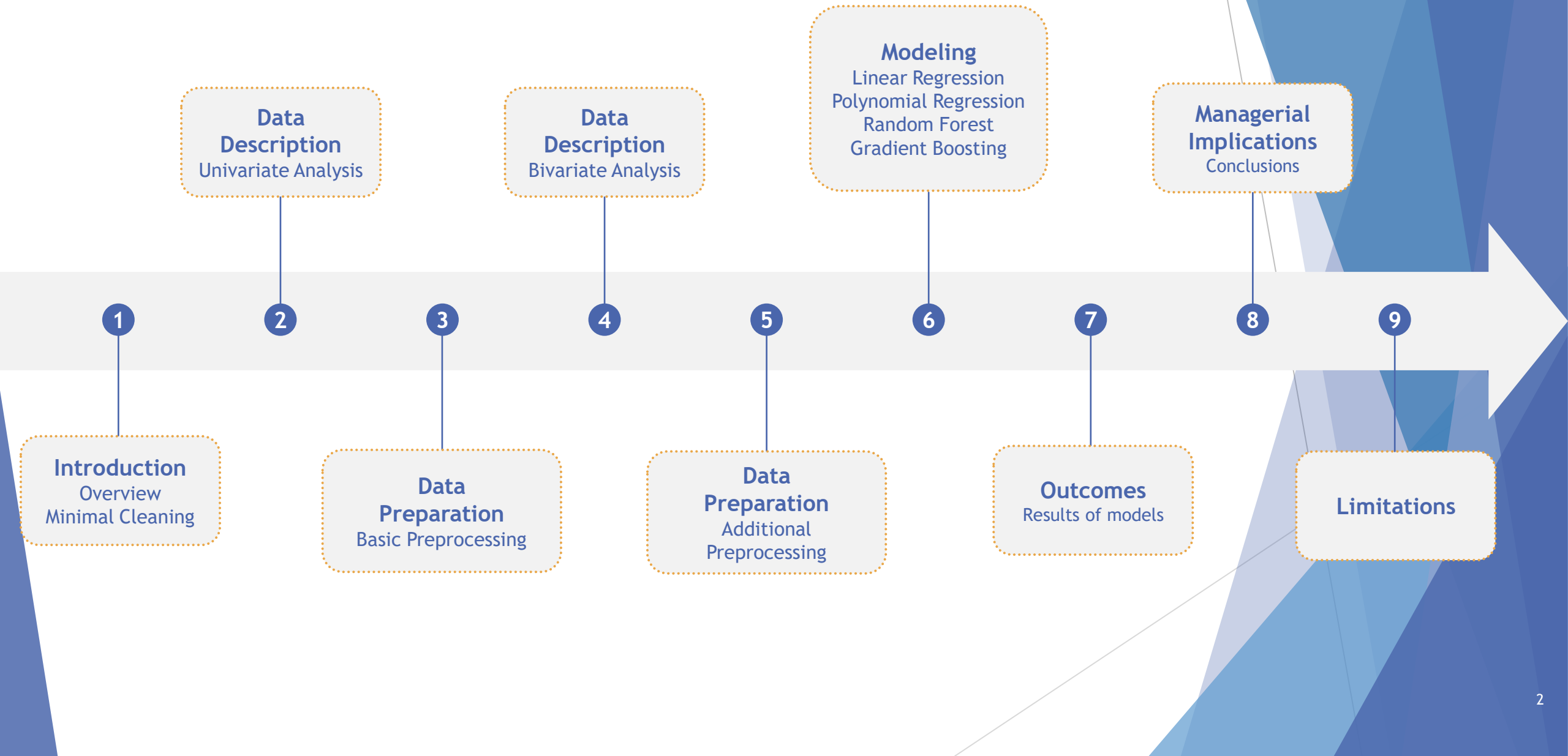
# Benchmarking Developer Compensation in the Age of AI Agents: A Stack Overflow 2025 Analysis

**Big Data & Databases Group Project**


| | |
|---|---|
| Lavinia Benetollo | 3265865 |
| Alessandro Ferraiolo | 3238219 |
| Giulia Marcantonio | 3307112 |
| Giulia Valentini | 3225807 |
| Vittorio Manfriani | 3245184 |


# Agenda


1
Introduction
Overview
Minimal Cleaning
2
Data Description
Univariate Analysis
3
Data Preparation
Basic Preprocessing
4
Data Description
Bivariate Analysis
5
Data Preparation
Additional Preprocessing
6
Modeling
Linear Regression
Polynomial Regression
Random Forest
Gradient Boosting
7
Outcomes
Results of models
8
Managerial Implications
Conclusions
9
Limitations

# 1. Introduction

# Overview

The dataset used in this project is the **2025 Stack Overflow Developer Survey**, the largest yearly survey of the global developer community. It provides anonymized information on developers' backgrounds, work environments, learning habits, technology stacks and, crucially for this study, their adoption and perception of AI tools and AI agents.
The raw data contain **49,191 respondents and 172 variables**. Each row corresponds to a respondent, and each column to a survey question; free-text answers are excluded, as they cannot be directly used for structured analysis. The dataset includes a heterogeneous mix of:

- **Numerical variables** – e.g. age, years of coding experience, firm size, annual compensation.
- **Single-select categorical variables** – e.g. employment status, work arrangement, trust in AI outputs.
- **Multi-select categorical variables** – e.g. programming languages, cloud platforms, AI models and tools used.

The project is framed from the perspective of a **software company** that wants to (i) **reassess internal salaries and hiring plans** in line with market benchmarks to reduce turnover risk and (ii) understand whether claims of higher productivity with AI justify higher pay. Accordingly, the research question is twofold:

- **How has the developer job market evolved, and which factors most strongly drive compensation?**
- **What is the *actual* impact of AI adoption on developers' salaries, and are our hypothetical offers below the market for similar profiles using AI?**

Answering these questions will help the company design salary adjustments and hiring strategies that are both competitive and evidence-based.

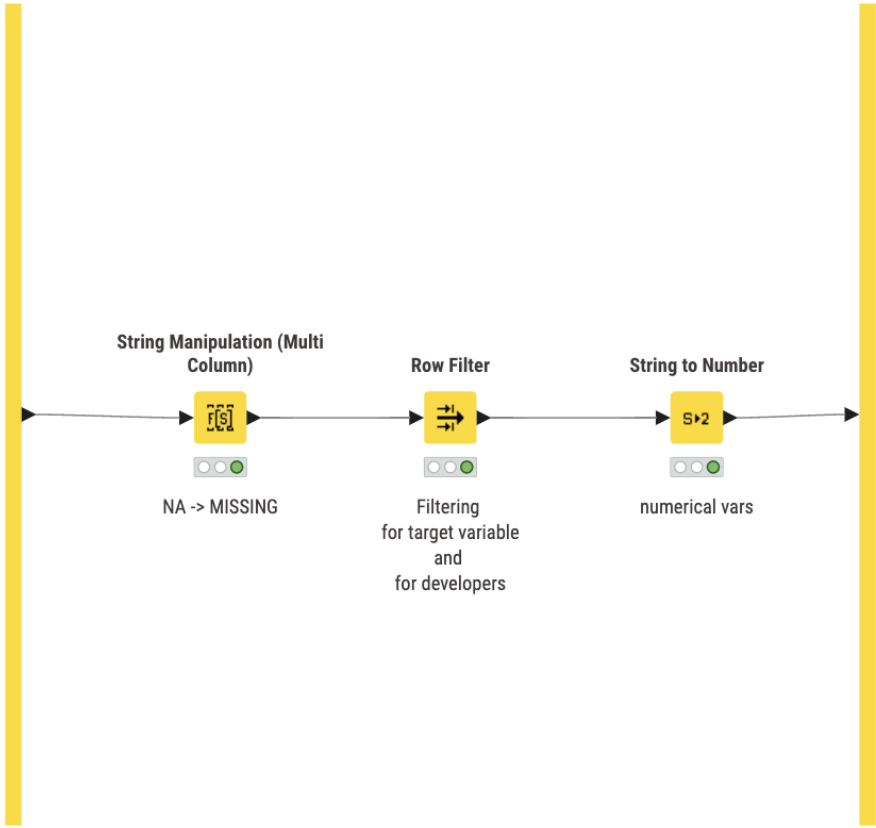
# Minimal cleaning

All occurrences of the string **“NA”**, which appear whenever a respondent leaves a question unanswered, were converted into proper missing values.

A **Row Filter** was applied to keep only relevant and complete observations: entries with non-missing *ConvertedCompYearly*, *MainBranch = “I am a developer by profession”*, and *DevType* containing a developer-related role (regex “*.Dev.*”).

Selected numerical fields (**WorkExp, YearsCode, ConvertedCompYearly**) were converted from text to numeric format using the **String to Number** node.

These essential steps ensure a consistent and reliable base dataset for further cleaning and modeling.



# 2. Data description

## UNIVARIATE ANALYSIS

# Overview

Before conducting detailed data preparation and feature engineering steps, the **univariate analysis** provides an initial understanding of each variable in the dataset. In the context of the Stack Overflow Developer Survey, we specifically explore

We begin by examining the main variables that are theoretically expected to influence developers' yearly compensation, based on preliminary assumptions regarding their potential relevance for salary prediction.

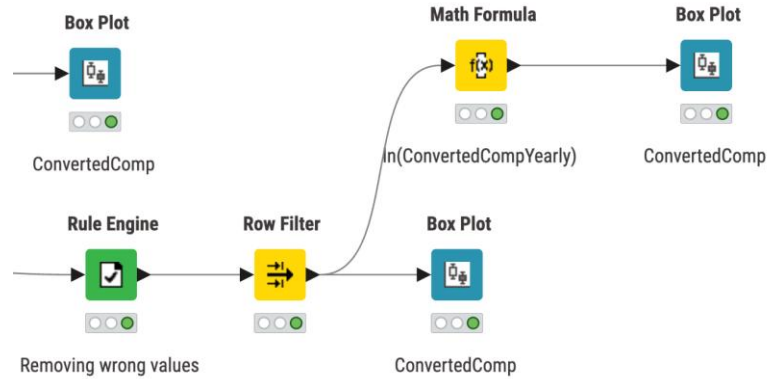
In the context of the Stack Overflow Developer Survey, we also explore variables that are more closely tied to developers' professional characteristics (e.g., programming languages) which may have a direct impact on compensation.

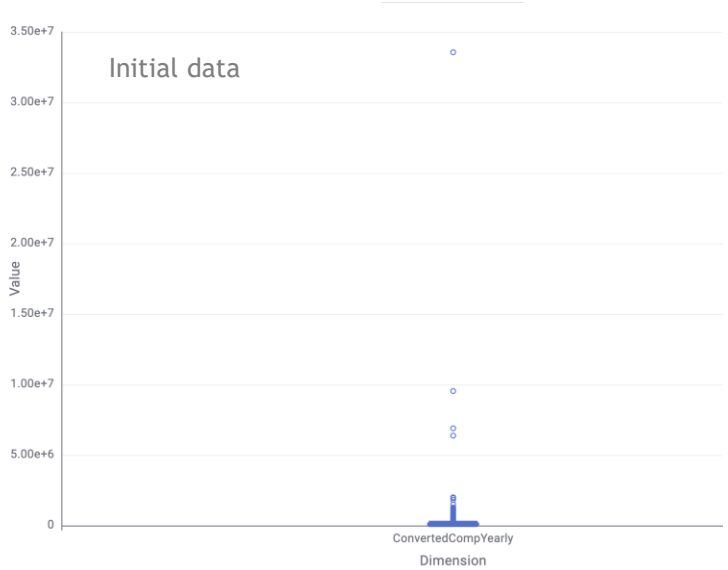
Finally, given the growing role of AI in software development, we include a set of behavioural variables related to AI usage and adoption to assess how they might influence or interact with compensation patterns.

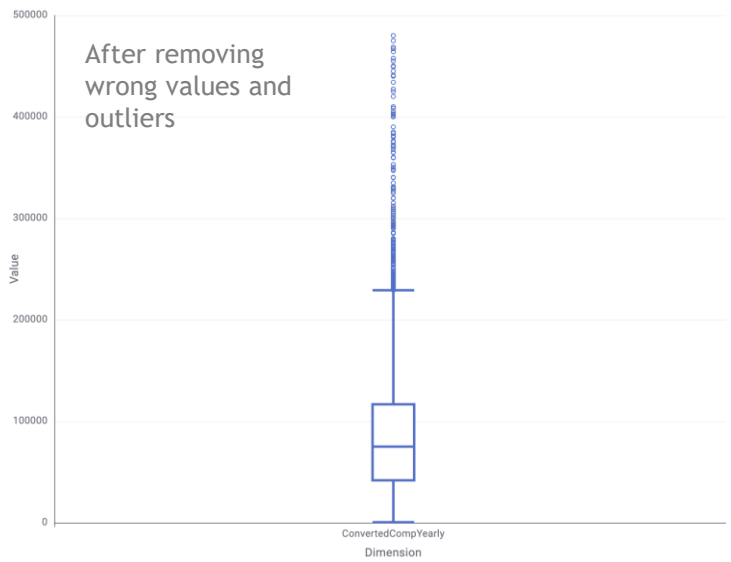
# ln(ConvertedCompYearly)
## target variable

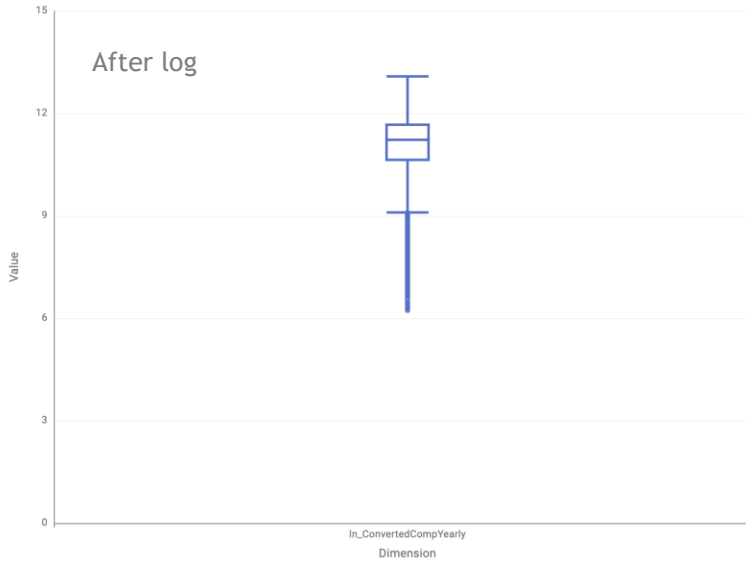
**Nature:** Numerical, continuous

**Description:** Self-reported total yearly compensation (USD)




**Insights:**
Before analysis, implausible compensation values and outliers were removed. Based on global salary benchmarks for software developers, entries **below 500 USD** or **above 500,000 USD** were considered unrealistic. A **Rule Engine** was used to replace these extreme values with 0, and a **Row Filter** subsequently removed them from the dataset to prevent distortion.
The raw distribution of yearly compensation was highly right-skewed, with several extreme outliers. After filtering unrealistic values, the distribution remained skewed but more stable. To address skewness and improve suitability for modelling, the **natural logarithm of ConvertedCompYearly** was computed. The log-transformed variable displays a more symmetric distribution and reduces the impact of high-income outliers.

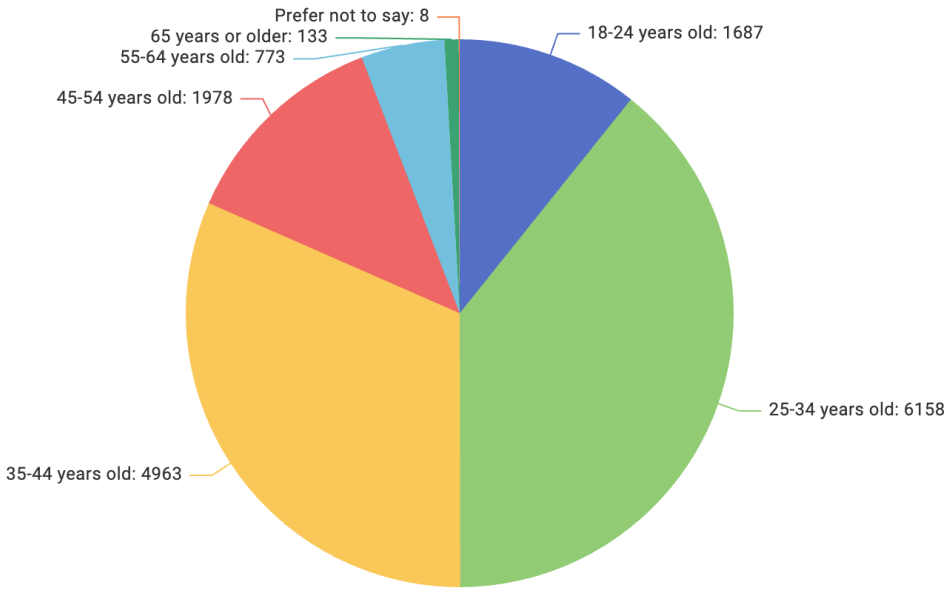
# Main Branch

**Nature:** Categorical, nominal, single-select.
**Description:** Respondents' primary identification regarding their relationship with software development

**Categories (in the raw dataset):**
- I am a developer by profession
- I am not primarily a developer, but I write code sometimes
- I used to be a developer
- I am learning to code
- I code primarily as a hobby
- I work with developers but am not a developer
- None of these

**Insights:**
In the original dataset, MainBranch captures a broad spectrum of roles, ranging from professional developers to hobbyists and learners. For this project, we **retain only respondents who identify as "developer by profession"**, ensuring that the analysis focuses exclusively on individuals who use software development in their primary occupation.

# Age



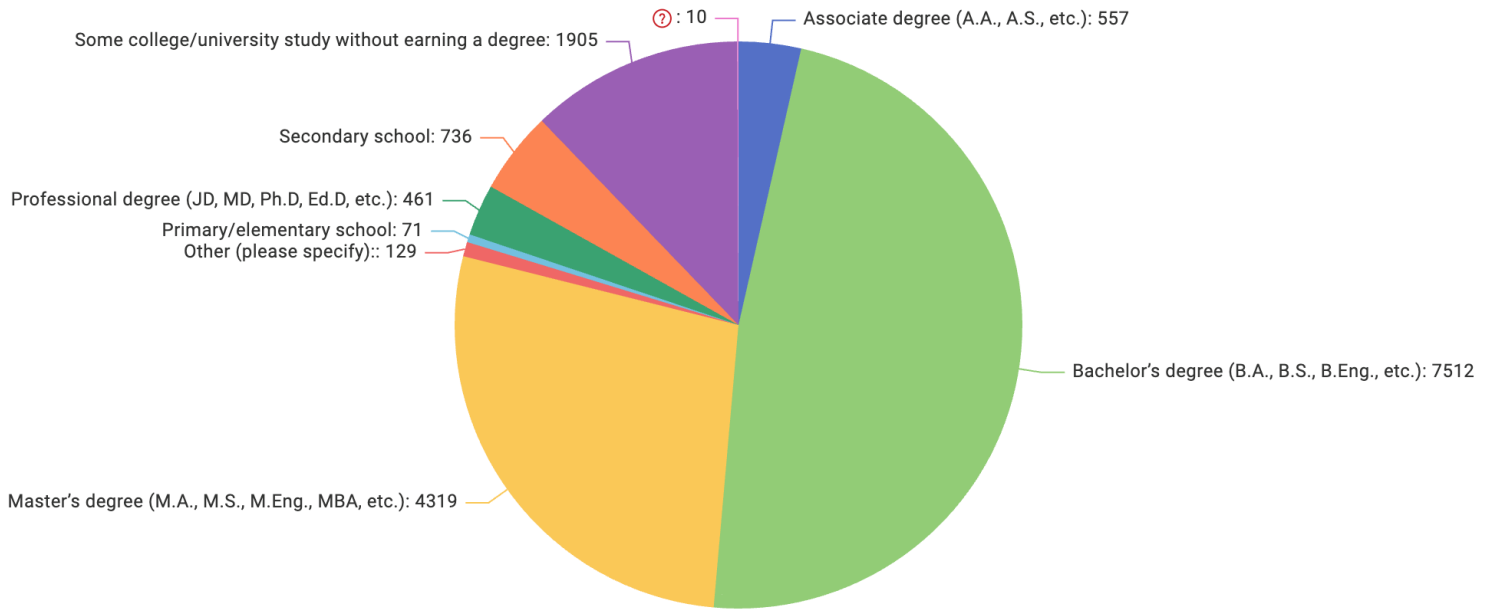

**Nature:** Categorical, ordinal, single-select

**Description:** Age group of survey respondents

**Categories:**
- 18–24 years old
- 25–34 years old
- 35–44 years old
- 45–54 years old
- 55–64 years old
- 65 or older
- Prefer not to say

**Insights:** The sample is concentrated in the 25–34 and 35–44 age brackets, which together represent the majority of respondents. Younger (18–24) and older participants (55+) are noticeably less represented.

# EdLevel




**Nature:** Categorical, nominal, single-select

**Description:** Highest level of formal education completed by respondents

**Categories:** Multiple education levels ranging from primary to postgraduate degrees, missing

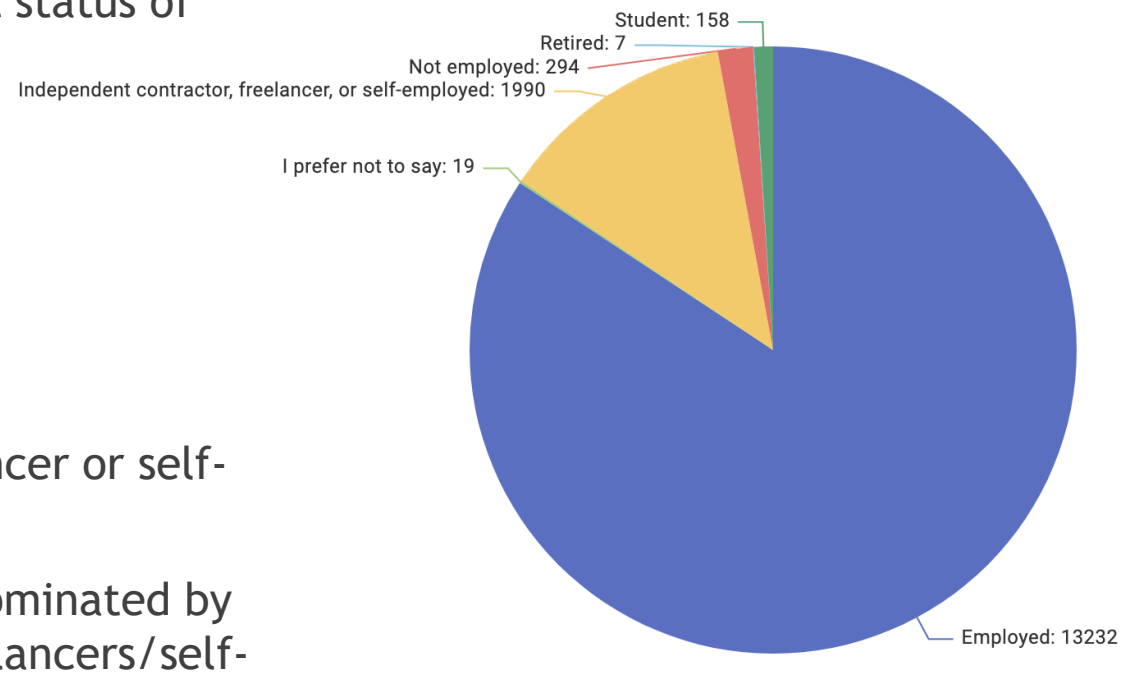
**Insights:** Most respondents report holding a higher education qualification, with bachelor's and master's degrees being the most frequent. Lower education levels and specialized professional degrees represent a considerably smaller portion of the sample.

# Employment

**Nature:** Categorical, nominal, single-select

**Description:** Type of employment status of respondents

**Categories:**
- I prefer not to say
- Employed
- Student
- Retired
- Not Employed
- Independent contractor, freelancer or self-employed




**Insights:** The sample is heavily dominated by employed respondents, with freelancers/self-employed forming a distant second. Students and the unemployed make up small portions, while retired participants and those who prefer not to say are nearly absent. Overall, employment status is highly concentrated in full-time employment.

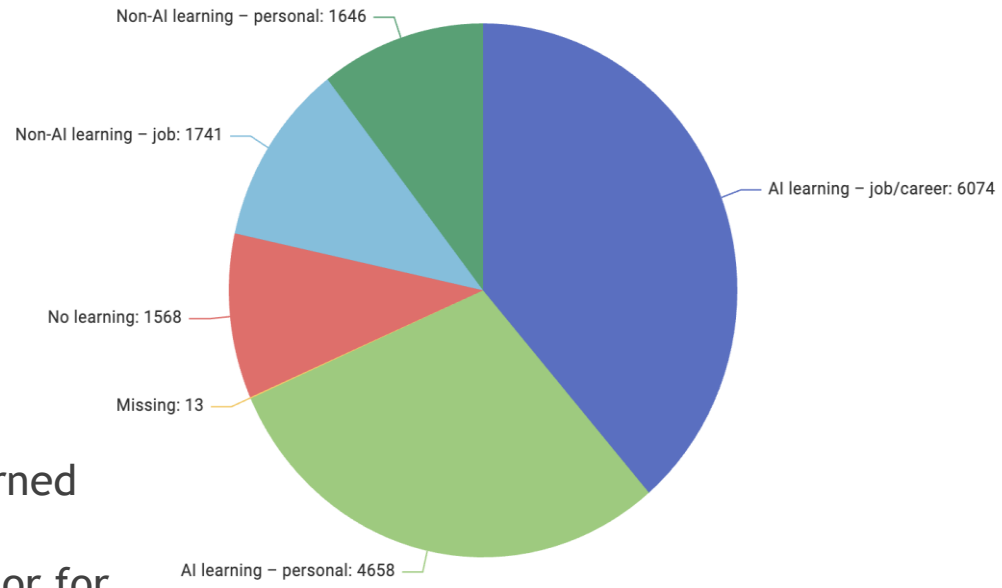
# LearnCodeAI

**Nature:** Categorical, nominal, single-select

**Description:** Indicates whether respondents spent time learning AI programming or AI-enabled tools in the past year, and for what purpose (job-related or personal).

**Categories:**

- AI learning – job/career
- AI learning – personal
- Non-AI learning – job
- Non-AI learning – personal
- No learning
- Missing




**Insights:**

AI-related learning is highly prevalent in the sample: most respondents report having learned AI-enabled tools, either to support their professional development (6,074 responses) or for personal curiosity (4,658). A smaller share focused on skills unrelated to AI, while 1,568 participants did not engage in any learning over the past year. Overall, the data highlights a strong trend toward AI upskilling among developers.

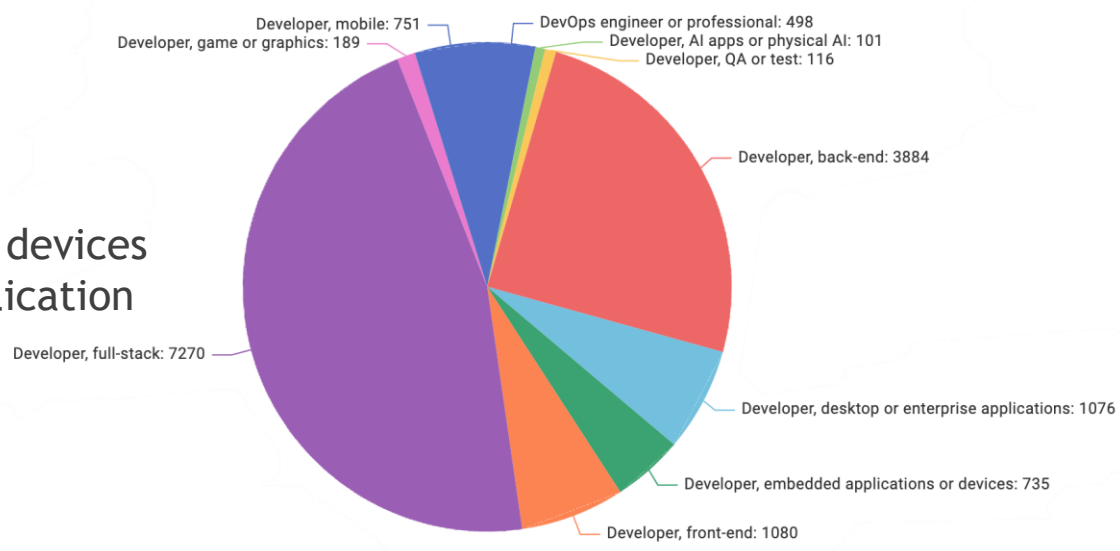
# DevType

**Nature:** Categorical, nominal, single-select

**Description:** Which job the respondents have or have had mostly in the current past

**Categories:**

- Developer, mobile
- Developer, game or graphics
- Developer, full-stack
- Developer, front-end
- Developer, embedded applications or devices
- Developer, desktop or enterprise application
- Developer, back-end
- Developer, QA or test
- Developer, AI apps or physical AI
- DevOps, engineer or professional




**Insights:** The sample is heavily dominated by **full-stack developers**, with **back-end**, **desktop/enterprise**, **front-end**, and **embedded** roles forming smaller but notable groups. More specialized areas, like **mobile**, **DevOps**, **QA**, **AI**, and **game/graphics,** are only lightly represented.

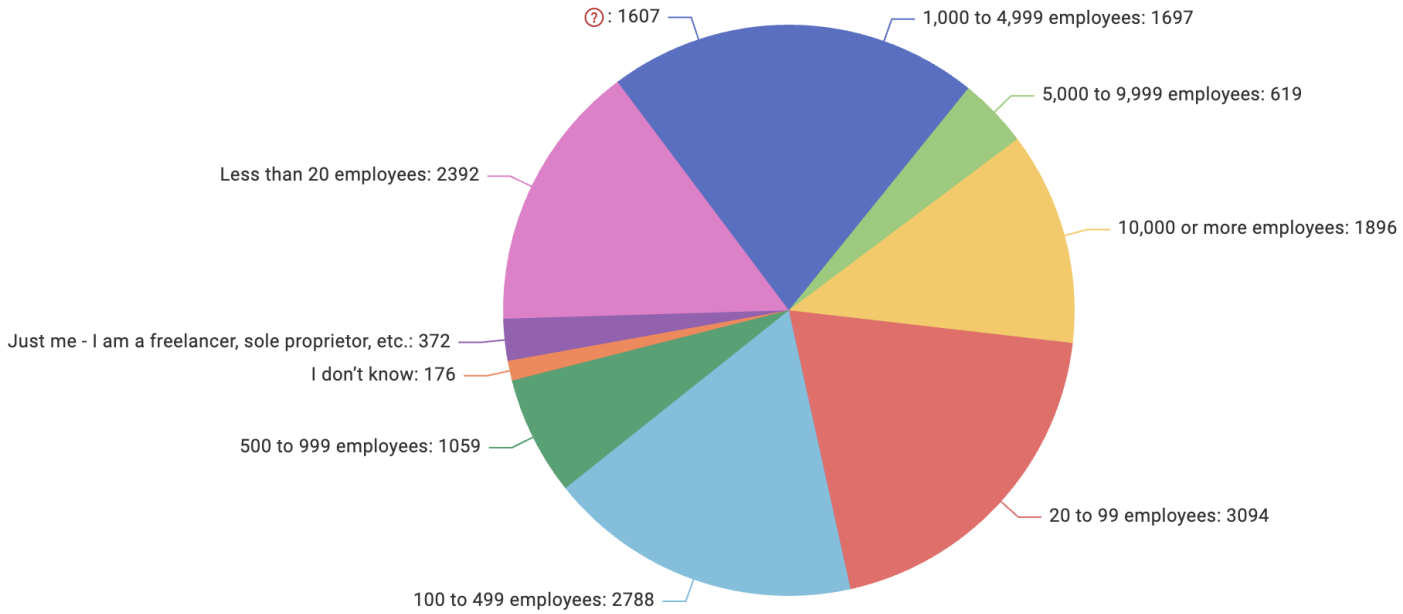
# OrgSize

**Nature:** Categorical, nominal, single-select

**Description:** Indicates the approximate number of employees at the respondent's primary organization.

**Categories:** Multiple company-size brackets ranging from solo workers to enterprises with over 10,000 employees, missing

**Insights:** The sample displays substantial heterogeneity in organizational size. Mid-sized companies (20–99 and 100–499 employees) represent the largest groups, with 3,094 and 2,788 respondents respectively. Smaller organizations, including freelancers and very small teams, are less represented, while large enterprises (1,000+ employees) also account for a notable portion of the sample. Overall, the distribution suggests that developers work across a wide variety of organizational scales.



# Remote Work

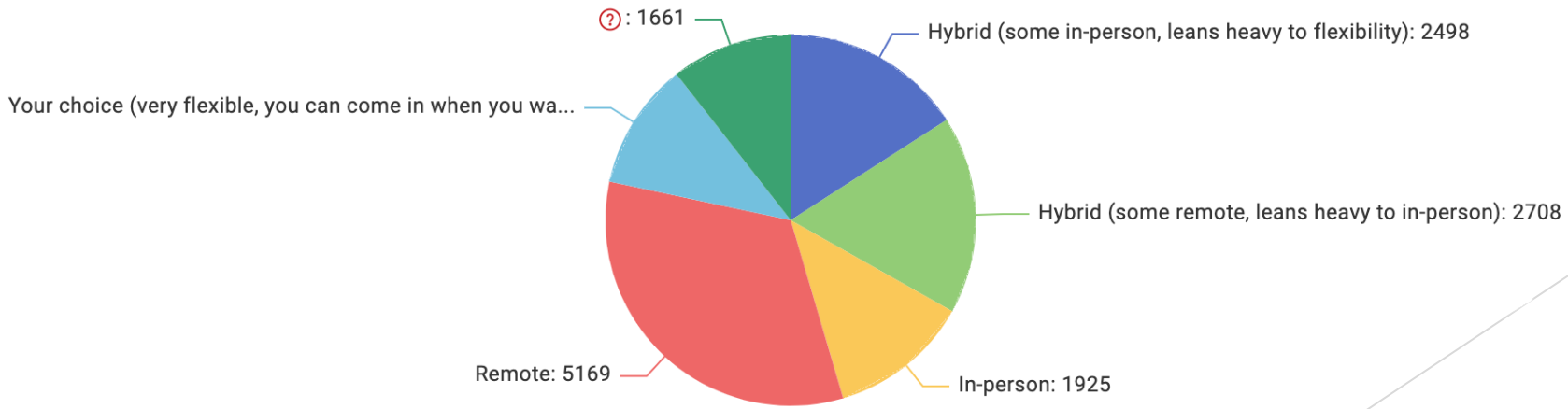
**Nature:** Categorical, nominal, single-select

**Description:** Current work situation

**Categories:**

- Hybrid (some in-person, leans heavy to flexibility)
- Hybrid (some remote, leans heavy to in-person)
- In-person
- Remote
- Your choice (very flexible, you can come in when you want)
- Missing

**Insights:** Remote work is the most common arrangement, followed by hybrid setups, with a slight preference for hybrid models leaning toward in-person. Fully in-person roles form a smaller share, while very flexible “your choice” arrangements and unspecified responses make up the remainder. Overall, the sample leans strongly toward remote or flexible work.



# Country

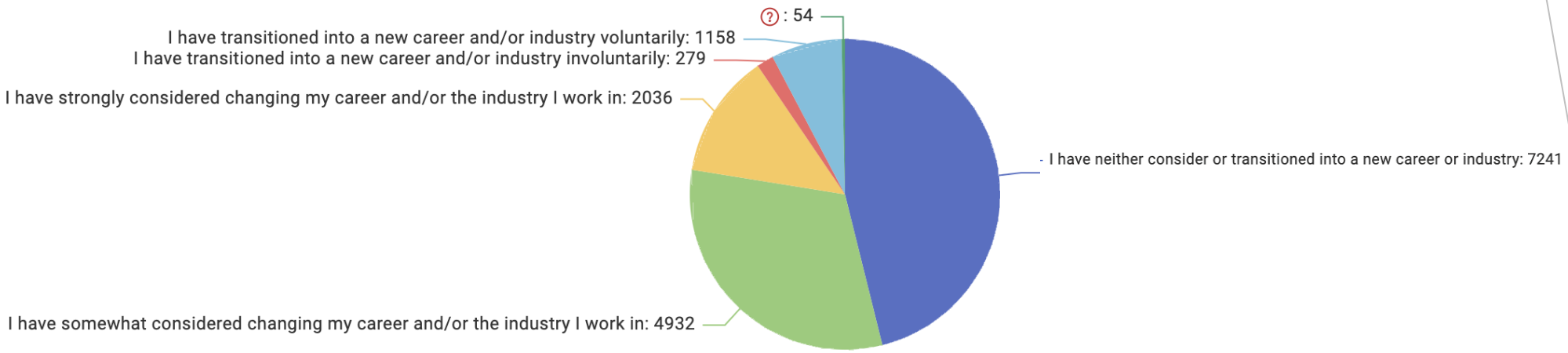
**Nature:** Categorical, nominal, single-select

**Description:** Country in which the developer works

**Categories (in the raw dataset):**
153 different countries from the world, from all continents

**Insights:**
The **Country** variable reflects the global reach of the Stack Overflow Developer Survey, capturing respondents from more than 150 nations. The distribution is highly uneven: while countries such as the United States, Germany, India, the United Kingdom, and Canada contribute the largest shares, many others appear with only a handful of entries. For this project, we retain all countries, as this diversity provides valuable insight into geographic patterns in developer behavior; however, we remain aware that countries with very small counts offer limited statistical representativeness and may introduce noise in downstream modeling.

# New Role



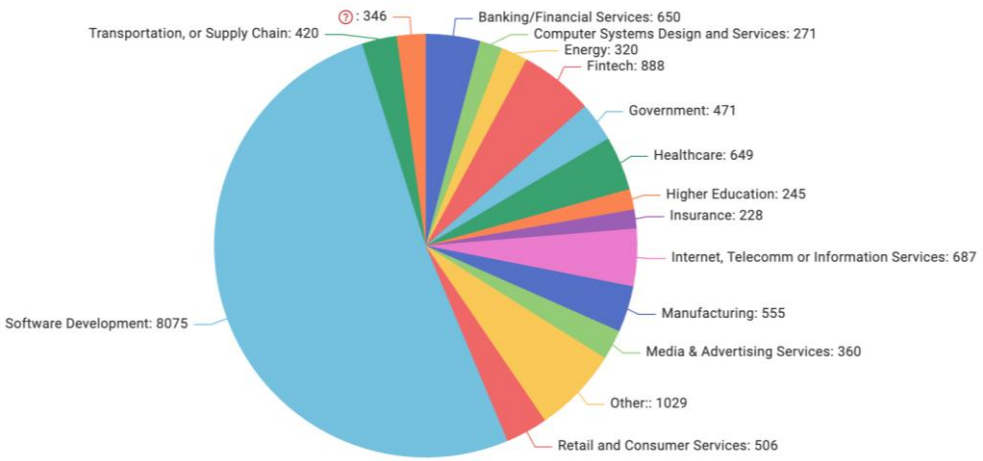

**Nature:** Categorical, nominal, single-select

**Description:** whether the respondent has considered a career change or transitioned into a new role

**Categories:**
- I have neither considered or transitioned into a new career or industry
- I have somewhat considered changing my career and/or the industry I work in
- I have strongly considered changing my career and/or the industry I work in
- I have transitioned into a new career and/or industry involuntarily
- I have transitioned into a new career and/or industry voluntarily
- Missing

**Insights**: Most respondents have neither considered nor transitioned into a new career, though a large group has somewhat considered making a change. Smaller portions have strongly considered switching fields or have already transitioned, with voluntary transitions far more common than involuntary ones. Unspecified responses make up only a tiny share.

# Industry




**Nature:** Categorical, nominal, single-select

**Description:** Indicates the industry sector in which respondents are currently employed.

**Categories:**
- Software Development
- Financial Services
- Government
- Healthcare
- Higher Education
- Internet, Telecom or Information Services
- Manufacturing
- Media & Advertising
- Retail and Consumer Services
- Energy
- Transportation or Supply Chain
- Computer Systems Design and Services
- Insurance
- Fintech
- Other

**Insights:** The industry distribution of respondents is **highly unbalanced**, with *Software Development* dominating the sample (over 8,000 respondents). This confirms that the Stack Overflow survey is strongly skewed toward roles and environments directly connected to software engineering. Other relevant sectors include *Fintech*, *Banking/Financial Services*, *Internet/Telecom*, and *Healthcare*, each representing several hundred developers. Traditional industries such as *Manufacturing*, *Energy*, *Retail*, and *Transportation* appear much less represented.

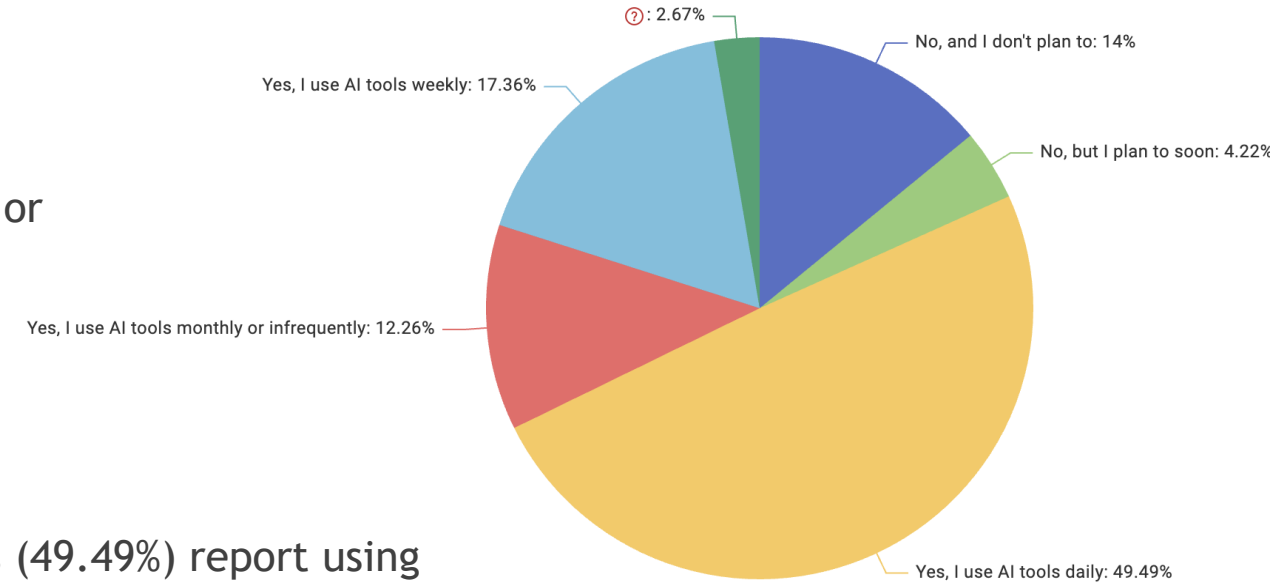
# AISelect

**Nature:** Categorical, nominal, single-select

**Description:** Indicates whether respondents currently use AI tools in their development workflow, and how frequently.

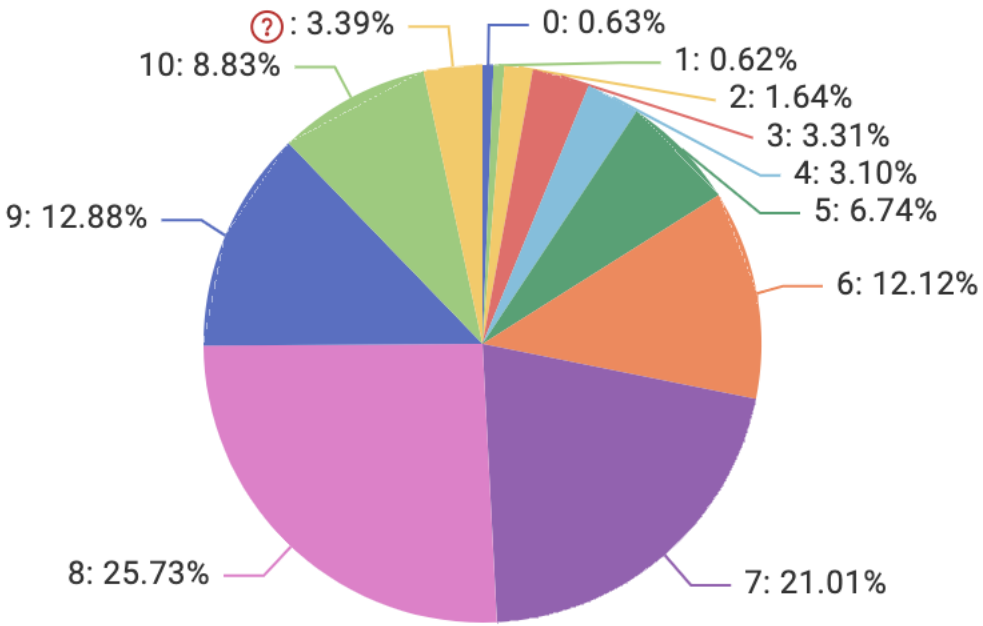
**Categories:**

- Yes, I use AI tools daily
- Yes, I use AI tools weekly
- Yes, I use AI tools monthly or infrequently
- No, but I plan to soon
- No, and I don’t plan to
- Missing




**Insights:**
Nearly half of the respondents (49.49%) report using AI tools on a daily basis, with additional groups using them weekly (17.36%) or occasionally (12.26%). Only a small share does not use AI tools: 4.22% plan to adopt them soon, while 14% do not intend to use them at all.

# JobSat




**Nature:** Categorical, ordinal, single-select

**Description:** how satisfied the respondent is in his current role

**Categories:** from 0 to 10, missing

**Insights:** Job satisfaction skews strongly positive, with the majority rating their satisfaction between 7 and 9, and 8 being the most common score. Mid-range ratings (5–6) form a smaller but notable portion, while low satisfaction scores (0–3) are rare. A modest share rates their satisfaction at the very top (10), and only a small fraction did not provide a clear rating.

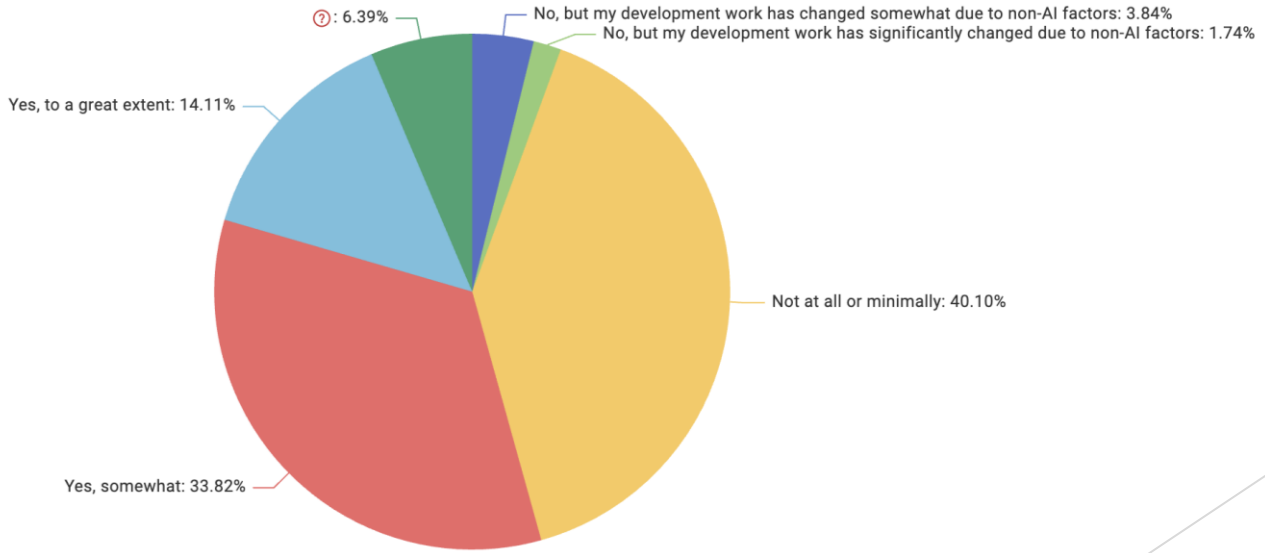
# AIAgentChange

**Nature:** Categorical, nominal, single-select

**Description:** Measures the extent to which respondents feel their development work has changed as a result of using AI agents or AI-enabled tools.

**Categories:** Multiple perceived levels of impact, ranging from no change to substantial transformation.

**Insights:** Most respondents report little to moderate change in their development work due to AI tools. While 40.10% indicate no or minimal impact and 33.82% report some change, only 14.11% perceive a substantial transformation. A small minority attributes changes to non-AI factors. Overall, AI's impact appears present but still developing.



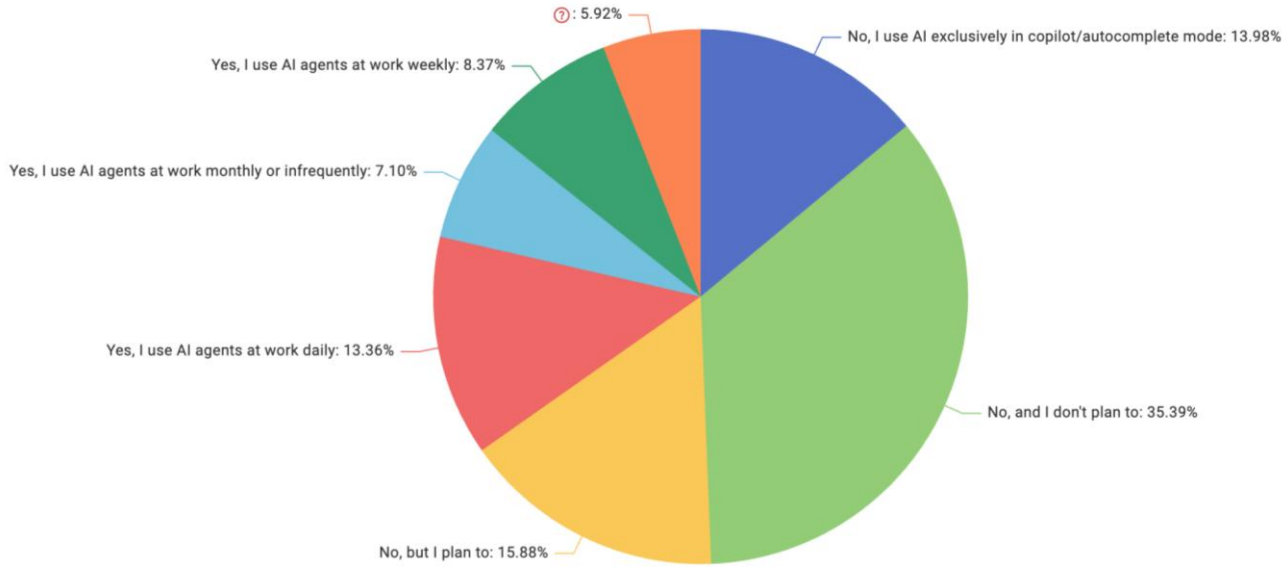
# AIAgents

**Nature:** Categorical, nominal, single-select

**Description:** Indicates whether respondents use AI agents in their work and at what frequency.

**Categories:** Multiple usage levels, from daily use to no adoption.

**Insights:** The largest group (35.39%) reports **no AI usage and no plans to adopt**, while 15.88% plan to adopt soon. Around 13–14% use AI either daily or exclusively in autocomplete mode, and another 15% use it weekly or monthly. Overall, AI-agent usage is growing but remains uneven, with a clear divide between early adopters and non-users.



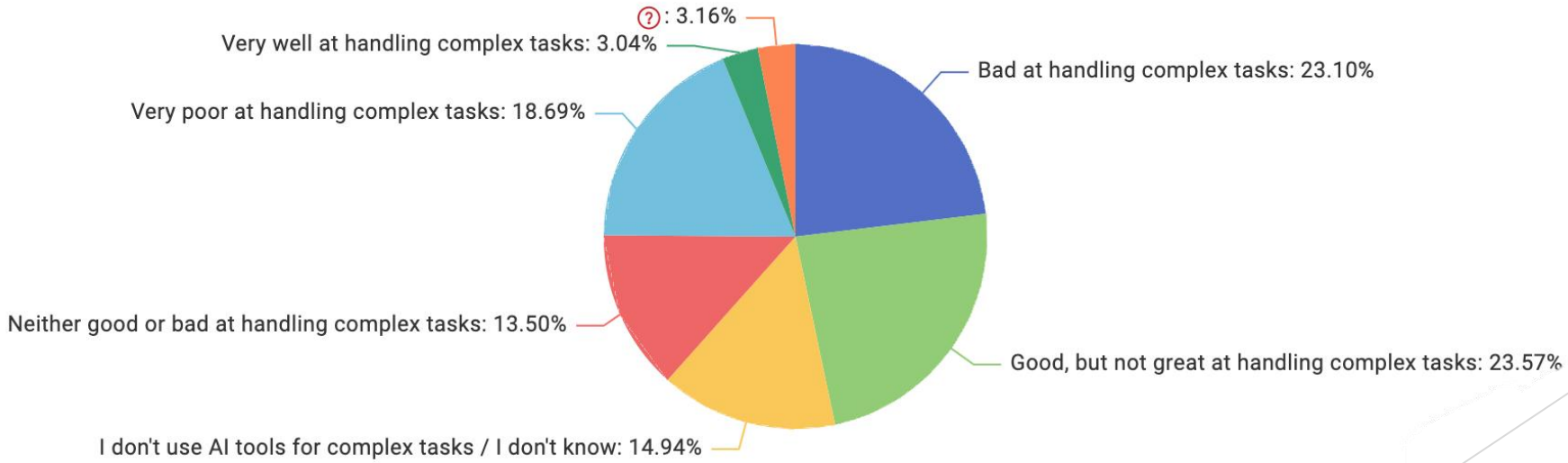
# AIComplex

**Nature:** Categorical, nominal, single-select

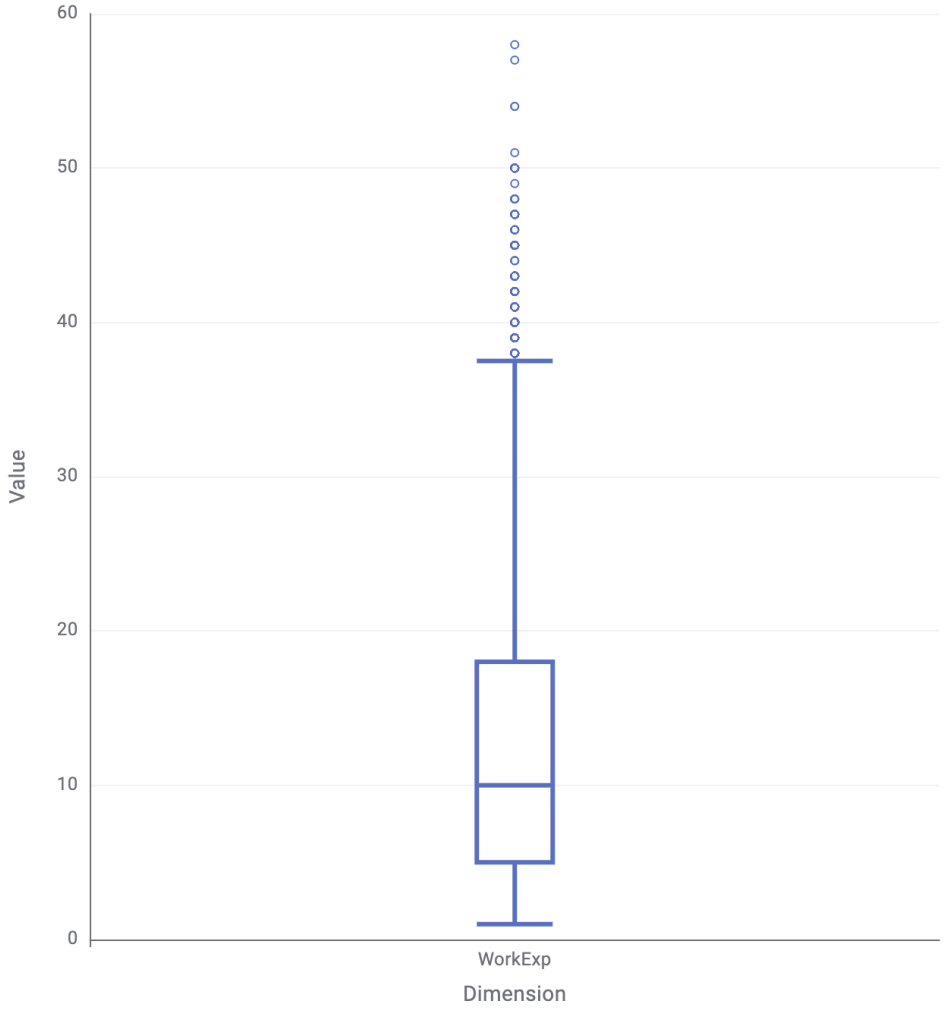
**Description:** Measures how well respondents feel that AI tools handle complex development tasks.

**Categories:** Multiple levels of skill, ranging from very well to very poor, including complete non usage of AI tools.

**Insights:** Opinions on AI tools for complex tasks are mixed. The largest groups rate them as good but not great or bad, with both views equally represented. A notable share feels they perform very poorly, while mid-level ratings are smaller. Only a small minority believes AI handles complex tasks very well, and some respondents don't use AI tools or are unsure.



# WorkExp



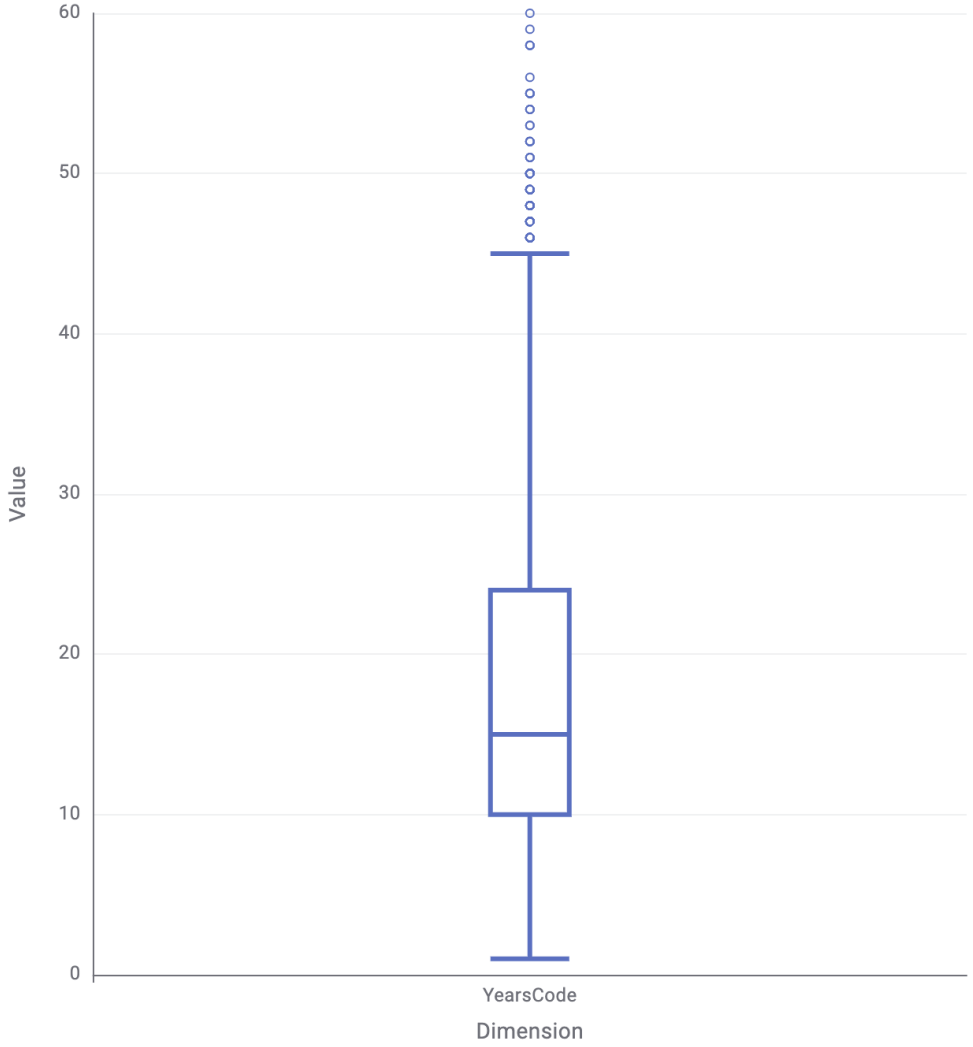

**Nature:** Numerical, single-select

**Description:** Work Experience of the respondent in years

**Range:**
- from 1 to 37.5

**Insights:** Work experience varies widely across respondents, but most fall within a moderate range, with a median around 10. There are many early-career participants, while a smaller group reports very high experience levels. Several outliers indicate respondents with exceptionally long careers. Overall, the distribution is skewed toward lower to mid-level experience.

# YearsCode




**Nature:** Numerical, single-select

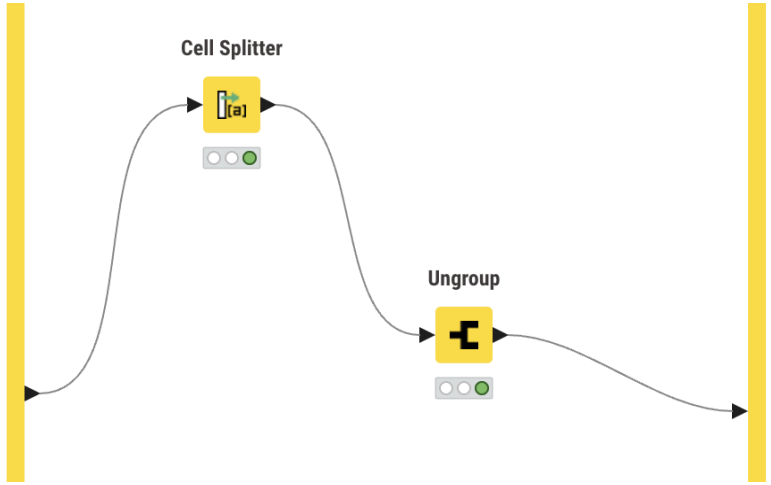
**Description:** How many years the respondent has been coding in total

**Range:**
- from 1 to 45

**Insights:** Coding experience spans a broad range, but most respondents cluster around early to mid-career levels. The median sits near the lower-middle portion of the scale, at 15 years of experience, with many reporting under 20 years of experience. A smaller group shows long-term expertise, and several high-value outliers indicate respondents with exceptionally extensive coding histories. Overall, the distribution skews toward lower to moderate experience.

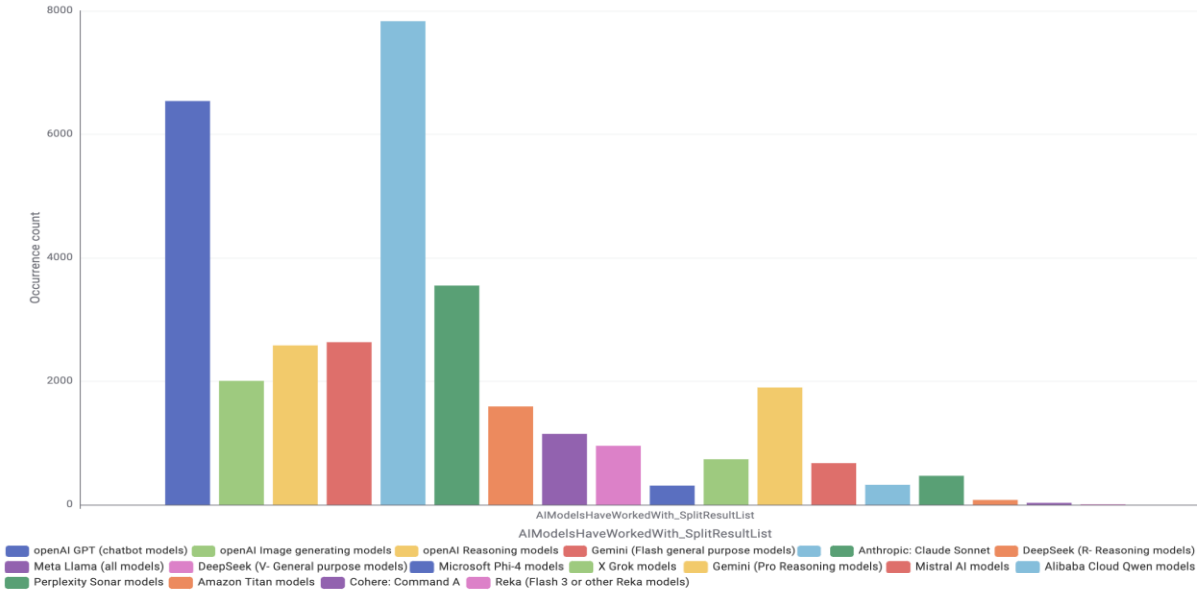
# Multi-select Variables

Many survey questions in the dataset allow respondents to choose multiple options, resulting in text fields containing several values separated by delimiters. Since these fields cannot be directly interpreted or analyzed as single categorical variables, a dedicated preprocessing step was required.




We used the *Multi-select* and *Ungroup* nodes to split each multi-response field into individual entries, transforming each selected item into a separate row. This process ensured that all options appear independently in the data, enabling accurate frequency counts and clear univariate exploration of technologies, tools, and platforms used by respondents.
The following slides present the distribution of each multi-select variable after this expansion.

# AIModels

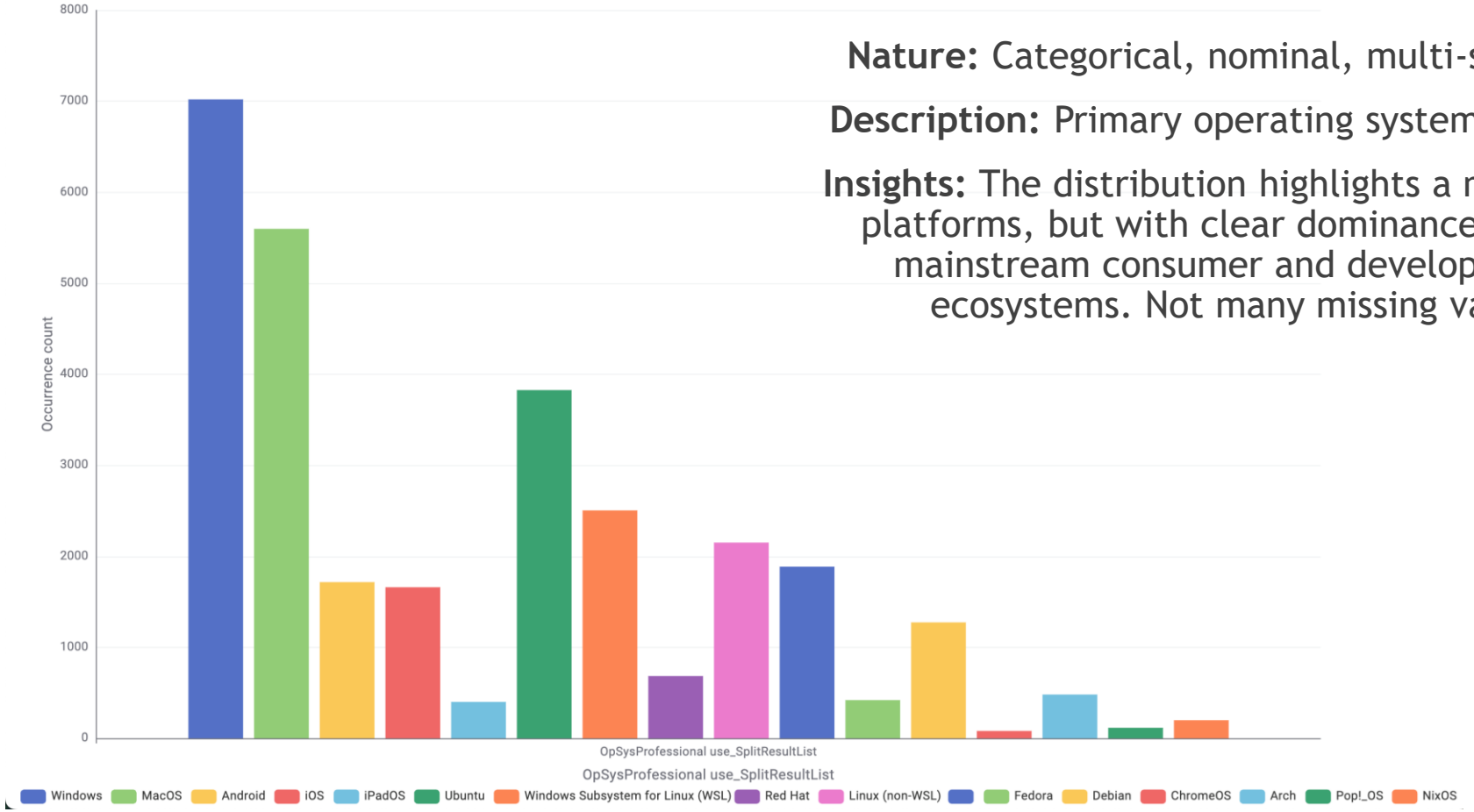



**Nature:** Categorical, nominal, multi-select

**Description:** Type of AI Model most used

**Insights:** AI model usage is highly concentrated around a few families. The empty category shows a very large number of missing values (7,832), indicating that many respondents did not specify any model. Among those who did, openAI GPT (6,539), Claude Sonnet (3,550) and Google's Gemini Flash (2,633) and Gemini Pro (1,899) are the most commonly used. Mid-level adoption appears for DeepSeek, Meta Llama, and openAI image/reasoning models, while providers like Reka, Cohere, and Amazon Titan show only minimal usage. Overall, a few dominant ecosystems capture most of the declared model usage.

# OpSystem

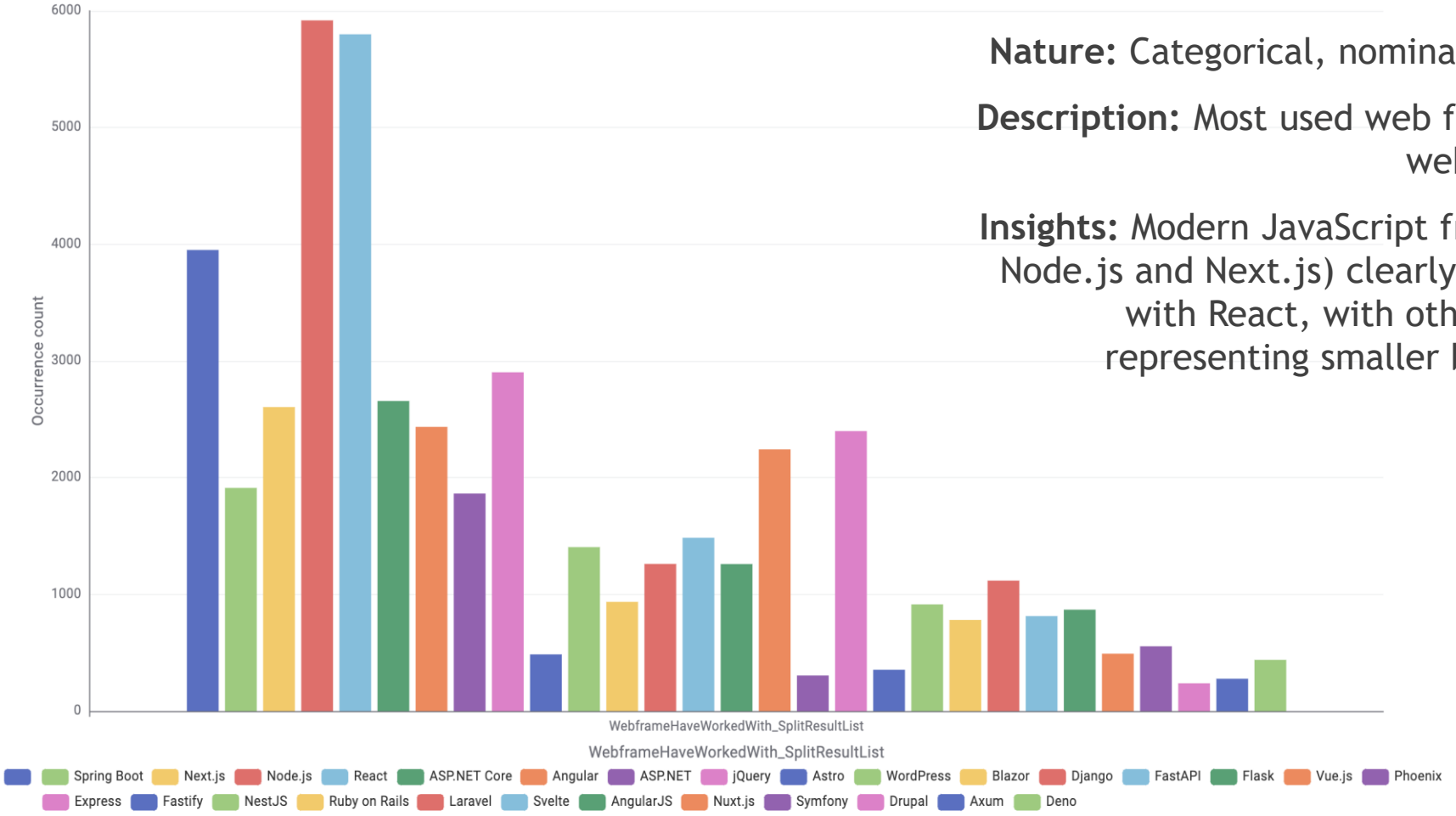



**Nature:** Categorical, nominal, multi-select

**Description:** Primary operating system used

**Insights:** The distribution highlights a mix of platforms, but with clear dominance from mainstream consumer and developer OS ecosystems. Not many missing values.

# WebFrame

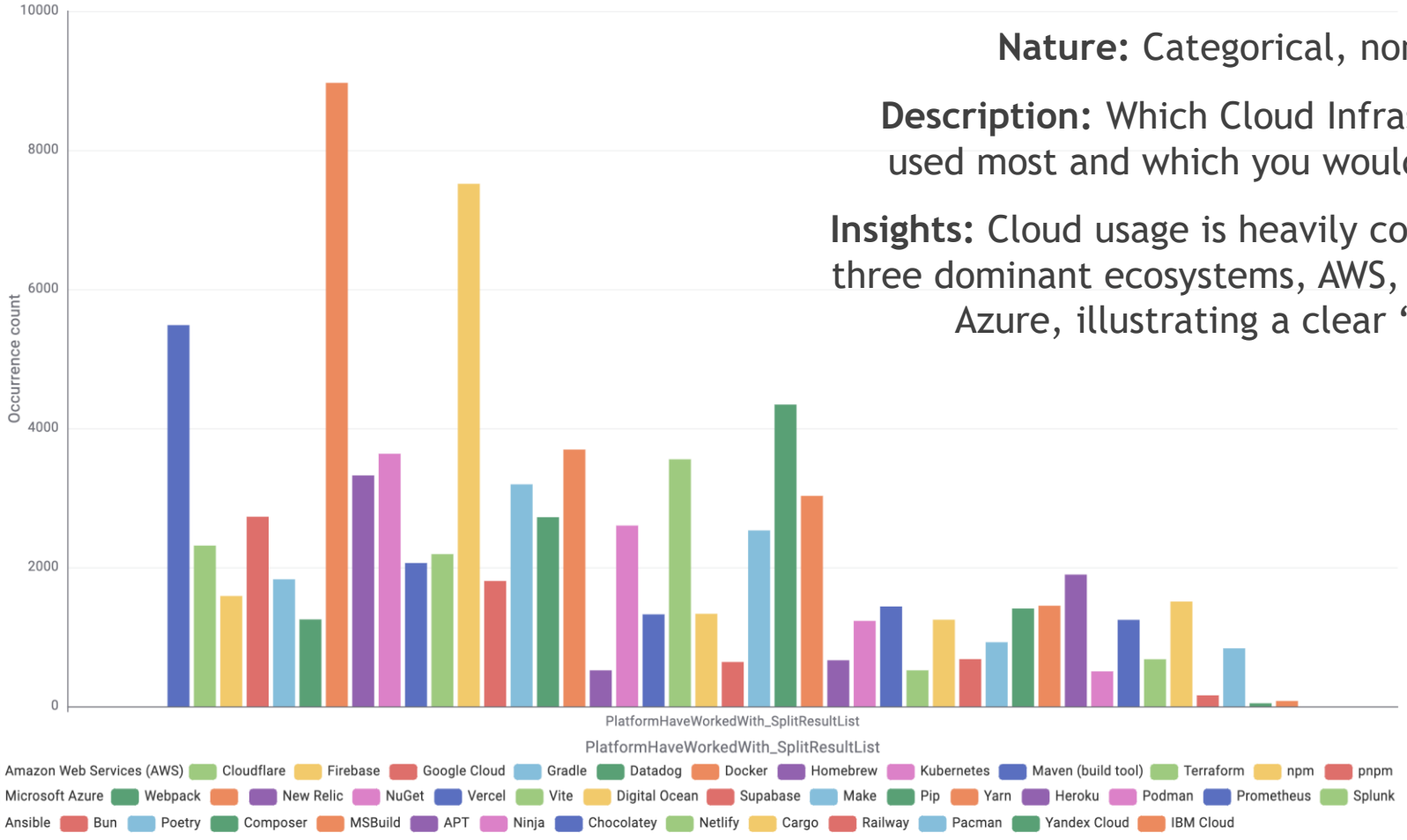



**Nature:** Categorical, nominal, multi-select

**Description:** Most used web framework and web technologies

**Insights:** Modern JavaScript frameworks (as Node.js and Next.js) clearly lead together with React, with other ecosystems representing smaller but still active communities.

# Platform

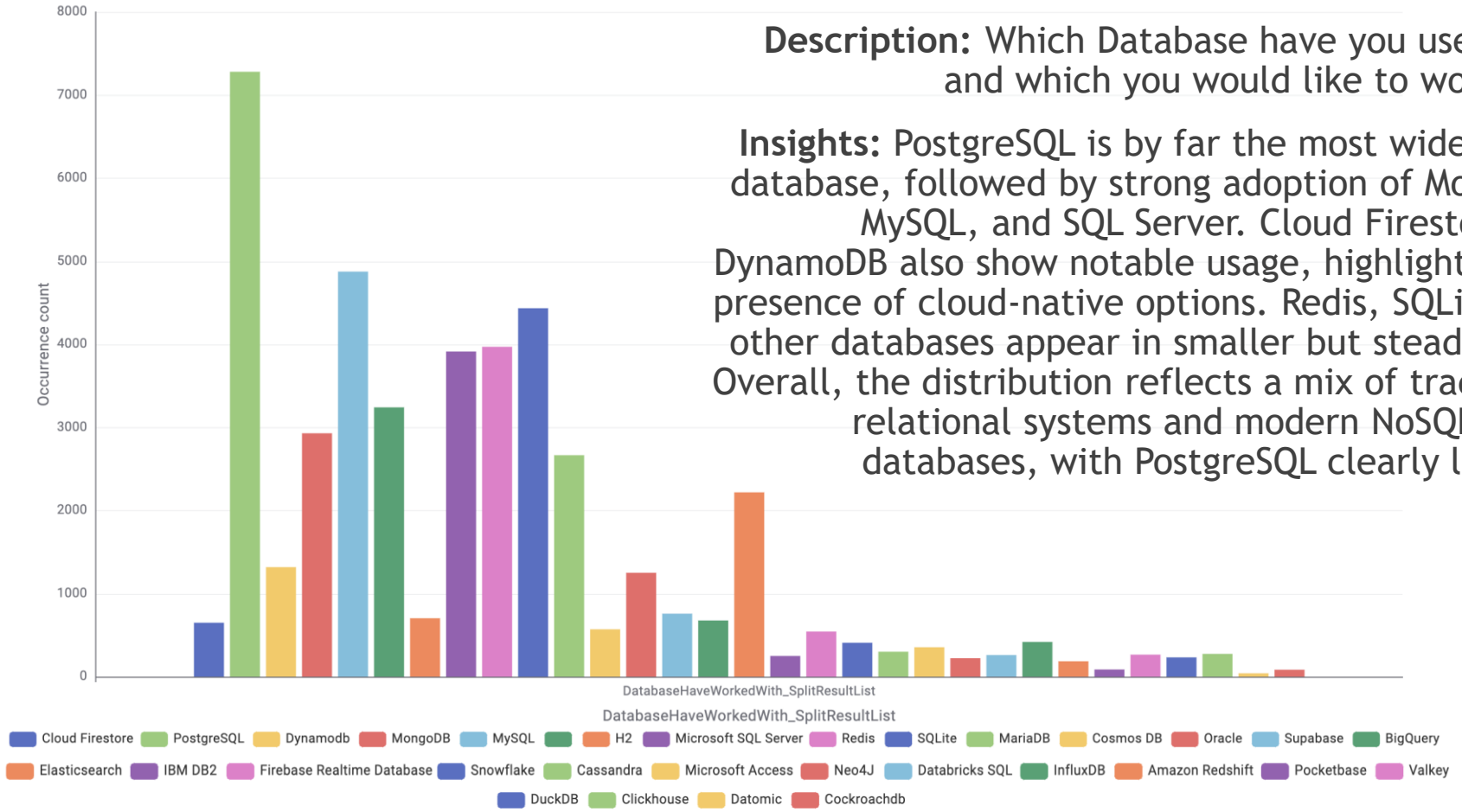



**Nature:** Categorical, nominal, multi-select

**Description:** Which Cloud Infrastructure have you used most and which you would like to work with

**Insights:** Cloud usage is heavily concentrated around three dominant ecosystems, AWS, Google Cloud, and Azure, illustrating a clear “big three” market structure.

# Database

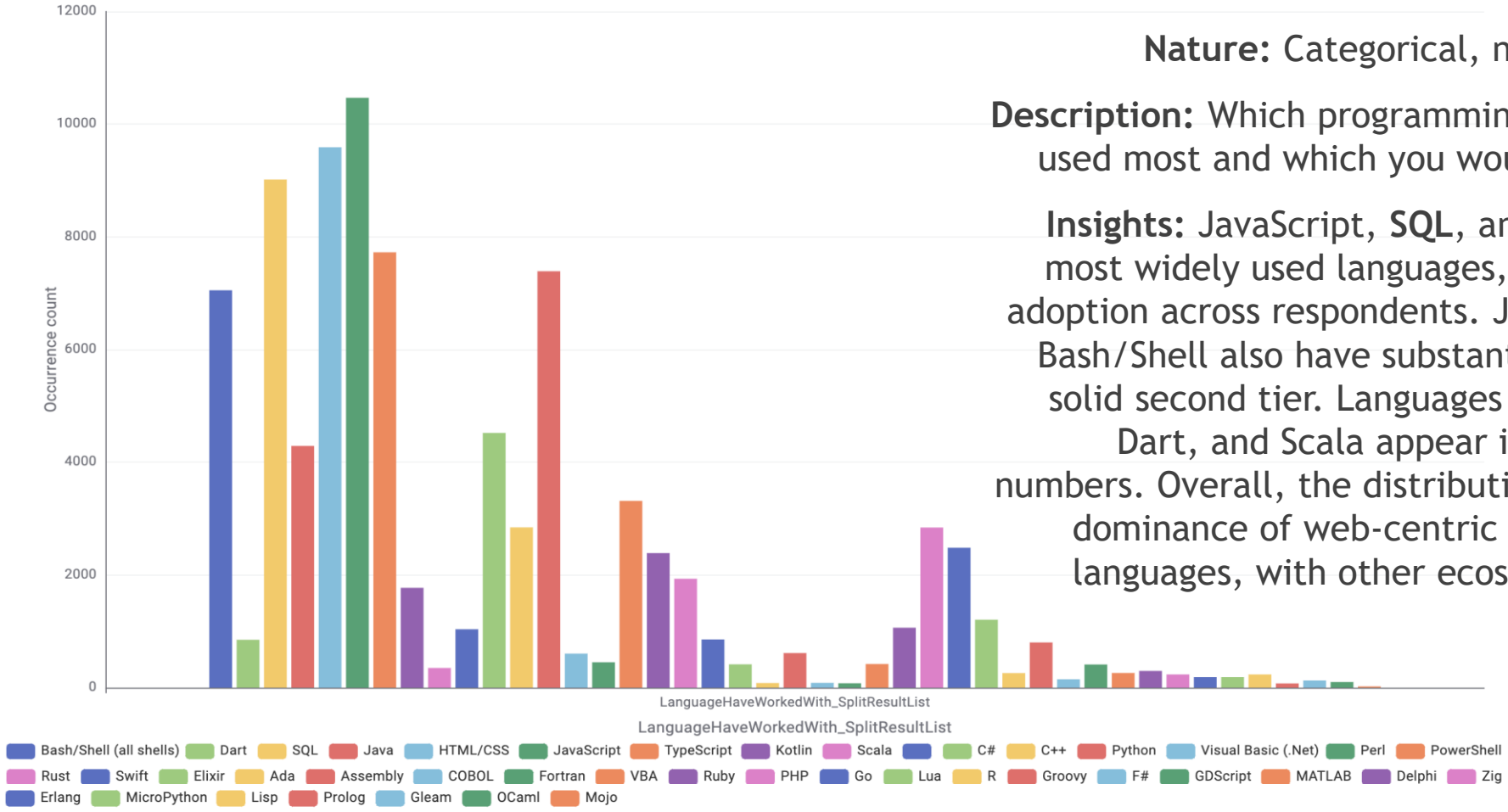



**Nature:** Categorical, nominal, multi-select

**Description:** Which Database have you used most and which you would like to work with

**Insights:** PostgreSQL is by far the most widely used database, followed by strong adoption of MongoDB, MySQL, and SQL Server. Cloud Firestore and DynamoDB also show notable usage, highlighting the presence of cloud-native options. Redis, SQLite, and other databases appear in smaller but steady roles. Overall, the distribution reflects a mix of traditional relational systems and modern NoSQL/cloud databases, with PostgreSQL clearly leading.

# Language




**Nature:** Categorical, nominal, multi-select

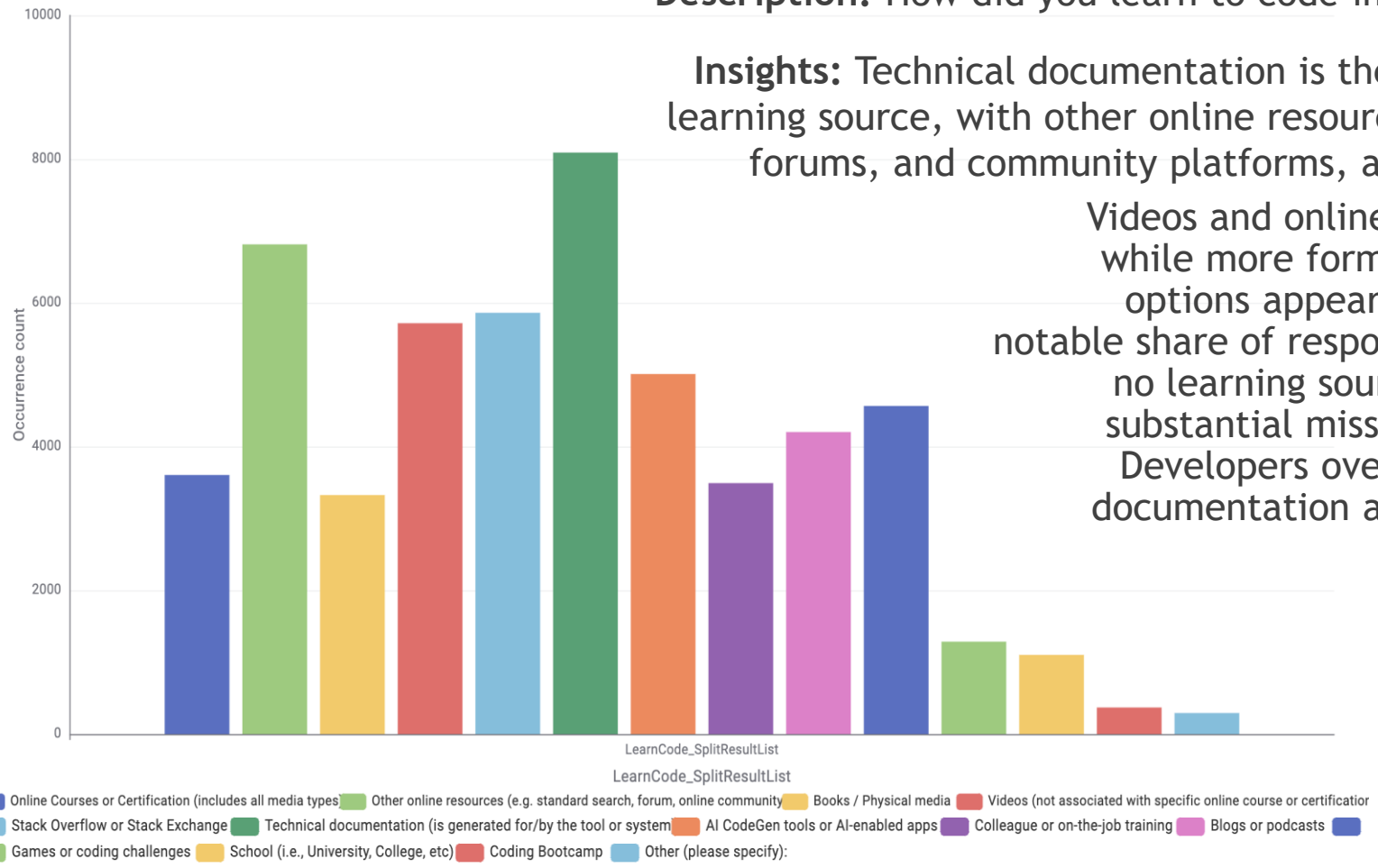
**Description:** Which programming language have you used most and which you would like to work with

**Insights:** JavaScript, **SQL**, and HTML/CSS are the most widely used languages, showing very strong adoption across respondents. Java, TypeScript, and Bash/Shell also have substantial usage, forming a solid second tier. Languages like Kotlin, C#, C++, Dart, and Scala appear in smaller but steady numbers. Overall, the distribution highlights a clear dominance of web-centric and general-purpose languages, with other ecosystems playing more specialized roles.

# LearnCode

**Nature:** Categorical, nominal, multi-select

**Description:** How did you learn to code in the past years




**Insights:** Technical documentation is the most relied-on learning source, with other online resources, like search, forums, and community platforms, also widely used.

Videos and online courses follow, while more formal or structured options appear less common. A notable share of respondents provided no learning source, highlighting substantial missing information. Developers overall lean toward documentation and self-directed online learning.

# 3. Data preparation

## BASIC PREPROCESSING

# Overview of data preparation

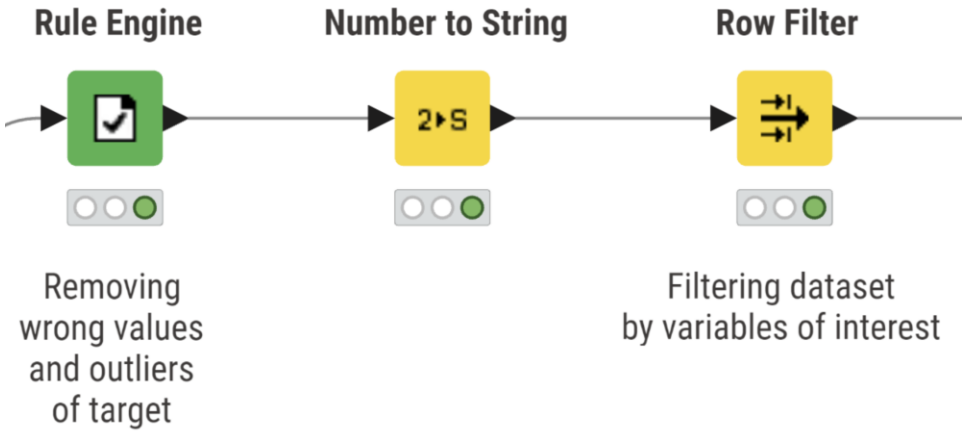
**Basic Preprocessing**

The data preparation phase ensures that the dataset is clean, coherent, and ready for modeling. Given the heterogeneous structure of the Stack Overflow survey, comprising numerical fields, **single-select categorical variables**, and **multi-select categorical variable**, **different preprocessing strategies are required**. This stage focuses on **standardizing numerical formats**, **handling missing values**, and **restructuring multi-response fields** while retaining only the **variables relevant** to the research question. Since tree-based models naturally handle categorical attributes without one-hot encoding, preprocessing remains minimal and targeted, preserving information while ensuring consistency across features.

**Additional Preprocessing**

Additional preparation is required for regression models and ANN. Multi-level categorical variables are one-hot encoded, and reference categories are removed to prevent multicollinearity. Numerical predictors are normalized to ensure comparable scaling when fitting linear and polynomial terms.

# Filtering observations



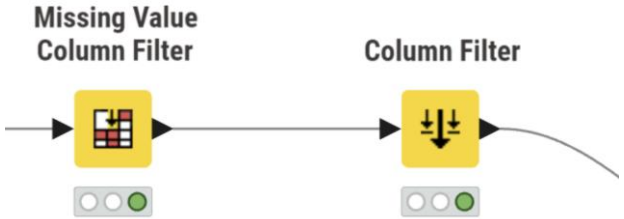

1. Refined the target variable **ConvertedCompYearly** based on insights from the univariate analysis (see Univariate analysis of target var).

2. Used the **Number to String** node (and corresponding conversions) to ensure consistent handling of numerical fields before filtering with various Rule Engines.

3. Then we used Row Filter to keep only relevant respondents based on our research question:
   - ConvertedCompYearly (point 1)
   - **Employed** variable: *Retired*, *Student*, *Not employed*, *Prefer not to say*
   - **Age** variable: *Prefer not to say*

# Filtering missing and irrelevant variables

1. Removed all variables with **20% or more missing values** with **Missing Value Column Filter**, reducing the dimensionality from **170 to 86 columns** and ensuring that the retained features had sufficient data coverage to support reliable analysis.



Filtering variables of interest

2. Applied a **Column Filter** to keep only variables that are **analytically meaningful, suitable for modelling** and **coherent with our research question**. This selection was guided by completeness, exploratory analysis, and the expected contribution of each feature to the model's $R^2$ and overall interpretability. Variables showing limited explanatory potential, as well as secondary survey items and endorsement statements, were excluded to avoid noise and redundancy.

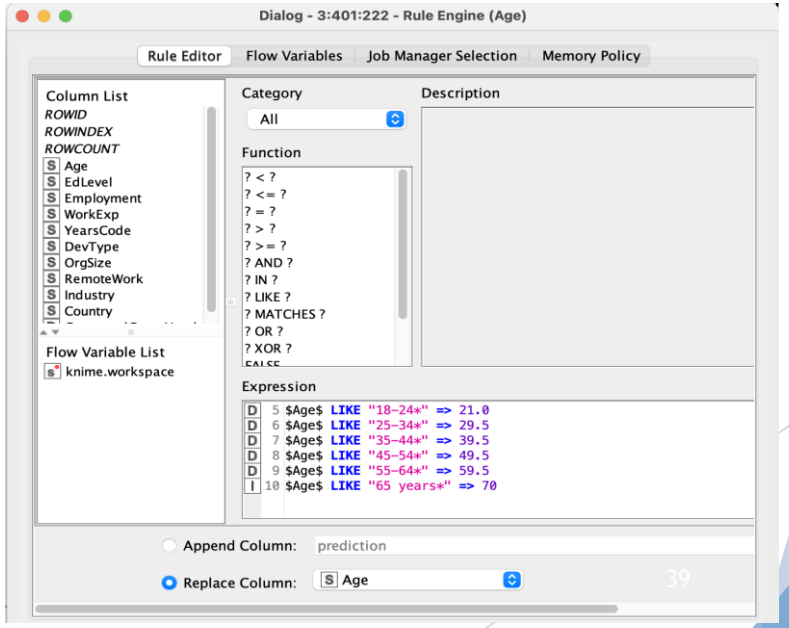
The resulting subset includes core demographic attributes, experience-related measures, AI adoption and behavioral indicators.

By removing low-quality, incomplete, or irrelevant variables, this combined filtering process reduces dimensionality, enhances dataset stability, and provides a cleaner, more consistent foundation for the subsequent preprocessing and modelling phases.

# Feature engineering - Age

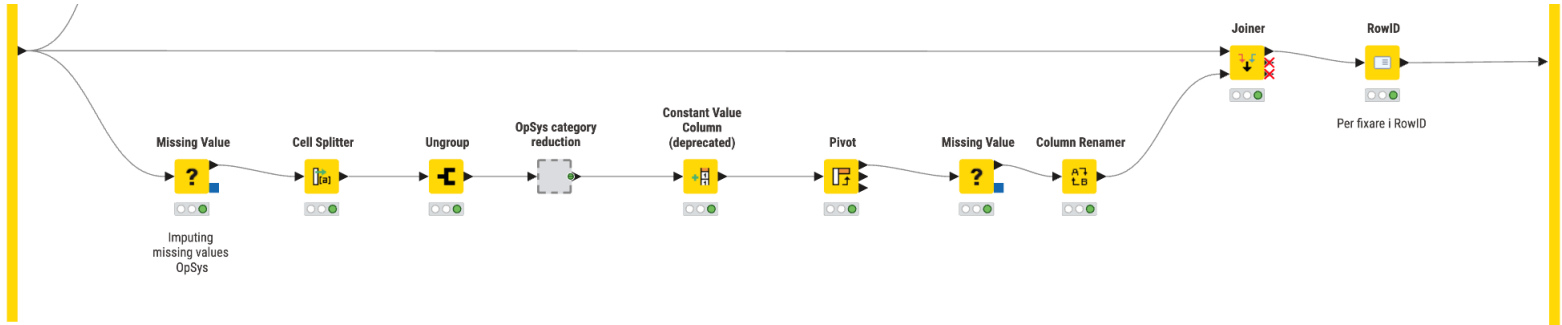
- We transformed the categorical variable **Age**, originally expressed as text ranges (e.g., *18-24 years*, *25-34 years*), into a **numerical variable** by assigning a representative numeric value to each age group.
- The following mapping scheme was adopted:
  - **18-24 years → 21.0**
  - **25-34 years → 29.5**
  - **35-44 years → 39.5**
  - **45-54 years → 49.5**
  - **55-64 years → 59.5**
  - **65+ years → 70.0**

- This encoding provides a **meaningful numeric representation** of the age groups, allowing the variable to be seamlessly integrated into downstream modelling tasks. Moreover, it maintains the ordinal structure of the original categories while avoiding an unnecessary expansion of dimensionality.



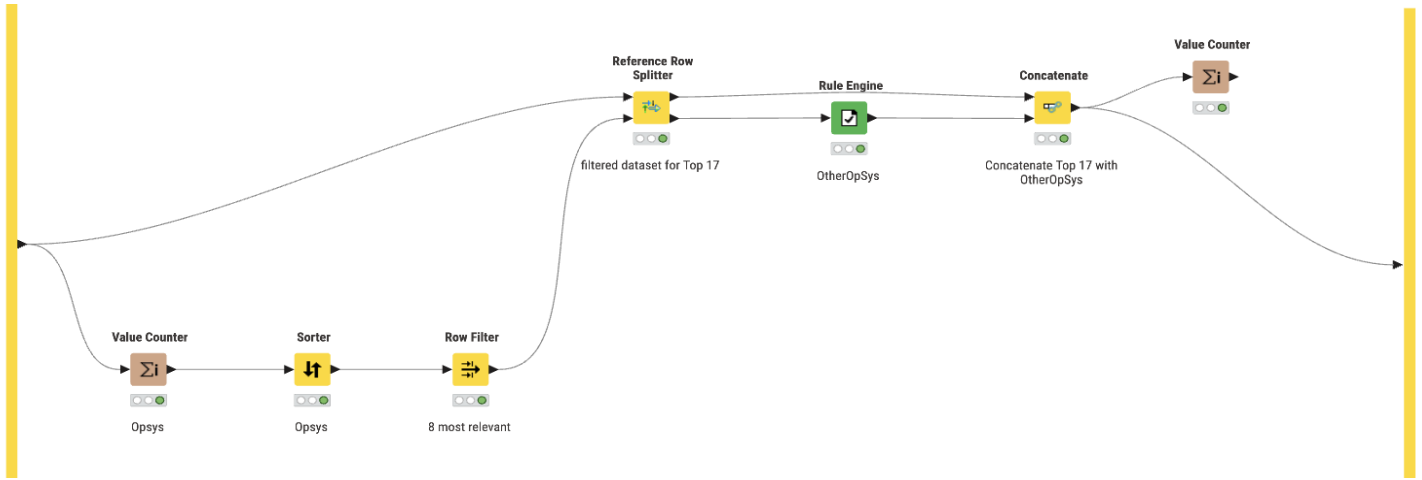
# Feature engineering – OpSys

The **OpSysProfessional use** variable contains multiple OS options selected by each respondent and stored as a semicolon-separated list. To transform it into a usable set of features, we first split each string into individual OS items using a **Cell Splitter** and expanded them into separate rows via **Ungroup**.
These entries were then processed through the **OpSys Category Reduction** metanode, which consolidates the operating systems into a manageable set of categories. After this step, a **Pivot** node converted the long-format table into a wide binary matrix, where each operating system corresponds to a dedicated column encoded as 1 (used) or 0 (not used).
Missing values were replaced appropriately, column names were standardized through a **Column Renamer**, and the engineered OS features were merged back into the main dataset using a **Joiner** based on *ResponseId*. A final **RowID** node ensured the dataset retained a clean and consistent row structure.
This process converts a complex multi-response text field into a structured, model-ready set of binary features that accurately represent each developer's professional OS environment.



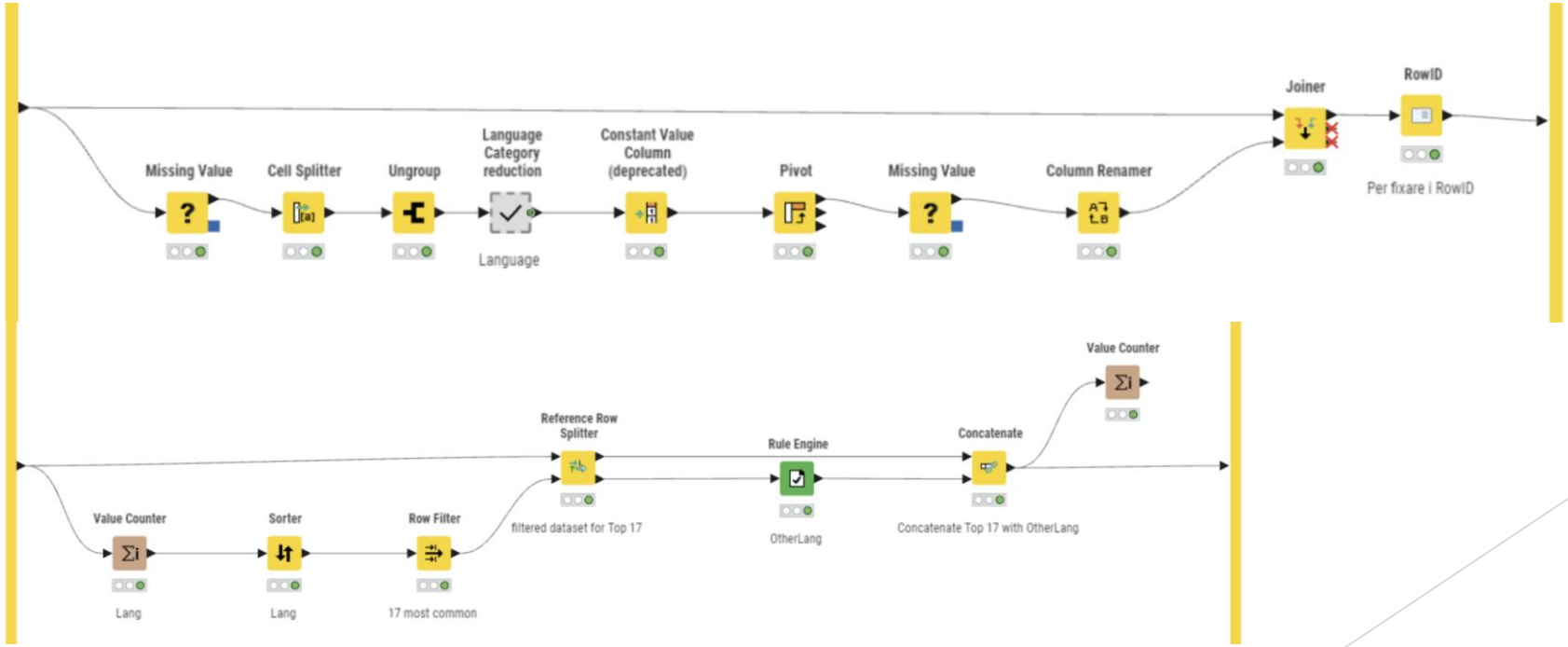
# Feature engineering – OpSys category reduction

- The **OpSysProfessional use** variable contains dozens of possible operating systems, many of which appear rarely. To avoid sparsity and reduce the dimensionality of the resulting pivot table, we first computed OS frequencies using a **Value Counter** and sorted them with a **Sorter**.
- The most frequent OS options were retained via a **Row Filter**, while all the remaining, less common OS values were isolated using a **Reference Row Splitter**. These rare categories were then consolidated into a single aggregated label (“**OtherOpSys**”) using a **Rule Engine**.
- Finally, the frequent categories and the aggregated “OtherOpSys” group were recombined using a **Concatenate** node, and a final **Value Counter** verified the resulting distribution.
- This reduction strategy preserves the most relevant OS categories while preventing an explosion of sparse dummy variables in downstream modeling.



# Feature engineering – Language

- The variable **LanguageHaveWorkedWith** contains a semicolon-separated list of all programming languages a respondent has used professionally, making it a multi-response field with very high cardinality. To turn this information into model-ready features, we followed the workflow we used for OpSys (See Feature Engineering – OpSys and OpSys category red.).
- This long-format representation was then passed into the **Language Category Reduction** metanode, which consolidates the potentially large number of languages into a manageable set of categories (8 plus the *OtherLang* category)



# Feature engineering – OrgSize

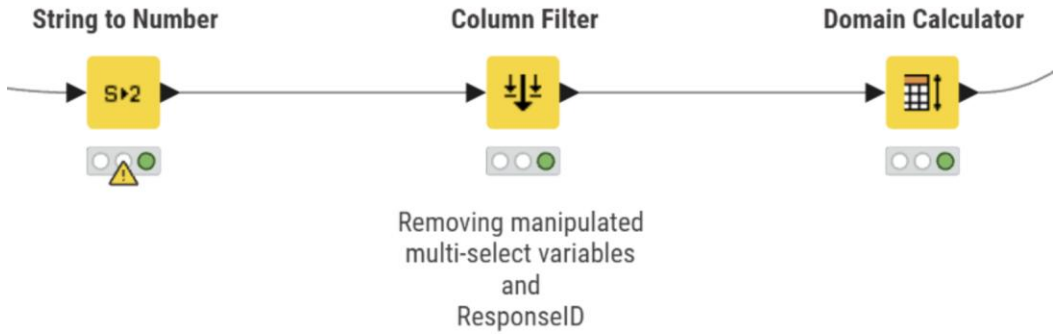
We transformed the **OrgSize** variable from an ordinal categorical feature into a **numerical variable**.

To allow our models to interpret the **magnitude** and linear relationship of workforce size, rather than treating these simply as distinct categories, we mapped each range to a **representative numerical value** (typically an estimated average or median).

We applied the following mapping logic:

- **Freelancers:** Mapped to **1**.
- **Small/Medium Ranges:** Converted to their approximate midpoint (e.g., *"20 to 99"* →**60**).
- **Large Enterprises:** Assigned conservative lower-bound estimates (e.g., *"10,000+* → **15,000**).
- Ambiguous entries such as *"I don't know"* were converted to **Missing Values (NA)** to ensure data quality and prevent the introduction of noise into the model. Finally, the column was converted to a Number (Integer) format.

# Final variable filtering




- **String to number:** numerical fields originally stored as strings (e.g., *WorkExp*, *YearsCode*, *ConvertedCompYearly*) are converted to the correct data type.
- **Column Filter:** temporary manipulated multi-select fields and *ResponseId* are removed, retaining only the variables relevant for modelling.
- **Domain calculator:** necessary to run the One To Many node in the following steps of the workflow

# Missing Values

To handle the remaining missing values, we used the **Missing Value** node, in multiple moments of the workflow which performs imputation according to predefined strategies for each data type. This ensures that all incomplete fields are consistently and systematically filled before moving to the modelling phase.

In our configuration, two imputation strategies were adopted:

- **Numerical variables (Float):** imputed using the **Median**, a robust measure of central tendency that is less sensitive to outliers.
- **String variables:** imputed using the **Most Frequent Value**, preserving the dominant category distribution within each feature.

This approach guarantees a complete dataset while maintaining the statistical properties of both numerical and categorical variables. By applying these tailored strategies, we ensure that the imputation process does not distort the underlying structure of the data and supports reliable downstream analysis

# 4. Data Description

## BIVARIATE ANALYSIS

# Overview

- We computed a **correlation matrix** on all modelling-ready variables related to AI usage (LearnCodeAI, AISelect, AIAgents, AIComplex), experience and work context (YearsCode, OrgSize, Employment, DevType, RemoteWork, Industry, NewRole, Country macro-areas), job satisfaction (JobSat) and the log-transformed target **ln(ConvertedCompYearly)**.
- OpSys and Programming Language dummies were excluded, as correlations involving individual operating systems or languages were not relevant to our research focus.
- In parallel, we produced **boxplots of ln(ConvertedCompYearly)** for the main categorical drivers (EdLevel, Employment, DevType, Industry, Country, RemoteWork, NewRole, AI-related variables) to visually compare compensation levels across groups.
- Together, the correlation matrix and boxplots provide a compact view of how compensation co-moves with developer characteristics and AI adoption, guiding feature selection for the subsequent models.

# Bivariate analysis of continuous variables

Correlation matrix

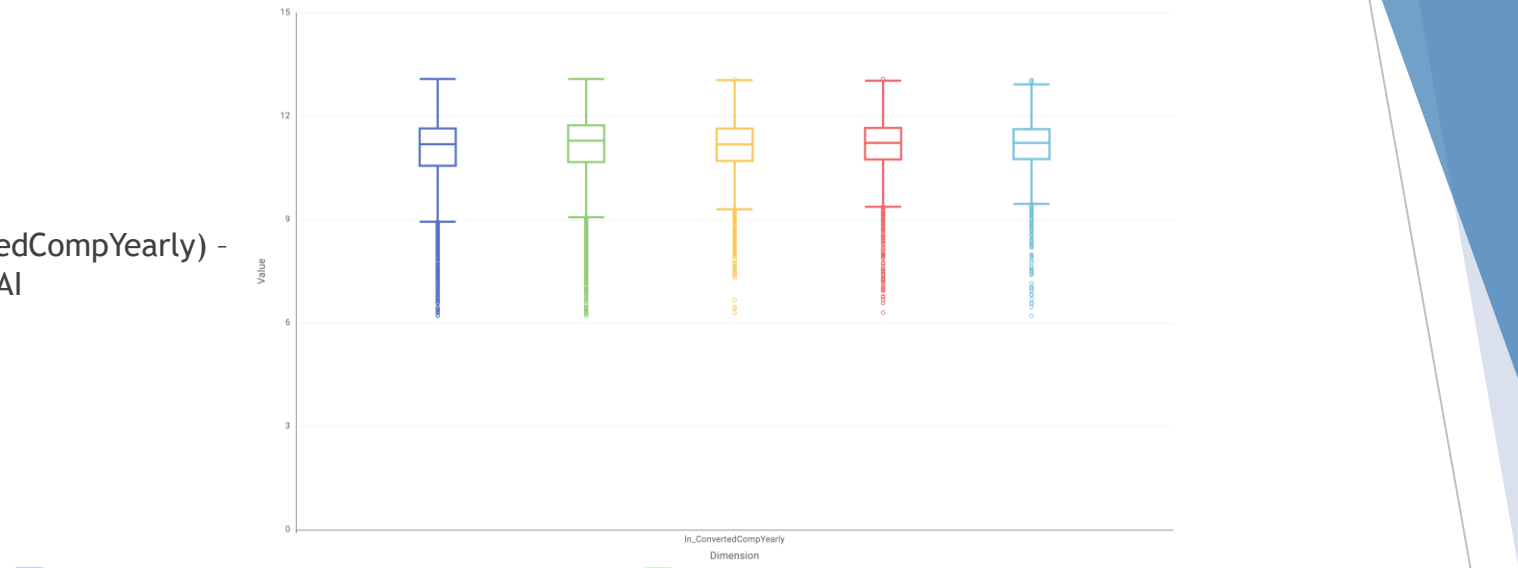
| RowID | Age Number (... | EdLevel Number (... | YearsCode Number (... | OrgSize Number (... | AISelect Number (... | AIAgents Number (... | JobSat Number (... | ln_ConvertedCompYearly Number (Float) |
|---|---|---|---|---|---|---|---|---|
| Age | 1 | 0.073 | 0.794 | 0.014 | -0.097 | -0.022 | 0.073 | 0.336 |
| EdLevel | 0.073 | 1 | 0.047 | 0.081 | -0.013 | 0.002 | -0 | 0.07 |
| YearsCode | 0.794 | 0.047 | 1 | 0.016 | -0.146 | -0.047 | 0.081 | 0.382 |
| OrgSize | 0.014 | 0.081 | 0.016 | 1 | -0.017 | -0.024 | -0.047 | 0.151 |
| AISelect | -0.097 | -0.013 | -0.146 | -0.017 | 1 | 0.268 | 0.018 | -0.09 |
| AIAgents | -0.022 | 0.002 | -0.047 | -0.024 | 0.268 | 1 | 0.052 | -0.043 |
| JobSat | 0.073 | -0 | 0.081 | -0.047 | 0.018 | 0.052 | 1 | 0.088 |
| ln_ConvertedCompYearly | 0.336 | 0.07 | 0.382 | 0.151 | -0.09 | -0.043 | 0.088 | 1 |

On the slide we only show **Age, EdLevel, YearsCode, OrgSize, AISelect, AIAgents, JobSat and ln_ConvertedCompYearly**, because they summarize the most relevant patterns and there was no room for all the variables. The full 56×56 correlation matrix, however, gives a quite clear overall picture:

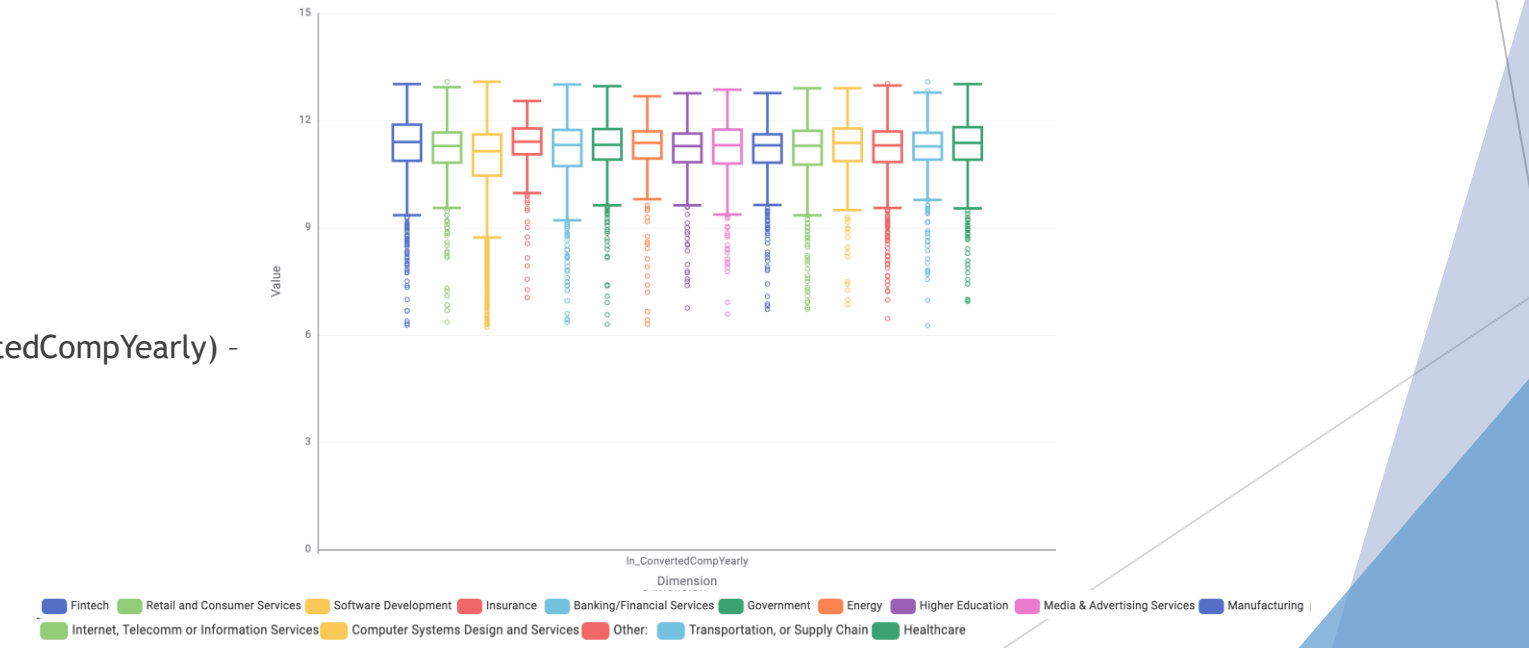
- **Demographics & experience:** Age and YearsCode are strongly correlated and both relate moderately to higher compensation. Education and company size are positively but more weakly linked to pay.
- **AI adoption:** AISelect, AIAgents and LearnCodeAI correlate with each other, but show only very small correlations with demographics, job satisfaction, or salary, suggesting AI use is broadly distributed across profiles.
- **Work arrangements:** Remote and hybrid work are strongly negatively correlated. Remote work shows mild positive correlations with compensation and independent contracting.
- **Job satisfaction:** JobSat has generally weak correlations, with only modest positive links to role stability and compensation.
- **Roles, industries & regions:** Dummy variables in these groups exhibit very small correlations, indicating high heterogeneity and no strong linear patterns.

# Bivariate analysis of continuous and categorical variables

ln(ConvertedCompYearly) – LearnCodeAI

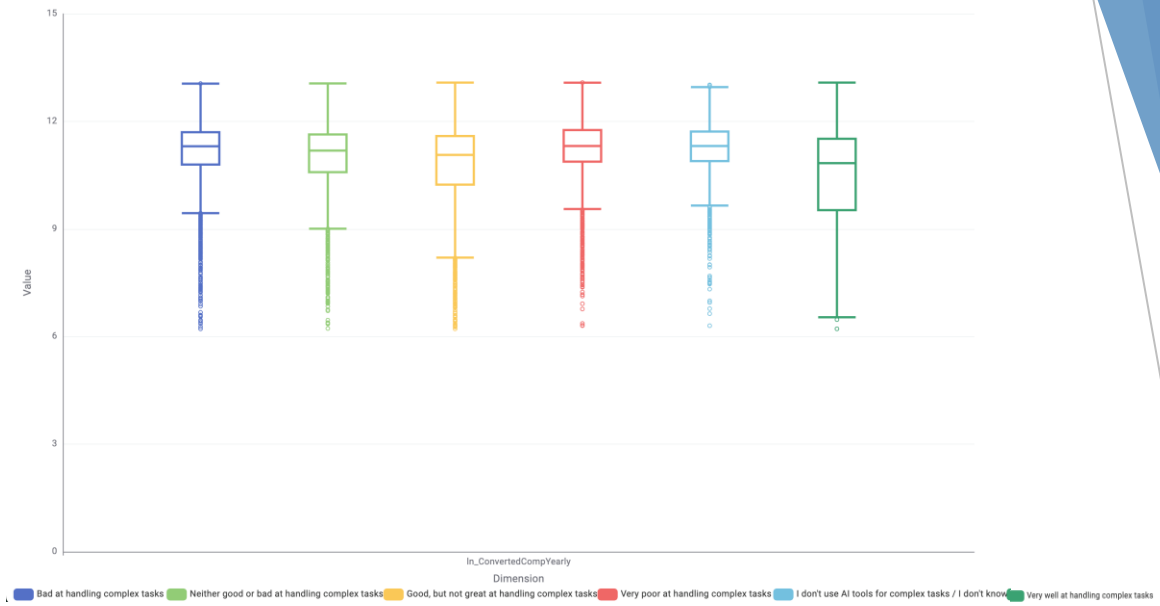



ln(ConvertedCompYearly) – Industry

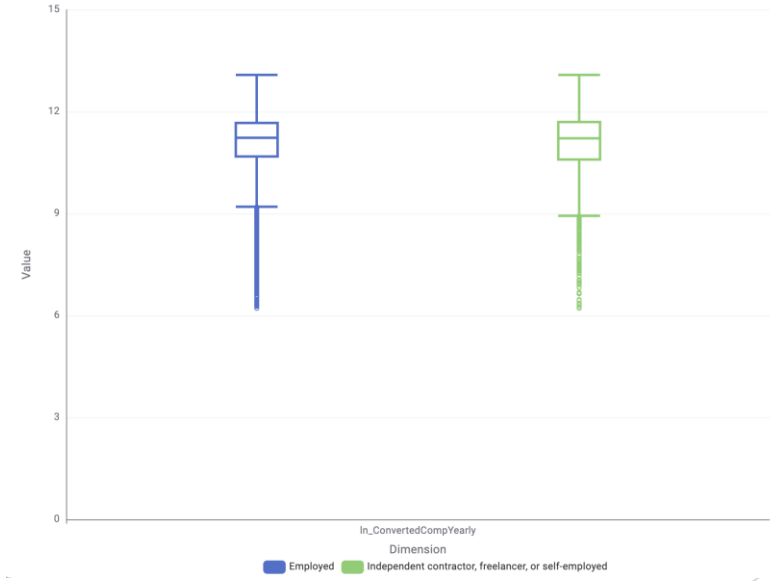


# Bivariate analysis of continuous and categorical variables

ln(ConvertedCompYearly) – AIComplex

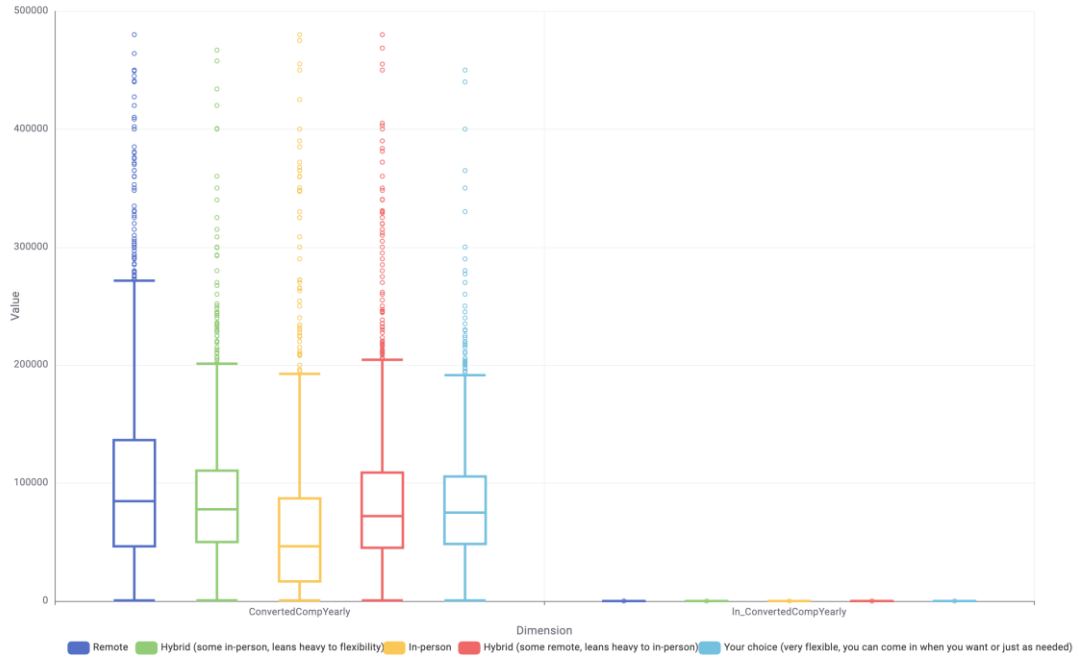



ln(ConvertedCompYearly) – Employment

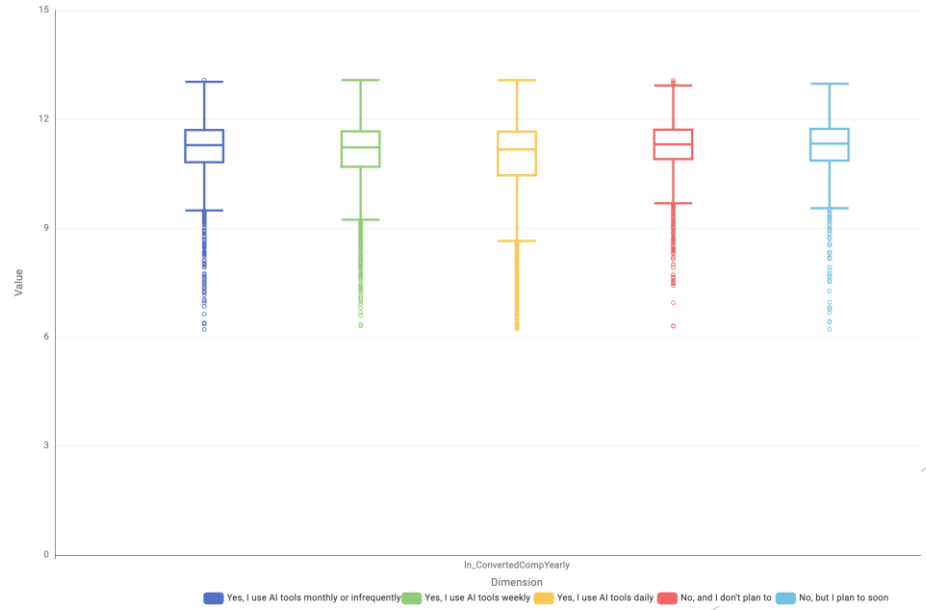


# Bivariate analysis of continuous and categorical variables

ln(ConvertedCompYearly) - RemoteWork

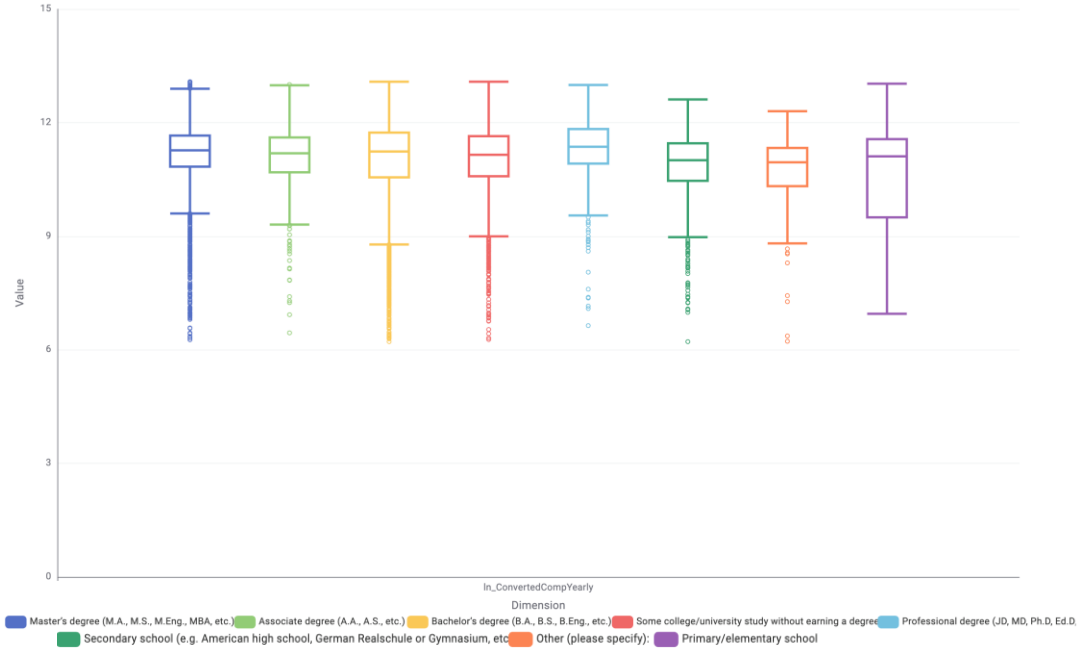



ln(ConvertedCompYearly) - AISelect

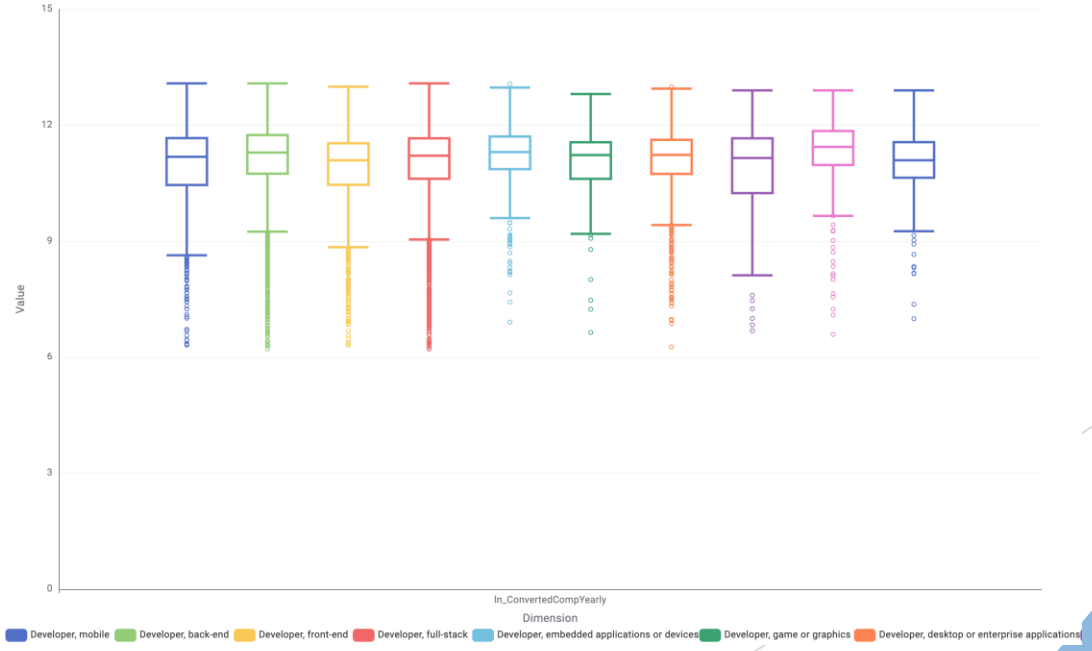


# Bivariate analysis of continuous and categorical variables

ln(ConvertedCompYearly) - EdLevel

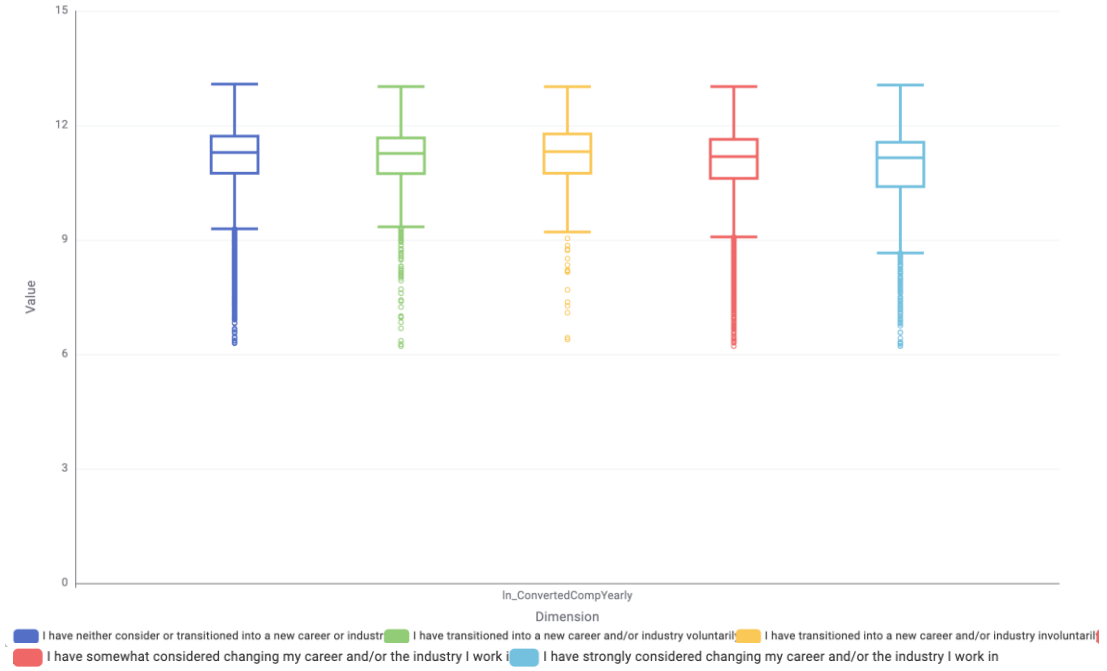



ln(ConvertedCompYearly) - DevType

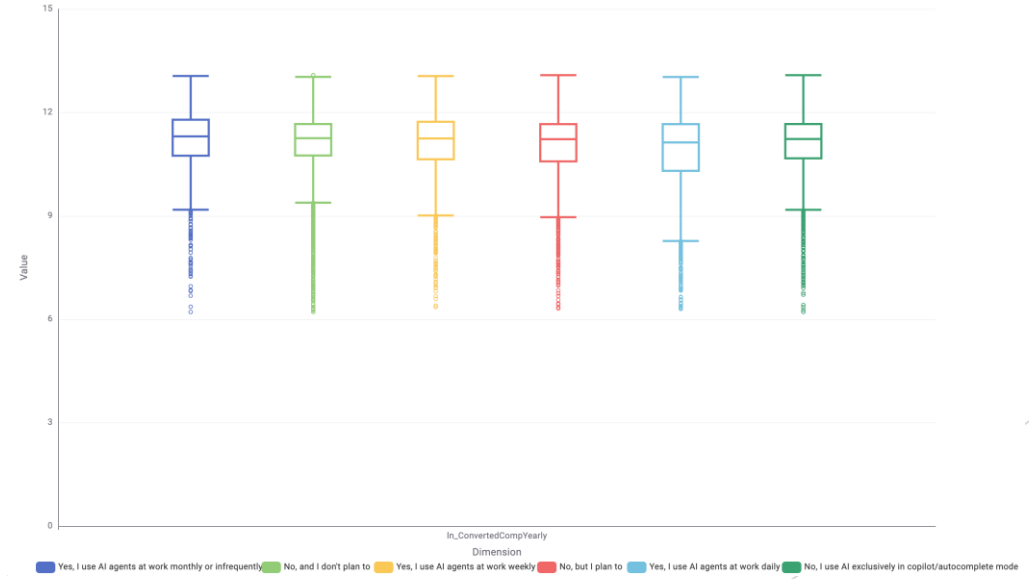


# Bivariate analysis of continuous and categorical variables

ln(ConvertedCompYearly) - NewRole




ln(ConvertedCompYearly) - AIAgents



# 5. Data preparation

## ADDITIONAL PREPROCESSING

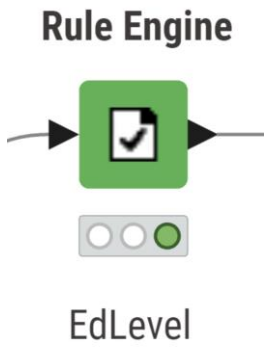
# Feature engineering – EdLevel

We transform the variable **EdLevel**, into a **numerical ordinal variable** to better capture the progressive nature of educational attainment.

Each education category is assigned a numeric score reflecting its relative level. The encoding preserves the inherent ordering across education levels while making the feature usable for numerical algorithms.

The following scoring scheme was adopted:

- Primary / elementary school → 0
- Secondary school → 1
- Some college / university without a degree → 2
- Associate degree → 3
- Bachelor's degree → 4
- Master's degree → 5
- Professional degree (JD, MD, PhD, etc.) → 6




This numeric encoding provides a structured representation of educational progression, supports consistent comparisons across respondents, and facilitates the use of the variable in downstream modelling and statistical analyses.

# Feature engineering – RemoteWork

We consolidate the variable **“RemoteWork”** into a smaller and more meaningful set of labels.

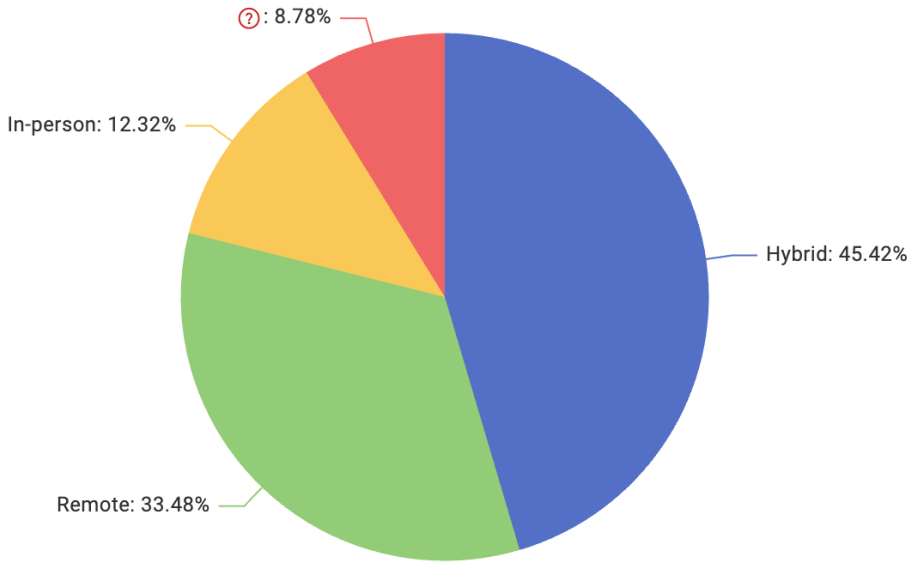
The following mapping scheme was implemented:

- Multiple hybrid descriptions

  → **“Hybrid”**
- Purely remote arrangements

  → **“Remote”**
- Fully on-site arrangements

  → **“In-person”**

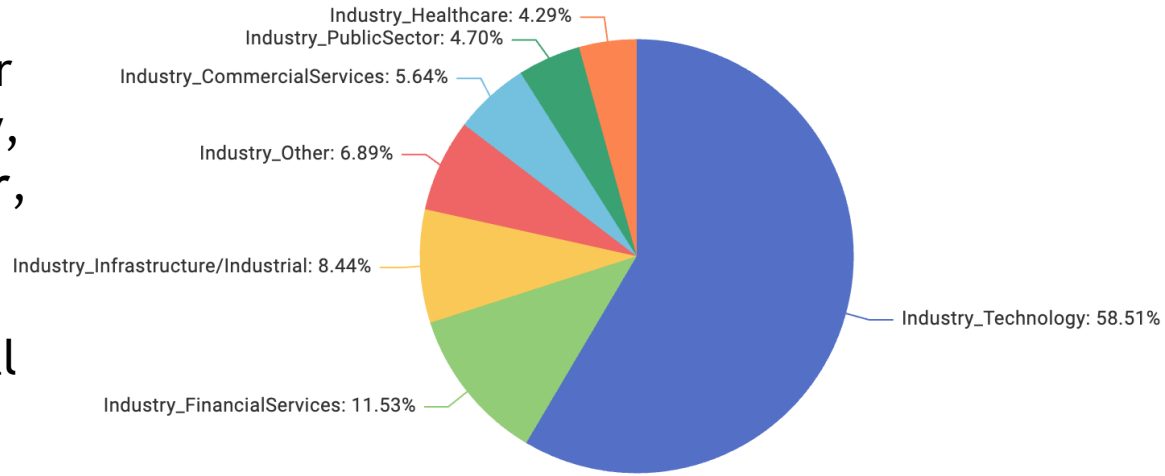



This consolidation reduces noise within the categorical variable, enhances interpretability, and supports cleaner downstream modelling by limiting the number of distinct classes to the three main working modalities.

# Feature engineering – Industry

To improve interpretability we consolidate the original categories into a smaller set of **higher-level industry groups**.

The consolidation groups semantically similar sectors under unified labels such as **Technology**, **Financial Services**, **Public Sector**, **Healthcare**, **Industrial/Infrastructure**, and **Commercial Services**. All residual or infrequent categories were assigned to a general **“Other”** class.

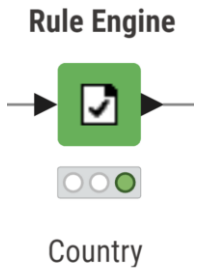



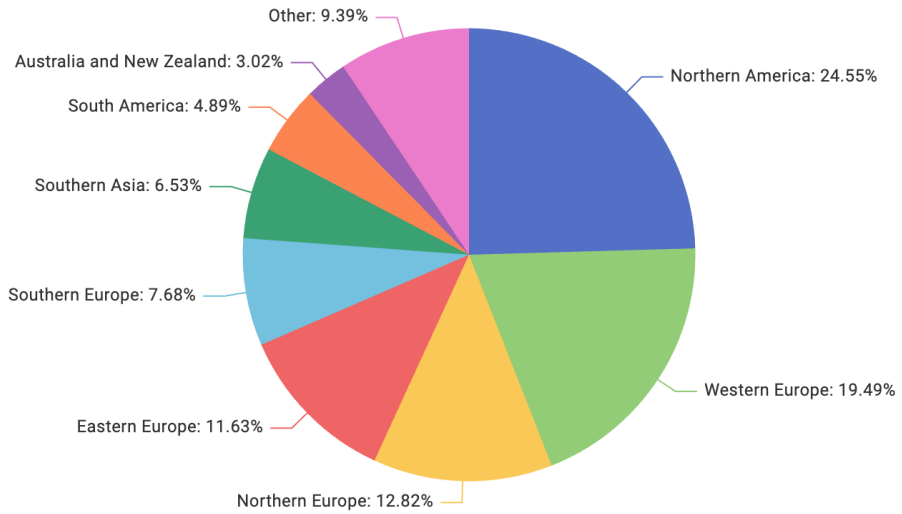

This harmonization significantly reduces categorical fragmentation and highlights the most represented sectors within the dataset.

# Feature engineering – Country

We grouped the original country-level responses into broader **geographical macro-regions** according to the United Nations Geoscheme to reduce sparsity and improve model interpretability. The dataset included a large number of distinct countries, many of which were represented by only a few observations. To address this issue, each country was mapped to its corresponding world region (e.g., *Western Europe*, *Southern Asia*, *Northern America*, *Middle Africa*).

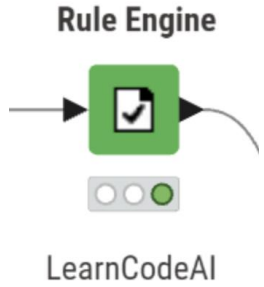
This transformation reduces the dimensionality of the variable, avoids overly fragmented categories, and allows the models to capture regional patterns more effectively. Countries that were ambiguous, rare, or did not fit into the predefined regions were assigned to the category *unknown*.



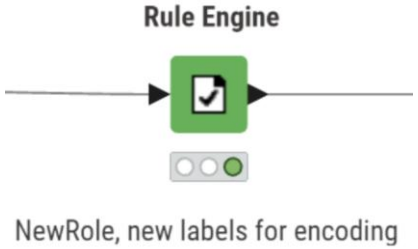



# Feature engineering – LearnCodeAI and NewRole

We transformed the original **LearnCodeAI** variable into a binary indicator capturing whether respondents engaged in AI-related learning during the past year. Responses indicating the learning of AI-enabled tools, either for professional development or personal curiosity, were grouped under **TRUE**, while all other responses, including non-AI learning or no learning activity, were assigned to **FALSE**.




This transformation reduces category complexity and allows the variable to be used effectively in models that require numerical or binary inputs

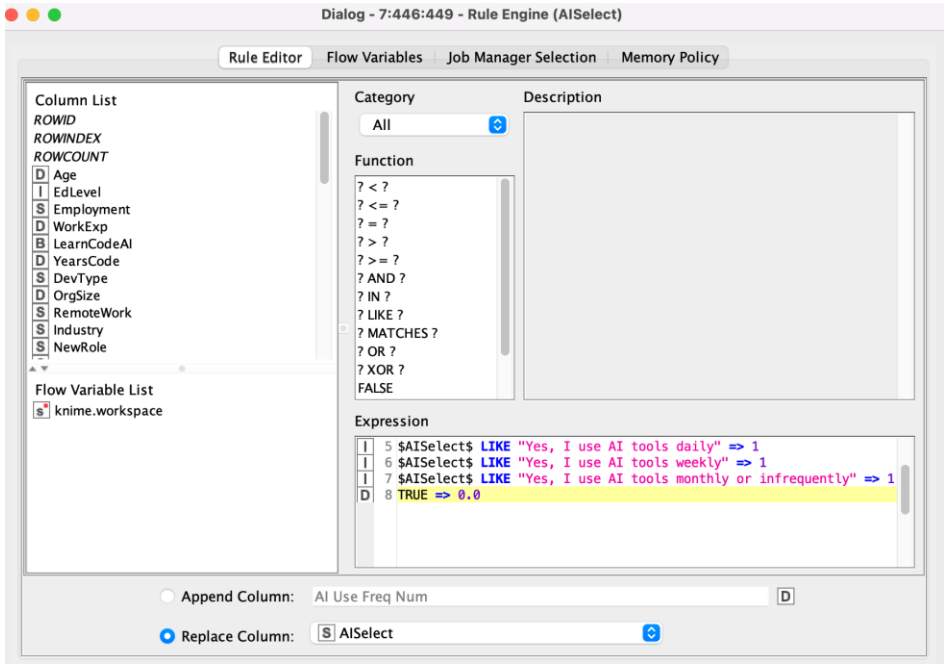



The answers to **NewRole** variable were just renamed to allow a clearer One-hot encoding: *NewRole_Stable*, *NewRole_Cons_Somewhat*, *NewRole_Cons_Strongly*, *NewRole_Trans_Invol*, *NewRole_Trans_Vol*.
Missing or uncategorized responses (missing values and remaining responses) were assigned to **NewRole_Unknown** or **NewRole_Other**.

# Feature engineering – AI Select

The variable **AISelect** originally included several textual descriptions about how frequently respondents use AI tools. To simplify this feature, we convert these qualitative responses into a **binary indicator** of AI usage.

All respondents who reported using AI tools *daily*, *weekly*, or *monthly* were assigned a value of **1**, identifying them as active AI users. All remaining or unspecified responses were assigned **0**, indicating little to no AI tool usage.



# Feature engineering – AIAgents

The variable **AIAgents** captures how often respondents use AI agents in their daily work. To simplify the feature and make it suitable for analysis, the original textual responses were converted into a **binary indicator**.

Respondents who reported using AI agents *daily*, *weekly*, or even *monthly/infrequently* were assigned a value of **1**, marking them as active AI-agent users. Those who use AI agents only in autocomplete/copilot mode, do not use them, or plan to use them in the future were assigned **0**.

# Feature engineering - YearsCode

**Outliers** in experience-related variables were handled across both preprocessing stages using explicit thresholds to ensure realistic values. In the basic preprocessing, **WorkExp** contained implausible entries such as **100 years of work experience**, which were directly removed as they could not represent credible career spans.

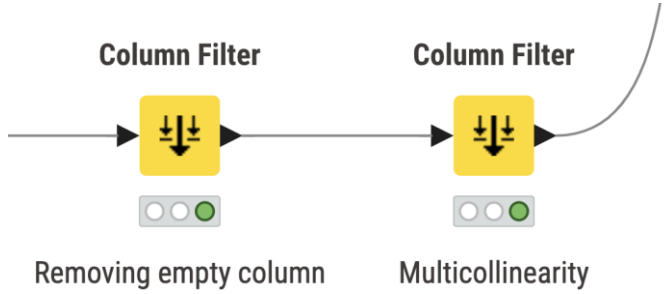
In the additional preprocessing, **YearsCode** was refined by identifying all values **greater than 45 years** as unrealistic for coding experience; these extreme entries were first consolidated into a placeholder to capture them consistently and were then removed through a row filter. This approach ensured that both variables remained within plausible ranges and did not introduce distortion into downstream analyses.

In the same step, **EdLevel** was cleaned by excluding observations where respondents selected “Other,” as this category did not provide meaningful educational information and would not be useful for downstream encoding or modelling.

# Variable encoding

In the final stage of the additional preprocessing, the **Domain Calculator** was used to update and correct the metadata of the cleaned dataset. Once the domains were aligned with the cleaned data, the **One-to-Many** node was applied to transform the selected categorical variables into dummy variables.

# Removing Empty Columns and Multicollinearity




In the final stage of the additional preprocessing, two Column Filter nodes were applied to refine the feature set generated after one-hot encoding. Although the dataset now contained **119 columns**, only a portion of these were dummy variables created from categorical features; the remaining columns were numerical or transformed variables.

The first Column Filter removed empty dummy columns, generated during one hot encoding with no data for the specific category.

The second Column Filter addressed **multicollinearity**: **WorkExp** was removed due to its strong correlation with **YearsCode**, which was retained as it shows the highest correlation with the target variable. Additional excluded columns are the baselines of the regression, removed after one hot encoding to address multicollinearity.

After applying both filters, the final dataset used for modeling consisted of **81 variables**.

# Baseline Categories in Regression Models

Baseline categories in the regression models are the result of the One-to-Many encoding, where one category per categorical variable is omitted. These omitted categories serve as the **reference levels** against which all other dummy variables are compared in the linear and logistic regression models.

The variables removed in the multicollinearity Column Filter (except for WorkExp) correspond exactly to these baselines. For example, removing *Employed_Employment* indicates that “Employed” is the baseline for Employment; removing *OpSys_MacOS* makes MacOS the reference operating system; removing *Language_Python* sets Python as the baseline programming language, and so on.

This structure ensures that regression coefficients are interpretable — each dummy coefficient represents the effect of being in a given category **relative to the omitted baseline** — and that the model matrix does not suffer from collinearity issues.

# 6. Modeling

# Models used

In this project, we implement a range of predictive models to capture both linear and non-linear relationships in the data. Tree-based methods such as Random Forest and Gradient Boosting are applied directly after the basic preprocessing pipeline, as these algorithms can inherently manage categorical variables. In contrast, linear and polynomial regression models rely on an extended preprocessing workflow that includes numerical encoding and feature normalization. The same preprocessed data is also used to train a Multilayer Perceptron, ensuring that all algorithms requiring numerical input operate on a consistent and properly scaled feature set.

# Regressions

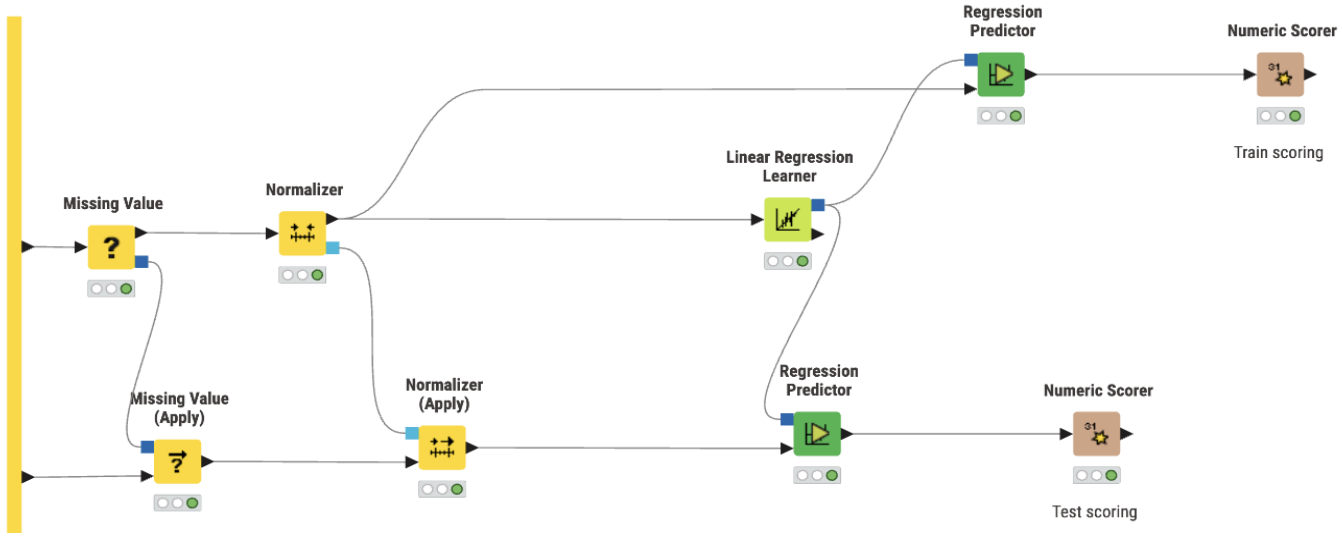
For both regression models, we applied an 80–20 table partitioning strategy, resulting in 11,706 rows for training and 2,927 rows for testing. Before model fitting, we created two parallel preprocessing pipelines: one applying z-score standardization to all numerical predictors, and one leaving the original, unscaled values. Standardization does not change the underlying structure of the data but places all predictors on a comparable scale, allowing regression coefficients to be directly interpreted in relative terms.
In contrast, the non-standardized models retain coefficients in their original units, making them useful for interpreting the absolute effect of each feature on the target variable. Using both approaches therefore provides a more comprehensive interpretability framework.

To prevent **data leakage**, missing values were imputed exclusively on the training data. The imputation model, based on the median of each numerical feature, was learned on the training split and then applied unchanged to the test split. This ensures that no information from the test set influences the model during preprocessing, keeping the test data completely untouched.

The workflow for both regression models is organized into two parallel branches. The first branch trains the model and evaluates its performance on the training set, while the second branch applies the trained model to the test set. This structure allows for a direct comparison between training and test performance, enabling us to detect and assess potential overfitting.

# Linear regression

In the upper branch, the normalized training data is passed to the Linear Regression Learner, which fits the regression model using the standardized features.

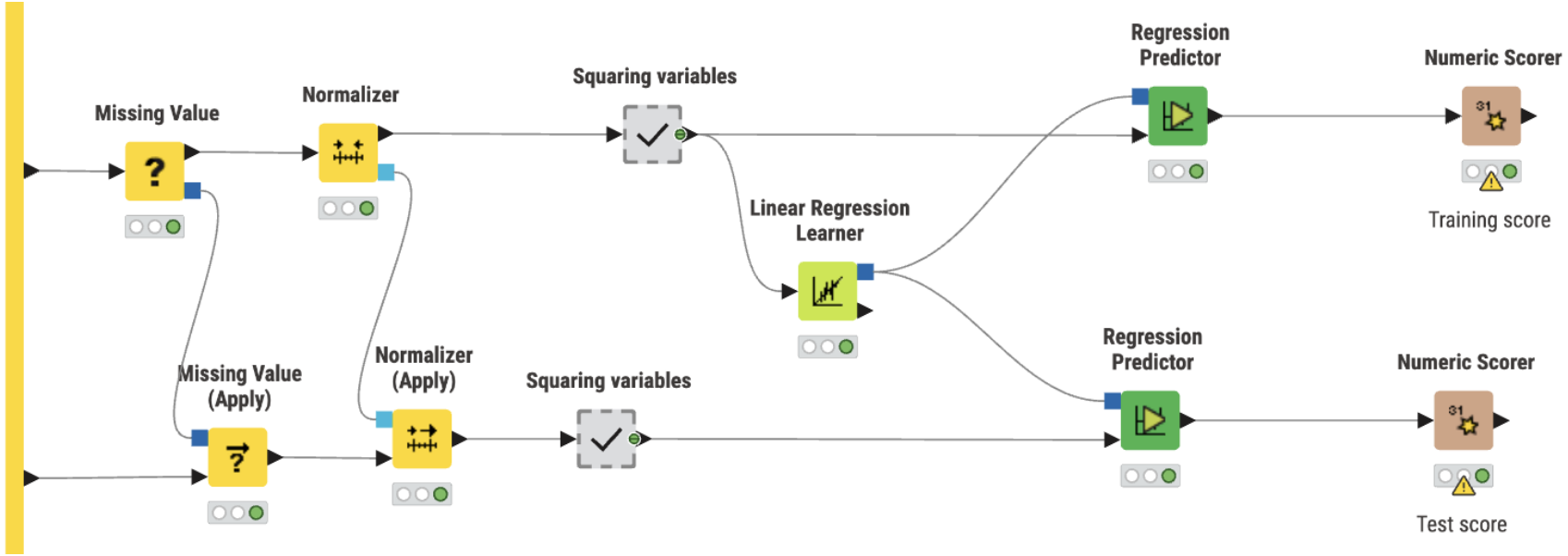



The trained model is then connected to a Regression Predictor, which generates predictions on the same training set. These predictions are evaluated by the Numeric Scorer, which computes performance metrics for the training phase.

In the lower branch, the normalized test data is then passed to another Regression Predictor, which uses the previously trained model to compute predictions on the hold-out test set. These predictions are evaluated using a second Numeric Scorer, providing the test-set performance metrics.
This structure guarantees a clean separation between training and testing, and a reliable evaluation of the model's generalization ability.

# Polynomial Regression

The workflow differs from the standard linear workflow in one key aspect: only a selected set of variables is transformed by adding their squared terms.
Instead of using KNIME's Polynomial Regression Learner, which automatically generates squared terms for all input variables, including one-hot encoded categorical features, leading to redundant or meaningless polynomial expansions, we manually created the quadratic features. This was done using a dedicated "Squaring variables" step applied only to the variables where a non-linear effect is reasonable. These squared columns were then appended to the dataset while all other variables remained in their original linear form. This approach avoids unnecessary expansion of number of predictors, and prevents the model from being cluttered with redundant or collinear features.




The resulting workflow extends the linear regression model with well-chosen quadratic terms, capturing potential curvature in the relationships without compromising model.

# Regressions results

Both regression model shows very similar results on the training and test sets, indicating good generalization and no evident overfitting.

**$R^2$ and Adjusted $R^2$ on Linear Regression**

- *Training* : $R^2$ = 0.515, Adjusted $R^2$ = 0.512
- *Test*: $R^2$ = 0.511, Adjusted $R^2$ = 0.497

**$R^2$ and Adjusted $R^2$ on Polynomial Regression**

- *Training*: $R^2$ = 0.545, Adjusted $R^2$ = 0.542
- *Test*: $R^2$ = 0.539, Adjusted $R^2$ = 0.524

Across both the linear and polynomial models, performance on the training and test sets is highly consistent, indicating good generalization and no signs of overfitting. The linear model explains about 51–52% of the variance in the log of yearly compensation, while the polynomial model increases this to roughly 54–55%, showing a modest improvement in explanatory power. In both cases, the **adjusted $R^2$ remains very close to the regular $R^2$**, indicating that the additional variables included in the models, particularly the polynomial terms, are not penalizing model complexity and therefore provide meaningful contribution.

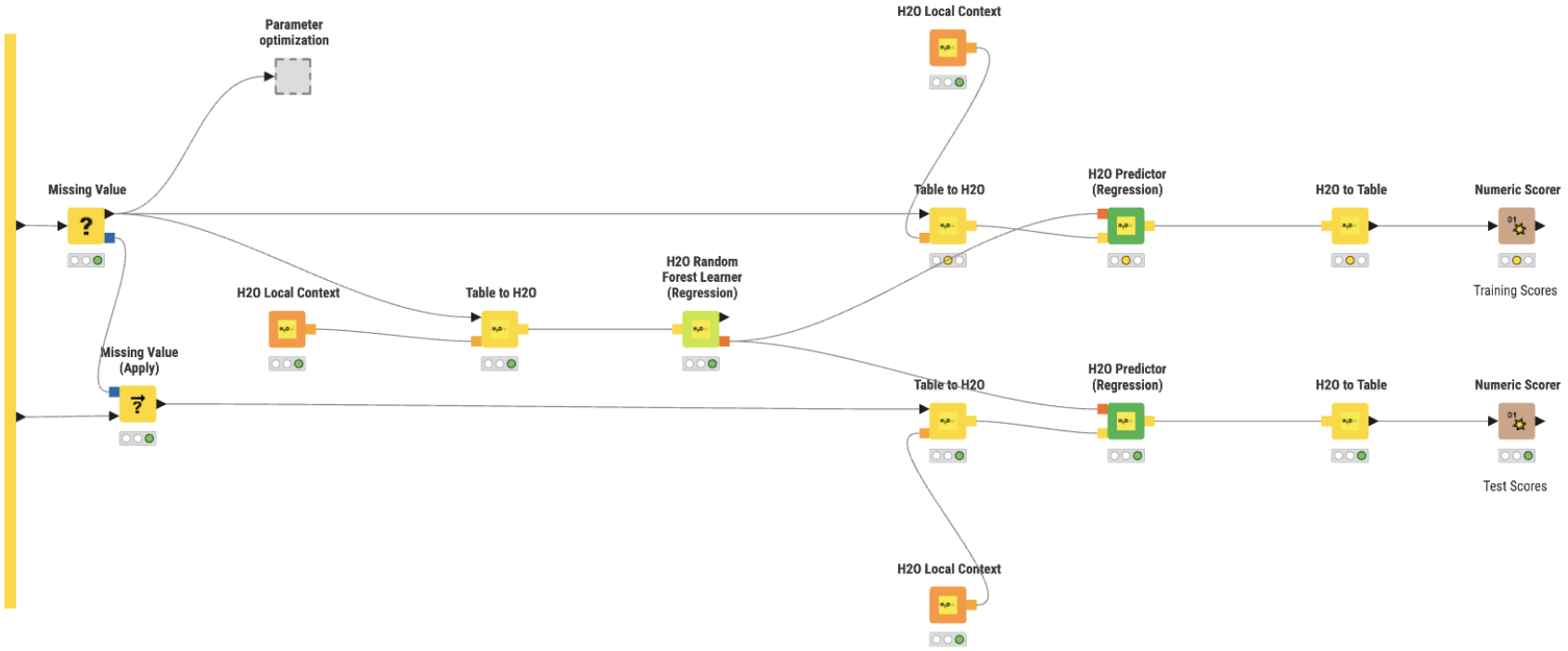
# Tree Based Models

For the tree-based models (Random Forest and Gradient Boosting), we first transformed all categorical variables using a One to Many node, which applies one-hot encoding. This step converts each categorical feature such as Education Level, Employment Type, DevType, Industry, Country, AISelect, AIComplex, and AIAgents—into a set of binary indicator columns, allowing the models to process them correctly.

After encoding, we applied a Table Partitioner to split the dataset into a 90% training set and a 10% test set. The training portion is later used not only to fit the models but also to perform hyperparameter tuning through cross-validation, ensuring that the selected parameters generalize well.

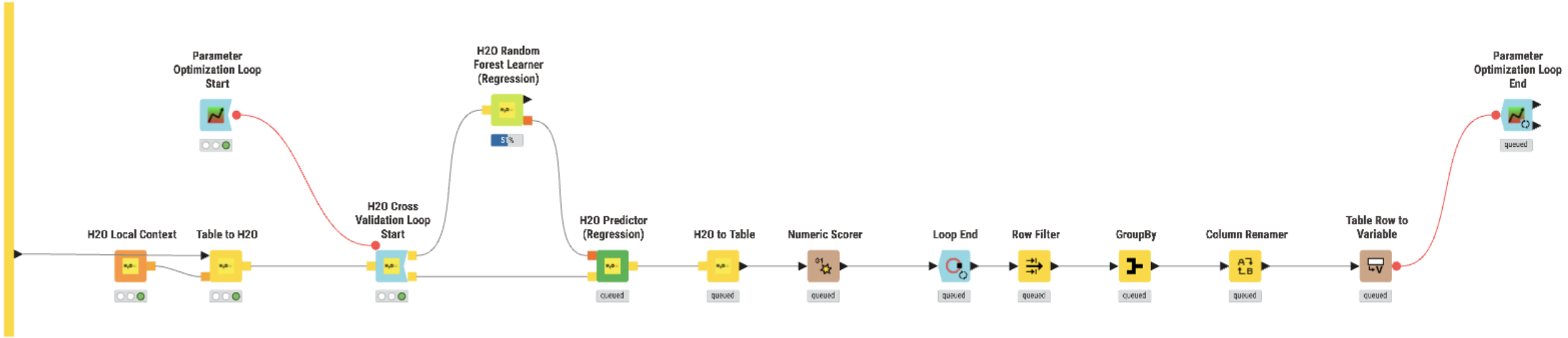
Following one-hot encoding, the dataset expanded to 276 columns, reflecting the large number of categorical levels. The resulting data split consists of 13,666 observations for training and 1,486 observations for testing, providing a solid basis for both model training and unbiased evaluation.

# Random Forest




The Random Forest workflow begins by imputing missing exclusively on the training data and applying the imputation model to the test split. Then we converted both training and testing data into an H2O frame using the Table to H2O node. Once in H2O format, the training dataset is fed into the H2O Random Forest Learner, which. The trained model is then applied to both the training data and the untouched 10% test split through two separate H2O Predictor branches. Predicted values are converted back into KNIME tables using H2O to Table nodes, and the model's performance is evaluated separately on each split using Numeric Scorer nodes.

# Hyperparameter tuning




Before training the final model, we optimized the Random Forest hyperparameters using KNIME's Parameter Optimization Loop in combination with H2O's cross-validation framework. The training data is first converted into an H2O frame and passed to the Parameter Optimization Loop Start, which generates different hyperparameter configurations to explore. Each configuration is evaluated inside an **H2O Cross Validation Loop**, where an H2O Random Forest model is trained and assessed through 5-fold cross-validation rather than a single validation split. A Numeric Scorer computes the performance for each iteration, and the results are collected by the Loop End. A subsequent GroupBy and Table Row to Variable sequence identifies the hyperparameter set that yields the best cross-validated performance. These optimal parameters are then fed back into the pipeline and used to train the final Random Forest model in the main workflow.

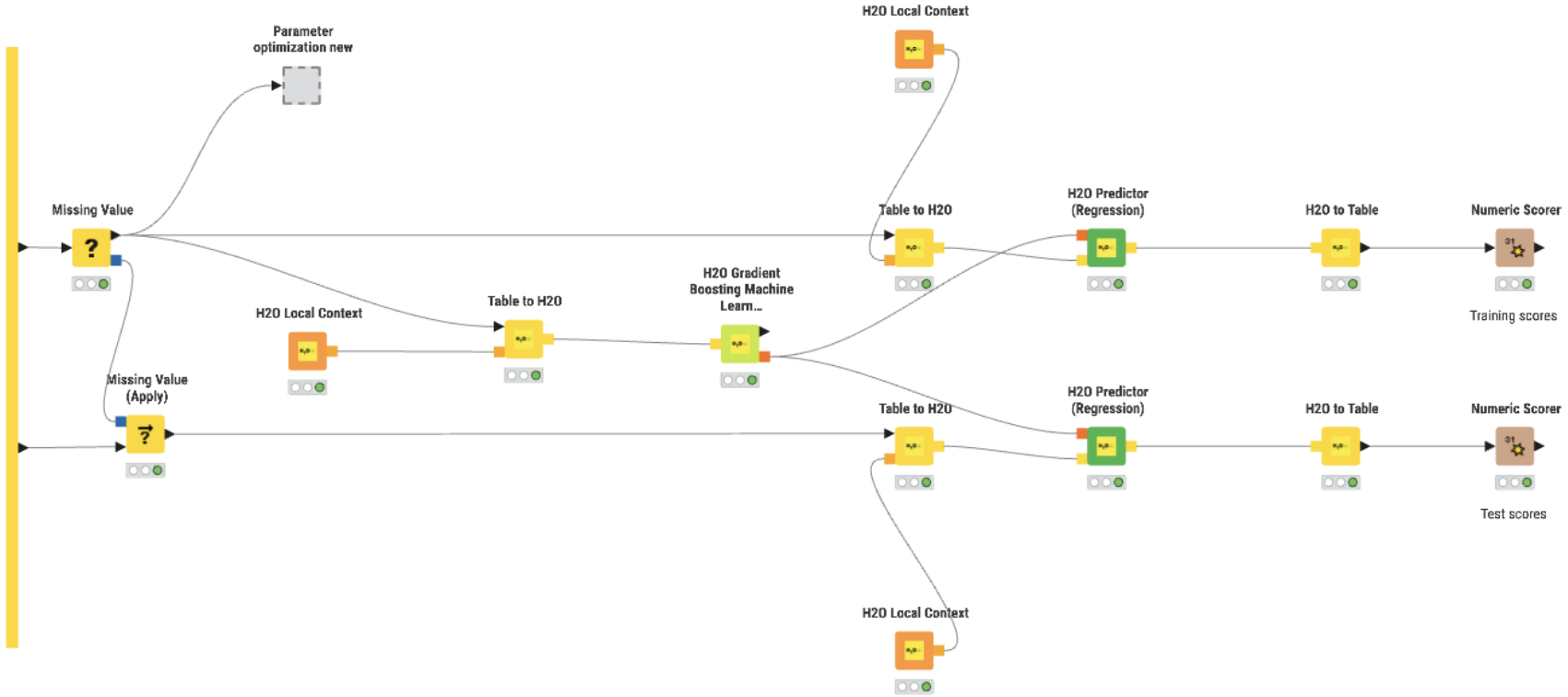
# Results Random Forest

The Random Forest model demonstrates a consistent predictive performance, with an $R^2$ of 0.608 on the training set and 0.546 on the test set, indicating good generalization to unseen data. Although the gap between $R^2$ and adjusted $R^2$ is minimal in training and noticeably larger in testing, this difference does not imply overfitting. Instead, it stems from the mathematical structure of adjusted $R^2$, which penalizes model complexity relative to the number of observations. Because the test set is much smaller than the training set, this penalty becomes disproportionately large in the test split, causing adjusted $R^2$ to drop sharply despite stable predictive performance. This interpretation is further supported by the root mean squared error, which is highly consistent across the two splits and increases by only about 5.5%.

| | Training set | Test set |
|---|---|---|
| $R^2$ | 0.608 | 0.546 |
| Adjusted $R^2$ | 0.6 | 0.442 |
| RMSE | 41330 | 43602 |

# Gradient Boosting

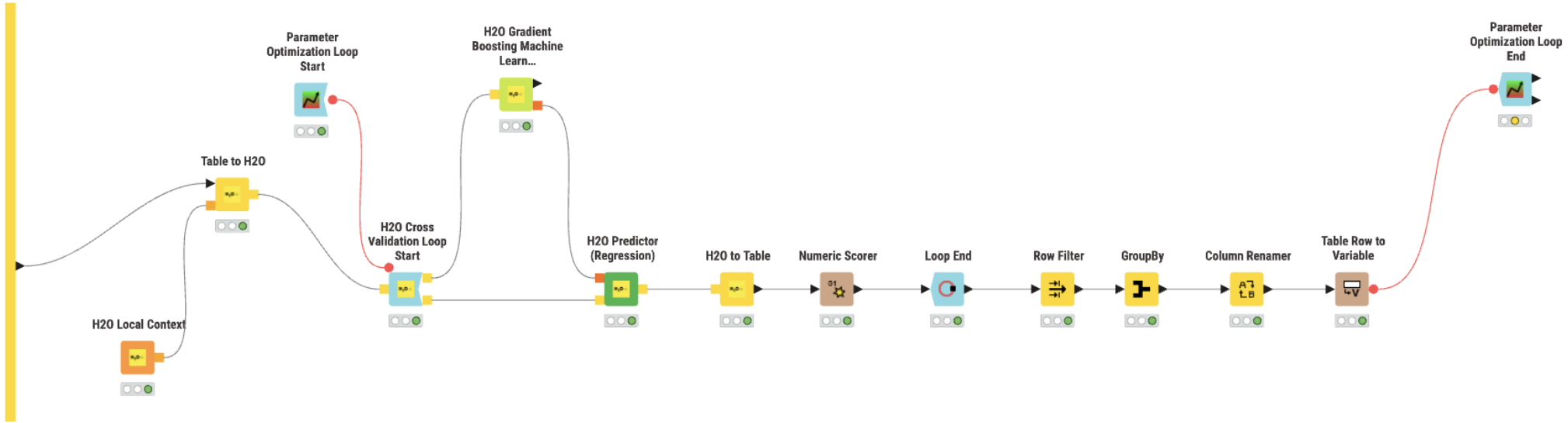
The Gradient Boosting workflow mirrors the structure used for Random Forest. Missing values are first imputed **only on the training set**, and the trained imputation model is then applied to the test split to avoid data leakage. Both datasets are converted into H2O frames using the *Table to H2O* node, after which the training frame is fed into the **H2O Gradient Boosting Machine (GBM) Learner** to train the model.




The trained GBM model is applied to both the training data and the untouched 10% test split through two separate *H2O Predictor* branches. Predictions are converted back to KNIME tables via the *H2O to Table* nodes, and model performance is evaluated separately on each split using *Numeric Scorer* nodes.

# Hyperparameter tuning

To improve the predictive power of the Gradient Boosting model, we ran a dedicated hyperparameter search using KNIME's *Parameter Optimization Loop*. Instead of relying on a single validation split, each hyperparameter configuration is tested through **H2O's internal 5-fold cross-validation**, which provides a more reliable error estimate for boosting-based models.
For every candidate configuration, such as learning rates and tree depths, the GBM is trained and validated inside the *H2O Cross Validation Loop*. The resulting performance scores are collected through the Loop End. A later aggregation step (*GroupBy* followed by *Table Row to Variable*) identifies the configuration with the lowest cross-validated error. This selected hyperparameter set is then injected back into the main workflow and used to fit the final GBM model on the full training partition.



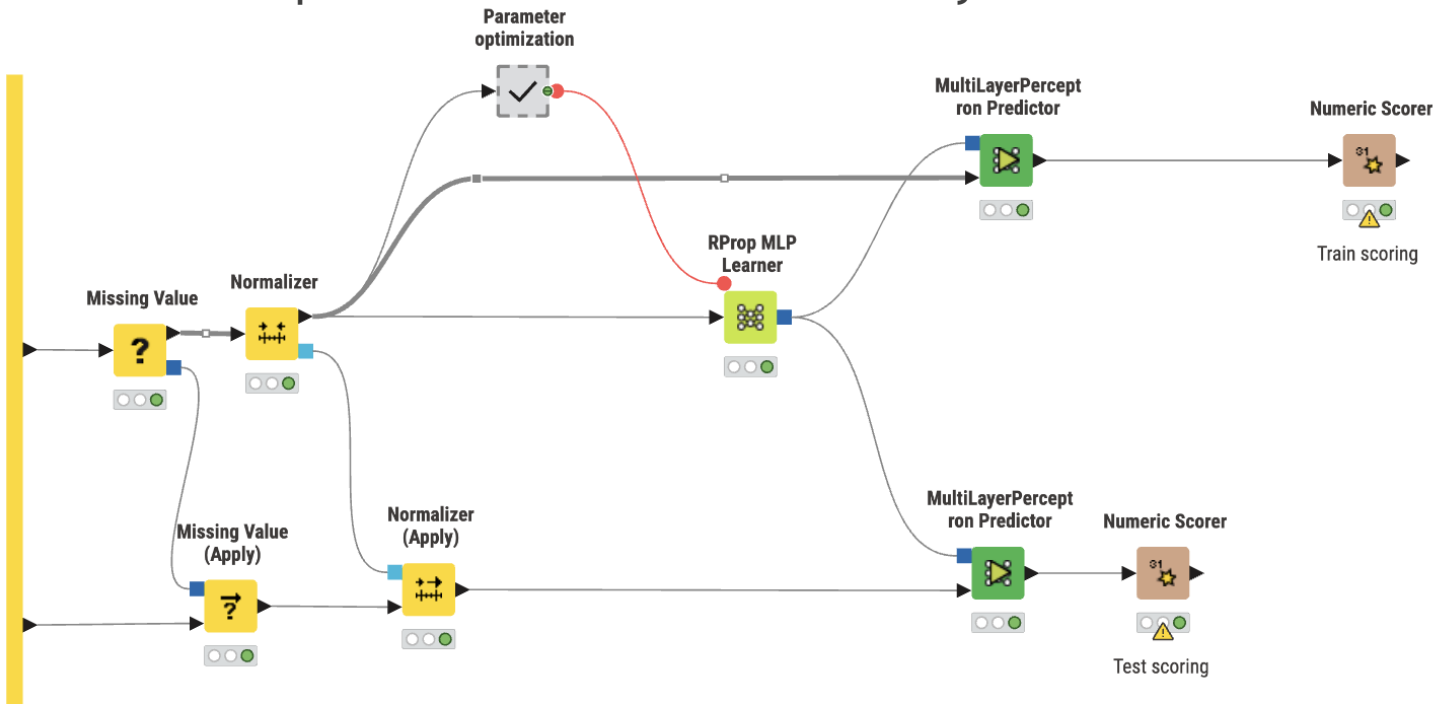
# Results Gradient Boosting

| | Training Set | Test Set |
|---|---|---|
| $R^2$ | 0.724 | 0.626 |
| Adjusted $R^2$ | 0.718 | 0.541 |
| RMSE | 34697 | 39551 |

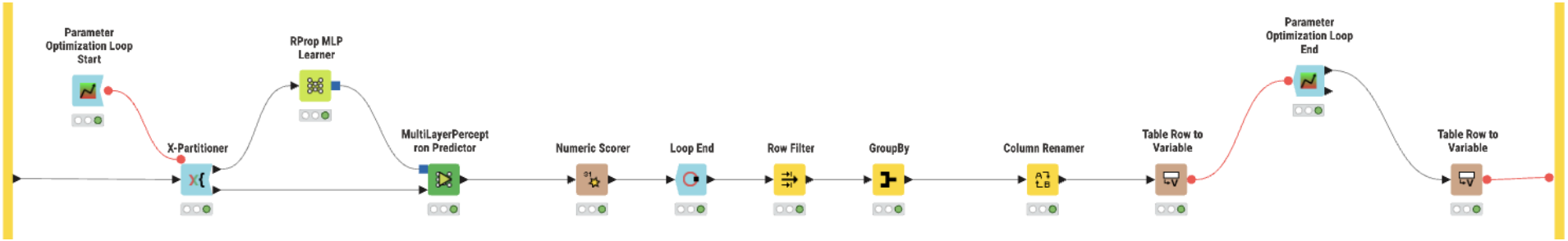
The Gradient Boosting model achieves an $R^2$ of 0.724 and an adjusted $R^2$ of 0.718 on the training set. When evaluated on the test set, $R^2$ decreases to 0.626 and adjusted $R^2$ to 0.541. The gap in adjusted $R^2$, similar to the case of Random Forests, is partly due to the much smaller size of the test set, which amplifies the complexity penalty relative to the number of predictors. Overall, despite the expected reduction in performance on the test split, the consistency of RMSE and the maintained predictive power suggest that the Gradient Boosting model does not suffer from severe overfitting and retains generalization capability.

# Multilayer Perceptron

The neural network workflow follows the same preprocessing strategy used for the regression models. Missing values are imputed **using only the training set**, and the resulting imputation model is then applied to the test data to prevent information leakage. After imputation, all numerical features are standardized with z-score normalization, ensuring that the neural network receives inputs on a comparable scale, an essential step for stable and efficient training. The preprocessed training data is passed to the **RProp MLP Learner**, which trains a multilayer. A Parameter Optimization block automatically evaluates different network configurations to identify the best-performing hyperparameters. The optimized MLP is then trained on the training set and applied to both the training and test splits through separate Predictor nodes. Performance on each split is assessed with Numeric Scorer nodes, allowing the detection of potential overfitting.



# Hyperparameter Tuning




To tune the Multilayer Perceptron, we used KNIME's Parameter Optimization Loop, which evaluates multiple network configurations to identify the best-performing set of hyperparameters. The training data is fed into the loop, and at each iteration the RProp MLP Learner trains a neural network using a different combination of parameters (such as the number of hidden neurons or the learning strategy). After each model is trained, a Numeric Scorer computes its predictive performance, and the results are collected by the Loop End. The Row Filter, GroupBy, and Table Row to Variable nodes then extract the configuration with the highest performance. This optimal configuration is passed back to the workflow via the Parameter Optimization Loop End, ensuring that the final MLP model is trained using the best hyperparameters found during the search.

# Results Multilayer Perceptron

The Multilayer Perceptron reaches an $R^2$ of 0.617 and an adjusted $R^2$ of 0.614 on the training set. When applied to the test split, performance decreases to an $R^2$ of 0.536 and an adjusted $R^2$ of 0.522, which reflects the model's sensitivity to the reduced sample size and the increased variability in unseen data. The RMSE moves from 0.097 on the training set to 0.105 on the test set; since the neural network was trained on a standardized version of the target variable, these values should be interpreted on the normalized scale rather than in the original salary units.

Overall, the MLP displays consistent behavior across training and test sets, with no signs of severe overfitting. The model successfully captures non-linear relationships in the data under the chosen network configuration and preprocessing strategy.

| | Training set | Test set |
|---|---|---|
| $R^2$ | 0.617 | 0.536 |
| Adjusted $R^2$ | 0.614 | 0.522 |
| RMSE | 0.097 | 0.105 |

# 7. Outcomes

| Comparative performance of the models (Test Set) | | |
|---|---|---|
| Model | $R^2$ | $R^2_{adj}$ |
| Linear Regression | 0.511 | 0.497 |
| Polynomial Regression | 0.539 | 0.524 |
| Random Forest | 0.546 | 0.442 |
| Gradient Boosting | 0.626 | 0.541 |
| Multilayer Perceptron | 0.536 | 0.522 |

# 8. Managerial Implications

# Big Picture

**Compensation Drivers: Model Insights**

- We analyzed correlation between compensation and regressors using two top $R^2$ models: Polyregression (interpretable coefficients) and XGBoost (validated via feature importance the regression's with p-values).
- **Primary Drivers of ln(salary):** (a) Geography (region/country), (b) Experience (YearsCode, Age, WorkExp), (c) OrgSize
- **Secondary Factors:** Role & tech stack, including dev type, languages (PHP, TypeScript, Go, HTML/CSS, C, Ruby, Kotlin, Bash), and OS (Windows/Mac/Linux/mobile)
- **Tertiary Influences:** Soft variables (JobSat, NewRole intentions, AI attitudes, AI learning) show lower impact than geography/experience
- **Model Agreement:** XGBoost and Polyregression largely concordant, with few exceptions on employment status (employed vs freelancer)
- **Notable Findings:** Low significance with EdLevel; "AI Complex" Paradox observed

# Age & Years of Coding

**Variables:** Age (β ≈ +0.0606, p ≈ 0.0), Sq_Age (β ≈ -0.000675, p ≈ 0.0), YearsCode (β ≈ +0.0658, p ≈ 0.0), Sq_YearsCode (β ≈ -0.00112, p ≈ 0.0)

**Significance:** All highly significant. Since dependent variable is ln(salary): small positive coefficient ≈ percentage salary increase; negative squared terms indicate concave relationships where benefits level off.

**XGBoost Validation:** YearsCode ranks #2 in importance, Age in top 20, WorkExp #3 (similar role). Both models strongly agree: experience & age are core drivers.

**Interpretation:** Early/mid-career experience heavily rewarded; returns diminish later. Age vs YearsCode disentangles "late starters" vs "early coders." Fitted curve shows: Age peaks ~45 years, YearsCode peaks ~29-30 years. Going 1→10 years coding yields ~+60% salary; 10→30 years adds ~+50% more.

**Managerial Implications:** Tie comp bands to experience with diminishing increments after 20-30 years. Pay compression at senior ages is natural; focus incentives on role scope/leadership/impact over tenure. Both factor should be considered jointly given their modest correlation of 0.7.

# Education

**Variables and Significance:** EdLevel (linear) (β ≈ +0.0360, p ≈ 0.19, not significant), Sq_EdLevel (β ≈ −0.00243, p ≈ 0.55, not significant)
*XGBoost Check:* XGBoost employs categorical EdLevel variables (Bachelor's, Master's, Professional degree, etc.), showing moderate but non-top-tier importance across categories.

**Interpretation:** Once geographic location, experience, tech stack, and developer type are controlled, overall education level contributes minimal predictive power when encoded as a simple numeric polynomial. XGBoost results indicate the relationship is nonlinear and category-specific (e.g., high school vs bachelor vs PhD), suggesting a single numeric EdLevel representation with quadratic term is insufficiently granular to capture true education effects.

**Managerial Implications:** Educational credentials do not independently drive compensation once location, role, and skill sets are accounted for. Organizations should avoid relying on numeric "education level scores" in salary modeling. When education matters for compensation decisions, explicit categorical distinctions (specific degree types) provide more accurate frameworks than continuous scales.
**Limitation:** The current model tests for a linear progression in salary across education levels (0–6). The variable's insignificance suggests the relationship is **non-linear** (i.e., higher degrees do not yield a uniform step-up in pay). Further analysis using categorical encoding is required to isolate specific premiums for degrees like PhDs, which our tree-based models successfully captured.

# Country / Region Effects

**Variables and Significance:** The regression employs regional dummies with Northern America_Country as baseline (removed). All coefficients are substantially negative and highly significant. Examples of approximate salary differences versus Northern America (computed as exp(β) − 1): Northern Europe (~34% lower), Western Europe (~37% lower), Western Asia (~62% lower), Eastern Europe (~69% lower), Southern Africa (~69% lower), South-eastern Asia (~84% lower), Western Africa (~94% lower).

**XGBoost Check:** XGBoost utilizes country-level dummies rather than regional aggregates, with United States, India, Switzerland, and UK ranking among top features. Both models concur that geographic location constitutes a primary compensation determinant.

**Interpretation:** These coefficients capture nominal salary differentials across regions, reflecting combined effects of cost of living, local labour market dynamics, and remote versus local contract structures.

**Managerial Implications:** Since compensation is USD-converted, these disparities reflect purchasing power differences and local market expectations. Remote hiring from lower-income regions offers cost advantages when roles permit asynchronous work, required skills are available locally, and cost optimization aligns with quality standards.

# Employment

**Variables and Significance:** Baseline (removed) is Employed_Employment. Independent contractor, freelancer, or self-employed_Employment (β ≈ +0.148, p ≈ 5e−10, strongly positive and significant). Effect: exp(0.148) − 1 ≈ ~16% higher salary versus an otherwise-similar "Employed" baseline.

**XGBoost Check:** This variable shows 0 importance in XGBoost, indicating disagreement between models. The tree model likely captures this signal through other variables such as region, remote work, developer type, and tech stack.

**Interpretation:** In the regression, after controlling for location, stack, and role, self-employed workers earn more. However, XGBoost ignores this variable, suggesting its predictive power overlaps substantially with other features already in the model.

**Managerial Implications:** Self-employed professionals typically command premium rates, with the regression suggesting an ~16% premium. However, given XGBoost's zero importance rating, this effect is fragile. Still, outsourcing projects to freelancers proves more expensive than hiring a dedicated developer when project duration extends beyond short-term engagements, when institutional knowledge retention matters, or when ongoing coordination costs exceed the premium differential.

# Remote Work

**Variables and Significance:** Baseline is In-person_RemoteWork (removed). Remote_RemoteWork (ß ≈ +0.0886, p ≈ 2.6e−5, significant) yields approximately +9% salary versus in-person. Hybrid_RemoteWork (ß ≈ +0.0453, p ≈ 0.025, significant) yields approximately +4-5% salary versus in-person.

**XGBoost Check:** In-person_RemoteWork, Remote_RemoteWork, hybrid variants, and "very flexible" remote categories all demonstrate clear but mid-tier importance in the model.

**Interpretation:** After controlling for other factors, fully remote roles command highest compensation, followed by hybrid, then in-person arrangements. This premium likely reflects remote positions accessing cross-border opportunities in higher-paying markets, increased competition for remote-capable talent, concentration of remote work in higher-paying sectors, and employer savings on office infrastructure costs.

**Managerial Implications:** Organizations limiting recruitment to in-person arrangements may need to offer premiums above local market rates to compensate for reduced flexibility. Enabling remote or hybrid work expands access to lower-cost labor markets but creates risk of internal pay disparities between geographic regions and between on-site versus remote staff, requiring careful compensation policy design.

# Job Satisfaction

**Variables and Significance:** JobSat (β ≈ +0.0284, p ≈ 3.7e−13, highly significant), roughly equivalent to +2.8% salary per one-unit increase in satisfaction.

**XGBoost Check:** JobSat demonstrates high importance, ranking approximately 15th among all features.

**Interpretation:** Higher satisfaction and higher compensation move together in a bidirectional relationship. Being well compensated likely boosts satisfaction levels, while satisfied employees may demonstrate stronger performance that drives compensation increases over time.

**Managerial Implications:** JobSat should not be used as a compensation input, as it is predominantly an outcome of salary rather than a causal driver. However, it serves as a valuable diagnostic tool: employee groups exhibiting both low salaries and low satisfaction represent clear retention and engagement hotspots requiring intervention.

**Methodological Note: Reverse Causality** While our model identifies Job Satisfaction as a significant predictor, it should be interpreted as an **outcome** of high compensation, not a driver. Including it improves model fit (R^2), but it absorbs variance that belongs to structural factors (like remote work or tech stack).

# AI Complex

**Variables and Significance:**
Baseline category removed: *I don't use AI tools for complex tasks / I don't know*.
Compared to the baseline, perceptions of AI ability show mixed salary effects. "Very poor at handling complex tasks" is associated with higher salaries (~+6–7%), while "very well at handling complex tasks" is associated with noticeably lower salaries (~−18–19%). Other categories ("bad," "good but not great," "neither good nor bad") are statistically non-significant.

**XGBoost Check:**
All AIComplex categories appear with non-zero importance, but none rank among top predictors.

**Interpretation:**
The pattern is not causal. High earners often work in domains where AI adds little value (infrastructure, legacy systems), leading them to rate AI poorly. Lower-paid developers in highly automatable roles find AI very effective. A possible Dunning–Kruger effect may also play a role: junior workers tend to overestimate AI's capabilities, while more experienced engineers recognize its limitations.

**Managerial Implications:**
Statements such as *"AI handles complex tasks very well"* should not be used as compensation signals. Treat these views as indicators of task type or experience level, not productivity. Teams highly confident in AI's ability may be in roles at higher automation risk; consider steering them toward harder, less automatable work through targeted upskilling.

# AISelect, AIAgents, LearnCodeAI

**Variables and Significance:**
Baseline dummy variables removed from the regression include *false_LearnCodeAI* and *"I don't use AI tools for complex tasks / I don't know"* for AIComplex.
Across both regression models and XGBoost, AI-related variables display only **small and mostly insignificant** effects on salary. Learning how to use AI-enabled tools (*LearnCodeAI*) and higher engagement with AI agents (*AIAgents*) show modest positive associations of roughly **+3-4%**. However, general AI usage frequency (*AISelect*) loses significance once other factors are controlled for.
In XGBoost, these features appear but are consistently ranked as **secondary**, far behind core predictors such as experience, region, and technical specialization.

**Interpretation:**
Overall, AI adoption and learning exhibit **non-zero but minor** relationships with compensation. These effects are real but weak, and they are overshadowed by structural factors like geography, seniority, and domain.

**Managerial Implications:**
While investing in AI upskilling can be encouraged as part of career development, its impact on compensation is **moderate**. It should be treated as one of several helpful signals—not as a primary driver of salary.

# Operating systems

**Variables and Significance:**
Using MacOS as the baseline, several OS categories show significant earnings differences. Windows users earn ~13% less, Ubuntu users ~4% less, and Android users ~6.5% less. iOS users earn ~7–8% more. Other systems (Linux, Debian, WSL, OtherOpSys) are not clearly significant.

**XGBoost Check:**
All OS categories appear in XGBoost, with Windows and MacOS relatively important, but overall OS effects remain secondary compared to region, experience, or role.

**Interpretation:**
These differences reflect **role and region mix**, not the OS itself. MacOS and iOS users cluster in higher-paid ecosystems (US, iOS development), while Windows/Android users are more common in lower-pay regions and broader job types.

**Managerial Implications:**
OS should not be a pay signal. Treat it as a **proxy for ecosystem or market segment**, useful for analytics but not for compensation decisions.

# Programming Languages

**Variables and Significance:**
Using Python as the baseline, several languages show significant salary differences. Higher-paying: TypeScript (~+8%), Go (~+8%), Ruby (~+7%), Kotlin (~+7%), and Bash/Shell (~+5%). Lower-paying: C (~−7%), HTML/CSS (~−9%), and PHP (~−16%). Most other languages are not significant.

**XGBoost Check:**
Languages are consistently important features and appear with non-zero importance, confirming that language stack meaningfully influences compensation.

**Interpretation:**
Compared to Python developers, engineers working with TypeScript, Go, Kotlin, Ruby, or Bash tend to earn more, likely because these languages cluster in modern backend, DevOps, and high-pay ecosystems. Lower salaries for HTML/CSS and PHP reflect their concentration in front-end or legacy web roles, which are often lower-compensation segments. Effects reflect **role and ecosystem**, not the language itself.

**Managerial Implications:**
For compensation: backend/infra stacks typically command higher salary bands. Roles centered on PHP or pure HTML/CSS may be easier to hire for and often correspond to lower-value work.
For upskilling: transitioning talent from PHP/HTML into more modern stacks (TypeScript, Go, Kotlin, cloud-focused tools) aligns with higher market compensation.

# Developer Role

**Variables and Significance:**
With *back-end developers* as the baseline, several roles show significantly lower salaries: front-end (≈−10%), full-stack (≈−7%), desktop/enterprise (≈−9%), mobile (≈−14%), QA/testing (≈−16%), and game/graphics (≈−18%). Embedded, DevOps, and AI-apps roles are **not statistically significant**, likely due to collinearity with languages/industry or low sample sizes.

**XGBoost Check:**
Several DevType categories appear with non-zero importance, though they rank below languages, region, and experience. The model agrees that role type contains meaningful, but secondary, salary signal.

**Interpretation:**
Backend engineers remain the highest-paid cluster, whereas front-end, full-stack, mobile, QA, and game development correspond to lower-pay segments in the market. DevOps, embedded, and AI-focused roles do not differ clearly, likely because their compensation is already explained by stack and industry.

**Managerial Implications:**
Expect higher salary bands for backend and infrastructure roles. Roles like QA, mobile, and game development often carry structural pay discounts. Encouraging career paths from front-end/QA/game into backend/infrastructure/cloud may help employees grow into higher-paid tracks.

# Industry

**Variables and Significance:**
Using *Technology* as the baseline, only two industry categories show clear salary differences. Financial Services pays substantially more (≈+13–14%), while the Public Sector pays noticeably less (≈−8–9%). Healthcare, Commercial Services, Infrastructure/Industrial, and "Other" industries show **no significant independent effects** once region, role, and stack are controlled for.

**XGBoost Check:**
More granular industry categories (Banking, Fintech, Government, Media, Energy, Healthcare, Manufacturing, etc.) appear with non-zero importance, confirming that industry adds signal, though less than core predictors like experience, role, and region.

**Interpretation:**
Relative to technology, Financial Services and Fintech tend to offer higher compensation, while Public Sector roles cluster at lower wages. Other industries show no strong salary premium or penalty after controlling for other factors, suggesting that industry effects mostly reflect market.

**Managerial Implications:**
Candidates coming from finance may have higher salary expectations. Public-sector talent is typically lower paid, so retention often depends on mission, stability, or benefits rather than compensation. For compensation planning, industry acts more as a context variable, useful for interpreting external offers but not a primary pay driver.

# 9. Limitations

# Limitations

- **Limited controls & omitted variables**
  Few variables directly capture key salary drivers (e.g. performance, internal level, negotiation), increasing the risk of **omitted variable bias** (examples of excluded variables could be: gender, productivity metrics, accurate seniority metric…)
- **Survey-based & self-reported data**
  Data quality depends on **survey design** and **respondents' accuracy**. Salary, experience and AI usage are self-reported and may suffer from recall error, misreporting and subjectivity.
- **Sample representativeness**
  Voluntary survey → **self-selection bias**. Some groups and rare categories (roles, countries, tech stacks) are **under-represented**, limiting generalizability.
- **Correlational, not causal**
  The analysis is **purely correlational**: we observe associations between AI usage and salaries, but we cannot claim causal effects.
- **Model complexity vs. data structure**
  In our setting, complex models (e.g. tree-based methods / ANN) only marginally outperform simpler linear specifications, partly because the data are dominated by dummy and categorical variables. This is a limitation in terms of pure predictive power, but also an advantage: it means that **most of the signal is already captured by simpler, transparent models**, which we can interpret in detail. As a result, our conclusions are **easier to explain and use for managerial decisions**, rather than being driven by an opaque black-box.

# Future Improvements

- **Longitudinal analysis:** Incorporate multiple survey years to track compensation trends over time, assess how AI adoption correlates with salary growth trajectories, and identify whether early AI adopters experience faster wage progression than late adopters or non-users.
- **Causal inference methods:** Apply propensity score matching, instrumental variables, or difference-in-differences approaches to move beyond correlation toward causal estimates of AI adoption effects on compensation, controlling for selection bias where high performers adopt AI earlier.
- **Hierarchical modeling:** Implement multilevel models that explicitly account for nested data structure (individuals within companies within regions), better separating individual-level effects from organization-level and market-level compensation determinants.
- **Interaction effects exploration:** Systematically test interactions between AI adoption and experience levels, geographic regions, or technical specializations to identify which developer segments benefit most from AI usage and where adoption translates into measurable compensation premiums.
- **External validation:** Combine Stack Overflow data with external sources (LinkedIn salary data, Glassdoor reports, company-disclosed compensation bands) to validate findings across multiple data collection methods and reduce survey-specific biases.

# Thank You