# Predictive Analytics and Career Trends in Software Development: A Data Mining Approach

Suyog Joshi

12/09/2024

## 1 Introduction

The continuously changing technological landscape shapes the demand for new skills and defines career trajectories within the industry. This project addresses one of the most important challenges in this regard: understanding how such changes affect the career path of developers through the analysis of the developer survey data of Stack Overflow. The major objectives are threefold: derivation of meaningful insights, prediction of developer salaries, and anticipation of future job roles. These objectives are essential for aligning workforce development with emerging trends, enabling both individuals and organizations to stay competitive.

Among the most sought-after insights for developers in their career journey are salary trends. The project, through the analysis of past and current data, provides a comprehensive understanding of how different factors, such as programming skills, location, and experience, influence earning potential. Employers can use this information to offer competitive packages, while developers can strategically invest in professional development.

The second part of this project involves predicting the job roles of the future, using synthetic data and machine learning models. New domains such as artificial intelligence, cloud computing, and quantum technologies are bringing about opportunities that were unimaginable a decade ago. The identification of these trends by this project will arm developers, educators, and policymakers with the knowledge to prepare for a rapidly shifting job market.

These insights are also helpful for policymakers and organizations in making decisions. Governments can use the findings to create education policies that meet market demands, and companies can improve workforce planning. This project also aims to predict future job roles, helping to drive innovation in those areas.

In essence, this is a step toward understanding and adapting to the complexities of the technology workforce. It lays the bedrock for a more informed and prepared technology sector, capable of meeting the demands of tomorrow's challenges through data-driven insights, salary predictions, and foresight into job trends.

## 2 Data

### 2.1 Data Collection

The data for this project comes from Stack Overflow's developer surveys, which collect responses from developers around the world across multiple years. These surveys provide valuable information about demographics, skills, employment details, salaries, and more. The dataset was obtained from Stack Overflow's publicly available archives. For this project, we focused on data from the years 2022 to 2024 for predicting salaries and performing clustering analysis. However, when working on career prediction, we had to exclude the data from 2022 because it lacked enough relevant columns needed for the analysis. By using more complete and consistent data from 2023 and 2024, we ensured that the predictions might be accurate and meaningful in the future as we predicted the job roles for future using the synthetic data we created. This approach helped us maintain the quality and relevance of the insights generated by the project.

### 2.2 Pre-processing

- **Data Cleaning:**
  Handling missing values was an essential step to ensure the data was accurate and reliable for analysis. Missing entries, especially in key fields like salaries, could lead to incorrect conclusions or introduce biases in the results. To address this, incomplete records related to salaries were dropped to minimize bias. Additionally, some column names in the dataset were inconsistent, so they were standardized to ensure all relevant columns could be included in the analysis. Where appropriate, missing

values in other fields were estimated using similar data (imputation) or removed entirely if they lacked enough information to be useful. These steps helped maintain the quality and fairness of the dataset, ensuring the analysis and predictions were as accurate and unbiased as possible. Below is the detailed preprocessing steps for the cleaning the data:

- **Loading and Cleaning Data:**
  The datasets for the year 2022, 2023, and 2024 were loaded from CSV files. Columns like ResponseId, which were not needed for analysis, were removed to simplify the data. Duplicate records were also eliminated to ensure there were no redundant entries. Additionally, a new column named Year was added to each dataset to indicate the year from which the data originated, making it easier to distinguish between records. And for career prediction, the datasets for 2023 and 2024 were merged into a single unified dataset for analysis, as the 2022 dataset was not used due to a lack of relevant columns. A copy of this merged dataset was created to ensure that the original data remained intact for potential reuse.

- **Handling Missing or Inconsistent Columns:**
  The datasets had some inconsistencies across years. Below are the steps for handling the missing data and also the inconsistent data:

  - In the 2023 dataset, the column names AIAcc and AIBen were swapped to correct an error.
  - New columns such as AIThreat and AIChallenge were added to the 2023 dataset with placeholder values (NaN) to align its structure with the 2024 dataset.
  - In the 2024 dataset, some AI-related columns were renamed to improve clarity and consistency, such as changing AINextMuch less integrated to AINextVery different.
  - Columns with missing values were filled with appropriate placeholders. For example, Categorical columns received the value "Missing" to indicate the absence of data.

- **Merging Data:**
  After cleaning, the datasets from all three years were combined into a single dataset for predicting the salary but the datasets used for career prediction was used from the year 2023 and 2024. This unified dataset made it possible to perform consistent analysis across multiple years. Merging the data ensured that no important information was missed while maintaining compatibility between years.

- **Feature Categorization:**
  The columns were grouped into categories to make analysis easier. These categories included:

  - Demographics:
    Information such as age, country, and education level.
  - Tech Stack:
    Programming languages, databases, platforms, and tools developers have worked with or want to work with.
  - AIStack:
    Columns related to AI tools, preferences, and future predictions.
  - Community Engagement:
    Details about participation in Stack Overflow.

  This categorization helped in organizing and targeting specific columns for analysis and prediction tasks.

- **Preparing Data for Salary Prediction:**
  Only important columns for salary prediction were chosen. Records missing salary information were removed for accuracy. Salaries in the smallest and largest 2% range were also excluded to avoid outliers. These steps helped make the dataset more reliable for predicting realistic salaries.

- **Transforming Data:**

  - Numerical Columns:
    Columns like YearsCode, YearsCodePro, and WorkExp had values such as "Less than 1 year" or "More than 50 years." These were converted into numerical values for easier analysis.
  - Categorical Columns:
    Missing values in categorical columns were replaced with the placeholder "Missing." Rare values in these columns were grouped into an "Other" category using a rare label encoder. This reduced the complexity of the data and improved the model's ability to process it effectively.

- **Final Adjustment for Modeling:**
  After preprocessing, the data was split into training and testing sets for model evaluation. Categorical columns were identified and indexed to ensure compatibility with the CatBoostRegressor model, which requires special

handling of categorical data. This final step ensured the data was clean, consistent, and ready for predictive modeling.

- **Synthetic Data:**
  Synthetic data is artificially created data that imitates real-world data while addressing its limitations. In career prediction, it helps fill gaps in existing datasets by simulating emerging roles and future scenarios, such as AI-driven jobs or advancements in quantum computing. It allows us to explore hypothetical career trends, overcome data shortages, and train models more effectively. Additionally, synthetic data preserves privacy and enables predictions for roles not yet widely represented in real data, helping us anticipate and prepare for the future job market.

By following these steps, the dataset was transformed into a reliable and well-organized form, ready for accurate analysis and modeling. Each step was carefully designed to handle inconsistencies, remove irrelevant information, and make the data suitable for machine learning tasks.

### 2.3 Example Data Rows

Here are a few rows of the processed data:

| Country | LanguageWorkedWith | YearsCoding | Salary (USD) | JobRole |
|---|---|---|---|---|
| USA | Python, Java | 10 | 120000 | Software Engineer |
| India | JavaScript, C++ | 5 | 25000 | Web Developer |
| Germany | Python, R | 8 | 90000 | Data Scientist |

## 3 Algorithm And Experimental Results:

Below are the algorithms used for this project and the results

### 3.1 CatBoost Regressor:

**Purpose:** Predict developer salaries based on features like programming languages, experience, country and more.

**Description:** CatBoost is a gradient boosting algorithm tailored for categorical data. It effectively handles categorical features without requiring explicit encoding, making it suitable for our dataset. CatBoost automatically encodes categorical features without requiring extensive preprocessing like one-hot encoding or label encoding. CatBoost uses ordered boosting, which reduces overfitting and speeds up the training process. It includes built-in techniques like regularization to minimize overfitting. CatBoost can handle missing data directly, eliminating the need for imputation. [1]

**Steps Involved:**

1. **Input Features:**
   This step involves selecting the most important pieces of information from the dataset that will help the model make accurate predictions. For example:

   - Country: Where the person lives can significantly influence salary due to differences in the cost of living and local job markets.

   - YearsCoding: The number of years a person has been coding reflects their experience level, which often correlates with higher salaries.

   - LanguageWorkedWith: The programming languages a person uses can also affect their salary, as some languages (like Python or Java) are in higher demand and command higher wages.

   After identifying these features, they are processed so the model can understand them. For instance, categorical features (like Country) might be converted into numerical values or encoded in a way that the model can interpret.

2. **Model Training:**
   This is the learning phase where the model tries to understand the relationship between the input features and the target variable (Salary). The process involves:

   - Splitting the data into training and testing sets. The training set is used to "teach" the model, and the testing set is used to check how well it has learned.

   - Configuring the hyperparameters of the model, which are like settings that control how the model learns. For example, in a CatBoost model, hyperparameters like the number of iterations (trees), learning rate (how fast it learns), and tree depth (complexity of decisions) are set to get the best results.

   - Feeding the training data into the model so it can learn patterns and relationships, like how the combination of country, experience, and skills translates into a specific salary.

   This step is iterative, meaning it involves trial and error. The model learns from errors in its predictions and improves over time.

3. **Prediction/Results:**

   Once the model has been trained, it can be used to predict salaries for new, unseen data. For example:

   - If someone provides their country, years of coding experience, and the programming languages they use, the model will use what it learned during training to estimate their likely salary.
   - The output is a numeric value (the predicted salary), which is based on the relationships the model discovered during training.

   Predictions are usually followed by an evaluation step, where the model's accuracy is tested using a separate set of data (testing data). If the model performs well, it means it can generalize to new data and is ready to be used in real-world applications.

   So, the first step is about deciding which information is most important for predicting salaries and preparing it in a way the model can understand. The second step is teaching the model to recognize patterns in the data by using training examples. The final step is using the trained model to make predictions and check how accurate those predictions are.

```
import pandas as pd

# Example data for higher salary prediction
new_data = pd.DataFrame({
    'Employment': ['Employed full-time'],  # Full-time employment
    'OrgSize': ['10,000 or more employees'],  # Large organization size
    'WorkExp': [15],  # Extensive work experience
    'YearsCode': [20],  # Significant coding experience
    'YearsCodePro': [15],  # Long professional coding experience
    'DevType': ['Machine Learning Specialist'],  # High-demand developer type
    'RemoteWork': ['Fully remote'],  # Fully remote
    'MainBranch': ['I am a developer by profession'],  # Professional developer
    'Age': [40],  # Experienced age bracket
    'Country': ['United States'],  # High-paying country
    'EdLevel': ["Doctoral degree"],  # Advanced education level
    'LanguageHaveWorkedWith': ['Python;C++;R'],  # High-demand programming languages
    'LanguageWantToWorkWith': ['Rust;Julia'],  # Niche, high-demand languages
    'DatabaseHaveWorkedWith': ['PostgreSQL;MongoDB'],  # Advanced databases
    'DatabaseWantToWorkWith': ['Neo4j'],  # Cutting-edge databases
    'PlatformHaveWorkedWith': ['AWS;Google Cloud Platform'],  # Cloud platforms
    'PlatformWantToWorkWith': ['Azure'],  # Interest in enterprise platforms
    'WebframeHaveWorkedWith': ['TensorFlow;PyTorch'],  # Specialized frameworks
    'WebframeWantToWorkWith': ['Hugging Face'],  # AI/ML framework
    'MiscTechHaveWorkedWith': ['Kubernetes;Terraform'],  # Infrastructure as code
    'MiscTechWantToWorkWith': ['Ansible'],  # Desired cutting-edge tools
    'ToolsTechHaveWorkedWith': ['VSCode;Jupyter'],  # Common developer tools
    'ToolsTechWantToWorkWith': ['JetBrains'],  # High-end tools
    'NEWCollabToolsHaveWorkedWith': ['Slack;Teams'],  # Collaboration tools
    'NEWCollabToolsWantToWorkWith': ['Zoom'],  # Desired collaboration tools
    'OpSysPersonal use': ['Linux'],  # Preferred operating system
    'OpSysProfessional use': ['Linux'],  # Professional operating system
    'OfficeStackAsyncHaveWorkedWith': ['Notion;Google Docs'],  # Async tools
    'OfficeStackAsyncWantToWorkWith': ['Evernote'],  # Desired async tools
    'OfficeStackSyncHaveWorkedWith': ['Microsoft Office'],  # Sync tools
    'OfficeStackSyncWantToWorkWith': ['Google Workspace']  # Desired sync tools
})

# Ensure data types are consistent with training data
for col in new_data.columns:
    new_data[col] = new_data[col].astype(str)

# Make prediction
y_pred = model.predict(new_data)
print(f"Predicted salary: {round(y_pred[0], 2)} kUSD/year")
```

```
Predicted salary: 85.42 kUSD/year
```

Figure 1: Predicted Salary for the new input data.

**Parameters:**

- Learning rate: 0.1
  The learning rate controls the step size during the optimization process. A value of 0.1 means the model updates its predictions by a fraction of 10% of the gradient in each iteration. This parameter balances the trade-off between convergence speed and stability. A higher learning rate allows the model to converge faster but risks overshooting the optimal solution, while a lower rate ensures stability but may require more iterations to achieve convergence.
- Depth: 10
  The depth specifies the maximum depth of each decision tree in the ensemble. A depth of 10 allows the trees to model complex patterns and interactions in the data, enabling the model to capture intricate relationships between features. However, deeper trees are more prone to overfitting, especially if the dataset is small or noisy. It also increases computational cost and memory usage.
- Iterations: 500
  The number of iterations (or boosting rounds) determines how many trees are constructed during training. With 500 iterations, the model builds 500 sequential trees, each correcting the errors of the previous trees. This ensures that the model progressively improves its predictions. However, too many iterations can lead to overfitting, while too few may result in underfitting.
- Loss function: RMSE
  The loss function defines the objective that the model aims to minimize during training. Root Mean Squared Error (RMSE) is a commonly used metric for regression problems that penalizes larger errors more heavily. It measures the square root of the average squared differences between predicted and actual values, providing a clear sense of how well the model is predicting continuous outcomes. Lower RMSE values indicate better model performance.

**Formula:**[1]

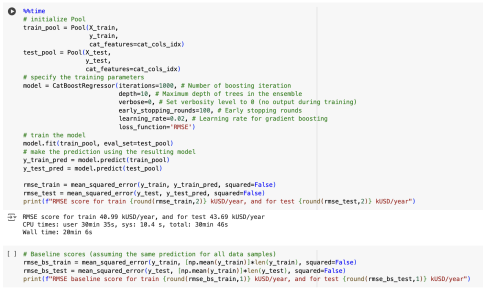
$$\text{Salary} = \text{CatBoost}(\text{Features}, \text{Parameters})$$

4. **Key Observations:**

| Iterations | Depth | Learning Rate | RMSE (Train) | RMSE (Test) | Time Taken(in sec) |
|---|---|---|---|---|---|
| 500 | 6 | 0.1 | 42.239 | 43.709 | 213 |
| 1000 | 8 | 0.05 | 41.161 | 43.544 | 685.5 |
| 2000 | 10 | 0.02 | 39.843 | 43.504 | 2440.5 |

CatBoost Performance Metrics

This table 4 provides insights into the performance and computational efficiency of the CatBoostRegressor model under different parameter configurations. Here's an analysis based on the given data:



```
%%time
# initialize Pool
train_pool = Pool(X_train,
                  y_train,
                  cat_features=cat_cols_idx)
test_pool = Pool(X_test,
                 y_test,
                 cat_features=cat_cols_idx)
# specify the training parameters
model = CatBoostRegressor(iterations=1000, # Number of boosting iteration
                          depth=10, # Maximum depth of trees in the ensemble
                          verbose=0, # Set verbosity level to 0 (no output during training)
                          early_stopping_rounds=100, # Early stopping rounds
                          learning_rate=0.02, # Learning rate for gradient boosting
                          loss_function='RMSE')
# train the model
model.fit(train_pool, eval_set=test_pool)
# make the prediction using the resulting model
y_train_pred = model.predict(train_pool)
y_test_pred = model.predict(test_pool)

rmse_train = mean_squared_error(y_train, y_train_pred, squared=False)
rmse_test = mean_squared_error(y_test, y_test_pred, squared=False)
print(f"RMSE score for train {round(rmse_train,2)} kUSD/year, and for test {round(rmse_test,2)} kUSD/year")
```

```
RMSE score for train 40.99 kUSD/year, and for test 43.69 kUSD/year
CPU times: user 30min 35s, sys: 18.4 s, total: 30min 46s
Wall time: 20min 6s
```

```
# Baseline scores (assuming the same prediction for all data samples)
rmse_bs_train = mean_squared_error(y_train, [np.mean(y_train)]*len(y_train), squared=False)
rmse_bs_test = mean_squared_error(y_test, [np.mean(y_train)]*len(y_test), squared=False)
print(f"RMSE baseline score for train {round(rmse_bs_train,1)} kUSD/year, and for test {round(rmse_bs_test,1)} kUSD/year")
```

```
RMSE baseline score for train 67.4 kUSD/year, and for test 67.4 kUSD/year
```

Figure 2: CatBoostRegressor Code

- Training RMSE Decreases with Increased Iterations and Depth:
  The RMSE (Train) decreases as the number of iterations and depth increase: For 500 iterations and depth 6: RMSE (Train) = 42.239
  For 2000 iterations and depth 10: RMSE (Train) = 39.843
  This indicates that the model is learning more complex patterns in the training data as it becomes more expressive with greater iterations and depth.
- Test RMSE Stabilizes Across Configurations:
  The RMSE values remain relatively stable. This suggests the model may already be close to its optimal performance for this dataset.
- Computational Cost Increases Significantly:
  The time taken grows subtantially as iterations and depth increases.

The configuration with 1000 iterations, depth 8, and learning rate 0.05 offers the best trade-off between performance and computational cost. Further increasing iterations and depth provides marginal benefits in test performance but results in significantly higher training times. This analysis underscores the importance of balancing model complexity with computational efficiency for practical applications.

The Root Mean Squared Error (RMSE) results provide insights into the performance of the CatBoostRegressor model across various parameter configurations. The RMSE on the training set decreases as the number of iterations and depth increase, reflecting improved model learning. For instance, with 500 iterations and a depth of 6, the training RMSE is 42.239, while with 2000 iterations and a depth of 10, it improves to 39.843. However, the test set RMSE remains relatively stable across configurations, ranging from 43.709 to 43.504, indicating that increasing model complexity yields diminishing returns on unseen data. This suggests that the model may already be near optimal for this dataset. The computational cost, however, rises significantly with increased iterations and depth, from approximately 213 seconds for the simplest configuration to over 2440 seconds for the most complex. For a balance between performance and efficiency, the configuration with 1000 iterations, a depth of 8, and a learning rate of 0.05 appears to be the most practical choice, offering a competitive test RMSE of 43.544 with considerably less computational cost than the more complex alternatives. Further improvements could focus on tuning additional hyperparameters, such as regularization parameters, or enhancing feature engineering to improve test performance.

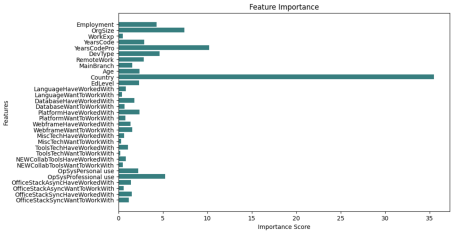



Figure 3: Feature Importance of the dataset

5. **Advantages:**
   - Handles missing data automatically.
   - High performance with categorical variables.
   - Handles Imbalanced Data.

## 3.2 K-Means Clustering

**Purpose:** Group respondents based on their age and yearly compensation (ConvertedCompYearly)

**Description:**
K-Means is an unsupervised learning algorithm that partitions data into clusters based on similarity. It iteratively assigns data points to clusters and updates the cluster centroids to minimize the variance within clusters. The optimal number of clusters is determined using the Elbow Method. This approach is particularly effective for exploring patterns in numerical data, such as age and compensation in this dataset. [2] [8]

**Steps Involved:**

1. **Initialization:** The algorithm begins by selecting a predefined number of clusters, k. It

then randomly assigns initial positions (centroids) for each cluster within the dataset. These centroids represent the center of each cluster and are crucial for grouping the data..

2. **Assignment of Data Points:** Each data point in the dataset is assigned to the nearest centroid based on a distance metric, typically Euclidean distance. This forms an initial grouping of data points into clusters, where each point belongs to the cluster with the closest centroid.

3. **Centroid Update:** Once all points are assigned to clusters, the centroids are recalculated. The new centroid for each cluster is computed as the mean (average) of all the data points assigned to that cluster. This step ensures that the centroid accurately represents its cluster.

4. **Iteration:** The assignment and update steps are repeated iteratively. Data points may shift from one cluster to another as the centroids are updated and become more accurate. This process continues until the centroids stabilize and no longer change significantly, or a maximum number of iterations is reached.

5. **Convergence:** The algorithm stops when the centroids remain constant or the movement of centroids falls below a predefined threshold. At this point, the clusters are finalized, and each data point is permanently assigned to a cluster.

# 4 Experimental Results:

Here are some of the visualization by performing k-means algorithm.

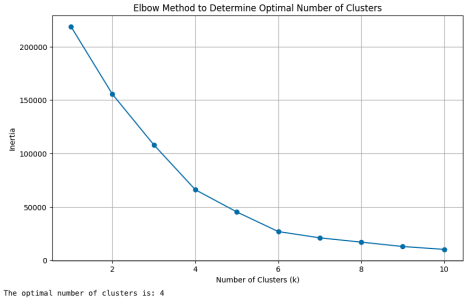



Figure 4: Elbow Method for finding optimal clusters.

From figure 4, elbow plot is used to identify the optimal number of clusters (k) for the K-means algorithm by analyzing the inertia (sum of squared distances of samples to their nearest cluster center). We can see the decrease in inertia as the number of clusters increases, reflecting better cluster separation. The "elbow" in the curve is observed at k=4. Beyond k=4, additional clusters yield diminishing returns in terms of compactness and separation. This scatter plot visualizes the results of a K-means clustering algorithm applied to the stackoverflow dataset with features Age and ConvertedCompYearly (yearly compensation). The color coding represents different clusters identified by the algorithm. The data points are grouped into four clusters based on similarities in age and compensation levels. The x-axis(scaled salary) shows a wide range of annual compensation levels and, the y-axis labels represent distinct age groups. The algorithm successfully identifies meaningful clusters, distinguishing groups based on salary and age patterns. From figure 5, high-income earners are generally associated with older professionals, while younger age groups dominate the lower-income clusters.

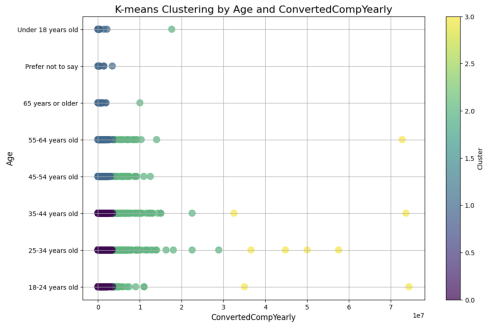



Figure 5: Age VS Salary

```
from sklearn.metrics import silhouette_score

silhouette_avg = silhouette_score(scaled_data, df['Cluster'])
print(f'Silhouette Score: {silhouette_avg:.2f}')
```

```
Silhouette Score: 0.64
```

Figure 6: Silhoutte Score for Age Vs Salary

A Silhouette Score of 0.64 suggests that the clusters are distinct enough to draw insights, but there might still be some overlap between groups. Overall, the clustering result is quite satisfactory.

**Parameters:**

- Number of clusters: 4
- Inertia: Measures the compactness of clusters; lower values are better.
- Silhoutte Score: Evaluates how well-separated the clusters are (range: -1 to 1). A score of 0.64 indicates moderate clustering quality.

**Formula:**

$$\sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2$$

Where:

- $S_i$: Cluster $i$.
- $\mu_i$: Centroid of cluster $i$. [8]

## 4.1 RAG model

**Purpose:** Predict potential future job roles using a Retrieval-Augmented Generation (RAG) framework with synthetic datasets.

**Description:** The Stack Overflow Developer Survey data is converted into a vector database. The LLM is pretrained with synthetic job roles, as discussed in the section below. When a user sends a query, FAISS indexes are generated for both the vector database and the LLM. The most similar chunks from these sources are retrieved and combined to provide relevant and informed predictions about potential job roles. [4] [7]

**Some terms for understanding the pipeline:**

- FAISS Index(Facebook AI Similarity Search): FAISS is a library designed for fast similarity search. The prepared data is indexed so that it can be searched quickly and efficiently for similar items (e.g., finding the closest matching data points). This indexing step allows for faster retrieval of relevant data, which is crucial for tasks like query processing.[6]
- RAG(Retrieval-Augmented Generation): Retrieval-Augmented Generation (RAG) is a machine learning framework that combines retrieval-based techniques and generative models to produce more accurate, contextually relevant, and factually grounded responses. It leverages external knowledge sources during generation to improve the quality and reliability of the output.[4] [7]
- Fine-Tuning: Fine-tuning is a powerful and efficient way to adapt pre-trained models to specific tasks or domains by refining their performance with smaller, specialized datasets. It combines the advantages of general-purpose pre-trained models with the ability to handle targeted applications effectively.[5]

**Steps Involved:**

1. **Text Embedding Preparation:** It combines various text features from the dataset into a single string per row. It uses SentenceTransformer model to create embeddings(numerical representations) of these combined text fetaures for semantic understanding. It is used to process and represent the textual data in machine-readable format for similarity-based retrieval.
2. **Numerical Feature Normalization:** In this step, we process numerical features to handle missing values and standardize the data using StandardScalar. It's purpose is to ensure that the numerical features are on the same scale which helps in improving model performance during similarity calculations.
3. **Combine Text and Numerical Embeddings:** After that we combine the text embeddings and the normalized numerical embeddings into unified row representations. It's purpose is to create a single, comprehensive representation of each row for better retrieval and similarity matching.
4. **FAISS Index Creation:** We construct a FAISS Index(a libraby for fast similarity search) using the combined embeddings. It allows quick and efficient retrieval of the most similar rows in the dataset based on the input queries.
5. **Retrieve Context:** It encodes a query into embeddings, searches FAISS Index for the most similar rows, and retrieves related contexts.
6. **Generate Synthetic Data:** To address the absence of predefined future job roles for training the LLM, we adopted an innovative approach by querying ChatGPT with records from the developer survey data. Using the most relevant columns for 50 records, we asked ChatGPT to suggest potential new job roles that could emerge in the future. This approach generated a mix of outputs—some roles were logical and plausible, while others were unrealistic or whimsical. We then used the synthesized data to pretrain the LLM, enabling it to better understand and predict potential future job roles. The synthesized job roles generated through this method are presented in the figure below. [5]
7. **Split Data into Training and Testing:** We then split the synthetic dataset into 5-folds for training and testing(80% training and 20% testing).
8. **Fine-Tuning GPT-2:** We then load the synthetic dataset and fine-tune the GPT-2 model using Hugging Face

```
roles = [
    "AI Healthcare Strategist", "Cloud Automation Engineer", "Blockchain System Architect",
    "Sustainable Tech Consultant", "Quantum AI Researcher", "IoT Cybersecurity Specialist",
    "Retail Data Scientist", "Space Exploration Analyst", "Autonomous Vehicle Programmer",
    "AI Ethics Consultant", "Digital Twin Specialist", "Metaverse Architect",
    "AI-Powered Climate Analyst", "Neurocomputing Engineer", "Augmented Reality Developer",
    "Synthetic Biology Data Scientist", "AI-Driven Robotics Trainer", "Personalized Education Technologist",
    "Renewable Energy Data Engineer", "AI Trust and Safety Specialist", "Quantum Cryptography Expert",
    "Next-Gen IoT Solution Designer", "AI-Powered Policy Analyst", "Autonomous Drone Fleet Manager",
    "Sustainable Supply Chain Engineer", "Blockchain-Powered Data Analyst", "Holographic Interface Designer",
    "AI-Based Virtual Therapist", "Space Resource Utilization Analyst", "Cybersecurity Threat Hunter",
    "Human-Machine Interaction Designer", "Green Technology Innovator", "AI Governance Specialist",
    "Predictive Maintenance Analyst", "Biometric Security Consultant", "AI-Powered Legal Advisor",
    "AI-Enhanced Content Creator", "Edge Computing Specialist", "Personal AI Trainer",
    "Virtual Reality Safety Officer", "AI-Powered Renewable Energy Forecaster",
    "Advanced AI Model Interpreter", "Data Privacy Strategist", "Emotion AI Developer",
    "Quantum Machine Learning Engineer", "AI-Enabled Urban Planner", "Digital Wellbeing Designer",
    "Energy Optimization AI Consultant", "AI for Accessibility Developer", "Digital Asset Management Specialist"
]
```

Figure 7: Synthetic Data

Trainer. The model is trained for 3 epochs with specific learning rates(5e-5) and batch sizes(2). Its purpose is to adapt the GPT-2 model for generating relevant job roles based on queries and contexts. [5]

9. **Save the Fine-Tuned Model:**
We then save the fine-tune model GPT-2 and tokenizer to a local directory. The purpose is to reuse the fine-tune model for predictions for further training.

10. **Evaluate Model Accuracy:**
We then generate responses for the test set using the fine-tuned model. We compare the generated responses with the ground truth using Cosine-Similarity.

11. **Pipeline:**

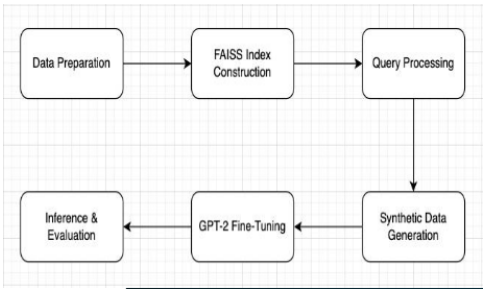



Figure 8: Pipeline

First, the Data Preparation stage processes both textual and numerical features, converting them into embeddings using the SentenceTransformer model and StandardScaler for normalization. These embeddings are combined into unified row representations and indexed in FAISS for efficient similarity-based retrieval. During Query Processing, a query is encoded into embeddings and matched against the FAISS index to retrieve the most relevant dataset rows. These retrieved contexts, along with predefined queries and job roles, are used to generate synthetic training data in the form of prompt-response pairs. The pipeline then moves to Fine-Tuning, where a pre-trained GPT-2 model is trained on this synthetic dataset to learn how to generate relevant job roles for given prompts. Finally, in the Inference Evaluation stage, the fine-tuned model predicts job roles for new queries, with BLEU scores and cosine similarity metrics used to evaluate its accuracy and semantic relevance. This seamless integration of retrieval and generation ensures accurate and contextually relevant job role recommendations.

12. **Results:**
Our RAG model is highly specific to the synthetic data it was trained on. Regardless of the query, it generates a future job role prediction. As shown in figure 9, a relevant query produces a meaningful result. However, in figure 10, even an unrelated query like "Hello Class" results in a job prediction. This behavior highlights the need for further enhancements. Limitations in our understanding of RAG and LLMs, coupled with scope and time constraints, were key factors contributing to these outcomes. We can see from the figure 11,the cosine similarity score of approximately 0.39 indicates a moderate level of alignment between the generated responses and the ground truth responses in the test dataset. While the results show some similarity, the generated outputs are not highly aligned with the expected responses, suggesting room for improvement in the generator's ability to produce contextually or semantically accurate outputs.

```
prompts = [
    "Query: AI in cloudcomputing",
    "Query: Cloud computing\nContext: Cloud Engineer\nPropose job roles:",
]

for prompt in prompts:
    response = generator(prompt, max_length=100, num_return_sequences=1)
    print("Prompt:", prompt)
    print("Generated Response:", response[0]["generated_text"])
    print("---")
```

```
Prompt: Query: AI in cloudcomputing
Generated Response: Query: AI in cloudcomputing
Context: Educator
Propose job roles:Quantum Cryptography Expert, Energy Optimization AI Consultant
---
Prompt: Query: Cloud computing
Context: Cloud Engineer
Propose job roles:
Generated Response: Query: Cloud computing
Context: Cloud Engineer
Propose job roles:Virtual Reality Safety Officer, AI-Powered Climate Analyst
---
```

Figure 9: Prompt 1

```
prompts = [
    "Query: Hello Class"
]

for prompt in prompts:
    response = generator(prompt, max_length=100, num_return_sequences=1)
    print("Prompt:", prompt)
    print("Generated Response:", response[0]["generated_text"])
    print("---")
```

```
Prompt: Query: Hello Class
Generated Response: Query: Hello Classifier
Context: nan
Propose job roles:Energy Optimization AI Consultant, Digital Twin Specialist
---
```

Figure 10: Prompt 2


Figure 11: Cosine Similarity

# 5 More Experimental Results:

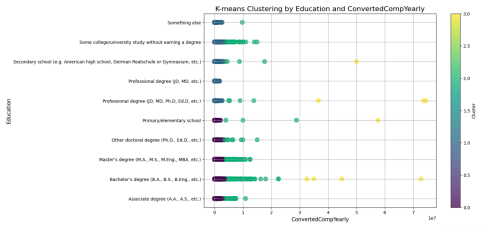



Figure 12: Clustering Based on Education Vs Salary

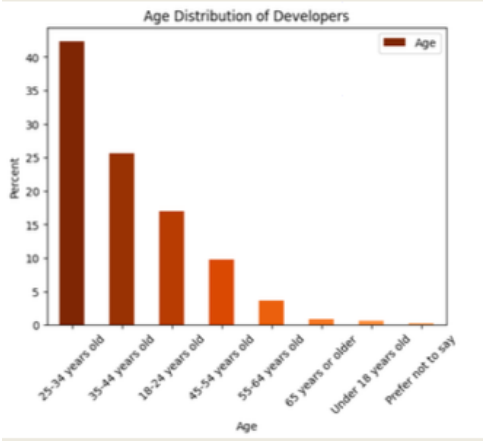



Figure 13: Age Distribution Of Developers

## 5.1 Observations from the graphs:

- From figure 13, Developers aged 25–34 dominate ( 40%), followed by 35–44 (30%), reflecting a young, active workforce. Also, minimal representation for those aged Under 18, 55+, or who prefer not to disclose.
- From figure 14, USA has the highest total compensation, surpassing other countries by a wide margin. (Maybe the reason many developers want to move to USA). Germany, UK, and Canada follow, with France having the least among the top 5.

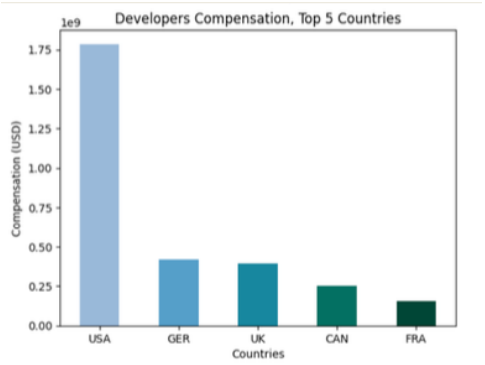



Figure 14: Compensation Based On Country

| | DevType | Percentage |
|---|---|---|
| cell output actions | Developer, full-stack | 35.84 |
| 1 | Developer, back-end | 19.60 |
| 2 | Developer, front-end | 6.62 |
| 3 | Developer, desktop or enterprise applications | 5.10 |
| 4 | Developer, mobile | 3.73 |
| 5 | Developer, embedded applications or devices | 2.81 |
| 6 | Other (please specify): | 2.18 |
| 7 | Engineering manager | 2.10 |
| 8 | Data scientist or machine learning specialist | 1.75 |
| 9 | DevOps specialist | 1.56 |
| 10 | Research & Development role | 1.52 |
| 11 | Senior Executive (C-Suite, VP, etc.) | 1.26 |
| 12 | Student | 1.23 |

Figure 15: DevType Percentile

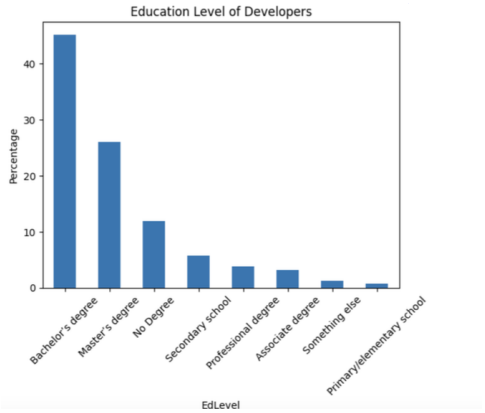



Figure 16: Education Of Developers

- From figure 15, Full-Stack Developers (35.84%) are the largest specialization, followed by Back- End Developers (19.6%) followed by Front- End Developers(6.62%). Emerging fields like Data Science (1.75%) and DevOps (1.56%) are small but significant.
- From figure 16, most developers have a bachelor's degree ( 40%), followed by master's de-

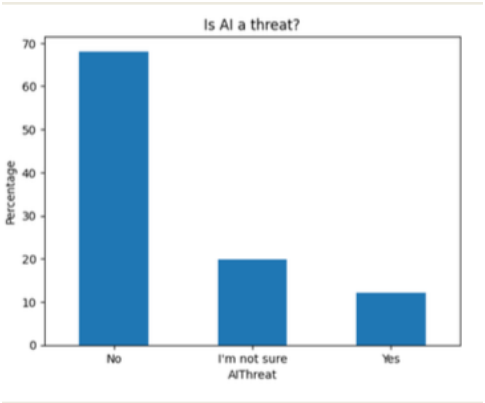



Figure 17: IS AI a Threat

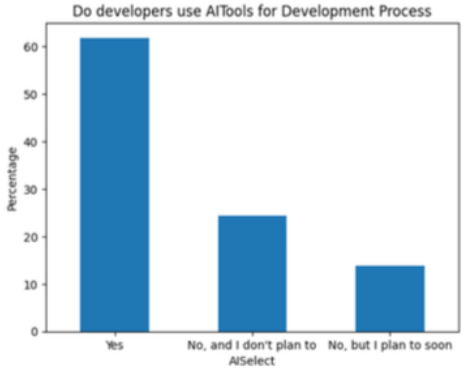



Figure 18: AITools for Development Process

| | language have worked with | Percent |
|---|---|---|
| 0 | JavaScript | 63.32 |
| 1 | HTML/CSS | 51.32 |
| 2 | SQL | 50.95 |
| 3 | Python | 44.76 |
| 4 | TypeScript | 42.26 |
| 5 | Bash/Shell (all shells) | 32.36 |
| 6 | Java | 29.55 |
| 7 | C# | 28.10 |
| 8 | C++ | 19.81 |
| 9 | PHP | 18.30 |

Figure 19: Language Have Worked With

| | language want to work with | Percent |
|---|---|---|
| 0 | JavaScript | 39.03 |
| 1 | TypeScript | 37.25 |
| 2 | Python | 36.19 |
| 3 | SQL | 35.47 |
| 4 | HTML/CSS | 33.11 |
| 5 | Rust | 28.83 |
| 6 | Go | 22.09 |
| 7 | C# | 21.51 |
| 8 | Bash/Shell (all shells) | 20.52 |
| 9 | Java | 16.18 |

Figure 20: Language Want To Work With

| | OS | Percent |
|---|---|---|
| 0 | Windows | 44.78 |
| 1 | MacOS | 35.29 |
| 2 | Ubuntu | 27.34 |
| 3 | Windows Subsystem for Linux (WSL) | 16.17 |
| 4 | Debian | 8.20 |
| 5 | Android | 8.19 |
| 6 | iOS | 7.63 |

Figure 21: Operating System Used

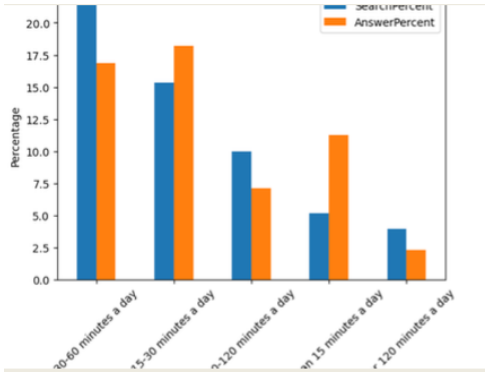



Figure 22: StackOverflow Searching And Answering

grees ( 20%). Although, it says education, but we don't know if the education is related to CS. Secondary school graduates and professionals contribute marginally.

- From figure 17, 68% of developers do not see AI as a threat. 19% are unsure, and 13% perceive it as a risk, indicating a generally optimistic view of AI in the tech industry.
- From figure 18, 61% of developers already use AI tools, indicating strong adoption. 20% plan to adopt AI tools soon, emphasizing AI's growing role in productivity and development workflows.
- From figure 19 and figure 20, JavaScript leads with 63.32%, followed by HTML/CSS

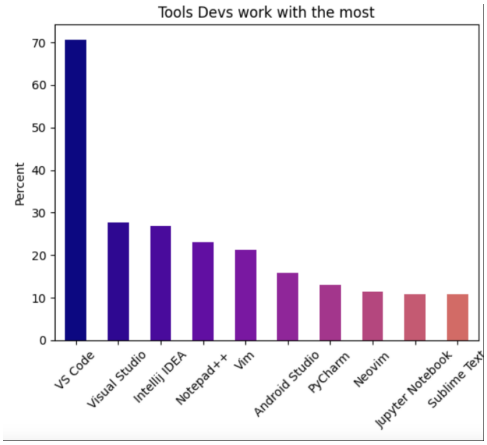



Figure 23: Tools Developers Work With

| | RemoteWork | Percentage |
|---|---|---|
| 0 | Hybrid (some remote, some in-person) | 35.30 |
| 1 | Remote | 33.54 |
| 2 | In-person | 15.00 |

Figure 24: Work Preference

(51.32%) and SQL (50.95%). Backend and scripting languages (e.g., Python, Java) maintain strong usage. JavaScript, TypeScript, and Python dominate, showing future demand trends. Rising languages like Rust (28.83%) and Go (22.09%) indicate shifts in developer interests.

- From figure 21, Windows (44.78%) is the most-used OS, followed by MacOS (35.29%). And, Ubuntu is the most popular Linux distribution (27.34%).
- From figure 22, Developers spend 30-60 minutes/day searching and answering questions. Searching occupies slightly more time than answering.
- From figure 23, VSCode is the most popular development tool, followed by Visual Studio and Intellij IDEA.
- From figure 24, 35.3% prefer hybrid work, 33.54% fully remote, and 15% in- person. It reflects a strong trend toward remote or hybrid work in the tech industry.

# 6 Problems and Limitations

- **Synthetic Data Limitation:**
  Working with non-existent job roles required creating synthetic data, which introduced challenges in ensuring the plausibility, diversity, and relevance of the roles. This reliance on assumptions and speculative trends limited the realism and generalizability of the model's predictions.
- **Future Uncertainty:**
  Predicting future job roles inherently involves uncertainty and speculation. The model is susceptible to generating unrealistic roles or omit important emerging trends.
- **Computational Limitations:**
  Most deep learning models, especially those with large datasets or complex architectures, such as CatBoost, demand substantial GPU power to train efficiently. Limited availability of GPU slows down training, hence prolonging experimentation and deployment timelines.

  For very large datasets or models, a single GPU may be insufficient, necessitating distributed training across multiple GPUs or nodes, which adds complexity and cost.

  Large-scale models, like CatBoost or deep generative models, have to load enormous datasets into RAM. Lack of RAM leads to swapping data between storage and memory, which is extremely slow.
- **Generative Model Constraints:**
  GPT-2 generates responses based on patterns and distributions observed in its training data. If the training data contains repetitive or widely used content, the model's outputs may lack originality or novelty.

  In certain cases, the model may repeat ideas or phrasing within a response, leading to outputs that feel redundant or less engaging.

## 6.1 Lesson Learnt:

- Effective Data Preprocessing:
  Cleaning and transforming data is essential for improving model accuracy and ensuring fairness in predictions. Handling missing or inconsistent data across years highlighted the importance of data alignment.
- Importance of Feature Engineering:
  Categorizing features (e.g., demographics, tech stack) and transforming categorical and numerical columns helped optimize model performance.
- Synthetic Data Utility:
  Synthetic data proved valuable for filling gaps

in real-world datasets and simulating future trends, especially for speculative tasks like career predictions.

- Evaluation Metrics Matter:
  Using metrics like RMSE for regression and Silhouette Score for clustering provided critical insights into model performance and cluster quality.

- Adaptability of RAG Framework:
  The RAG pipeline effectively combines retrieval and generation for accurate job role predictions.

### 6.2 Improvements

- Enhanced Synthetic Data Generation:
  Use more advanced techniques for synthetic data creation, such as conditional generative models, to ensure better diversity and realism.

- Optimize Computational Resources:
  Employ distributed training systems or utilize cloud-based GPUs to handle large-scale models efficiently.

- Fine-Tuning Generative Models:
  Explore more advanced language models like GPT-3 or GPT-4, which might generate more novel and contextually relevant responses.

- Improved Metrics:
  Use additional evaluation metrics, such as F1 score or cosine similarity, for a more comprehensive assessment of model performance.

## 7 Conclusion

The project successfully demonstrated the potential of using advanced machine learning techniques, such as CatBoost Regressor, K-Means Clustering, and the RAG framework, to analyze developer data and predict emerging trends in career roles and salaries. By leveraging data from multiple years and generating synthetic data, the study provided insights into the impact of factors like age, tech stack, and education on salaries.

However, challenges such as computational limitations, synthetic data realism, and generative model constraints highlighted the need for continuous improvement in pipeline design and modeling techniques. Future iterations of this project could focus on enhancing computational efficiency, improving synthetic data generation, and utilizing more advanced generative models for better predictions. Ultimately, this study lays the groundwork for applying AI in real-world career and trend prediction tasks, helping individuals and organizations adapt to a rapidly evolving job market.

## References


[1] GeeksforGeeks, "Regression Using CatBoost," Accessed: Dec. 2024. [Online]. Available: https://www.geeksforgeeks.org/regression-using-catboost/

[2] GeeksforGeeks, "K-Means Clustering Introduction," Accessed: Dec. 2024. [Online]. Available: https://www.geeksforgeeks.org/k-means-clustering-introduction/

[3] Stack Overflow, "Stack Overflow Developer Survey," Accessed: Dec. 2024. [Online]. Available: https://survey.stackoverflow.co/

[4] Amazon Web Services, "What is Retrieval-Augmented Generation?" Accessed: Dec. 2024. [Online]. Available: https://aws.amazon.com/what-is/retrieval-augmented-generation/

[5] OpenAI, "Fine-Tuning Guide," Accessed: Dec. 2024. [Online]. Available: https://platform.openai.com/docs/guides/fine-tuning

[6] FAISS, "FAISS: A library for efficient similarity search and clustering of dense vectors," Accessed: Dec. 2024. [Online]. Available: https://faiss.ai/index.html

[7] Google Cloud, "Retrieval-Augmented Generation (RAG) Use Case," Accessed: Dec. 2024. [Online]. Available: https://cloud.google.com/use-cases/retrieval-augmented-generation

[8] Wikipedia, "K-means clustering," Accessed: Dec. 2024. [Online]. Available: https://en.wikipedia.org/wiki/K-meansclustering