulent, it might be due to:

`3.1 - Noise or imbalance in the data.`

`3.2 - Possible feature encoding error.`

`3.3 - Or the correlation is very weak and not meaningful in practice.`

Conclusion:

A positive correlation with fraudulent means that as the other variable increases (e.g., has_questions = 1), the likelihood of being fake (fraudulent = 1) also increases.

But correlation only shows association, not causation.

1.2.2 Step 2: Structuring

Structuring data transforms features to uniform formats, units, and scales.

Purpose: Format text fields in a uniform format

[11]:
```
import re

def format_text(text: str) -> str:
  """
  Clean the input text converting to lowercase and removing whitespaces. We␣
 ↪will leave non-ascii characters as it could be an indicator of a fraudulent␣
 ↪job.
  Args:
      text (str): The input text to be cleaned.

  Returns:
      str: The cleaned text
  """
  if not isinstance(text, str):
      raise ValueError("Input must be a string.")
  text = text.lower()
  text = re.sub(r'[^\w\s]', '', text)
  return text

  df['description'] = df['description'].apply(format_text)
  df['requirements'] = df['requirements'].apply(format_text)
  df['title'] = df['title'].apply(format_text)
  df['company_profile'] = df['company_profile'].apply(format_text)
```

Results: Now the text fields we care about for our model are formatted consistently, which will be especially useful when applying TF-IDF. Due to the nature of our dataset we did not have to

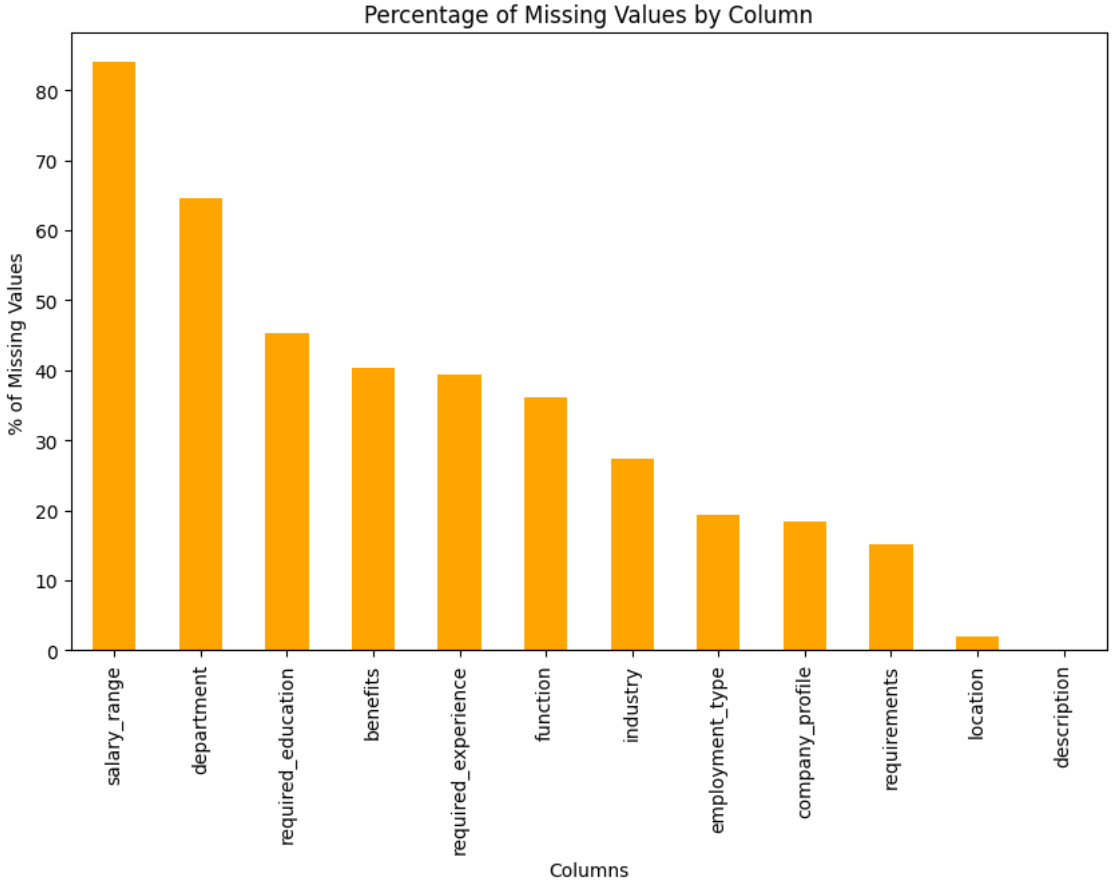
do too much structuring to it.

1.2.3 Step 3: Cleaning

Cleaning data removes or replaces missing and outlier data.

Purpose: We saw earlier in a table that we have certain coumns with missing data ordered by percentage, now we either want to drop/filter those columns out of our dataset also because they have poor real-world usability for our Data Science Model.

[12]:
```
# Count and display percentage of missing values:
missing_percent = (df.isnull().sum() / len(df)) * 100
missing_percent = missing_percent[missing_percent > 0].
 ↪sort_values(ascending=False)

plt.figure(figsize=(10, 6))
missing_percent.plot(kind='bar', color='orange')
plt.title('Percentage of Missing Values by Column')
plt.ylabel('% of Missing Values')
plt.xlabel('Columns')
plt.show()
```



Results: We've visualized some of the fields we intend to filter in the coming steps.

Purpose: We want to filter the dataframe to only have our feature and target columns since some of the other fields have too mnay nulls or are not usable for our goals and have poor real-world application.

Features:

- Text: title, description, requirements, company_profile
- Structured: employment_type (Categorical), telecommuting (Boolean)

Target: - fraudulent

[13]:
```
def filter_dataframe(df: pd.DataFrame, required_columns: List[str]) -> pd.
↪DataFrame:
    """
    Filters the input DataFrame to retain only the specified columns, in the␣
↪given order.

    Parameters
    ----------
    df: pd.DataFrame - The input DataFrame to process.

    required_columns: List[str]
        A list of columns that must be present in the DataFrame. Only these␣
↪columns will be retained, in this order.

    Returns
    -------
    pd.DataFrame: A new DataFrame containing only the specified columns, in the␣
↪specified order.

    Raises
    ------
    ValueError: If any of the required columns are missing in the input␣
↪DataFrame.
    """
    missing = set(required_columns) - set(df.columns)
    if missing:
        raise ValueError(f"Missing required column(s): {', '.join(missing)}")

    return df[required_columns]

required_columns = [
    "job_id",
```

```
    "title",
    "description",
    "requirements",
    "company_profile",
    "employment_type",
    "telecommuting",
    "fraudulent"
]

filtered_df = filter_dataframe(df, required_columns)
```

[14]:
```
filtered_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17880 entries, 0 to 17879
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   job_id           17880 non-null  int64
 1   title            17880 non-null  object
 2   description      17879 non-null  object
 3   requirements     15184 non-null  object
 4   company_profile  14572 non-null  object
 5   employment_type  14409 non-null  object
 6   telecommuting    17880 non-null  int64
 7   fraudulent       17880 non-null  int64
dtypes: int64(3), object(5)
memory usage: 1.1+ MB
```

Results: Now we have only our feature and target columns along with the "job_id" field to easily identify a given row.

Purpose: This function is responsible for cleaning the dataset by addressing missing or incomplete data in key text and categorical fields. Specifically, it performs two main operations:

1. Row Removal: It drops rows where all of the major text fields (title, description, requirements, company_profile) are missing, as such records provide no useful information.
2. Value Imputation: It fills missing or blank values in the employment_type column with the placeholder value 'Unknown', ensuring consistent treatment of missing employment data.

This cleaning step improves data quality by removing irrelevant records and standardizing missing values, making the dataset more reliable for analysis and modeling.

[15]:
```
def clean_dataset(df: pd.DataFrame) -> pd.DataFrame:
    """
    Cleans the dataset by applying the following transformations:
    1. Drops rows where all four specified text columns are missing (NaN).
```

```
    2. Fills missing and empty string values in the 'employment_type' column␣
↪with 'Unknown'.

    Parameters
    ----------
    df: pd.DataFrame - The input DataFrame to clean.

    Returns
    -------
    pd.DataFrame - A cleaned DataFrame with the specified transformations␣
↪applied.
    """
    text_columns = ['title', 'description', 'requirements', 'company_profile']
    rows_before = len(df)

    # Drop rows where all text columns are missing:
    df_cleaned = df.dropna(subset=text_columns, how='all')
    rows_after = len(df_cleaned)
    dropped = rows_before - rows_after

    # Fill missing or blank 'employment_type' with 'Unknown':
    df_cleaned['employment_type'] = df_cleaned['employment_type'].apply(lambda␣
↪x: 'Unknown' if pd.isna(x) or str(x).strip() == '' else x)

    print(f"[INFO] Dropped {dropped} rows with all text fields missing.")
    print(f"[INFO] Filled 'employment_type' blanks and NaNs with 'Unknown'.")

    return df_cleaned

# Supply the filtered dataframe from the previous step:
cleaned_df = clean_dataset(filtered_df)
```

```
[INFO] Dropped 0 rows with all text fields missing.
[INFO] Filled 'employment_type' blanks and NaNs with 'Unknown'.
```

Results: We now have our cleaned dataframe and can continue with the next steps.

1.2.4 Step 4: Enriching

Enriching data derives new features from existing features and appends new data from external sources.

Purpose We will add metadata around the description feature as it is our main text feature. Specifically, we are looking to add word count, readability using the flesch reading ease score, and complexity score using the text standard in textstat as these can all be an indicator of fraud.

[16]:
```
import textstat

cleaned_df[f'description_wordcount'] = cleaned_df['description'].astype(str).
 ↪apply(lambda x: len(x.split()))
cleaned_df[f'description_readability'] = cleaned_df['description'].astype(str).
 ↪apply(textstat.flesch_reading_ease).fillna(0).astype(float)
cleaned_df['description_complexity_score'] = cleaned_df['description'].
 ↪astype(str).apply(lambda x: textstat.text_standard(x, float_output=True)).
 ↪fillna(0).astype(float)
cleaned_df[f'company_profile_wordcount'] = cleaned_df['company_profile'].
 ↪astype(str).apply(lambda x: len(x.split()))
cleaned_df[f'company_profile_readability'] = cleaned_df['company_profile'].
 ↪astype(str).apply(textstat.flesch_reading_ease).fillna(0).astype(float)
cleaned_df['company_profile_complexity_score'] = cleaned_df['company_profile'].
 ↪astype(str).apply(lambda x: textstat.text_standard(x, float_output=True)).
 ↪fillna(0).astype(float)

cleaned_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17880 entries, 0 to 17879
Data columns (total 14 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   job_id                           17880 non-null  int64
 1   title                            17880 non-null  object
 2   description                      17879 non-null  object
 3   requirements                     15184 non-null  object
 4   company_profile                  14572 non-null  object
 5   employment_type                  17880 non-null  object
 6   telecommuting                    17880 non-null  int64
 7   fraudulent                       17880 non-null  int64
 8   description_wordcount            17880 non-null  int64
 9   description_readability          17880 non-null  float64
 10  description_complexity_score     17880 non-null  float64
 11  company_profile_wordcount        17880 non-null  int64
 12  company_profile_readability      17880 non-null  float64
 13  company_profile_complexity_score  17880 non-null  float64
dtypes: float64(4), int64(5), object(5)
memory usage: 1.9+ MB
```

Result We now have 13 features in our dataset, including key metadata about description and company_profile features.

1.2.5 Step 5: Validating

Validating data verifies that the dataset is internally consistent and accurate.

Purpose First, we will verify that our boolean-type features have only 0 and 1 for values.

[17]:
```
cleaned_df[['fraudulent', 'telecommuting']].value_counts()
```

[17]:
```
fraudulent  telecommuting
0           0                16311
1           0                  802
0           1                  703
1           1                   64
Name: count, dtype: int64
```

Results We can see that only valid values of 0 and 1 exist in fraudulent and telecommuting features.

1.2.6 Step 6: Publishing

Now that we have completed our initial data wrangling, we can save our cleaned dataset in a git repository, upload it to Kaggle, or submit our final findings to the original dataset. This will allow others to benefit and learn from the modificaiton we have done to the original dataset.

GitHub Repo for Project Fake Finder: https://github.com/jdeleon3/Projects.RealFakeJobPredictor

[18]:
```
import os
if os.path.exists('./data'):
    cleaned_df.to_csv('./data/cleaned_job_postings.csv', index=False)
    print("Cleaned dataset saved as 'cleaned_job_postings.csv'.")
```

```
Cleaned dataset saved as 'cleaned_job_postings.csv'.
```

1.3 Data Exploration

1.3.1 Step 1: Understand the data

Find the size of the dataset (number of rows and columns), identify and categorize the features (categorical, numerical).

1.3.2 Purpose

Now that we have a cleaned dataset, we should again evaluate what our dataset looks like, starting with it's shape and the type of features included.

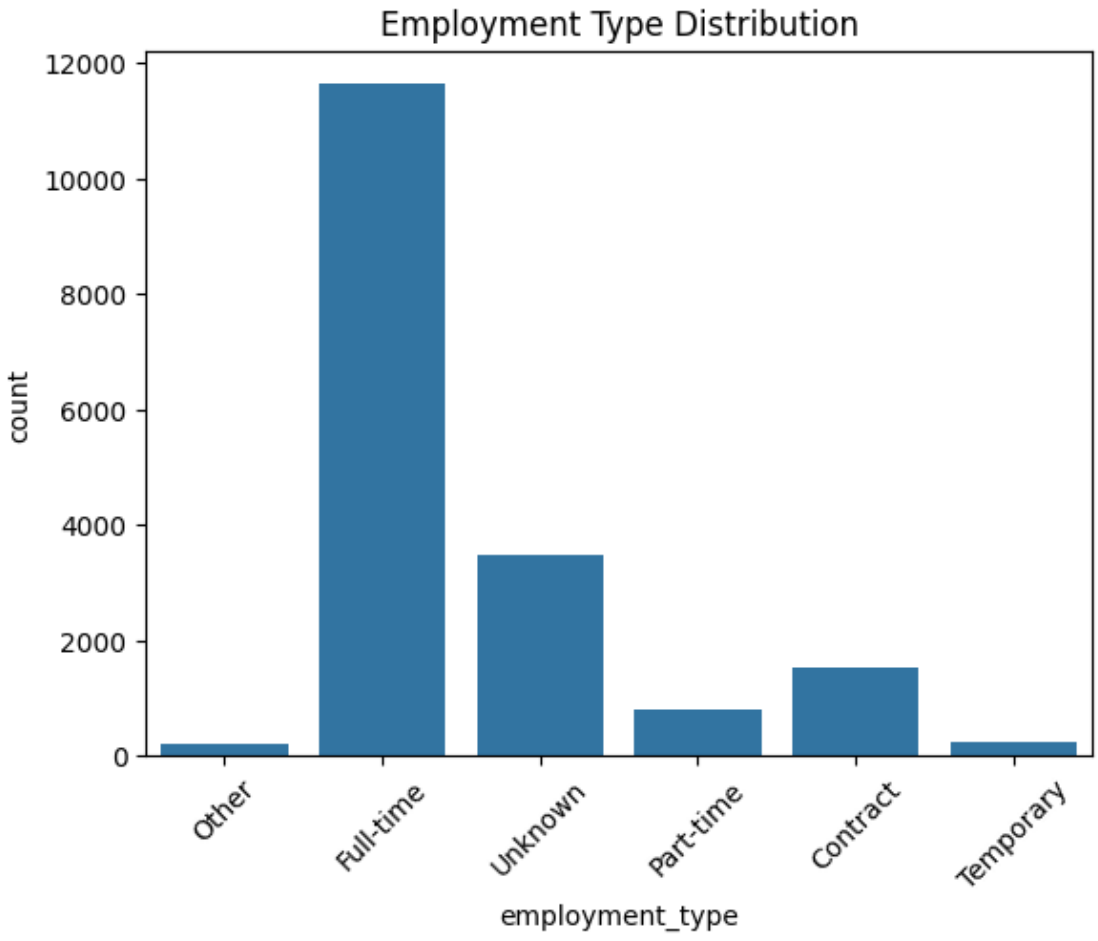
[19]:
```
print(f"{cleaned_df.shape[0]} rows and {cleaned_df.shape[1]} columns after␣
 ↪cleaning.")

numeric_columns = cleaned_df.select_dtypes(include=[np.number]).columns.tolist()
text_columns = cleaned_df.select_dtypes(include=[object]).columns.tolist()

print(f"{len(numeric_columns)} Numeric columns: {numeric_columns}")
print(f"{len(text_columns)} Text columns: {text_columns}")
```

```
17880 rows and 14 columns after cleaning.
9 Numeric columns: ['job_id', 'telecommuting', 'fraudulent',
'description_wordcount', 'description_readability',
'description_complexity_score', 'company_profile_wordcount',
'company_profile_readability', 'company_profile_complexity_score']
5 Text columns: ['title', 'description', 'requirements', 'company_profile',
'employment_type']
```

Results After cleaning our dataset, we have 17880 row and 13 features (8 numeric and 5 text).

Purpose Part of understanding our data is looking for features that have an imbalance in values. The employment_type feature could be one of those features, as most job postings tend to be Full-time.

[20]:
```
sns.countplot(cleaned_df, x='employment_type')
plt.title('Employment Type Distribution')
plt.xticks(rotation=45)
plt.show()
```



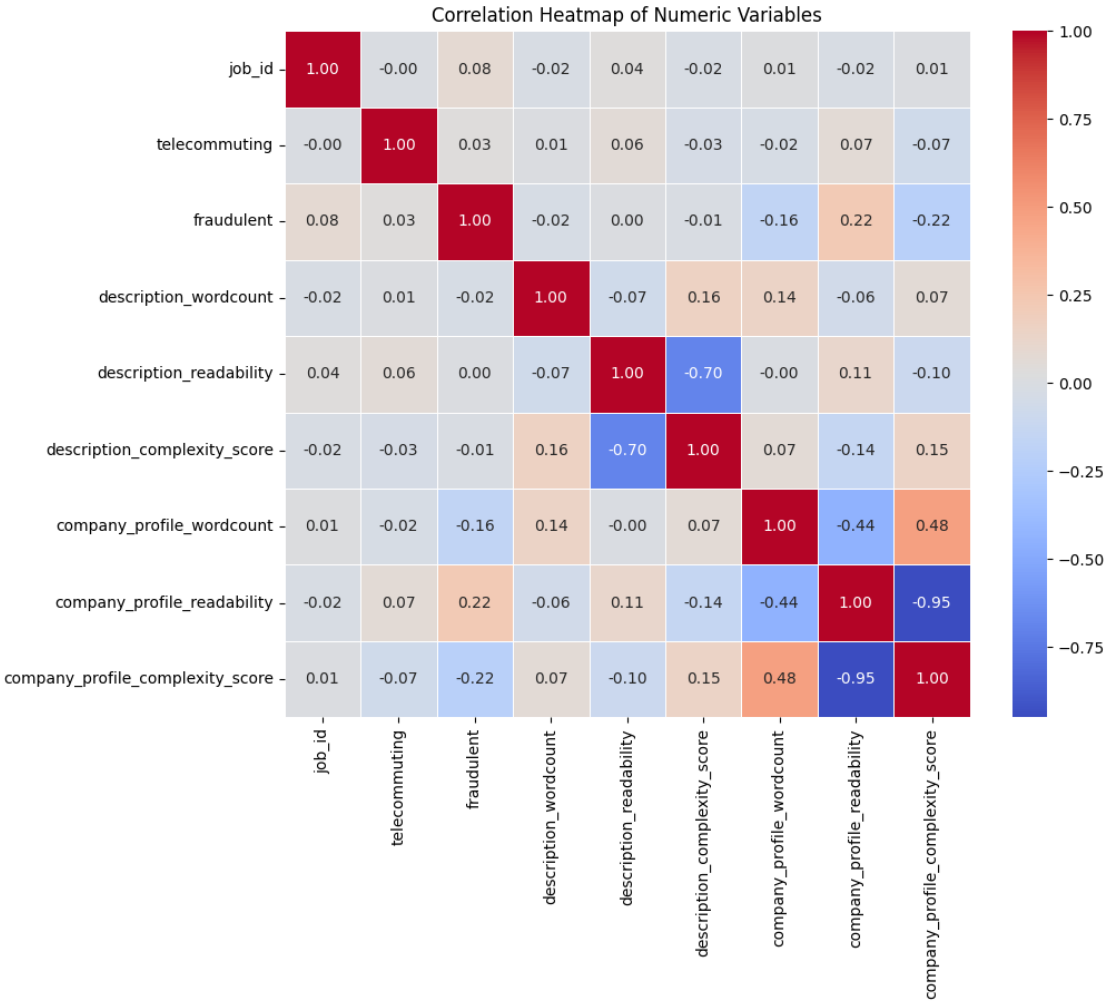
Results As expected, Full-time does have considerably more than all other types. We might consider combining some types if we find employment_type being overfit by the model, but for now, we will leave it as is.

1.3.3 Step 2: Identify relationships between features

Find the direction (positive, negative) and strength (strong, moderate, weak) of correlation between the features.

Purpose Next, we will verify that we do not have any highly coorelated features that can be combined. We will consider correlation values over 0.7 as candidates. To do this, we will look at a heatmap of the dataset features.

[21]:
```
corr_matrix = cleaned_df.select_dtypes(include=[np.number]).corr()
plt.figure(figsize=(10, 8))

sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True,␣
↪square=True, linewidths=0.5)

plt.title('Correlation Heatmap of Numeric Variables')
plt.show()

cleaned_df.drop(columns=['descriptioin_complexity_score',␣
↪'company_profile_complexity_score'], inplace=True, errors='ignore')
```



Results We found that the readability features created during our Enriching step is negatively correlated with the complexity score feature. Since the correlation is above 0.70, we dropped company_profile_complexity_score and descriptioin_complexity_score features.

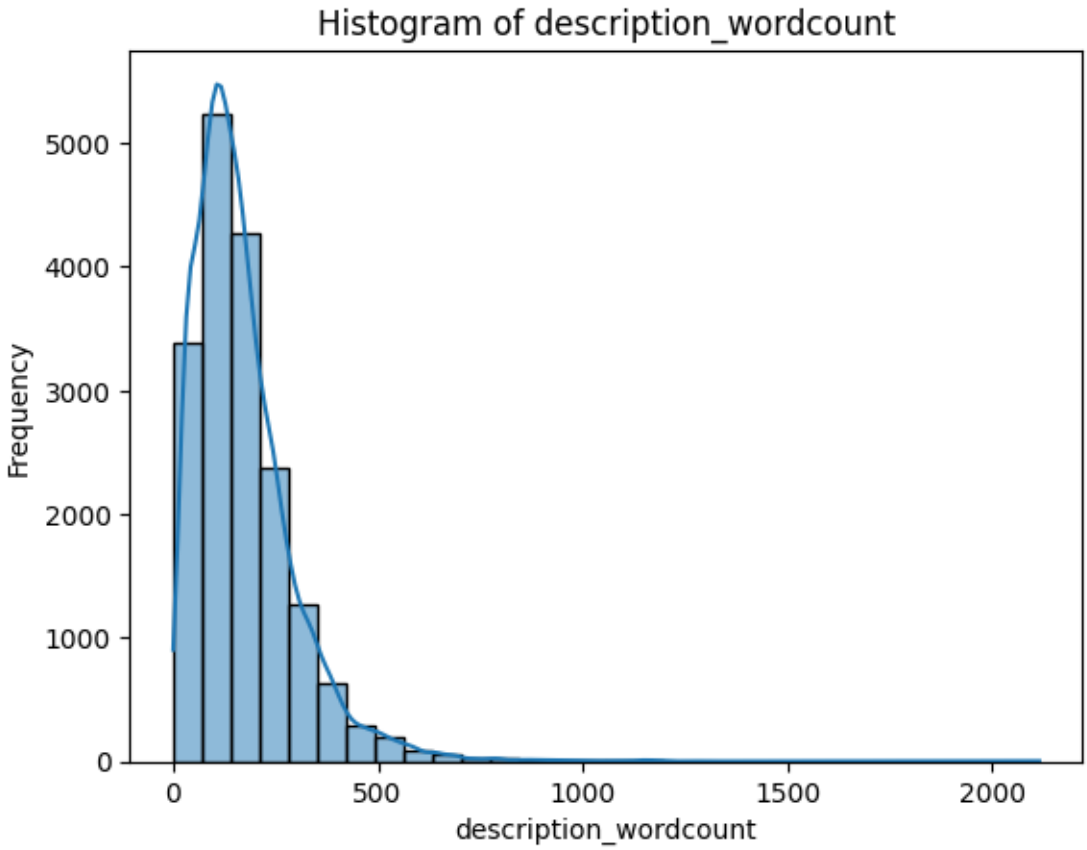
1.3.4 Step 3: Describe the Data

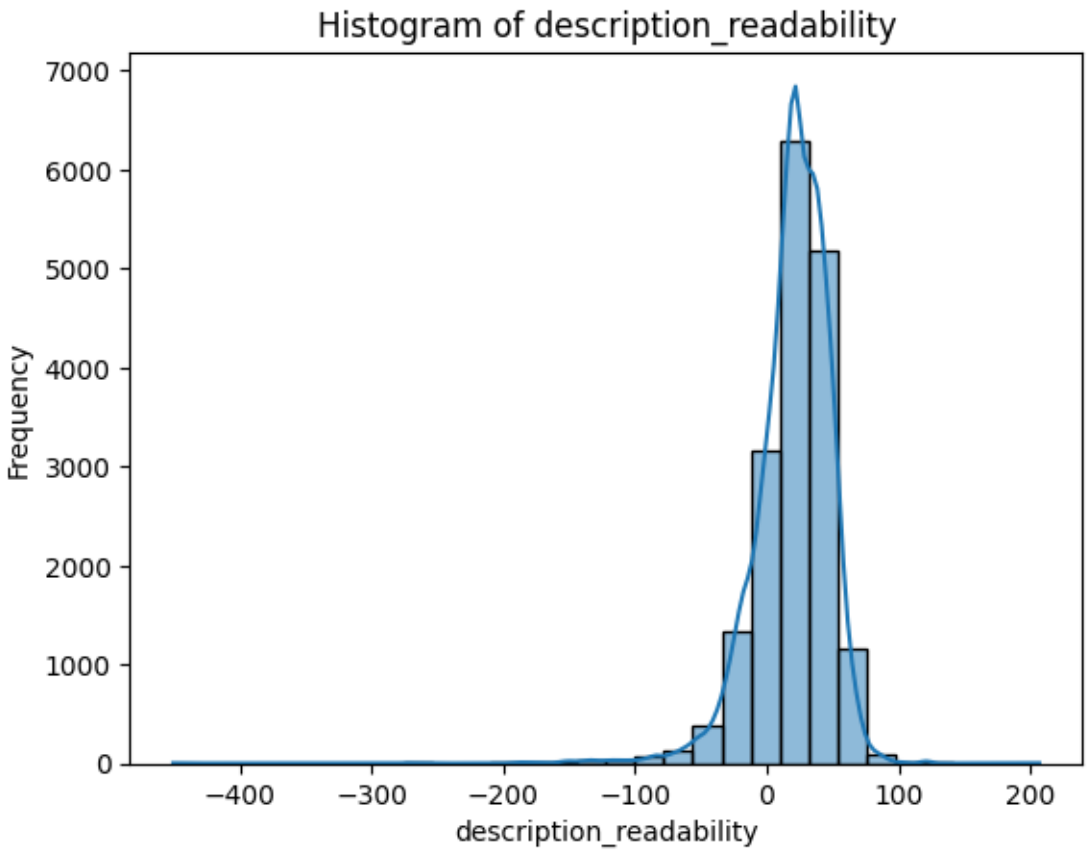
Determine the shape of the distribution (symmetric, skewed)

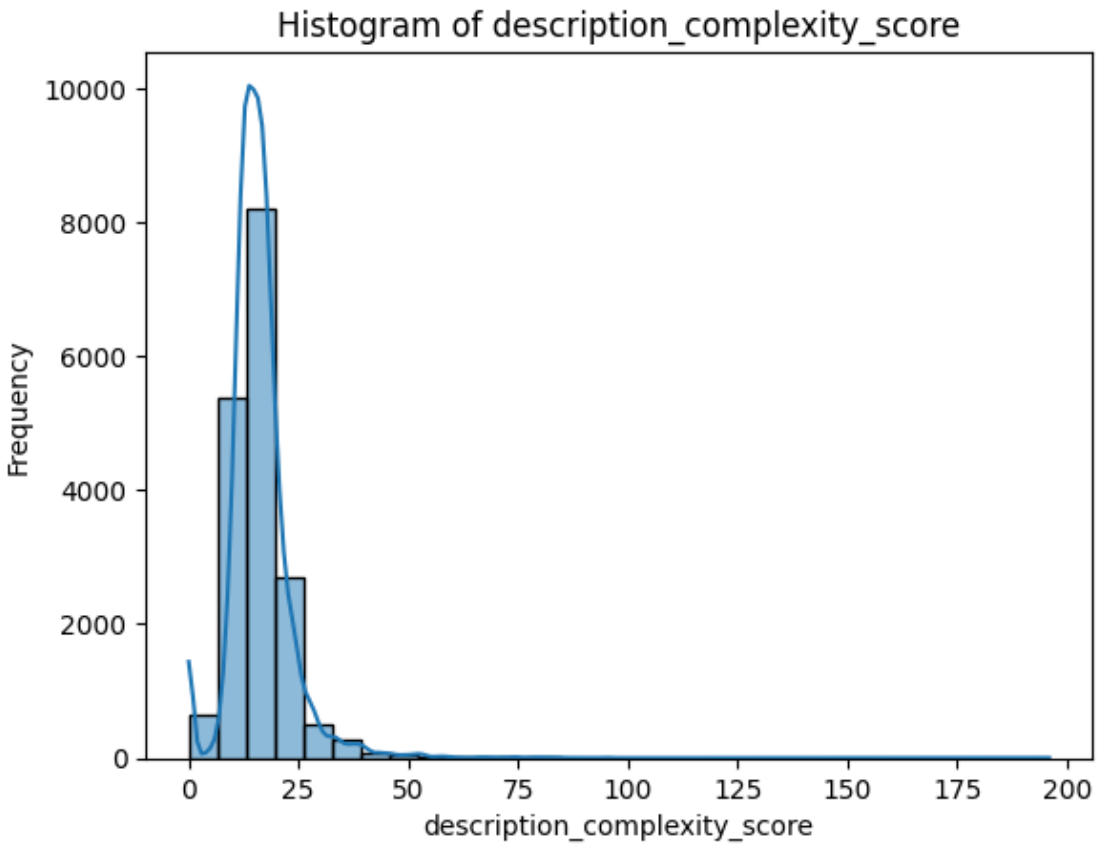
Purpose We will review the histogram plots of our engineered features to check their distribution.

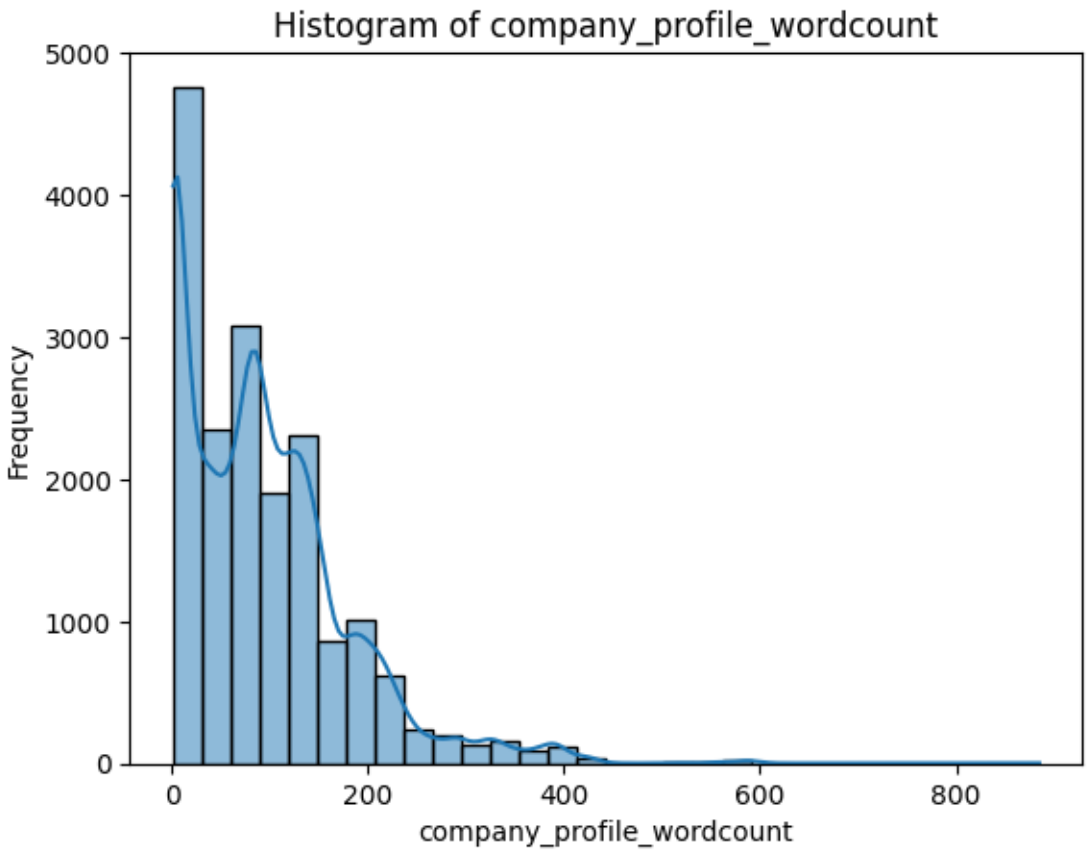
[22]:
```
numeric_columns = ['description_wordcount', 'description_readability',␣
 ↪'description_complexity_score', 'company_profile_wordcount',␣
 ↪'company_profile_readability']

for col in numeric_columns:
    sns.histplot(cleaned_df[col], bins=30, kde=True)
```

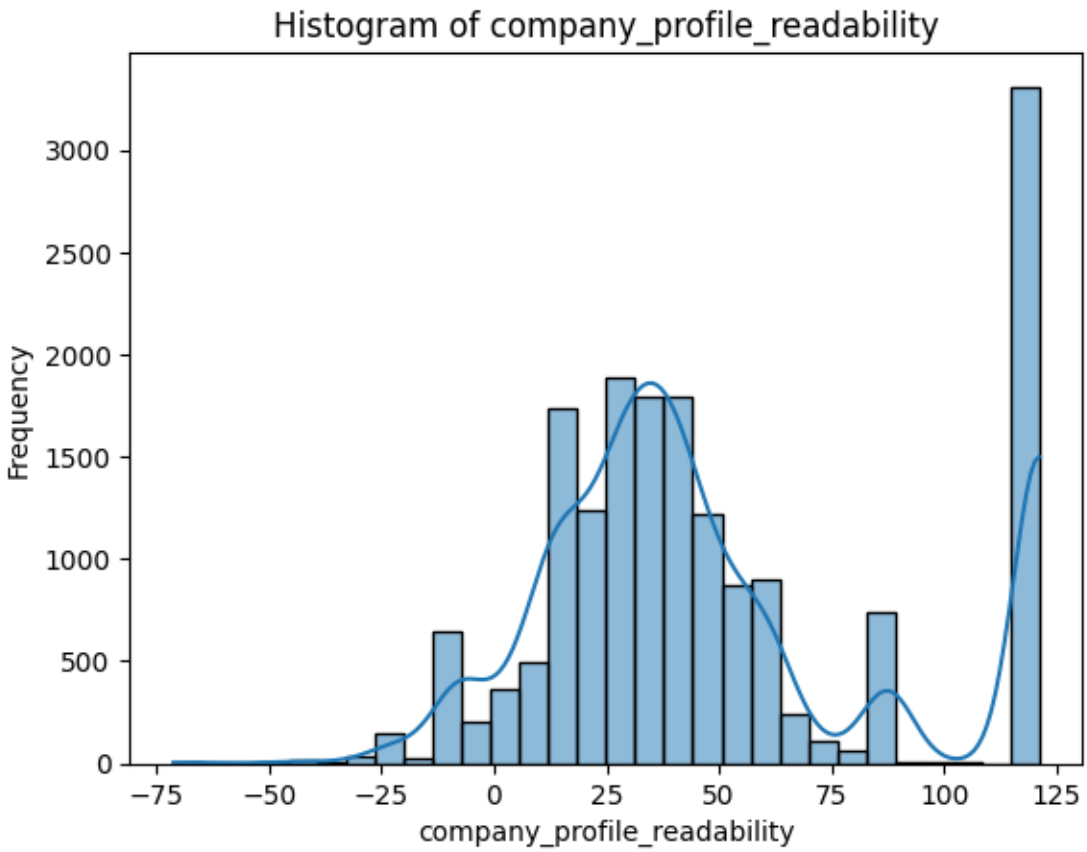
```
    plt.title(f'Histogram of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```




Histogram of description_readability
Frequency
7000
6000
5000
4000
3000
2000
1000
0
−400
−300
−200
−100
0
100
200
description_readability


Histogram of description_complexity_score
Frequency
description_complexity_score
0
2000
4000
6000
8000
10000
0
25
50
75
100
125
150
175
200


Histogram of company_profile_wordcount
Frequency
company_profile_wordcount
0
1000
2000
3000
4000
5000
0
200
400
600
800



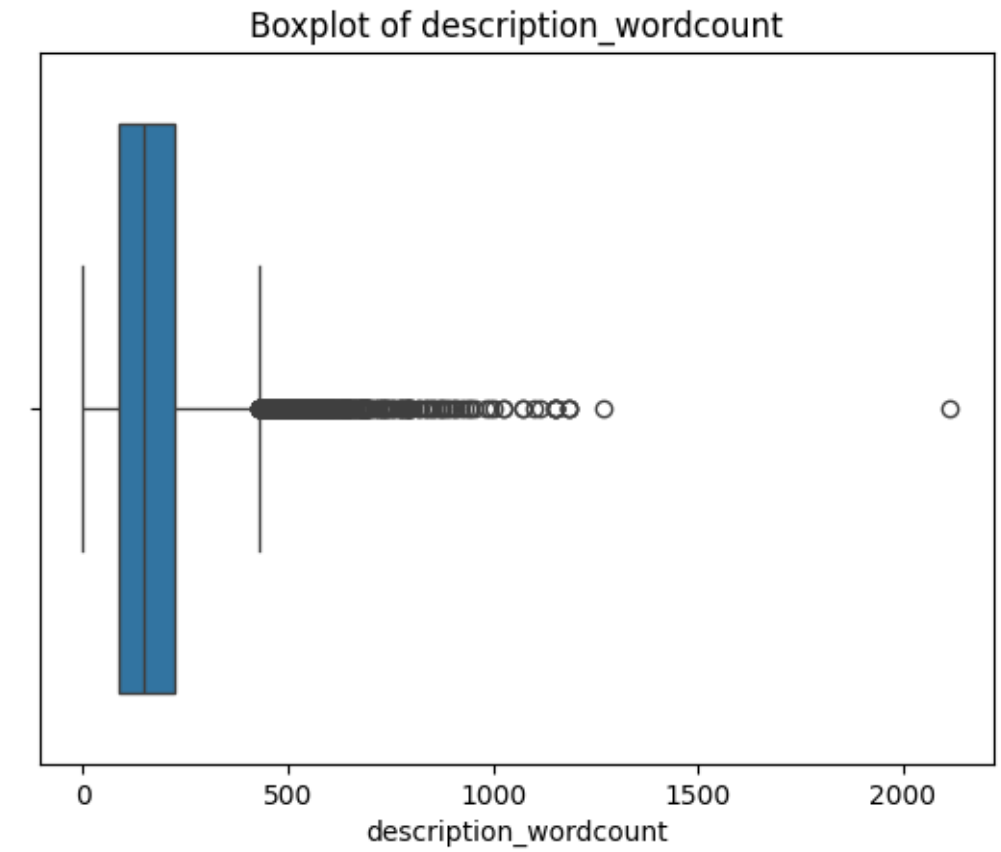
Results Each of the features are skewed. Description_wordcount, company_profile_wordcount, and description_complexity_score are all skewed right with a long tail while company_profile_readability and description_readability are skewed left with a long tail. This warrants further analysis to consider outliers.

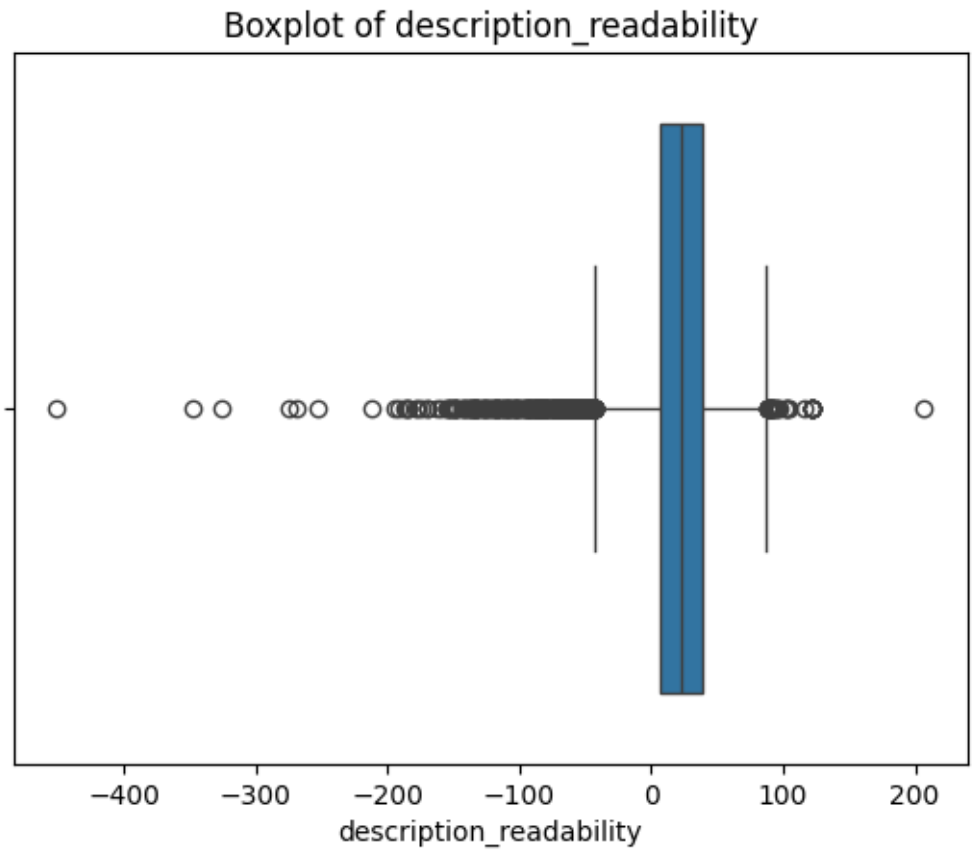
1.3.5 Step 4: Detect outliers and missing data

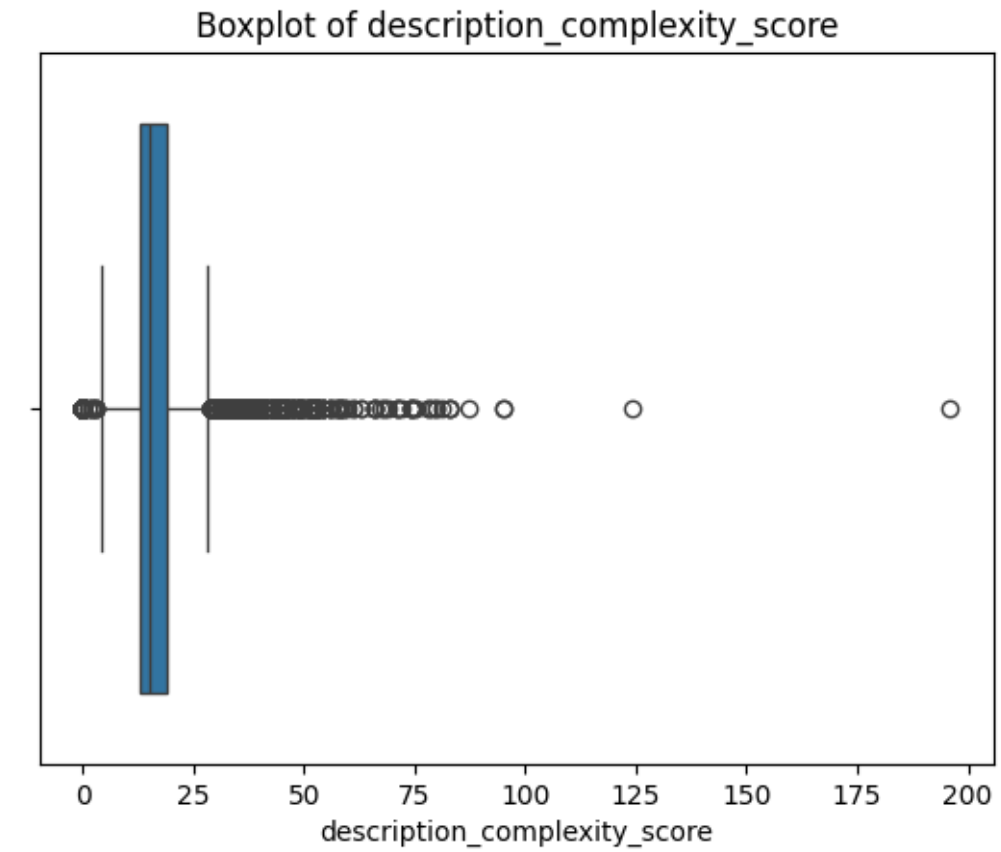
Find values that are much higher or lower than the rest of the data or values that strongly affect the shape of the data.

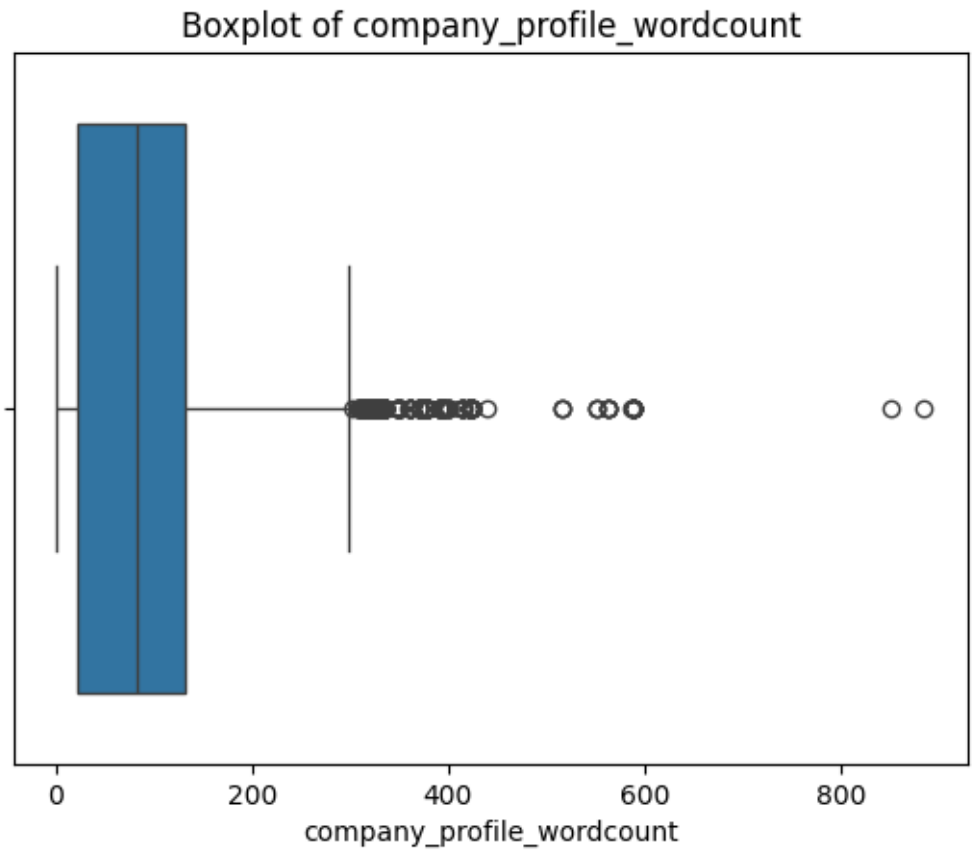
Purpose With our engineered features being skewed, we will look at the boxplot of each to identify if there are any outliers that should be addressed.

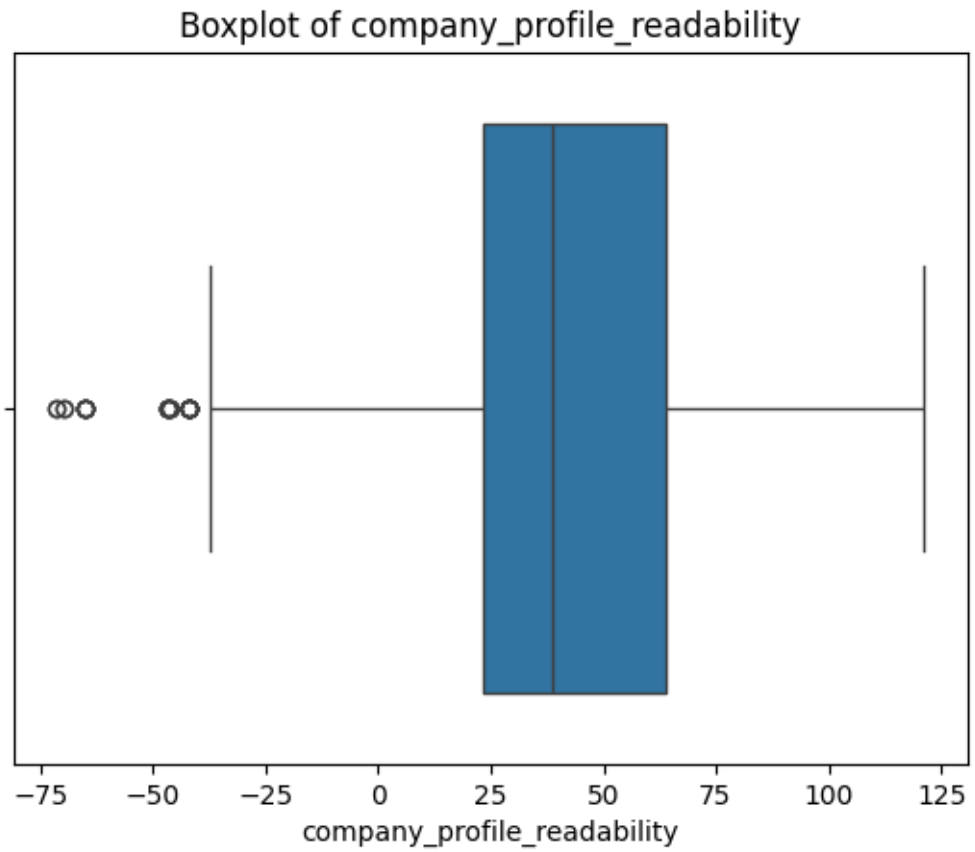
[23]:
```
for col in numeric_columns:
    sns.boxplot(data=cleaned_df, x=col)
    plt.title(f'Boxplot of {col}')
    plt.xlabel(col)
    plt.show()
```


Boxplot of description_wordcount
0
500
1000
1500
2000
description_wordcount


Boxplot of description_readability
−400
−300
−200
−100
0
100
200
description_readability


Boxplot of description_complexity_score
0
25
50
75
100
125
150
175
200
description_complexity_score


Boxplot of company_profile_wordcount
0
200
400
600
800
company_profile_wordcount

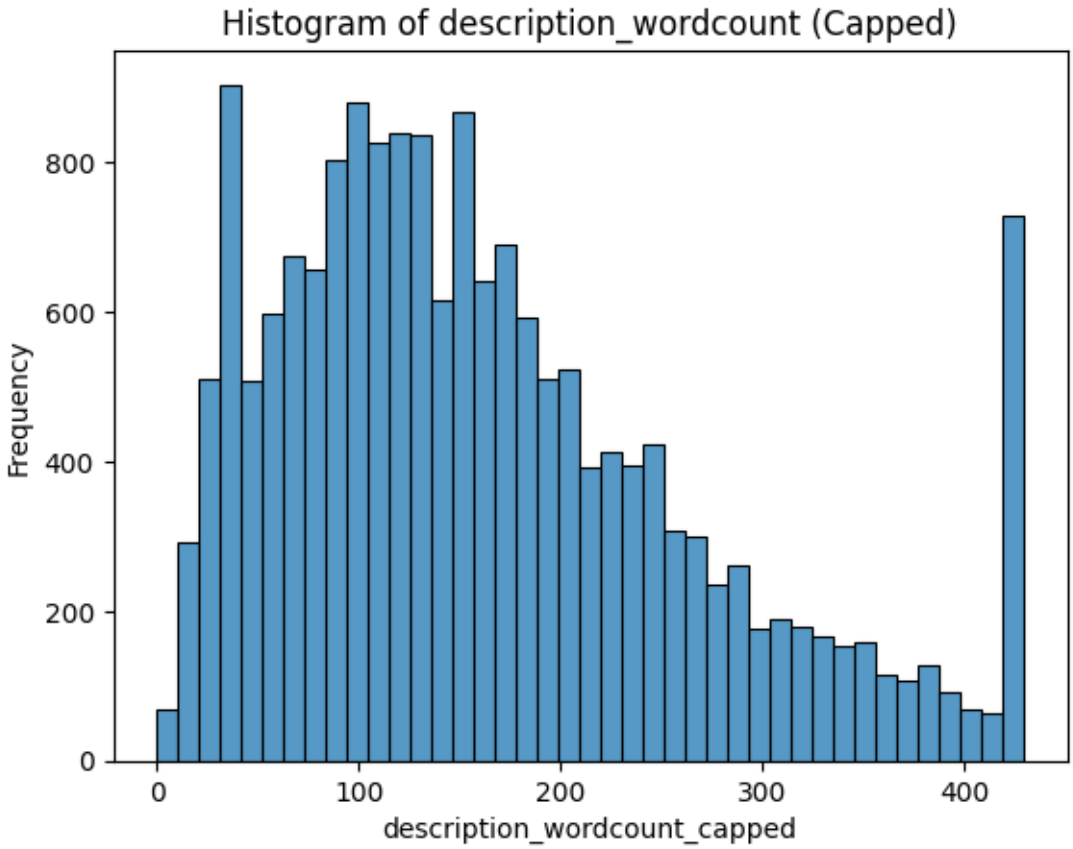


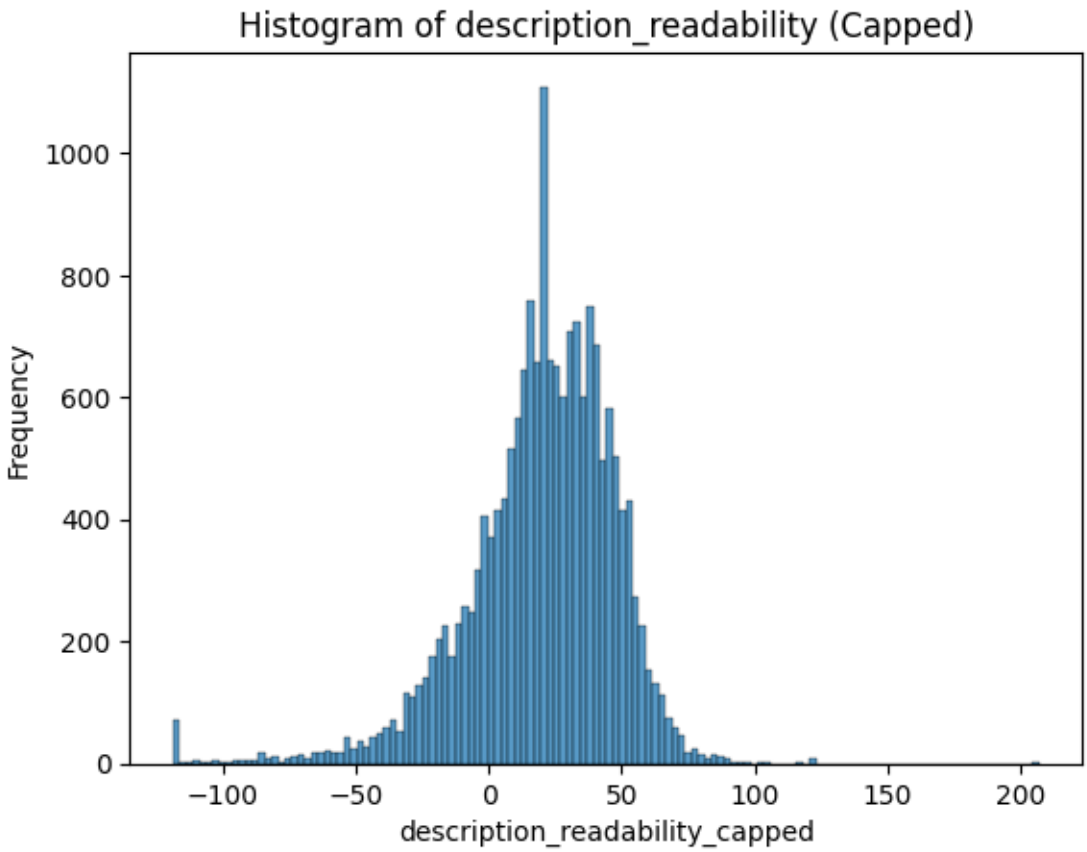
Results We can see that each of our engineered features have quite a few potential outliers. As outliers could be a signal for fraud, we do not want to completely remove our outliers.

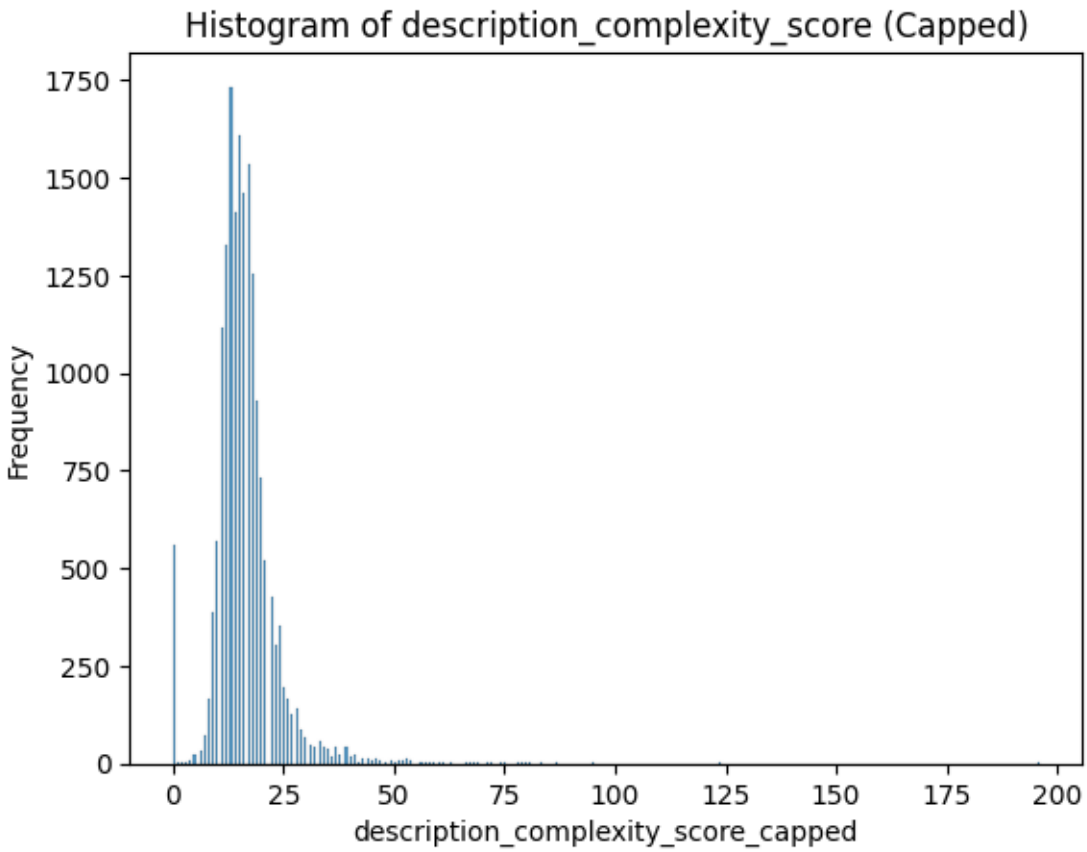
Purpose Since we have a number of outliers that could affect the results of some models, such as regression models, we will add additional features that cap values at the q1 and q3 values of each feature. We will review the histogram of each of the new capped features to ensure that the overall data is still consistent with the uncapped.

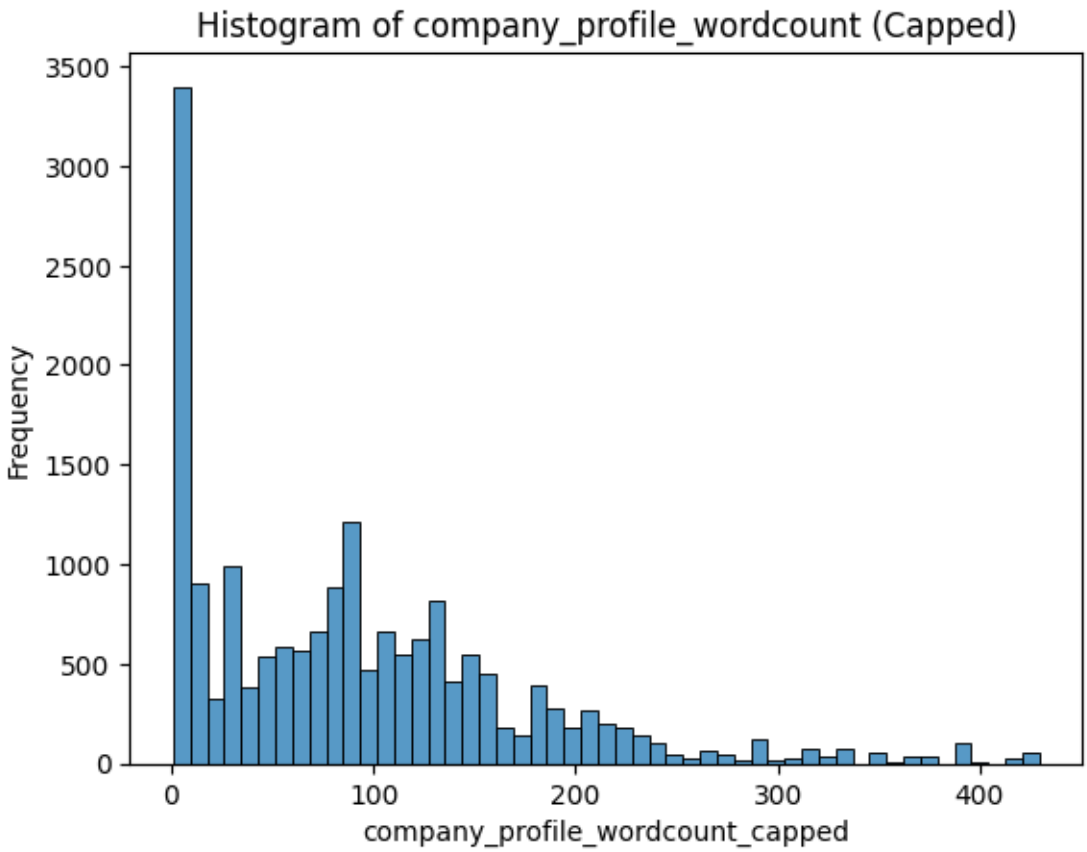
[24]:
```
for col in numeric_columns:
    q1 = cleaned_df['description_wordcount'].quantile(0.25)
    q3 = cleaned_df['description_wordcount'].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    cleaned_df[f'{col}_capped'] = cleaned_df[col].clip(lower=lower_bound,␣
 ↪upper=upper_bound)
    sns.histplot(data=cleaned_df, x=f'{col}_capped')
    plt.title(f'Histogram of {col} (Capped)')
    plt.xlabel(f'{col}_capped')
```

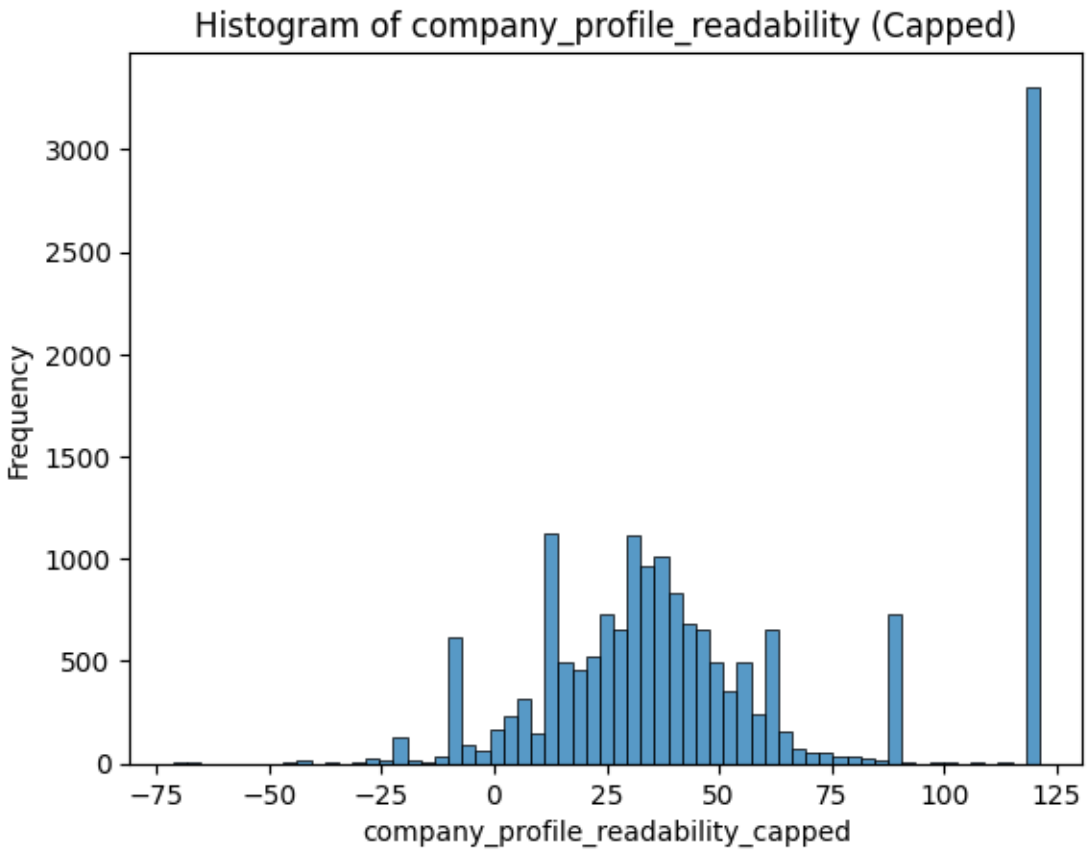
```
    plt.ylabel('Frequency')
    plt.show()
```




Histogram of description_readability (Capped)
Frequency
1000
800
600
400
200
0
−100
−50
0
50
100
150
200
description_readability_capped


Histogram of description_complexity_score (Capped)
Frequency
1750
1500
1250
1000
750
500
250
0
0
25
50
75
100
125
150
175
200
description_complexity_score_capped


Histogram of company_profile_wordcount (Capped)
Frequency
company_profile_wordcount_capped
0
500
1000
1500
2000
2500
3000
3500
0
100
200
300
400



Results As we can see, the histograms are still similarly shapped as the uncapped histograms with values more suitable for regression models. With models that are sensitive to outliers, we can use 'description_wordcount_capped', 'description_readability_capped', 'description_complexity_score_capped', 'company_profile_wordcount_capped', and 'company_profile_readability_capped'. For models less sensitive to outliers, we can use the original data.

1.4 Conclusion

Now that we have done a thorough analysis and exploration of our dataset, we are finally ready to start to apply models. We have deffered applying TF-IDF to our categorical features such as company_profile and description until we get to more advanced models.