Job Market Analysis & Recommendation System

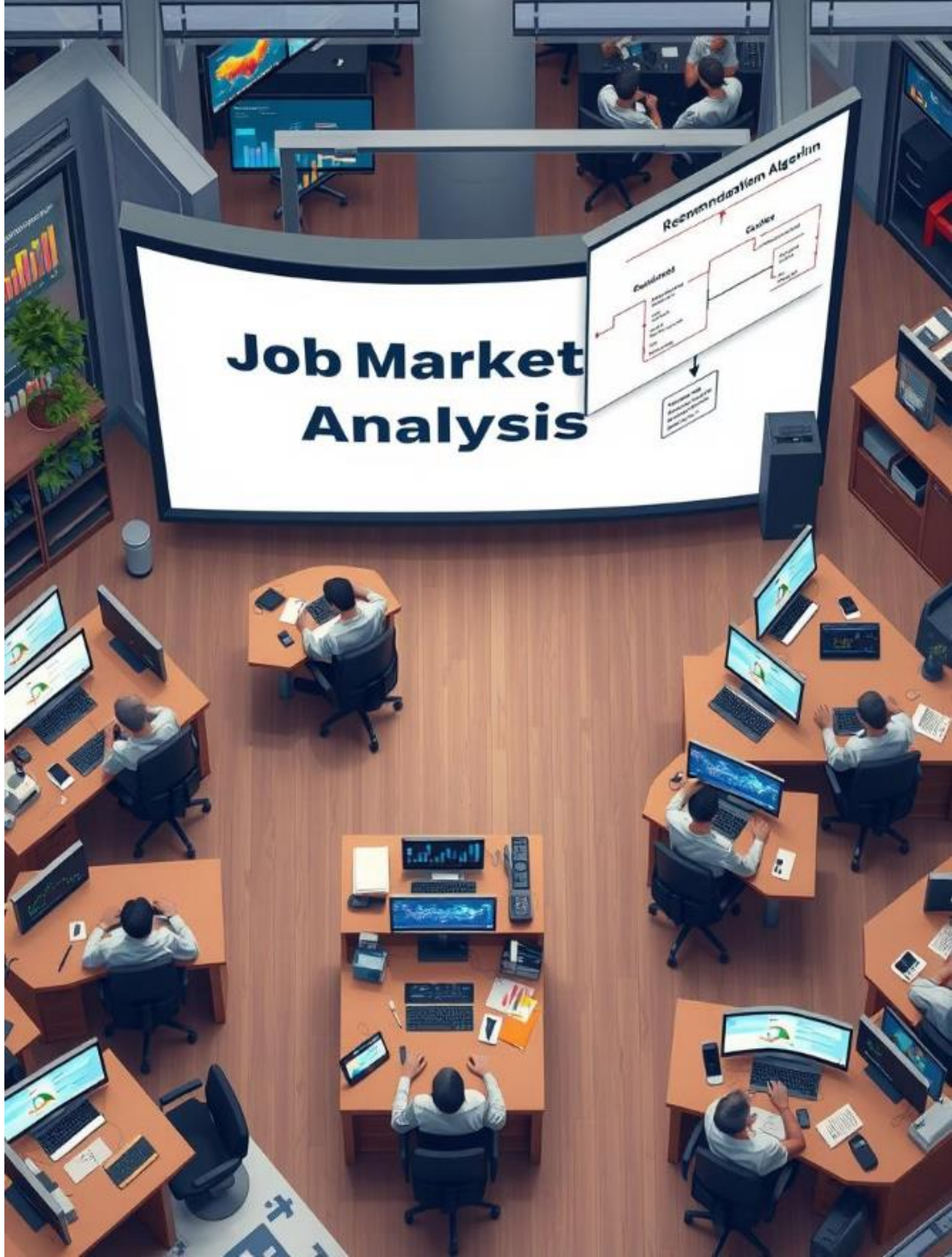
In this project, I developed a Job Market Analysis and Recommender System to identify trending job roles and recommend suitable positions based on user input. I began by collecting and preprocessing job data, standardizing fields like location, experience, and job type. I performed exploratory analysis to uncover patterns in demand across companies, countries, and time.

Using engineered temporal and categorical features, I applied time series analysis to detect emerging job trends and used KMeans clustering to group similar roles based on keywords and skill patterns. For personalized recommendations, I combined TF-IDF vectorization with a Nearest Neighbors model to match user descriptions to the most relevant job postings.

I built an interactive Streamlit app that accepts job descriptions, allows filtering by country, experience, and job type, and returns the top 10 recommended roles with similarity scores and keyword highlights. The app also includes visual insights and a feature to download results as a PDF.

This end-to-end project brought together data preprocessing, trend analysis, clustering, NLP, and deployment into a real-time, user-friendly job role recommender.

by Debasis Baidya



Task 1: EDA on Upwork Job Data

1

Dataset Overview

- ❑ Worked with **244,828 job postings** from Upwork (Feb–Mar 2024)
- ❑ Cleaned missing data in fields like **budget** and **hourly rate**
- ❑ Converted **published_date** to datetime format for trend analysis

2

Skill Keyword Extraction

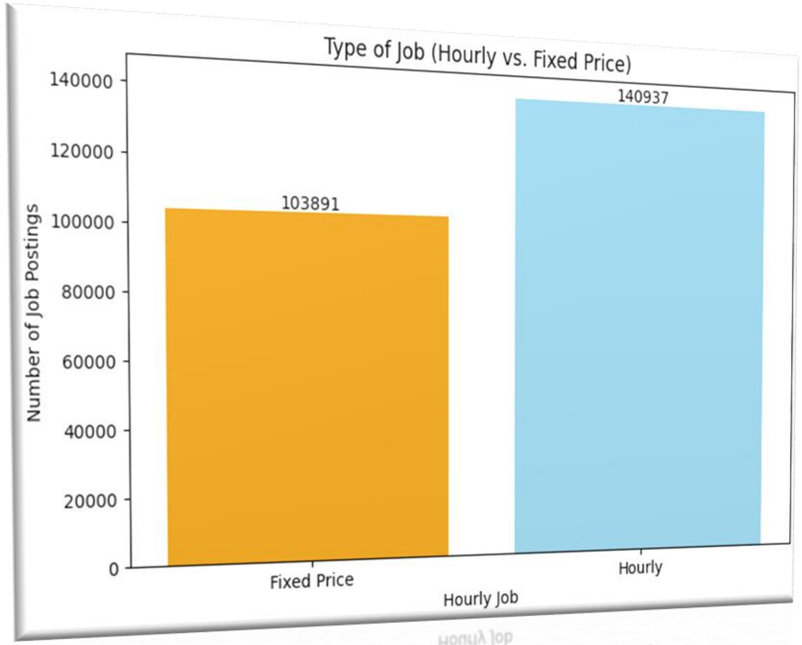
- ❑ Extracted keywords from **job titles** to find trending skills
- ❑ Found repeated terms mostly in **tech, writing & digital marketing**
- ❑ Currently rule-based; can improve with **NLP or embeddings**

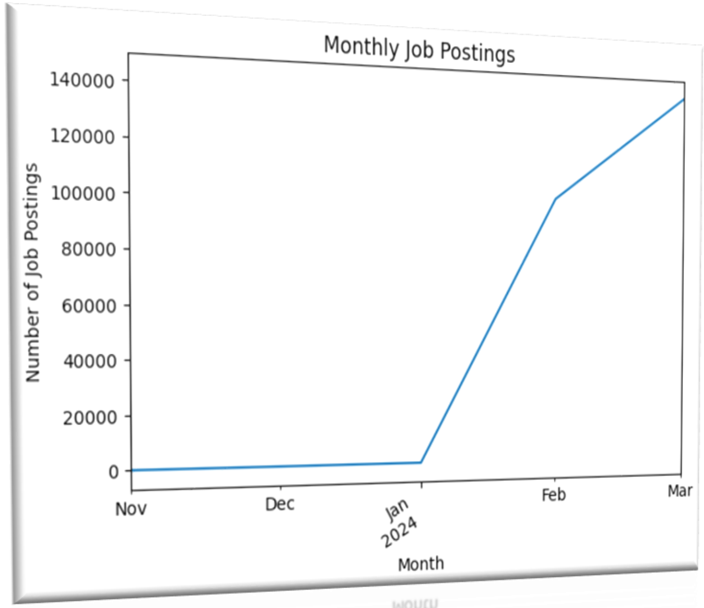
3

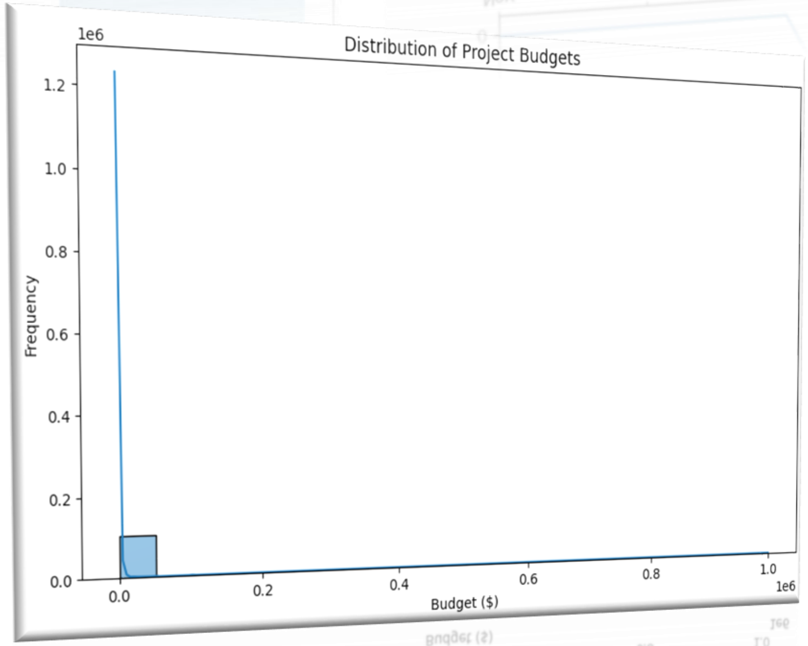
Top Job-Posting Countries

- ❑ Grouped jobs by country to find where most roles come from
- ❑ **Top 5 Countries:**
  - US USA – 99,834 jobs
  - GB UK – 19,149
  - IN India – 15,835
  - AU Australia – 12,618
  - CA Canada – 11,658
- ❑ US USA clearly dominates the platform

Task 1: Key Insights & Early Prototype







1 ⏳ Hourly vs Fixed Price Jobs

- ❑ Compared job types based on **payment model**
- ❑ ✅ **Fixed-price jobs** were slightly more common than hourly
- ❑ Shows clients prefer **pay-per-project** structure

2 💸 Budget Distribution

- ❑ Most job budgets are **low to moderate**
- ❑ A few **very high budgets** created skewed visuals
- ❑ Plan to apply **outlier caps** for better clarity next time

3 📅 Monthly Posting Trend

- ❑ Grouped job posts by **month** to see job flow
- ❑ Created line charts to show **posting volume changes**
- ❑ Data is limited to 2 months, so trend scope is small

4 🤝 Basic Job Recommender (Prototype)

- ❑ Built a simple system to **match skills to jobs** by title
- ❑ Example: Input "Data Engineer" → returns all matching job titles
- ❑ Early version worked well; can be improved using **NLP techniques**

📊 Task 1 – Job Titles & Budget Analysis

01 ✅ Missing Value Cleanup

- ❖ Dropped rows with missing title or link — critical fields
- ❖ Filled missing hourly_low, hourly_high, and budget with 0💰
- ❖ Replaced blank country entries with 'Other' 🌍
- ❖ ➡️ Result: No missing values left in the dataset! 😌

02 🔍 Keyword Extraction from Titles

- ❖ Cleaned job titles using lemmatization and stopword removal
- ❖ Extracted keywords and calculated frequency using NLTK + Python 🧠
- ❖ Merged similar terms (e.g. "need" + "needed", "design" + "designer") for cleaner insights

- ❖ 📌 Top Keywords (Post-cleaning):
  - need – 28K+
  - design – 26K+
  - website, developer, expert, video, etc.

03 🌐 Word Cloud Snapshot

Built a Word Cloud to visualize the most frequent job-related terms
➡️ Offered a quick view of in-demand freelancing roles 💼💡

📌 **Check and handle missing values**

```python
# Checking missing values BEFORE cleaning:
print("\nChecking missing values BEFORE cleaning:")
missing_before = df.isnull().sum()
print(missing_before[missing_before > 0].sort_values(ascending=False))

# Drop rows where 'title' or 'link' is missing, as these are essential
df = df.dropna(subset=['title', 'link'])

# Fill missing values in 'hourly_high', 'hourly_low', and 'budget' with 0
df['hourly_high'].fillna(0, inplace=True)
df['hourly_low'].fillna(0, inplace=True)
df['budget'].fillna(0, inplace=True)

# Fill missing 'country' with 'Other'
df['country'].fillna('Other', inplace=True)

# Checking missing values AFTER cleaning:
print("\nChecking missing values AFTER cleaning:")
missing_after = df.isnull().sum()
print(missing_after.sort_values(ascending=False))
```

```
Checking missing values BEFORE cleaning:
hourly_high    146053
hourly_low     142406
budget         140937
country          5077
link                1
title               1
dtype: int64

Checking missing values AFTER cleaning:
title             0
link              0
published_date    0
is_hourly         0
hourly_low        0
hourly_high       0
budget            0
country           0
dtype: int64
```

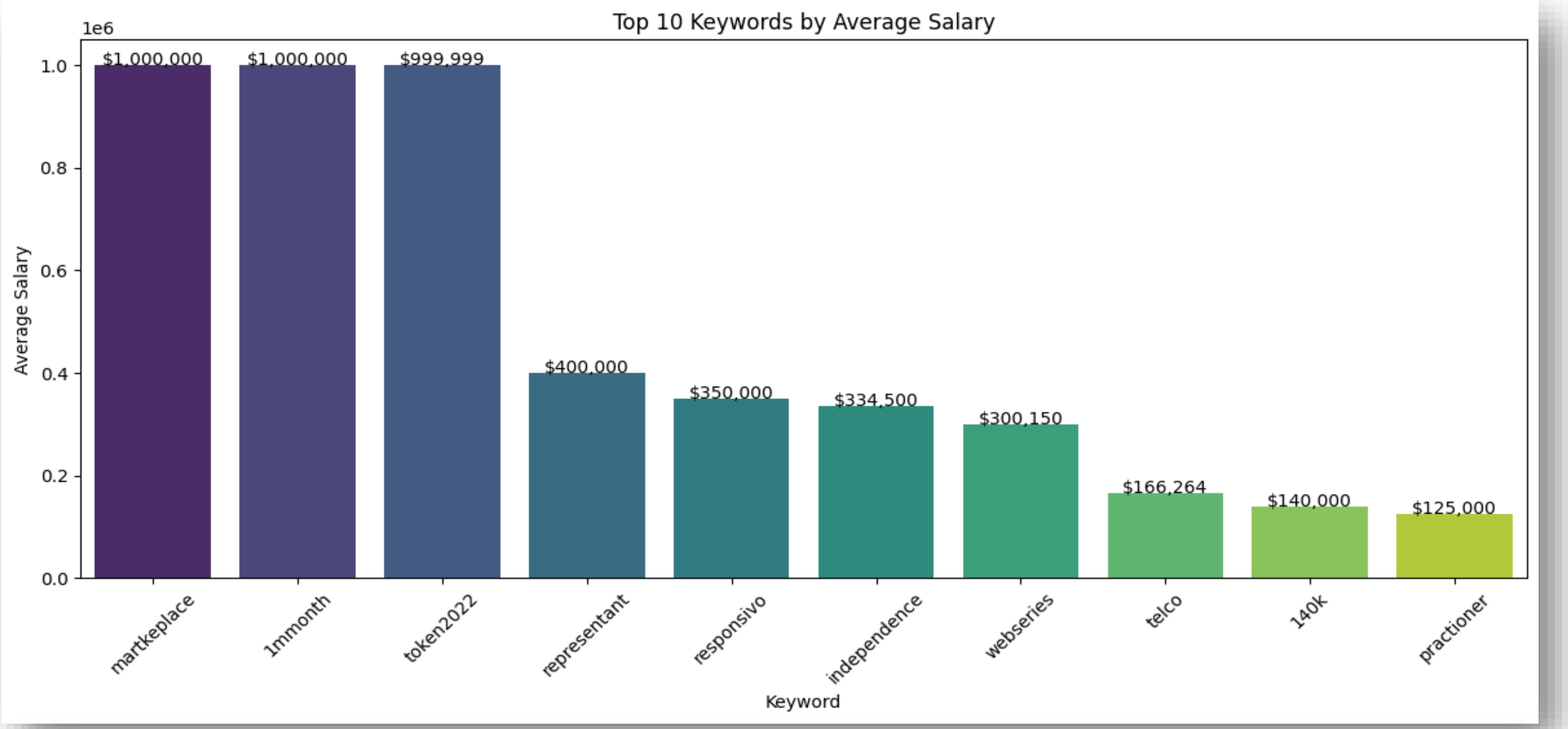
```python
# Lemmatizing and Extracting Keywords from Job Titles
lemmatizer = WordNetLemmatizer()  # Initialize the lemmatizer

# Function to lemmatize a single keyword
def lemmatize_keyword(keyword):
    return lemmatizer.lemmatize(keyword)

# Function to extract and lemmatize keywords from a job title
def extract_keywords(title):
    # Lowercase the title and remove punctuation
    title = re.sub(r'[^\w\s]', '', title.lower())
    # Tokenize the cleaned title into words
    tokens = word_tokenize(title)
    # Define English stop words to exclude common words
    stop_words = set(stopwords.words('english'))
    # Lemmatize tokens and filter out stop words
    keywords = [lemmatize_keyword(token) for token in tokens if token not in stop_words]
    return keywords

# Extract keywords from each job title and store them in a new column
df['keywords'] = df['title'].apply(extract_keywords)

# Flatten the list of keyword lists into a single list of keywords
keywords_flat = [keyword for sublist in df['keywords'] for keyword in sublist]

# Count how often each keyword appears
keyword_counts = Counter(keywords_flat)

# Get the top 10 most common keywords and their frequencies
top_10_keywords = keyword_counts.most_common(10)

# Convert the top keywords into a DataFrame for easier viewing
top_10_keywords_df = pd.DataFrame(top_10_keywords, columns=['Keyword', 'Frequency'])

# Display the top 10 keywords
top_10_keywords
```

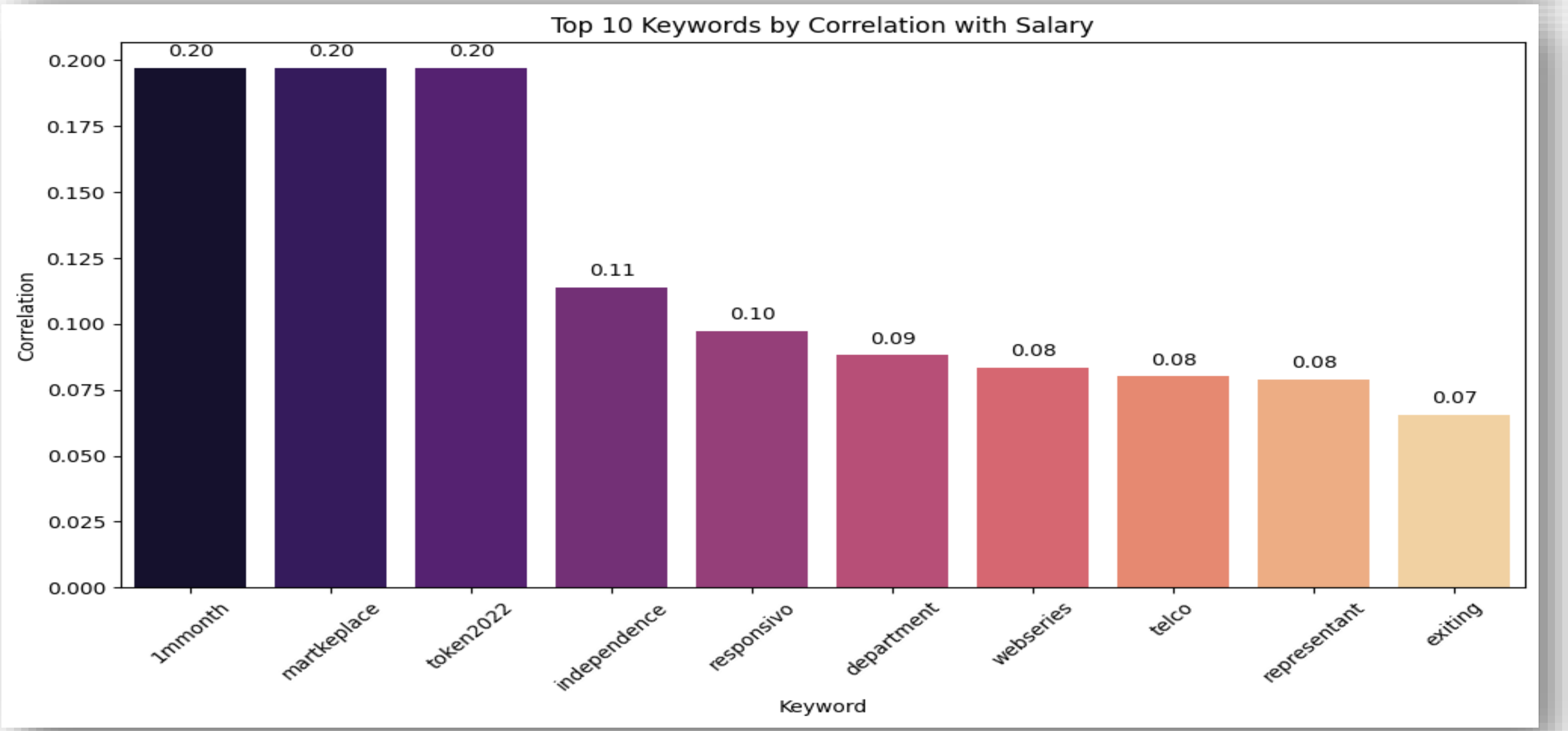
- *Lemmatizing and Extracting Keywords from Job Titles*

```python
# Import necessary NLP libraries
import re  # For text cleaning with regular expressions
import nltk  # Natural Language Toolkit for NLP tasks

# Download required NLTK datasets
nltk.download('punkt')       # Tokenizer models
nltk.download('punkt_tab')
nltk.download('stopwords')   # Common stopwords list
nltk.download('wordnet')     # Lexical database for lemmatization
nltk.download('omw-1.4')     # Additional WordNet data

from collections import Counter  # For counting word frequencies
from nltk.corpus import stopwords  # Stopwords to filter out common words
from nltk.tokenize import word_tokenize  # To split text into tokens
from nltk.stem import WordNetLemmatizer  # To lemmatize words to base forms
from collections import defaultdict  # Dictionary with default value type
```

Word Cloud of Job Title Keywords

Task 1 – Salary Insights from Keywords





01

💵 Top Keywords by Avg. Salary

- ❑ Calculated average **budget per keyword**
- ❑ Found that **highest-paying keywords ≠ most frequent**
  - ➡ Helped spot **niche roles** with **higher value** 🔍💰

02

📈 Keyword-Salary Correlation

- ❑ Used **MultiLabelBinarizer** to create binary matrix for keywords
- ❑ Applied **Pearson correlation** between keyword presence & budget
- ❑ Found keywords with **strong positive salary correlation**
  - ➡ Indicates potential **premium skills** worth focusing on 🧠💼

03

🧠 Key Takeaways

- ❑ Keywords like **"developer"**, **"design"**, and **"expert"** are popular
- ❑ But **niche keywords** (less frequent) often pay more
  - ➡ Shows **low-supply, high-demand** opportunities in the freelance space

Task 2 – Identifying Emerging Job Categories

Classifying Job Titles into Categories Based on Keywords

```python
def categorize_job(title):
    # Loop through each category and its associated keywords
    for category, keywords in categories.items():
        # Check if any keyword is present in the job title (case-insensitive)
        if any(keyword in title.lower() for keyword in keywords):
            return category  # Return the matching category
    return 'Other'  # If no keywords match, assign category as 'Other'

# Apply the categorize_job function to the 'title' column
df['category'] = df['title'].apply(categorize_job)

# Print the first 5 job titles with their assigned categories to check results
df[['title', 'category']].head(5)
```

	title	category
0	Experienced Media Buyer For Solar Pannel and R...	Other
1	Full Stack Developer	Software Development
2	SMMA Bubble App	Other
3	Talent Hunter Specialized in Marketing	Marketing
4	Data Engineer	Engineering

Calculate growth rate and identify emerging categories & Sort categories by growth rate to identify emerging categories

```python
# Calculate month-over-month percentage change in job postings per category
growth_rate = category_trends.pct_change().fillna(0)

# Calculate the average growth rate for each category over the entire period
average_growth = growth_rate.mean()

# Sort categories by their average growth rate in descending order
emerging_categories = average_growth.sort_values(ascending=False)

# Print the sorted categories with their average growth rates
print("Categories sorted by average monthly growth rate:")
print(emerging_categories)
```

```
Categories sorted by average monthly growth rate:
category
Administration               inf
Construction                 inf
Consulting                   inf
Customer Service             inf
Data Science                 inf
Design                       inf
Education                    inf
Engineering                  inf
Finance                      inf
Healthcare                   inf
Hospitality                  inf
Human Resources              inf
IT Support                   inf
Legal                        inf
Manufacturing                inf
Marketing                    inf
Operations                   inf
Product Management           inf
Retail                       inf
Project Management           inf
Real Estate                  inf
Research                     inf
Software Development         inf
Sales                        inf
Writing and Editing          inf
Other                 261.991489
dtype: float64
```

01 Data Cleanup & Setup

- ❑ Converted published_date to datetime
- ❑ Created a new year_month column for monthly trend analysis

02 Job Categorization

- ❑ Mapped job titles into categories like:
  - Data Science, Design, Consulting, Healthcare, etc.
- ❑ Used custom keyword matching
- ❑ Unmatched titles were labeled as 'Other'

03 Monthly Job Posting Trends

- ❑ Used .groupby() + .unstack() to reshape data for time-series analysis
- ❑ Tracked job volume by category across months

04 Growth Rate Calculation

- ❑ Calculated month-over-month % change in job counts per category
- ❑ Averaged growth to identify fastest-growing categories

🔥 Key Insights & Emerging Categories

STEP 01

🌟 Top Emerging Categories (by Avg. Growth)

1 Administration | 2 Construction | 3 Consulting → inf ⚠️
4 Customer Service | 5 Data Science → inf ⚠️
⚠️ inf = No jobs earlier, new now → indicates strong upward trend

STEP 02

🖼️ Visualization

- ❑ Line chart created to track top 5 growing categories
- ❑ Showed fields like Data Science and Consulting rising steadily

STEP 03

📉 Caveats

- ❑ Some spikes due to new categories appearing for the first time
- ❑ Not all indicate consistent demand yet - but useful early indicators 🔍

STEP 04

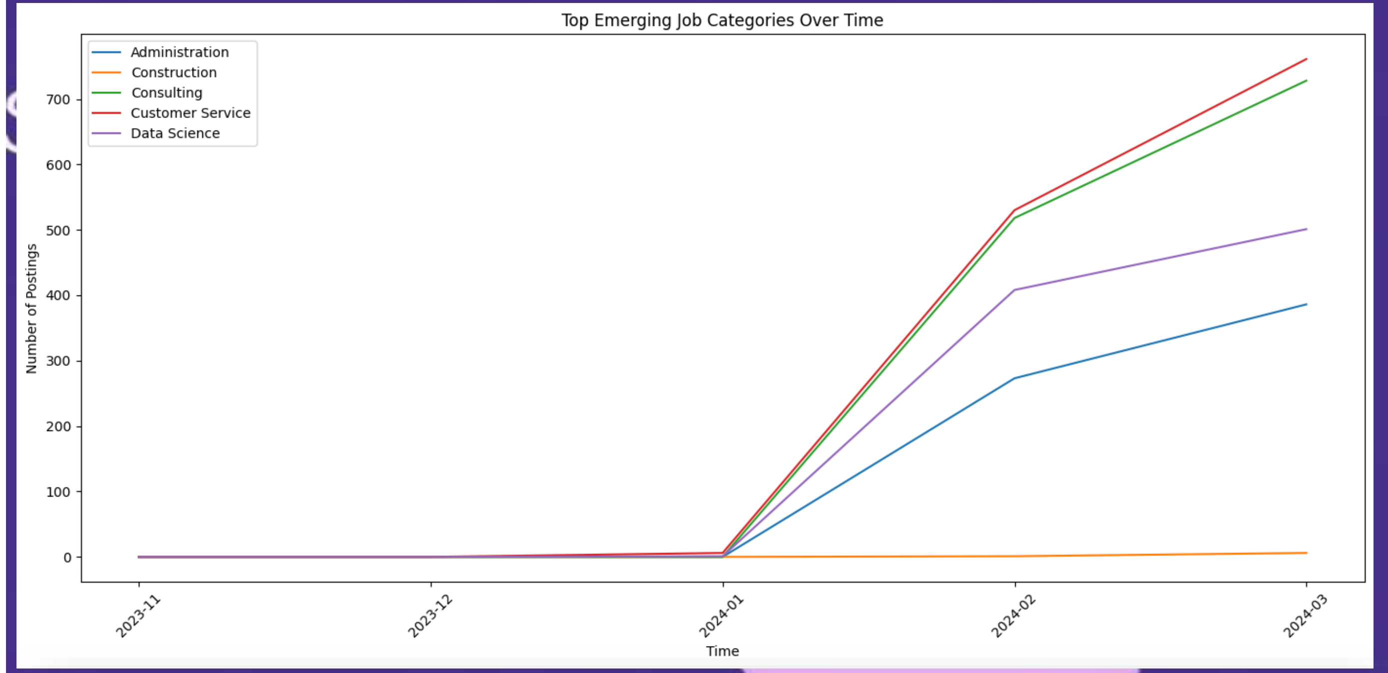
💾 Final Step

- ❑ Saved the cleaned and categorized dataset for future use
  - 📁 /job_data.csv

STEP 05

✅ Summary

- ❑ Discovered emerging job categories by tracking monthly trends
- ❑ Fields like Data Science, Customer Service, Consulting show high momentum
- ❑ Useful for platforms and recruiters to focus on upcoming demand areas 🔮



Tracking Top Emerging Job Categories Over Time

```python
import matplotlib.pyplot as plt

# Set figure size for better visibility
plt.figure(figsize=(14, 7))

# Loop through the top 5 emerging categories based on average growth rate
for category in emerging_categories.index[:5]:
    # Plot the number of postings over time for each category
    plt.plot(category_trends.index.astype(str), category_trends[category], label=category)

# Add title and axis labels
plt.title('Top Emerging Job Categories Over Time')
plt.xlabel('Time')
plt.ylabel('Number of Postings')

# Add legend to identify categories
plt.legend()

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Adjust layout to prevent clipping of labels
plt.tight_layout()

# Display the plot
plt.show()

# Print the top 10 emerging categories with their average growth rates
print("Emerging Job Categories with Average Growth Rates:")
print(emerging_categories.head(10))
```

Saving Job data to CSV file

```python
# folder to save file
folder_path = '/content/drive/MyDrive/Job Market Analysis & Recommendation System'

# csv file name
file_name = 'job_data.csv'

# Combine folder path and file name to create full file path
csv_filepath = f"{folder_path}/{file_name}"

# Save DataFrame to the specified folder with the given file name
df.to_csv(csv_filepath, index=False)

print(f"Data saved as {csv_filepath} with Filename: {file_name}")
```

```
Data saved as /content/drive/MyDrive/Job Market Analysis & Recommendation System/job_data.csv with Filename: job_data.csv
```

```
Emerging Job Categories with Average Growth Rates:
category
Administration      inf
Construction        inf
Consulting          inf
Customer Service    inf
Data Science        inf
Design              inf
Education           inf
Engineering         inf
Finance             inf
Healthcare          inf
dtype: float64
```

Task 3: Predicting High-Demand Job Roles | Forecasting & Insights

1

🛠️ Data Preparation

- ❑ Extracted key fields: **title, category, month**
- ❑ Created a **target variable** for monthly job demand
- ❑ Applied **scaling, encoding & dimensionality reduction** for model training

2

🧩 Modeling

- ❑ Trained a **Gradient Boosting Regressor**
- ❑ Used **MAE, MSE, and R²** for evaluation
  - ⚠️ Model performance was low due to **data sparsity & noise**

3

🦋 Trend Analysis

- ❑ Analyzed job posting changes month-by-month
- ❑ Identified **Top 10 Emerging Job Titles** based on growth:

✅ Social Media Manager | ✅ Video Editor | ✅ Logo Design | ✅ Graphic Designer
✅ Virtual Assistant | ✅ Content Writer | ✅ YouTube Manager | ✅ Personal Assistant
✅ Web Designer | ✅ Bookkeeper

4

📉 Time Series Forecasting

- ❑ Applied **exponential smoothing** to predict future trends
- ❑ Generated 12-month forecasts for each category
  - 📌 Example: **Data Science** shows steady upward trend
- ❑ Plotted forecasts to visualize **emerging or seasonal demand**

5

🔍 Key Insights

🧑💻 Creative + remote-friendly roles like
→ **Social Media Manager** and **Virtual Assistant** are rising fast
📄 Steady demand remains for
→ **Web Designers, Content Writers,** and **Bookkeepers**
📊 Forecasts give job seekers and platforms
→ A clear view of **what's trending and where to focus** 🔮

Define Features, Target, and Data Preprocessing Pipeline

```python
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Create 'demand' target by counting job titles per 'year_month'
job_data['demand'] = job_data.groupby('year_month')['title'].transform('count')

# Define features by dropping columns not used as inputs
features = job_data.drop(columns=['demand', 'published_date', 'year_month', 'link'])

# Define target variable
target = job_data['demand']

# Specify numerical and categorical feature columns
numerical_features = ['hourly_low', 'hourly_high', 'budget']
categorical_features = ['title', 'country', 'keywords', 'category', 'is_hourly']

# Define transformers for numerical and categorical features
numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore', sparse_output=True)

# Create ColumnTransformer to apply transformations appropriately
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ],
    sparse_threshold=0.3 # output sparse matrix if >30% zeros
)

print("Preprocessing pipeline created successfully.")
```

```
Preprocessing pipeline created successfully.
```

Gradient Boosting Regressor

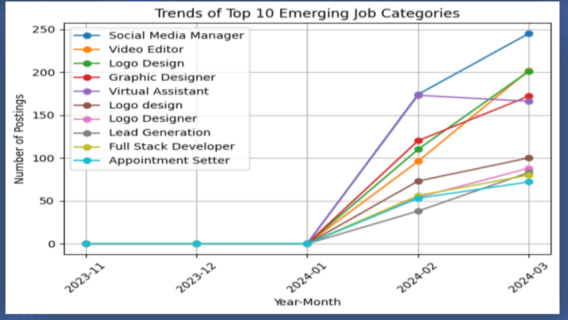
```python
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

# Initialize the Gradient Boosting Regressor
gb_model = GradientBoostingRegressor(n_estimators=100, random_state=42)

# Train the model on the training data
gb_model.fit(X_train, y_train)

# Predict on the test set
y_pred_gb = gb_model.predict(X_test)

# Evaluate the model using Mean Absolute Error
mae_gb = mean_absolute_error(y_test, y_pred_gb)
print(f'Mean Absolute Error with Gradient Boosting: {mae_gb}')
```

```
Mean Absolute Error with Gradient Boosting: 19814.832581437848
```

```
title
Social Media Manager    61.25
Video Editor            50.50
Logo Design             50.25
Graphic Designer        43.00
Virtual Assistant       41.50
Logo design             25.00
Logo Designer           22.00
Lead Generation         20.75
Full Stack Developer    20.00
Appointment Setter      18.00
dtype: float64
```

Prepare Time Series Data for Job Postings by Category

```python
import pandas as pd

# Convert the 'year_month' column to datetime
job_data['year_month'] = pd.to_datetime(job_data['year_month'])

# Group by date and category, and count the number of postings
time_series_data = job_data.groupby(['year_month', 'category']).size().unstack().fillna(0)

# Display the prepared data
print(time_series_data.head())
```



🌍 Task 4 – Exploring Hourly Rate Differences Across Countries

01 Objective

To explore how **freelance hourly rates vary** across countries & identify **high-paying regions** 💰🌎

02 What I Did

- ❑ Grouped jobs by **country** and calculated average of hourly_high
- ❑ Built a **choropleth map** using Plotly Express to visualize global pay trends
- ❑ Used **Viridis colour scale** for contrast and customized layout for clarity 🌈
- ❑ Created a description field combining **job title + country** for future keyword-based tasks

03 Key Findings

🌟 Countries like **Vietnam** and **Philippines** showed **surprisingly high average rates**
🟣 Lower rates were common in **India, Africa, and South America**
🤖 Helped highlight **regional pay disparities** and where to target high-paying remote work
📍 Interactive map made it easy to explore hourly rates country-wise by hovering

Average Hourly Rates by Country

```python
import plotly.express as px

# Calculate average hourly rates by country
avg_hourly_rates = job_data.groupby('country')['hourly_high'].mean().reset_index()

# Plot choropleth map
fig = px.choropleth(
    avg_hourly_rates,
    locations='country',
    locationmode='country names',
    color='hourly_high',
    hover_name='country',
    title='Average Hourly Rates by Country (Values can be seen on hover)',
    color_continuous_scale='Viridis'
)

# Increase figure size to better fit screen width
fig.update_layout(
    width=1000,
    height=700
)

fig.show()
```


```python
# Show full column content
pd.set_option('display.max_colwidth', None)

# Create a new 'description' column by concatenating 'title' and 'country' with a space in between
job_data['description'] = job_data['title'] + ' ' + job_data['country']

# Print the first 5 rows to verify the new column
job_data[['title', 'country', 'description']].head()
```

	title	country	description
0	Experienced Media Buyer For Solar Pannel and Roofing installation companies.	Other	Experienced Media Buyer For Solar Pannel and Roofing installation companies. Other
1	Full Stack Developer	United States	Full Stack Developer United States
2	SMMA Bubble App	United States	SMMA Bubble App United States
3	Talent Hunter Specialized in Marketing	United States	Talent Hunter Specialized in Marketing United States
4	Data Engineer	India	Data Engineer India

Task 5 – Job Similarity Recommendation Using TF-IDF & Cosine Similarity

Extracting Features from Job Descriptions Using TF-IDF Vectorization

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TF-IDF Vectorizer with English stop words removed
vectorizer = TfidfVectorizer(stop_words='english')

# Fit the vectorizer on the 'description' column and transform the text data into TF-IDF features
X = vectorizer.fit_transform(job_data['description'])

# X is a sparse matrix of shape (number_of_samples, number_of_features)
print(f"TF-IDF matrix shape: {X.shape}")

# Optionally, get the feature names (words)
feature_names = vectorizer.get_feature_names_out()
print(f"Number of unique features: {len(feature_names)}")
```

```
TF-IDF matrix shape: (244827, 38928)
Number of unique features: 38928
```

Building a Nearest Neighbors Model with Cosine Similarity for Job Descriptions

```
[ ]
from sklearn.neighbors import NearestNeighbors

# Initialize Nearest Neighbors model with cosine distance and brute-force algorithm
model = NearestNeighbors(metric='cosine', algorithm='brute')

# Fit the model on the TF-IDF feature matrix
model.fit(X)
```

```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine')
```

```
# Find the 5 nearest neighbors for the first job description
distances, indices = model.kneighbors(X[0], n_neighbors=5)

# Print formatted output
print("Top similar job descriptions to the first job:\n")

for rank, (idx, dist) in enumerate(zip(indices[0], distances[0]), start=1):
    similarity = 1 - dist  # cosine similarity = 1 - cosine distance
    description = job_data.iloc[idx]['description']
    print(f"{rank}. Index: {idx}, Similarity: {similarity:.3f}")
    print(f"   Description: {description}\n")
```

```
Top similar job descriptions to the first job:

1. Index: 0, Similarity: 1.000
   Description: Experienced Media Buyer For Solar Pannel and Roofing installation companies. Other

2. Index: 221025, Similarity: 0.545
   Description: Experienced Facebook and YouTube Ads Media Buyer for Roofing companies Morocco

3. Index: 190320, Similarity: 0.523
   Description: Database of Solar Installation companies in US United Kingdom

4. Index: 160365, Similarity: 0.514
   Description: Media Buyer Needed to generate leads for Solar pannel company (Google + Meta Ads) Belgium

5. Index: 114504, Similarity: 0.511
   Description: Expert Media Buyer For Roofing United States
```

🖌️ Text Vectorization Using TF-IDF

- Cleaned job descriptions and applied **TF-IDF** to highlight important terms
- Final matrix: **244,827 jobs × 38,928 features** — showcasing rich textual variety
- Removed common filler words (e.g., "the", "and") to focus on meaningful terms

🔍 Similarity Matching with Nearest Neighbors

- Used **cosine similarity** with Nearest Neighbors to compare job descriptions
- Retrieved **top 5 most similar jobs** for any given listing
- **Lower cosine distance = higher similarity**
- Added a custom description field **(title + country)** for improved context

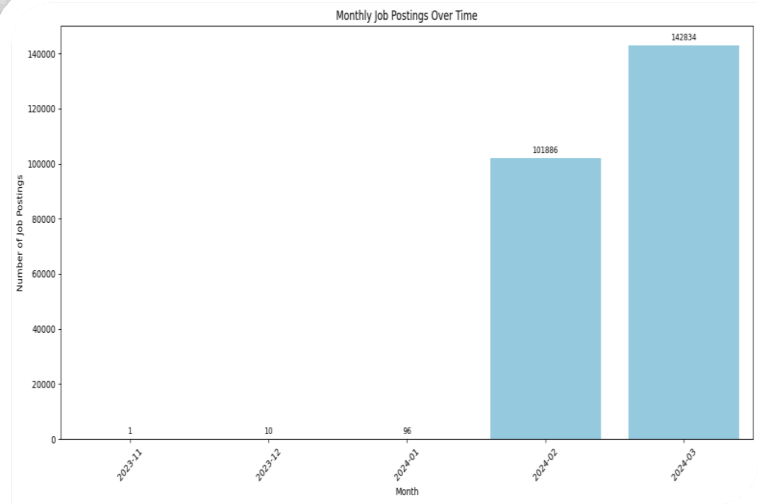
💾 Model Saving for Reuse

- Saved both **TF-IDF vectorizer** and **Nearest Neighbors model** using pickle
- Enabled **reusability** without retraining — ideal for deployment or integration

📊 Performance & Output Insights

- ✅ Recommendations were **accurate and relevant** — cosine scores around **0.51–0.54**
- Similar jobs were found even across **different countries** with contextual overlap
- ⚡ Model is **fast, scalable,** and **easy to interpret**, even with **200K+ jobs**
- 📍 Extracted full **country list** to support **region-specific recommendations**

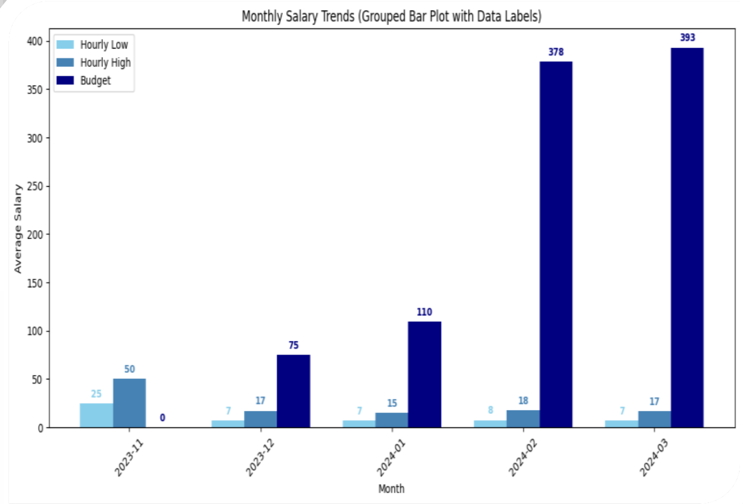
Task 6 – Tracking Monthly Changes in Job Market Demand



Monthly Trends in Job Postings, Hourly Rates, and Budgets

```python
# Group by 'year_month' to get the count of job postings per month
monthly_job_postings = job_data1.groupby('year_month').size().reset_index(name='job_postings')

# Calculate average hourly_low, hourly_high, and budget per month
monthly_salary_trends = job_data1.groupby('year_month').agg({
    'hourly_low': 'mean',
    'hourly_high': 'mean',
    'budget': 'mean'
}).reset_index()

# Merge the counts and average salary/budget trends into one DataFrame
monthly_trends = pd.merge(monthly_job_postings, monthly_salary_trends, on='year_month')

# Display the first few rows of the combined monthly trends
print(monthly_trends.head())
```

	year_month	job_postings	hourly_low	hourly_high	budget
0	2023-11	1	25.000000	50.000000	0.000000
1	2023-12	10	6.900000	17.200000	75.000000
2	2024-01	96	6.739583	15.083333	109.656250
3	2024-02	101886	7.523045	17.520601	378.296832
4	2024-03	142834	7.062233	16.514786	393.039976



Tracking Monthly Job Postings Over Time

```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(14, 7))
barplot = sns.barplot(data=monthly_trends, x='year_month', y='job_postings', color='skyblue')

# Add data labels on top of each bar
for p in barplot.patches:
    height = p.get_height()
    barplot.annotate(f'{int(height)}',
                     (p.get_x() + p.get_width() / 2, height),
                     ha='center', va='bottom',
                     fontsize=9, color='black', xytext=(0, 3),
                     textcoords='offset points')

plt.title('Monthly Job Postings Over Time')
plt.xlabel('Month')
plt.ylabel('Number of Job Postings')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

01

Most In-Demand Job Roles

- Frequently posted roles include:
  - **Data Scientist** | **Data Analyst** | **ML Engineer** | **Business Analyst**
- Highlights a strong industry focus on **data-driven roles**

02

Overall Growth in Job Postings

- Observed a **steady rise in job volume** month-over-month
- Suggests rising demand for skilled professionals
- Driven by **digital adoption**, **AI/ML integration** & **data-first strategies**

03

High Demand for Technical Skills

- Most roles require:
  - **Python / R** | **Machine Learning** | **SQL** | **Power BI / Tableau**
- Shows strong demand for **analytical** & **programming expertise.**

04

Seasonal Fluctuations Noted

- Despite the growth, **up-and-down patterns** appear over months
- Likely caused by:
  - **Quarterly hiring cycles** | **Budget approvals** | **Project kick-offs**

05

Guidance for Job Seekers To Focus On

In-demand roles & tools

Staying updated on job market trends

Practicing real-world tech skills

🏠 Task 7 – Remote Work Trends Analysis Using Rolling Average & Forecasting

Monthly Analysis of Remote Job Postings Based on Keywords

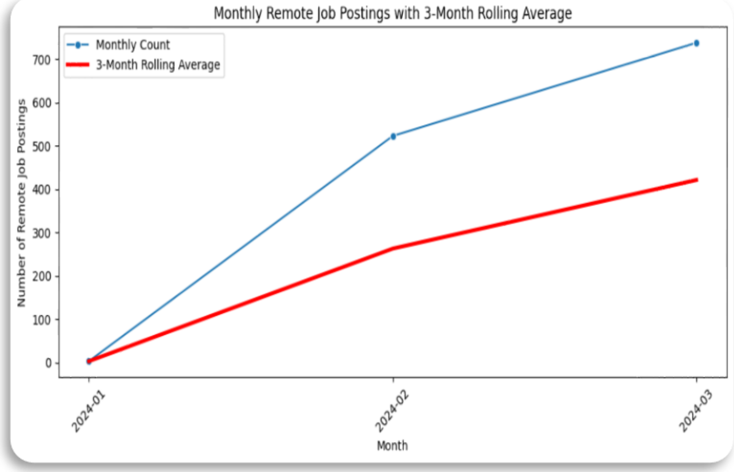
```python
# Fill missing values in 'keywords' column with empty strings
job_data1['keywords'] = job_data1['keywords'].fillna('')

# Filter job postings that mention 'remote' in the keywords (case-insensitive)
remote_jobs = job_data1[job_data1['keywords'].str.contains('remote', case=False, na=False)]

# Group by 'year_month' and count the number of remote job postings per month
monthly_remote_jobs = remote_jobs.groupby('year_month').size().reset_index(name='remote_job_postings')

# Convert 'year_month' to string type for easier plotting
monthly_remote_jobs['year_month'] = monthly_remote_jobs['year_month'].astype(str)

# Print the first few rows of the result
print(monthly_remote_jobs.head())
```

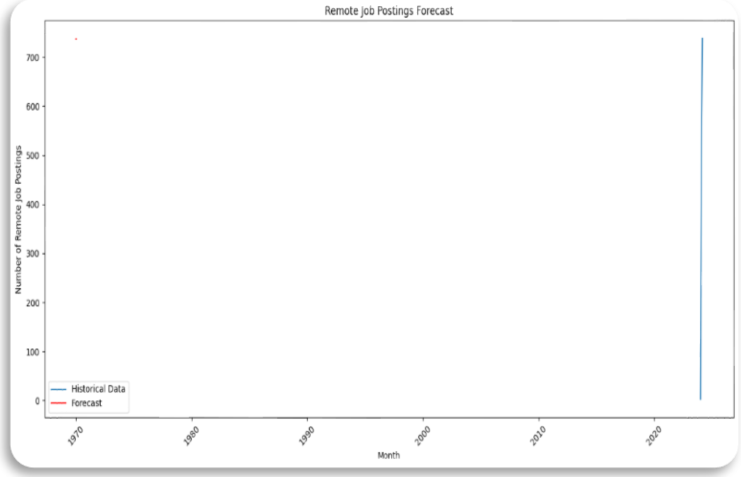
```
  year_month  remote_job_postings
0    2024-01                    3
1    2024-02                  523
2    2024-03                  738
```

Forecasting Remote Job Posting Trends Using Exponential Smoothing

```python
import pandas as pd
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Convert 'year_month' to datetime timestamp for modeling
monthly_remote_jobs['year_month'] = pd.PeriodIndex(monthly_remote_jobs['year_month'], freq='M').to_timestamp()

# Calculate initial seasonal values using a 12-month rolling mean
seasonal_initial = monthly_remote_jobs['remote_job_postings'].rolling(window=12, min_periods=1).mean()

# Set initial level and seasonal components for the model
initial_level = monthly_remote_jobs['remote_job_postings'].iloc[0]
initial_seasonal = seasonal_initial.iloc[0]

# Define and fit the Holt-Winters Exponential Smoothing model with additive seasonality
model = ExponentialSmoothing(
    monthly_remote_jobs['remote_job_postings'],
    seasonal='add',
    seasonal_periods=12,
    initialization_method='known',
    initial_level=initial_level,
    initial_seasonal=initial_seasonal
)
fit = model.fit()

# Forecast the next 12 months
forecast = fit.forecast(12)

# Print the forecasted values
print(forecast)
```

```
3     736.912000
4     736.912000
5     736.912000
6     736.912000
7     736.912000
8     736.912000
9     736.912000
10    736.912000
11    736.912000
12    736.911700
13    736.963999
14    736.912000
dtype: float64
```





Visualization of Monthly Trends in Remote Job Postings with 3-Month Rolling Average

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate 3-month rolling average to smooth data
monthly_remote_jobs['rolling_avg'] = monthly_remote_jobs['remote_job_postings'].rolling(window=3, min_periods=1).mean()

plt.figure(figsize=(10, 5))  # Set plot size

# Plot original monthly remote job postings
sns.lineplot(data=monthly_remote_jobs, x='year_month', y='remote_job_postings', label='Monthly Count', marker='o')

# Plot rolling average with emphasis
sns.lineplot(data=monthly_remote_jobs, x='year_month', y='rolling_avg', label='3-Month Rolling Average', color='red', linewidth=3)

plt.title('Monthly Remote Job Postings with 3-Month Rolling Average')
plt.xlabel('Month')
plt.ylabel('Number of Remote Job Postings')
plt.xticks(rotation=45)  # Rotate x-axis labels for readability
plt.legend()
plt.tight_layout()
plt.savefig('remote_job_postings_rolling_avg.png')
plt.show()
```

01

📈 Surge in Remote Job Postings

- 🚀 Remote jobs increased **from 3 in Jan to 738 in Mar 2024**
- 📢 Proves remote work is **quickly rising in popularity**

02

📊 Rolling Average Reveals Consistent Growth

- 📉 Applied a **3-month rolling average** to smooth out spikes
- 📈 Clearly shows a **steady upward momentum** in demand

03

🔮 Forecast Indicates Stable Future Demand

- 📅 Used **Exponential Smoothing** to predict next 12 months
- 🔁 Remote job postings likely to **stay above 736/month**

04

📅 Shift in Job Market Behaviour

- ⚡ Compared to early months, demand shows **rapid change**
- 🏠 Remote flexibility is now **a mainstream hiring norm**

📊 Task 8 – Analyzing Overall Job Market Trends & Forecasts

Analyze Monthly Job Postings by Year and Month

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.arima.model import ARIMA
```

```python
# Group the job data by 'year_month' to count the number of job postings per month
monthly_job_postings = job_data1.groupby('year_month').size().reset_index(name='job_postings')

# Print the first few rows to verify the grouping and counts
print("Grouped monthly job postings (first 5 rows):")
print(monthly_job_postings.head())

# Convert 'year_month' to string type for plotting or display purposes
monthly_job_postings['year_month'] = monthly_job_postings['year_month'].astype(str)

# Print data types to confirm conversion
print("\nData types after conversion:")
print(monthly_job_postings.dtypes)
```

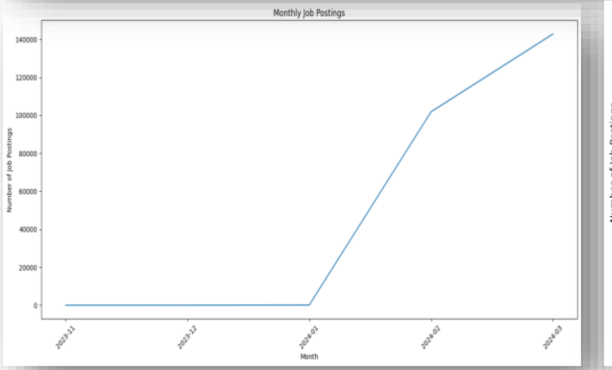
```
Grouped monthly job postings (first 5 rows):
  year_month  job_postings
0    2023-11             1
1    2023-12            10
2    2024-01            96
3    2024-02        101886
4    2024-03        142834

Data types after conversion:
year_month      object
job_postings     int64
dtype: object
```

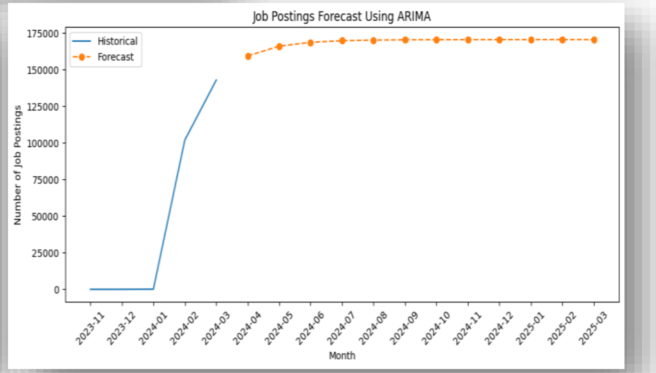
Forecast Future Job Postings Using ARIMA Model

```python
from statsmodels.tsa.arima.model import ARIMA
import pandas as pd

# Sort data by date
monthly_job_postings = monthly_job_postings.sort_values('year_month')

# Fit ARIMA model
model = ARIMA(monthly_job_postings['job_postings'], order=(1, 1, 1))
fit_model = model.fit()

# Forecast next 12 months
forecast_steps = 12
forecast = fit_model.forecast(steps=forecast_steps)

# Prepare index for forecasted months
last_date = pd.to_datetime(monthly_job_postings['year_month'].iloc[-1])
forecast_index = pd.date_range(start=last_date + pd.offsets.MonthBegin(),
                               periods=forecast_steps, freq='MS').strftime('%Y-%m')

# Combine forecasted values with their dates
forecast_df = pd.DataFrame({'year_month': forecast_index, 'forecasted_job_postings': forecast.values})

print("Forecasted job postings for next 12 months:\n")
print(forecast_df)
```

Forecasted job postings for next 12 months:

	year_month	forecasted_job_postings
0	2024-04	159319.147364
1	2024-05	165955.874347
2	2024-06	168627.742791
3	2024-07	169703.405690
4	2024-08	170136.454957
5	2024-09	170310.795513
6	2024-10	170380.982976
7	2024-11	170409.239625
8	2024-12	170420.615421
9	2025-01	170425.195183
10	2025-02	170427.038941
11	2025-03	170427.781217





🚀 Huge Spike in Q1 2024

- 📅 Job postings surged in **Feb (101K)** and **Mar (143K)**
- 💼 Indicates **high hiring demand** after a quiet quarter

📉➡️📈 Sharp Recovery from Zero Activity

- ❄️ Nov 2023 to Jan 2024 had **near-zero postings**
- 🔁 Feb 2024 brought a strong **market rebound** — likely seasonal

🌤️ Future Forecast – Growth Then Stability

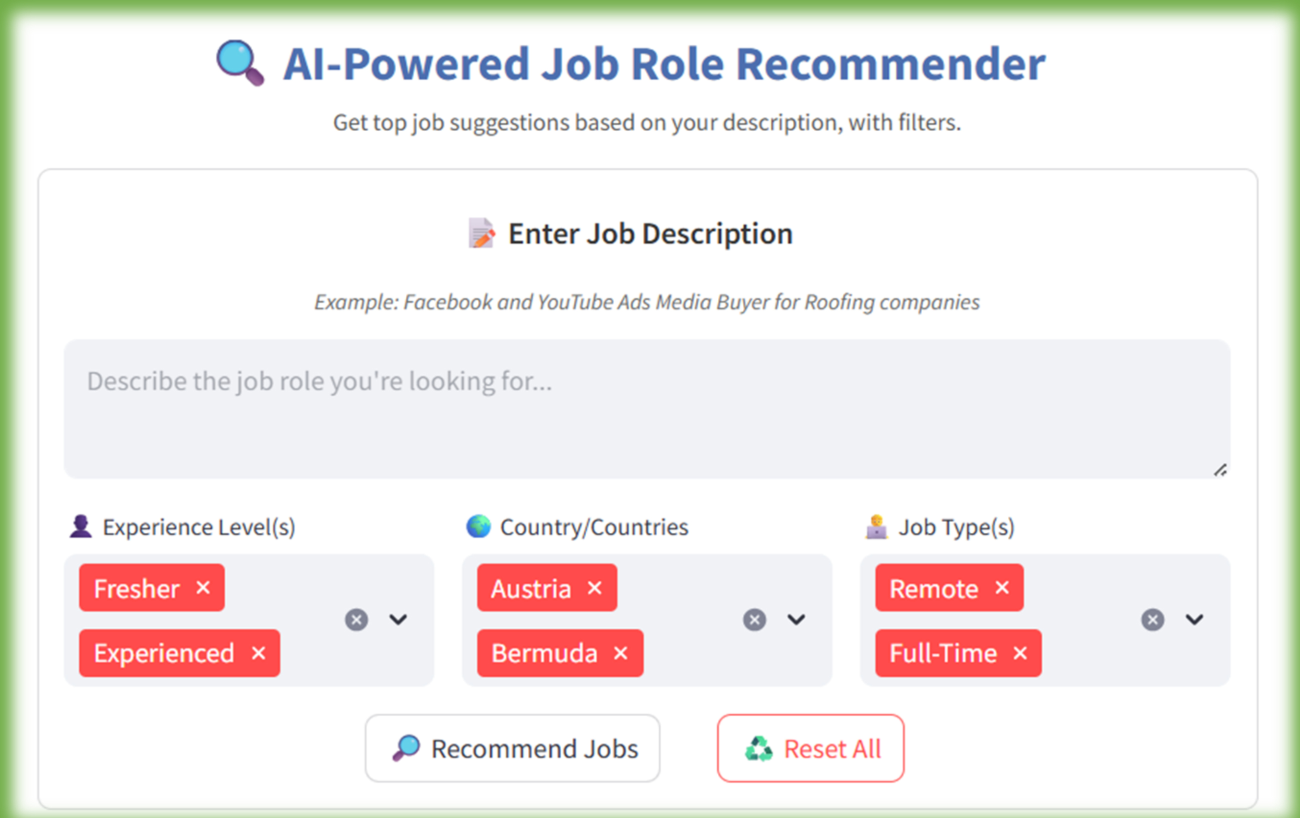
- 📈 ARIMA model shows growth until **July 2024 (~170K)**
- 📊 After that, **job volume plateaus**, showing **market stabilization**

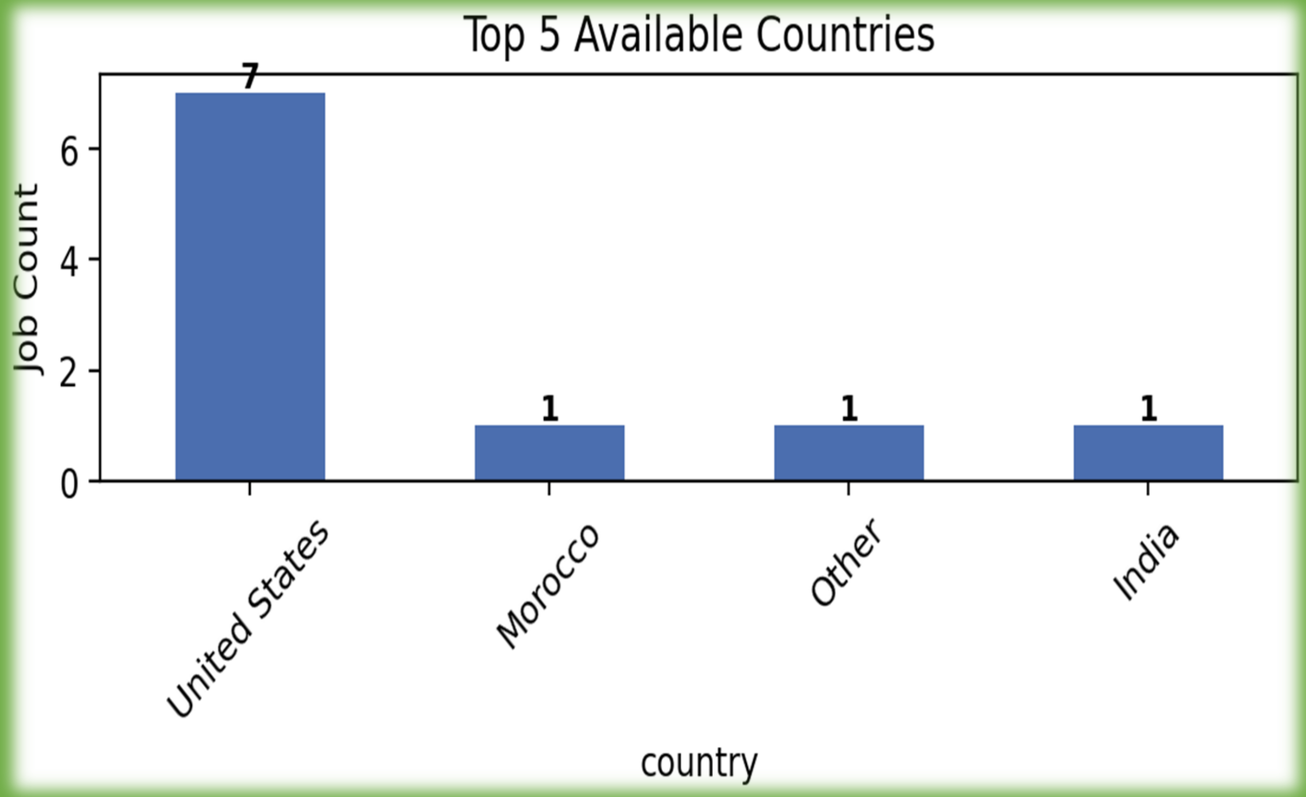
🔮 Forecast Shows High Confidence

- 📉 Forecast trend is **smooth, not volatile**
- 🧠 Suggests market is entering a **mature, stable phase**

📅 Seasonality Influences Hiring Patterns

- 📌 Spike aligns with **post-holiday budgets** & **hiring cycles**
- 🛠️ Future trend looks like **steady, strategic hiring**





Building the Streamlit-Based Job Recommendation System

🛠️ End-to-End App Functionality

- Built an **AI-powered job recommender app** using **Streamlit**
- Accepts a **user-entered job description** and returns **top matching jobs**
- Allows filters for:
    - 🧑 **Experience level** (Fresher / Experienced)
    - 🌍 **Country**
    - 🧑💻 **Job type** (Remote, On-site, Freelance, etc.)

🔍 Recommendation Logic

- 🧠 Vectorized input using a **pre-trained TF-IDF model**
- 📁 Compared against 240K+ job descriptions using **cosine similarity**
- 🔟 Returned **top 10 most similar roles** ranked by match percentage
- ✨ Used **highlighted keywords** to show why a job was recommended
- 📦 Used a **Nearest Neighbors model,** loaded via pickle for efficiency

📄 Data Handling & Preprocessing

- 🧹 Cleaned the dataset by filling missing values and creating extra fields
- 🌍 Merged **job title + country** into a single description field for better context
- 🔧 Handled missing fields like **experience, job_type** and **keywords** smoothly
- ❌ App shows an **error and stops** if the model or data isn't available

Output Features & User Experience of the Streamlit App

Live Streamlit App: App Link

All About My App: Video Link

01

Output Visualization

- Displays job matches with:
  - Highlighted titles
  - Location, Experience, Type, and Posting Date
  - Similarity score in %
- Results are styled for **readability** and **scrollable exploration**

02

Top Country Insights

- Extracted and visualized **top 5 countries** with most matching jobs
- Used **Matplotlib bar chart** for a clean and clear overview

03

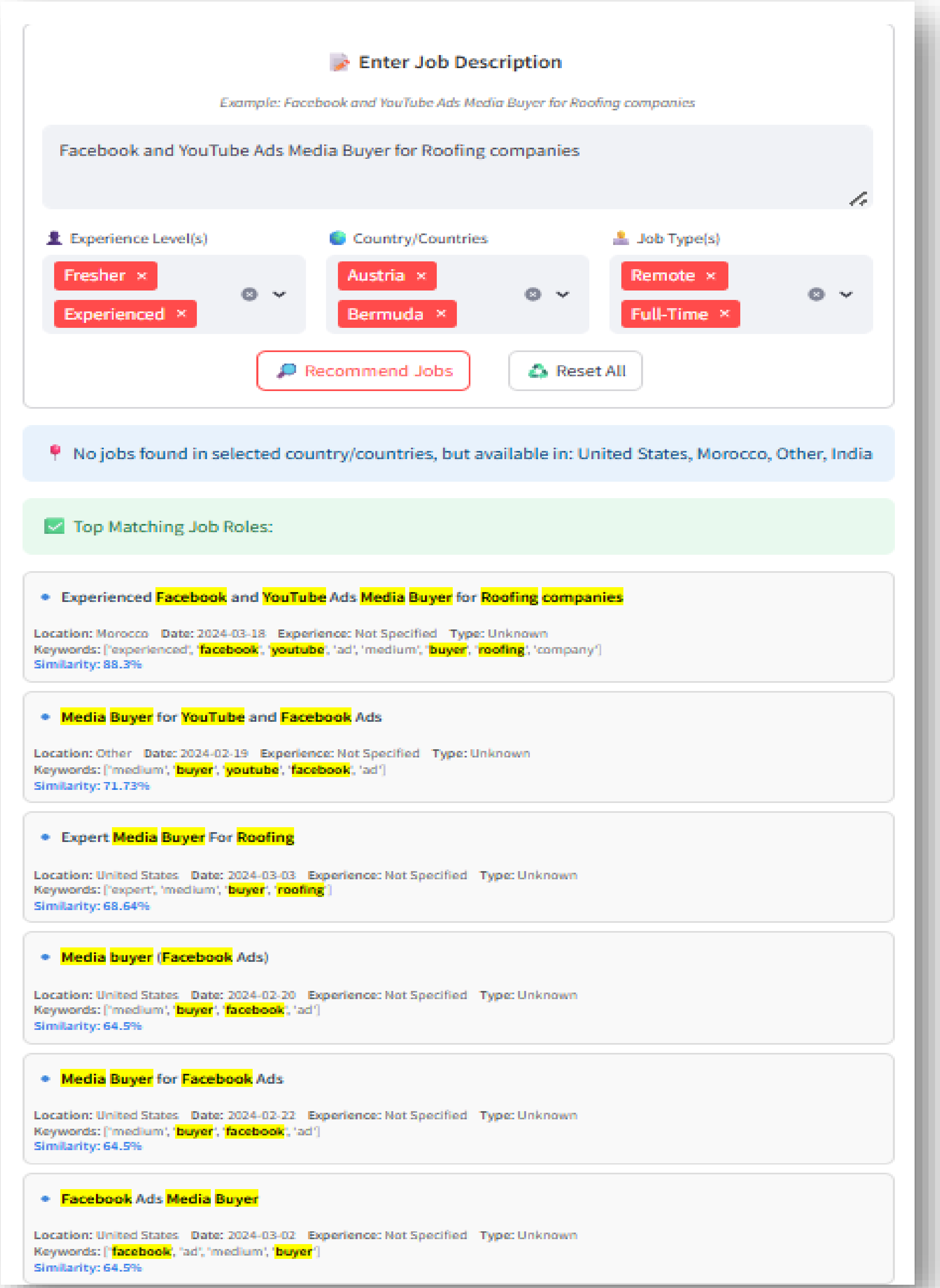
PDF Report Generation

- Added a feature to **download results as a PDF**:
- Job title, location, experience, job type
- Keywords and similarity %
- Styled with **FPDF** for clear formatting and readable output

04

Final Highlights

- Real-time filtering, smart recommendations, and export-ready results
- Fast, interpretable, and scalable solution for large job datasets
- All resources (model, vectorizer, dataset) loaded efficiently using caching
- Offers a **complete job search experience** within a single app



Thank You!

Hope You find it helpful