JOB MARKET TREND ANALYSIS

Data Mining CS-5593-995 | Fall 2025

Venu Sirisanagandla
Dept. of Comp. Sci.
University of Oklahoma
Oklahoma, USA
Venu.Sirisanagandla-1@ou.edu

Sri Sairam Pothuri
Dept. of Comp. Sci.
University of Oklahoma
Oklahoma, USA
Sri.Sai.Ram.Pothuri-1@ou.edu

Natalie Hill
Dept. of Comp. Sci.
University of Oklahoma
Oklahoma, USA
Natalie.a.hill-1@ou.edu

ABSTRACT

Our project aims to provide a clear understanding of job market trends using data mining algorithms. By analyzing the historical data provided by Kaggle, we explore job postings across different industries to find emerging trends, segment the market, classify roles based on skills and qualifications, forecast salary trends, and detect outliers that highlight niche roles or market gaps.

To achieve these goals, we are using several algorithms, each designed for a specific task. K-Nearest Neighbors (KNN) helps classify job postings by experience level, job title, and salary, allowing job seekers to find roles that suit their profiles. K-Means clustering groups job postings by attributes like salary, company size, and required experience, helping us find patterns in the job market. The ARIMA (AutoRegressive Integrated Moving Average) model is used to predict salary trends for different roles and experience levels, providing insights into future job market needs. We also use the Interquartile Range (IQR) method to detect outliers, highlighting unusual or niche job postings that could offer hidden opportunities.

Our approach brings these algorithms together in a user-friendly interface developed with React.js and Flask API and offering data visualizations tools like pie charts, line graphs, and box plots also. We have successfully completed salary trend analysis, job experience level classification, clustering. The final product allows users to explore job market trends interactively, download the results, and align their strategies based on real-time interactivity. This combination of algorithms and practical tools ensures our project is useful for both job seekers and employers, helping them make informed decisions in a rapidly changing job market.

KEYWORDS

KNN classification, clustering, time-series analysis, salary forecasting,

1 INTRODUCTION

In today's rapidly evolving economy, understanding job market trends is essential for both job seekers and recruiters. Job seekers need insights into the skills and qualifications that are in high demand and know about the future salary trends in a particular field, while recruiters need the key factors to look into throughout the interview process. Our project, "Job Market Trend Analysis" addresses these needs by leveraging advanced data mining techniques to analyze job postings.

Our project focuses on providing in-depth insights into the job market using a mix of machine learning techniques. We classify experience level for a job role, track emerging trends, segment the market and forecast salary patterns. By analyzing historical job posting data, we assist job seekers in adapting their skills to meet current and future market demands while also helping recruiters adjust their recruitment strategies accordingly.

The rest of this report will discuss the related work in this field, our methodology, and the implementation details of our approach. We will also present the results obtained from applying our algorithms, followed by a discussion of the conclusions and future work. Our goal is to provide actionable insights that make navigating the job market easier and more strategic for all stakeholders involved.

2 RELATED WORK

A significant body of research has focused on job market trend analysis using data mining and machine learning techniques, particularly clustering, salary forecasting, and job role classification. Studies like Xie et al. (2016) applied natural language processing (NLP) to classify job roles and identify relevant skills, while Liao et al. (2019) used ARIMA models for salary prediction. However, many of these approaches rely on static datasets, often limited to specific industries or regions, and do not integrate multiple analytical techniques. For example, clustering methods like K-Means segment job postings by attributes like salary and

experience but generally fail to include salary forecasting or outlier detection, which leaves gaps in understanding job market dynamics. Additionally, existing tools such as Glassdoor and LinkedIn provide salary data but typically lack interactivity or advanced analysis features, limiting their usefulness for deeper market insights.

Our project advances the field by offering a more comprehensive and integrated solution for analyzing job market trends. We combine several data mining techniques—including K-Nearest Neighbors (KNN) for job role classification, K-Means for clustering, ARIMA for salary forecasting, and Interquartile Range (IQR) for outlier detection—into a cohesive framework that provides a multi-dimensional view of the job market. This approach allows us to classify job postings, segment the market, predict salary trends, and identify niche roles with greater depth and accuracy. Additionally, our application includes powerful data visualization tools, such as pie charts, line graphs, and box plots, to help users interpret the results interactively. The interactive interface enhances user experience by allowing customization of analysis based on specific criteria such as industry or experience level. While we rely on historical data from platforms like Kaggle, rather than real-time data, our application's combination of advanced algorithms and visualizations provides actionable insights for both job seekers and employers, delivering a more robust and user-friendly tool compared to existing platforms.

3 PROPOSED WORK AND RESULTS

3.1 Application Description

The "Job Market Trend Analysis" application uses data mining and machine learning techniques to provide actionable insights into the evolving job market, benefiting both job seekers and recruiters. By analyzing historical job posting data from sources like Kaggle, the application employs algorithms such as K-Nearest Neighbors (KNN), K-Means clustering and ARIMA for the overall analysis of the job market. With an intuitive and responsive React.js interface, users can interactively explore, and filter job data based on criteria like industry, experience level, and salary, visualizing the results through dynamic charts such as pie charts, line graphs, and box plots. The application's backend, powered by Flask API, processes user inputs, applies the machine learning models, and delivers comprehensive, downloadable reports. This combination of advanced analytics and interactive visualizations helps job seekers align their skills with market demands and allows recruiters to refine hiring strategies, making the application a valuable tool in navigating the competitive job market.

3.2 Dataset

The project uses one primary dataset, salaries.csv, which contains information about job salaries across various roles, industries, and geographic regions. The dataset is about **12.95MB** in size which includes **29,562** records, each representing a unique job detail. The dataset has **11** attributes that capture details such as job titles, experience levels, salary information, company size, remote ration, work year and so on.

3.3 System Architecture

The system architecture of the job market trend analysis application is structured into four primary layers: the User Interface (React.js), Backend (Flask API), Machine Learning Models, and Data Storage. Users interact with the system through the React.js frontend, where they provide inputs and view results via interactive data visualizations. These inputs are processed by the Flask API, which communicates with various machine learning models, including K-Nearest Neighbors (KNN), K-Means for clustering and ARIMA Time-Series for salary forecasting. The historical data is stored in the server. Initially the models are well trained with the data and saved so, that they need not go through the entire pre-processing and building phase of model for every request.

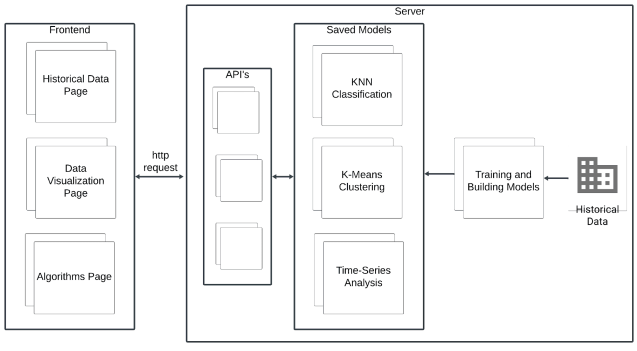

Figure 1: Job trend analysis system architecture

3.4 Algorithms

3.4.1 KNN classification

3.4.1.1 Introduction to KNN

The K-Nearest Neighbors (KNN) algorithm is a simple, yet powerful machine learning technique used for classification tasks. The basic idea behind KNN is that it assigns a label based on the majority class (in classification) of its k nearest neighbors in the feature space. It makes predictions by calculating the distance between the data point to be predicted and all the points in the training dataset.
For this project, we use KNN to predict the experience level of employees based on several features like job title, employment type, salary, employee residence, and remote ratio. This way, based on these parameters a job seeker can understand on what kind of experience level role jobs he/she needs to apply.

3.4.1.1 Euclidean Distance

The core of the KNN algorithm involves calculating the Euclidean distance between the data point. The data point with more neighbors within give distance is labeled same as the group label. The formula for Euclidean distance between two points x and y is:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

3.4.1.1 Justification for feature selection

The selected features for this KNN classification task — job title, employment type, salary, employee residence, remote ratio, and company size — are directly relevant to predicting the experience level of employees. Job title serves as a strong indicator of experience level, with titles like "Junior" or "Senior" reflecting varying levels of seniority and expertise. Similarly, employment type (e.g., full-time, part-time) is likely correlated with experience, as full-time roles are typically associated with more experienced employees. Salary is another key factor, with higher salaries often correlating with higher experience and seniority. Additionally, employee residence can provide insight into regional salary norms and job market competition, which can influence an individual's experience level. Remote ratio, or the amount of time an employee works remotely, may also serve as an indicator of seniority, as more experienced workers often have the flexibility to work remotely. Lastly, company size can reflect organizational structure, where larger companies often have more hierarchical roles requiring greater experience.
These features were chosen based on their logical connection to the target variable — experience level — and their expected ability to differentiate between employees with varying levels of seniority. Each feature provides valuable information that can help classify employees into different experience categories. Job title, salary, and company size are particularly strong predictors, as they directly relate to an employee's position and compensation, which typically increase with experience. Meanwhile, employment type and remote ratio are included for their potential to reflect the flexibility and responsibility typically associated with more experienced roles. By using these features, we can leverage both categorical and numerical data to build a model that effectively predicts the experience level of employees in a variety of work settings.

3.4.1.1 Data preprocessing

As we have chosen the features like job titles, employment type, salary in USD, and employee residence. Here's how preprocessing is handled:

- **Categorical Features:** The categorical features like job_title, employment_type, employee_residence, company_size are transformed using OneHotEncoding, which converts them into a number format which is suitable for the KNN algorithm.
- **Numerical Features:** Numerical features like salary_in_usd, remote_ratio are scaled using StandardScaler to normalize the range of the data and prevent certain features from dominating the distance calculations due to their larger scales.
- **Target Variable:** The target variable, experience_level is encoded into numeric labels using LabelEncoder for classification purpose. This encoding transforms the categorical values like “Junior”, “Mid”, and “Senior” into integers.

3.4.1.1 KNN algorithm

Here are detailed steps to perform the algorithm.

- Calculate the Euclidean distance between the test instance and all points in the dataset.
- Store the distances along with their corresponding labels in a list.
- Sort the list of distances in ascending order.
- Select the k nearest neighbors (the k value in the sorted list).
- Perform a majority vote among the k neighbors to determine the predicted label.
- Return the predicted label for the test instance.

3.4.1.1 Choosing optimal K value

To work the algorithm efficiently, we need to provide the optimal k value. There are several ways for finding the optimal k value. We have used Elbow Method.
The **Elbow Method** is a heuristic used to choose the optimal number of neighbors (k) for the KNN algorithm. The main idea is to plot the error rate (or MSE) for different values of k and identify the point where the error rate starts to decrease more slowly. This point is referred to as the **elbow**, and it represents the best balance between bias and variance.

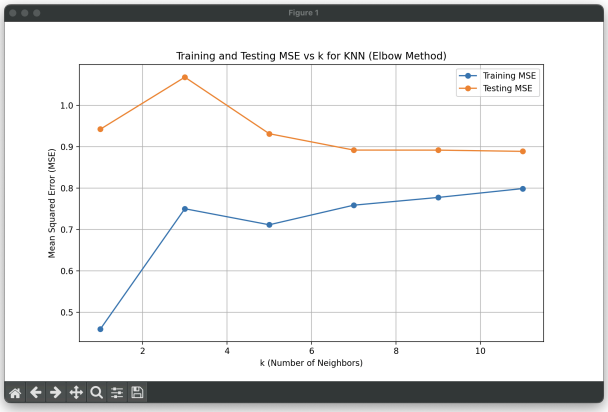

Figure 2: Training and Testing MSE vs k

Interpreting the plot: From the plot, if we notice that the test MSE decreases rapidly for smaller values of k (e.g., k=1,3,5) and starts to level off around a specific k, that is the point where the elbow occurs. From the plot, we could observe that the elbow point occurs at when k value is 7. So, we have chosen **k=7.**

3.4.1.1 K-fold cross validation

K-Fold Cross-Validation is a technique used to assess the performance of a machine learning model by splitting the dataset into k subsets (or folds). In our code, the data is divided into 5 equal folds. The model is trained on 4 of these folds and tested on the remaining fold. This process is repeated 5 times, with each fold serving as the test set once.
After executing the k-fold cross validation for the algorithm, we have got the average cross validation score as around 65%. According to the historical data distribution, the model performed decent and good enough to classify the user data points.

```
(salaries-env) (base) venusirisanagandla@Venus-MacBook-Pro flask-backend % python knn.py
Cross-Validation Scores: [0.64755623 0.64485033 0.63785521 0.64935724 0.63920839]
Average Cross-Validation Score: 0.6437654801569075
```

Figure 3: K-fold cross validation score for KNN

3.4.2 K-Means clustering

3.4.2.1 Introduction to K-Means

K-Means is a popular and powerful technique in unsupervised machine learning, widely used for clustering tasks. Clustering is a process where we group similar data points together into distinct categories or clusters, based on their similarities. The goal of K-Means clustering is to divide data into K clusters, where K is a number chosen by the user, such that the data points within each cluster are as similar as possible, and data points in different clusters are as dissimilar as possible.

In this project, K-Means clustering was used to group employees based on four key features: **salary in USD**, **job title**, **company location**, and **experience level**. The goal was to identify distinct clusters of employees with similar characteristics, enabling a deeper understanding of workforce dynamics.

3.4.3.2 Justification for feature selection

Before applying K-Means clustering, it's essential to carefully select the features that will drive the clustering process. Effective feature selection ensures that the chosen variables contribute meaningfully to grouping the data into distinct, interpretable clusters. In our case, the goal is to segment employees based on job title, location, salary, and experience level, which are the key determinants of an employee's position, compensation, and career trajectory.

Key Considerations for Feature Selection

- **Relevance:** The features should directly contribute to the clustering task. In this case, job title, salary, experience, and location are highly relevant as they define an employee's role and position within the company.
- **Distinctness:** Features should capture different aspects of the data to avoid redundancy. Correlated features may not add value to the clustering process.
- **Interpretability:** The features should be understandable and actionable. Using intuitive variables like job title, salary, and location makes the resulting clusters easier to interpret.

We selected **job title**, **company location**, **salary in USD**, and **experience level** because they directly impact employees' roles and compensation. These features are varied and relevant for distinguishing employees, and they provide complementary insights without redundancy.

Correlation Analysis and Heatmap: To ensure the selected features were complementary, we analyzed their correlations using a heatmap. Key insights include:

- **Job Title vs. Salary**: A moderate positive correlation, with senior roles earning higher salaries.
- **Location vs. Salary**: Moderate correlation, indicating higher salaries in certain regions.
- **Experience vs. Salary**: Strong correlation, as more experience leads to higher pay.
- **Job Title vs. Location**: Moderate correlation, suggesting certain roles are more common in specific locations.

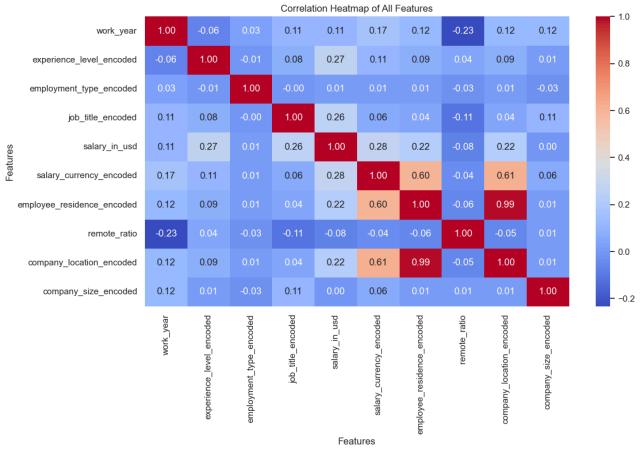

Figure 4: Correlation Analysis of all features

Excluding the Remote Ratio: Although the **remote ratio** showed weak correlation with other features, it was excluded from the clustering process. While it indicates work flexibility, it does not have the same direct impact on role, compensation, or career stage as the other features.

Based on these insights, the final feature set for clustering, **job title**, **location**, **salary**, and **experience level** capture key dimensions of an employee's role and career, providing a robust basis for meaningful clusters.

3.4.3.3 Data Preprocessing

Data preprocessing is a critical step in preparing our dataset for K-Means clustering. It ensures that the data is clean, consistent, and ready for analysis. Below are the key preprocessing steps we applied to the dataset:

- **Handling Missing Data:** Missing values in salary_in_usd were filled with the median to mitigate the impact of outliers. Missing values in job_title, company_location, and experience_level were imputed using the mode (most frequent value) to maintain consistency without introducing bias.
- **Label Encoding:** Features like job_title, company_location, and experience_level were converted into numerical values using Label Encoding. This is preferable over One-Hot Encoding, as One-Hot would increase the dimensionality unnecessarily, especially for features with many unique categories, while Label Encoding keeps the feature space more manageable
- **Standardization:** Since K-Means clustering is sensitive to feature scales, we applied Standardization to ensure that each feature has a mean of 0 and a standard deviation of 1. This prevents features like salary and experience level, which may have different units, from dominating the clustering process.

3.4.3.4 Algorithm

Initialization:

- Randomly select k initial centroids C_1, C_2…., C_k from the dataset.

Step 1: Assign Data Points to Clusters (Cluster Assignment):

- For each data point x_i in the dataset, calculate the Euclidean distance between x_i and each centroid C_j:

$$D(x_i, C_j) = \sqrt{\sum_{d=1}^{m} (x_{i,d} - C_{j,d})^2}$$

where m is the number of dimensions (features), and $x_{i,d}$ and $C_{j,d}$ are the d-th features of the data point x_i and centroid C_j, respectively.

- Assign each data point x_i to the cluster of the nearest centroid:

$$clusterAssignment(x_i) = arg\ min_j\ D(x_i, C_j)$$

Step 2: Update Centroids (Centroid Recalculation):

- After assigning all data points to clusters, update the centroid Cj for each cluster j by calculating the mean of all data points assigned to that cluster:

$$C_j^{\text{new}} = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$

Step 3: Convergence Check:
Check if the centroids have converged, i.e., if the centroids do not change significantly between iterations. The stopping condition is:

$$\| C_j^{\text{new}} - C_j^{\text{old}} \|_2 < \epsilon$$

where $\|\cdot\|_2$ is the Euclidean norm (distance), and ϵ is a small threshold value (tolerance).
Step 4: Repeat Steps 1 and 2:

- Repeat the assignment of data points to clusters and the recalculation of centroids until convergence is reached (centroids do not change or changes are minimal).

Output:
Once convergence is reached, the final centroids and the cluster assignments are the output of the K-Means algorithm.

3.4.3.4 Implementation

- **Preprocessing the data**: We handle missing values by filling salary with the median and other categorical features (job title, location, experience level) with the mode.
- **Labeling encode categorical variables**: Encode job title, company location, and experience level using LabelEncoder to convert them into numeric format.
- **Standardizing the features**: Using StandardScaler to scale the features (salary, job title, location, and experience) to ensure they have similar ranges and contribute equally to the clustering.
- **Splitting the data**: Split the dataset into training (80%) and testing (20%) sets to train the model and evaluate its performance.
- **Applying the K-Means algorithm**: Implementing a custom K-Means function to cluster the data into k clusters using Euclidean distance and iteratively update the centroids until convergence.
- **Selecting the optimal number of clusters**: Using the Elbow Method to determine the optimal value of k by plotting the Within-Cluster Sum of Squares (WSS) for different values of k.
- **Performing PCA for 2D visualization**: Applying Principal Component Analysis (PCA) to reduce the dimensionality of the data to two components and visualize the clusters in 2D space.
- **Evaluating clustering quality**: We use Silhouette Score to assess the quality of the clusters and check the consistency of the model.
- **Summarizing each cluster**: For each cluster, we calculate and display key statistics such as average salary, typical job title, location, and experience level to interpret the characteristics of each cluster.
- **Predicting cluster for new user data**: For new user input, we predict the cluster by calculating the Euclidean distance to the centroids and plot the user data point on the cluster plot, showing the corresponding cluster summary.

3.4.3.5 Choosing the Optimal K

The process of choosing the optimal number of clusters, k, is crucial to ensuring that the K-Means algorithm performs effectively and provides meaningful segmentation of the data. To determine the best value for k, we used the Elbow Method, which helps in finding the point where the addition of more clusters no longer significantly reduces the Within-Cluster Sum of Squares (WSS). Here's the process in brief:

- **Plotting the WSS:** We plotted the WSS for different values of k (ranging from 1 to 10). The WSS measures the compactness of the clusters, indicating how tight or spread out the clusters are. A lower WSS suggests that the data points are closer to their centroids, resulting in more distinct clusters.
- **Identifying the Elbow Point**: As **k** increases, the WSS decreases because more clusters allow for a better fit to the data. However, beyond a certain number of clusters, the reduction in WSS becomes less significant. The **elbow point** represents the value of **k** where this decrease in WSS starts to slow down, indicating the optimal number of clusters.

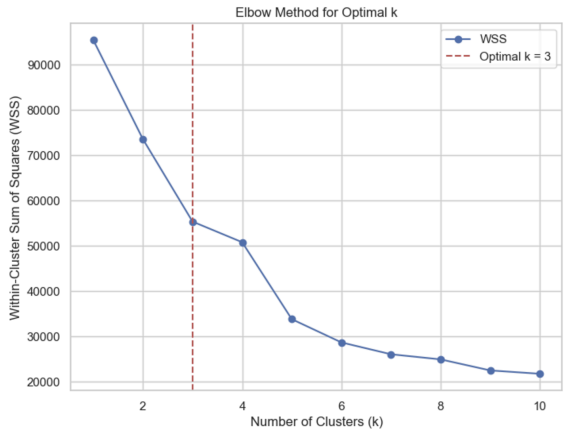

Figure 5: Elbow method to find optimal k for clusters

Interpreting the plot: Based on the plot, we observed that **k = 3** was the ideal choice. This value of **k** balances a sufficiently low WSS with an intuitive number of clusters that can be meaningfully interpreted. Adding more clusters did not provide a significant improvement in cluster compactness, making **k = 3** the optimal choice for our dataset.

3.4.3.6 Evaluation and Validation

To evaluate the performance of our K-Means clustering model, we used the **Silhouette Score** and visualized the clustering results with the **PCA** transformation. Below are the plots that we obtained:

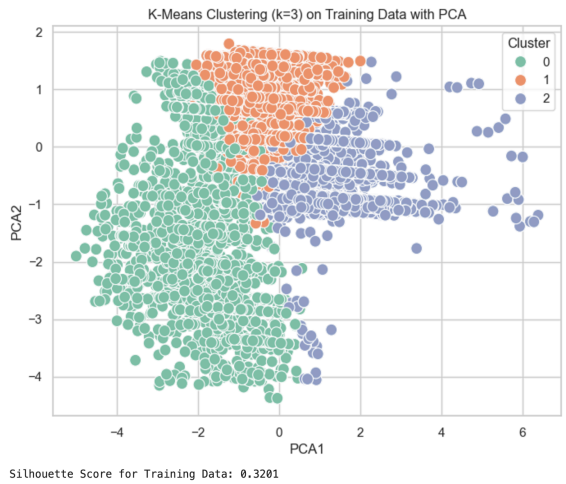

Figure 6: Clustering on training data

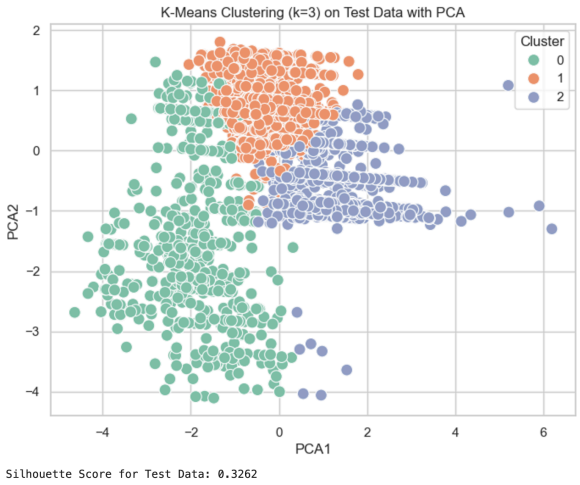

Figure 7: Clustering on test data using the model built by train data

Silhouette Score: The Silhouette Score measures the quality of clustering by calculating how similar each point is to its own cluster compared to other clusters. A higher score (closer to 1) indicates well-separated clusters.

- Training Data Silhouette Score: 0.3201
- Test Data Silhouette Score: 0.3262

Cluster Visualizations: The PCA transformation was applied to both the training and test data to visualize the clusters in a 2D space. The PCA plot for the training data (Figure 6) clearly shows three distinct clusters, which are derived from features like job title, salary, company location, and experience level. The separation between the clusters in the PCA space indicates that the features we selected for clustering are indeed meaningful, and that the K-Means algorithm has successfully identified distinct groups in the data.

The PCA plot for the test data (Figure 7) shows the clusters obtained by applying the model trained on the training data to the test set. While we observe a slight shift in the positions of the clusters, the general structure remains consistent, confirming that the model can generalize well to unseen data.

Comparison of Silhouette Scores

The Silhouette Scores for both the training and test data are quite close (0.3201 for training and 0.3262 for test). This consistency indicates that the clustering model is stable, and the clusters identified during training are well-preserved when applied to new, unseen data.

The slight increase in the test data score compared to the training score (a difference of 0.0061) suggests that the model generalizes well and performs slightly better on the test data. This minimal difference may indicate that the test set's inherent characteristics align well with the training set, or that the clustering process is robust enough to handle minor variations in the data.
Overall, the Silhouette Scores indicate that the clustering model has performed effectively, with well-separated and cohesive clusters in both the training and test datasets. The clustering model can be considered reliable and capable of generalizing to new data with similar quality.

Insights from the visualizations:

- **Cluster 0**: Employees in this group are typically lower to mid-level employees, as indicated by lower salary and job title encoding.
- **Cluster 1**: The second cluster seems to represent mid-level to senior employees, with a noticeable increase in salary and experience.
- **Cluster 2**: The third cluster likely represents high-salary, high-experience employees, possibly in senior or executive roles, with strong ties to specific locations.

Patterns Observed:

- **Job Title and Salary**: The positive correlation between **job title** and **salary** is evident, with more senior roles in clusters 1 and 2 commanding higher salaries.
- **Location Influence**: Company **location** impacts salary, with employees in high-cost regions (like Silicon Valley) likely appearing in higher clusters, with **location** showing some correlation to **job title** and **experience level**.
- **Experience Level**: As expected, more experienced employees appear in clusters 1 and 2, with higher salaries and job titles associated with greater experience.

3.4.3 ARIMA Time-Series analysis

3.4.3.1 Introduction to ARIMA

ARIMA (AutoRegressive Integrated Moving Average) is a popular time series forecasting technique used to model and predict future values based on historical data. ARIMA combines three key components:

- **AR (AutoRegressive):** The model uses the relationship between an observation and a number of lagged observations (previous time steps).
- **I (Integrated):** The process of differencing the series to make it stationary by removing trends or seasonality.
- **MA (Moving Average):** The model uses the relationship between an observation and a residual error from a moving average model applied to lagged observations.

The ARIMA model is defined by three parameters: p (autoregressive order), d (degree of differencing), and q (moving average order). These parameters control the model's ability to capture dependencies in the data and make accurate predictions.
For this project, we use ARIMA time-series to forecast the future trends of salary for a particular job title based on the historical data. Using this, any job seeker can know the future trends and helps in choosing the right job title.

3.4.3.2 Justification for feature selection

Choosing salary as the target variable for prediction in this project is an intuitive decision, primarily driven by the availability and relevance of the data. Salary is a key metric for understanding job market trends and is widely used by professionals, employers, and policy-makers to assess compensation standards across industries. While other potential parameters, such as remote ratio or experience level or employment type, could also be considered, salary is the most concrete and measurable indicator available in the dataset. Additionally, salary data is often readily accessible and is a direct reflection of the economic value placed on different job titles, making it the most suitable parameter for forecasting. Given the absence of a more relevant or comprehensive alternative, salary remains the best choice for predicting job market trends in this case.

3.4.3.3 Data preprocessing

Before applying the ARIMA model, data preprocessing is crucial to ensure that the time series is in an appropriate format. The key preprocessing steps involved in this model are:

- **Handling categorical values:** The job_title column is categorical. It is encoded into numerical values using LabelEncoder, which assigns a unique integer to each job title. This allows the model to

process the job title as numerical input, which is required for time series forecasting.

- **Grouping data:** The dataset is grouped by work_year (the year of salary data) and job_title_encoded (the numerical encoding of the job title). The average salary (salary_in_usd) is computed for each combination of job title and year to form a clean time series dataset.
- **Date conversion:** The work_year column, which initially contains year information, is converted into a datetime format. This ensures compatibility with ARIMA, which requires time series data to have a datetime index.
- **Handling insufficient Data:** For job titles with insufficient data (fewer than 3 years), the ARIMA model is not applied, as ARIMA requires a minimum of 3 data points to model trends accurately.
- **Sorting data:** The data is sorted by the work_year to ensure chronological order, which is essential for time series analysis.

3.4.3.4 Algorithm

Here are detailed steps to perform the algorithm.

- Collect and preprocess the time series data (ensure stationarity through differencing if necessary).
- Identify the optimal values for p, d, and q using ACF and PACF plots or grid search.
- Split the data into training and testing sets using time series cross-validation.
- Fit the ARIMA model on the training set using the identified p, d, and q values.
- Forecast future values based on the trained model.
- Evaluate model performance using error metrics like RMSE on the test set.
- Save the trained model for future predictions.

3.4.3.4 Choosing optimal p, d, q values

Finding the optimal p, d, and q values for an ARIMA model involves determining the best combination of three parameters that minimizes the error in predictions. These parameters correspond to the following:

1. p: The order of the autoregressive (AR) part — the number of lag observations in the model.
2. d: The degree of differencing required to make the series stationary (i.e., removing trends).
3. q: The order of the moving average (MA) part — the number of lagged forecast errors in the model.

- **Determine d value:**
  A stationary time series is one whose properties, such as mean and variance, do not change over time. ARIMA models require stationary data. In our application the work_year and salary are stationary. We have chosen the default value of d as 1.
- **Determine p and q value using PACF and ACF Plots:**
  We can determine p and q by analyzing the AutoCorrelation Function (ACF) and Partial AutoCorrelation Function (PACF) plots.

  The ACF plot helps to determine the number of lag observations that are correlated with the series. For MA (Moving Average), you can determine q by looking at where the ACF plot cuts off (the first lag after which the ACF is close to zero).
  The PACF plot helps to determine the number of lag observations to be included in the AR (AutoRegressive) part of the model. For AR (AutoRegressive), you can determine p by looking at where the PACF plot cuts off (the first lag after which the PACF is close to zero).

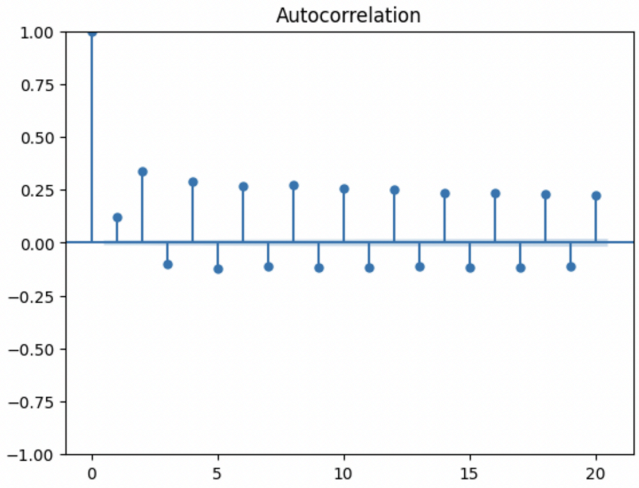

Figure 4: ACF plot for determining q value

Interpreting the plot: Here we can see a significant spike at lag 1 (and potentially lag 2), the MA (Moving Average) order q should likely be 1 or 2. If the ACF decays quickly to zero after a few lags, it indicates that higher-order q values. From

the plot we can it may not require higher q value. So, we choose q as 1.

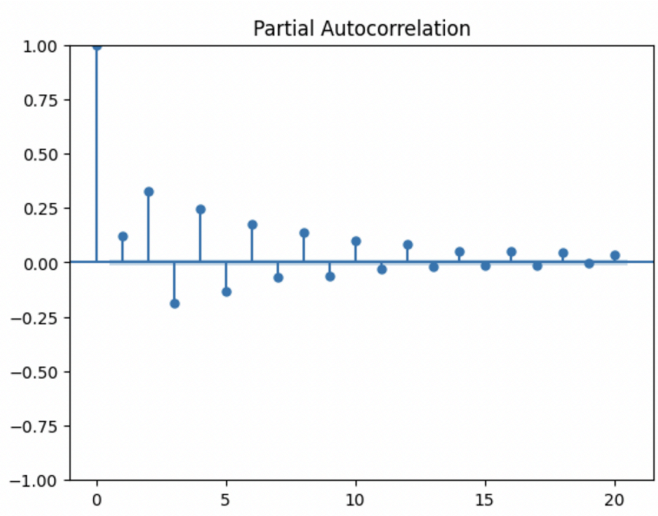

Figure 5: PACF plot for determining p value

Interpreting the plot: Here we can see that there's a significant spike at lag 1 (and possibly lag 2) and the plot quickly drops to zero, it suggests that AR (AutoRegressive) order p should be 1 or 2. If the PACF doesn't drop to zero after lag 2 and instead decays slowly, it might suggest a higher p value. Here we need a higher p value. So we have choose p as 5.

3.4.3.4 Validation

The validation for the ARIMA model is performed using TimeSeriesSplit cross-validation. This approach ensures that the model is tested on different time periods and that the temporal order of the data is respected (i.e., no future data is used to predict past values). The following validation steps are used:

- **TimeSeriesSplit:** The dataset is split into multiple training and test sets using time series cross-validation. This is especially important for time series data because it prevents the model from peeking into future data.
- **Root Mean Squared Error (RMSE):** The RMSE is used to evaluate the model's prediction accuracy. It measures the difference between the actual and predicted salary values. A lower RMSE indicates better predictive performance.
- **K-fold cross validation:** TimeSeriesSplit is used to perform cross-validation with multiple folds. The model is trained and validated across different periods, and the average RMSE for each fold is calculated. This helps ensure that the model generalizes well to unseen data.

In our code, the data is divided into 5 equal folds. The model is trained on 4 of these folds and tested on the remaining fold. This process is repeated 5 times, with each fold serving as the test set once.

After executing the k-fold cross validation for the algorithm, we have got the average mean squared error of **12%** which gives a prediction difference of **$4382** amount in the prediction. With this result we can say that the model has performed well and could predict the future trend salaries appropriately.

```
Overall Average RMSE: 4382.90
Overall RMSE as Percentage of Mean Salary: 12%
```

Figure 6: K-fold cross validation score for ARIMA

3.5 User-interface

3.6 Justifications of Development Choices

3.6.1 Choice of Machine Learning Models (KNN, K-Means, ARIMA, IQR):

The selection of K-Nearest Neighbors (KNN), K-Means clustering, ARIMA, and Interquartile Range (IQR) was made based on their effectiveness in solving specific tasks within the application. KNN is used for job role classification, where similarity-based predictions help categorize job postings by experience level and salary. K-Means clustering efficiently segments the job market into meaningful groups based on features like salary, company size, and experience level, allowing for better market segmentation. ARIMA is chosen for its ability to predict salary trends over time, a crucial feature for forecasting future job market demands. IQR helps detect outliers, identifying rare or niche job roles that may not be immediately visible through traditional analysis, adding an element of hidden opportunity detection.

3.6.1 Use of Flask for Backend Development:

Flask was selected as the backend framework due to its simplicity, scalability, and flexibility. As a lightweight micro-framework, Flask provides an ideal environment for handling API requests and integrating machine learning models without the overhead of larger frameworks. This

choice allowed us for easy deployment, faster development cycles, and a more manageable backend structure, ensuring that the application can scale and evolve as new features are added.(percent of this flask in industries)

3.6.2 Choice of React.js for Frontend Development:

React.js was chosen for its ability to build dynamic, interactive, and responsive user interfaces. The application's frontend relies on React's component-based architecture to handle complex visualizations, such as pie charts, line graphs, and box plots, efficiently. To further enhance the user experience, the application utilizes the PrimeReact UI library, which provides a wide range of pre-built, customizable components, including forms, charts, and data tables. This library streamlines the development process, offering responsive and visually appealing elements that ensure a consistent user experience across different devices (mobile, tablet and web page view). By using PrimeReact, the application can deliver a enterprise, user-friendly interface without the need for extensive custom design work. (explain the demand of reactjs)

3.6.2 Choice of GitHub for Collaborative Development:

GitHub was selected as the platform for collaborative development due to its robust version control, ease of collaboration, and support for team-based workflows. We have used GitHub to efficiently manage code changes, track project progress, and collaborate seamlessly across different stages of the application's development. GitHub's branching and pull request features allow team members to work on separate features or bug fixes independently before merging them into the main project. With GitHub, the project benefits from an open and transparent development process, enhancing team productivity and collaboration.

3.7 Experiments with analysis

3.8 Screenshots of analysis

3.9 User-manual

4 CONCLUSION

The Job Market Trend Analysis application offers a powerful, data-driven approach to understanding the complexities of today's job market. By combining machine learning models with interactive visualizations, it empowers both job seekers and employers to make informed, strategic decisions based on trends, salary forecasts, and market segmentation. Its ability to classify, cluster, and predict job market dynamics with historical data provides a clear advantage for users seeking to align their career paths with emerging opportunities.

5 FUTURE WORK

There is a lot of scope for the future work in terms of scalability, performance and features. Here are few future advancements that will aim to leverage the standard of the application.

- Incorporate real-time job posting data from platforms like LinkedIn, Indeed, and Glassdoor to enhance the application's responsiveness and provide up-to-date insights.
- Explore and integrate additional machine learning algorithms, such as Random Forests or Support Vector Machines (SVM), to improve the accuracy of job role classification, salary forecasting, and market segmentation.
- Develop algorithms that provide personalized job role recommendations and salary insights based on users' experience, qualifications, and career goals.
- Develop a mobile version of the application, making it more accessible and allowing users to track job market trends and receive alerts on the go.
- Integrations with HR management and recruitment platforms to provide real-time data for recruitment strategies and workforce planning.

REFERENCES

[1] Patricia S. Abril and Robert Plant, 2007. The patent holder's dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan, 2007), 36-44. DOI: https://doi.org/10.1145/1188913.1188915.

[2] Sten Andler. 1979. Predicate path expressions. In *Proceedings of the 6th. ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '79).* ACM Press, New York, NY, 226-236. DOI:https://doi.org/10.1145/567752.567774

[3] Ian Editor (Ed.). 2007. *The title of book one* (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago. DOI:https://doi.org/10.1007/3-540-09237-4.

[4] David Kosiur. 2001. *Understanding Policy-Based Networking* (2nd. ed.). Wiley, New York, NY..