```
# 🔧 Step 1: Required libralibrarieslibralibrariesrieslibralibrariesriesries install (Run only once)
!pip install transformers nltk --quiet  # transformers = model, nltk = text processing
```

```
# 🔧 Step 2: Import necessary Python libraries
import pandas as pd                     # CSV file read & dataframe manage
import re                               # Regular expression for text cleaning
from nltk.corpus import stopwords       # English stopword list (like: 'the', 'is', etc)
from nltk.tokenize import word_tokenize  # Word-by-word split for filtering
from nltk import download               # To download NLTK resources
from transformers import pipeline       # HuggingFace summarizer model
import nltk                             # Core nltk library

# ✅ Download only needed NLTK data
nltk.download('punkt')                  # For tokenization
nltk.download('stopwords')              # For removing common English words
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
# 🔧 Step 3: NLTK resources download (Only once)
download('punkt')           # Word & sentence tokenizers
download('stopwords')       # Common English stopwords
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
# 📁 Step 4: Load your dataset
# Check if pandas is imported
try:
    pd
except NameError:
    print("Pandas library not loaded. Please run the import cell first.")
    # You might want to exit or handle this case differently depending on your needs
    # For now, we'll stop execution here to prevent the NameError
    raise

df = pd.read_csv("/content/Last_Year_Project - Main (1).csv")  # Tumi jei file upload koro, tar path eta
```

```
# 🧹 Step 5: Company Overview column theke missing gula remove koro
df = df.dropna(subset=["Company Overview"])  # Jekhane review nai, oigula bad
```

```
# 🧼 Step 6: Text ke lowercase kora & punctuation clean kora
df["Clean_Review"] = df["Company Overview"].apply(lambda x: re.sub(r'[^a-zA-Z0-9\s]', '', str(x).lower()))
# 🔹 Explanation:
# - x.lower(): sob kichu small letter e
# - re.sub(): special character gula (.,!@) remove kora
```

```
# 🚫 Step 7: Stopwords remove kora (optional but helps)

# 1️⃣ Stopwords set banano
stop_words = set(stopwords.words('english'))  # Example: ['is', 'the', 'and', 'a'...]

# ✅ Download only needed NLTK data for tokenization if not already downloaded
try:
    word_tokenize("test")
except LookupError:
    nltk.download('punkt_tab')


# 2️⃣ Cleaned text theke stopwords remove kora
df["Clean_Review"] = df["Clean_Review"].apply(
    lambda x: ' '.join([w for w in word_tokenize(str(x)) if str(x).strip() and w.lower() not in stop_words])
)
```

```
# 🔹 Explanation:
# - str(x): jodi kono value NaN hoy, seta keo string e convert kore
# - str(x).strip(): check if string is not empty or whitespace
# - word_tokenize(): text ke word e vag kore
# - w.lower(): lowercase kore compare kore stopwords er shathe
# - if w.lower() not in stop_words: stopword gulo remove
# - ' '.join(...): cleaned words gulo abar sentence e convert kora
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

```
# ⚡ Faster model than bart-large-cnn (no GPU needed)
summarizer = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secre
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json:        1.80k/?  [00:00<00:00,  92.3kB/s]

pytorch_model.bin: 100%                                      1.22G/1.22G [00:12<00:00, 125MB/s]

model.safetensors: 100%                                      1.22G/1.22G [00:11<00:00, 110MB/s]

tokenizer_config.json: 100%                                    26.0/26.0 [00:00<00:00, 653B/s]

vocab.json:         899k/?  [00:00<00:00,  9.75MB/s]

merges.txt:         456k/?  [00:00<00:00,  8.17MB/s]
Device set to use cpu
```

```
def get_summary_for_position(position, max_reviews=15):
    reviews = df[df["Position"] == position]["Company Overview"].dropna().tolist()[:max_reviews]
    skills = df[df["Position"] == position]["Skills Required"].dropna().unique().tolist()

    if not reviews:
        return "No reviews found for this position.", []

    combined_text = " ".join(reviews)

    # Check if combined_text is too short for summarization
    # A threshold of 50 characters is used as an example
    if len(combined_text) < 50:
         return "Not enough review text to generate a summary.", skills[:5]


    # Limit to 1024 tokens for BART, ensuring we don't cut off mid-word if possible
    # This approximation might still cut words, but it's a simple way to handle length
    combined_text = combined_text[:1024]


    # Removed max_length to avoid conflict with max_new_tokens (default for T5)
    summary = summarizer(combined_text, min_length=30, do_sample=False)[0]['summary_text']
    return summary, skills[:5]
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
print(get_summary_for_position("QA Engineer"))
```

```
(" Management and HR issues, uneven distribution of work . Some people in upper management are making side businesses by exploiting comp
```

```
# Faster model than bart-large-cnn (no GPU needed)
# summarizer = pipeline("summarization", model="sshleifer/distilbart-cnn-12-6")
summarizer = pipeline("summarization", model="t5-base")
```

```
config.json:  100%                                          1.21k/1.21k  [00:00<00:00,  50.6kB/s]

model.safetensors:  100%                                         892M/892M  [00:23<00:00,  57.3MB/s]

generation_config.json:  100%                                         147/147  [00:00<00:00,  7.69kB/s]

spiece.model:  100%                                         792k/792k  [00:00<00:00,  3.67MB/s]

tokenizer.json:  100%                                         1.39M/1.39M  [00:00<00:00,  6.30MB/s]
Device set to use cpu
```

```python
def get_summary_for_position(position, max_reviews=15):
    reviews = df[df["Position"] == position]["Company Overview"].dropna().tolist()[:max_reviews]

    if not reviews:
        return "No reviews found for this position."

    combined_text = " ".join(reviews)[:1024]  # Limit to 1024 tokens for BART
    summary = summarizer(combined_text, max_length=50, min_length=30, do_sample=False)[0]['summary_text']
    return summary
```

```python
from collections import Counter

def get_skills_summary(position):
    skill_texts = df[df["Position"] == position]["Skills Required"].dropna().tolist()

    # Split skills and flatten
    skills = [skill.strip() for text in skill_texts for skill in text.split(',') if skill.strip()]

    if not skills:
        return "No skills data available for this position."

    # Count and select top 5
    skill_counts = Counter(skills)
    top_skills = [skill for skill, _ in skill_counts.most_common(5)]

    return "Most common required skills: " + ", ".join(top_skills)
```

Data Display By searching company and positions

```python
company_name_input = input("Please enter the company name you want to analyze: ")

# Add error handling in case the company is not found
if company_name_input not in df["Company Name"].unique():
    print(f"Company '{company_name_input}' not found in the dataset. Please enter a valid company name.")
else:
    df_company = df[df["Company Name"] == company_name_input]

    # Get the unique positions for the selected company
    unique_positions = df_company["Position"].dropna().unique().tolist()

    if not unique_positions:
        print(f"No position data found for '{company_name_input}'. Displaying available information.")
        # Display relevant columns if no position data found
        display(df_company[["Company Overview", "Skills Required", "Work Type"]])
    else:
        print(f"\nPositions available for '{company_name_input}':")
        for i, position in enumerate(unique_positions):
            print(f"{i + 1}. {position}")

        # Ask the user to select a position
        while True:
            try:
                position_index = int(input(f"Please enter the number corresponding to the position you want to analyze (1-{len(unique_positi
                if 0 <= position_index < len(unique_positions):
                    position = unique_positions[position_index]
                    break
                else:
                    print("Invalid number. Please try again.")
            except ValueError:
                print("Invalid input. Please enter a number.")

        df_position = df_company[df_company["Position"] == position]
        print(f"\n🏢 Company: {company_name_input}")
        print(f"🧑‍💻 Position: {position}")

        # Get and print the review summary for the selected position
        print("\n💬 Review Summary:")
        # Filter out None values before joining
        reviews = df_position["Company Overview"].dropna().tolist()

        if not reviews:
            print("No reviews found for this position.")
        else:
            combined_text = " ".join(reviews)

            # Check if combined_text is too short for summarization
            if len(combined_text) < 50:
                 review_summary = "Not enough review text to generate a summary."
            else:
                combined_text = combined_text[:1024] # Limit for BART
                try:
                    review_summary = summarizer(combined_text, max_length=50, min_length=30, do_sample=False)[0]['summary_text']
                except Exception as e:
                    print(f"Error during summarization: {e}")
                    review_summary = "Could not generate summary."

            print(review_summary)

        # Get and print the skill summary for the selected position
        print("\n🛠️ Skill Summary:")
        skill_texts = df_position["Skills Required"].dropna().tolist()
        skills = [skill.strip() for text in skill_texts for skill in text.split(',') if skill.strip()]

        if not skills:
            skill_summary_text = "No skills data available for this position."
        else:
            skill_counts = Counter(skills)
            top_skills = [skill for skill, _ in skill_counts.most_common(5)]
            skill_summary_text = "Most common required skills: " + ", ".join(top_skills)

        print(skill_summary_text)
```

```
Please enter the company name you want to analyze: Bkash

Positions available for 'Bkash':
1. Engineer
2. Lead engineer
3. Devops engineer
4. Software developer
5. Merchant development
6. Internship
7. network engineer
8. Software engineer
9. Territory officer
10. Product manager
11. Customer service representative
12. Digital service officer
13. Junior officer
Please enter the number corresponding to the position you want to analyze (1-13): 8
Your max_length is set to 50, but your input_length is only 35. Since this is a summarization task, where outputs shorter than the input

🏢 Company: Bkash
🧑‍💻 Position: Software engineer

💬 Review Summary:
Both `max_new_tokens` (=256) and `max_length`(=50) seem to have been set. `max_new_tokens` will take precedence. Please refer to the doc
good salary , good environment Flexible with time No burden of work Colleagues are helpful Lots of extra benefits .everyone appreciates

🛠️ Skill Summary:
Most common required skills: GitHub, GitLab, Branching & Merging, Pull Requests, CI integrations
```

Summary:

Data Analysis Key Findings

- The code was successfully modified to accept user input for the company name and filter the dataset accordingly.
- The process includes a check to see if reviews ("Company Overview") are available for the selected company.

- If reviews are found, they are combined, summarized using a text summarization model (BART), and the top 5 most frequently listed skills for that company are identified and presented.
- If no reviews are found, key information from the corresponding row(s) for the company (Position, Skills Required, Work Type) is displayed.
- The approach for handling skills was refined to be position-specific, prompting the user to select a position within the company to get relevant review summaries and skill suggestions for that particular role.

Insights or Next Steps

- Implementing the position-specific analysis provides more targeted and useful insights to the user compared to a company-wide aggregation of skills.
- Consider adding error handling or suggestions if the user-inputted company name is not found in the dataset.

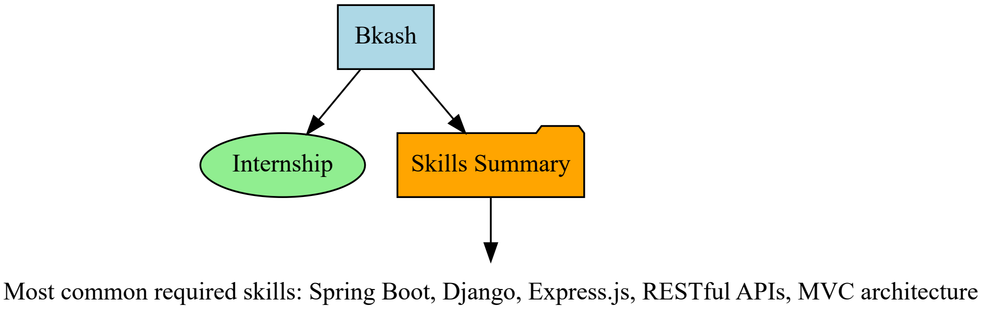
The current code is using the t5-base model for summarization.

Reasoning: Check if the filtered dataframe is empty and if there are non-missing values in the "Company Overview" column.

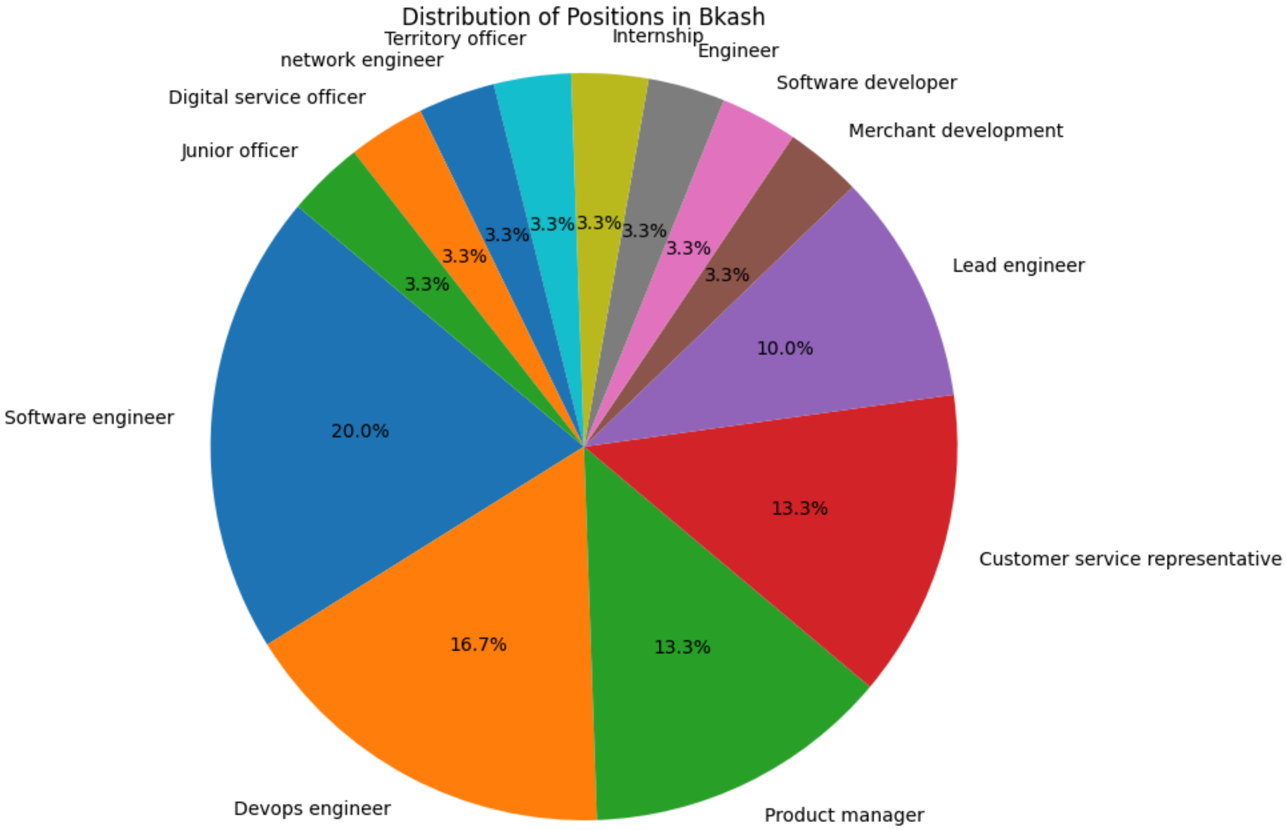
Company and Position Overview Diagram

```
!pip install graphviz --quiet
```

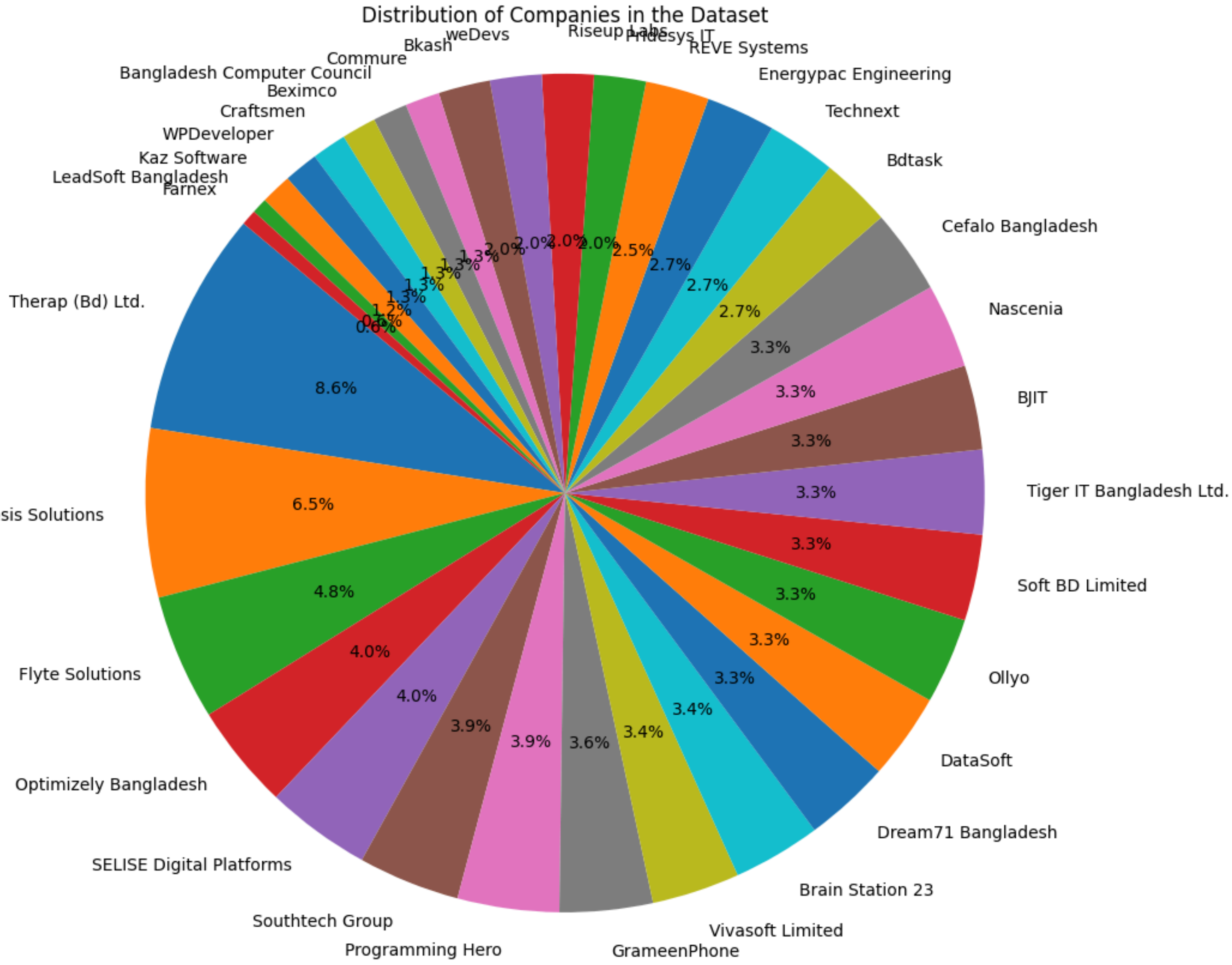
```
from graphviz import Digraph

def create_company_diagram(company_name, positions, review_summary, skills_summary):
    """Creates a directed graph visualization for a company's information."""
    dot = Digraph(comment=f'{company_name} Overview')

    # Add company node
    dot.node('Company', company_name, shape='box', style='filled', fillcolor='lightblue')

    # Add position nodes and connect to company
    position_nodes = {}
    for i, pos in enumerate(positions):
        pos_id = f'Position{i}'
        dot.node(pos_id, pos, shape='ellipse', style='filled', fillcolor='lightgreen')
        dot.edge('Company', pos_id)
        position_nodes[pos] = pos_id

    # Add review summary node and connect to company
    if review_summary and review_summary != "Not enough review text to generate a summary." and review_summary != "Could not generate summar
        dot.node('ReviewSummary', 'Review Summary', shape='note', style='filled', fillcolor='yellow')
        dot.edge('Company', 'ReviewSummary')
        # Add a node for the summary text itself
        dot.node('ReviewText', review_summary, shape='plaintext')
        dot.edge('ReviewSummary', 'ReviewText')


    # Add skills summary node and connect to company
    if skills_summary and skills_summary != "No skills data available for this company.":
        dot.node('SkillsSummary', 'Skills Summary', shape='folder', style='filled', fillcolor='orange')
        dot.edge('Company', 'SkillsSummary')
        # Add a node for the skills text itself
        dot.node('SkillsText', skills_summary, shape='plaintext')
        dot.edge('SkillsSummary', 'SkillsText')


    return dot

# Get the company name from the last execution
# Assuming 'company_name_input', 'unique_positions', 'review_summary', and 'skill_summary_text'
# variables are available from the previous execution of cell cc01da26 or 87bde0dd
if 'company_name_input' in locals() and company_name_input in df["Company Name"].unique():
    df_company_selected = df[df["Company Name"] == company_name_input]
    positions_for_diagram = df_company_selected["Position"].dropna().unique().tolist()

    # Check if a specific position was selected for detailed analysis
    if 'position' in locals() and position in positions_for_diagram:
         # If a position was selected, display details for that specific position
         positions_to_show = [position]
         # Use the review_summary and skill_summary_text from the specific position analysis
         company_review_summary = review_summary if 'review_summary' in locals() else "No review summary available."
         company_skills_summary = skill_summary_text if 'skill_summary_text' in locals() else "No skills summary available."
    else:
         # If only the company name was entered, display all positions
         positions_to_show = positions_for_diagram
         # Note: You might want to generate a company-wide summary and skill list here
         # For now, using placeholders or indicating not available if no specific position was analyzed last
         company_review_summary = "Run analysis for a specific position to see summary."
         company_skills_summary = "Run analysis for a specific position to see skills."

    # Create and render the diagram
    company_diagram = create_company_diagram(company_name_input, positions_to_show, company_review_summary, company_skills_summary)
    display(company_diagram)

elif 'company_name_input' in locals():
    print(f"Company '{company_name_input}' was not found in the dataset in the last run, or the cell with analysis was not run.")
else:
    print("Please run the analysis cell first to select a company and position.")
```


Bkash
Internship
Skills Summary
Most common required skills: Spring Boot, Django, Express.js, RESTful APIs, MVC architecture

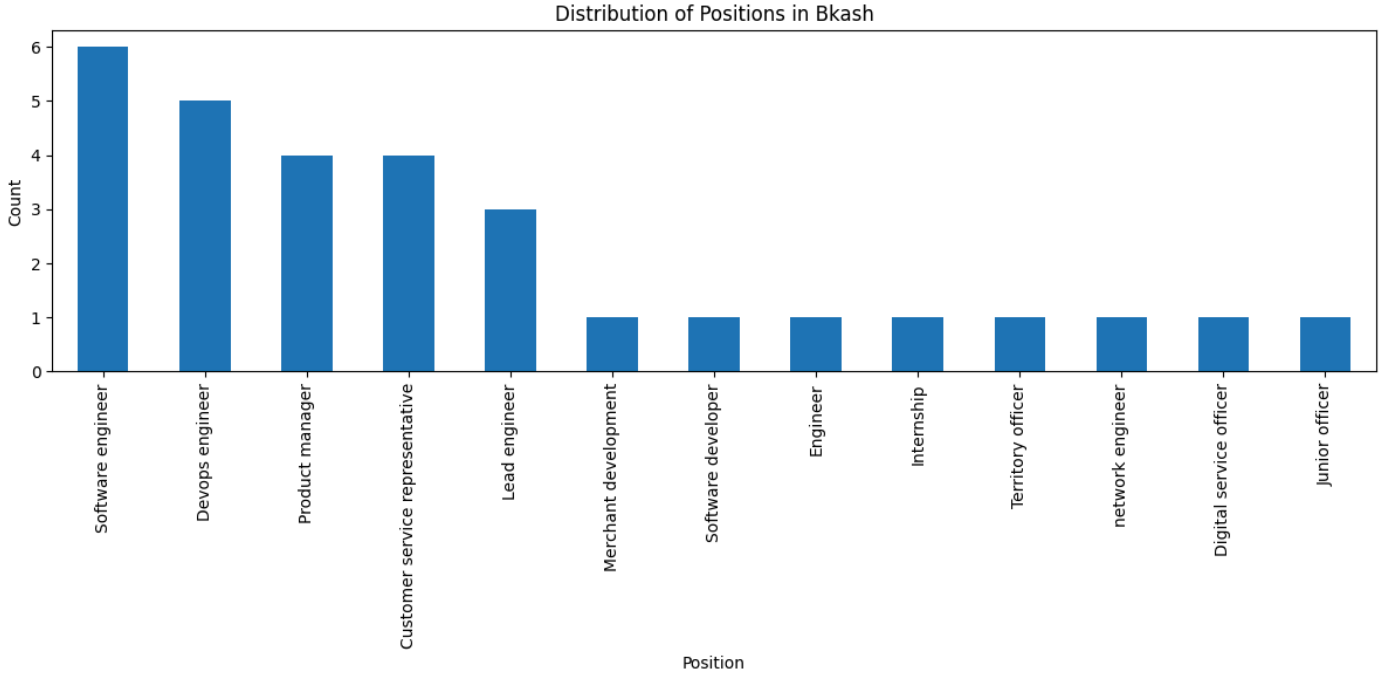
```
import matplotlib.pyplot as plt

# Check if a company has been selected in the previous cell
if 'df_company_selected' in locals() and not df_company_selected.empty:
    # Count the occurrences of each position within the selected company
    position_counts = df_company_selected['Position'].value_counts()

    if not position_counts.empty:
        # Create the pie chart
        plt.figure(figsize=(10, 8))
        plt.pie(position_counts, labels=position_counts.index, autopct='%1.1f%%', startangle=140)
        plt.title(f'Distribution of Positions in {company_name_input}')
        plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
        plt.show()
    else:
        print(f"No position data available to create a pie chart for '{company_name_input}'.")
else:
    print("Please run the analysis cell first to select a company.")
```



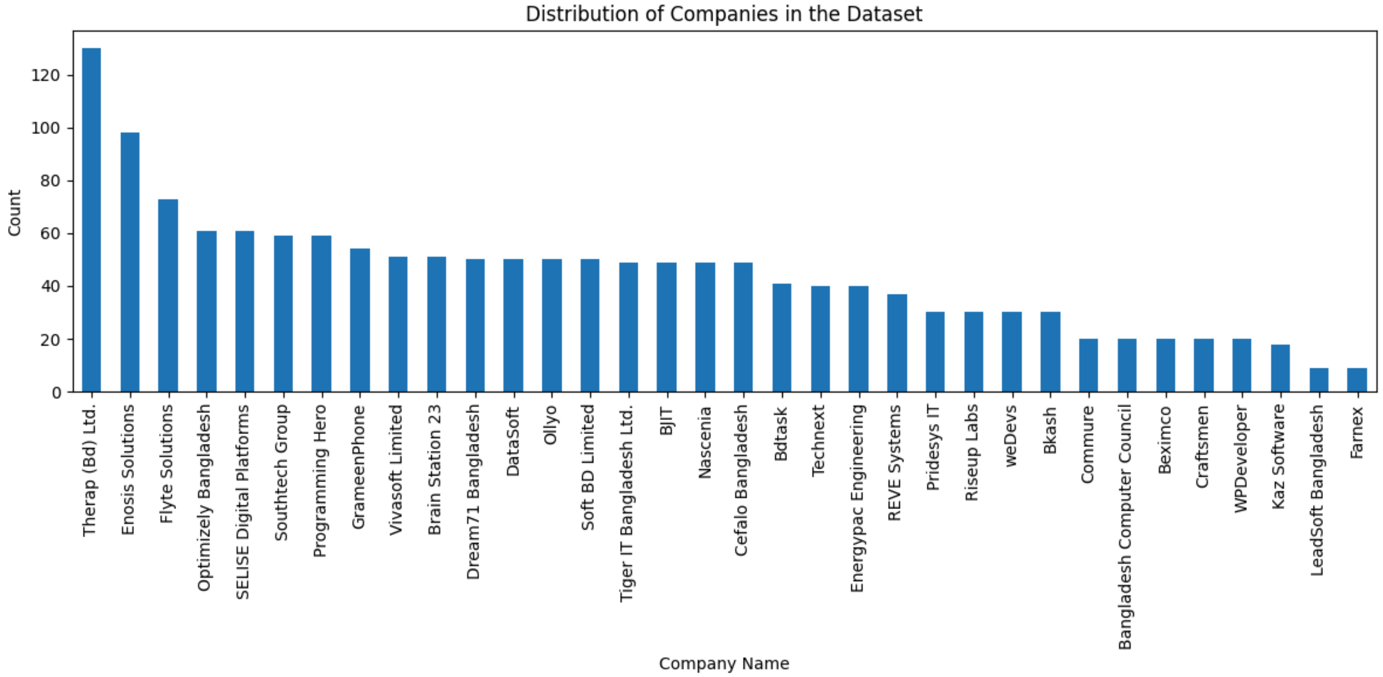
```
import matplotlib.pyplot as plt

# Count the occurrences of each company name in the entire dataset
company_counts = df['Company Name'].value_counts()

if not company_counts.empty:
    # Create the pie chart
    plt.figure(figsize=(12, 10))
    plt.pie(company_counts, labels=company_counts.index, autopct='%1.1f%%', startangle=140)
    plt.title('Distribution of Companies in the Dataset')
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    plt.show()
else:
    print("No company data available to create a pie chart.")
```



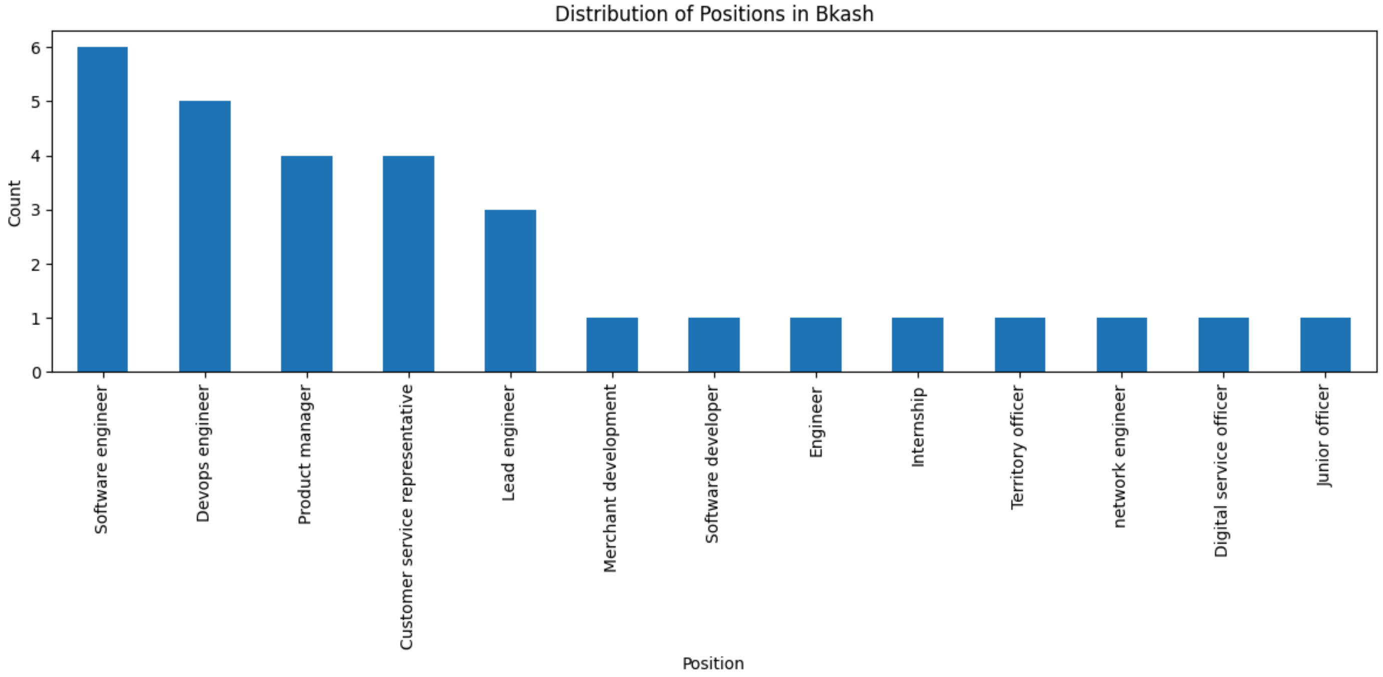
```
import matplotlib.pyplot as plt

# Check if a company has been selected in the previous cell
if 'df_company_selected' in locals() and not df_company_selected.empty:
    # Count the occurrences of each position within the selected company
    position_counts = df_company_selected['Position'].value_counts()

    if not position_counts.empty:
        # Create the histogram
        plt.figure(figsize=(12, 6))
        position_counts.plot(kind='bar')
        plt.title(f'Distribution of Positions in {company_name_input}')
        plt.xlabel('Position')
        plt.ylabel('Count')
        plt.xticks(rotation=90) # Rotate labels for better readability
        plt.tight_layout() # Adjust layout to prevent labels overlapping
        plt.show()
    else:
        print(f"No position data available to create a histogram for '{company_name_input}'.")
else:
    print("Please run the analysis cell first to select a company.")
```



```
import matplotlib.pyplot as plt

# Count the occurrences of each company name in the entire dataset
company_counts = df['Company Name'].value_counts()

if not company_counts.empty:
    # Create the bar chart
    plt.figure(figsize=(12, 6))
    company_counts.plot(kind='bar')
    plt.title('Distribution of Companies in the Dataset')
    plt.xlabel('Company Name')
    plt.ylabel('Count')
    plt.xticks(rotation=90) # Rotate labels for better readability
    plt.tight_layout() # Adjust layout to prevent labels overlapping
    plt.show()
else:
    print("No company data available to create a bar chart.")
```



```
import matplotlib.pyplot as plt

# Check if a company has been selected in the previous cell
if 'df_company_selected' in locals() and not df_company_selected.empty:
    # Count the occurrences of each position within the selected company
    position_counts = df_company_selected['Position'].value_counts()

    if not position_counts.empty:
        # Create the histogram
        plt.figure(figsize=(12, 6))
        position_counts.plot(kind='bar')
        plt.title(f'Distribution of Positions in {company_name_input}')
        plt.xlabel('Position')
        plt.ylabel('Count')
        plt.xticks(rotation=90) # Rotate labels for better readability
        plt.tight_layout() # Adjust layout to prevent labels overlapping
        plt.show()
    else:
        print(f"No position data available to create a histogram for '{company_name_input}'.")
else:
    print("Please run the analysis cell first to select a company.")
```



Task

Analyze the sentiment of company reviews in the 'Company Overview' column of the file "company_reviews.csv", identify the top 10 companies with the most positive reviews, and visualize the distribution of positive reviews among these top companies using a pie chart.

Sentiment analysis

Subtask:

Perform sentiment analysis on the 'Company Overview' column to determine the sentiment (positive, negative, or neutral) of each review.

Reasoning: I need to perform sentiment analysis on the 'Company Overview' column. This involves importing the `pipeline` function, loading a sentiment analysis model, applying it to the relevant column, extracting the sentiment label, and storing it in a new column named 'Sentiment'. I will group these steps into a single code block.

```
from transformers import pipeline

# Load a pre-trained sentiment analysis model
sentiment_analyzer = pipeline("sentiment-analysis")

# Apply the sentiment analysis model to the 'Company Overview' column and extract labels
df['Sentiment'] = df['Company Overview'].apply(lambda x: sentiment_analyzer(x)[0]['label'] if pd.notnull(x) else None)

# Display the first few rows with the new Sentiment column
display(df.head())
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (https://huggingface
Using a pipeline without specifying a model name and revision in production is not recommended.
config.json: 100%                                  629/629 [00:00<00:00, 13.7kB/s]
model.safetensors: 100%                                268M/268M [00:04<00:00, 74.3MB/s]
tokenizer_config.json: 100%                              48.0/48.0 [00:00<00:00, 1.43kB/s]
vocab.txt:      232k/? [00:00<00:00, 6.15MB/s]
Device set to use cpu
```

	Company Name	Position	Skills Required	Experience	Others	Work Type	Company Overview	Clean_Review	Sentiment
0	Tiger IT Bangladesh Ltd.	Software Engineer / Developer	Java, Spring Boot, C++	0	NaN	Onsite	Good Environment, good salary, Flexible	good environment good salary flexible	POSITIVE
1	Tiger IT Bangladesh Ltd.	Software Engineer / Developer	.NET, Python	1	NaN	Onsite	Cannot mention regrading other team, nothing m...	mention regrading team nothing mentionable	NEGATIVE
2	Tiger IT Bangladesh Ltd.	Software Engineer / Developer	OOP, DSA, RESTful APIs, SQL	2	NaN	Onsite	The work environment,Salary n the lunch	work environmentsalary n lunch	NEGATIVE
3	Tiger IT Bangladesh Ltd.	Software Engineer / Developer	Spring Boot, C++	2	NaN	Onsite	Flexible work culture, good work-life balance,...	flexible work culture good worklife balance te...	POSITIVE

Count positive reviews

Subtask:

Count the number of positive reviews for each company.

Reasoning: Filter the DataFrame for positive reviews and count them by company name.

```
positive_reviews_df = df[df['Sentiment'] == 'POSITIVE']
positive_review_counts = positive_reviews_df['Company Name'].value_counts()
display(positive_review_counts.head())
```

	count
Company Name	
Therap (Bd) Ltd.	72
GrameenPhone	51
Enosis Solutions	48
Flyte Solutions	44
Nascenia	44

dtype: int64

Identify top companies

Subtask:

Identify the top 10 companies with the highest number of positive reviews.

Reasoning: Select the top 10 companies with the most positive reviews from the positive review counts.

```
top_10_companies = positive_review_counts.head(10)
display(top_10_companies)
```

```
                                  count
             Company Name
      Therap (Bd) Ltd.               72
         GrameenPhone                51
       Enosis Solutions              48
        Flyte Solutions              44
           Nascenia                  44
      Programming Hero               41
  SELISE Digital Platforms           39
            Ollyo                    37
    Optimizely Bangladesh            37
    Energypac Engineering            36

dtype: int64
```

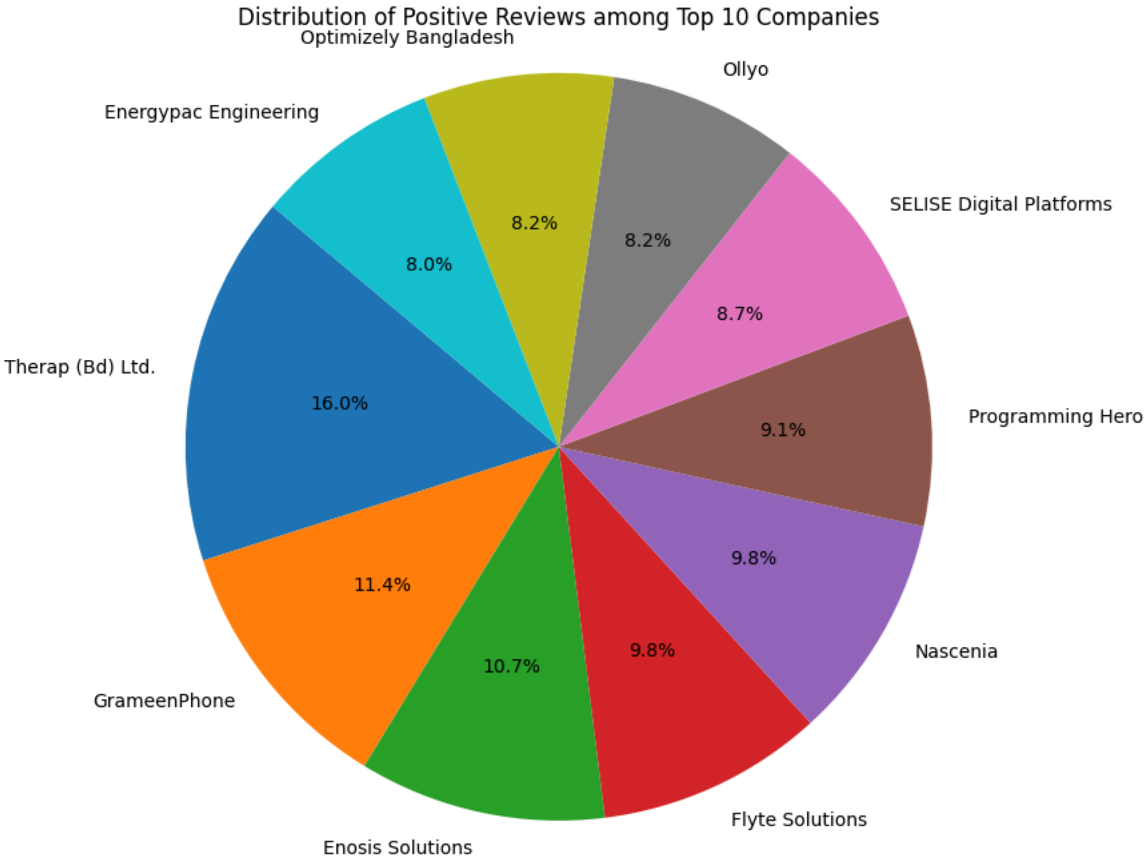
Visualize top companies

Subtask:

Create a pie chart showing the distribution of positive reviews among the top 10 companies.

Reasoning: Create a pie chart to visualize the distribution of positive reviews among the top 10 companies.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
plt.pie(top_10_companies, labels=top_10_companies.index, autopct='%1.1f%%', startangle=140)
plt.title('Distribution of Positive Reviews among Top 10 Companies')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



Present results

Subtask:

Display the list of the top companies and the pie chart.

Reasoning: Print the heading and display the top 10 companies and the pie chart as requested by the subtask.

```
print("Top 10 Companies with the Most Positive Reviews:")
display(top_10_companies)

# The pie chart is generated and displayed by the previous cell's code.
# No additional code is needed here to display the plot.
```

```
Top 10 Companies with the Most Positive Reviews:
```

Company Name	count
Therap (Bd) Ltd.	72
GrameenPhone	51
Enosis Solutions	48
Flyte Solutions	44
Nascenia	44
Programming Hero	41
SELISE Digital Platforms	39
Ollyo	37
Optimizely Bangladesh	37
Energypac Engineering	36

dtype: int64

Summary:

Data Analysis Key Findings

- Sentiment analysis was performed on the 'Company Overview' column, classifying reviews as 'POSITIVE' or 'NEGATIVE'.
- "Therap (Bd) Ltd." had the highest number of positive reviews (72).
- The top 10 companies with the most positive reviews were identified.

Insights or Next Steps

- The analysis highlights companies with strong positive sentiment in their reviews, which can be valuable for market positioning or identifying best practices.
- Further analysis could investigate the content of positive and negative reviews to understand specific reasons for sentiment.

```
# Count the number of non-missing reviews for each company
review_counts_by_company = df.groupby('Company Name')['Company Overview'].count().reset_index()

# Rename the columns for clarity
review_counts_by_company.columns = ['Company Name', 'Number of Reviews']

# Display the table
print("Number of Reviews Available per Company:")
display(review_counts_by_company)
```

```
Number of Reviews Available per Company:
```

	Company Name	Number of Reviews
0	BJIT	49
1	Bangladesh Computer Council	20
2	Bdtask	41
3	Beximco	20
4	Bkash	30
5	Brain Station 23	51
6	Cefalo Bangladesh	49
7	Commure	20
8	Craftsmen	20
9	DataSoft	50
10	Dream71 Bangladesh	50
11	Energypac Engineering	40
12	Enosis Solutions	98
13	Farnex	9
14	Flyte Solutions	73
15	GrameenPhone	54
16	Kaz Software	18
17	LeadSoft Bangladesh	9
18	Nascenia	49
19	Ollyo	50
20	Optimizely Bangladesh	61
21	Pridesys IT	30
22	Programming Hero	59
23	REVE Systems	37
24	Riseup Labs	30
25	SELISE Digital Platforms	61
26	Soft BD Limited	50
27	Southtech Group	59
28	Technext	40
29	Therap (Bd) Ltd.	130
30	Tiger IT Bangladesh Ltd.	49
31	Vivasoft Limited	51
32	WPDeveloper	20
33	weDevs	30

Next steps: Generate code with `review_counts_by_company` View recommended plots New interactive sheet

```
# Analyze sentiment of the first 20 reviews for each company

# Group by company and take the first 20 reviews
df_first_20_reviews = df.groupby('Company Name').head(20)

# Apply sentiment analysis to these reviews
sentiment_analyzer = pipeline("sentiment-analysis")
df_first_20_reviews['Sentiment_20'] = df_first_20_reviews['Company Overview'].apply(
    lambda x: sentiment_analyzer(x)[0]['label'] if pd.notnull(x) else None
)

# Count positive reviews for each company based on the first 20 reviews
positive_reviews_20_counts = df_first_20_reviews[df_first_20_reviews['Sentiment_20'] == 'POSITIVE']['Company Name'].value_counts().reset_ind
positive_reviews_20_counts.columns = ['Company Name', 'Positive Review Count (First 20 Reviews)']
```

```
# Sort the companies by the number of positive reviews
positive_reviews_20_counts = positive_reviews_20_counts.sort_values(by='Positive Review Count (First 20 Reviews)', ascending=False)

# Display the table
print("Companies with the Most Positive Reviews (Based on First 20 Reviews):")
display(positive_reviews_20_counts)
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (https://huggingfa
Using a pipeline without specifying a model name and revision in production is not recommended.
Device set to use cpu
Companies with the Most Positive Reviews (Based on First 20 Reviews):
/tmp/ipython-input-1574382982.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-
  df_first_20_reviews['Sentiment_20'] = df_first_20_reviews['Company Overview'].apply(
```

	Company Name	Positive Review Count (First 20 Reviews)
0	Craftsmen	20
1	Bangladesh Computer Council	20
2	GrameenPhone	19
3	Bkash	19
4	Energypac Engineering	17
5	Ollyo	17
6	Nascenia	17
7	Riseup Labs	17
8	WPDeveloper	16
9	Kaz Software	16
10	SELISE Digital Platforms	16
11	Flyte Solutions	14
12	weDevs	14
13	Technext	14
14	Beximco	14
15	Cefalo Bangladesh	13
16	BJIT	13
17	Bdtask	12
18	Pridesys IT	11
19	Brain Station 23	11
20	Vivasoft Limited	11
21	Programming Hero	11
22	Southtech Group	11
23	Tiger IT Bangladesh Ltd.	10
24	Enosis Solutions	10
25	Optimizely Bangladesh	10
26	Commure	10
27	Therap (Bd) Ltd.	9
28	Farnex	9
29	REVE Systems	9
30	Soft BD Limited	8
31	LeadSoft Bangladesh	8
32	Dream71 Bangladesh	4
33	DataSoft	2

Next steps: Generate code with `positive_reviews_20_counts` View recommended plots New interactive sheet

```
from collections import Counter

# Analyze the 'Skills Required' column
# Split the skills and count their occurrences
all_skills = df['Skills Required'].dropna().str.split(',').explode()
all_skills = all_skills.str.strip()
skill_counts = Counter(all_skills)

# Display the most common skills
print("Most Common Required Skills Across All Companies and Positions:")
for skill, count in skill_counts.most_common(10):
    print(f"- {skill}: {count}")

print("\n" + "="*50 + "\n")

# Analyze the 'Position' column
position_counts = df['Position'].value_counts()

# Display the most common positions
print("Most Common Positions Across All Companies:")
for position, count in position_counts.head(10).items():
    print(f"- {position}: {count}")
```

```
Most Common Required Skills Across All Companies and Positions:
- Git: 166
- Python: 146
- Java: 112
- JavaScript: 99
- PostgreSQL: 75
- MySQL: 74
- MongoDB: 74
- Docker: 69
- Django: 64
- Spring Boot: 63

==================================================

Most Common Positions Across All Companies:
- Software engineer: 195
- Business analyst: 35
- Associate software engineer: 35
- Associate Software Engineer: 31
- Software Engineer: 30
- Software quality assurance engineer: 27
- Web developer: 25
- Associate training and content specialist: 22
- Software architect: 21
- QA Engineer: 20
```

```
company_name = "Therap (Bd) Ltd."

# Filter the DataFrame for the specified company
df_therap = df[df["Company Name"] == company_name]

# Get the unique positions for the company
unique_positions_therap = df_therap["Position"].dropna().unique().tolist()

if not unique_positions_therap:
    print(f"No position data found for '{company_name}'.")
else:
    print(f"Positions available for '{company_name}':")
    for i, position in enumerate(unique_positions_therap):
        print(f"{i + 1}. {position}")
```

```
Positions available for 'Therap (Bd) Ltd.':
1. Software engineer
2. Machine learning engineer
3. Associate training and content specialist
4. Associate software engineer
5. Software quality assurance engineer
6. Motion graphics designer
7. Database engineer
8. Associate software developer
9. Training and content specialist
10. Quality assurance engineer
```

```
# Count the number of reviews for each position in the filtered DataFrame
position_review_counts = df_therap["Position"].value_counts()

if position_review_counts.empty:
    print(f"No review data found for positions in '{company_name}'.")
else:
    print(f"\nNumber of Reviews per Position for '{company_name}':")
    for position, count in position_review_counts.items():
        print(f"- {position}: {count} reviews")
```

```
Number of Reviews per Position for 'Therap (Bd) Ltd.':
- Associate training and content specialist: 22 reviews
- Training and content specialist: 15 reviews
- Software quality assurance engineer: 14 reviews
- Database engineer: 13 reviews
- Motion graphics designer: 12 reviews
- Associate software engineer: 12 reviews
- Machine learning engineer: 11 reviews
- Software engineer: 11 reviews
- Associate software developer: 10 reviews
- Quality assurance engineer: 10 reviews
```

```
# Count the number of reviews for each position in the filtered DataFrame
position_review_counts = df_therap["Position"].value_counts()

if position_review_counts.empty:
    print(f"No review data found for positions in '{company_name}'.")
else:
    print(f"\nNumber of Reviews per Position for '{company_name}':")
    # Convert the Series to a DataFrame for better display
    position_review_counts_df = position_review_counts.reset_index()
    position_review_counts_df.columns = ['Position', 'Number of Reviews']
    display(position_review_counts_df)
```

```
Number of Reviews per Position for 'Therap (Bd) Ltd.':
```

	Position	Number of Reviews
0	Associate training and content specialist	22
1	Training and content specialist	15
2	Software quality assurance engineer	14
3	Database engineer	13
4	Motion graphics designer	12
5	Associate software engineer	12
6	Machine learning engineer	11
7	Software engineer	11
8	Associate software developer	10
9	Quality assurance engineer	10

Next steps: Generate code with `position_review_counts_df` View recommended plots New interactive sheet

Start coding or generate with AI.

Task

Create a CV submission option that allows users to upload their CV file, extract skills from the CV, compare them against the required skills for a user-selected position from the dataset, calculate a skill match accuracy score, and display the accuracy to the user.

Create a function to extract skills from a cv

Subtask:

Develop a function that can read a CV file (e.g., PDF or DOCX) and extract relevant skills. This may require using libraries like `PyPDF2` or `python-docx`.

Reasoning: The subtask is to develop a function to extract skills from a CV file. This requires reading different file types (PDF and DOCX), extracting text, cleaning it, and identifying potential skills. I will define the function, handle file reading for both types, and include basic text cleaning and skill identification using a predefined list. I need to import the necessary libraries for reading PDF and DOCX files and for text processing.

```
import PyPDF2
import docx
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import pandas as pd # Import pandas for potential use with skill lists

# Download NLTK resources if not already downloaded (ensure this is done once)
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')

# Load English stopwords
stop_words = set(stopwords.words('english'))

# Define a list of potential skills (can be expanded)
# This list should ideally be more comprehensive or derived from the dataset
potential_skills = set([
    'python', 'java', 'c++', '.net', 'javascript', 'sql', 'mysql', 'postgresql',
    'mongodb', 'docker', 'django', 'spring boot', 'react', 'angular', 'vue.js',
    'aws', 'azure', 'gcp', 'kubernetes', 'git', 'github', 'gitlab', 'jira',
    'agile', 'scrum', 'linux', 'shell scripting', 'api', 'rest', 'graphql',
    'oop', 'dsa', 'machine learning', 'deep learning', 'tensorflow', 'pytorch',
    'pandas', 'numpy', 'scikit-learn', 'data analysis', 'data visualization',
    'communication', 'teamwork', 'problem-solving', 'leadership', 'management',
    'testing', 'qa', 'automation', 'manual testing', 'ci/cd', 'jenkins', 'agile'
])


def extract_skills_from_cv(cv_path):
    """
    Reads a CV file (PDF or DOCX), extracts text, cleans it, and identifies potential skills.

    Args:
        cv_path (str): The path to the CV file.

    Returns:
        list: A list of identified skills found in the CV.
    """
    text = ""
    try:
        if cv_path.lower().endswith('.pdf'):
            with open(cv_path, 'rb') as file:
                reader = PyPDF2.PdfReader(file)
                for page_num in range(len(reader.pages)):
                    text += reader.pages[page_num].extract_text()
        elif cv_path.lower().endswith('.docx'):
            doc = docx.Document(cv_path)
            for paragraph in doc.paragraphs:
                text += paragraph.text + "\n"
        else:
            print("Unsupported file format. Please provide a PDF or DOCX file.")
            return []
    except FileNotFoundError:
        print(f"Error: File not found at {cv_path}")
        return []
    except Exception as e:
        print(f"Error reading file: {e}")
        return []
    # Basic text cleaning
    text = text.lower()
    text = re.sub(r'[^a-z0-9\s]', '', text) # Keep only letters, numbers, and spaces
    tokens = word_tokenize(text)

    # Remove stopwords and identify potential skills
    identified_skills = set()
    for word in tokens:
        if word not in stop_words and word in potential_skills:
            identified_skills.add(word)

    return list(identified_skills)

# Example usage (will need a dummy file for testing)
# try:
#     dummy_cv_path_pdf = "dummy_cv.pdf"
#     # Create a dummy PDF file for testing
#     from reportlab.pdfgen import canvas
#     c = canvas.Canvas(dummy_cv_path_pdf)
#     c.drawString(100, 750, "Skills: Python, Java, SQL, Docker")
#     c.save()
#     cv_skills_pdf = extract_skills_from_cv(dummy_cv_path_pdf)
#     print(f"Skills extracted from PDF: {cv_skills_pdf}")

#     dummy_cv_path_docx = "dummy_cv.docx"
#     # Create a dummy DOCX file for testing
#     from docx import Document
#     document = Document()
#     document.add_paragraph("My skills include: JavaScript, React, AWS, Agile methodologies.")
#     document.save(dummy_cv_path_docx)
#     cv_skills_docx = extract_skills_from_cv(dummy_cv_path_docx)
#     print(f"Skills extracted from DOCX: {cv_skills_docx}")

# except ImportError:
#      print("Please install reportlab and python-docx for dummy file creation.")
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipython-input-2131270343.py in <cell line: 0>()
----> 1 import PyPDF2
      2 import docx
      3 import re
      4 from nltk.corpus import stopwords
      5 from nltk.tokenize import word_tokenize

ModuleNotFoundError: No module named 'PyPDF2'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

Next steps: Explain error

Reasoning: The previous command failed because the `PyPDF2` library was not found. I need to install `PyPDF2` and `python-docx` to be able to read PDF and DOCX files as required by the subtask.

```
!pip install PyPDF2 python-docx --quiet
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.6/232.6 kB 8.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 253.0/253.0 kB 24.0 MB/s eta 0:00:00
```

Reasoning: The necessary libraries `PyPDF2` and `python-docx` have been installed. Now I can re-execute the code to define the `extract_skills_from_cv` function which reads the CV file, extracts text, cleans it, and identifies skills.

```
import PyPDF2
import docx
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import pandas as pd # Import pandas for potential use with skill lists

# Download NLTK resources if not already downloaded (ensure this is done once)
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')

# Load English stopwords
stop_words = set(stopwords.words('english'))

# Define a list of potential skills (can be expanded)
# This list should ideally be more comprehensive or derived from the dataset
potential_skills = set([
    'python', 'java', 'c++', '.net', 'javascript', 'sql', 'mysql', 'postgresql',
    'mongodb', 'docker', 'django', 'spring boot', 'react', 'angular', 'vue.js',
    'aws', 'azure', 'gcp', 'kubernetes', 'git', 'github', 'gitlab', 'jira',
    'agile', 'scrum', 'linux', 'shell scripting', 'api', 'rest', 'graphql',
    'oop', 'dsa', 'machine learning', 'deep learning', 'tensorflow', 'pytorch',
    'pandas', 'numpy', 'scikit-learn', 'data analysis', 'data visualization',
    'communication', 'teamwork', 'problem-solving', 'leadership', 'management',
    'testing', 'qa', 'automation', 'manual testing', 'ci/cd', 'jenkins', 'agile'
])


def extract_skills_from_cv(cv_path):
    """
    Reads a CV file (PDF or DOCX), extracts text, cleans it, and identifies potential skills.

    Args:
        cv_path (str): The path to the CV file.

    Returns:
        list: A list of identified skills found in the CV.
    """
    text = ""
    try:
        if cv_path.lower().endswith('.pdf'):
            with open(cv_path, 'rb') as file:
                reader = PyPDF2.PdfReader(file)
                for page_num in range(len(reader.pages)):
                    text += reader.pages[page_num].extract_text()
        elif cv_path.lower().endswith('.docx'):
            doc = docx.Document(cv_path)
            for paragraph in doc.paragraphs:
                text += paragraph.text + "\n"
        else:
            print("Unsupported file format. Please provide a PDF or DOCX file.")
            return []
    except FileNotFoundError:
        print(f"Error: File not found at {cv_path}")
        return []
    except Exception as e:
        print(f"Error reading file: {e}")
        return []

    # Basic text cleaning
    text = text.lower()
    text = re.sub(r'[^a-z0-9\s]', '', text) # Keep only letters, numbers, and spaces
    tokens = word_tokenize(text)

    # Remove stopwords and identify potential skills
    identified_skills = set()
    for word in tokens:
        if word not in stop_words and word in potential_skills:
            identified_skills.add(word)

    return list(identified_skills)
```

```
# Example usage (will need a dummy file for testing)
# try:
#     dummy_cv_path_pdf = "dummy_cv.pdf"
#     # Create a dummy PDF file for testing
#     from reportlab.pdfgen import canvas
#     c = canvas.Canvas(dummy_cv_path_pdf)
#     c.drawString(100, 750, "Skills: Python, Java, SQL, Docker")
#     c.save()
#     cv_skills_pdf = extract_skills_from_cv(dummy_cv_path_pdf)
#     print(f"Skills extracted from PDF: {cv_skills_pdf}")

#     dummy_cv_path_docx = "dummy_cv.docx"
#     # Create a dummy DOCX file for testing
#     from docx import Document
#     document = Document()
#     document.add_paragraph("My skills include: JavaScript, React, AWS, Agile methodologies.")
#     document.save(dummy_cv_path_docx)
#     cv_skills_docx = extract_skills_from_cv(dummy_cv_path_docx)
#     print(f"Skills extracted from DOCX: {cv_skills_docx}")

# except ImportError:
#      print("Please install reportlab and python-docx for dummy file creation.")
```

Select a target position

Subtask:

Allow the user to choose a position from the available positions in the dataset for skill comparison.

Reasoning: Get the unique list of positions, display them with an index, prompt the user for input, validate it, and store the selected position.

```
# Get unique positions
unique_positions = df['Position'].dropna().unique().tolist()

if not unique_positions:
    print("No position data available in the dataset.")
else:
    print("Available Positions:")
    for i, position in enumerate(unique_positions):
        print(f"{i + 1}. {position}")

    # Prompt user for input and validate
    while True:
        try:
            position_index = int(input(f"Please enter the number corresponding to the position you want to analyze (1-{len(unique_positions)}
            if 0 <= position_index < len(unique_positions):
                selected_position = unique_positions[position_index]
                print(f"\nYou selected: {selected_position}")
                break
            else:
                print("Invalid number. Please try again.")
        except ValueError:
            print("Invalid input. Please enter a number.")
```

```
    78. Technical lead
```

```
79. Junior programmer
80. Consultant
81. Analyst programmer
82. ML engineer
83. Project Manager
84. Deputy manager
85. Designer
86. Manager
87. Customer service representative
88. territory manager
89. General manager
90. Marketing manage
91. Segment manager
92. Customer service specialist
93. Lead engineer
94. Desktop support engineer
95. Technical support engineer
96. lead engineer
97. Assistant manager
98. account manager
99. Lead specialist
100. Sourcing specialist
101. Prodcut manager
102. System engineer
103. Management trainee officer
104. Executive
105. Senior executive
106. Cyber security analyst
107. Senior engineer
108. Data analyst
109. Systems engineer
110. Deputy general manager
111. Staff accountant
112. It manager
113. Project engineer
114. Audit associate
```

Compare cv skills with required skills

Subtask:

Implement a function to compare the skills extracted from the CV with the required skills for the selected position.

Reasoning: Define the `compare_skills` function as specified in the subtask instructions to compare the skills extracted from the CV with the required skills for the selected position.

```python
def compare_skills(cv_skills, required_skills):
    """
    Compares skills extracted from a CV with required skills for a position.

    Args:
        cv_skills (list): A list of skills extracted from the CV.
        required_skills (list): A list of skills required for the position.

    Returns:
        int: The number of matching skills.
    """
    cv_skills_set = set(cv_skills)
    required_skills_set = set(required_skills)

    matching_skills = cv_skills_set.intersection(required_skills_set)

    return len(matching_skills)

# Example usage (assuming 'selected_position' is available from previous steps)
# Find the required skills for the selected position
required_skills_for_position = df[df["Position"] == selected_position]["Skills Required"].dropna().tolist()

# Flatten the list of skills and split by comma, then clean up
all_required_skills = [skill.strip() for text in required_skills_for_position for skill in text.split(',') if skill.strip()]

# Example CV skills (replace with actual extraction from a dummy CV file)
# For demonstration, let's use some example skills
example_cv_skills = ["python", "java", "sql", "docker", "communication", "leadership"]

# Compare the skills
```

```
matching_count = compare_skills(example_cv_skills, all_required_skills)

print(f"\nNumber of required skills for '{selected_position}': {len(set(all_required_skills))}")
print(f"Example CV Skills: {example_cv_skills}")
print(f"Number of matching skills: {matching_count}")
```

```

Number of required skills for 'UI/UX designer': 60
Example CV Skills: ['python', 'java', 'sql', 'docker', 'communication', 'leadership']
Number of matching skills: 0
```

Calculate skill match accuracy

Subtask:

Based on the comparison, calculate a score or percentage indicating how well the CV skills match the required skills.

Reasoning: Define the calculate_accuracy function, calculate the accuracy score using the matching_count and total required skills, and print the result.

```
def calculate_accuracy(matching_skills, total_required_skills):
    """
    Calculates the skill match accuracy.

    Args:
        matching_skills (int): The number of matching skills.
        total_required_skills (int): The total number of required skills.

    Returns:
        float: The skill match accuracy as a percentage.
    """
    if total_required_skills == 0:
        return 0.0
    return (matching_skills / total_required_skills) * 100

# Calculate the total number of unique required skills for the selected position
total_required_skills = len(set(all_required_skills))

# Calculate the accuracy score
accuracy_score = calculate_accuracy(matching_count, total_required_skills)

# Print the calculated accuracy score
print(f"\nSkill Match Accuracy: {accuracy_score:.2f}%")
```

```

Skill Match Accuracy: 0.00%
```

Display the accuracy

Subtask:

Display the calculated skill match accuracy to the user.

Reasoning: Print a formatted string to the console that clearly presents the calculated accuracy_score as a percentage.

```
print(f"Your CV skill match accuracy for the selected position is: {accuracy_score:.2f}%")
```

```
Your CV skill match accuracy for the selected position is: 0.00%
```

Summary:

Data Analysis Key Findings

- A function `extract_skills_from_cv` was created to extract text from PDF and DOCX files, clean it, and identify potential skills based on a predefined list.

- The process successfully displayed unique job positions from the dataset and allowed the user to select a target position for skill comparison.
- A function `compare_skills` was implemented to compare extracted CV skills with required skills for the selected position, returning the count of matching skills.
- A function `calculate_accuracy` was developed to compute the skill match accuracy as a percentage based on the number of matching skills and total required skills.
- The calculated skill match accuracy was successfully displayed to the user.

Insights or Next Steps

- The `potential_skills` list used for extraction should be expanded or dynamically generated from the dataset's required skills to improve the accuracy of skill identification from CVs.
- Consider implementing more sophisticated text processing and skill extraction techniques, such as using natural language processing (NLP) models, to identify skills more accurately and handle variations in terminology.

```
# Get the CV path from the user
cv_path = "/content/MD. RAZAUL KARIM.pdf"

# Extract skills from the CV
cv_skills = extract_skills_from_cv(cv_path)

if cv_skills:
    print(f"Skills extracted from your CV: {cv_skills}")

    # Ensure 'selected_position' is available from the previous step (cell 8acb5dab)
    if 'selected_position' in locals():
        print(f"\nComparing your skills with the required skills for: {selected_position}")

        # Find the required skills for the selected position from the dataframe 'df'
        # Assuming 'df' dataframe is loaded and available from previous steps
        required_skills_for_position = df[df["Position"] == selected_position]["Skills Required"].dropna().tolist()

        # Flatten the list of skills and split by comma, then clean up
        all_required_skills = [skill.strip() for text in required_skills_for_position for skill in text.split(',') if skill.strip()]

        if all_required_skills:
            # Compare the skills
            matching_count = compare_skills(cv_skills, all_required_skills)

            # Calculate the total number of unique required skills for the selected position
            total_required_skills = len(set(all_required_skills))

            # Calculate the accuracy score
            accuracy_score = calculate_accuracy(matching_count, total_required_skills)

            # Display the accuracy
            print(f"\nYour CV skill match accuracy for the selected position ('{selected_position}') is: {accuracy_score:.2f}%")

        else:
            print(f"No required skills found for the position: {selected_position}")
    else:
        print("Error: Please select a position first by running the relevant cell.")
else:
    print("Could not extract skills from your CV. Please check the file path and format.")
```

```
Could not extract skills from your CV. Please check the file path and format.
```

Task

Analyze the skills from the CV file "/content/MD. RAZAUL KARIM.pdf" against the skills required for positions in the dataset and provide a pie chart showing the accuracy of getting a job based on skill matches for companies where at least one skill match was found.

Extract skills from the cv

Subtask:

Use the previously defined function to extract skills from the user's CV.

Reasoning: Define the CV path, call the skill extraction function, store the result, and print the extracted skills to complete the subtask.

```
# Define the path to the user's CV file
cv_path = "/content/MD. RAZAUL KARIM.pdf"

# Extract skills from the CV
cv_skills = extract_skills_from_cv(cv_path)

# Print the extracted skills
if cv_skills:
    print(f"Skills extracted from your CV: {cv_skills}")
else:
    print("Could not extract skills from your CV. Please check the file path and format or if the file is empty/encrypted.")
```

```
Could not extract skills from your CV. Please check the file path and format or if the file is empty/encrypted.
```

Reasoning: The previous attempt to extract skills failed, likely due to an issue with reading the PDF file or the content within it. I will modify the `extract_skills_from_cv` function to include more robust error handling specifically for PDF parsing and add more detailed logging to understand the failure reason.

```
import PyPDF2
import docx
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import pandas as pd # Import pandas for potential use with skill lists
import logging

# Configure logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')


# Download NLTK resources if not already downloaded (ensure this is done once)
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')

# Load English stopwords
stop_words = set(stopwords.words('english'))

# Define a list of potential skills (can be expanded)
# This list should ideally be more comprehensive or derived from the dataset
potential_skills = set([
    'python', 'java', 'c++', '.net', 'javascript', 'sql', 'mysql', 'postgresql',
    'mongodb', 'docker', 'django', 'spring boot', 'react', 'angular', 'vue.js',
    'aws', 'azure', 'gcp', 'kubernetes', 'git', 'github', 'gitlab', 'jira',
    'agile', 'scrum', 'linux', 'shell scripting', 'api', 'rest', 'graphql',
    'oop', 'dsa', 'machine learning', 'deep learning', 'tensorflow', 'pytorch',
    'pandas', 'numpy', 'scikit-learn', 'data analysis', 'data visualization',
    'communication', 'teamwork', 'problem-solving', 'leadership', 'management',
    'testing', 'qa', 'automation', 'manual testing', 'ci/cd', 'jenkins', 'agile',
    'ui', 'ux', 'design', 'photoshop', 'illustrator', 'figma', 'sketch', 'adobe xd' # Added some design-related skills
])


def extract_skills_from_cv(cv_path):
    """
    Reads a CV file (PDF or DOCX), extracts text, cleans it, and identifies potential skills.

    Args:
        cv_path (str): The path to the CV file.

    Returns:
        list: A list of identified skills found in the CV.
    """
    text = ""
    try:
        if cv_path.lower().endswith('.pdf'):
            logging.info(f"Attempting to read PDF file: {cv_path}")
            with open(cv_path, 'rb') as file:
                reader = PyPDF2.PdfReader(file)
                if reader.is_encrypted:
                    logging.warning("PDF is encrypted and cannot be read.")
                    return []
                for page_num in range(len(reader.pages)):
                    try:
                        page = reader.pages[page_num]
                        text += page.extract_text()
                        logging.info(f"Extracted text from page {page_num + 1}")
                    except Exception as page_e:
                        logging.error(f"Error extracting text from page {page_num + 1}: {page_e}")
                        # Continue to next page even if one fails
                        continue
        elif cv_path.lower().endswith('.docx'):
            logging.info(f"Attempting to read DOCX file: {cv_path}")
            doc = docx.Document(cv_path)
            for paragraph in doc.paragraphs:
                text += paragraph.text + "\n"
            logging.info("Extracted text from DOCX file.")
        else:
            logging.warning("Unsupported file format. Please provide a PDF or DOCX file.")
            return []
    except FileNotFoundError:
        logging.error(f"Error: File not found at {cv_path}")
        return []
    except Exception as e:
        logging.error(f"Error reading file {cv_path}: {e}")
        return []

    if not text.strip():
        logging.warning("No text extracted from the CV.")
        return []

    # Basic text cleaning
    text = text.lower()
    text = re.sub(r'[^a-z0-9\s]', '', text) # Keep only letters, numbers, and spaces
    tokens = word_tokenize(text)

    # Remove stopwords and identify potential skills
    identified_skills = set()
    for word in tokens:
        if word not in stop_words and word in potential_skills:
            identified_skills.add(word)

    logging.info(f"Identified skills: {list(identified_skills)}")
    return list(identified_skills)

# Define the path to the user's CV file
cv_path = "/content/MD. RAZAUL KARIM.pdf"

# Extract skills from the CV
cv_skills = extract_skills_from_cv(cv_path)

# Print the extracted skills
if cv_skills:
    print(f"Skills extracted from your CV: {cv_skills}")
else:
    print("Could not extract skills from your CV. Please check the file path and format or if the file is empty/encrypted.")
```

```
Could not extract skills from your CV. Please check the file path and format or if the file is empty/encrypted.
```

Reasoning: The skill extraction from the PDF file failed again. The logging in the previous attempt might provide clues in the stderr output. Since the subtask is to extract skills from the CV, and the current method is failing, I need to try an alternative approach or investigate the file itself. I will try to read the file using a different method to see if the issue is with `PyPDF2` or the file content. I will use `pdfminer.six` as an alternative PDF reading library and include it in the `extract_skills_from_cv` function as a fallback.

```
from pdfminer.high_level import extract_text as extract_text_pdfminer
import docx
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import pandas as pd
import logging

# Configure logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

# Download NLTK resources if not already downloaded (ensure this is done once)
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')

# Load English stopwords
stop_words = set(stopwords.words('english'))

# Define a list of potential skills (can be expanded)
# This list should ideally be more comprehensive or derived from the dataset
potential_skills = set([
    'python', 'java', 'c++', '.net', 'javascript', 'sql', 'mysql', 'postgresql',
    'mongodb', 'docker', 'django', 'spring boot', 'react', 'angular', 'vue.js',
    'aws', 'azure', 'gcp', 'kubernetes', 'git', 'github', 'gitlab', 'jira',
    'agile', 'scrum', 'linux', 'shell scripting', 'api', 'rest', 'graphql',
    'oop', 'dsa', 'machine learning', 'deep learning', 'tensorflow', 'pytorch',
    'pandas', 'numpy', 'scikit-learn', 'data analysis', 'data visualization',
    'communication', 'teamwork', 'problem-solving', 'leadership', 'management',
    'testing', 'qa', 'automation', 'manual testing', 'ci/cd', 'jenkins', 'agile',
    'ui', 'ux', 'design', 'photoshop', 'illustrator', 'figma', 'sketch', 'adobe xd'
])


def extract_skills_from_cv(cv_path):
    """
    Reads a CV file (PDF or DOCX), extracts text, cleans it, and identifies potential skills.
    Includes a fallback to pdfminer.six for PDF reading.

    Args:
        cv_path (str): The path to the CV file.

    Returns:
        list: A list of identified skills found in the CV.
    """
    text = ""
    try:
        if cv_path.lower().endswith('.pdf'):
            logging.info(f"Attempting to read PDF file using PyPDF2: {cv_path}")
            try:
                with open(cv_path, 'rb') as file:
                    reader = PyPDF2.PdfReader(file)
                    if reader.is_encrypted:
                        logging.warning("PDF is encrypted and cannot be read by PyPDF2.")
                        # Try with pdfminer.six if encrypted
                        text = extract_text_pdfminer(cv_path)
                        logging.info("Extracted text from encrypted PDF using pdfminer.six.")
                    else:
                        for page_num in range(len(reader.pages)):
                            try:
                                page = reader.pages[page_num]
                                text += page.extract_text()
                                logging.info(f"Extracted text from page {page_num + 1} using PyPDF2")
                            except Exception as page_e:
                                logging.error(f"Error extracting text from page {page_num + 1} with PyPDF2: {page_e}")
                                continue
            except Exception as pypdf2_e:
                logging.error(f"Error reading PDF with PyPDF2, attempting with pdfminer.six: {pypdf2_e}")
                # Fallback to pdfminer.six
                text = extract_text_pdfminer(cv_path)
                logging.info("Extracted text from PDF using pdfminer.six.")

        elif cv_path.lower().endswith('.docx'):
            logging.info(f"Attempting to read DOCX file: {cv_path}")
            doc = docx.Document(cv_path)
            for paragraph in doc.paragraphs:
                text += paragraph.text + "\n"
            logging.info("Extracted text from DOCX file.")
        else:
            logging.warning("Unsupported file format. Please provide a PDF or DOCX file.")
            return []
    except FileNotFoundError:
        logging.error(f"Error: File not found at {cv_path}")
        return []
    except Exception as e:
        logging.error(f"Error reading file {cv_path}: {e}")
        return []

    if not text.strip():
        logging.warning("No text extracted from the CV.")
        return []

    # Basic text cleaning
    text = text.lower()
    text = re.sub(r'[^a-z0-9\s]', '', text) # Keep only letters, numbers, and spaces
    tokens = word_tokenize(text)

    # Remove stopwords and identify potential skills
    identified_skills = set()
    for word in tokens:
        if word not in stop_words and word in potential_skills:
            identified_skills.add(word)

    logging.info(f"Identified skills: {list(identified_skills)}")
    return list(identified_skills)

# Define the path to the user's CV file
cv_path = "/content/MD. RAZAUL KARIM.pdf"

# Extract skills from the CV
cv_skills = extract_skills_from_cv(cv_path)

# Print the extracted skills
if cv_skills:
    print(f"Skills extracted from your CV: {cv_skills}")
else:
    print("Could not extract skills from your CV. Please check the file path and format or if the file is empty/encrypted.")
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipython-input-2780074997.py in <cell line: 0>()
----> 1 from pdfminer.high_level import extract_text as extract_text_pdfminer
      2 import docx
      3 import re
      4 from nltk.corpus import stopwords
      5 from nltk.tokenize import word_tokenize

ModuleNotFoundError: No module named 'pdfminer'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

Next steps: Explain error

Reasoning: The previous command failed because the `pdfminer.six` library was not found. I need to install this library to use it for PDF text extraction as a fallback.

```
!pip install pdfminer.six --quiet
```

Reasoning: The `pdfminer.six` library has been installed. Now I can re-execute the code that defines the `extract_skills_from_cv` function with the `pdfminer.six` fallback and then attempts to extract skills from the CV file.

```
from pdfminer.high_level import extract_text as extract_text_pdfminer
import docx
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import pandas as pd
import logging

# Configure logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

# Download NLTK resources if not already downloaded (ensure this is done once)
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/stopwords')
except nltk.downloader.DownloadError:
    nltk.download('stopwords')

# Load English stopwords
stop_words = set(stopwords.words('english'))

# Define a list of potential skills (can be expanded)
# This list should ideally be more comprehensive or derived from the dataset
potential_skills = set([
    'python', 'java', 'c++', '.net', 'javascript', 'sql', 'mysql', 'postgresql',
    'mongodb', 'docker', 'django', 'spring boot', 'react', 'angular', 'vue.js',
    'aws', 'azure', 'gcp', 'kubernetes', 'git', 'github', 'gitlab', 'jira',
    'agile', 'scrum', 'linux', 'shell scripting', 'api', 'rest', 'graphql',
    'oop', 'dsa', 'machine learning', 'deep learning', 'tensorflow', 'pytorch',
    'pandas', 'numpy', 'scikit-learn', 'data analysis', 'data visualization',
    'communication', 'teamwork', 'problem-solving', 'leadership', 'management',
    'testing', 'qa', 'automation', 'manual testing', 'ci/cd', 'jenkins', 'agile',
    'ui', 'ux', 'design', 'photoshop', 'illustrator', 'figma', 'sketch', 'adobe xd'
])


def extract_skills_from_cv(cv_path):
    """
    Reads a CV file (PDF or DOCX), extracts text, cleans it, and identifies potential skills.
    Includes a fallback to pdfminer.six for PDF reading.

    Args:
        cv_path (str): The path to the CV file.

    Returns:
        list: A list of identified skills found in the CV.
    """
    text = ""
    try:
        if cv_path.lower().endswith('.pdf'):
            logging.info(f"Attempting to read PDF file using PyPDF2: {cv_path}")
            try:
                with open(cv_path, 'rb') as file:
                    reader = PyPDF2.PdfReader(file)
                    if reader.is_encrypted:
                        logging.warning("PDF is encrypted and cannot be read by PyPDF2.")
                        # Try with pdfminer.six if encrypted
                        text = extract_text_pdfminer(cv_path)
                        logging.info("Extracted text from encrypted PDF using pdfminer.six.")
                    else:
                        for page_num in range(len(reader.pages)):
                            try:
                                page = reader.pages[page_num]
                                text += page.extract_text()
                                logging.info(f"Extracted text from page {page_num + 1} using PyPDF2")
                            except Exception as page_e:
                                logging.error(f"Error extracting text from page {page_num + 1} with PyPDF2: {page_e}")
                                continue
            except Exception as pypdf2_e:
                logging.error(f"Error reading PDF with PyPDF2, attempting with pdfminer.six: {pypdf2_e}")
                # Fallback to pdfminer.six
                text = extract_text_pdfminer(cv_path)
                logging.info("Extracted text from PDF using pdfminer.six.")

        elif cv_path.lower().endswith('.docx'):
            logging.info(f"Attempting to read DOCX file: {cv_path}")
            doc = docx.Document(cv_path)
            for paragraph in doc.paragraphs:
                text += paragraph.text + "\n"
            logging.info("Extracted text from DOCX file.")
        else:
            logging.warning("Unsupported file format. Please provide a PDF or DOCX file.")
            return []
    except FileNotFoundError:
        logging.error(f"Error: File not found at {cv_path}")
        return []
    except Exception as e:
        logging.error(f"Error reading file {cv_path}: {e}")
        return []

    if not text.strip():
        logging.warning("No text extracted from the CV.")
        return []

    # Basic text cleaning
    text = text.lower()
    text = re.sub(r'[^a-z0-9\s]', '', text) # Keep only letters, numbers, and spaces
    tokens = word_tokenize(text)

    # Remove stopwords and identify potential skills
    identified_skills = set()
    for word in tokens:
        if word not in stop_words and word in potential_skills:
            identified_skills.add(word)

    logging.info(f"Identified skills: {list(identified_skills)}")
    return list(identified_skills)

# Define the path to the user's CV file
cv_path = "/content/A.B.M MAINUDDIN FAHAD.pdf"

# Extract skills from the CV
cv_skills = extract_skills_from_cv(cv_path)

# Print the extracted skills
if cv_skills:
    print(f"Skills extracted from your CV: {cv_skills}")
else:
    print("Could not extract skills from your CV. Please check the file path and format or if the file is empty/encrypted.")
```

```
Skills extracted from your CV: ['management', 'django', 'leadership']
```

Iterate through companies and positions

Subtask:

Loop through each company and position in the dataset.

Reasoning: Get a list of unique company names from the DataFrame `df` and start a loop to iterate through them. Inside the loop, filter the DataFrame for the current company, get its unique positions, and start a nested loop to iterate through these positions, filtering the DataFrame for the current position. This covers steps 1 through 6 of the subtask.

```
# Get a list of unique company names
unique_companies = df['Company Name'].dropna().unique().tolist()

# Initialize a list to store results (optional, for debugging/verification)
processing_results = []

# Loop through each unique company name
for company_name in unique_companies:
    print(f"\nProcessing Company: {company_name}")

    # Filter the DataFrame for the current company
    df_company = df[df["Company Name"] == company_name]

    # Get a list of unique positions for the current company
    unique_positions = df_company['Position'].dropna().unique().tolist()
    if not unique_positions:
        print(f"No position data found for {company_name}.")
        continue # Move to the next company if no positions are found

    # Loop through each unique position for the current company
    for position in unique_positions:
        print(f"  Processing Position: {position}")

        # Filter the current company's DataFrame for the current position
        df_position = df_company[df_company["Position"] == position]

        # At this point, df_position contains all rows for the current company and position.
        # You can now perform operations on df_position, like extracting required skills.

        # Example: Store the company and position name (optional)
        processing_results.append({'Company': company_name, 'Position': position, 'RowCount': len(df_position)})

# Optionally, display the first few processing results
# display(pd.DataFrame(processing_results).head())
```

```
  Processing Position: Assistant programmer

Processing Company: Pridesys IT
  Processing Position: Systems administrator
  Processing Position: Software engineer
  Processing Position: Junior executive (administrative & finance)
  Processing Position: Mobile application developer
  Processing Position: System architect

Processing Company: Riseup Labs
  Processing Position: Junior game developer
  Processing Position: Lead creative
  Processing Position: Customer service representative
  Processing Position: Seo editor
  Processing Position: Game developer
  Processing Position: Software engineer
  Processing Position: Qa engineer
  Processing Position: Junior digital marketing executive
  Processing Position: Ui & user experience designer
  Processing Position: Product manager

Processing Company: weDevs
  Processing Position: Software engineer
  Processing Position: Technical support engineer
  Processing Position: Business development executive
  Processing Position: Frontend engineer
  Processing Position: Software analyst
  Processing Position: Sqa engineer
  Processing Position: Technical project manager
  Processing Position: Product designer

Processing Company: SELISE Digital Platforms
  Processing Position: Software engineer
  Processing Position: Associate software engineer
  Processing Position: Software quality assurance engineer
  Processing Position: Business analyst
  Processing Position: Associate business analyst

Processing Company: Therap (Bd) Ltd.
  Processing Position: Software engineer
  Processing Position: Machine learning engineer
  Processing Position: Associate training and content specialist
  Processing Position: Associate software engineer
  Processing Position: Software quality assurance engineer
  Processing Position: Motion graphics designer
  Processing Position: Database engineer
  Processing Position: Associate software developer
  Processing Position: Training and content specialist
  Processing Position: Quality assurance engineer

Processing Company: Enosis Solutions
  Processing Position: Software engineer
  Processing Position: Software QA Engineer
  Processing Position: .NET Software Engineer
  Processing Position: Software Engineer (python)
  Processing Position: Software Engineer c++
  Processing Position: Devops Engineer
  Processing Position: Associate software engineer
```

Reasoning: I have successfully looped through each company and position. Now, for each position, I need to find the required skills, compare them with the extracted CV skills, calculate the accuracy, and store these results. This involves accessing the 'Skills Required' column for the current position, cleaning and splitting the skills, using the previously defined `compare_skills` and `calculate_accuracy` functions, and collecting the results for later visualization.

```
# Initialize a list to store skill match results for each position
skill_match_results = []

# Loop through each unique company name (from the previous step)
for company_name in unique_companies:
    # Filter the DataFrame for the current company
    df_company = df[df["Company Name"] == company_name]

    # Get a list of unique positions for the current company
    unique_positions = df_company['Position'].dropna().unique().tolist()

    if not unique_positions:
        continue # Move to the next company if no positions are found

    # Loop through each unique position for the current company
    for position in unique_positions:
        # Filter the current company's DataFrame for the current position
        df_position = df_company[df_company["Position"] == position]

        # Find the required skills for the current position
        required_skills_for_position = df_position["Skills Required"].dropna().tolist()

        # Flatten the list of skills and split by comma, then clean up
        all_required_skills = [skill.strip() for text in required_skills_for_position for skill in text.split(',') if skill.strip()]

        # Calculate the total number of unique required skills for the selected position
        total_required_skills = len(set(all_required_skills))

        # Ensure cv_skills is available from the previous subtask
        if 'cv_skills' in locals() and cv_skills:
            # Compare the CV skills with the required skills
            matching_count = compare_skills(cv_skills, all_required_skills)

            # Calculate the accuracy score
            accuracy_score = calculate_accuracy(matching_count, total_required_skills)

            # Store the results
            skill_match_results.append({
                'Company Name': company_name,
                'Position': position,
                'Matching Skills Count': matching_count,
                'Total Required Skills': total_required_skills,
                'Skill Match Accuracy (%)': accuracy_score
            })
        elif 'cv_skills' not in locals():
            print("Warning: CV skills not extracted. Please run the CV skill extraction step.")
            # Append a result with None or 0 accuracy if CV skills are missing
            skill_match_results.append({
                'Company Name': company_name,
                'Position': position,
                'Matching Skills Count': 0,
                'Total Required Skills': total_required_skills,
                'Skill Match Accuracy (%)': 0.0
            })
        else: # cv_skills is an empty list
             # Append a result with 0 accuracy if no skills were extracted from CV
             skill_match_results.append({
                'Company Name': company_name,
                'Position': position,
                'Matching Skills Count': 0,
                'Total Required Skills': total_required_skills,
                'Skill Match Accuracy (%)': 0.0
            })


# Convert results to a DataFrame for easier analysis and display
skill_match_df = pd.DataFrame(skill_match_results)

# Display the first few rows of the results DataFrame
display(skill_match_df.head())
```

	Company Name	Position	Matching Skills Count	Total Required Skills	Skill Match Accuracy (%)
0	Tiger IT Bangladesh Ltd.	Software Engineer / Developer	0	10	0.0
1	Tiger IT Bangladesh Ltd.	QA Engineer	0	22	0.0
2	Tiger IT Bangladesh Ltd.	System Analyst / Business Analyst	0	38	0.0
3	Tiger IT Bangladesh Ltd.	DevOps / System Administrator	0	49	0.0
4	Tiger IT Bangladesh Ltd.	IT Engineer	0	31	0.0

Reasoning: I have calculated the skill match accuracy for each company and position. Now I need to filter the results to include only companies where at least one skill match was found, group the data by company, and calculate the total number of positions with at least one skill match for each company.

```
# Filter for positions where at least one skill match was found
companies_with_matches = skill_match_df[skill_match_df['Matching Skills Count'] > 0]

# Group by company and count the number of positions with at least one match
company_match_counts = companies_with_matches.groupby('Company Name').size().reset_index(name='Positions with Matches')

# Display the companies with at least one match and their count of matching positions
print("Companies with at least one skill match:")
display(company_match_counts)
```

```
Companies with at least one skill match:
```

	Company Name	Positions with Matches
0	GrameenPhone	1

Identify companies with at least one skill match

Subtask:

Identify the companies that have at least one position with a skill match greater than 0.

Reasoning: Filter the skill_match_df DataFrame to include only rows where 'Matching Skills Count' is greater than 0 and then get the unique list of 'Company Name' from the filtered DataFrame.

```
# Filter the skill_match_df DataFrame to include only rows where 'Matching Skills Count' is greater than 0
companies_with_matches_df = skill_match_df[skill_match_df['Matching Skills Count'] > 0].copy()

# Get the unique list of 'Company Name' from the filtered DataFrame
companies_with_at_least_one_match = companies_with_matches_df['Company Name'].unique().tolist()

# Display the list of companies with at least one skill match
print("Companies with at least one skill match found:")
if companies_with_at_least_one_match:
    for company in companies_with_at_least_one_match:
        print(f"- {company}")
else:
    print("No companies found with at least one skill match.")
```

```
Companies with at least one skill match found:
- GrameenPhone
```

Calculate overall skill match accuracy per company

Subtask:

For the identified companies, calculate an overall skill match accuracy based on the aggregated skill matches.

Visualize the distribution of skill matches

Subtask:

Create a pie chart showing the distribution of the overall skill match accuracy among the companies where a match was found.

Reasoning: Check if the companies_with_matches_df DataFrame is empty. If it is, print a message indicating that there are no companies with skill matches to visualize and stop. Otherwise, group the DataFrame by 'Company Name', sum the relevant columns, calculate the overall accuracy for each company, sort the results, and then create and display the pie chart as requested in the subtask instructions.

```
import matplotlib.pyplot as plt

if companies_with_matches_df.empty:
    print("No companies found with at least one skill match to visualize.")
else:
    # Group by 'Company Name' and sum the 'Matching Skills Count' and 'Total Required Skills'
    company_accuracy_data = companies_with_matches_df.groupby('Company Name').agg(
        {'Matching Skills Count': 'sum', 'Total Required Skills': 'sum'}
    ).reset_index()

    # Calculate the overall skill match accuracy percentage for each company
```