JOB RECOMMENDATION SYSTEM

Pheminah Wambui

Isaac Munyaka

Caroline Gesaka

Otiende Ogada

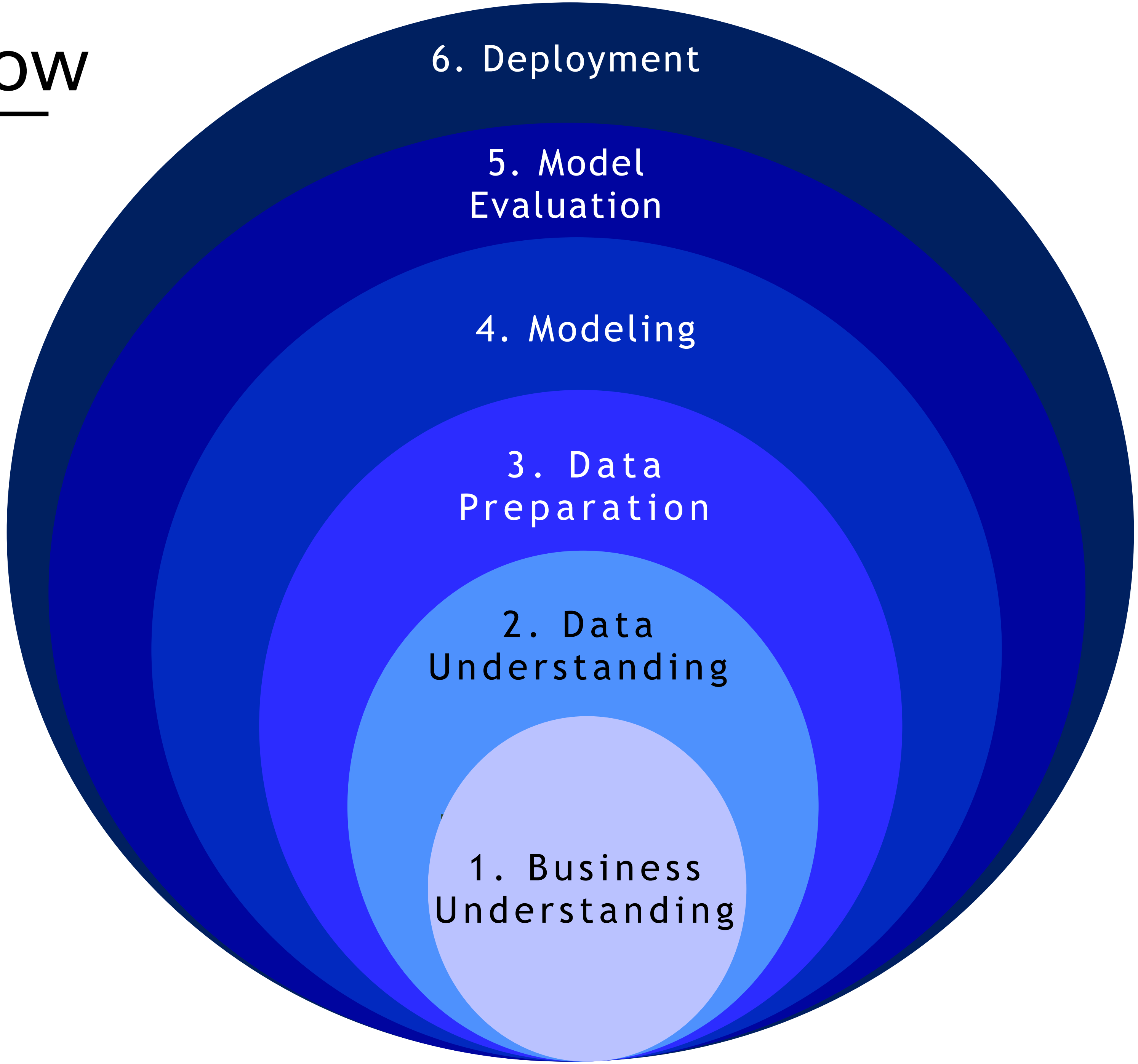
Anne Njoroge

Joan Maina

Project Engineers

Capstone Project

Content Flow



Background Story

The gap of job searching/workforce seeking remains weakly bridged in the employment sector.

Difficulty finding jobs:
Job seekers often experience challenges in getting jobs that match their skills.

Difficulty finding qualified employees:
Employers struggle to attract qualified candidates

Advanced Data Analytics & Machine Learning:
To enhance job matching efficiency and effectiveness for both job seekers and employers.

Business Understanding

Job Seekers
Individuals seeking suitable job opportunities.

Employers
Companies looking to recruit qualified candidates.

Recruitment Agencies
Agencies assisting in the recruitment process

Problem Statement

Goal
Enhance the **accuracy and relevance** of job matching.

Benefits
Improved job search experience for seekers. Streamlined recruitment process for employers.

Approach
Implement an **advanced recommendation system** using machine learning models.

Objectives

1. **Enhance Job Posting Quality:** Improve job descriptions to attract qualified candidates.
2. **Predict Candidate Interest:** Develop a model to predict job seekers' likelihood to apply.
3. **Optimize Job Recommendations:** Provide personalized job recommendations.
4. **Increase Application Rates:** Boost application rates to ensure better applications.

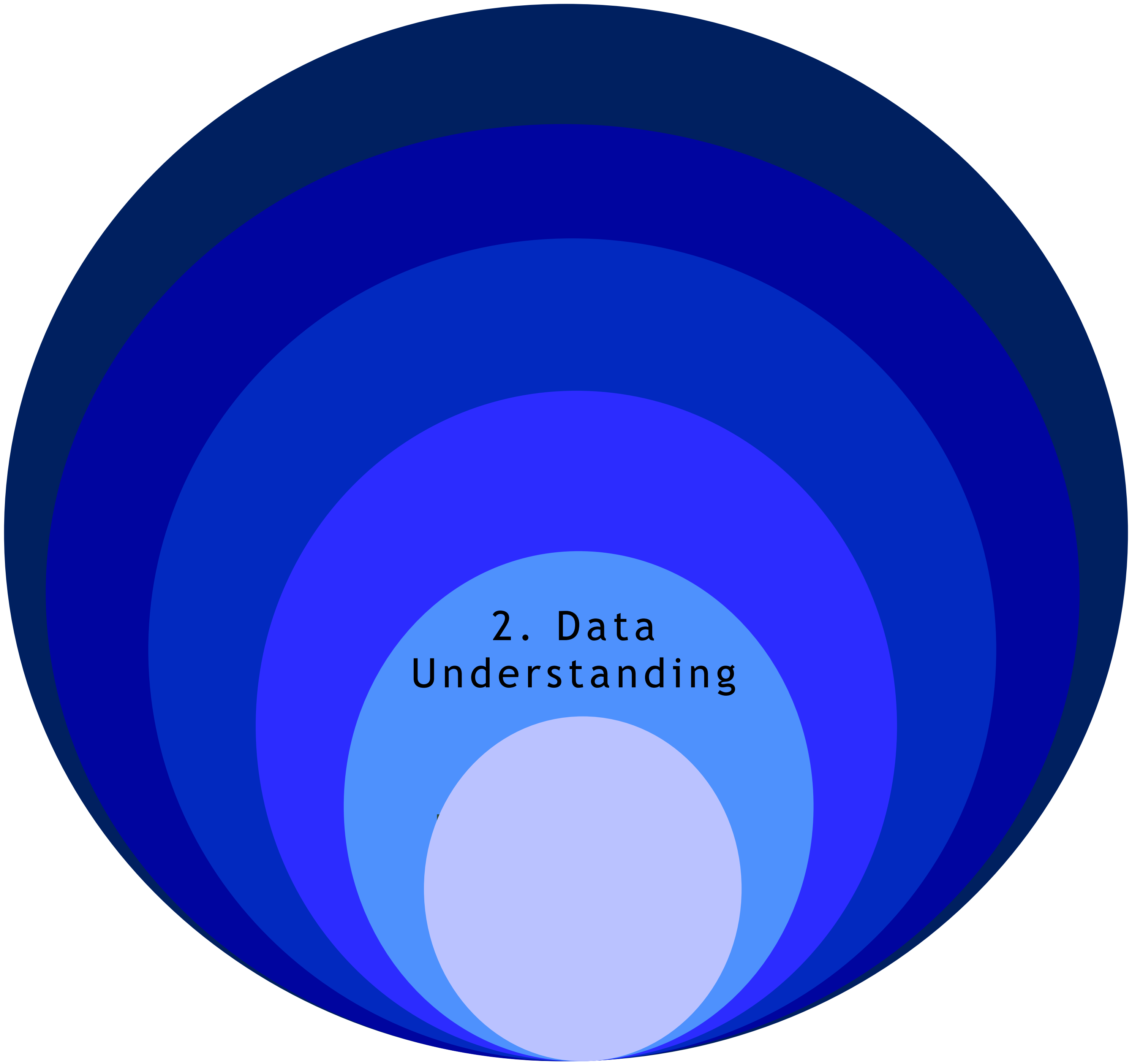
Success Metrics

Application Rate: Percentage of job postings receiving applications.

Qualified Application Rate: Percentage of applications meeting job requirements.

Time to Fill: Average time to fill a job position.

Click-Through Rate (CTR): Percentage of job seekers clicking on job postings.


2. Data
Understanding

Data Sources

Data Collection

- Collected data from job boards like LinkedIn and BrighterMonday.

LinkedIn Job Postings (2023 - 2024) (kaggle.com)

Types of Data

- Job Data: Titles, descriptions, requirements, locations, salaries, and companies.

- User Data: Aggregated from user profiles, resumes, skills, interactions, and historical applications.

Data Insights

- Job Titles: Standardizing titles helps in matching similar job roles.

- Skills: Extracted and normalized from job descriptions and user resumes using NLP techniques.

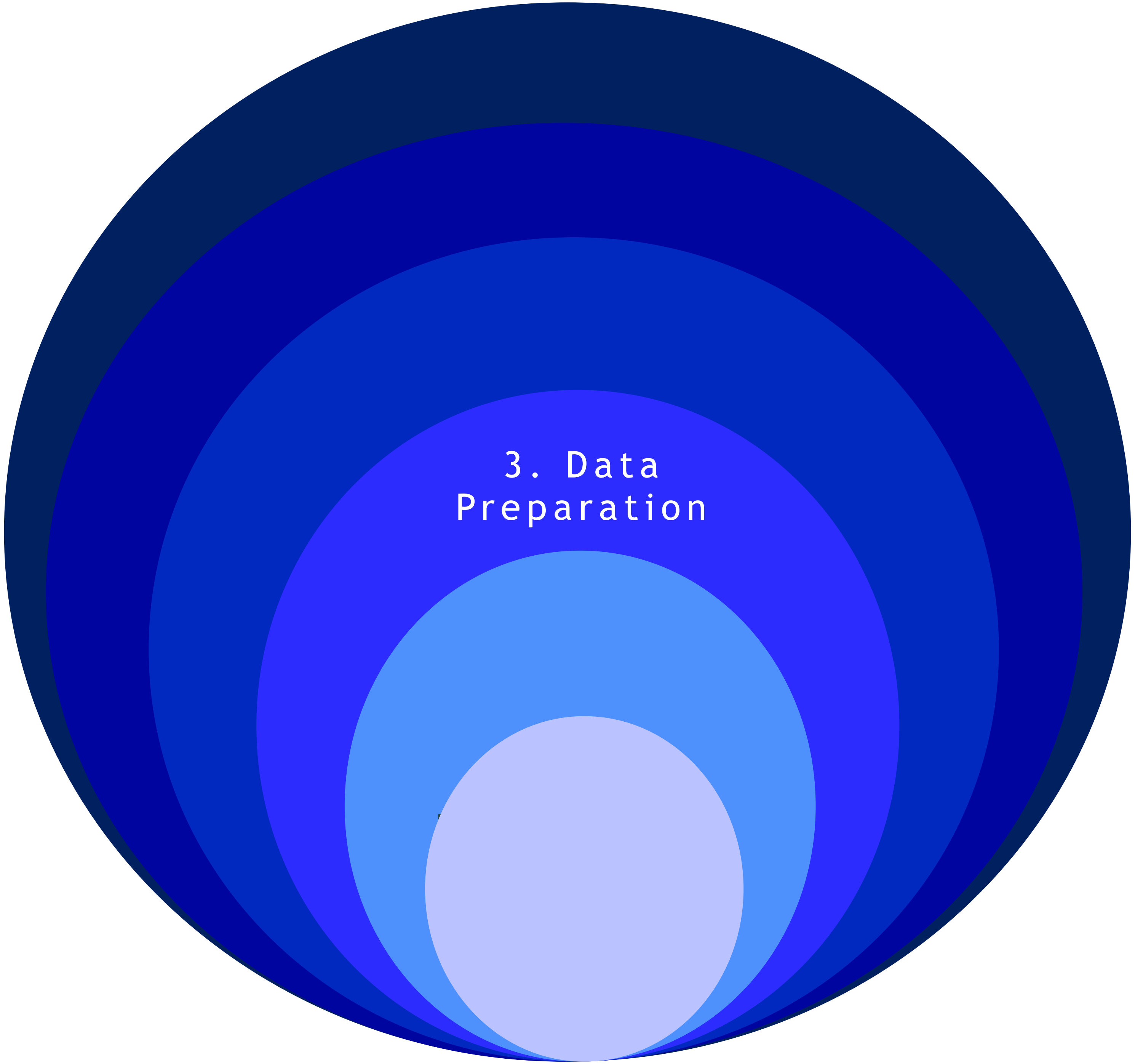
- Location: Geographical data is standardized to ensure consistency in job location matching.

- User Behavior: Historical application data provides insights into user preferences and behavior patterns.

Dataset Columnal View & Understanding:

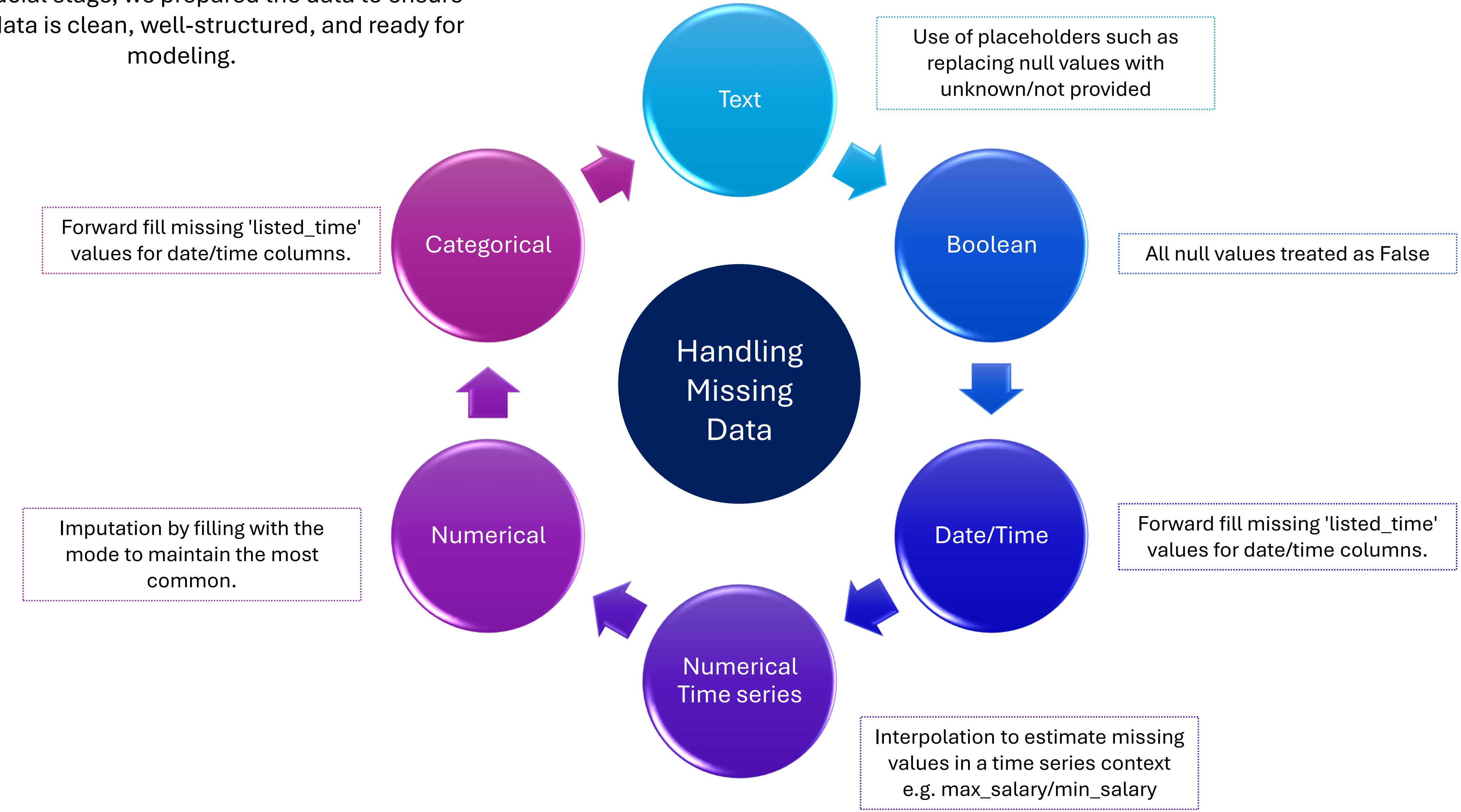
1. **job_id:** Unique identifier for each job posting, essential for tracking and referencing individual jobs.
2. **company_name:** Provides the name of the company offering the job, which helps in understanding the job context and employer branding.
3. **title:** The job title is critical for identifying the nature of the job and matching it with user preferences.
4. **description:** Contains detailed information about job responsibilities and requirements, which is crucial for matching the job with user skills and interests.
5. **max_salary:** Indicates the highest salary offered for the job, helping to match job seekers' salary expectations.
6. **min_salary:** Shows the lowest salary offered, providing a range for matching job seekers' financial requirements.
7. **location:** Geographic location of the job, which is important for matching based on job seekers preferred or available locations.
8. **views:** Number of views the job posting has received, which can indicate the popularity or competitiveness of the job.
9. **med_salary:** Median salary for the job, providing a central measure of compensation, useful for understanding typical earnings.
10. **applies:** Number of applications received, which can reflect job demand and help gauge the job's attractiveness.
11. **remote_allowed:** Indicates if remote work is an option, which is increasingly relevant for job seekers preferring or needing remote work arrangements.
12. **formatted_experience_level:** Specifies the required experience level (e.g., entry-level, senior), aiding in matching jobs with job seekers' experience.
13. **skills_desc:** Describes the skills required for the job, crucial for aligning job seekers' skills with job requirements.
14. **listed_time:** Timestamp of when the job was posted, helping to understand the job’s recency and relevance.
15. **posting_domain:** Indicates the industry or sector of the job, useful for matching jobs with job seekers’ industry interests.
16. **currency:** Specifies the currency in which the salary is offered, important for job seekers in different regions or countries.
17. **compensation_type:** Details the type of compensation (e.g., base salary, bonuses), helping job seekers understand the total compensation package.

Dataset Shape: Rows = 123849, Columns = 17


3. Data
Preparation

Handling Missing Data

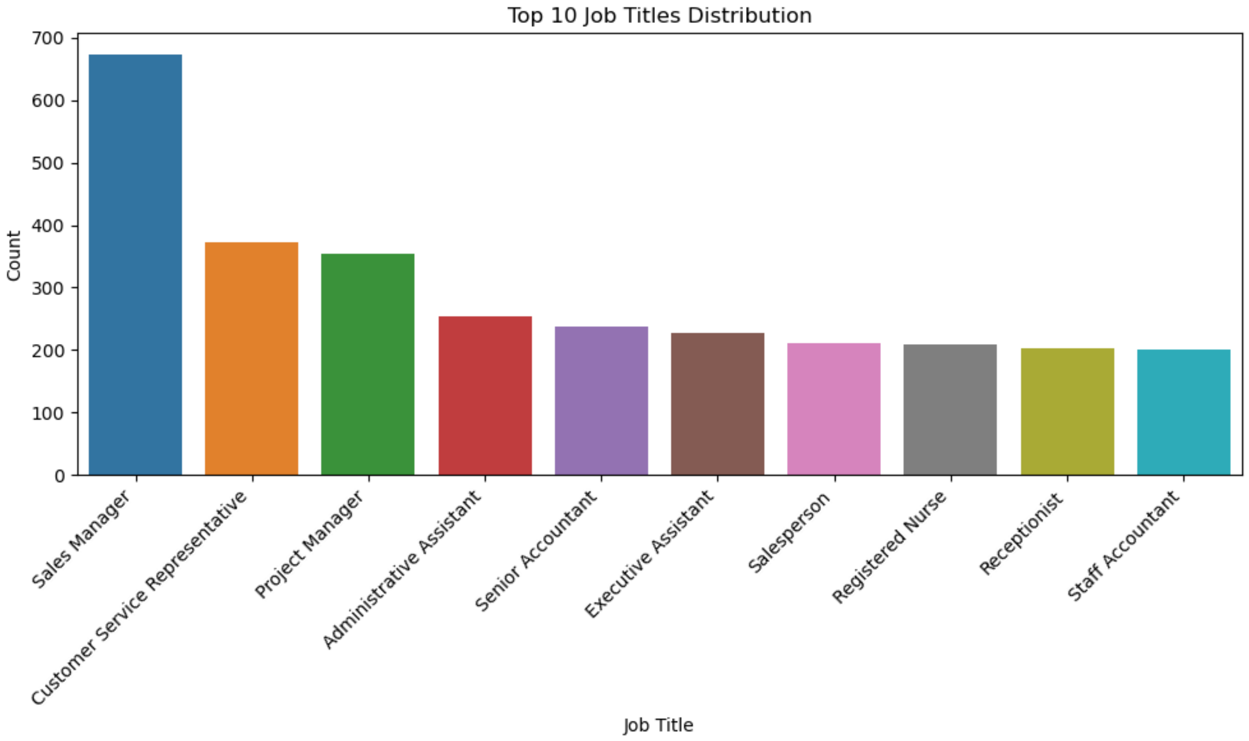
In this crucial stage, we prepared the data to ensure that the data is clean, well-structured, and ready for modeling.



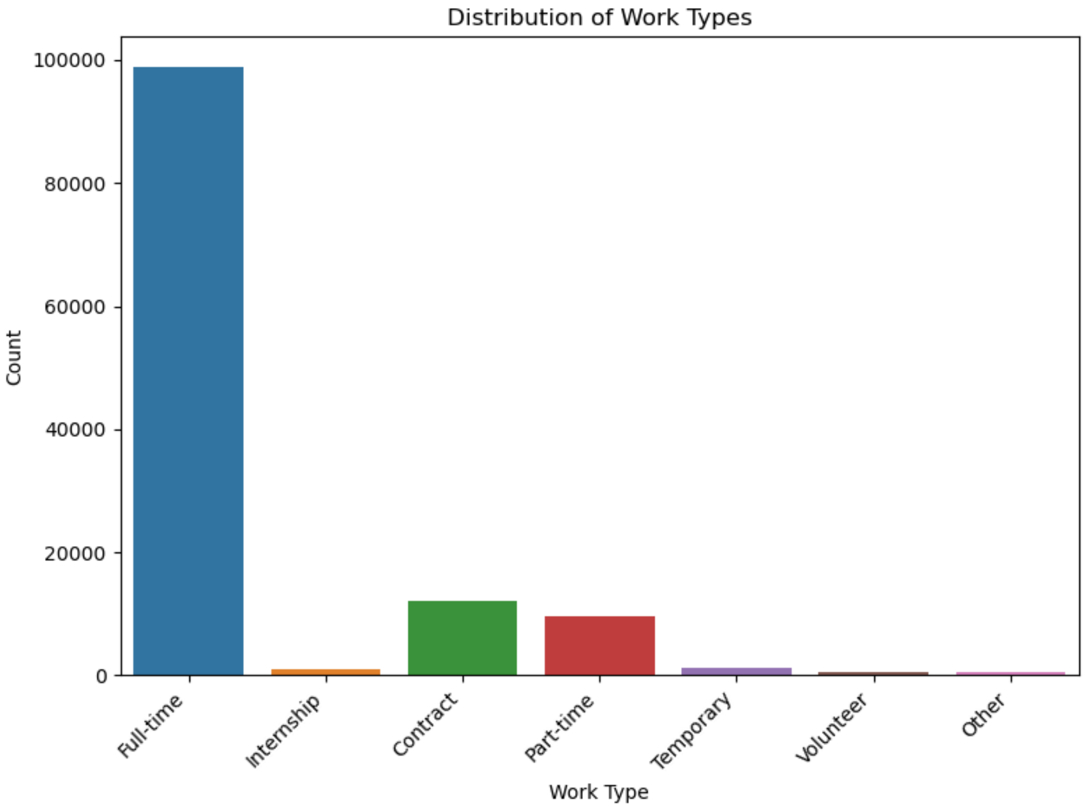
Exploratory Data Analysis: Univariate Analysis

A crucial step to understand the statistical metrics and distributions of the dataset.

Job Title Distribution



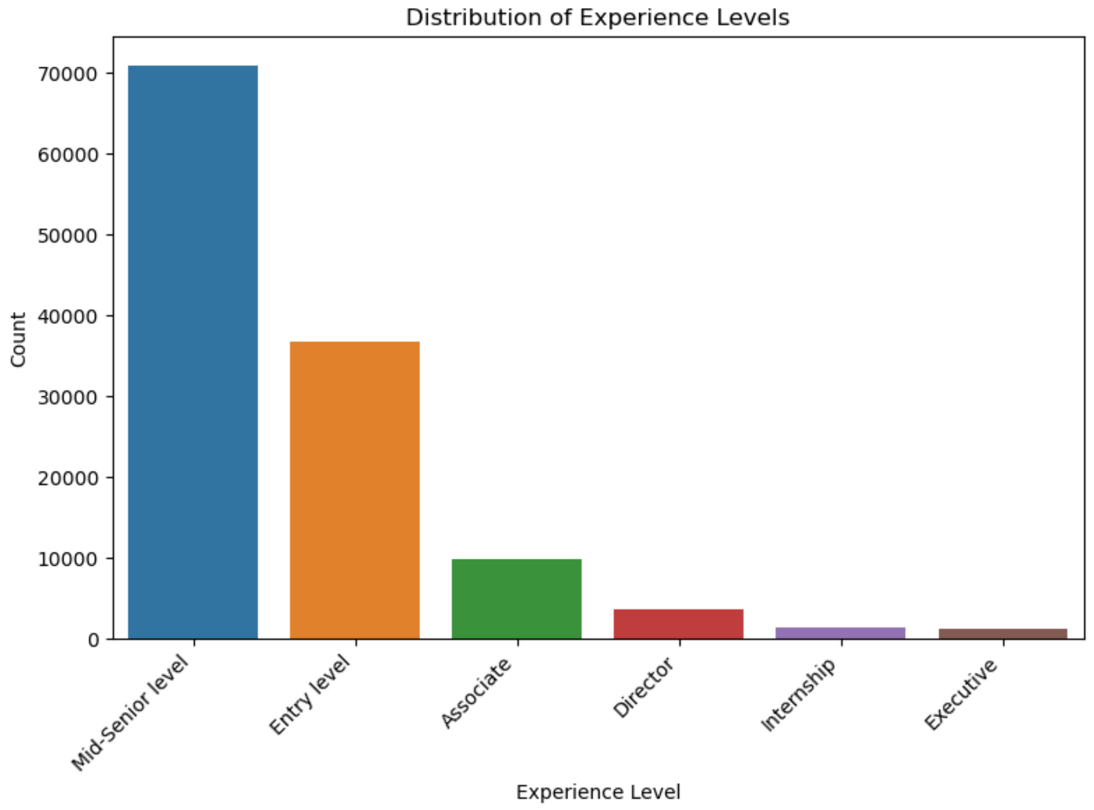
Work type Distribution

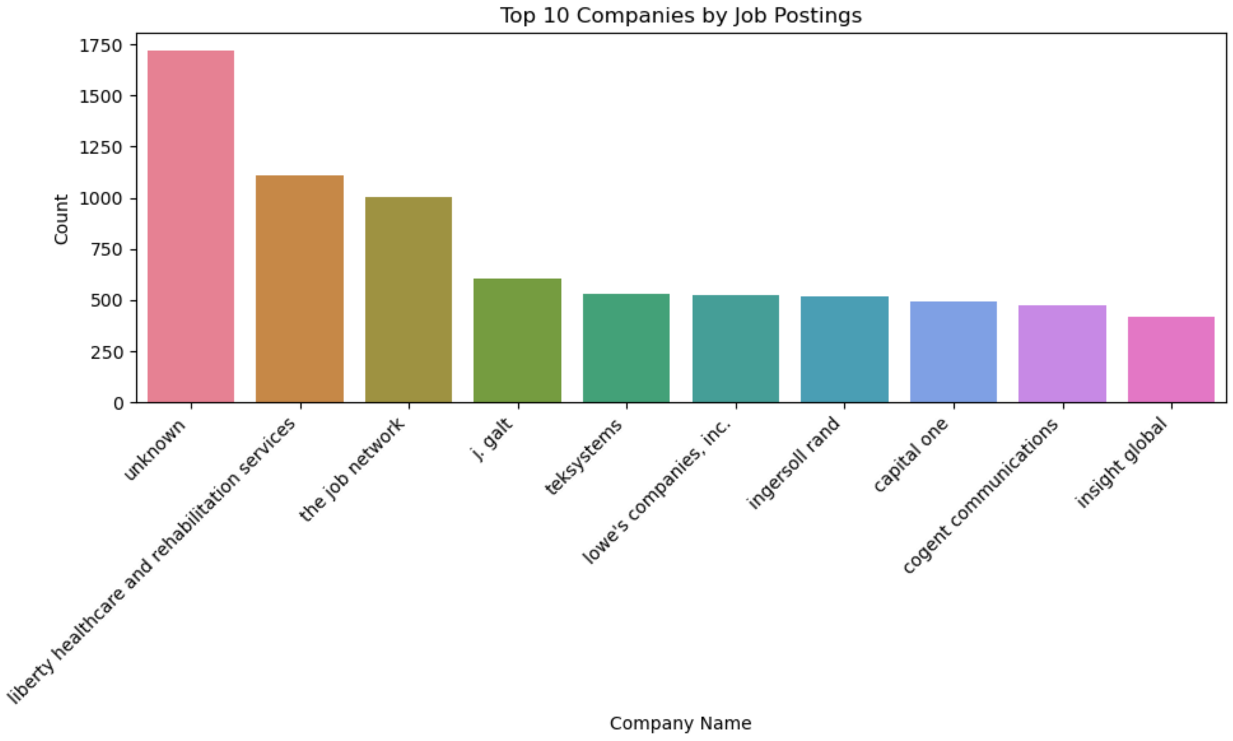


Sales Manager tops in job titles while there is high demand of **full-time** jobs.

Exploratory Data Analysis: Univariate Analysis

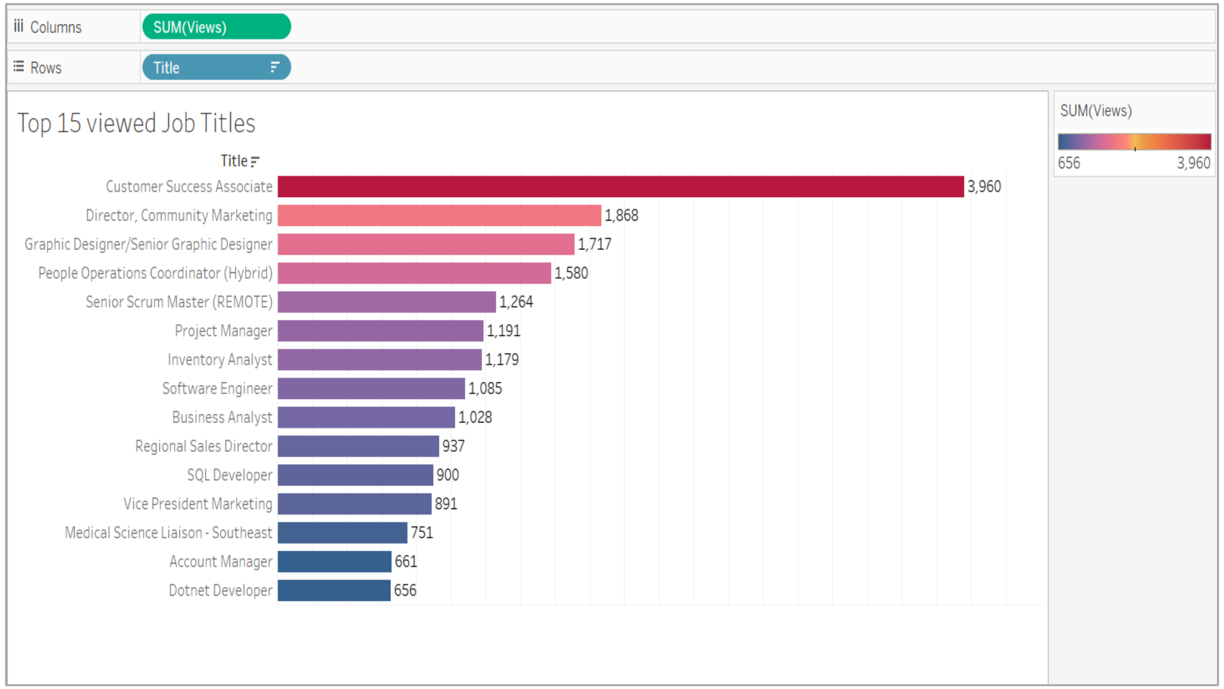
Experience Level Distribution



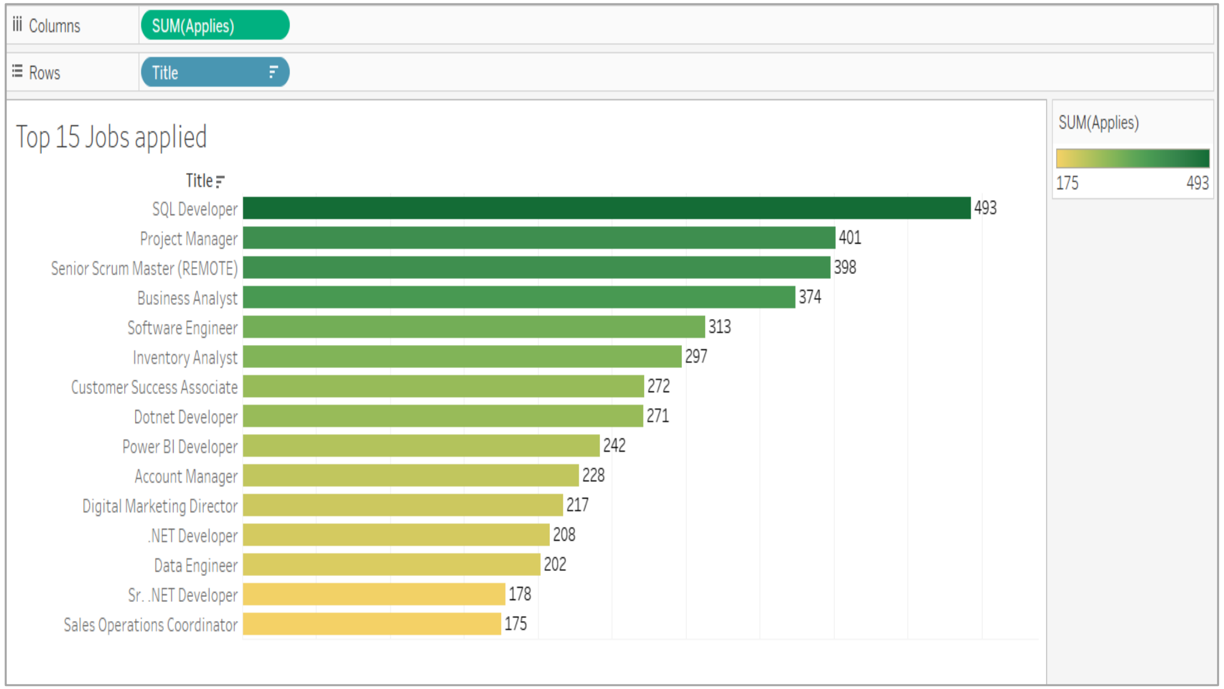


There is a high demand of **Mid-Senior Level** in job postings. On the other hand, most of the companies did not include their name (Unknown) when posting a job. This may reduce visibility in the pool of job seekers and result in very low applies.

Exploratory Data Analysis: Bivariate Analysis



n = 1000



n = 1000

Customer Success Associate seems to be a lucrative title for many job seekers garnering at least **3,960 views** with the second job title garnering nearly half less views.

SQL Developer has high demand across job seekers with at least 493 **applies.** Uniquely, most of the job seekers display a market growth in data science.

Exploratory Data Analysis: Bivariate Analysis

Work Type vs Experience level vs Pay Period

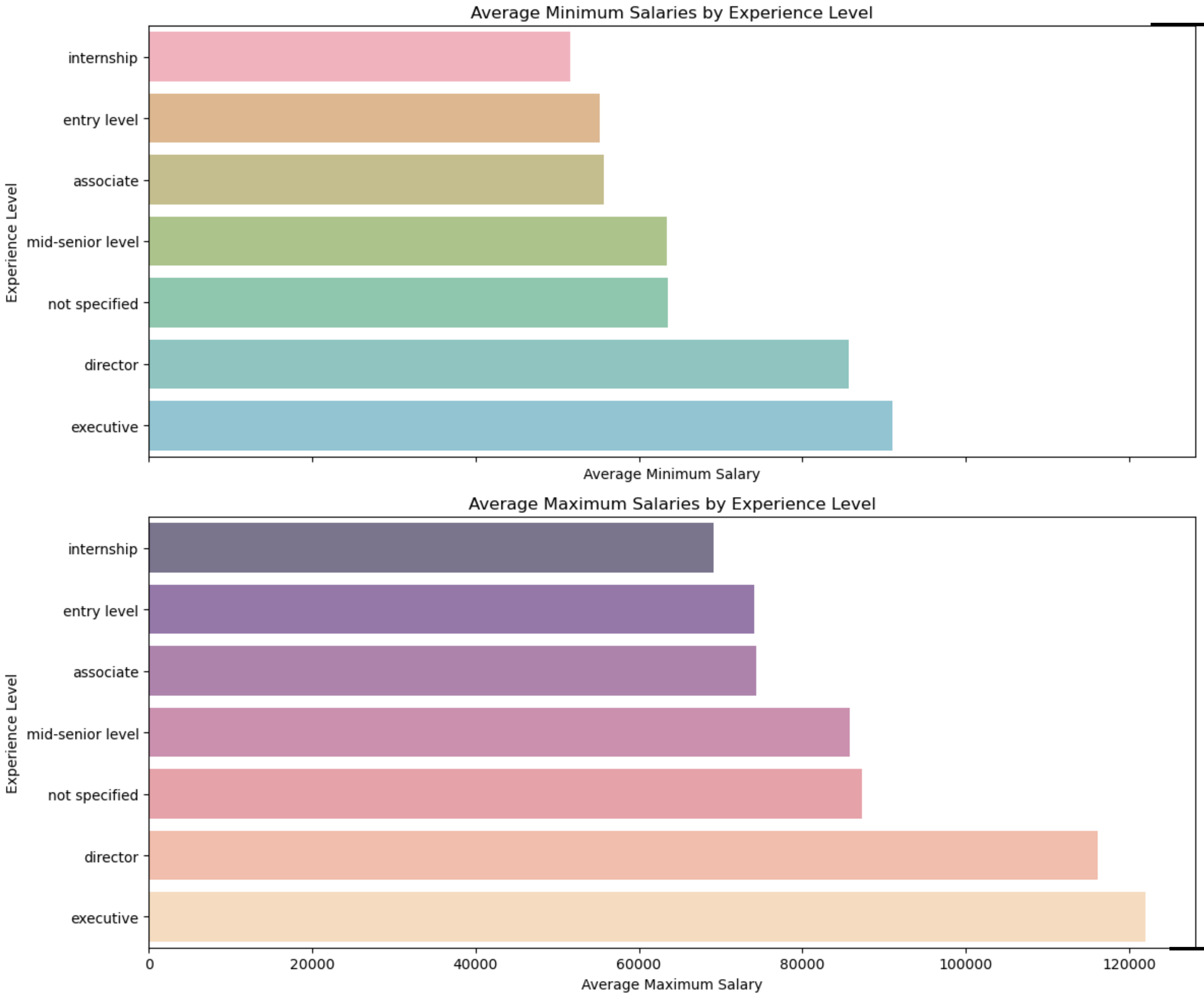
Formatted Experience Level	Pay Period HOURLY	Pay Period YEARLY
Mid-Senior level	6,621	1,430,398
Entry level	660	133,220
Associate	2,118	
Director	165	
Mid-Senior level	1,155	51,299,677
Director	75	10,926,691
Associate	807	10,788,222
Entry level	801	6,892,582
Executive	55	3,723,000
Internship	258	120,000
Internship	187	80,000
Entry level	24	
Mid-Senior level	3,337	225,643
Entry level	28	47,522
Associate	121	
Mid-Senior level	249	
Associate	76	

AGG(Average Salary): 24 – 51,299,677

n = 1000

Despite the experience level, most hiring personnel/companies offer jobs with an **hourly or yearly pay period,** and **mid-senior level** is attractively higher than other levels.

Exploratory Data Analysis: Bivariate Analysis



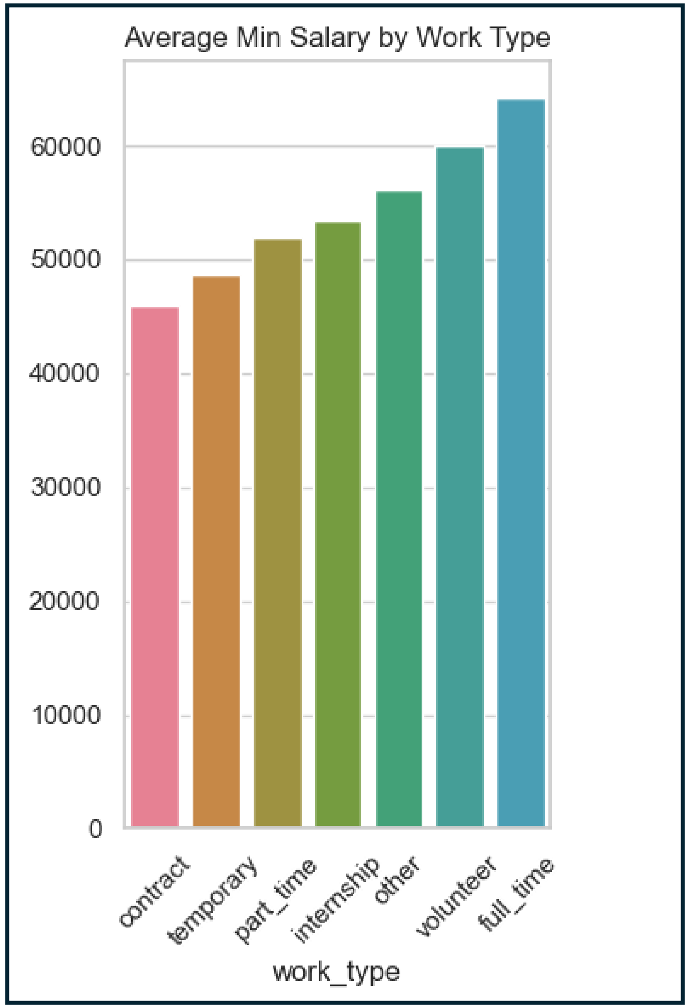
Average Minimum Salaries

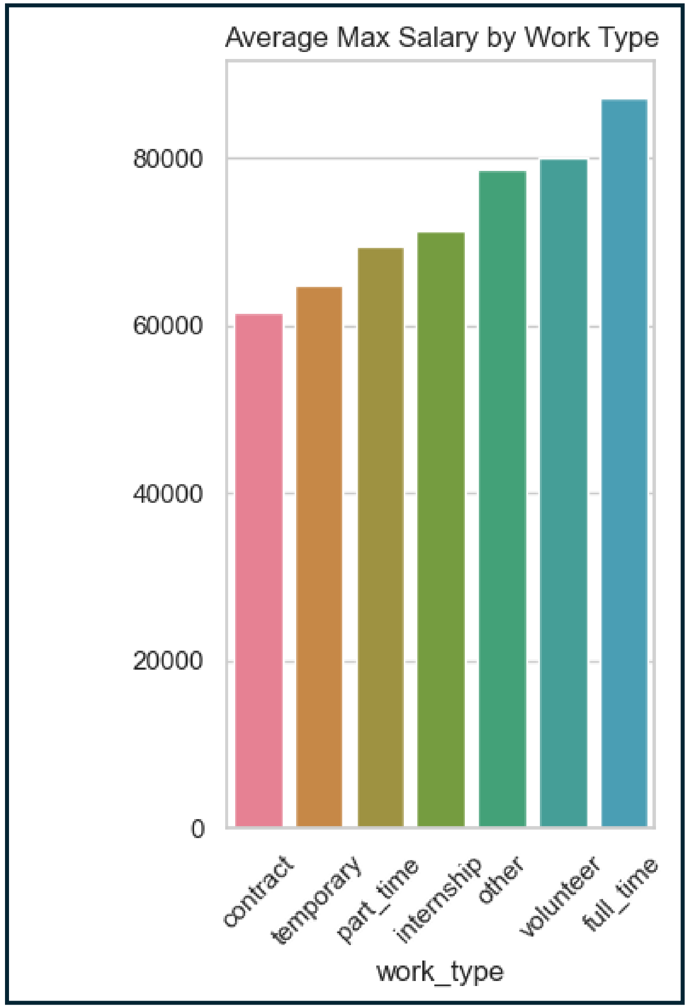
Average Maximum Salaries

n = 123849

Increase in both average **minimum and maximum salaries** is proportionate to **rise in experience level.**

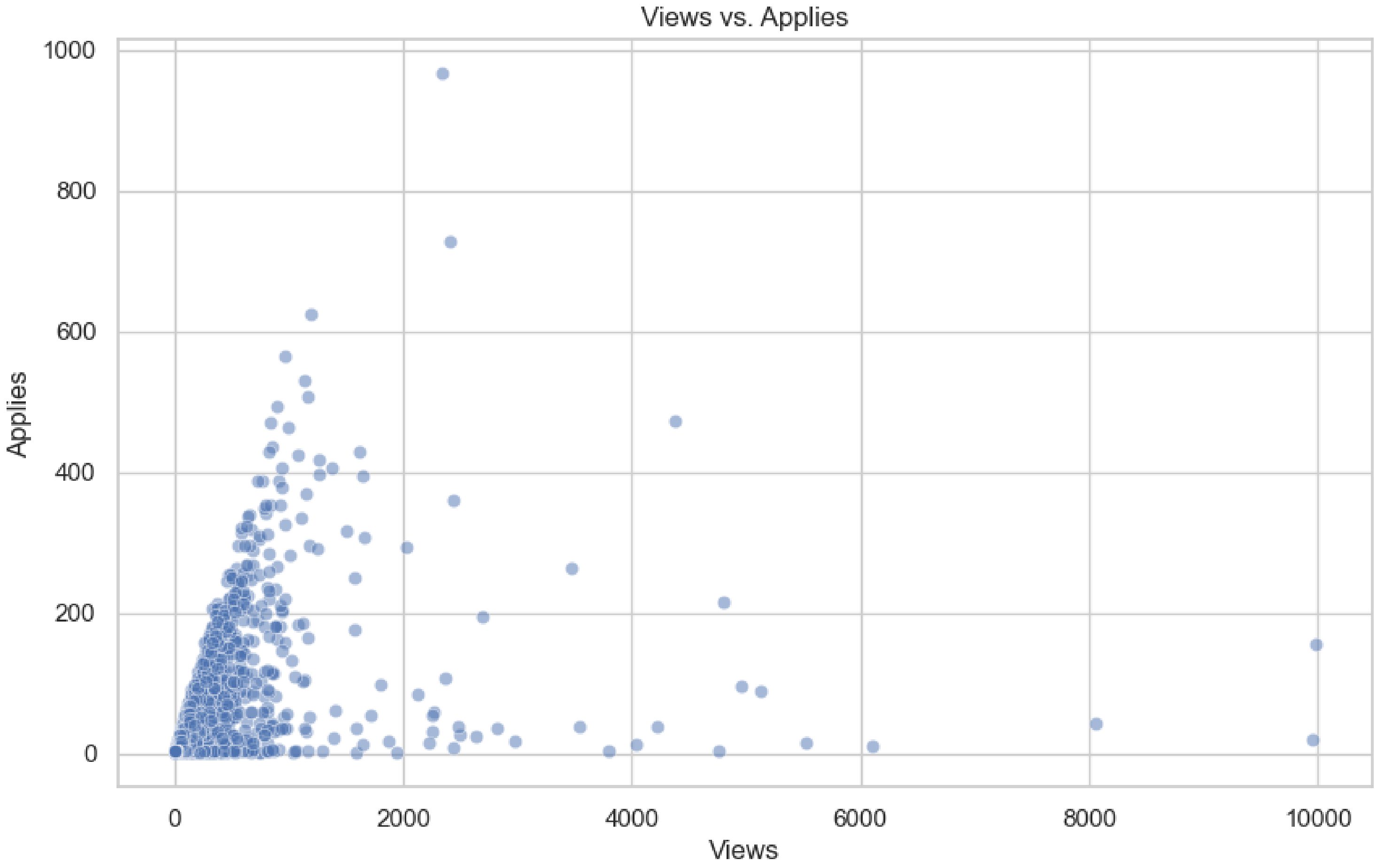
Exploratory Data Analysis: Bivariate Analysis





Jobs with **higher salaries** tend to offer **full-time positions**, while **contract roles** generally offer **lower salaries.**
This suggests that **full-time roles are valued more** highly or are more financially rewarding.

Exploratory Data Analysis: Bivariate Analysis



Although there is some correlation, increase in views does not necessarily suggest an increase in applies. Factors such salary, location, job title and experience level contribute to rate of views and applies.

Exploratory Data Analysis: Multivariate Analysis

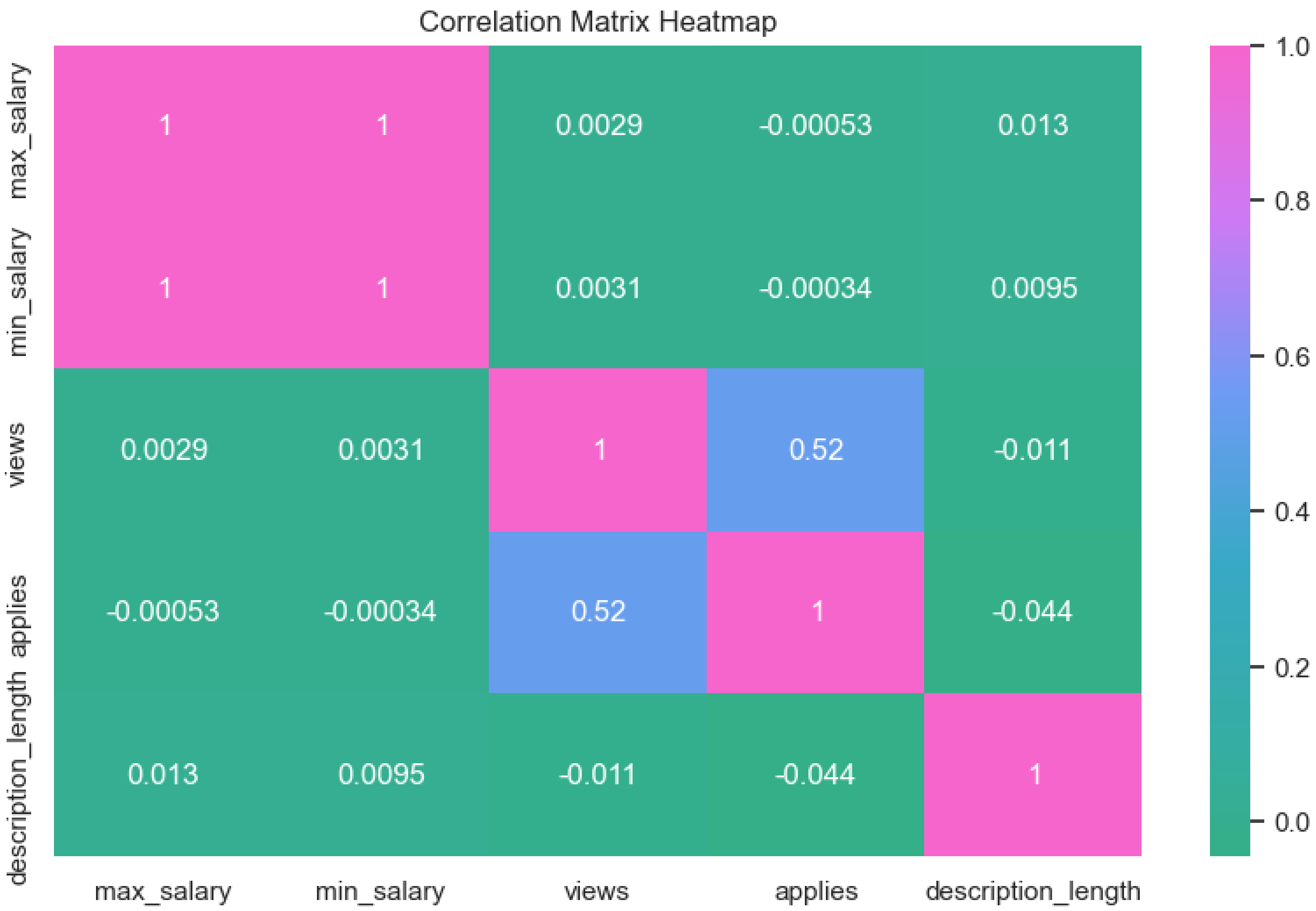
Word Cloud of Job Titles

Word Cloud of Job location

Hiring companies seem to have a demand of job titles such as Project manager, Director, Sales Associate, Customer service, Nurses.

Majority of the jobs are offered in New York, Metropolitan area, Chicago, Illinois, Atlanta, and Houston Texas.

Exploratory Data Analysis: Multivariate Analysis



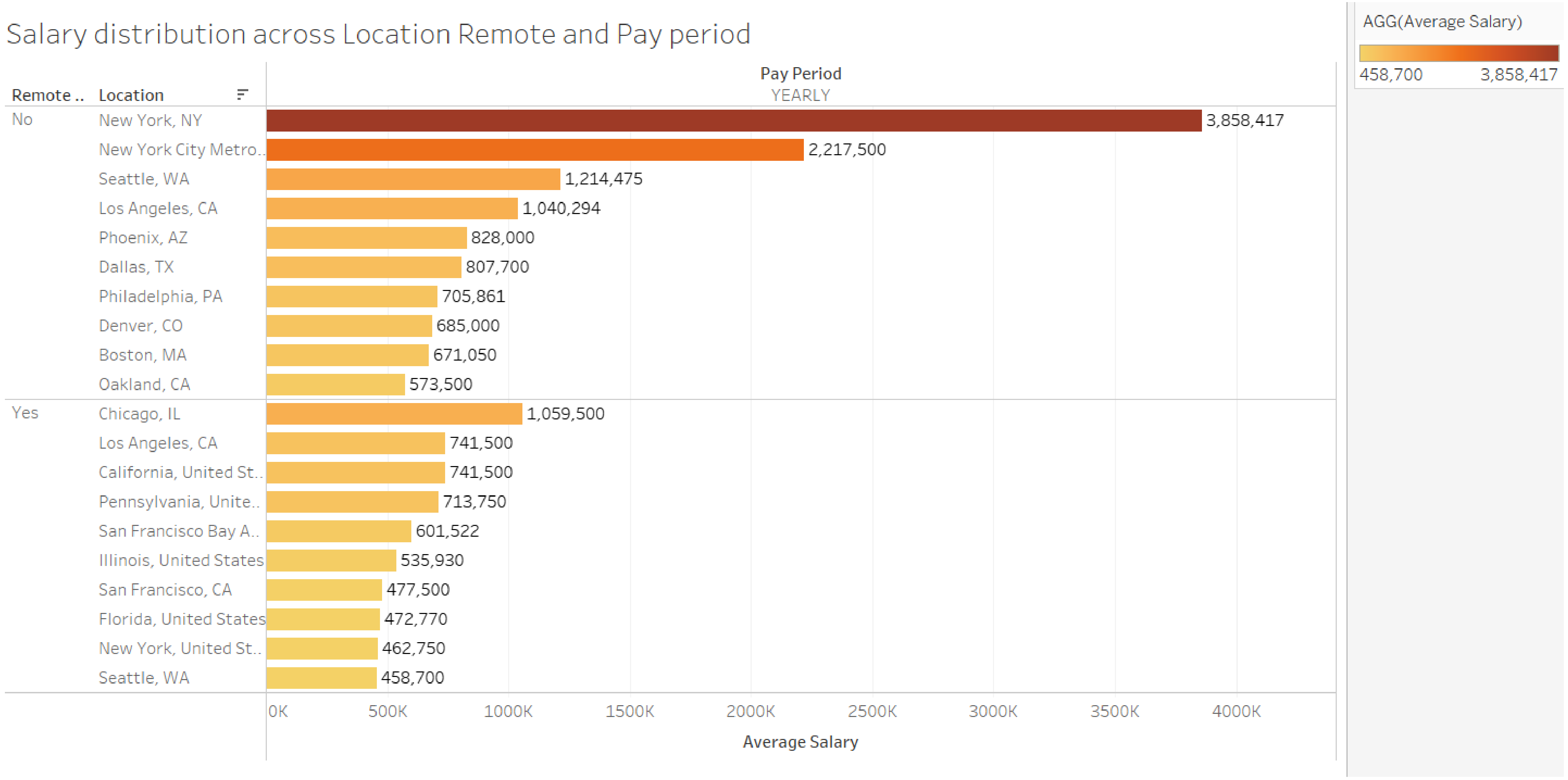
Low Correlations: Most of the correlations between the variables are very close to zero, indicating weak or no linear relationships.

Max_salary and min_salary: These variables have a perfect positive correlation (1.0), which is expected as maximum salary is always greater than or equal to minimum salary.

Views and Applies: There's a moderate positive correlation (0.52) between the number of views and applications. This suggests that job postings with more views tend to receive more applications.

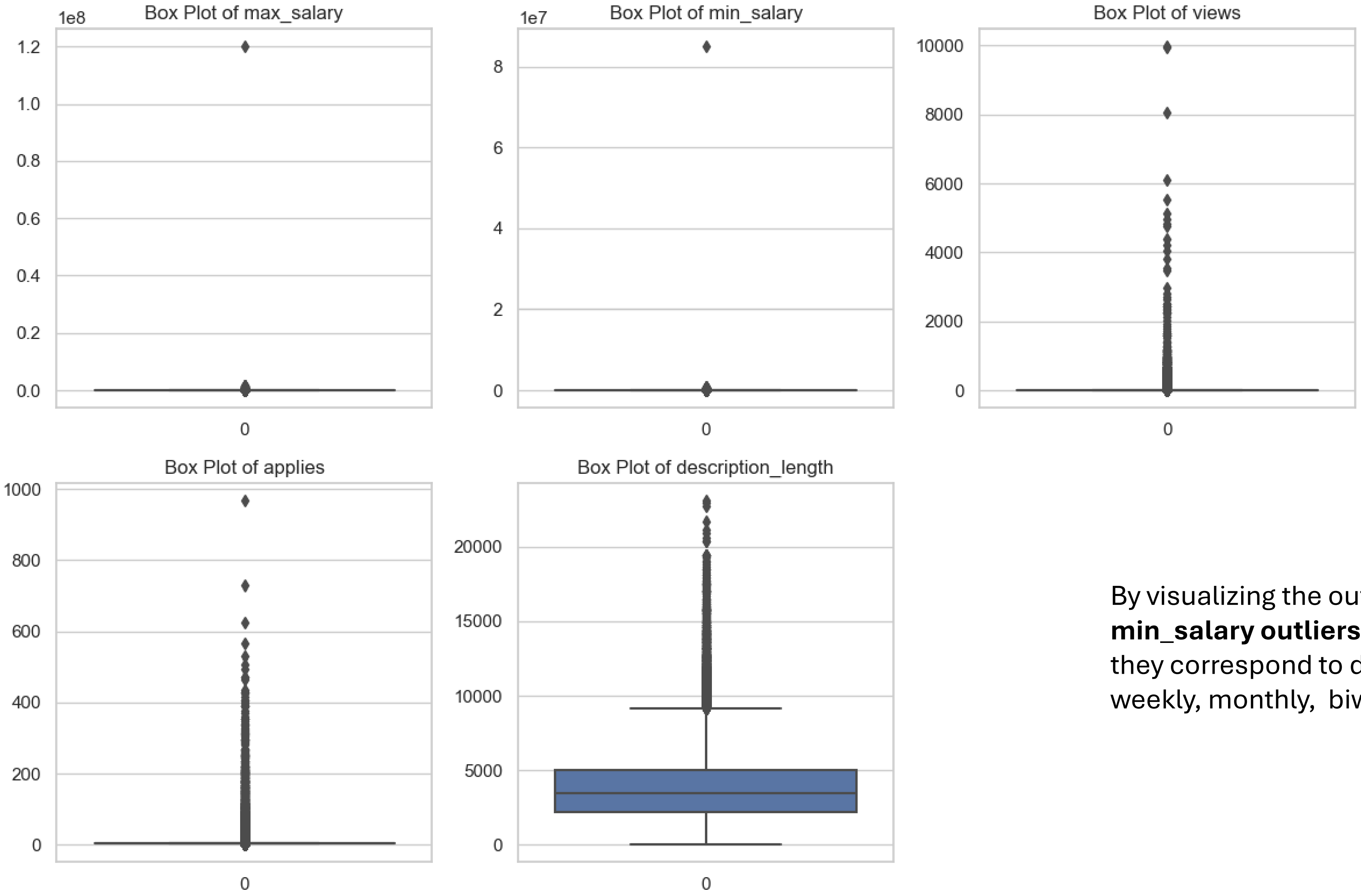
Description Length and Applies: There's a weak negative correlation (-0.044) between description length and the number of applications. This could imply that longer job descriptions might slightly discourage applications, but the relationship is very weak.

Exploratory Data Analysis: Multivariate Analysis



More jobs on demand that don't offer remote working environment are largely offered in New York, while Chicago Illinois offers most jobs with remote working.

Exploratory Data Analysis: Handling Outliers



By visualizing the outliers, **max_salary and min_salary outliers were not be dropped** since they correspond to different pay periods i.e. yearly, weekly, monthly, biweekly, and hourly.

Feature Engineering

We used feature engineering to transform raw data into features that can be used in machine learning models.

In [ ]:
```
# Engineering the new feature 'average_salary'
df['average_salary'] = (df['max_salary'] + df['min_salary']) / 2

# Dropping the 'min_salary' and 'max_salary' columns
df.drop(columns=['min_salary', 'max_salary'], inplace=True)
```

A sample of feature engineering the max and min salaries to average salary.

title	description	location	views	applies	formatted_experience_level	listed_time	work_type	currency	description_length	average_salary
marketing coordinator	job descriptiona leading real estate firm in n...	princeton, nj	20	2	not specified	2024-09-04 15:27:04.559352832	full_time	KSH	2525	18
ıental health ist/counselor	at aspen therapy and wellness , we are committ...	fort collins, co	1	3	not specified	2024-08-01 14:49:32.302924544	full_time	KSH	3560	40
ıt restaurant manager	the national exemplar is accepting application...	cincinnati, oh	8	3	not specified	2024-07-10 23:26:26.855408128	full_time	KSH	460	55,000
ɔr elder law / . and estates associat...	senior associate attorney - elder law / trusts	new hyde park, ny	16	3	not specified	2024-08-28 16:27:10.837676544	full_time	KSH	1594	157,500

A sample data frame after feature engineering max and min salaries.

Feature Engineering: data types conversion

```
# Convert 'listed_time' to a numerical feature (e.g., days since listing)
df['listed_time'] = pd.to_datetime(df['listed_time'])
df['days_since_listed'] = (pd.Timestamp.now() - df['listed_time']).dt.days

# Drop the original 'listed_time' column
df = df.drop(columns=['listed_time'])

# Convert object columns to categorical
categorical_columns = ['job_id', 'company_name', 'title', 'description', 'location', 'formatted_experience_level', 'work_type
df[categorical_columns] = df[categorical_columns].astype('category')

# Convert float64 columns to int
# Handle possible missing values before conversion by filling or dropping as appropriate
df['views'] = df['views'].fillna(0).astype(int)
df['applies'] = df['applies'].fillna(0).astype(int)
df['average_salary'] = df['average_salary'].fillna(0).astype(int)

# Check data types after conversion
print("Data types after conversion:")
print(df.dtypes)
```

More feature engineering on selected columns by converting them to relevant data types suitable for analysis and modeling.

Data Preprocessing

1.Encode categorical columns

```
[ ]: from sklearn.preprocessing import LabelEncoder

# Identify categorical columns
categorical_columns = ['formatted_experience_level', 'work_type', 'currency']

# Initialize LabelEncoders for each categorical column
label_encoders = {}
for col in categorical_columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

Encoding is essential to convert categorical data into a numerical format that the model can understand.

2.Standardise Numerical Columns

```
from sklearn.preprocessing import StandardScaler

# Identify numerical column
numerical_features = ['views','applies','description_length','average_salary']

# Innitialise standard scaller
scaler = StandardScaler()

df[numerical_features] = scaler.fit_transform(df[numerical_features])
```

Standardization is a valuable tool for preprocessing numerical data in machine learning, for feature scaling, outlier handling, algorithm performance, and better interpretation of Coefficients.

Data Preprocessing/2

3.Tokenise Text Columns using NLP

```python
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Download necessary NLTK data
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

# Initialize lemmatizer and stopwords list
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Define text preprocessing function
def preprocess_text(text):
    if isinstance(text, str):  # Ensure the text is a string
        tokens = word_tokenize(text.lower())  # Tokenize and lower case
        tokens = [lemmatizer.lemmatize(token) for token in tokens if token.isalpha() and token not in stop_words]
        return ' '.join(tokens)
    return text  # Return text as is if it's not a string

# Specify the columns to preprocess
text_columns = ['company_name', 'title', 'location', 'description']

# Apply preprocessing to the specified columns
for column in text_columns:
        df[f'processed_{column}'] = df[column].apply(preprocess_text)
```

Word Tokenization

This approach helps ensure that the dataset is ready for binary classification tasks, allowing us to analyze and model the likelihood of job postings receiving high or low numbers of applications.

Data Preprocessing: Principal Component Analysis(PCA)

```
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA


# Create a pipeline with preprocessing and PCA
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('pca', PCA(n_components=0.95))  # Retain 95% of variance
])

# Fit and transform the data
pca_transformed_data = pipeline.fit_transform(df)

# Get the number of components
num_components = pipeline.named_steps['pca'].n_components_

# Create DataFrame from the transformed data
df_pca = pd.DataFrame(pca_transformed_data, columns=[f'PC{i+1}' for i in range(num_components)])

# Display the updated DataFrame
print("PCA-transformed DataFrame:")
print(df_pca.head())
```

```
PCA-transformed DataFrame:
   PC1  PC2  PC3  PC4  PC5
0    2   -0   -1    0   -0
1    2   -0   -0   -0    0
2    2   -0   -1    1   -0
3    2   -0   -1    1   -0
4    2   -0   -1    1   -0
```

PCA was performed by creating a pipeline (preprocessor & pca). This step performs PCA with **n_components=0.95**, meaning it aims to retain 95% of the variance in the data.

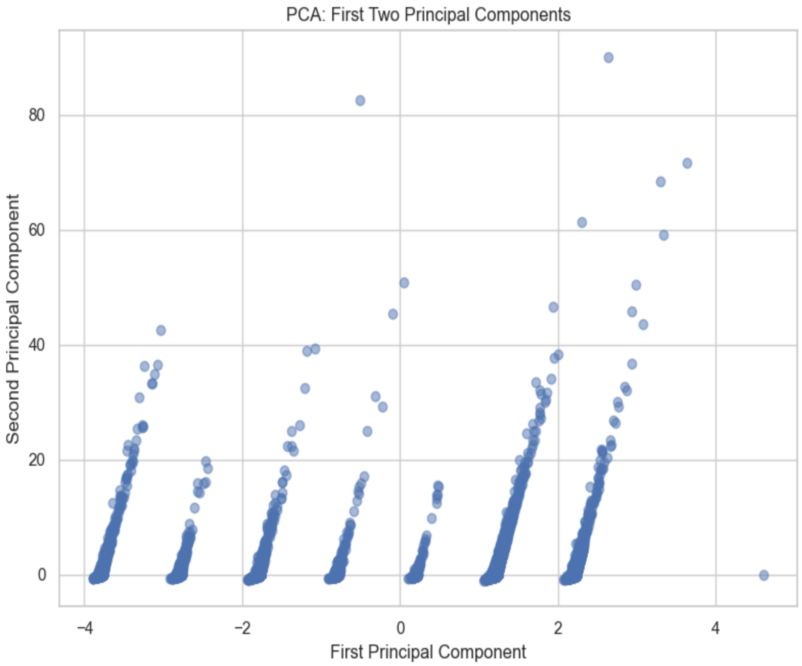
```
# Extract the PCA component from the pipeline
df_pca= pipeline.named_steps['pca']

# Explained variance by each principal component
explained_variance = df_pca.explained_variance_ratio_

# Cumulative explained variance
cumulative_explained_variance = np.cumsum(explained_variance)

# Print explained variance
print("Explained variance by each component:\n", explained_variance)
print("Cumulative explained variance:\n", cumulative_explained_variance)
```

```
Explained variance by each component:
 [0.47832477 0.18272502 0.12231278 0.11891283 0.0577925 ]
Cumulative explained variance:
 [0.47832477 0.66104979 0.78336258 0.90227541 0.96006791]
```

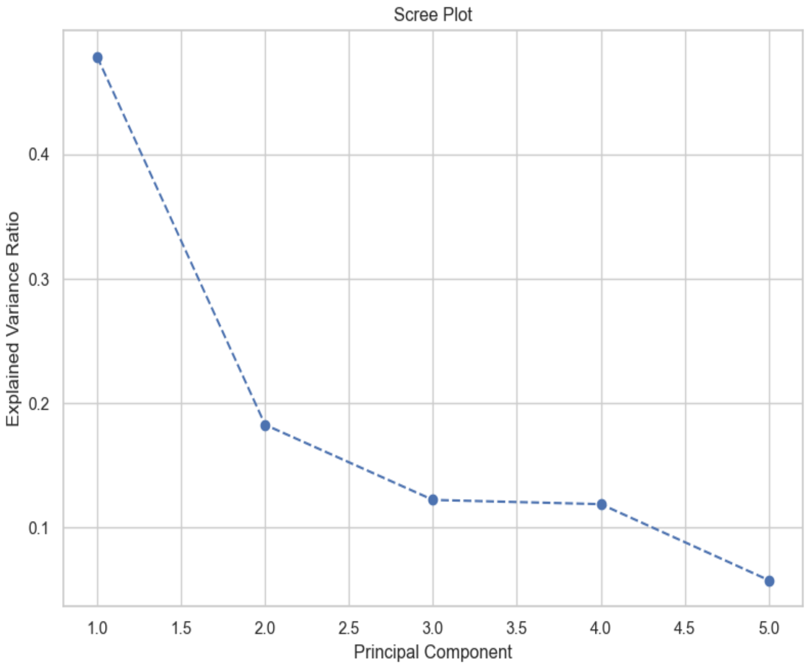
Cumulative Variance: By using the first five principal components, **it explains 96.01%** of the variance in the data.

This means we can reduce the **dimensionality of our data from its original number of features to 5 principal components** without losing much information.

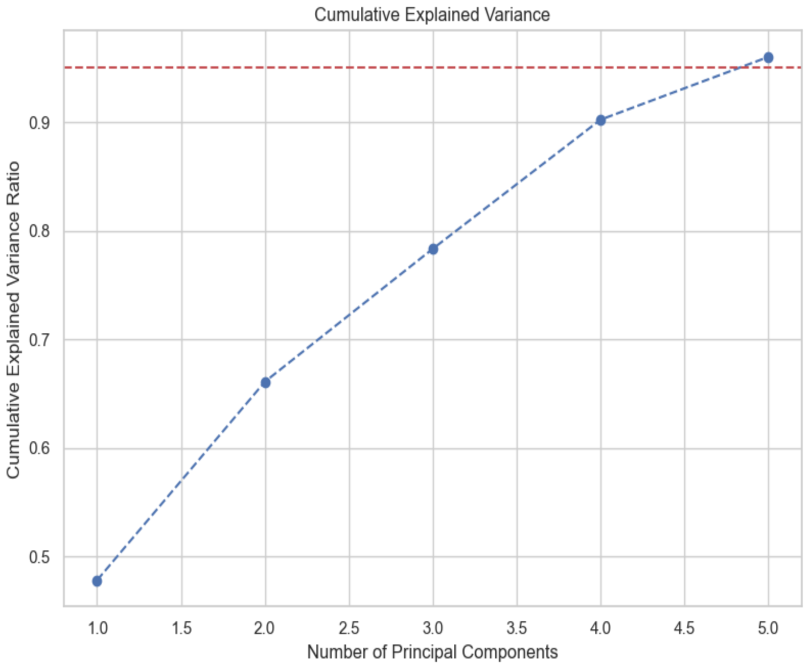
Data Preprocessing: Principal Component Analysis(PCA)/2



Scatter Plot: Displays the relationship between the first two principal components.



Scree Plot: Shows the explained variance ratio for each principal component.



Cumulative Explained Variance Plot: Illustrates the cumulative explained variance as the number of principal components increases.

The PCA results indicate that a significant portion of the data's variability can be explained by a relatively small number of principal components.

Feature Selection: Predictor Data Frame

1.Predictor Dataframe

[ ]:

```
# Create a copy of the DataFrame for prediction
predictor_df = df[['views', 'description_length', 'average_salary', 'formatted_experience_level',
                   'days_since_listed', 'work_type']].copy()

# Create the target variable 'high_applications'
median_applies = df['applies'].median()  # Calculate the median of 'applies'
predictor_df['high_applications'] = (df['applies'] > median_applies).astype(int)  # Assign to the copy

# Display the predictor DataFrame
predictor_df.head()
```

Out[71]:

	views	description_length	average_salary	formatted_experience_level	days_since_listed	work_type	high_applications
0	0	-1	-0	6	-29	1	0
1	-0	-0	-0	6	5	1	0
2	-0	-2	-0	6	27	1	0
3	0	-1	0	6	-22	1	0
4	-0	-2	-0	6	-7	1	0

The predictor Data Frame is the cornerstone of the machine learning model's deployment. It encapsulates the features or inputs that a model uses to make predictions.

Feature Selection: Recommender Data Frame

2.Recommender Dataframe

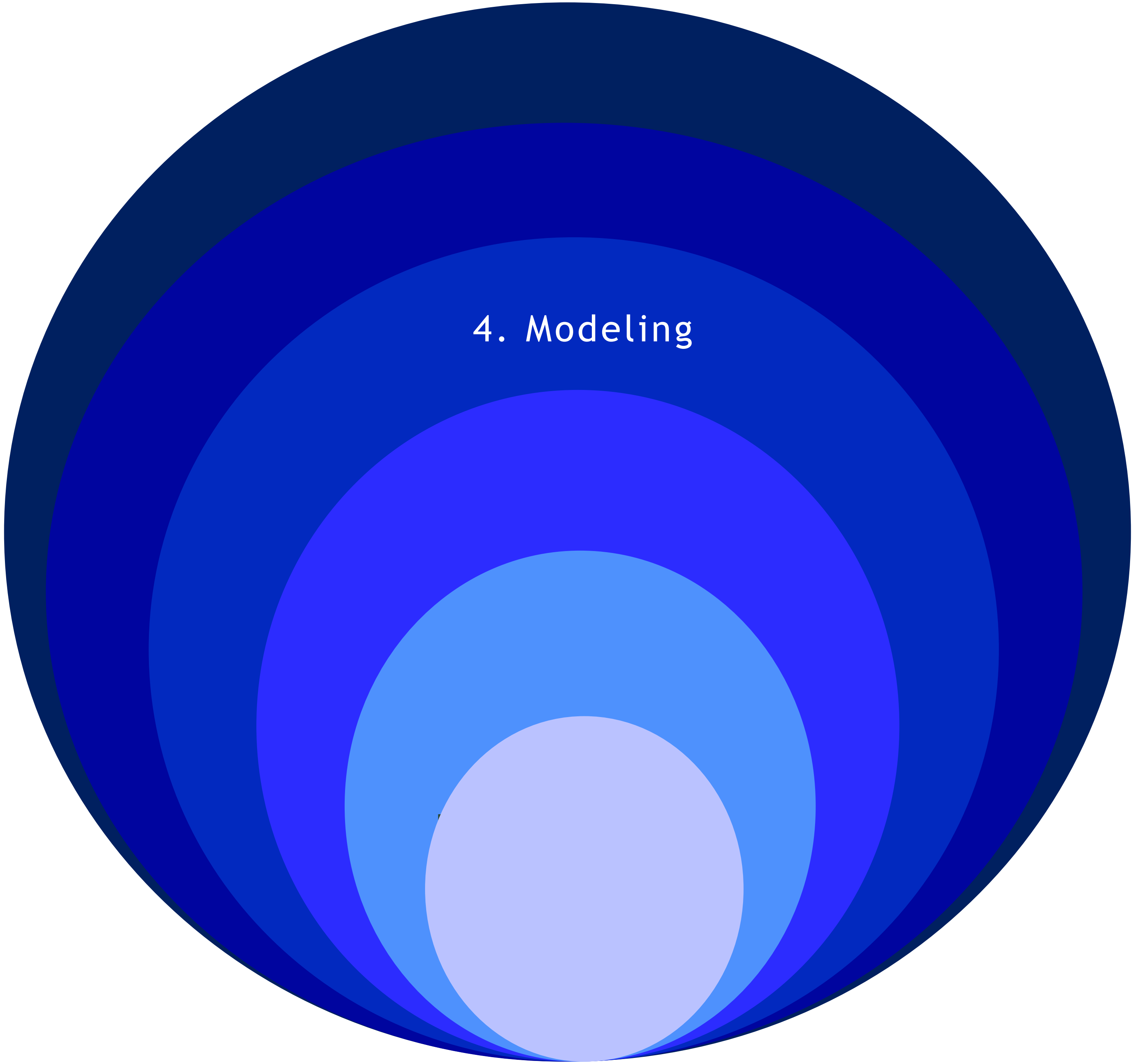
[ ]:

```python
# Features
recommender_df = df[['job_id','processed_title', 'processed_description', 'processed_location', 'views','applies',
                     'processed_company_name','work_type', 'average_salary']]

# Display the recommender DataFrame
recommender_df.head()
```

Out[72]:

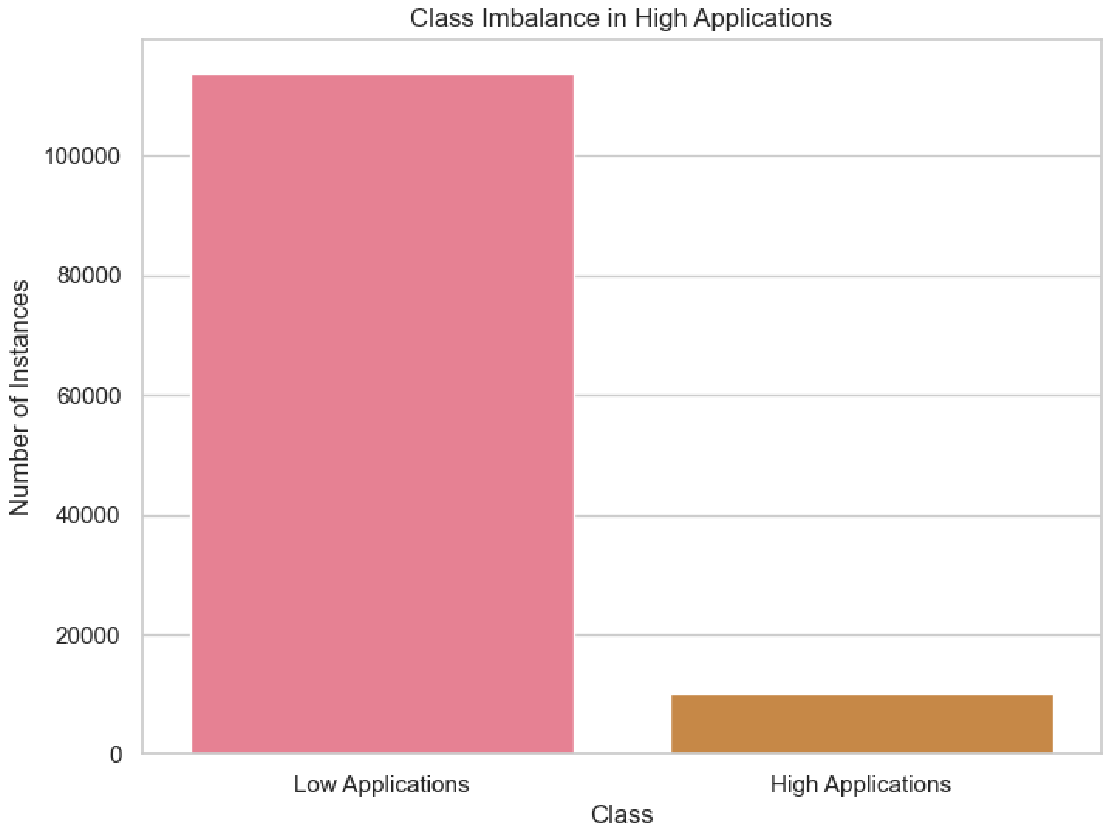
	job_id	processed_title	processed_description	processed_location	views	applies	processed_company_name	work_type	average_salary
0	921716	marketing coordinator	job descriptiona leading real estate firm new ...	princeton nj	0	-0	corcoran sawyer smith	1	-0
1	1829192	mental health	aspen therapy wellness committed serving clien...	fort collins co	-0	-0	unknown	1	-0
2	10998357	assitant restaurant manager	national exemplar accepting application assist...	cincinnati oh	-0	-0	national exemplar	1	-0
3	23221523	senior elder law trust estate associate attorney	senior associate attorney elder law trust esta...	new hyde park ny	0	-0	abrams fensterman llp	1	0
4	35982263	service technician	looking hvac service tech experience commerica...	burlington ia	-0	-0	unknown	1	-0

The Recommender Data Frame is a crucial component in building a recommendation system. It serves as the foundation for capturing and storing the necessary data to generate personalized recommendations.


4. Modeling

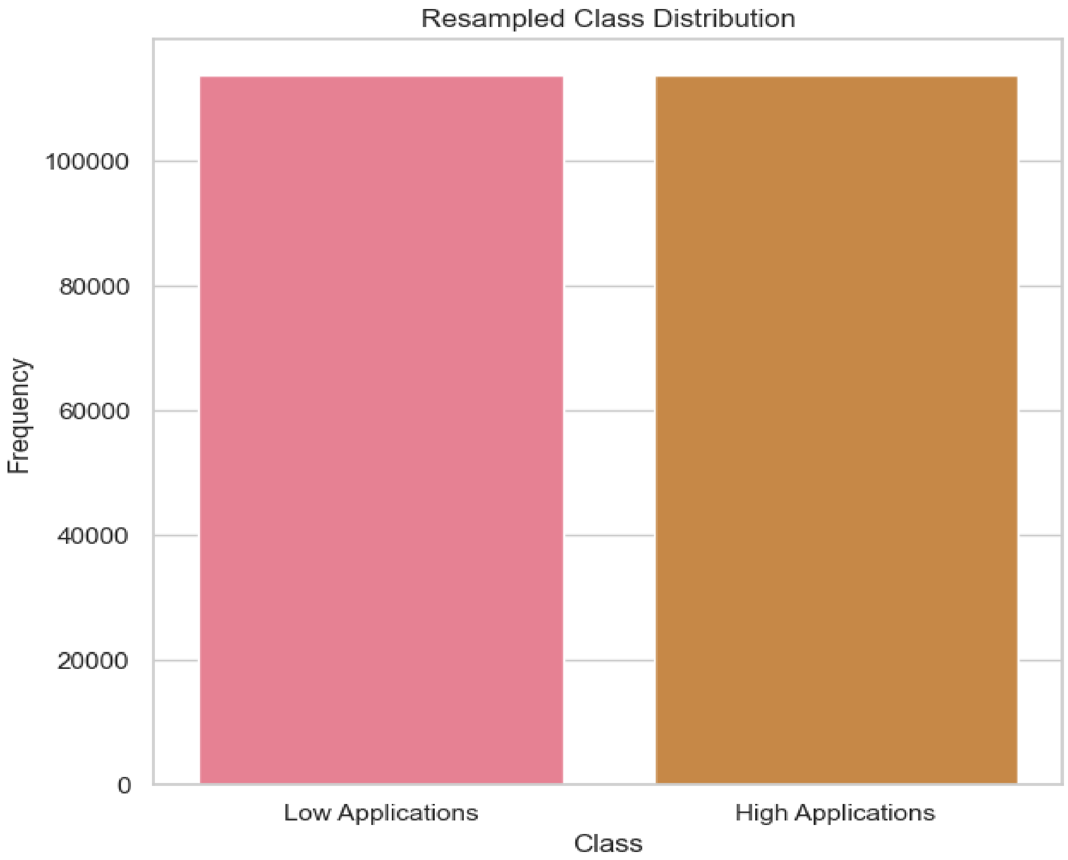
Modeling: Predictor Model

With Class Imbalance



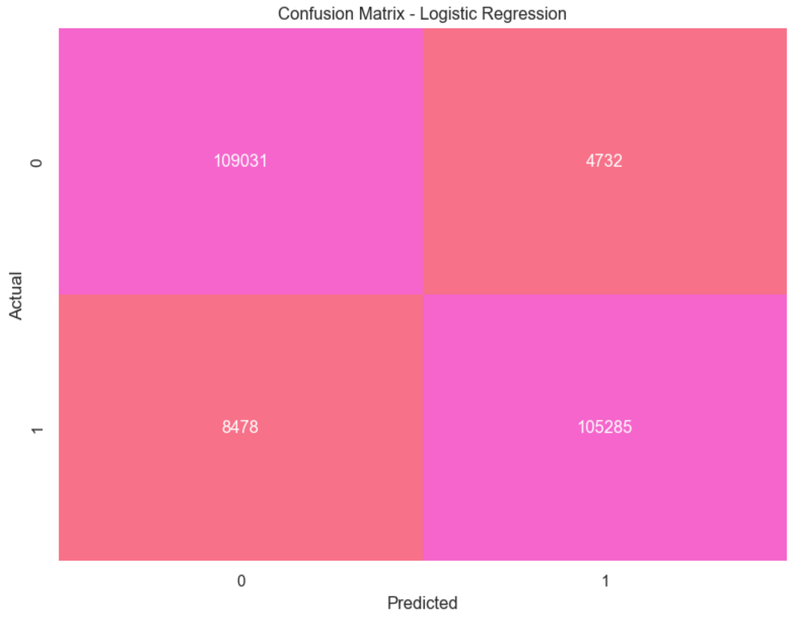
About 8.14% (10,086 out of 123,849) of the job postings received a high number of applications, while the remaining 91.86% (113,763 out of 123,849) did not.

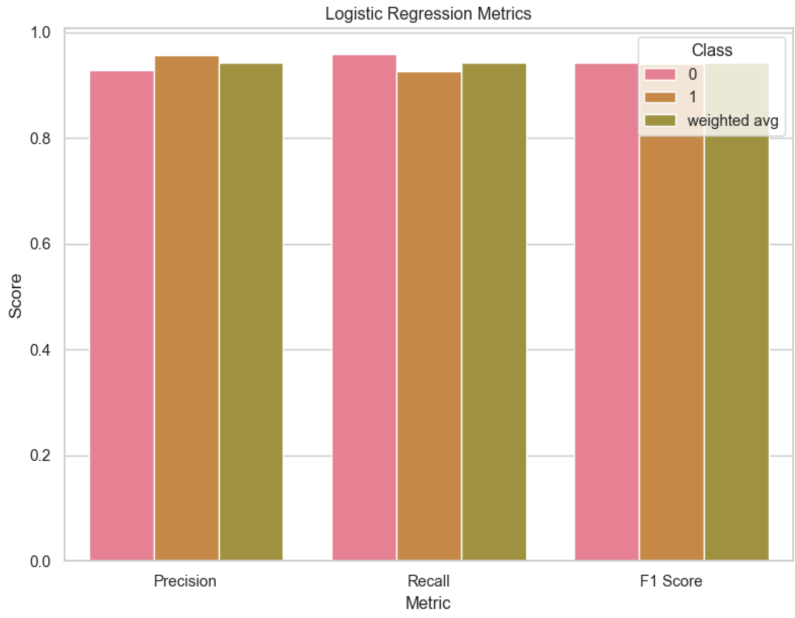
Handling Class Imbalance

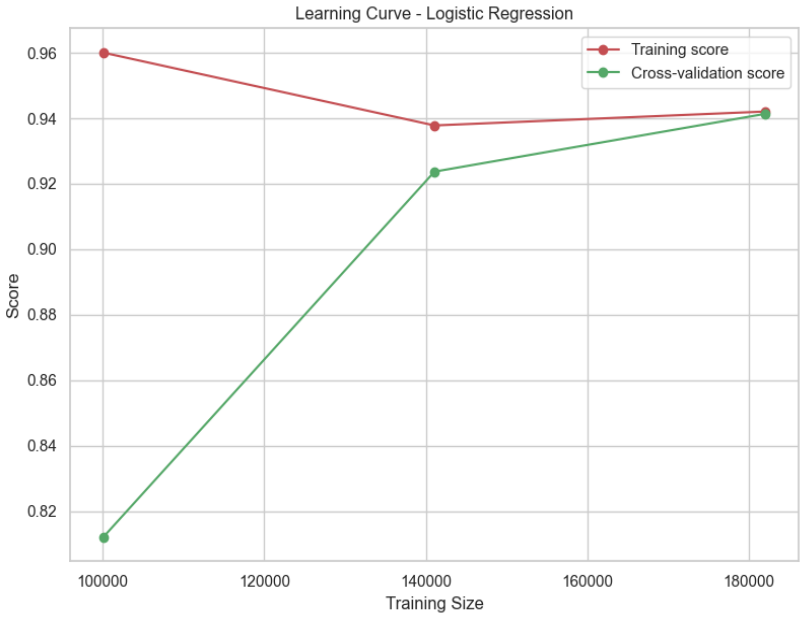


After applying **SMOTE**, the dataset now has an equal number of instances for both **'Low Applications'** and **'High Applications',** effectively addressing the class imbalance issue present in the original dataset

Modeling: Predictor Model – Baseline Modeling







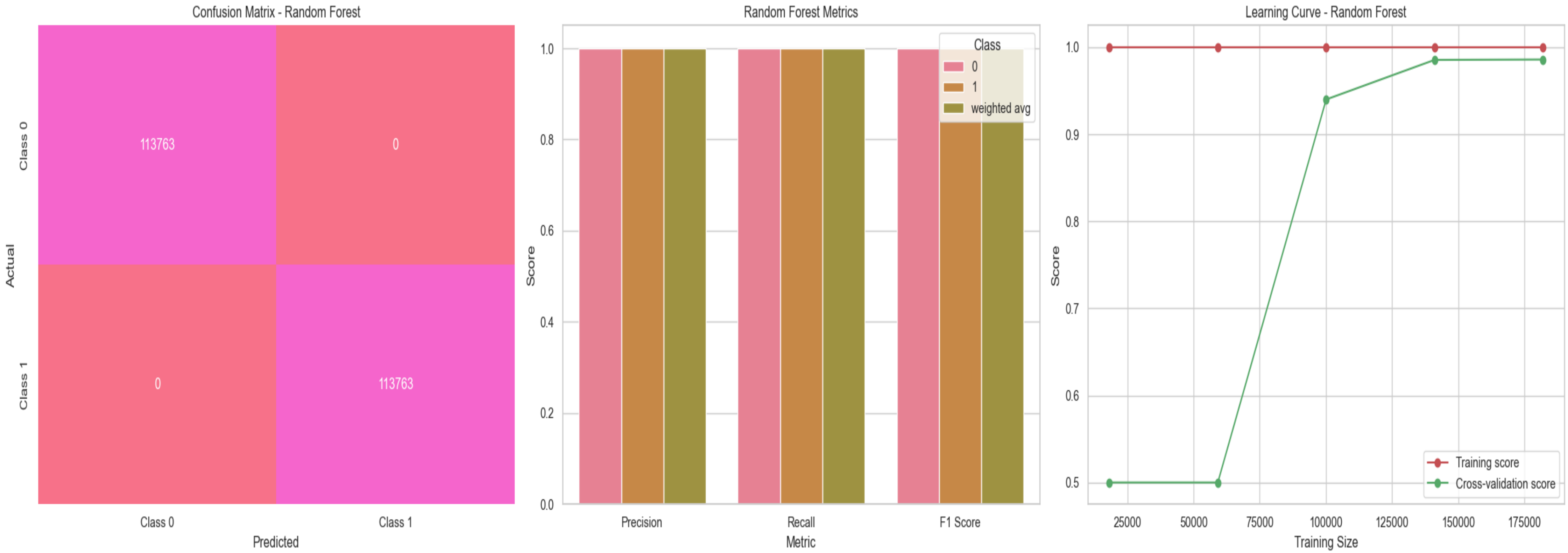
F1-Score: Both classes: 0.94 ROC-AUC Score: 0.9879

Precision: Class 0: 0.93 Class 1: 0.96

Recall: Class 0: 0.96 Class 1: 0.93

The logistic regression model is highly accurate and reliable in predicting the target variable

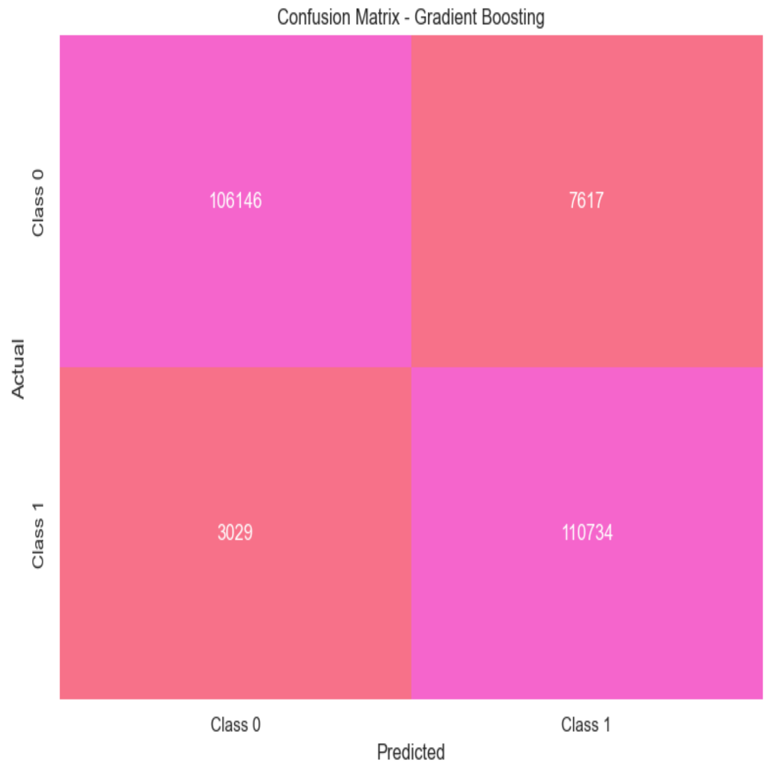
Modeling: Predictor Model – RandomForestClassifier

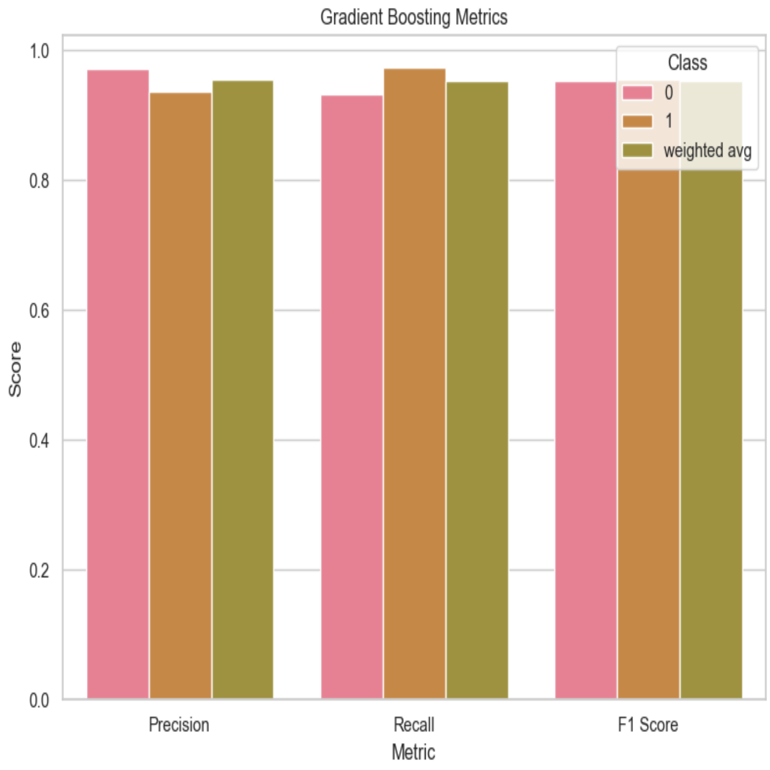


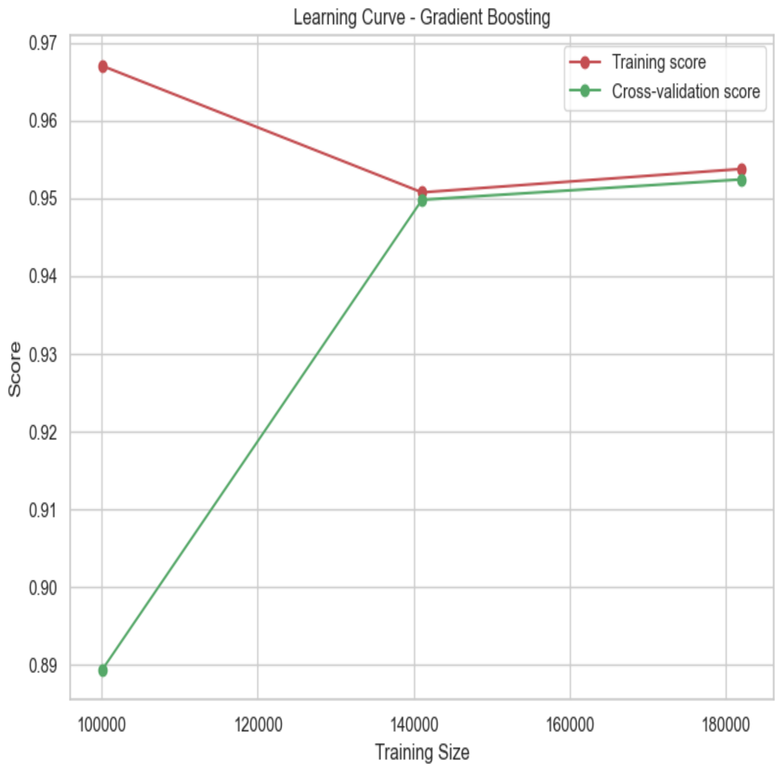
The **Random Forest model** shows exceptional performance with perfect accuracy, precision, recall, F1-scores, and an almost perfect ROC-AUC score.

The model achieved 100% accuracy on the test set, meaning it correctly predicted all instances.

Modeling: Predictor Model – GradientBoostingClassifier



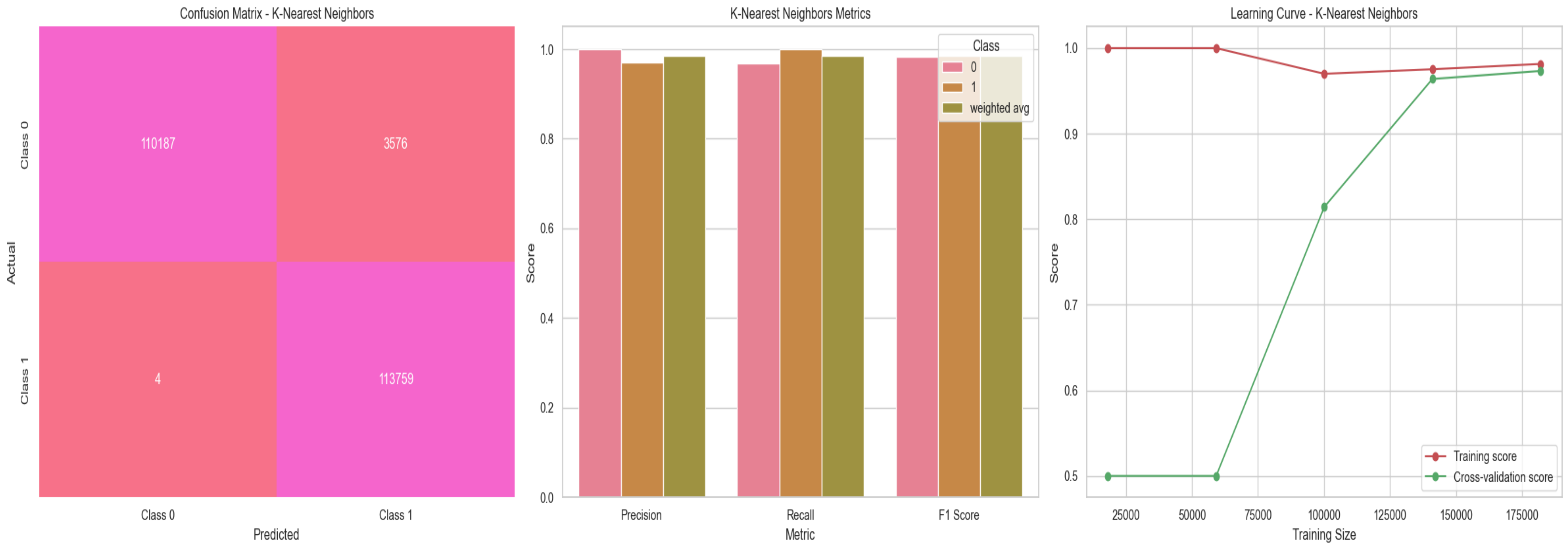




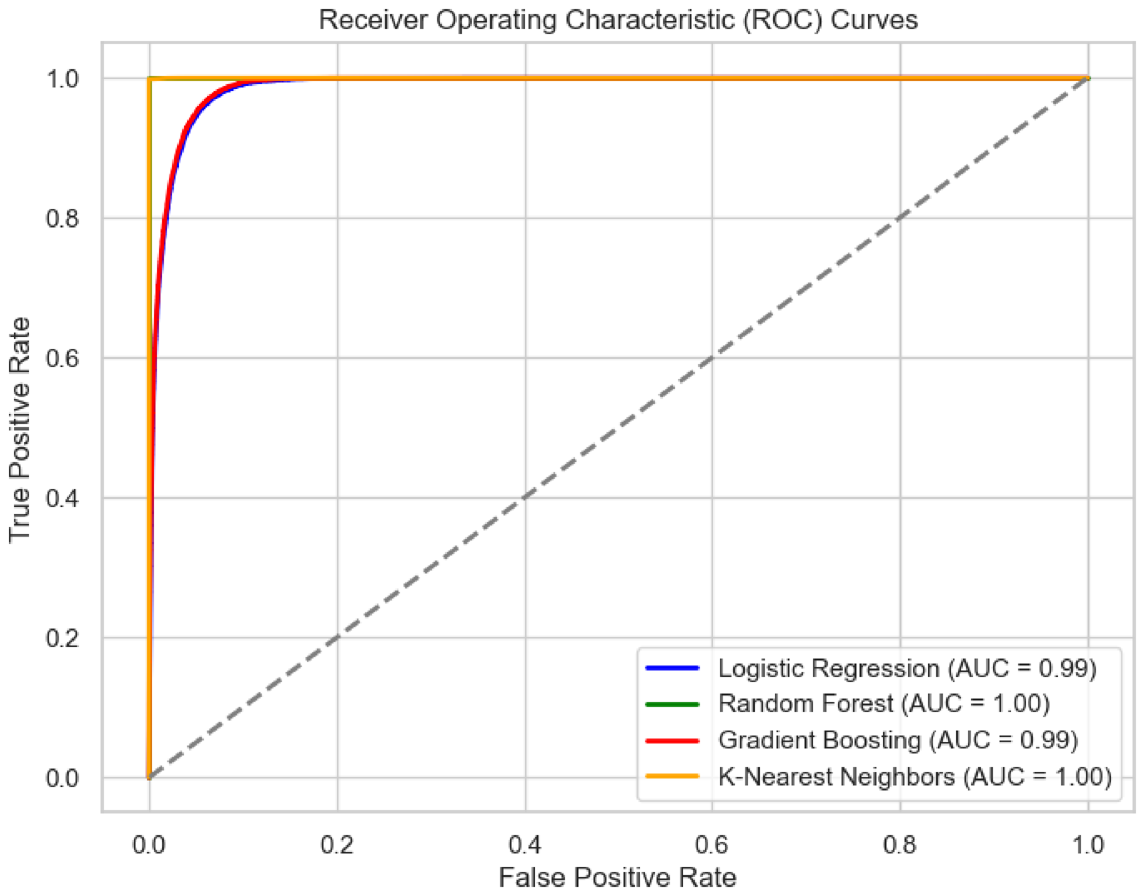
The K-Nearest Neighbors model shows exceptional performance with a cross-validation accuracy of approximately 97.28% and a test accuracy of approximately 98.41%.

These results suggest that the KNN model is well-tuned and capable of making accurate predictions.

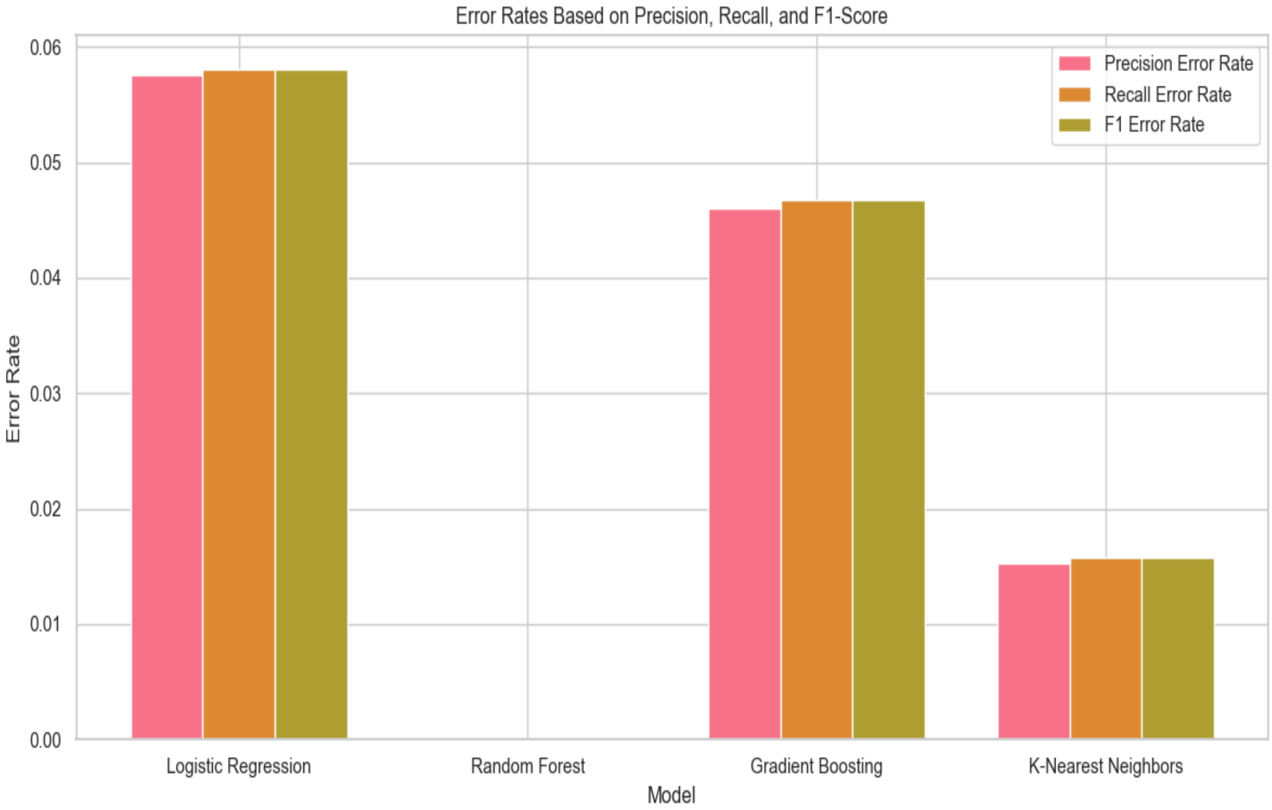
Modeling: Predictor Model – KNeighborsClassifier



Modeling: Predictor Model – Receiver Operating Characteristic (ROC) Curves for the Models & Errors on Precision,, Recall, F1-Score

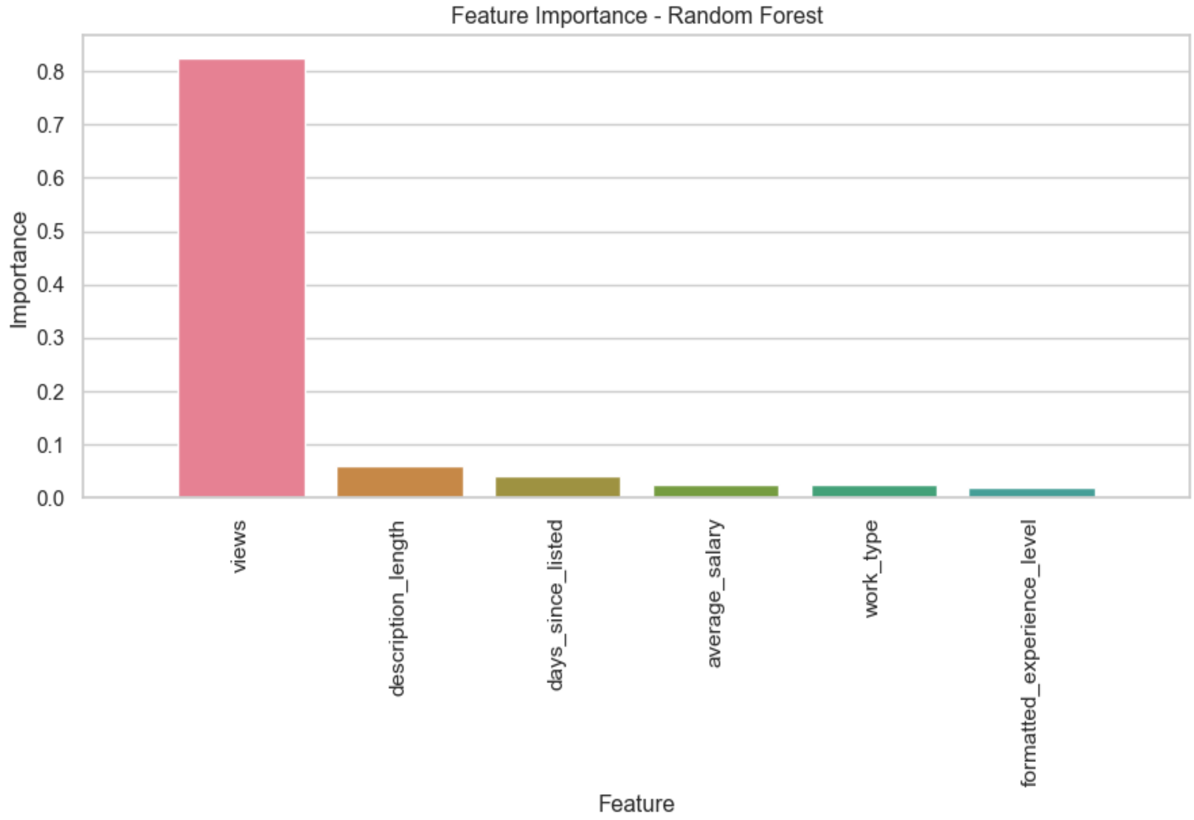


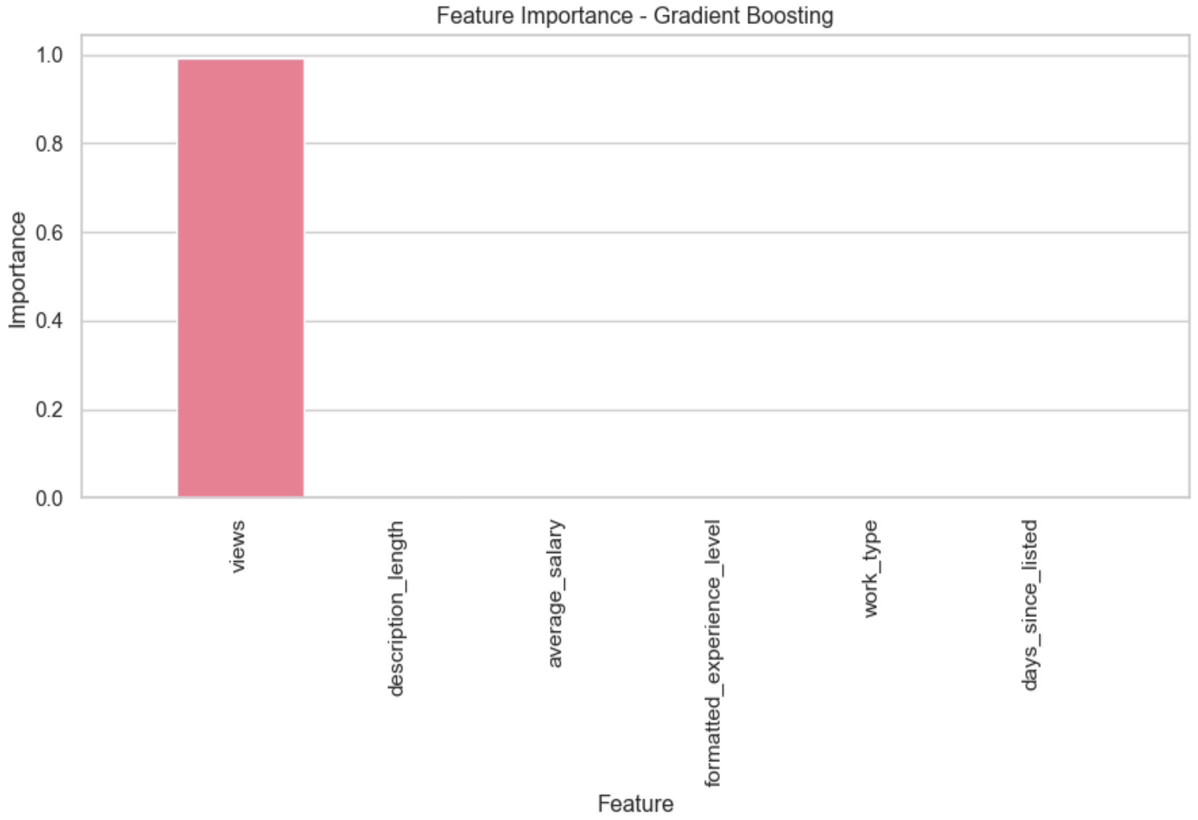
RandomForest model is error-free while K-NearestNeighbors is second with least errors.



RandomForest model is error-free while K-NearestNeighbors is second with least errors.

Modeling: Predictor Model – Feature Importance(For Tree Based Models)

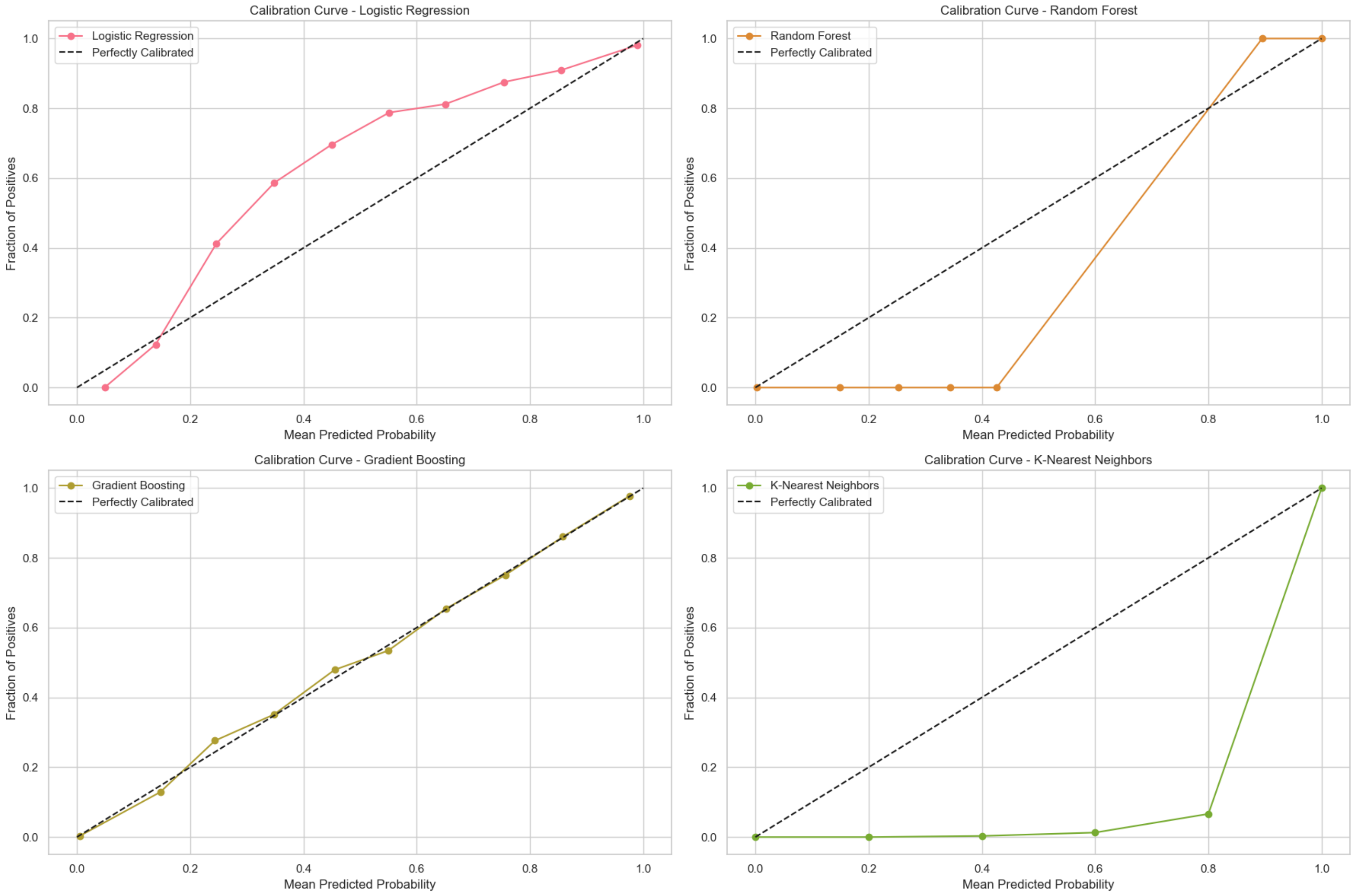




Views: The high importance score suggests that this feature has a strong predictive power in the model.
Description: The length or quality of the job description is the second most important feature.
Days_since_listed: This feature might help in understanding the freshness of the job posting and its attractiveness over time.
average_salary: The average salary offered by the job posting has some impact on the prediction.
formatted_experience_level: Suggests that while the experience level is a factor, it may not a strong predictor of the number of applications.
work_type: The type of work (e.g., full-time, part-time, remote) also has minimal importance in the model is a factor but does not significantly influence the prediction.

Modeling: Predictor Model – Calibration Curve

To assess how well the predicted probabilities are calibrated



Logistic Regression generally performs well in terms of calibration.

Random Forest and Gradient Boosting show moderate calibration issues, especially in the higher probability range.

K-Nearest Neighbors has significant calibration problems, indicating that its predicted probabilities are not reliable.

Modeling: Recommender Model: Content-Based Recommendation

Features in place: job_id, processed title, processed description,   processed location, views, applies, processed_company_name, work type, average salary; dtype=object

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re

# Initialize the TF-IDF Vectorizer and transform the job descriptions
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(df['description'])

def preprocess_text(text):
    # Improved text preprocessing
    text = text.lower()
    text = re.sub(r'\W', ' ', text)  # Remove non-word characters
    text = re.sub(r'\s+', ' ', text)  # Remove extra whitespace
    return text

def recommend_jobs(input_description, top_n=10):
    input_description_processed = preprocess_text(input_description)
    input_vector = vectorizer.transform([input_description_processed])
    similarities = cosine_similarity(input_vector, tfidf_matrix).flatten()
    indices = similarities.argsort()[-top_n:][::-1]
    return df.iloc[indices]

def main():
    while True:
        print("\nJob Recommendation System")
        print("1. Recommend Jobs based on Description")
        print("2. Exit")
        choice = input("Enter your choice (1/2): ")

        if choice == '1':
```

Output

```
Job Recommendation System
1. Recommend Jobs based on Description
2. Exit
Enter your choice (1/2): 1
Enter job description to find recommendations: i am a certified customer care agent.
```

Modeling: Recommender Model: KNN Recommendation System

2.4. KNN recommendation system

```
from sklearn.neighbors import NearestNeighbors

# Prepare feature matrix
X_features = recommender_df[['views', 'applies', 'average_salary']].values

# Apply KNN for job recommendations
knn = NearestNeighbors(n_neighbors=2, algorithm='auto').fit(X_features)
distances, indices = knn.kneighbors(X_features)

# Calculate average distance of nearest neighbors
average_distance = np.mean(distances)

# Print average distance
print(f"Average Distance to Nearest Neighbors: {average_distance:.2f}")

# Display KNN recommendations for a specific job
job_id = 5
recommendations = indices[job_id]
top_recommendations = recommender_df.iloc[recommendations[0]]

print("\nKNN Recommendations based on JOB ID:")
print(top_recommendations)
```

```
Average Distance to Nearest Neighbors: 0.00

KNN Recommendations based on JOB ID:
job_id                                                           91700727
processed_title                        economic development planning intern
processed_description     job summary economic development planning inte...
processed_location                                             raleigh nc
views                                                                  -0
applies                                                                -0
processed_company_name                          downtown raleigh alliance
work_type                                                               2
average_salary                                                         -0
Name: 5, dtype: object
```

The code provides a basic example of using KNN for recommendations based on numerical features.

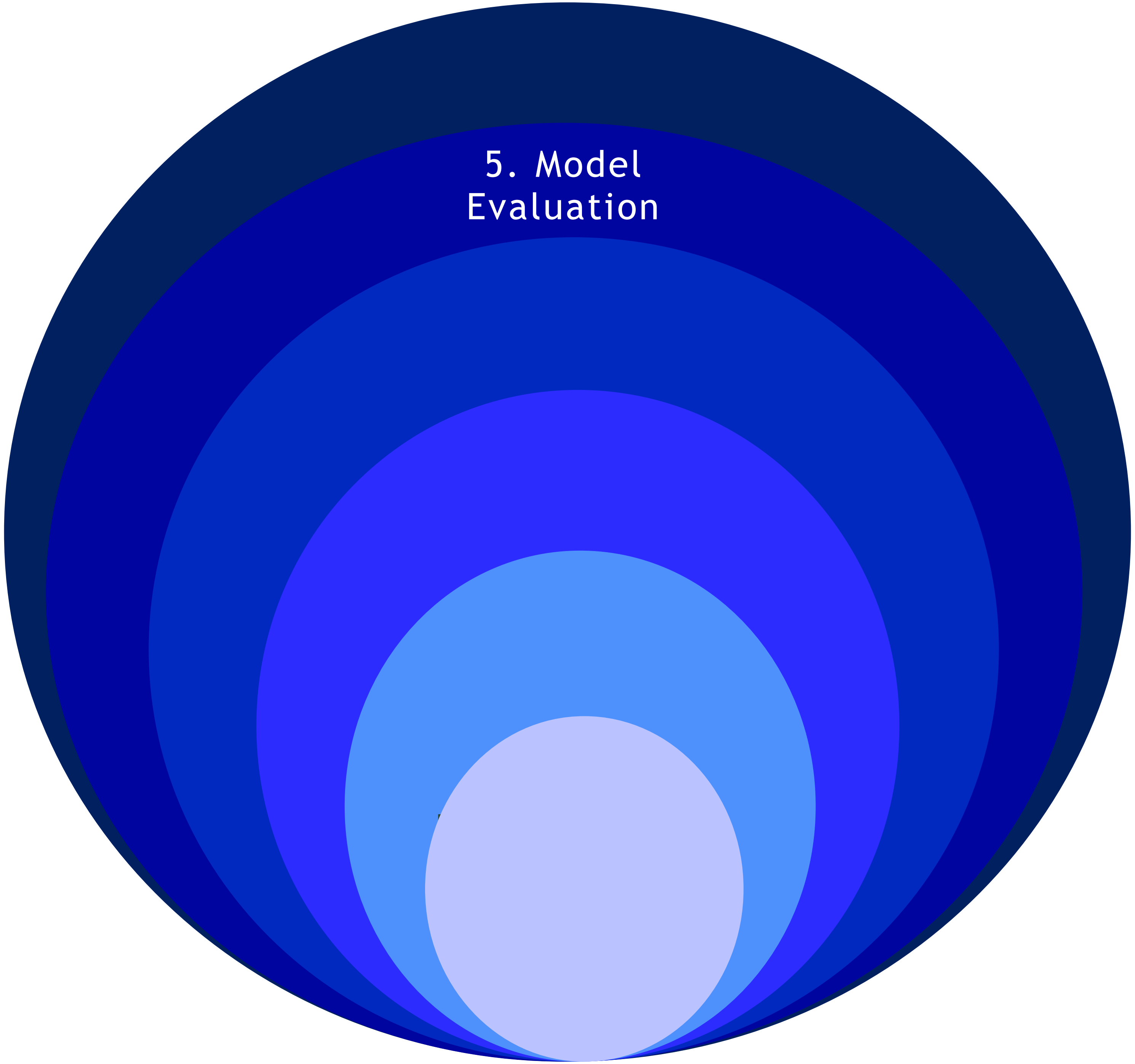
Modeling: Recommender Model: Sample Output

Recommend Most Viewed Jobs

```
Enter the number of top jobs to recommend: 6
Enter the job title to filter by: engineer
Top recommended jobs based on your input:
           job_id              processed_title  \
6273   3885111542    vp mechanical engineering
73254  3902922986  associate software engineer
53525  3901651266          full stack engineer
48742  3901349539         senior data engineer
66459  3902745810      software engineer intern
96926  3904952655            frontend engineer
```

Recommend Based on Job Title

```
Job Recommendation System
1. Recommend Jobs
2. Exit
Enter your choice (1/2): 1
Enter job title to find recommendations: nurse

Recommended Jobs:
                                         title               company_name  \
64046                                    nurse        my houston surgeons
87432                                    nurse          davita kidney care
32496                                    nurse          healthsearch group
84587                            nurse manager           ri international
32791                            nurse manager  complete staffing solutions
99188                         nurse technician             the job network
19937      travel nurse - registered nurse - rn  professional case management
```


5. Model
Evaluation

Model Evaluation

1.Logistic Regression Model Performance

Cross-Validation Accuracy: 0.941
Test Accuracy: 0.942
Precision:
Class 0: 0.93
Class 1: 0.96
Recall:
Class 0: 0.96
Class 1: 0.93
F1-Score: 0.94 (both classes)
ROC-AUC Score: 0.988

2.Random Forest Model Performance

Cross-Validation Accuracy: 0.986
Test Accuracy: 1.0
Precision: 1.00 (both classes)
Recall: 1.00 (both classes)
F1-Score: 1.00 (both classes)
ROC-AUC Score: 1.0
Conclusion: Random Forest

3.Gradient Boosting Model Performance

Cross-Validation Accuracy: 0.952
Test Accuracy: 0.953
Precision:
Class 0: 0.97
Class 1: 0.94
Recall:
Class 0: 0.93
Class 1: 0.97
F1-Score: 0.95 (both classes)
ROC-AUC Score: 0.989

4.K-Nearest Neighbors (KNN) Model Performance

Cross-Validation Accuracy: 0.973
Test Accuracy: 0.984
Precision:
Class 0: 1.00
Class 1: 0.97
Recall:
Class 0: 0.97
Class 1: 1.00
F1-Score: 0.98 (both classes)
ROC-AUC Score: 0.99998

Model Evaluation/2

A. Predictor Models

1.Logistic Regression performs well with balanced precision and recall, and an excellent ROC-AUC score. It's reliable for distinguishing between high and low application likelihoods.

2.Random Forest achieves perfect accuracy, precision, recall, and F1-Score on both classes, and an ROC-AUC score of 1.0. It's the best-performing model in terms of classification metrics.

3.Gradient Boosting shows strong performance with high precision and recall, a good balance between the two, and a very high ROC-AUC score. It performs slightly less well than Random Forest but still effectively distinguishes between classes.

4.KNN performs exceptionally well with very high accuracy, precision, recall, and F1-Score. It has an almost perfect ROC-AUC score, making it highly effective for classification.

B. Recommender Models

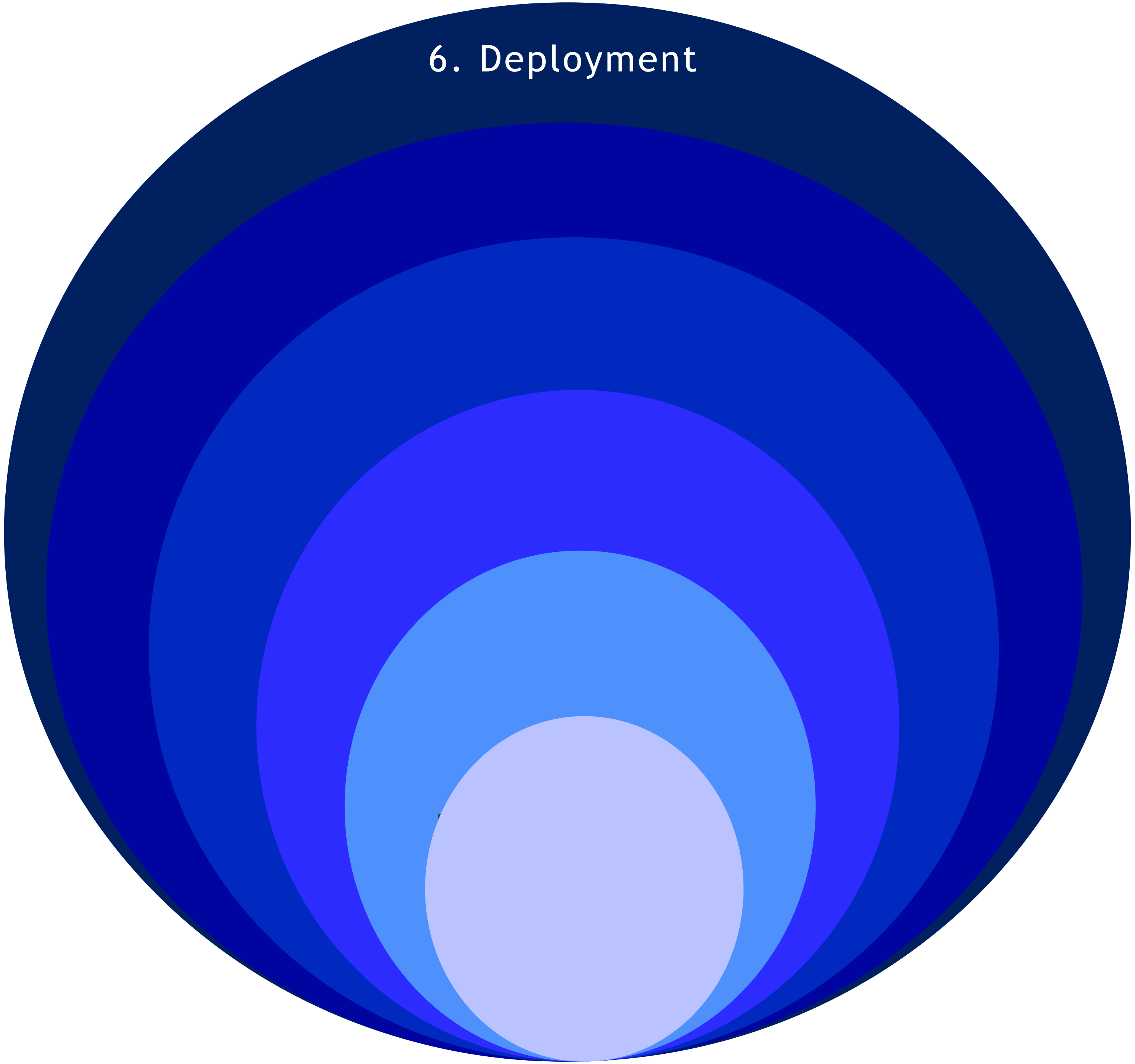
1.Job Recommendations Based on Description: Summary: The system recommends jobs based on similarity to the input job description. For instance, a description like "certified customer care agent" led to recommendations in related fields such as sales and insurance.

2.Job Recommendations Based on Job ID (KNN Model) Using job features like views, applies, and average_salary, the KNN model provided recommendations based on the similarity of job attributes. The output showed similar jobs based on these features.

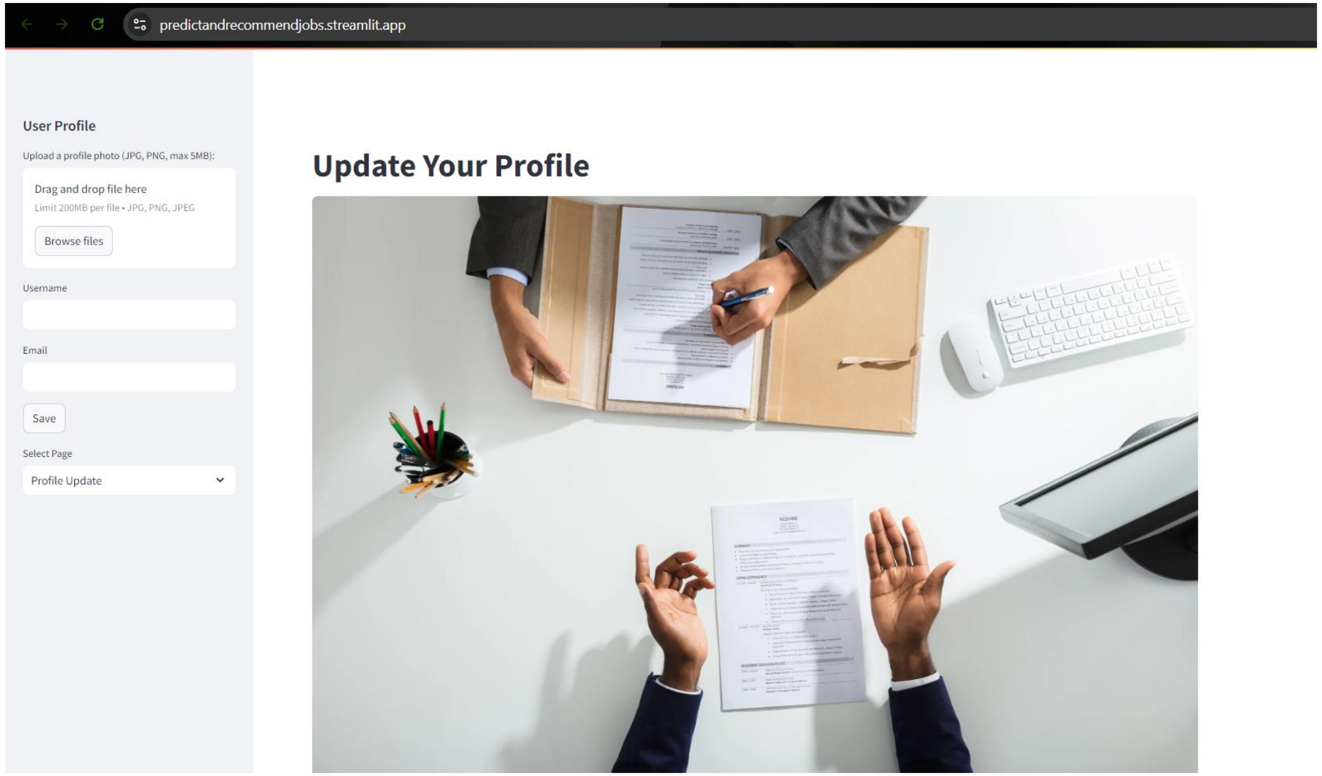
3.Job Recommendations Based on Title Filter: Recommendations based on a keyword filter (e.g., "engineer") returned jobs with titles containing the keyword, like "full stack engineer" and "software engineer intern."

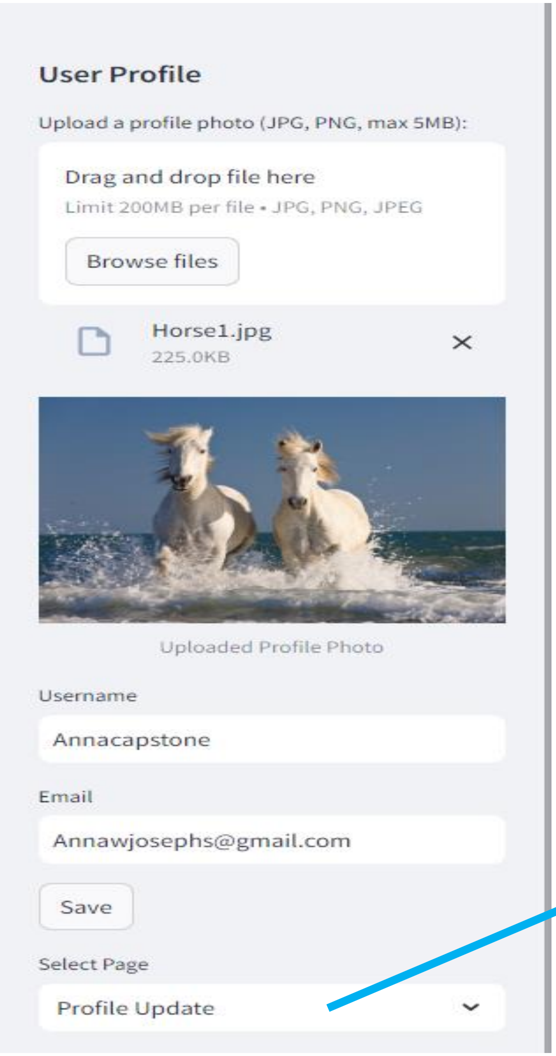
4.Job Recommendations Based on Average Distance (KNN Results): The KNN model returned jobs with very low average distances, indicating high similarity to the input job ID. The results were very similar in terms of job attributes.

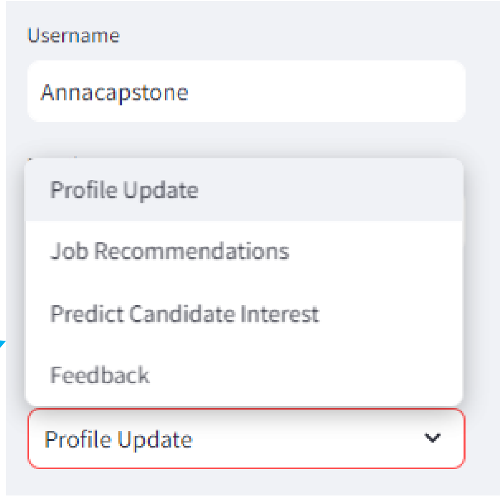
5.Job Recommendations Based on Input Job Title (General System): The system provided recommendations based on the job title input, returning jobs with similar titles. For example, inputting "nurse" resulted in various nursing-related job recommendations.


6. Deployment

Job Recommendations App: Interface Preview

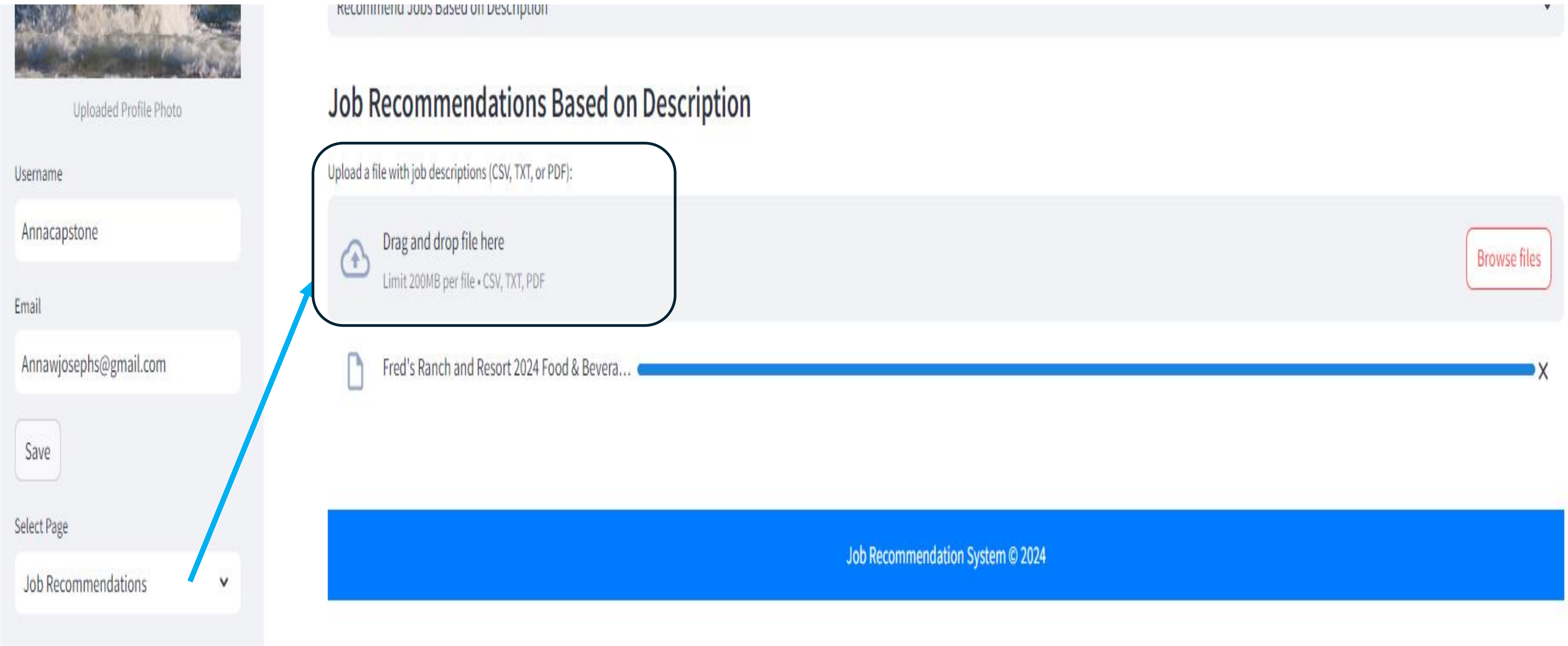




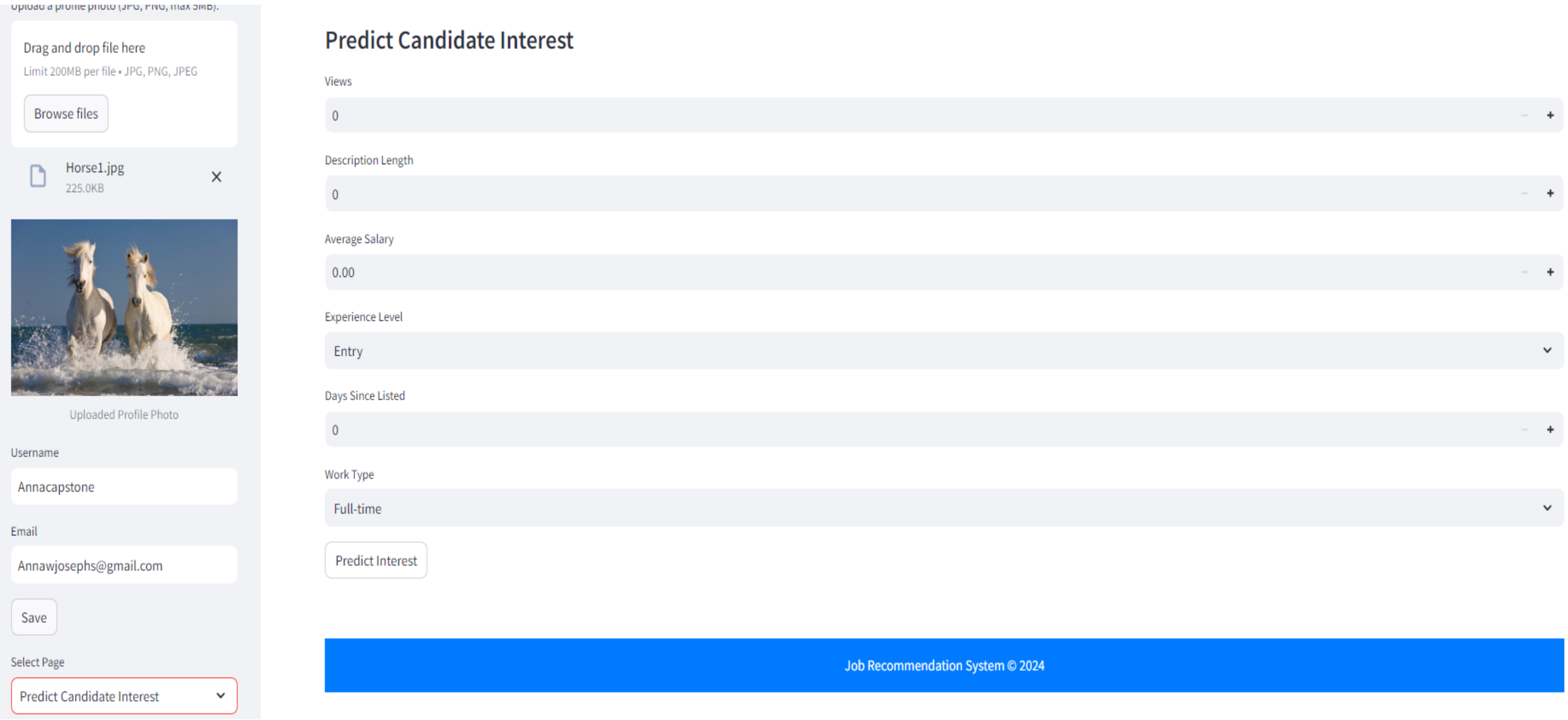


Profile creation

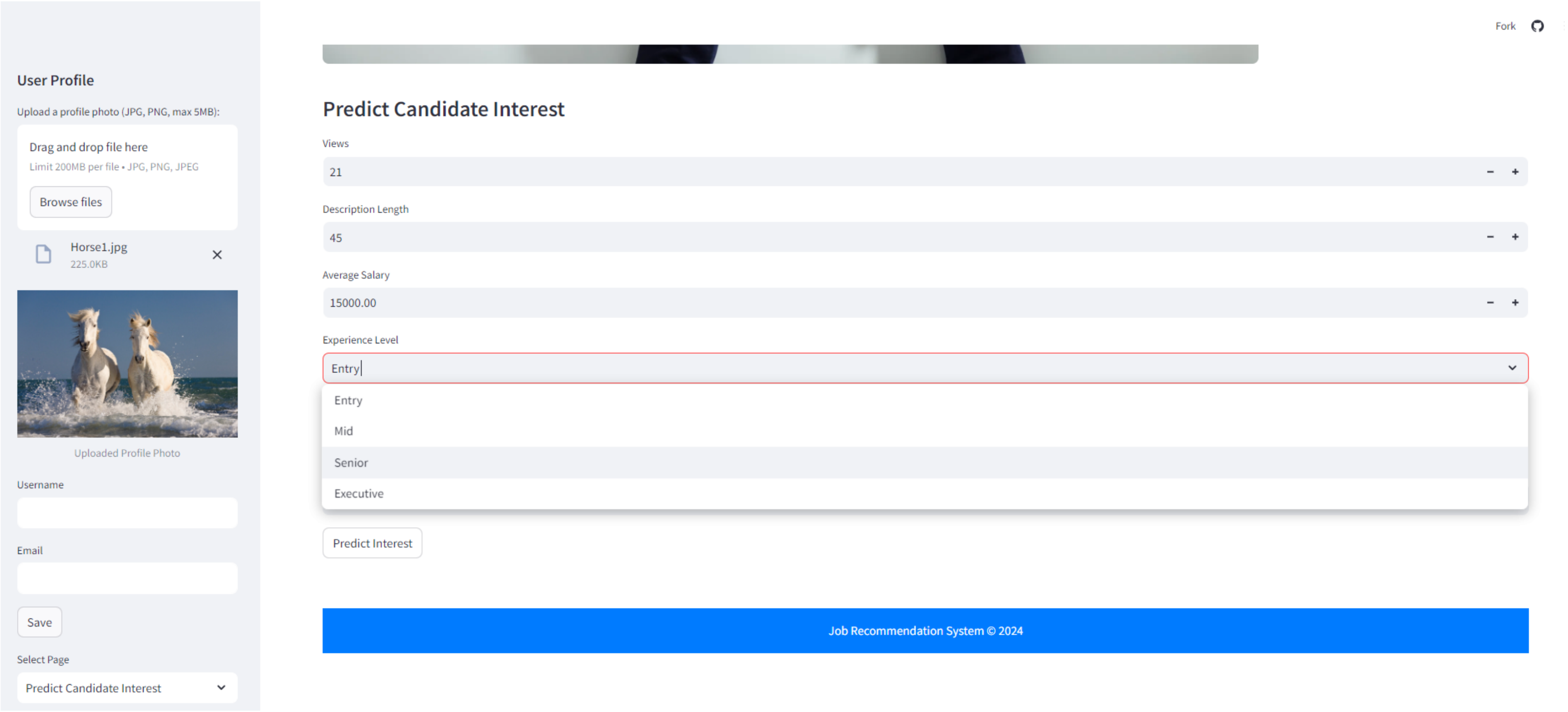
Job Recommendations App: Page Selection



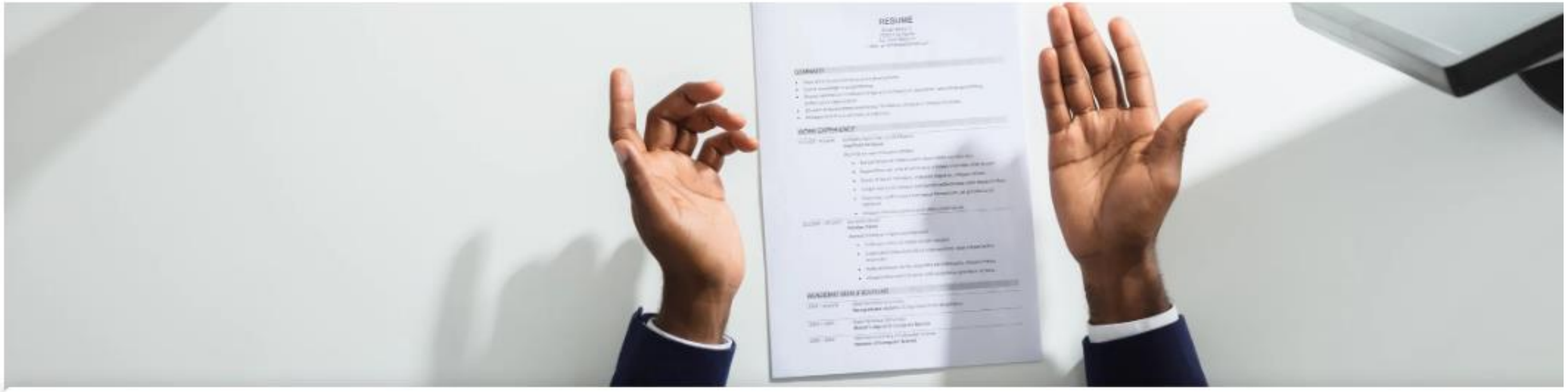
Job Recommendations App: Sample on Predict Candidate Interest



Job Recommendations App: Sample on Predict Candidate Interest/2



Job Recommendations App: Recommendations by Job Title



Dental CAD/CAM Designer - $20-$30/hour

Production Planner

Customer Service Representative

Associate Planner

Full Stack Engineer

Computer Scientist

Front end specialist

Marketing Coordinator

Get Recommendations

RECOMMENDATIONS

1. Model Selection:
Random Forest is the top performer based on perfect test accuracy, precision, recall, F1-Score, and ROC-AUC. It's a robust choice for this task due to its high performance across all metrics. KNN also performs extremely well, particularly in terms of the ROC-AUC score and F1-Score. It can be used as a strong alternative or in conjunction with Random Forest.

2. Model Deployment:
Deploy Random Forest as the primary model due to its flawless performance on the test set. It is suitable for production environments where high accuracy is critical. Consider KNN for applications where interpretability and simplicity are valued, as it offers very high-performance metrics. Further Testing:

3. Hyperparameter Tuning:
For Random Forest and KNN, consider tuning hyperparameters to potentially improve performance further. Ensemble Methods: Explore combining models to leverage the strengths of different models and improve overall performance. Model Monitoring and Updates:

4. Regular Monitoring:
Continuously monitor model performance to ensure it remains accurate over time, especially as new data becomes available. Periodic Updates: Update the model periodically with new data to maintain its relevance and performance. Feature Engineering:

5. Explore Additional Features:
Consider incorporating additional features or refining existing ones to enhance model performance further. Dimensionality Reduction: Use techniques like PCA or LDA to explore if reducing feature dimensions improves model efficiency and accuracy.

6. Incorporate Multi-Modal Features:
Combine text-based features (job descriptions, titles) with numerical features (views, salary) to provide a more holistic recommendation system.

7. Enhance Data Processing:
Use advanced NLP techniques and embeddings (like BERT) to better capture job descriptions and titles.

8. User Interaction:
Allow users to provide feedback on recommendations to continually improve the system.

Thank you!

Job Recommendation System App:
https://predictandrecommendjobs.streamlit.app/

Github: https://github.com/ge-saka/CAPSTONE-Group2