Project Title: JOB MARKET ANALYSIS A Web Mining Approach

TABLE OF CONTENTS

S.No Contents Page no

1. Introduction……3

- Abstract

2. Dataset Description……4

- Data Source
- Dataset Overview
- Key Attributes

3. Type of Analysis……..5

- Descriptive Statistics
- Data Cleaning
- Exploratory Data Analysis (EDA)
- Data Visualization
- Statistical Analysis
- Text Analysis

4. Technologies……...16

- Software Used

5. Project Implementation…….……...17

- Data Preprocessing
  Handling Missing Values
- Descriptive Statistics
  Mean and Medium Salary Calculation
  Standard Deviation
  Minimum and Maximum Salary Calculation
- Data Visualization
  Bar Charts
  Word Cloud
  Tree Map
  Heatmaps
- Statistical Analysis
  Comparing Mean Salaries
- Text Analysis
  Keyword and Skill Extraction

6. Conclusion……..……...23

- Team Achievement and Project Summary

Abstract

This study employs a web mining approach to conduct a comprehensive analysis of the contemporary job market, utilizing data extracted from Monster.com and leveraging Kaggle as a repository for relevant datasets. The primary objective is to extract actionable insights to inform and guide job seekers, employers, and policymakers in navigating the dynamic landscape of employment opportunities.

The methodology involves a multi-faceted approach, starting with Exploratory Data Analysis (EDA) to uncover patterns and trends in job postings. Descriptive statistics, data distribution analyses, and correlation studies provide a foundational understanding of the key features in the dataset. Employing data visualization techniques, such as time series analysis and geospatial mapping, enhances the presentation of trends over time and regional variations in job demand.

The outcomes of this study offer practical implications for job seekers seeking to align their skills with market demand, employers refining recruitment strategies, and policymakers devising informed labor market policies. The findings are communicated through clear and informative visualizations, statistical models, and actionable recommendations, ensuring the relevance and applicability of the analysis in addressing real-world challenges in the job market.

Dataset Description

Data Source:

US Job Posts on Monster.com (for US Job market)

[Perfect Dataset to get the hands dirty ! | Kaggle]

Dataset Overview:

The goal of our project is to analyze and derive insights from a dataset related to job listings, obtained through web mining. The "Job Posts" dataset that was obtained from Kaggle will be used for our project. This dataset is stored in a CSV file ('jobdata.csv') which contains information about job types, industry sectors, salaries, locations, and other relevant details.

Key Attributes:

- Job Posts- role
- Country and Country-code
- Date of the Job posted and Expiry date
- Title of the Job
- Name of the Company
- Location of the Job
- Salary offering

Type of Analysis

Descriptive Statistics:

We performed descriptive statistics on the job market dataset extracted through web mining techniques. Our descriptive statistics include measures such as mean, median, standard deviation, maximum and minimum for the salaries. These statistics provide a summary of key numerical variables, offering insights into the central tendency and dispersion of data. Descriptive statistics are crucial for understanding the basic characteristics of the dataset, laying the groundwork for subsequent analyses.

Descriptive statistics provide a summary of the main aspects of a dataset, offering insights into its central tendencies and variability. In the context of your project, descriptive statistics are applied to the 'salary' column, revealing key numerical characteristics. The mean salary, denoted by μ (mu), represents the average salary across all job listings. It is calculated by summing up all individual salaries and dividing by the total number of observations. The median, denoted as Md or M, is the middle value in the sorted list of salaries and provides a measure of central tendency that is less sensitive to extreme values. The standard deviation, symbolized as σ (sigma), quantifies the amount of variability or dispersion in the salary data. It is calculated as the square root of the variance, which is the average of the squared differences between each salary and the mean. The maximum and minimum salaries provide the range of values within the dataset, indicating the spread of salary levels. By examining these descriptive statistics, you gain a comprehensive understanding of the salary distribution, including the average, middle point, variability, and range, which collectively offer valuable insights into the overall salary landscape within the dataset.

Data Cleaning:

Data cleaning is an essential step to ensure the quality and integrity of the dataset in our project. This involves handling missing values, removing duplicates, addressing outliers, and standardizing formats. In our project we worked on the handling of missing values and removal of duplicates. Cleaning the data enhances its suitability for analysis, ensuring that subsequent exploratory and statistical analyses are based on reliable and consistent information. By addressing data quality issues, we aim to produce robust and accurate results in the job market analysis.

Exploratory Data Analysis (EDA):

Exploratory Data Analysis (EDA) involves a comprehensive examination of the dataset to identify patterns, relationships, and trends. EDA sets the stage for more in-depth analyses, guiding the selection of appropriate statistical methods and visualization techniques to extract meaningful information. Through visualizations and statistical summaries, we as a team have gained insights in Job Postings, Job Titles, Locations, Job description, Geographical distribution of Job Openings, Required skills for the job. Notable visualizations include bar plots for the top 20 job titles and top 20 job locations, as well as a histogram depicting the distribution of job description lengths.

It is a critical phase in the data analysis process that involves summarizing, visualizing, and interpreting the main characteristics, patterns, and trends within a dataset. Its primary goal is to gain insights into the underlying structure of the data, identify relationships between variables, and generate hypotheses for further investigation. In the provided Python code for a job dataset, EDA is conducted using various statistical and visual methods.

Descriptive statistics, such as mean, median, standard deviation, and quartiles, are computed using the '**describe()**' function, offering a concise summary of the dataset's central tendency and dispersion. Missing values are identified using the **'isnull().sum()'** function, providing an overview of potential data gaps. Visualizations, including bar plots, treemaps, word clouds, histograms, and heatmaps, are employed to represent the distribution of job types, industry sectors, job titles, salaries, and other relevant features.

Additionally, the code performs data cleaning operations to extract and organize salary information, demonstrating techniques like regular expressions for pattern matching and manipulation. The application of statistical analysis involves assessing salary statistics, checking for stationarity in time series data using the Augmented Dickey-Fuller test, and exploring job listings trends over time. Hypothesis testing is introduced with the aim of extracting meaningful insights from the dataset.

In summary, EDA is a multifaceted process that combines statistical measures and visualizations to uncover patterns, outliers, and relationships within a dataset. It serves as a crucial precursor to more advanced analyses and aids in formulating hypotheses for subsequent modeling or hypothesis testing. The techniques showcased in the code contribute to a comprehensive exploration of the job dataset, revealing valuable information for further investigation and decision-making.

Data Visualization:

Data visualization is a critical component for conveying complex patterns and trends in a clear and understandable manner. Employing charts, graphs, and maps, we illustrate key findings from the job market dataset. Visual representations enhance the communication of insights related to job postings, salary distributions, geographical variations, and skill demand. Effective data visualization facilitates better interpretation and understanding of the job market dynamics. In our project we used Bar Charts, Word Cloud, Tree map and Heatmaps to represent our data in a more understanding and clear manner.

1. Heatmaps:
   Heatmap is created to visualize the distribution of hourly salaries across different job titles. The 'hourly_salary_matrix' is a matrix where rows represent job titles, columns represent salary bins, and the values represent the count of job listings falling into each combination of job title and salary bin. The color intensity in the heatmap represents the count.
2. Bar Charts:
   Illustrate the distribution of job categories, industries, or educational requirements. Bar charts can represent the frequency of different categories.

3. Box Plots for Salary Analysis:
   Visualize the distribution of salaries, including median, quartiles, and potential outliers. Box plots provide a comprehensive view of salary variations within the dataset.
4. Word Clouds:
   Create word clouds to visually highlight the most frequently occurring words in job descriptions. This provides an intuitive representation of the most sought-after skills or qualifications.

Statistical Analysis:

Statistical analysis involves applying quantitative methods to explore relationships, test hypotheses, and derive meaningful conclusions. Statistical analysis provides a rigorous framework for extracting actionable insights from the dataset. In our project we used statistical techniques to uncover insights into factors influencing salary distributions of the job market. We compared the mean and medium salaries, standard deviation, maximum and minimum salaries too. However, our mean and medium salaries gave us “zero” values since we have missing values.

The statistical analysis primarily focuses on exploring and summarizing the 'salary' column in the dataset. The initial steps involve checking descriptive statistics, identifying missing values, and cleaning the 'salary' column. Descriptive statistics, such as mean, median, standard deviation, maximum, and minimum values, are computed to gain insights into the salary distribution.

To clean the 'salary' column, a function is defined to extract numeric salary values from text, considering the presence of non-numeric characters. Subsequently, non-numeric entries are handled, and rows with NaN salary values are removed from the dataset.

The cleaned 'salary' column allows for more meaningful statistical analysis. Measures like the mean salary provide the average value, while the median salary represents the middle value in the salary distribution, helping to mitigate the impact of outliers. The standard deviation quantifies the dispersion of salaries around the mean. Furthermore, the maximum and minimum salary values offer insights into the salary range within the dataset.

The statistical analysis provides a comprehensive understanding of the central tendency, variability, and distribution of salaries, enabling data-driven insights into the salary landscape of the job dataset. Additionally, the exploration of job listings over time, job counts by industry sector, and job type distribution enhances the overall understanding of the dataset and supports more informed decision-making.

Input:

```
# 1. Descriptive statistics for the 'salary' column
salary_stats = df['salary'].describe()
print(salary_stats)
```

Figure: Descriptive Statistics

This code snippet prints the summary statistics of the entire DataFrame, which includes statistical measures such as count, mean, standard deviation, minimum, 25th percentile (Q1), median (50th percentile or Q2), 75th percentile (Q3), and maximum.

```
# Define a function to extract the numeric salary value from text
def extract_salary(salary_text):
    # Use a regular expression to extract numeric values
    matches = re.findall(r'\d+\.\d+', str(salary_text))
    if matches:
        return float(matches[0])
    else:
        return None

# Apply the function to the 'salary' column to clean it, and handle non-numeric values
df['salary'] = df['salary'].apply(extract_salary)

# Remove rows with NaN salary values
df = df.dropna(subset=['salary'])

# Now we can perform statistical analysis on the 'salary' column
mean_salary = df['salary'].mean()
median_salary = df['salary'].median()
std_dev_salary = df['salary'].std()
max_salary = df['salary'].max()
min_salary = df['salary'].min()
```

Figure: Statistics of Cleaned 'salary' Column

This section defines a function (‘extract_salary’) to extract numeric salary values from text. It uses regular expressions to find and extract floating-point numbers. The 'salary' column is then cleaned using this function, and rows with NaN values are removed.

After cleaning the 'salary' column, this code calculates and prints various statistics of the 'salary' distribution, including mean, median, standard deviation, maximum, and minimum values.

Output:

```
count                                3446
unique                               1737
top       40,000.00 - 100,000.00 $ /year
freq                                   50
Name: salary, dtype: object
```

Figure: Descriptive Statistics

```
Mean Salary: $863.05
Median Salary: $0.00
Standard Deviation of Salary: $7555.93
Maximum Salary: $150000.00
Minimum Salary: $0.00
```

Figure: Statistical analysis on salary column

These analyses provide insights into the distribution and characteristics of job data, specifically focusing on salary statistics.

Text Analysis:

Text analysis focuses on extracting insights from textual data, such as job descriptions. Natural Language Processing (NLP) techniques are applied to analyze the language used in job postings, identify key skills, and assess sentiment. Text analysis contributes to a deeper understanding of the qualitative aspects of the job market, providing valuable information for job seekers and employers regarding the language and requirements in job postings. In our project, we have used Keyword and Skill Extraction.

1. Word Clouds for Industry Sectors and Job Titles:
    a. Word clouds are generated using the 'WordCloud' library to visually represent the most frequent words in the 'sector' and 'job_title' columns.
    b. The 'sector' word cloud illustrates the distribution of industry sectors based on the frequency of occurrence.
    c. The 'job_title' word cloud provides insights into the most common job titles.

Figure: Industry Sector Word Cloud

Figure: Job Title Word Cloud

2. Cleaning and Extracting Salary Information:
    a. The 'clean_salary' function is defined to extract salary information from the 'salary' column using regular expressions.
    b. The cleaned salary information is then separated into 'year' and 'hour' columns, and the data is sorted based on job title and salary.

```
[ ] # Clean and extract salary information
    def clean_salary(salary_str):
        if pd.isna(salary_str):
            return None
        if 'year' in salary_str:
            match = re.search(r'(\d+(\.\d+)? - \d+(\.\d+)?) \$ \/year', salary_str)
            if match:
                return match.group(0)
        if 'hour' in salary_str:
            match = re.search(r'(\d+(\.\d+)? - \d+(\.\d+)?) \$ \/hour', salary_str)
            if match:
                return match.group(0)
        return None
```

```
[ ] df['cleaned_salary'] = df['salary'].apply(clean_salary)

    # Separate salary into 'year' and 'hour' columns using str.extractall
    df['year_salary'] = df['cleaned_salary'].str.extractall(r'(\d+(\.\d+)?) - (\d+(\.\d+)?) \$ \/year')[0].unstack()
    df['hour_salary'] = df['cleaned_salary'].str.extractall(r'(\d+(\.\d+)?) - (\d+(\.\d+)?) \$ \/hour')[0].unstack()

    # Sort data based on job role and salary for 'year' and 'hour'
    sorted_year_salary = df[df['year_salary'].notnull()].sort_values(by=['job_title', 'year_salary'])
    sorted_hour_salary = df[df['hour_salary'].notnull()].sort_values(by=['job_title', 'hour_salary'])
```

Figure: Cleaning and Extracting Salary Information

3. Hourly Salary Distribution Heatmap:
    Hourly salary data is visualized using a heatmap to show the distribution of job titles across different hourly salary bins.

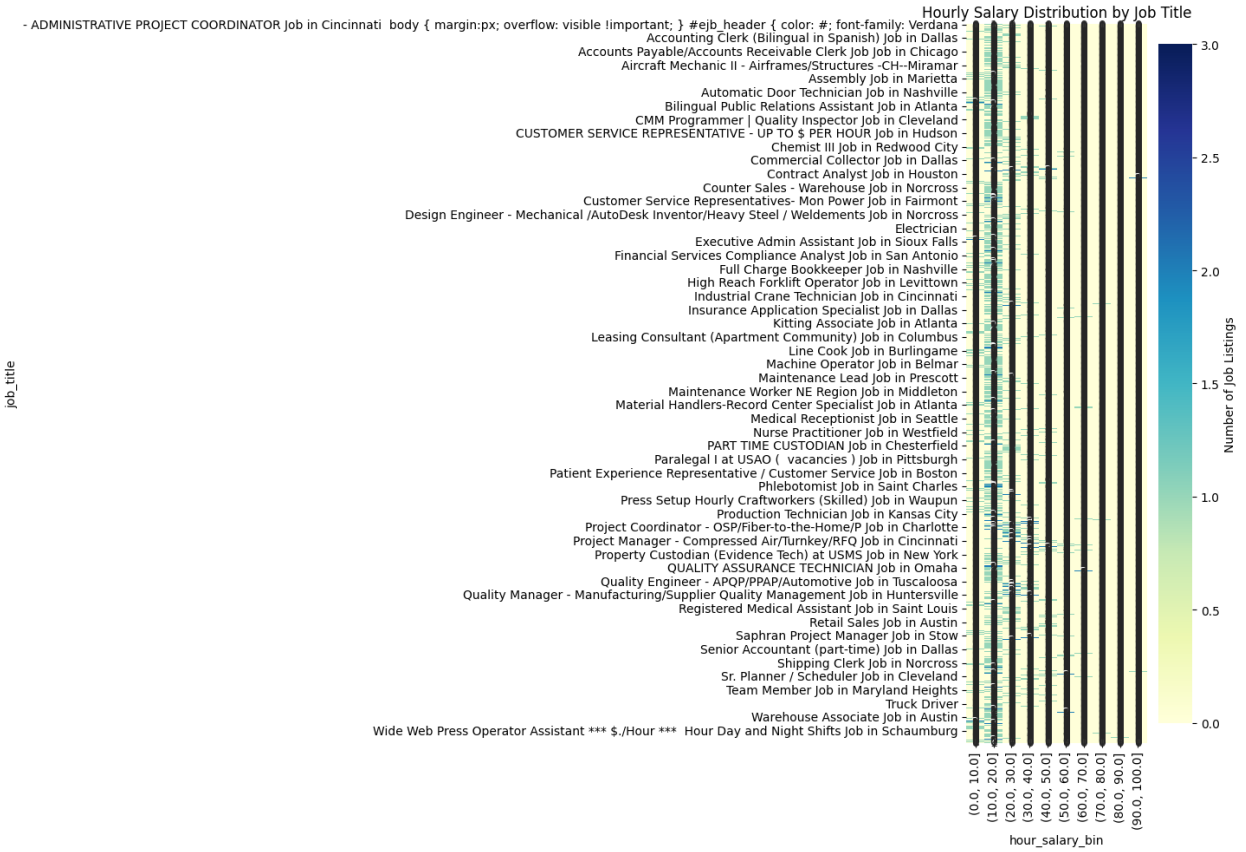

Figure: Hourly Salary Distribution by Job Title

TF-IDF Vectorization:

TF-IDF (Term Frequency-Inverse Document Frequency) is a technique employed in natural language processing to transform textual data into numerical vectors, providing a numerical representation of the importance of words in a document. In the context of your project, TF-IDF is utilized to convert job descriptions into vectors by assigning each word a weight based on its frequency in a document and its rarity across the entire dataset. The term frequency (TF) measures how often a word appears in a document, and the inverse document frequency (IDF) quantifies the importance of a word by considering how frequently it occurs across all documents. By multiplying TF and IDF, TF-IDF emphasizes words that are both common within a document and unique to that document relative to the entire dataset. This process is crucial for capturing the distinctive characteristics of each job description and creating a meaningful numerical representation that can be used for further analysis.

1. Term Frequency (TF): Measures how often a term (word) appears in a document. It is calculated as the number of occurrences of a term in a document divided by the total number of terms in that document.

   $$\text{TF(t,d)} = \frac{Number\ of\ times\ term\ t\ appears\ in\ document\ d}{Total\ number\ of\ terms\ in\ document\ d}$$

   where, t typically represents a term or word and d represents a document.

2. Inverse Document Frequency (IDF): Measures the importance of a term across a collection of documents. It is calculated as the logarithm of the total number of documents divided by the number of documents containing the term, with 1 added to the divisor to prevent division by zero.

   $$\text{IDF(t,D)} = \log\left(\frac{Total\ number\ of\ documents\ in\ the\ collection\ \boldsymbol{D}}{Number\ of\ documents\ containing\ term\ \boldsymbol{t}\ in\ \boldsymbol{D}+1}\right) + 1$$

3. TF-IDF: The product of TF and IDF. It represents the importance of a term in a specific document relative to its importance across the entire collection.

   $$\text{TF-IDF(t,d,D)} = \text{TF(t,d)} \times \text{IDF(t,D)}$$

```
# Step 1: TF-IDF Vectorization
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)  # We can adjust max_features as needed
tfidf_matrix = tfidf_vectorizer.fit_transform(job_descriptions)
```

Figure: TF-IDF Vectorization Implementation Code

Cosine Similarity:

Cosine Similarity, on the other hand, is a metric employed to quantify the similarity between two vectors by measuring the cosine of the angle between them. In the context of your project, the TF-IDF vectors generated for different job descriptions are compared using Cosine Similarity to assess how similar or dissimilar the textual content of these job descriptions is. A cosine similarity of 1 indicates perfect similarity, while 0 denotes no similarity. The reason for employing Cosine Similarity in this project is to facilitate the identification of job postings that share similar content, which can be valuable for tasks such as recommending similar job roles or clustering jobs based on their descriptions. By calculating the cosine similarity matrix, we obtain a comprehensive measure of the textual similarities between different job postings, aiding in the exploration and categorization of the job dataset based on content similarities.

1. The Cosine Similarity Matrix ('cosine_sim'):
   a. The matrix is symmetric, and each element 'cosine_sim[i][j]' represents the cosine similarity between the job listing at index 'I' and the job listing at index 'j'.
   b. The diagonal elements ('cosine_sim[i][i]') represent the similarity of a job listing with itself, and these values are always 1 (maximum similarity).
   c. Values close to 1 indicate high similarity, and values close to 0 indicate low similarity.

```
# Step 2: Calculate Cosine Similarity
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

# Display the Cosine Similarity matrix
cosine_sim_df = pd.DataFrame(cosine_sim, index=df.index, columns=df.index)
print(cosine_sim_df)
```

Figure: Calculating and printing cosine similarity matrix

2. Cosine Similarity Formula: Measures the cosine of the angle between two non-zero vectors. In the context of text data, these vectors represent the TF-IDF representations of documents.

   $$\text{Cosine Similarity(A,B)} = \frac{A \cdot B}{||A|| \cdot ||B||}$$

   where A·B is the dot product of vectors A and B, and ||A|| is the Euclidean norm (magnitude) of vector A.

3. Cosine Similarity Matrix Explaination: The cosine similarity matrix presented here is a square matrix where each row and column corresponds to a unique job listing. The values within the matrix quantify the cosine similarity between pairs of job listings, essentially measuring the degree of similarity in their descriptions. The diagonal elements of the matrix, representing the self-similarity of each job listing, are uniformly 1, as a job listing is perfectly similar to itself. Off-diagonal elements capture the similarity between different job listings, ranging from 0 to 1, where higher values signify greater similarity. For example, a value of 0.050836 at the intersection of row 13 and column 14 indicates a relatively low similarity between the job descriptions of listings 13 and 14. This cosine similarity matrix is particularly useful in tasks such as job recommendation systems, where it can be employed to identify jobs that are closely related based on the content of their descriptions. Analyzing the matrix allows for the exploration of patterns, clusters, and relationships among the various job listings in the dataset, aiding in the extraction of meaningful insights for decision-making or recommendation purposes.

	13	14	19	29	30	32	36	\
13	1.000000	0.050836	0.045293	0.031737	0.013648	0.043453	0.035253	
14	0.050836	1.000000	0.064552	0.072109	0.021474	0.060150	0.022282	
19	0.045293	0.064552	1.000000	0.101919	0.046504	0.724116	0.087874	
29	0.031737	0.072109	0.101919	1.000000	0.038444	0.093147	0.169140	
30	0.013648	0.021474	0.046504	0.038444	1.000000	0.069409	0.045058	
...	...	...	...	...	...	...	...	
21987	0.027066	0.058900	0.023660	0.080077	0.014699	0.023886	0.021325	
21995	0.040279	0.018820	0.028382	0.057303	0.036913	0.043377	0.033919	
21996	0.029044	0.039845	0.054069	0.061855	0.033612	0.047012	0.041245	
21998	0.018446	0.041332	0.160777	0.128328	0.041562	0.056189	0.101420	
21999	0.010319	0.053286	0.045268	0.059150	0.032790	0.052104	0.033543	

	41	42	43	...	21960	21971	21973	\
13	0.018326	0.039343	0.027682	...	0.036855	0.051195	0.018210	
14	0.034360	0.037114	0.088781	...	0.049497	0.058984	0.021054	
19	0.036435	0.049505	0.065426	...	0.092276	0.067849	0.016939	
29	0.068836	0.034208	0.072852	...	0.119219	0.050262	0.073812	
30	0.023666	0.022367	0.057676	...	0.066993	0.029069	0.037440	
...	...	...	...	...	...	...	...	
21987	0.029652	0.043595	0.052881	...	0.064950	0.036296	0.045950	
21995	0.013833	0.037867	0.023636	...	0.137128	0.064936	0.165931	
21996	0.020262	0.052970	0.032559	...	0.199838	0.066499	0.247505	
21998	0.034973	0.037527	0.057376	...	0.093625	0.051067	0.036078	
21999	0.017243	0.082650	0.045262	...	0.070200	0.046284	0.039186	

	21976	21982	21987	21995	21996	21998	21999
13	0.032607	0.018931	0.027066	0.040279	0.029044	0.018446	0.010319
14	0.028244	0.055005	0.058900	0.018820	0.039845	0.041332	0.053286
19	0.077488	0.084553	0.023660	0.028382	0.054069	0.160777	0.045268
29	0.048507	0.036512	0.080077	0.057303	0.061855	0.128328	0.059150
30	0.048296	0.055163	0.014699	0.036913	0.033612	0.041562	0.032790

Figure: Display the Cosine Similarity matrix

Output:

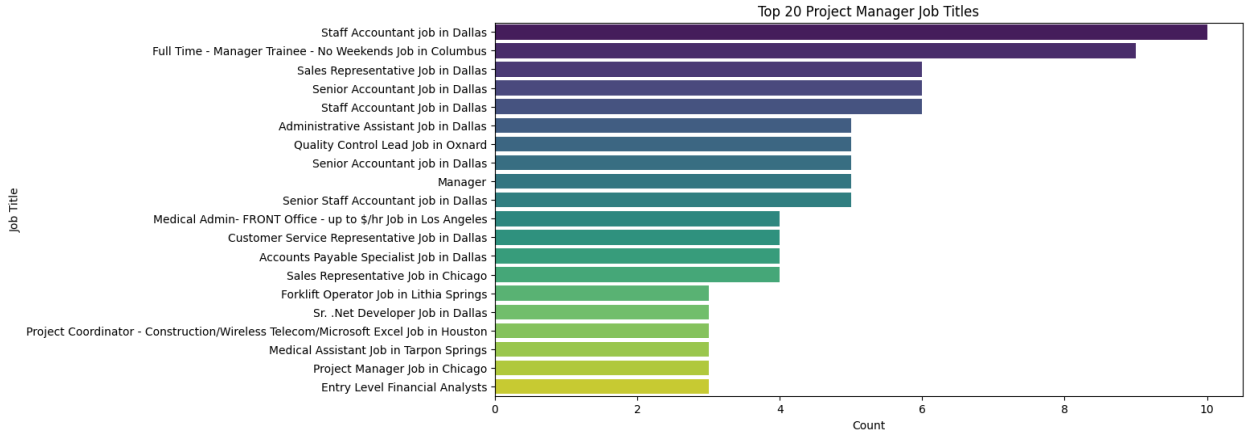

Figure: Top 20 Project Manager Job Titles

The output shows the top 5 job postings that are most similar to the job titled 'Project Manager' based on the calculated cosine similarity scores. Each row in the output represents

a job posting, including the job title and job description. It provides a list of job postings that are most similar to the specified job title, allowing you to identify other relevant positions based on the textual content of the job descriptions.

Salary Analysis and Visualization Algorithm for Job Listings:

For this Analysis, we have implemented a data cleaning and analysis algorithm to handle salary information from a DataFrame containing job listings. The primary goal is to clean and extract salary details, categorize them into yearly and hourly formats, and then visualize the salary distribution for different job titles.

Firstly, we define a function called **'clean_salary'** to extract and format salary information. We use regular expressions to identify patterns indicating yearly or hourly wages. We then apply this function to a 'salary' column in the DataFrame, creating a new 'cleaned_salary' column.

Next, we separate the cleaned salary information into 'year' and 'hour' columns using the **'str.extractall'** function. We sort the data based on job titles and the corresponding yearly or hourly salaries to facilitate better analysis.

For hourly salaries, we convert the 'hour_salary' column to numeric values, drop rows with NaN values, and categorize the salaries into bins. We create a heatmap to visualize the distribution of hourly salaries across different job titles.

Similarly, for yearly salaries, we clean the 'year_salary' column, calculate average yearly and hourly salaries for each job title, and generate bar plots and heatmaps to visualize the results. The final heatmaps include color scales representing the average yearly salaries for various job titles. We set an upper limit on the color scale for better visualization, ensuring that salaries exceeding this limit are appropriately colored.

Output:

```
Yearly Salaries:
                                                job_title year_salary
3806   ./hr Experienced RN supervisor (Smithtown) Job...       40.00
19461  ./hr Experienced RN supervisor (Smithtown) Job...       40.00
17658                Accounting Assistant Job in Dallas       17.00
8576   Accounts Receivable/Collections Clerk Job in D...       15.00
4509         Advisor Relationship Manager Job in Dallas        0.00
...                                                 ...         ...
19277        Surgical Dental Assistant Job in Ann Arbor       14.00
7406   Travel Sales Agent - Fluent in Spanish Job in ...       13.00
9161   Wanted: Hard Working Machine Operators! Job in...       12.00
9873                 Warehouse Worker Job in Jonesville       10.50
9712              rd Shift Packers Job in East Vineland        9.00

[66 rows x 2 columns]

Hourly Salaries:
                                                job_title hour_salary
21603  - ADMINISTRATIVE PROJECT COORDINATOR Job in Ci...       17.00
15640  .ELECTRICAL TECHNICIAN Job in Washington  body...       22.00
5983                    .Net Web Developer Job in Dallas       50.00
16315                AP Specialist Job in Jacksonville       13.00
10335          ASNT NDT Level II Inspector Job in Salem       22.92
...                                                 ...         ...
15453              open house monday // Job in Seminole       11.41
4451   rd Shift Assembly Positions $/hr Job in Chatta...       10.00
20174  st Shift Print Operator Job in Grove City  bod...       11.50
20154  st Shift Skilled Trade Position $/HR Job in Mo...       10.00
3918   st shift Warehouse Position $ - $ per hour Job...       12.00

[1029 rows x 2 columns]
```

Figure: Displaying the sorted salary data for both yearly and hourly formats

Figure: Average Yearly Salaries Heatmap by Job Title

In summary, we implemented a data cleaning and analysis algorithm to handle salary information, categorize it into yearly and hourly formats, and visualize the salary distribution for different job titles using bar plots and heatmaps. The code demonstrates the use of pandas, regular expressions, and visualization libraries like matplotlib and seaborn to gain insights into the salary landscape of job listings.

Technologies

Software Used:

Programming Language:

- Python

Python Libraries:

- Pandas, Matplotlib, Seaborn, Plotly Express, WordCloud, NumPy, Statsmodels, etc.

Project Implementation

Data Loading and Exploration:

1. Loading the Data:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[ ] df = pd.read_csv('jobdata.csv')
```

Figure: Loading the job data from the 'jobdata.csv' file

2. Data Exploration:

```
# Descriptive Statistics
print(df.describe())

# Missing Values
print(df.isnull().sum())
```

```
print(df.columns)
```

```
Index(['country', 'country_code', 'date_added', 'has_expired', 'job_board',
       'job_description', 'job_title', 'job_type', 'location', 'organization',
       'page_url', 'salary', 'sector', 'uniq_id'],
      dtype='object')
```

Figure: Descriptive statistics and missing value analysis were performed

Data Visualization:

1. Visualizing Job Types Distribution:

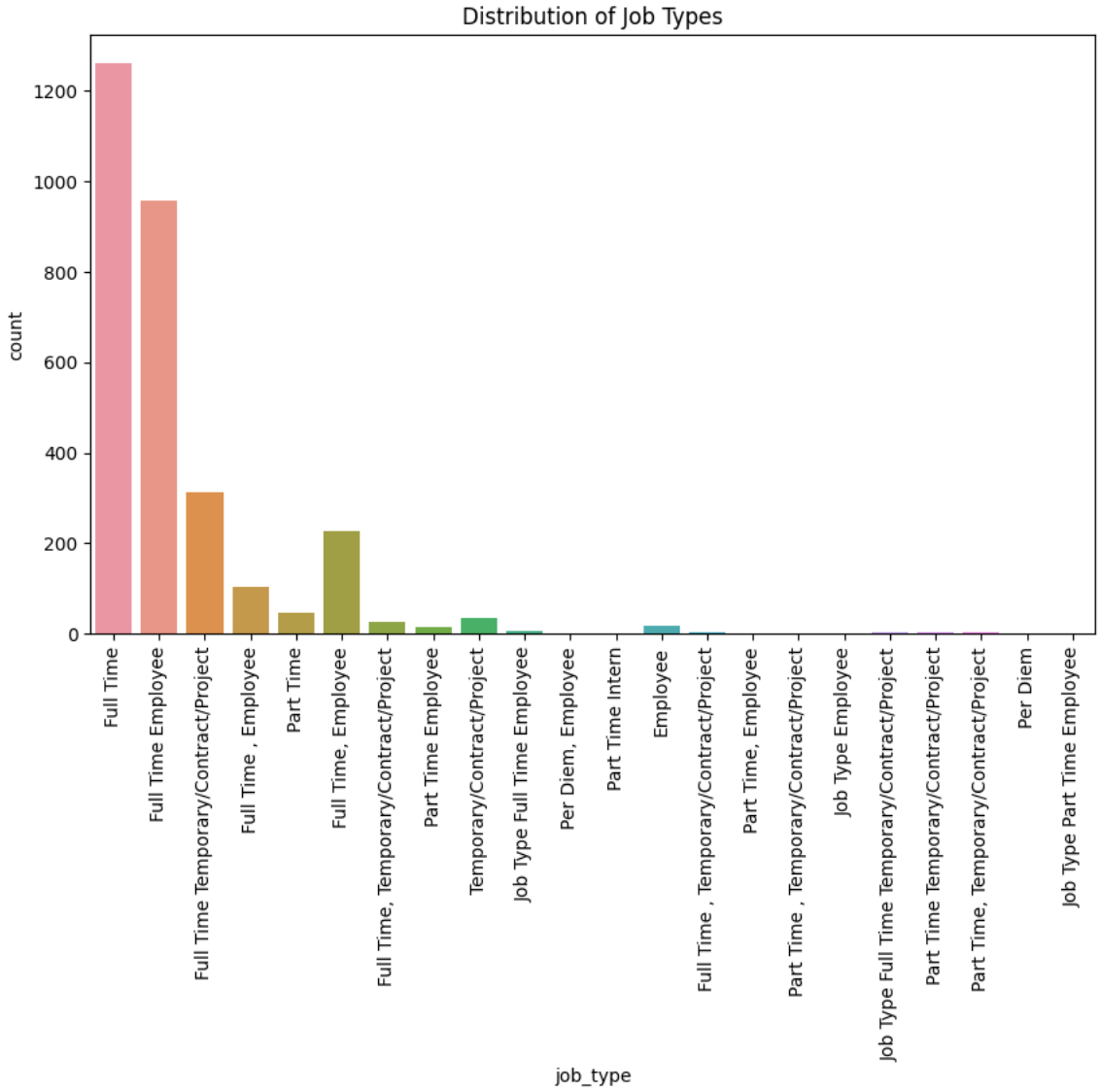

Figure: Distribution of Job Types

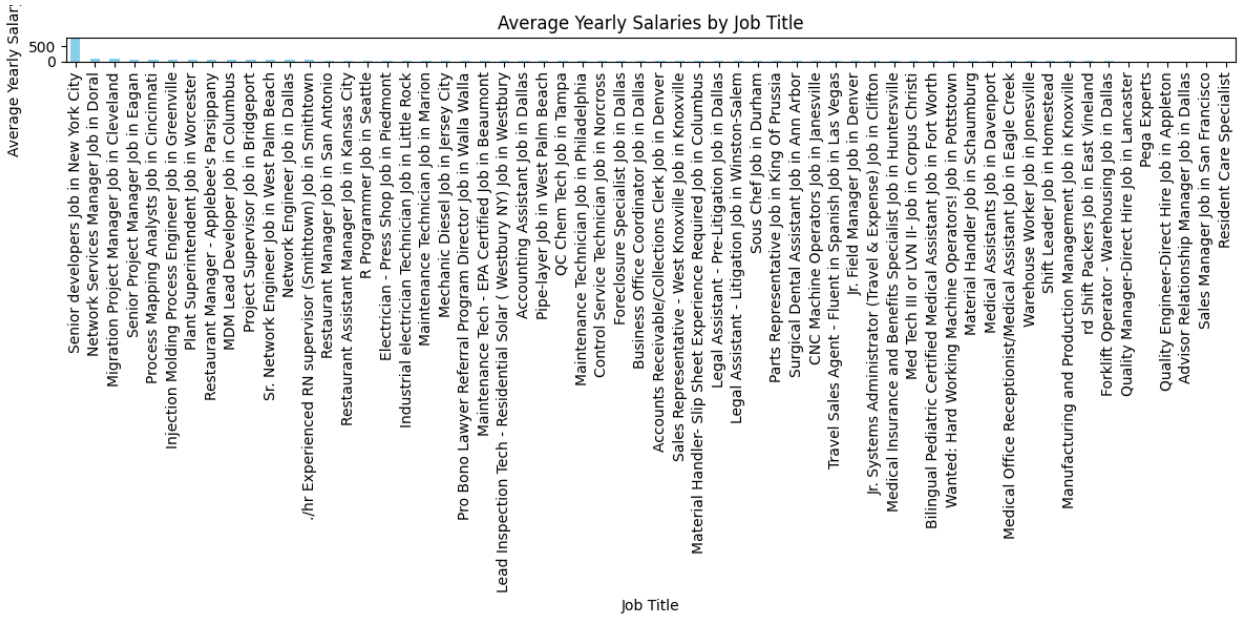

Figure: Average Yearly Salary in USD

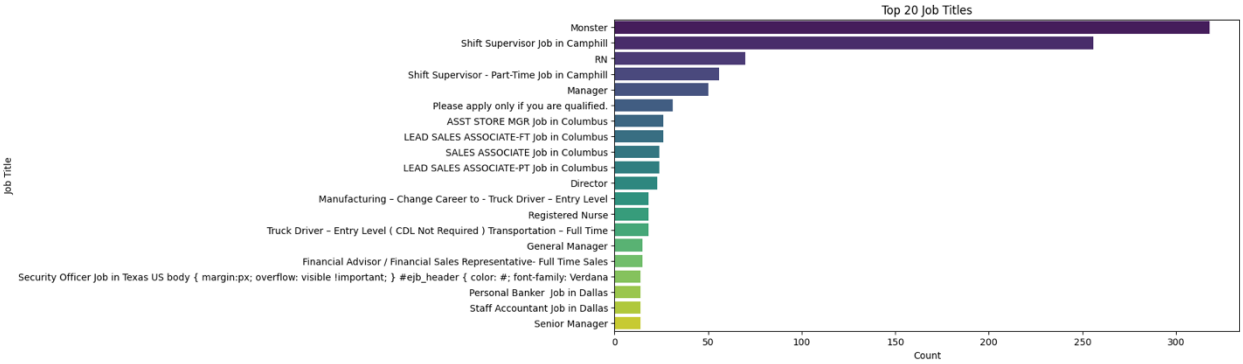

Figure: Top 20 Job Titles

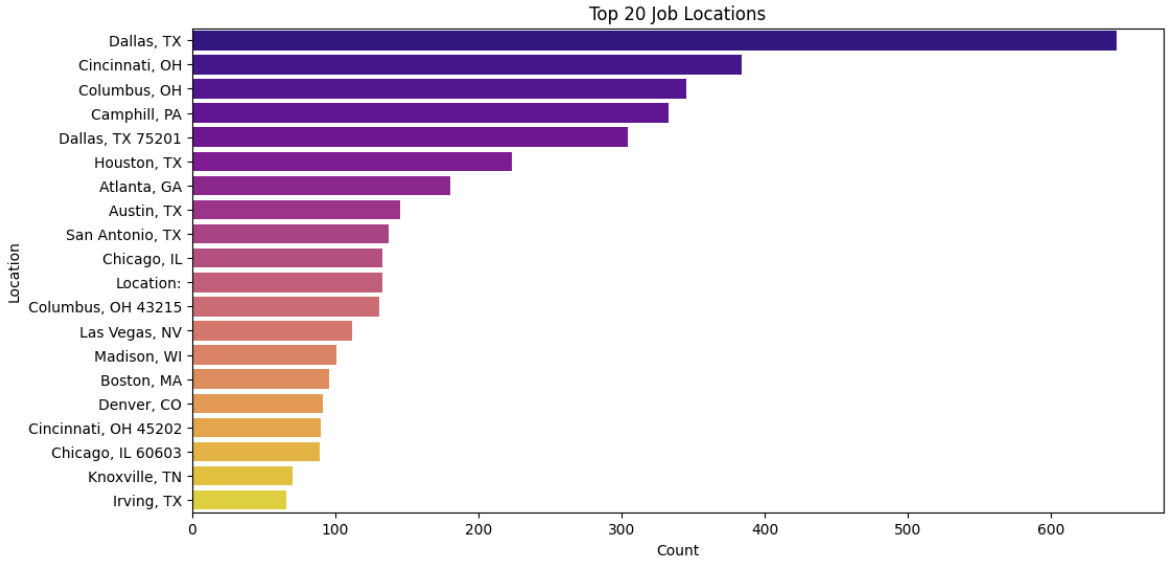

Figure: Top 20 Job Locations

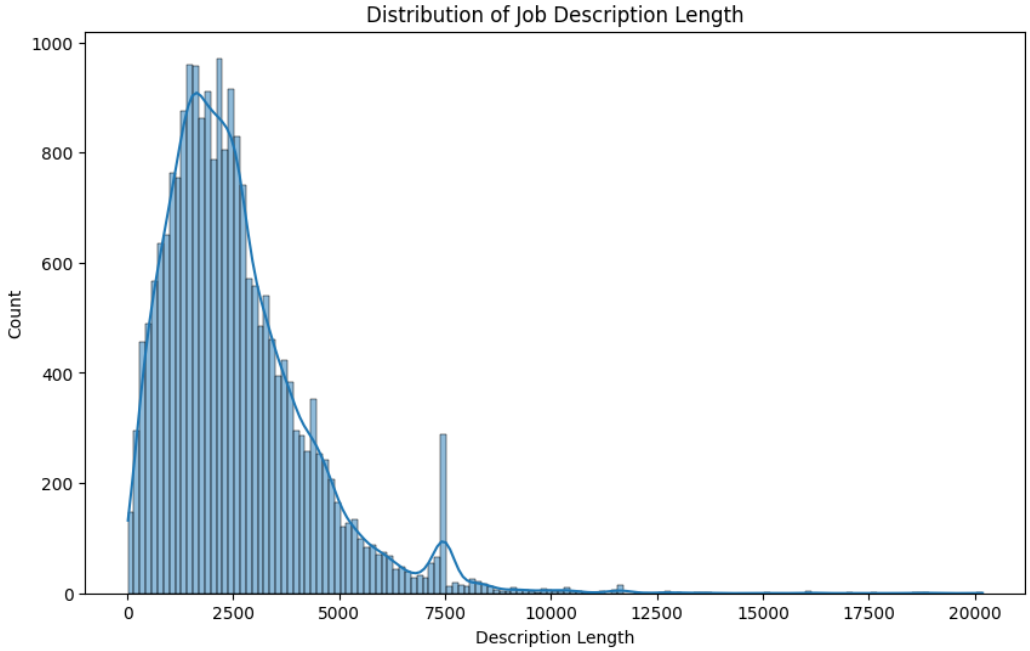

Figure: Distribution of Job Description Length

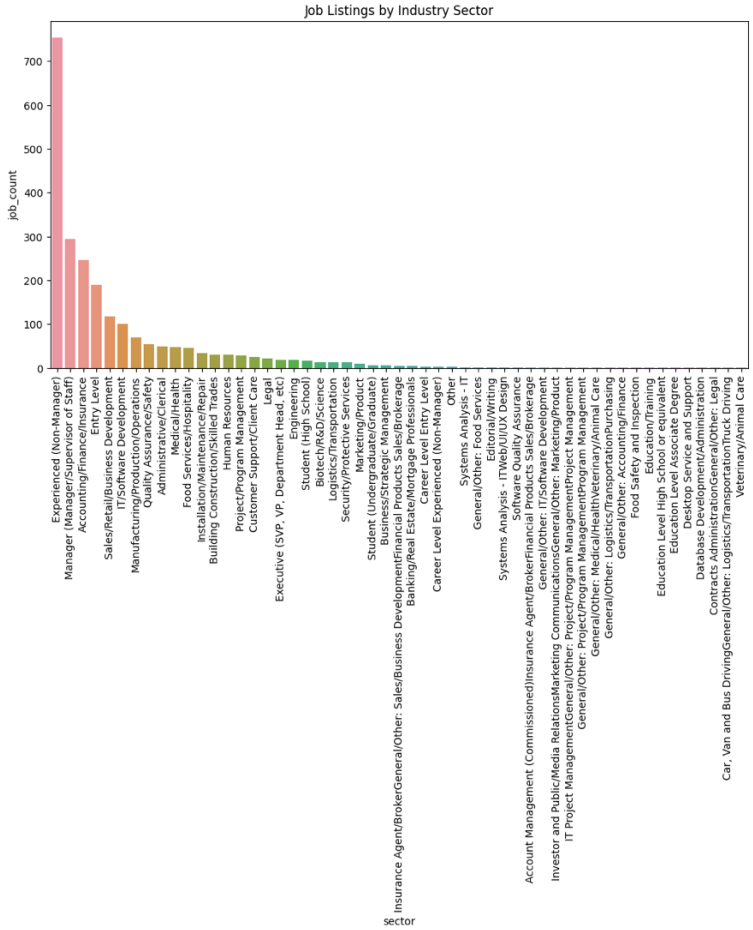

Figure: Job Listings by Industry Sector

2. Word Clouds:

Figure: Industry Sector Word Cloud

Figure: Job Title Word Cloud

3. Salary Analysis:

Figure: Average Yearly Salaries by Job Title

Figure: Average Yearly Salaries by Job Title (Max $250k)

Conclusion:

Team Analysis Report:

1. Data Loading and Exploration:

- We start by loading a CSV file ('jobdata.csv') into a Pandas DataFrame (df).
- We print descriptive statistics of the data using df.describe() to get a summary of numerical features.
- We check for missing values using df.isnull().sum() to understand data quality.
- We print the column names using df.columns.

2. Job Types Distribution Visualization:

- We use Seaborn to create a count plot of the distribution of job types ('job_type' column).
- The plot provides insights into the frequency of each job type.

3. Industry Sectors Visualization with Plotly Express:

- We use Plotly Express to create a treemap visualizing the distribution of job listings per industry sector ('sector' column).
- The treemap allows you to see the hierarchy and proportions of different sectors.

4. Word Clouds:

We create two word clouds:

- One based on the 'sector' column, providing a visual representation of industry sectors' frequencies.
- Another based on the 'job_title' column, giving insights into the most common words in job titles.

5. Salary Analysis:

- We define a function (clean_salary) to extract and clean salary information.
- We use regular expressions to extract salary values and create separate columns for yearly and hourly salaries.
- We visualize the distribution of hourly salaries with a heatmap.
- We calculate and visualize average yearly salaries for each job title.

6. Data Exploration and Visualization:

- We explore and visualize various aspects of the dataset, including job titles, locations, job description lengths, and more.
- We create bar plots to show the top 20 job titles and top 20 job locations.
- We visualize the distribution of job description lengths with a histogram.

7. Statistical Analysis:

- We use Statsmodels to perform statistical analysis.
- We check for stationarity using the Augmented Dickey-Fuller test.
- We perform descriptive statistics for the 'salary' column, cleaning and handling non-numeric values.
- We visualize job listings by industry sector over time and analyze job type distribution over time.

Project Summary:

The study effectively utilized web mining techniques to extract valuable insights from the job market dataset, providing a comprehensive analysis of trends and patterns related to job postings, industry sectors, salaries, locations, and skill requirements. The findings offer actionable recommendations for job seekers, employers, and policymakers:

The study's strengths include its comprehensive approach, employing descriptive statistics, data visualization, and statistical analysis to extract meaningful insights. Additionally, the use of text analysis techniques provides a deeper understanding of the qualitative aspects of the job market.

The study's limitations include its reliance on a single dataset from Monster.com, which may not fully represent the entire job market. Additionally, the analysis focused on US job postings, limiting its generalizability to other countries.

Future research directions include expanding the analysis to include data from multiple sources and countries, providing a more comprehensive global perspective on the job market. Furthermore, incorporating real-time data streams could enable continuous monitoring of job market trends and patterns.

Overall, the study provides valuable insights for job seekers, employers, and policymakers, contributing to a better understanding of the dynamics of the job market and informing strategies for navigating its complexities.