Lambton College at Queen's College of Business, Technology and Public Safety

Program: Big Data Analytics (DSMM)

Project Name:

US Developer Salary Prediction using Stack Overflow 2023 Data

Course: Advanced Python AI and ML Tools 01 (AML 2203)

Professor: Sagara Samarawickrama

Group Members:

Jennylynne Dominguez (C0929140)
Joyce Ann Murillo (C0927648)
Chaw Su Su Thinn (C0916347)
Hazel Portia Elaine Santos (C0915982)

Project Overview

This project aims to build a machine learning model that predicts the annual salary of U.S.-based software developers using the Stack Overflow 2023 Developer Survey. By analyzing developer demographics, professional background, technology usage, and job satisfaction, the model provides insights into the factors that influence compensation in the U.S. tech industry.

The original Stack Overflow 2023 dataset contained over 89,000 global responses and 84 features. After filtering for U.S.-based respondents and valid salary entries, we retained over 18,000 rows, which provided a large and consistent sample for modeling.

Why Focus Only on U.S. Respondents?

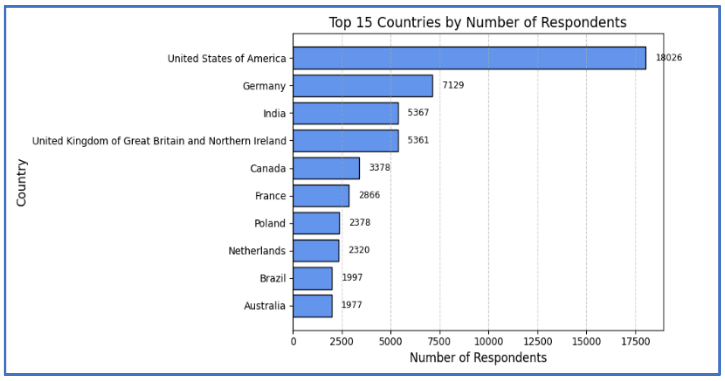
To ensure consistency and comparability in salary values, this project limits its scope to U.S.-based respondents only. The global dataset includes compensation reported in dozens of local currencies, making direct modeling difficult without introducing exchange rates, inflation adjustments, or cost-of-living normalization, all of which can introduce noise or bias. By focusing on the U.S. subset:

Currency is standardized (USD), eliminating the need for conversion.

Data volume is sufficient, with over 18,000 respondents, providing a robust sample size for modelling.

Based on the chart of the Top 15 Countries by Number of Respondents, it's evident that the United States has the highest number of survey participants, with 18,026 respondents, which is more than double the second highest (Germany, 7,129).

This large representation makes the US data more statistically robust and reliable for modelling.



The U.S. represents one of the largest and most mature tech markets, making it a valuable benchmark for salary trends in software development. According to the U.S. Bureau of Labor Statistics (BLS), the employment of software developers is projected to grow 17% from 2023 to 2033, which is much faster than the average for all occupations.

Dataset Summary

Source: Stack Overflow Developer Survey 2023 (public release)

Filtered for: U.S. respondents only

Format: Tabular CSV with ~80 structured survey features (free-text responses excluded)

Target : CompTotal

Data Cleaning and Preprocessing

Convert to Numeric and Drop all NA of CompTotal (Target)

- We dropped rows where the target (CompTotal) is missing using dropna()
- We converted the target feature to numeric by using pd.to_numeric

Drop all rows where the Currency column does NOT contain "USD"

- We filtered the dataset to retain only respondents who are located in the United States and whose Currency includes "USD", ensuring a consistent basis for salary analysis.
- The conditions use **.str.contains() with na=False** to handle missing values safely.

Removing Irrelevant Columns

- This step removes columns considered irrelevant, non-predictive, or intention-based (like tool preferences or browsing habits) using df.drop() with a filtered list.
- It ensures only meaningful, modeling-relevant features are retained for analysis.

Dropping Columns with Excessive Missing Values - 70%

- We dropped any column where 70% or more of its values are missing, using a calculated threshold based on the dataset length.
- If more than 70% of a column is missing, imputing or interpreting it becomes unreliable and introduces noise.

Dropping rows with more than 50% values missing

- We dropped rows with more than 50% missing values which often lack enough information for accurate imputation or analysis.
- This threshold is a common practice to maintain data integrity without being too aggressive in data loss.

Checking for Duplicate Columns

- This step checks for duplicate column names using .duplicated() on df.columns to ensure no redundancy or confusion during analysis or modeling. The result confirms that no duplicate columns exist in the dataset, maintaining data integrity.

Standardizing Age Group Format

- This step standardizes the Age column by removing the extra text " years old" using .str.replace() and .str.strip(), leaving clean age ranges like '25-34'.

Standardizing EdLevel Values

- This step standardizes the EdLevel column by removing the text inside parentheses using .str.replace() with a regex pattern, leaving clean values like 'Bachelor’s degree'.
- It ensures consistent and simplified education labels for easier grouping, filtering, and analysis.

Convert OrgSize Ranges to Numeric Midpoints

- This step standardizes the OrgSize column by removing the word "employees" and replacing " to " with "-" using .str.replace() and .str.strip(), resulting in a cleaner format like '100-499'.
- This transformation helps normalize company size ranges for downstream processing such as converting to numeric midpoints or group-based analysis

Standardizing YearsCode and YearsCodePro

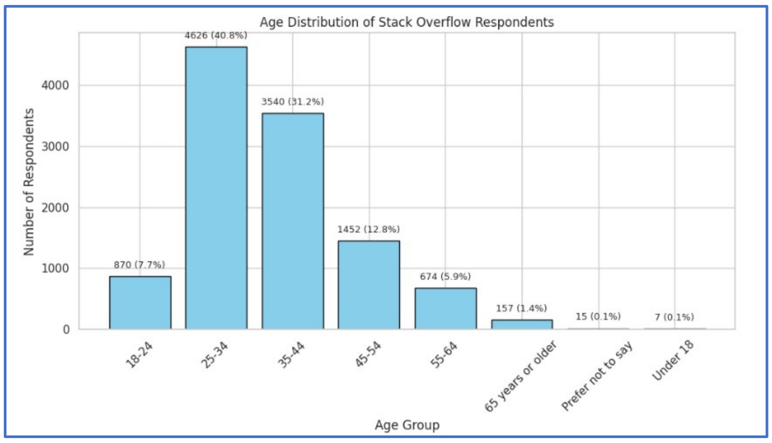
- This step standardizes the YearsCode and YearsCodePro columns by replacing special string values like "Less than 1 year" and "More than 50 years" *using a mapping dictionary and .replace().*
- It then converts all values to numeric *format with pd.to_numeric(errors='coerce'*) to enable accurate analysis and plotting of coding experience.

Standardizing Frequency_1, Frequency_2, and Frequency_3

- This step cleans the Frequency_1, Frequency_2, and Frequency_3 columns by removing the text " times a week" using .str.replace(), leaving only the numeric frequency ranges like '1-2', '6-10', etc.
- It ensures uniform string values for easier parsing, grouping, or conversion to numeric bins later.

Exploratory Data Analysis

Age Distribution of Stack Overflow Respondents



Interpretation:

Dominant Age Group: The 25–34 age group is the largest demographic among Stack Overflow respondents, with 4,626 people, making up 40.8% of the total.

Young Professionals Make Up the Majority: Combined, the 18–24 and 25–34 age groups account for 48.5% of respondents, suggesting the platform is heavily used by younger developers or those early in their careers.

Mid-career Representation: The 35–44 age group follows with 31.2% (3,540 respondents), indicating a solid presence of experienced professionals as well.

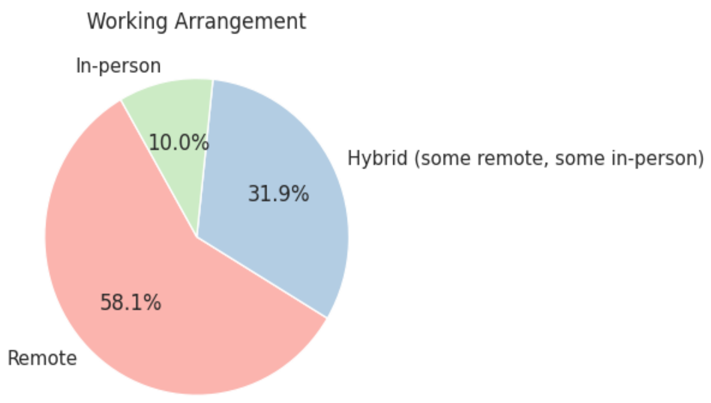
Sharp Decline in Older Age Groups: There's a steep drop in representation after age 44:

- 45–54: 12.8%
- 55–64: 5.9%
- 65+: Only 1.4%

This shows that older demographics are much less active or represented on Stack Overflow.

Minimal Responses from Youth and Non-disclosure: Very few respondents are under 18 (0.1%) or prefer not to disclose their age (0.1%), suggesting age transparency is common.

Working Arrangement Distribution



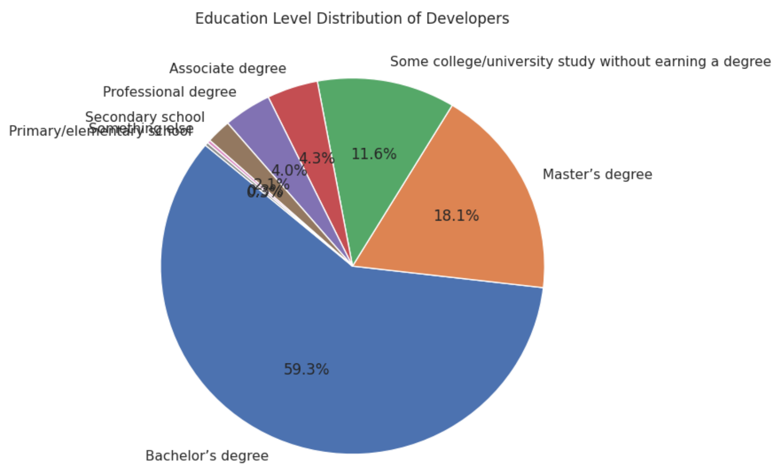
Interpretation:

Most Stack Overflow respondents work remotely, highlighting a strong shift toward remote work in the tech industry. Hybrid models are also common, while fully in-person roles are now the minority. This suggests remote-friendly roles are the norm among developers and tech professionals post-pandemic.

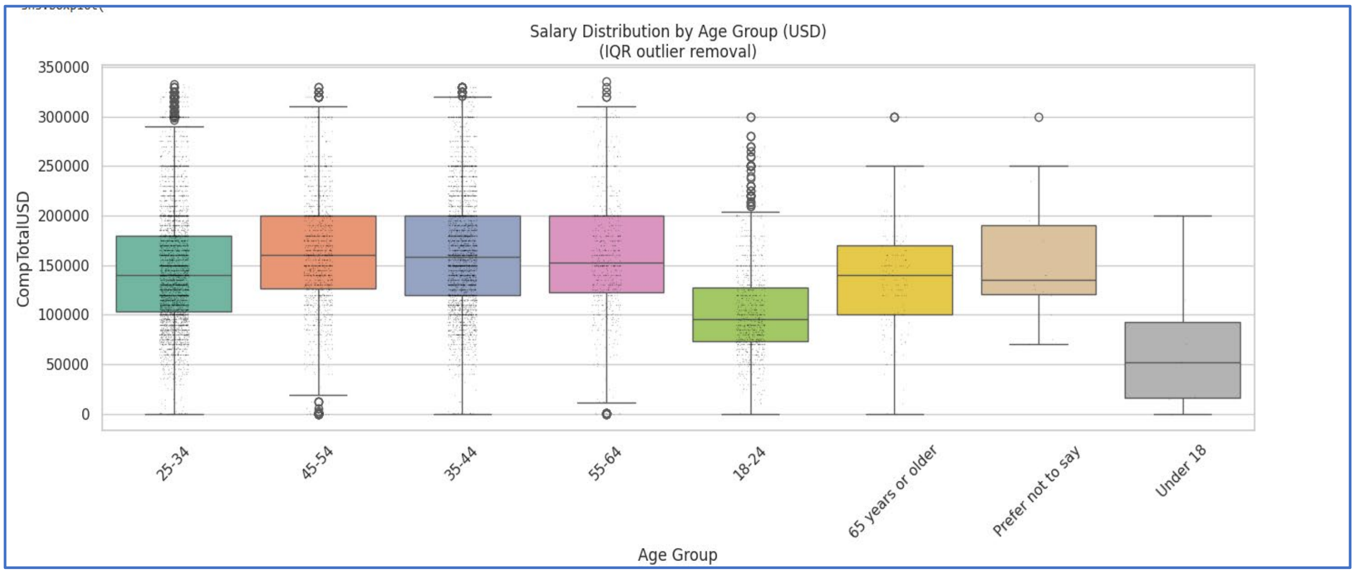
Education Level Distribution of Developers

Interpretation:

Most developers hold formal post-secondary education, with nearly 60% holding a bachelor's degree and an additional 18% holding a master's degree. This highlights that traditional academic paths remain the most common route into the developer profession. However, the presence of developers with non-degree education (including self-taught and partial university), making up over 17%, suggests there is still considerable room for alternative pathways into tech.



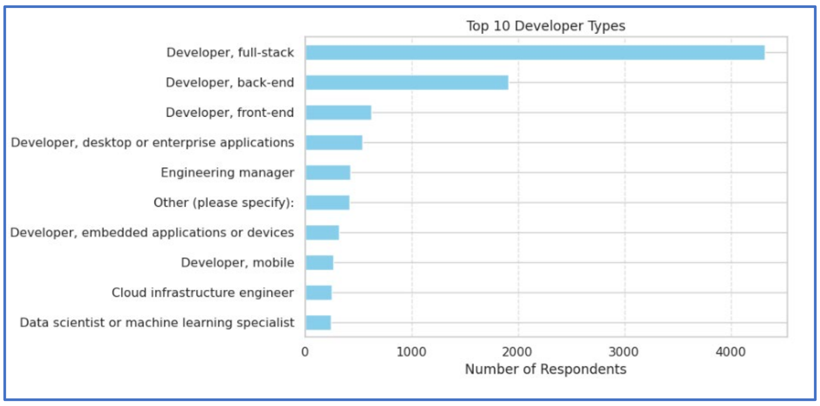
Understanding the Salary Distribution by Age Group



Interpretation:

Salaries tend to increase with age and experience up to a point, with the **35–44** and **45–54** age groups showing the highest median and upper salary ranges. The **18–24** and **Under 18** groups have noticeably lower median salaries, reflecting entry-level positions or internships. Interestingly, the **65+** and **prefer not to say** groups also report relatively competitive salaries, possibly due to senior consulting or niche roles. Overall, the chart underscores a correlation between age and earning potential, with mid-career developers earning the most.

Top 10 Developer Types



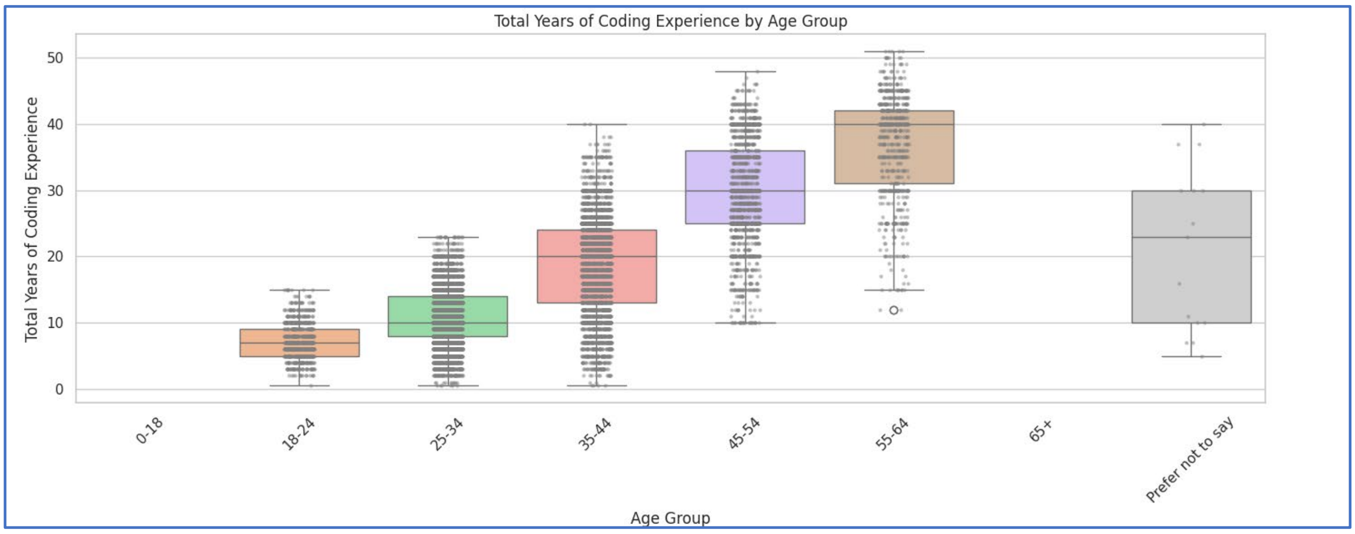
Interpretation:

Full-stack development dominates the profession, indicating high demand for developers with end-to-end skills.

Back-end and front-end specializations also form a large portion of the workforce, but niche and emerging roles like data science, mobile, and cloud engineering still lag in numbers.

This suggests that while full-stack skills are widespread and sought after, there may be growth opportunities in specialized or underrepresented domains.

Total Years of Coding Experience by Age Group



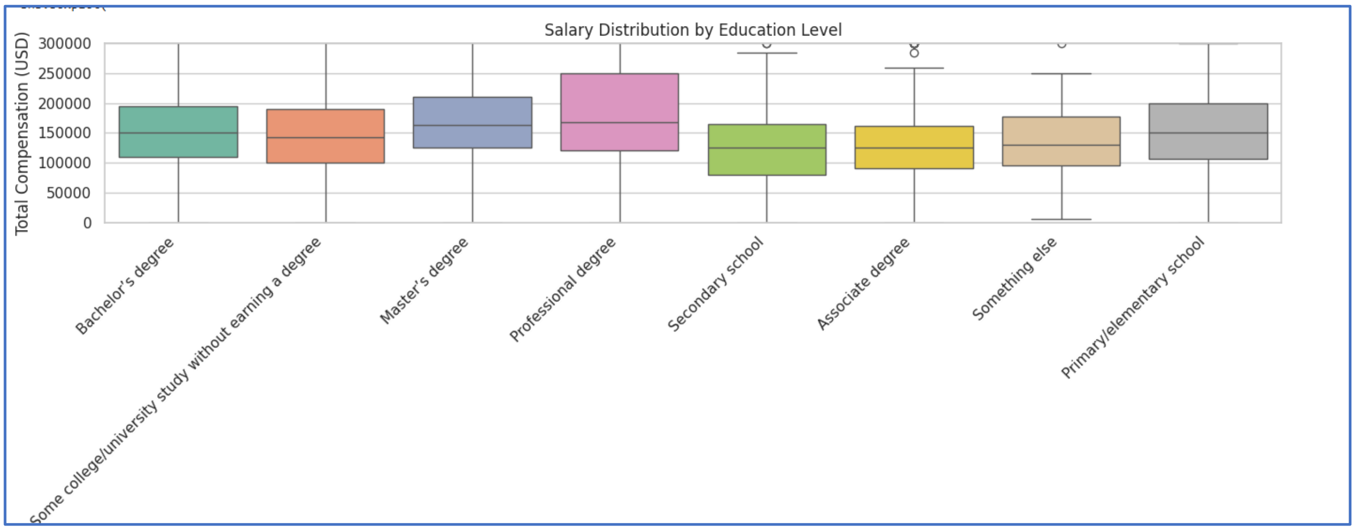
Interpretation:

- There is a strong positive correlation between age and coding experience, especially up to the 55–64 group.
- Median experience increases steadily from 18–24 through 55–64.
- The 65+ and 'Prefer not to say' groups show more variance but fewer data points.

The boxplots are consistent with expected career progression: older developers typically have more experience.

Outliers (small dots) show that some people in younger age brackets already have considerable experience, and vice versa.

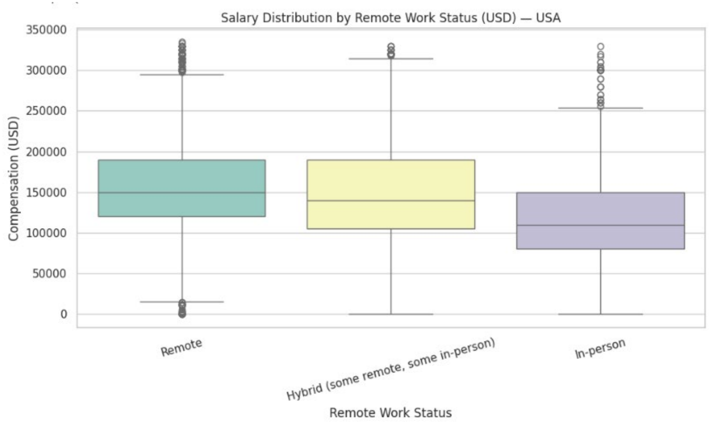
Education Level vs. Salary



Interpretation:

- Higher education generally correlates with higher compensation, though there are exceptions.
- Some lower education levels (e.g., primary school, something else) have high variance, likely due to individuals with exceptional skills, seniority, or entrepreneurship.
- Wide box and whisker ranges in all groups suggest salary is influenced by factors beyond education (e.g., role, experience, location).

Salary vs. Remote Work Status



Interpretation:

Median Salary:

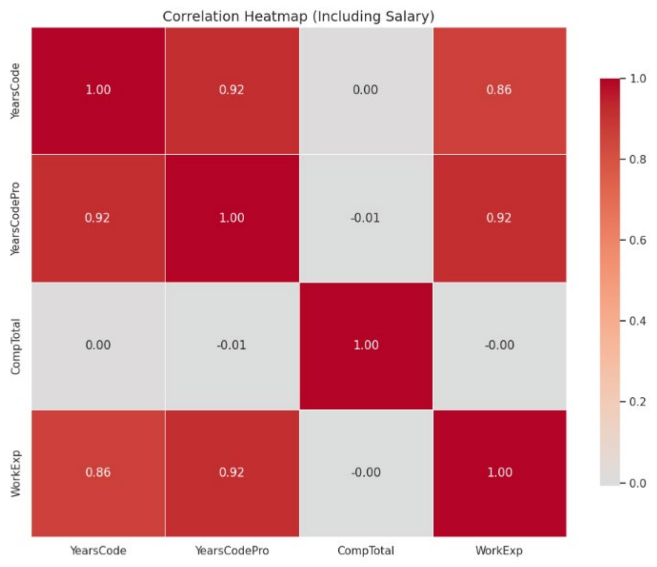
Remote and Hybrid workers both have higher median salaries than In-person workers.
The median for Remote and Hybrid appears around $140k–$150k, while In-person is closer to $100k–$110k.

Spread (IQR - Interquartile Range):

Remote and Hybrid jobs show wider salary ranges, indicating greater variability.
In-person roles are more compressed, suggesting more consistent but generally lower salaries.

Outliers:
All groups show outliers at the high end, but Remote and Hybrid include more extreme high earners (some near or over $300k).
The lower end of all three includes some near-zero salaries, possibly due to students, part-timers, or data entry errors.

Heatmap of Salary Correlation (numeric fields)



Interpretation:

Strong positive correlations between experience variables:

YearsCode, YearsCodePro, and WorkExp are highly correlated (0.86 to 0.92).

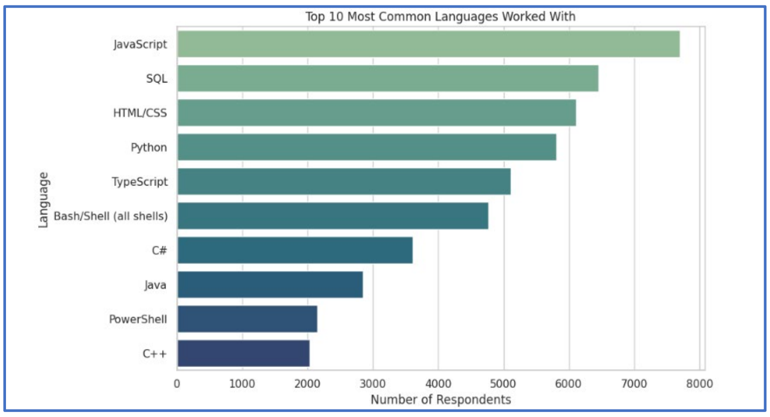
These features are redundant, meaning they capture overlapping information.

Near-zero correlation with CompTotal:

CompTotal (salary) has no linear correlation with any of the experience-related variables:

YearsCode: 0.00
YearsCodePro: -0.01
WorkExp: ~0.00

Most Common Languages



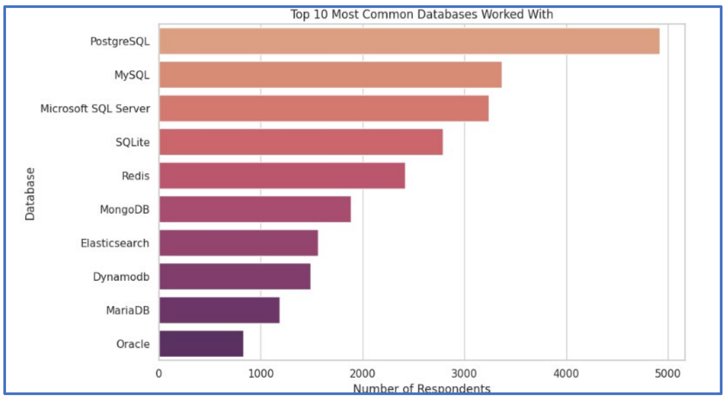
Interpretation:

JavaScript is the most commonly used language among U.S. respondents, with nearly 8,000 developers reporting working with it.

Other widely used languages include SQL, HTML/CSS, and Python, each with over 5,000 users.

Languages like C++, PowerShell, and Java are still popular but have smaller user bases (around 2,000–3,000 respondents). This reflects the dominance of web development and scripting tools in the U.S. developer landscape.

Top 10 Most Common Databases Worked With



Interpretation:

PostgreSQL is the most popular database, with close to 5,000 respondents using it.

MySQL and Microsoft SQL Server follow closely, showing strong usage in both open-source and enterprise environments.

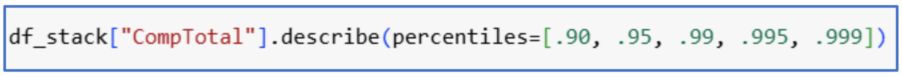
SQLite, Redis, and MongoDB are also commonly used, reflecting a mix of lightweight, in-memory, and NoSQL solutions.

Less common but still notable are Elasticsearch, DynamoDB, MariaDB, and Oracle, as they are often used in specialized or enterprise contexts.

Handling Target Outliers

Addressing Outliers in CompTotal (Target Variable) by using below :

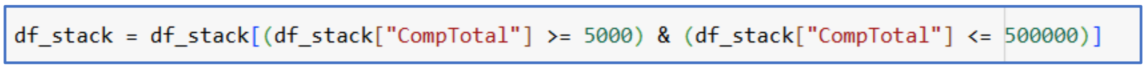
The *describe()* function with high percentiles (.90 to .999) is used to examine the upper distribution of the target variable CompTotal, helping identify extreme salary values that may skew model training.



```
df_stack["CompTotal"].describe(percentiles=[.90, .95, .99, .995, .999])
```

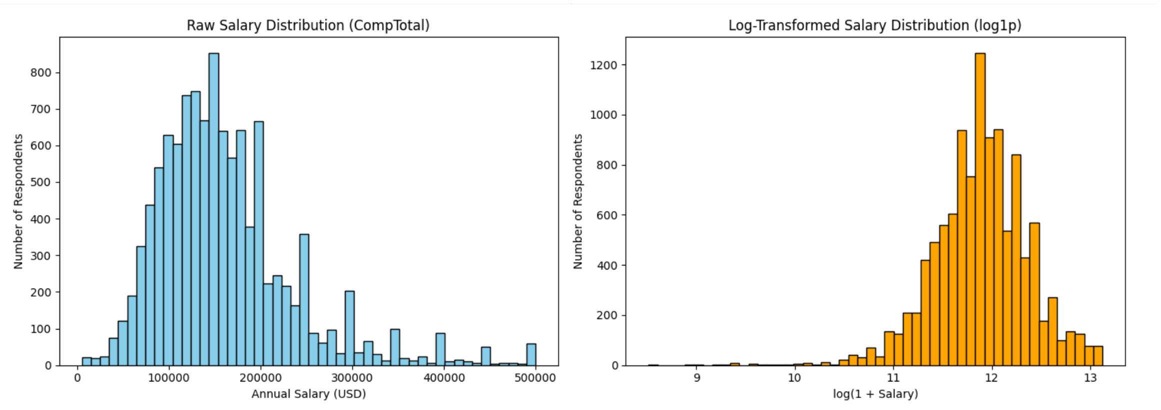
Outlier Removal:

To reduce the influence of unrealistic salaries, only data points with CompTotal between $5,000 and $500,000 are retained. This helps filter out noise and ensures more stable regression performance.



```
df_stack = df_stack[(df_stack["CompTotal"] >= 5000) & (df_stack["CompTotal"] <= 500000)]
```

Compare skewness of raw and log-transformed salary values:



The left plot shows the skewed distribution of raw `CompTotal` values.
The right plot shows the normalized shape after applying `log1p()`.
To evaluate which version is better for modeling, we plan to train models using both the raw and log-transformed targets.
This comparison will help us determine whether the log-transformed target leads to more stable and accurate predictions.

Feature Engineering

Our cleaned dataset contains **11,111 rows and 188 features**. In preparing the dataset for salary prediction, we applied one-hot encoding only to a selected set of categorical features.

Prior to encoding, we conducted a cardinality check across all object-type columns to assess the number of unique categories in each feature. This allowed us to distinguish between low-cardinality features, such as EdLevel, OrgSize, and RemoteWork, which were suitable for direct one-hot encoding, and high-cardinality features, such as LanguageHaveWorkedWith and ToolsTechHaveWorkedWith, which required alternative handling due to their complexity and sparsity.

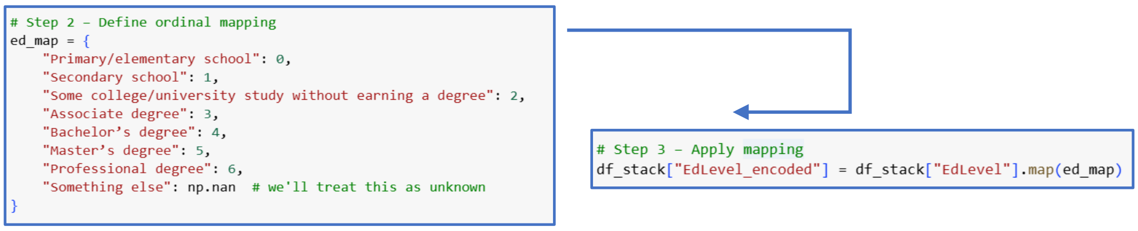
Rather than encoding all categorical columns (many of which had dozens of values or weak links to salary), we focused only on the most relevant ones. *This helped us reduce noise, keep the dataset manageable, and avoid unnecessary complexity that could affect model performance*. By targeting features that truly influence pay, we aimed to build a more efficient and accurate prediction model.

Encoding EdLevel

Rather than one-hot encoding EdLevel, we chose to apply ordinal mapping, assigning increasing numerical values to education levels based on their progression (e.g., from "Primary school" to "Doctoral degree"). This method reflects the natural order of education and preserves that structure for the model to learn from.

We also cleaned the column by stripping whitespace and handling missing or ambiguous values (e.g., "Something else"). This approach kept the data compact and interpretable, while still capturing the educational progression that may influence salary.

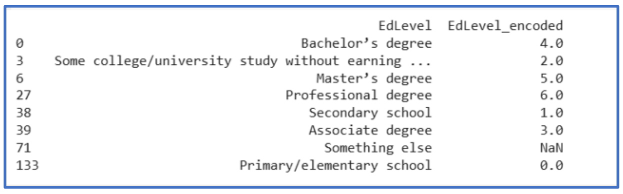
Ordinal mapping:



```
# Step 2 – Define ordinal mapping
ed_map = {
    "Primary/elementary school": 0,
    "Secondary school": 1,
    "Some college/university study without earning a degree": 2,
    "Associate degree": 3,
    "Bachelor’s degree": 4,
    "Master’s degree": 5,
    "Professional degree": 6,
    "Something else": np.nan  # we'll treat this as unknown
}
```

```
# Step 3 – Apply mapping
df_stack["EdLevel_encoded"] = df_stack["EdLevel"].map(ed_map)
```

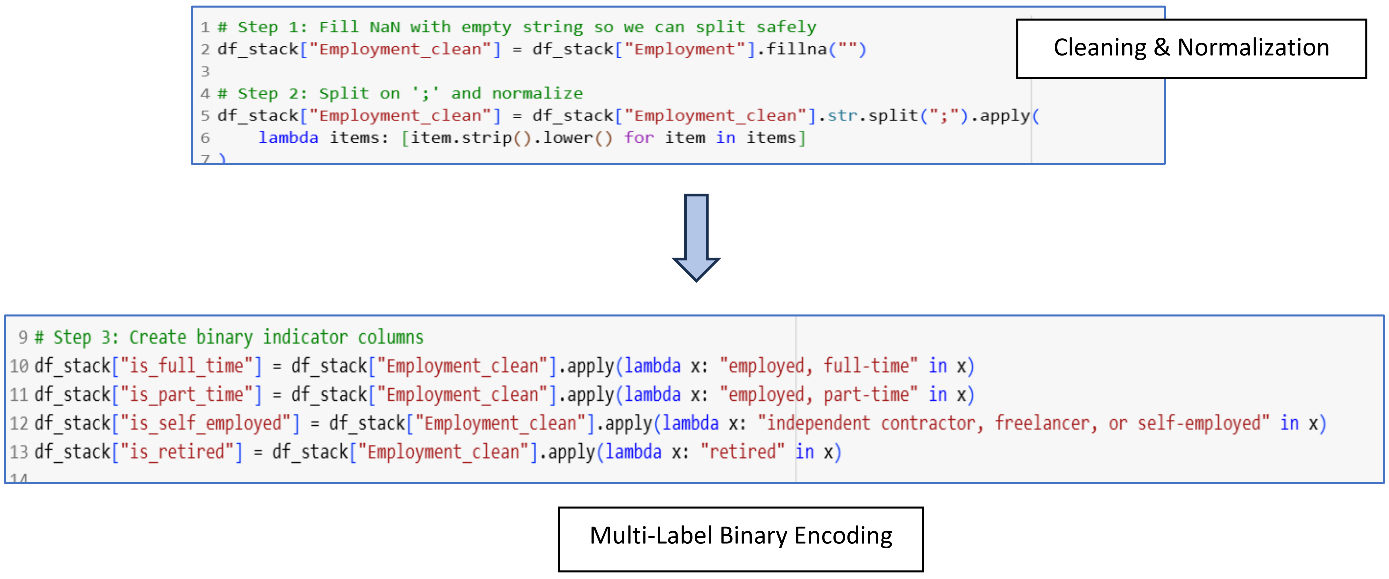
Output:



```
                                              EdLevel  EdLevel_encoded
0                                   Bachelor’s degree              4.0
3    Some college/university study without earning ...              2.0
6                                     Master’s degree              5.0
27                                Professional degree              6.0
38                                   Secondary school              1.0
39                                   Associate degree              3.0
71                                     Something else              NaN
133                         Primary/elementary school              0.0
```

Encoding Employment Status

The Employment column in the dataset allowed multiple selections (e.g., “Employed full-time; Self-employed”), so we needed a way to handle and represent these multi-label responses in a machine-learning-friendly format. To handle this, we applied the following steps:



```
# Step 1: Fill NaN with empty string so we can split safely
df_stack["Employment_clean"] = df_stack["Employment"].fillna("")

# Step 2: Split on ';' and normalize
df_stack["Employment_clean"] = df_stack["Employment_clean"].str.split(";").apply(
    lambda items: [item.strip().lower() for item in items]
)
```

Cleaning & Normalization

```
# Step 3: Create binary indicator columns
df_stack["is_full_time"] = df_stack["Employment_clean"].apply(lambda x: "employed, full-time" in x)
df_stack["is_part_time"] = df_stack["Employment_clean"].apply(lambda x: "employed, part-time" in x)
df_stack["is_self_employed"] = df_stack["Employment_clean"].apply(lambda x: "independent contractor, freelancer, or self-employed" in x)
df_stack["is_retired"] = df_stack["Employment_clean"].apply(lambda x: "retired" in x)
```

Multi-Label Binary Encoding

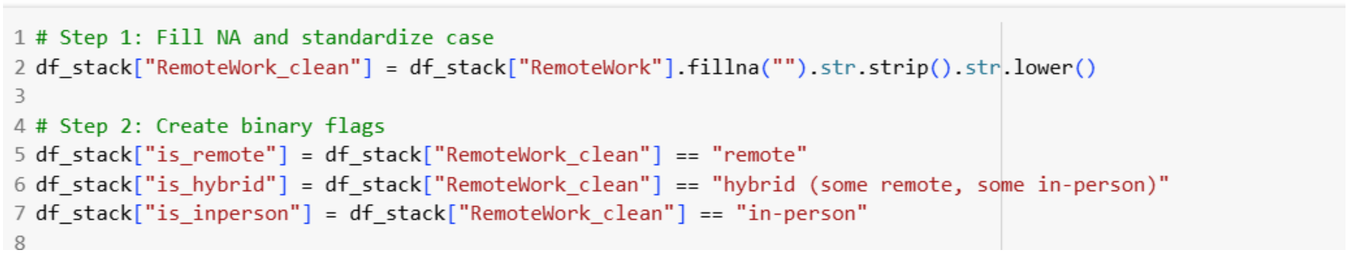
Encoding Remote Work Preferences

The RemoteWork column includes three types of responses: ***Remote, Hybrid, and In-person***. Since these responses are categorical and mutually exclusive, we transformed them into binary flags to make them usable in modeling. To handle this, we applied the following steps:

Cleaning and Standardization - We filled missing values with empty strings and normalized the text by converting all entries to lowercase and removing whitespace.

Binary Feature Creation - We created three new binary columns:

- is_remote: True if the respondent works fully remotely.
- is_hybrid: True if the respondent works partly remotely and partly in person.
- is_inperson: True if the respondent works entirely in person.



```python
# Step 1: Fill NA and standardize case
df_stack["RemoteWork_clean"] = df_stack["RemoteWork"].fillna("").str.strip().str.lower()

# Step 2: Create binary flags
df_stack["is_remote"] = df_stack["RemoteWork_clean"] == "remote"
df_stack["is_hybrid"] = df_stack["RemoteWork_clean"] == "hybrid (some remote, some in-person)"
df_stack["is_inperson"] = df_stack["RemoteWork_clean"] == "in-person"

```

Encoding Developer Roles (DevType)

The DevType column in the Stack Overflow dataset contains multiple roles per respondent (e.g., "Developer, full-stack; Developer, back-end"), making it a multi-label categorical feature. To handle this, we applied the following steps:

Cleaning and Splitting

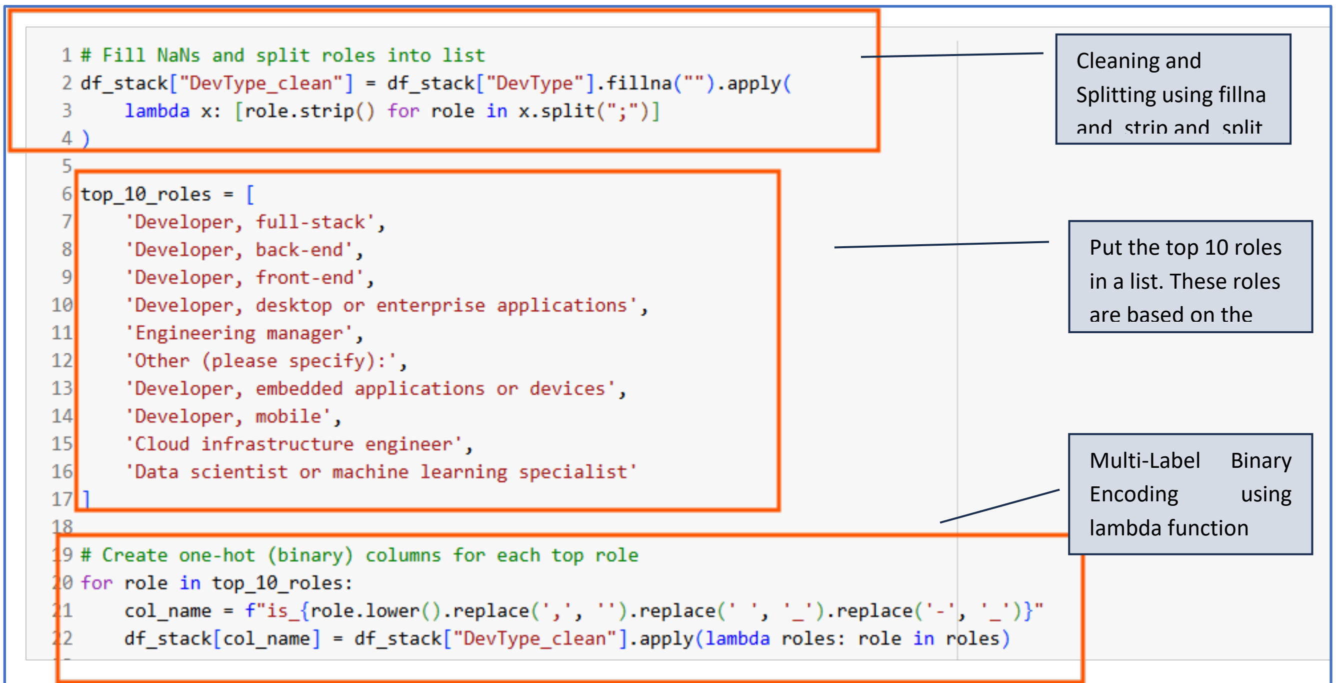
- We filled missing values with empty strings and split the entries on semicolons to extract individual roles. Each entry was also normalized (stripped and lowercased) to ensure consistency.

Top 10 Role Selection

- Since the dataset includes over 20 unique roles, we selected the top 10 most common developer roles based on EDA insights. This reduced dimensionality while still preserving the most impactful data.

Multi-Label Binary Encoding

- We performed manual multi-label one-hot encoding for the DevType column by first splitting roles into lists, then creating binary columns for the top 10 most frequent roles identified during EDA. This allowed us to capture relevant role information while keeping the feature space compact.



```python
# Fill NaNs and split roles into list
df_stack["DevType_clean"] = df_stack["DevType"].fillna("").apply(
    lambda x: [role.strip() for role in x.split(";")]
)

top_10_roles = [
    'Developer, full-stack',
    'Developer, back-end',
    'Developer, front-end',
    'Developer, desktop or enterprise applications',
    'Engineering manager',
    'Other (please specify):',
    'Developer, embedded applications or devices',
    'Developer, mobile',
    'Cloud infrastructure engineer',
    'Data scientist or machine learning specialist'
]

# Create one-hot (binary) columns for each top role
for role in top_10_roles:
    col_name = f"is_{role.lower().replace(',', '').replace(' ', '_').replace('-', '_')}"
    df_stack[col_name] = df_stack["DevType_clean"].apply(lambda roles: role in roles)
```

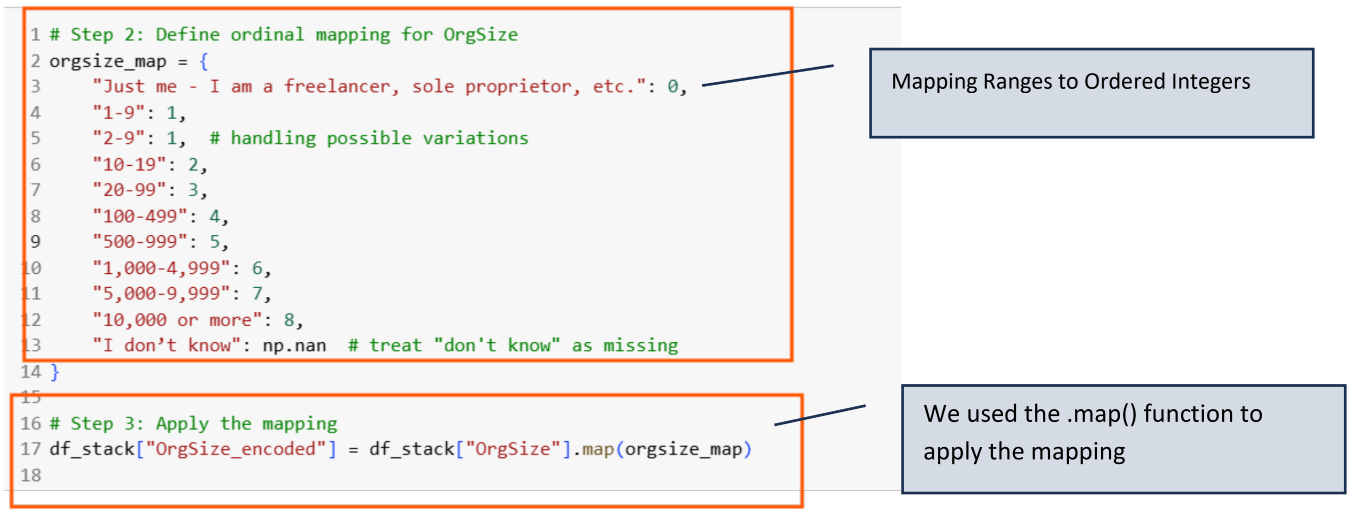
Cleaning and Splitting using fillna and strip and split
Put the top 10 roles in a list. These roles are based on the
Multi-Label Binary Encoding using lambda function

Encoding Organization Size

The OrgSize column indicates the size of the respondent's organization, using predefined size ranges. Since these ranges follow a natural increasing order, we applied ordinal encoding to convert them into meaningful numeric values.

Mapping Ranges to Ordered Integers - We created a custom dictionary to map each size category (e.g., "1-9", "100-499") to an increasing integer, starting from 0 for the smallest (freelancers) up to 8 for the largest (10,000 or more). The category "I don't know" was treated as missing (NaN).

Applying the Mapping - We used the .map() function to apply this mapping directly to the OrgSize column and stored the result in a new column called OrgSize_encoded.



```
# Step 2: Define ordinal mapping for OrgSize
orgsize_map = {
    "Just me - I am a freelancer, sole proprietor, etc.": 0,
    "1-9": 1,
    "2-9": 1,  # handling possible variations
    "10-19": 2,
    "20-99": 3,
    "100-499": 4,
    "500-999": 5,
    "1,000-4,999": 6,
    "5,000-9,999": 7,
    "10,000 or more": 8,
    "I don't know": np.nan  # treat "don't know" as missing
}

# Step 3: Apply the mapping
df_stack["OrgSize_encoded"] = df_stack["OrgSize"].map(orgsize_map)

```

Mapping Ranges to Ordered Integers
We used the .map() function to apply the mapping

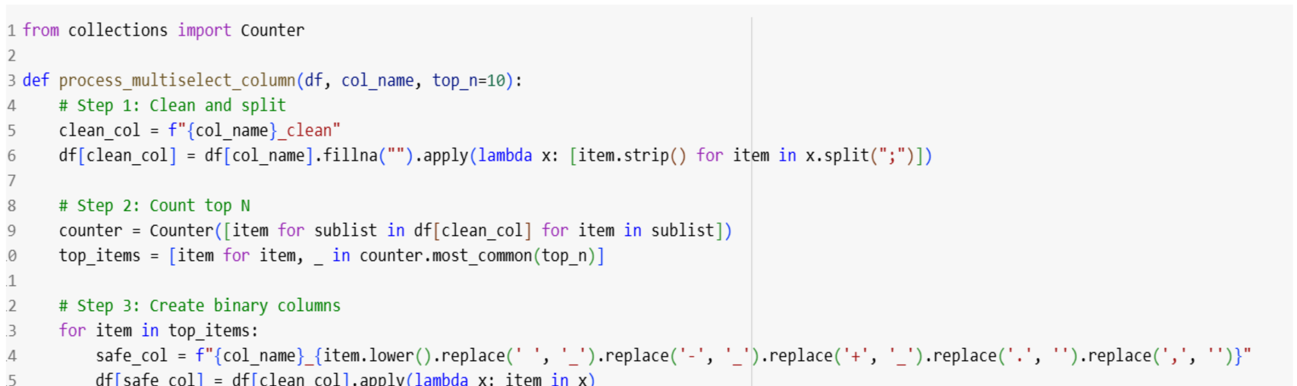
One-Hot Encoding Multi-Select Tech Stack Columns

The columns such as LanguageHaveWorkedWith, DatabaseHaveWorkedWith, and related technologies contain multiselect categorical data, where respondents could select multiple tools or platforms they have used. While we did not include full visual EDA for each of these columns, we used domain knowledge and general industry trends to guide our preprocessing decisions.

For example, we know from industry experience and prior Stack Overflow reports that certain technologies (e.g., SQL, Python, PostgreSQL, Docker) consistently rank among the most used. We used this context to extract and encode only the top N most relevant tools from each category using a consistent processing function.

This approach allowed us to:

- Focus on technologies with high practical importance or adoption
- Avoid over-encoding rare or outdated tools
- Keep the feature set manageable and relevant for modelling



```
from collections import Counter

def process_multiselect_column(df, col_name, top_n=10):
    # Step 1: Clean and split
    clean_col = f"{col_name}_clean"
    df[clean_col] = df[col_name].fillna("").apply(lambda x: [item.strip() for item in x.split(";")])

    # Step 2: Count top N
    counter = Counter([item for sublist in df[clean_col] for item in sublist])
    top_items = [item for item, _ in counter.most_common(top_n)]

    # Step 3: Create binary columns
    for item in top_items:
        safe_col = f"{col_name}_{item.lower().replace(' ', '_').replace('-', '_').replace('+', '_').replace('.', '').replace(',', '')}"
        df[safe_col] = df[clean_col].apply(lambda x: item in x)
```

Step 1: Clean and Split

- Replaces NaN with empty strings
- Splits the string on ; (since multiple values are semicolon-separated)
- Strips whitespace from each item
- Result: a list of selected tech tools per respondent

Step 2: Count Top N Items

- Flattens all responses into a single list of tech items
- Uses Counter from Python's collections module to get the **top N most frequent tools**
- Users control top_n (default = 10)

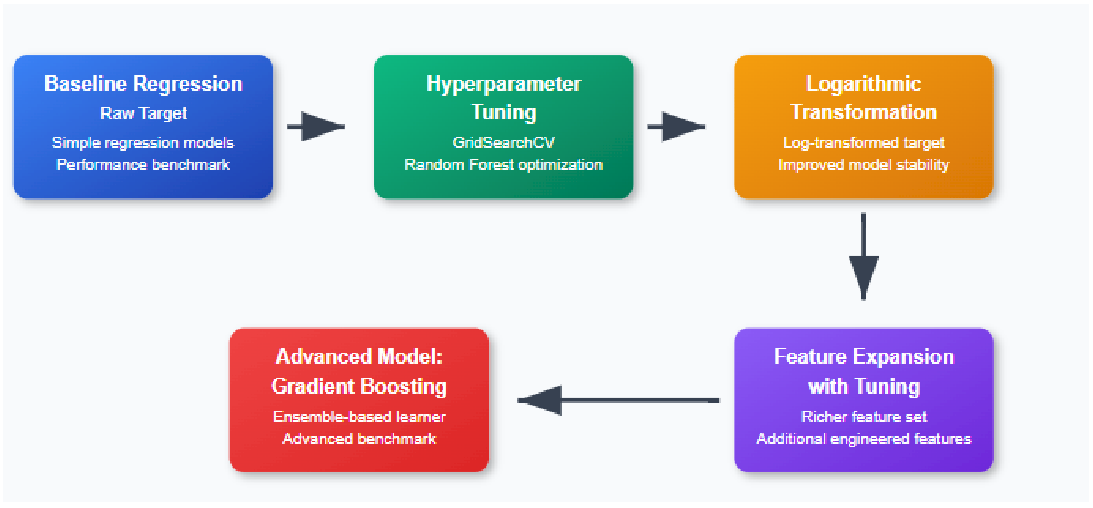
Step 3: Create Binary Columns

For each of the top N tools:

- Creates a new column like LanguageHaveWorkedWith_python
- Sets it to True or False depending on whether the respondent used that tool

Regression Modeling Roadmap: From Baseline to Boosted Models

To predict developer compensation (CompTotal), we implemented a staged regression modeling pipeline, starting with basic models and progressing to more complex, tuned variants. Our process involved five main steps:



Justification for Model Choices

We selected **Random Forest** as one of our core models due to its strong performance on tabular, structured datasets like the Stack Overflow survey data. It offers several advantages:

- Handles non-linear relationships and complex feature interactions well
- Robust to outliers and noise in the data
- Performs automatic feature selection by assessing feature importance

- Low risk of overfitting due to averaging across multiple decision trees
- No need for scaling or strict assumptions about data distribution

Random Forest served as an effective baseline for tuned ensemble models and provided interpretability through feature importance scores.

We introduced **Gradient Boosting** as an advanced model due to its ability to incrementally correct errors made by previous models, resulting in higher accuracy when tuned well. It is particularly suitable because:

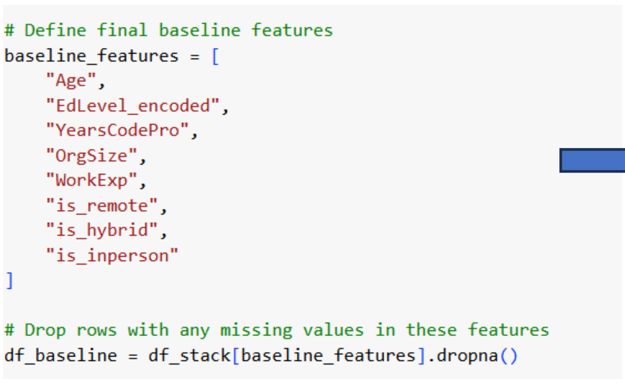
- It captures subtle patterns and non-linear relationships
- It often outperforms bagging methods (like Random Forest) when tuned properly
- It supports regularization techniques (e.g., learning rate, tree depth) to reduce overfitting
- It provides fine-grained control over model complexity and performance

Gradient Boosting is widely used in winning solutions for structured prediction tasks and offers a strong benchmark for high-performing models.

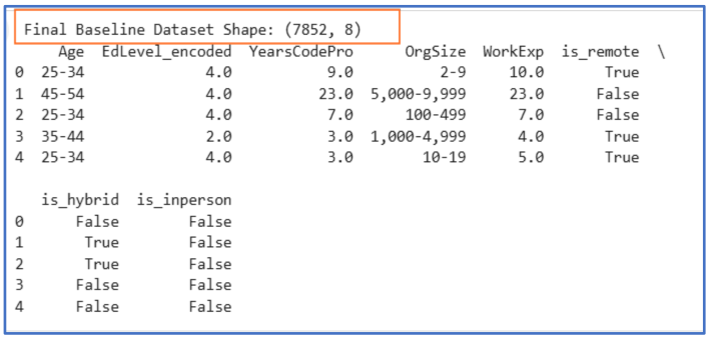
Baseline Regression with Raw Target (No Hyperparameter Tuning)

We began with simple regression models using the untransformed compensation values and a small set of baseline features. This established a performance benchmark for comparison.

Features Used:



```python
# Define final baseline features
baseline_features = [
    "Age",
    "EdLevel_encoded",
    "YearsCodePro",
    "OrgSize",
    "WorkExp",
    "is_remote",
    "is_hybrid",
    "is_inperson"
]

# Drop rows with any missing values in these features
df_baseline = df_stack[baseline_features].dropna()
```

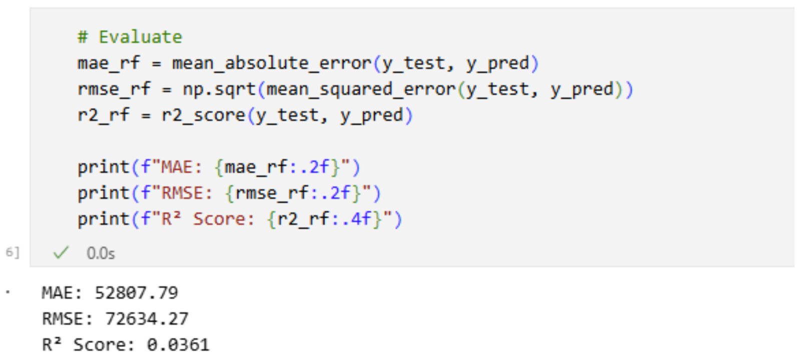


```
Final Baseline Dataset Shape: (7852, 8)
     Age  EdLevel_encoded  YearsCodePro      OrgSize  WorkExp  is_remote  \
0  25-34              4.0           9.0          2-9     10.0       True
1  45-54              4.0          23.0  5,000-9,999     23.0      False
2  25-34              4.0           7.0      100-499      7.0      False
3  35-44              2.0           3.0  1,000-4,999      4.0       True
4  25-34              4.0           3.0        10-19      5.0       True

   is_hybrid  is_inperson
0      False        False
1       True        False
2       True        False
3      False        False
4      False        False
```

For our baseline Random Forest regression model (using the raw CompTotal target), we selected a compact set of features that are:

- Numerical or ordinal (no further encoding required)
- Available early in the survey
- Likely to influence compensation based on domain knowledge and EDA

This baseline setup allows us to measure how much value is added when additional features (like tech stacks, developer roles, etc.) are incorporated later in the pipeline.



```python
# Evaluate
mae_rf = mean_absolute_error(y_test, y_pred)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred))
r2_rf = r2_score(y_test, y_pred)

print(f"MAE: {mae_rf:.2f}")
print(f"RMSE: {rmse_rf:.2f}")
print(f"R² Score: {r2_rf:.4f}")
```

```
6]  ✓ 0.0s

MAE: 52807.79
RMSE: 72634.27
R² Score: 0.0361
```

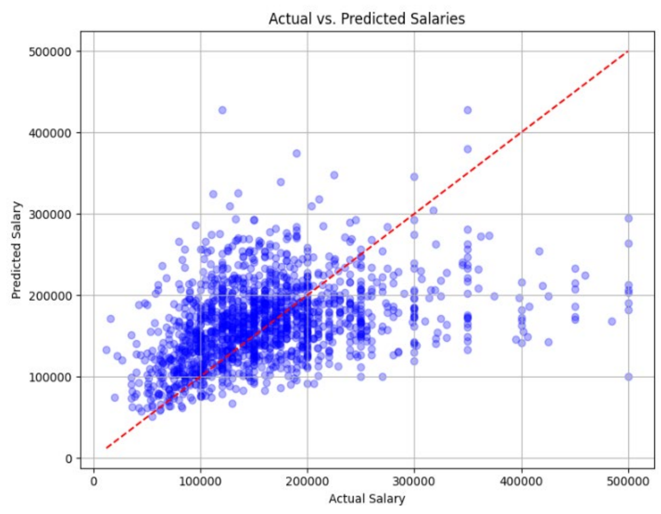
Model Evaluation (Raw Target, Baseline Features)

After training the Random Forest model on the baseline feature set with the raw CompTotal target, we evaluated performance using three standard regression metrics

MAE (Mean Absolute Error)
RMSE (Root Mean Squared Error)
R^2 Score (Coefficient of Determination)

Metric	Value	Interpretation
MAE (Mean Absolute Error)	52807.79	On average, the model's salary predictions are off by about $53,000.
RMSE (Root Mean Squared Error)	72634.27	A $72K error suggests that the model occasionally makes very large prediction errors, especially for high-salary earners, where it can under- or overestimate significantly
R^2 Score	0.0361	The model explains only about 3% of the variation in salaries. That's quite low, meaning other important factors (like job title, industry, or location) are probably missing from the data. It shows that the model is not very reliable at capturing what drives compensation.

Actual vs Predicted Salaries



Interpretation:

The model generally captures the upward trend, higher actual salaries tend to correspond with higher predictions.

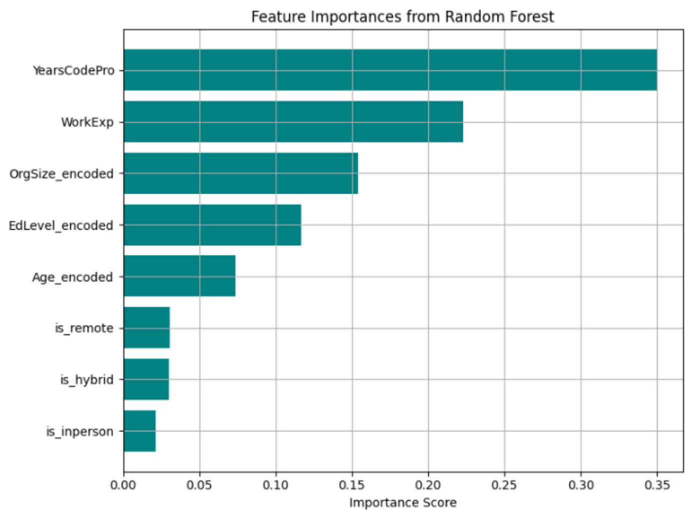
However, predictions are compressed toward the center, especially for higher salary ranges, indicating:

- Underestimation of high salaries
- Limited ability to model extreme values with the current feature set

The scatter's spread and deviation from the diagonal suggest there's room to improve accuracy, particularly for outlier compensation cases

This visualization supports the numerical metrics (MAE, RMSE, R^2) and highlights the need for further improvements, such as target transformation and feature expansion.

Feature Importance :



The chart shows how much each baseline feature contributed to the Random Forest model's salary predictions.

Years of professional coding experience (**YearsCodePro**) and overall work experience (**WorkExp**) were the most influential factors, strongly impacting predicted compensation.

Org size and education level also had moderate importance, while age played a smaller role.

Remote work status (is_remote, is_hybrid, is_inperson) had the least influence on predictions in the baseline model.

Experience-related features (especially YearsCodePro and WorkExp) were the most powerful predictors of salary in our baseline model. In contrast, remote work arrangement had minimal predictive value on its own.

Hyperparameter Tuning for Random Forest (Same Baseline Feature)

To improve the performance of our baseline Random Forest model, we applied hyperparameter tuning using a grid search strategy. The parameter grid included combinations of:

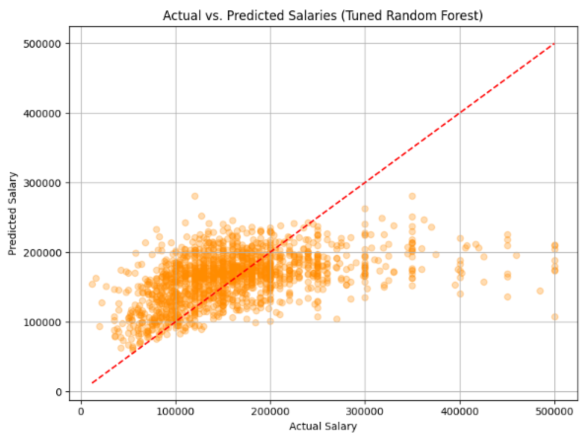
- *n_estimators:* Number of trees in the forest (100, 200)
- *max_depth:* Maximum depth of each tree (None = unlimited, or fixed at 10 or 20)
- *min_samples_split:* Minimum number of samples required to split an internal node (2 or 5)
- *min_samples_leaf:* Minimum number of samples required to be at a leaf node (1 or 2)
- *max_features:* Number of features to consider when looking for the best split ('sqrt', 'log2', or all features)

This grid allowed the model to explore a variety of complexity levels, from shallow and regularized trees to fully grown ones, and to optimize how features are selected at each decision node.

Model Evaluation:

Metric	Value	Interpretation
MAE	47662.42	On average, the model's salary predictions are off by around $48,000
RMSE	65957.22	This means the model still makes some large errors, especially for higher salaries.
R^2 Score	0.2052	The model explains about 21% of the variation in salaries using the input data.

Actual vs Predicted Salaries



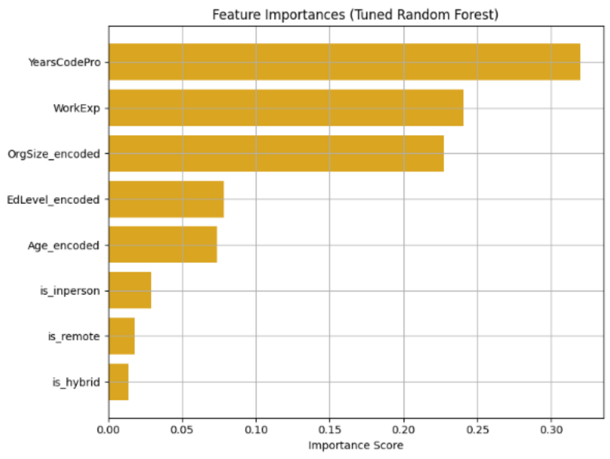
Interpretation:

Most points are clustered around the line, especially in the lower and mid salary ranges. This means the model is making reasonable predictions in these areas.

For higher salaries (right side), the dots fall below the red line. This suggests the model tends to underpredict high salaries.

Overall, the predictions follow the right trend but still leave room for improvement, especially at the higher end.

Feature Importance



Interpretation:

Years of professional coding experience (YearsCodePro) was by far the most important factor. The model uses it heavily to estimate salary.

Overall work experience (WorkExp) and organization size (OrgSize_encoded) also played major roles.

Education level and age had moderate influence, while the work format (remote, hybrid, in-person) had very little impact.
This suggests that experience and workplace context matter more for salary prediction than whether someone works remotely or not.

RandomForrest Regression with Logarithmic Transformation of Target (with Tuning)

To address the skewness in salary distribution, we applied a logarithmic transformation to the CompTotal target variable using **np.log1p()**. *This stabilizes variance and helps the model focus on relative differences rather than extreme salary outliers*.

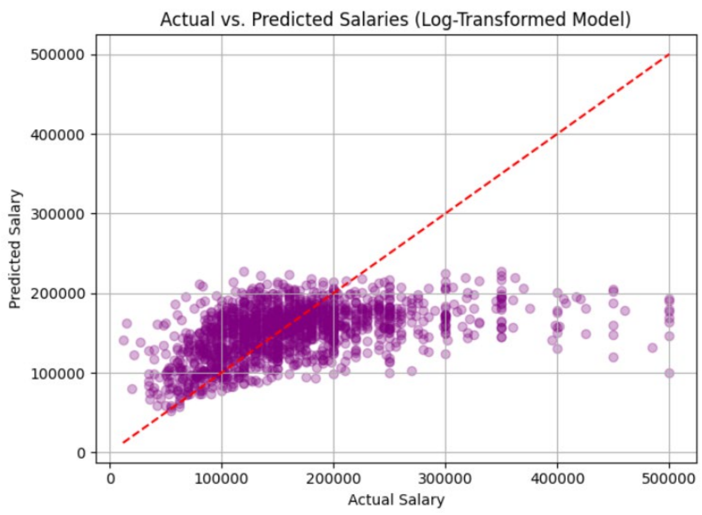
After model training and prediction in the log scale, we used np.expm1() to revert predictions and targets back to the original salary scale for evaluation.

```
# 1. Set up features and log-transformed target
X = df_baseline.copy()
y = df_stack.loc[df_baseline.index, 'CompTotal']
y_log = np.log1p(y)  # log1p handles 0 or small values safely
```

Model Evaluation:

Metric	Value	Interpretation
MAE	$ 46,055.74	On average, predictions are off by about $46K
RMSE	$ 66,231.88	The model still makes some big mistakes, especially for people on the high or low ends of the salary range.
R^2 Score	0.1985	The model explains only 19% of what affects salary. That means 81% of the variation in salaries is due to other factors that the model didn't capture (e.g., role, seniority, company, etc.).

Actual vs Predicted



Interpretation:

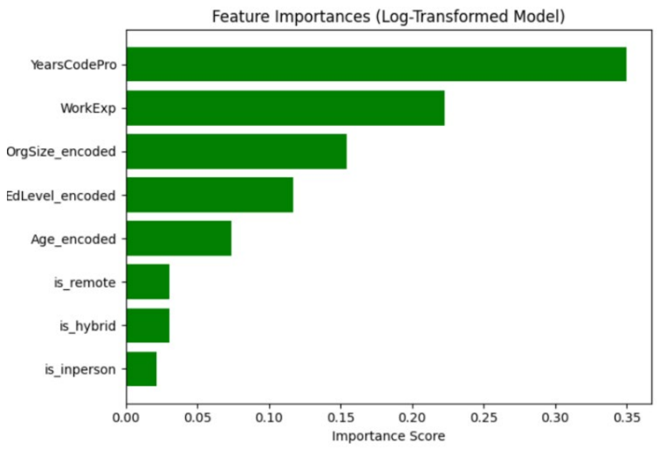
The predictions are more tightly clustered along the ideal line compared to the untransformed model, a sign of improved stability and better fit.

The model performs especially well for mid-range salaries, where most points are close to the red line.

Some underprediction is still visible for higher salaries, but overall, the spread is more consistent and less extreme than before.

This confirms that applying a log transformation to the target helped the model handle salary variability better and make more accurate predictions across most of the range.

Feature Importance:



Interpretation:

YearsCodePro remains the most important feature, contributing over 35% to the model's predictive power. This suggests that more professional coding experience strongly correlates with higher salaries.

WorkExp and OrgSize_encoded also contribute meaningfully, indicating that total work experience and company size affect salary levels.

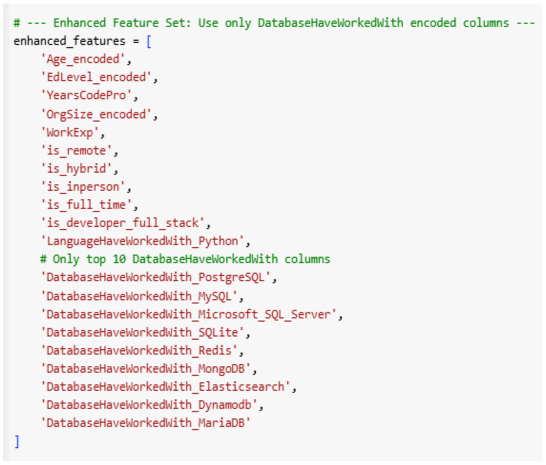
Education level (EdLevel_encoded) and Age have moderate impact, while remote/hybrid/in-person status play smaller roles.

Feature Expansion with Tuning

After establishing a tuned Random Forest model using a compact baseline feature set, we proceeded to evaluate whether adding more features could improve model performance. To improve performance, we expanded our feature set by adding key technology indicators, especially the top databases developers reported working with:

Examples include: DatabaseHaveWorkedWith_PostgreSQL, MySQL, MongoDB, SQLite, Redis, and others.

We then applied Random Forest Regressor with GridSearchCV for hyperparameter tuning to better capture non-linear relationships and improve prediction accuracy.

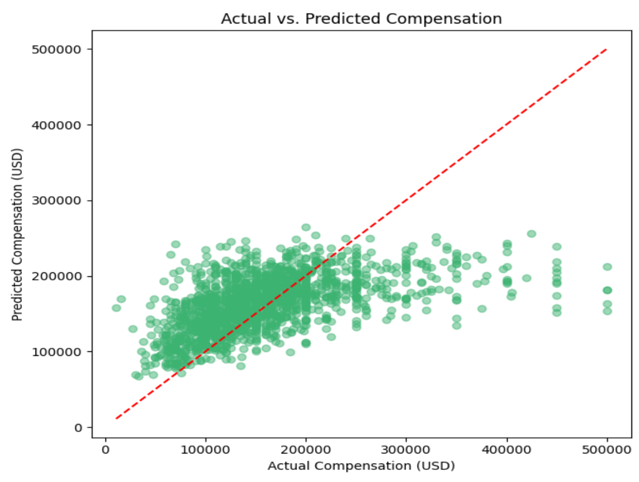


```
# --- Enhanced Feature Set: Use only DatabaseHaveWorkedWith encoded columns ---
enhanced_features = [
    'Age_encoded',
    'EdLevel_encoded',
    'YearsCodePro',
    'OrgSize_encoded',
    'WorkExp',
    'is_remote',
    'is_hybrid',
    'is_inperson',
    'is_full_time',
    'is_developer_full_stack',
    'LanguageHaveWorkedWith_Python',
    # Only top 10 DatabaseHaveWorkedWith columns
    'DatabaseHaveWorkedWith_PostgreSQL',
    'DatabaseHaveWorkedWith_MySQL',
    'DatabaseHaveWorkedWith_Microsoft_SQL_Server',
    'DatabaseHaveWorkedWith_SQLite',
    'DatabaseHaveWorkedWith_Redis',
    'DatabaseHaveWorkedWith_MongoDB',
    'DatabaseHaveWorkedWith_Elasticsearch',
    'DatabaseHaveWorkedWith_Dynamodb',
    'DatabaseHaveWorkedWith_MariaDB'
]
```

Model Evaluation:

Metric	Value	Interpretation
MAE	$ 44,702.47	On average, the model's predictions are off by more than $44K
RMSE	$ 62,722.40	It shows that the model still struggles especially with very high or very low salaries.
R2	0.2812	The model can explain about 28% of the salary differences between developers. That means it's picking up on some important factors, but most of what affects salary is still not captured.

Actual vs Predicted



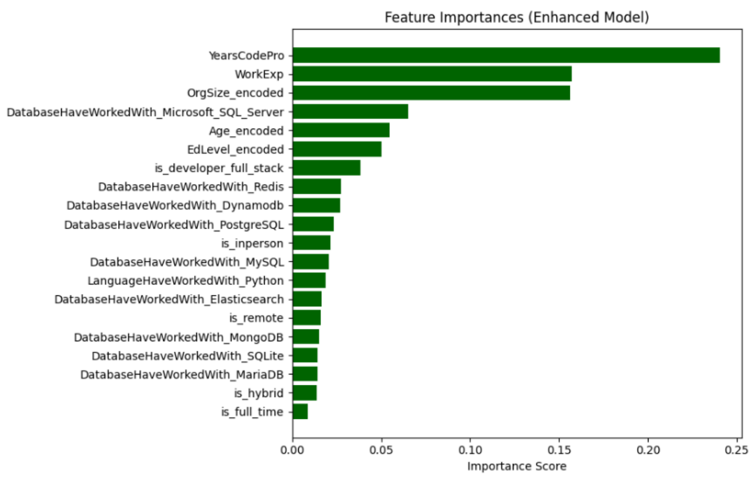
Interpretation:

Most predictions are clustered below the red line, especially for higher salaries.
This means the model tends to underpredict high salaries.

For mid-range salaries (around $100K–$200K), predictions are fairly close to actual values.
The model performs best in this range.

For very high salaries (>$300K), predictions are much lower than actual.
The model struggles to capture top-end compensation accurately.

Feature Importance:



Interpretation:

Top Predictors:

YearsCodePro (years coding professionally) is by far the most influential factor in predicting salary.

WorkExp (total work experience) and OrgSize_encoded (company size) are also strong indicators, showing that experience and organization context heavily influence pay.

Moderate Impact Features:

Technologies like DatabaseHaveWorkedWith_Microsoft_SQL_Server and Redis, as well as demographics like Age_encoded and EdLevel_encoded, have noticeable but smaller influence.

Being a full-stack developer (is_developer_full_stack) adds some predictive value.

Low Impact Features:

Some database tools (like SQLite, MongoDB, MariaDB) and work arrangement indicators (is_remote, is_hybrid, is_full_time) have minimal impact in this model.

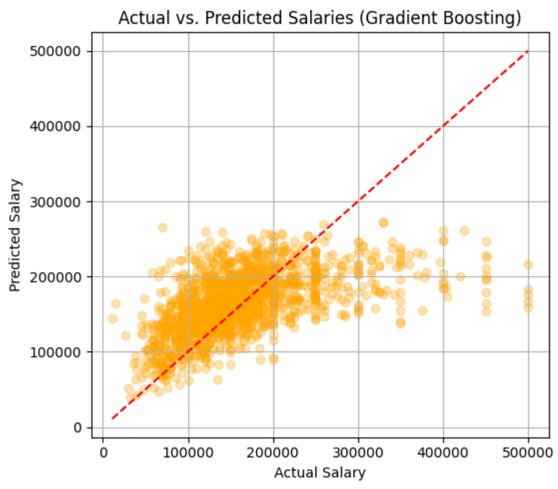
Gradient Boosting

To further improve prediction accuracy, we applied a Gradient Boosting Regressor (GBR) using the same enhanced feature set. Gradient Boosting builds an ensemble of decision trees sequentially, where each tree corrects the errors of the one before it. This makes it well-suited for capturing complex, non-linear relationships in structured data. By incorporating both developer demographics and technical experience, including database tools and employment type, GBR offers a more refined approach to modeling salary compared to baseline methods.

Model Evaluation:

Metric	Value	Interpretation
MAE	$ 44,496.89	On average, the model's salary predictions are off by around $44K, which is a moderate error given the wide range of tech salaries.
RMSE	$ 62,320.92	This shows that while many predictions are close, there are still some larger errors, especially for very high or very low salaries.
R2	0.2904	The model explains about 29% of the differences in developer salaries. This is an improvement over earlier models, suggesting the added features helped the model better understand what affects compensation

Actual vs Predicted :



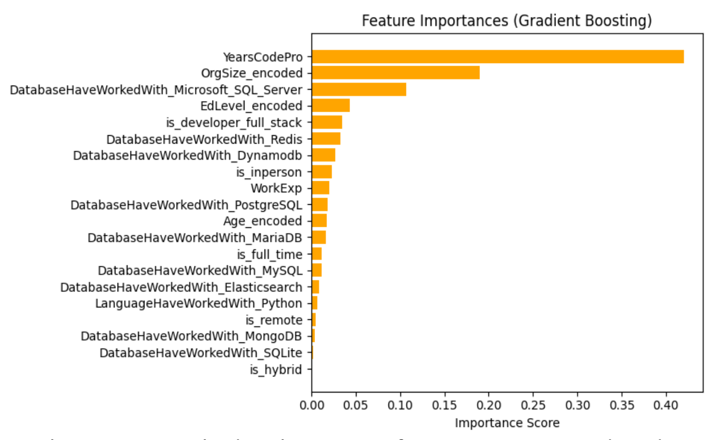
Interpretation:

Most points are clustered below the red line, indicating that the model tends to underestimate salaries, especially for higher earners.

The predictions are close for many mid-range salaries (around $100k–$200k), where the model performs best.

There's more spread and underprediction in the higher salary ranges (above $250k), which is common in salary data due to high variability and fewer examples.

Feature Importance:



Interpretation:

YearsCodePro (professional coding experience) was by far the most important predictor. The more experience, the more it influenced the predicted salary.

OrgSize_encoded (organization size) also played a strong role, likely because larger companies often offer higher pay.

DatabaseHaveWorkedWith_Microsoft_SQL_Server and EdLevel_encoded (education level) contributed noticeably, suggesting experience with enterprise tools and higher education matters.

Technical traits like full-stack developer and experience with databases like Redis and DynamoDB had moderate influence.

On the other hand, features like is_hybrid, is_remote, or working with SQLite had very low impact on salary prediction in this model.

Conclusion

Model	MAE	RMSE	R2
Gradient Boosting	**$44,496.89**	**$62,320.92**	**0.2904**
Feature Expansion with Tuning	$44,702.47	$62,722.40	0.2812
RandomForrest Regression with Logarithmic Transformation of Target	$46,055.74	$66,231.88	0.1985
Hyperparameter Tuning for Random Forest	$47,662.42	$65,957.22	0.2052
Baseline Random Forest	$52,807.79	$72,634.27	0.0361

Among all the models tested, Gradient Boosting delivered the best overall performance. It achieved the lowest prediction errors (MAE: ~$44K, RMSE: ~$62K) and the highest explanatory power ($R^2 = 0.2904$), capturing nearly 30% of the variation in developer salaries.

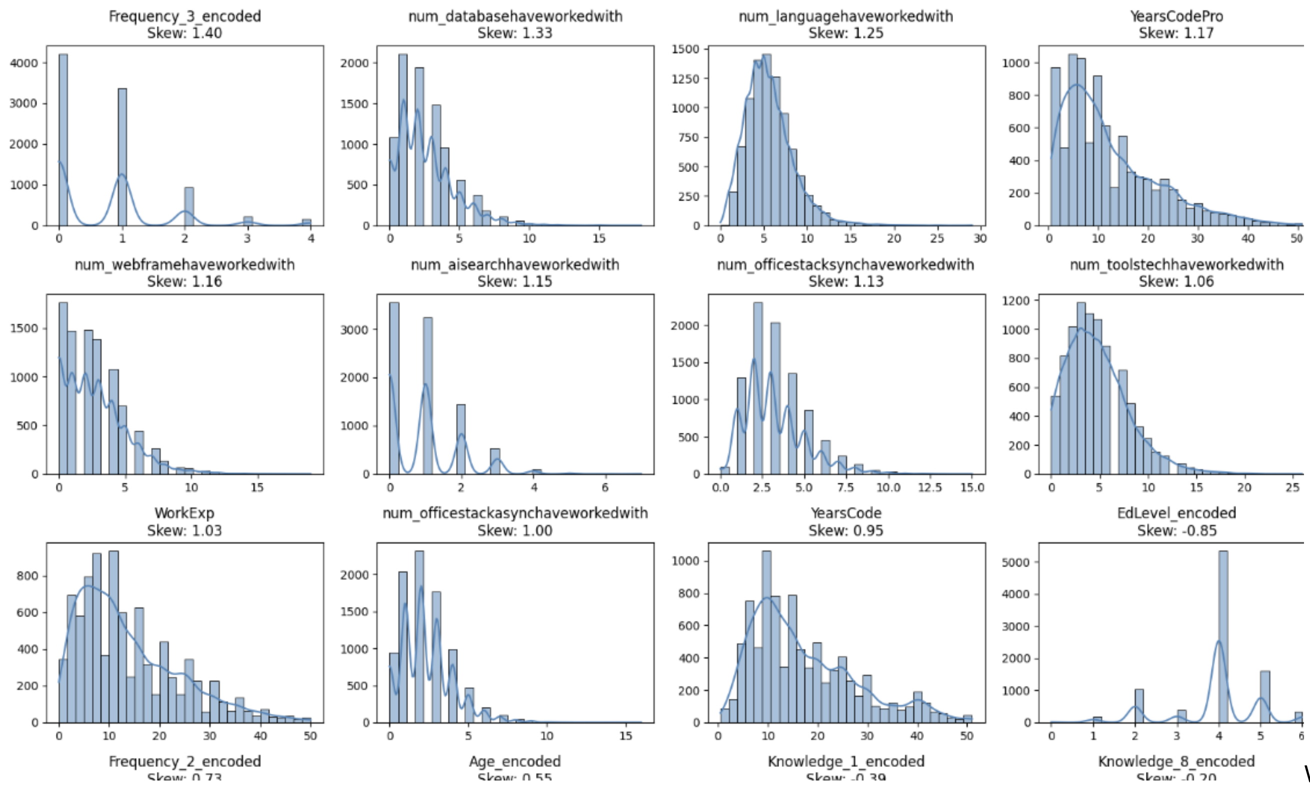
The Feature Expansion model using Random Forest with Tuning closely followed, demonstrating that incorporating database-related and developer-type features can substantially improve predictive accuracy.

By contrast, both the log-transformed model without tuning and the baseline random forest performed significantly worse, showing lower R^2 values (0.1985 and 0.0361, respectively) and higher error rates.

Notably, even applying hyperparameter tuning alone (without additional feature engineering) led to moderate gains. This suggests that both careful model tuning and the inclusion of more relevant features are key drivers of improved salary prediction.

Comprehensive Feature Engineering

Log transform skewed features



We observed strong right-skew in several numeric features, especially around experience and tool usage counts. To reduce skewness and stabilize variance, we applied log1p transformation to a selected subset with high skew (mostly > 1.0), excluding binary or categorical encodings. This helped normalize the distributions and reduced the impact of extreme values.

Create Experience Level Binning

To capture potential non-linear effects of experience on salary, we grouped the WorkExp variable into five bins: Newbie (0–1 years), Junior (2–5), Mid (6–10), Senior (11–20), and Expert (21+). These bins were then one-hot encoded and added to the feature set. This transformation helps the model differentiate between early-career and experienced individuals without assuming a strictly linear relationship between years of experience and salary.

Create Interaction Features

To capture nuanced relationships between key variables, we engineered several interaction terms based on domain logic.

We combined Work Experience and Education to create a **WorkExp_x_Education** feature. This captures how formal education might amplify or moderate the effect of years of experience.

We added an **Education_x_OrgSize** feature to reflect how the value of education could vary depending on company size. For example, a degree might carry more weight in a larger organization with structured pay bands.

We multiplied each **role indicator** (e.g. is_data_scientist, is_engineering_manager, etc.) with **WorkExp** to reflect role-specific experience. This lets the model treat five years of experience as more meaningful if the respondent is in a senior technical role.

We introduced **Age_x_Exp** to represent alignment between career stage and actual coding or work exposure. Disparities here may reflect career changes or non-traditional paths.

Lastly, we added interactions between **remote work types** (remote, hybrid, in-person) with both **OrgSize** and **EdLevel_encoded**. These help assess how work setting and background education interact across company types.

These interactions help the model pick up on real-world complexities that raw features may not capture individually.

Create Domain-Specific Features

We wanted to introduce features that reflect not just what someone knows, but how broadly and confidently they operate in the tech space.

The first idea was to capture technical breadth. Instead of treating each tech/tool column in isolation, we summed the number of tools each person marked as used. This became the TechBreadth feature. The intuition is that someone exposed to many tools likely has broader experience, which might correlate with higher compensation.

Next, we looked at the knowledge rating columns. Since these were Likert-scale values, we averaged them to create a KnowledgeScore. This gives a single, interpretable value reflecting how confident someone feels across different domains. We expected this score to serve as a soft indicator of experience or self-assessed skill level.

Both features were designed to be simple, interpretable, and generalize well. Together, they helped us reduce dimensionality while preserving meaningful signals.

Check Multicollinearity

To address potential multicollinearity, we computed the pairwise correlation matrix across all numeric features and identified pairs with correlation coefficients greater than 0.95. These indicate highly redundant features that may introduce instability in linear models.

However, rather than dropping all correlated pairs blindly, we cross-referenced them against the top 150 features most correlated with the salary target (CompTotal). Only features not present in this top-150 list were considered for removal. This allowed us to retain important features even if they were correlated, while removing redundant ones that provided little predictive signal.

This filtered approach helped simplify the feature set without sacrificing performance-critical predictors.

Remove Low-Variance Features

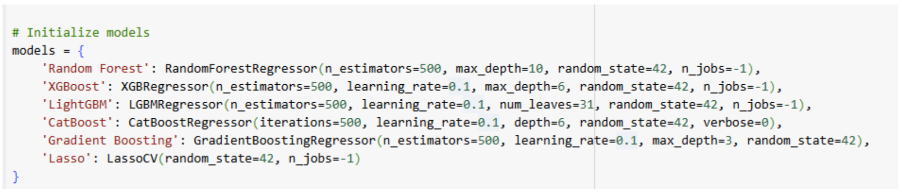
We wanted to clean out features that show little to no variation across the dataset. If almost every row has the same value for a column, it's unlikely that column helps explain salary differences.

We focused only on numeric columns and flagged those with very low variance. Most of them were edge-case signals like rarely used AI tools or employment flags that applied to very few people.

Before dropping anything, we checked whether any of these features appeared among the top 150 most correlated with salary. None of them did. That confirmed they were not only low in variance but also not useful for prediction.

With that reassurance, we safely removed them to reduce noise and make the model's job easier.

Baseline Models Training



```python
# Initialize models
models = {
    'Random Forest': RandomForestRegressor(n_estimators=500, max_depth=10, random_state=42, n_jobs=-1),
    'XGBoost': XGBRegressor(n_estimators=500, learning_rate=0.1, max_depth=6, random_state=42, n_jobs=-1),
    'LightGBM': LGBMRegressor(n_estimators=500, learning_rate=0.1, num_leaves=31, random_state=42, n_jobs=-1),
    'CatBoost': CatBoostRegressor(iterations=500, learning_rate=0.1, depth=6, random_state=42, verbose=0),
    'Gradient Boosting': GradientBoostingRegressor(n_estimators=500, learning_rate=0.1, max_depth=3, random_state=42),
    'Lasso': LassoCV(random_state=42, n_jobs=-1)
}
```

To establish a strong reference point, we selected six widely used regression models that offer a balance of interpretability, flexibility, and scalability. The lineup included ensemble-based learners like Random Forest, XGBoost, LightGBM, CatBoost, and Gradient Boosting, as well as a regularized linear model, Lasso.

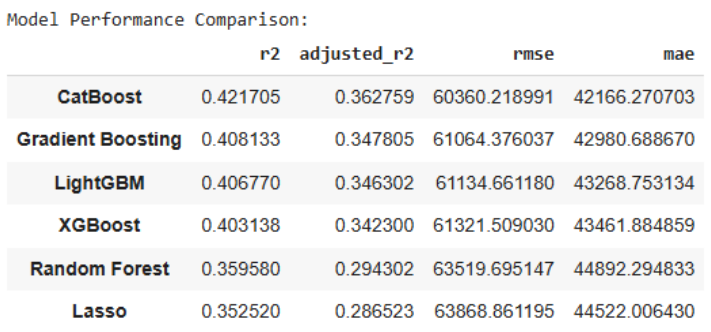
All tree-based models were initialized with 500 estimators and consistent learning rates or depth constraints to maintain fairness during comparison. These configurations reflect commonly used defaults in practical scenarios while still allowing the models enough capacity to learn from the engineered features.

The goal was to see which model architecture could best capture salary variation across a diverse and feature-rich dataset, before diving into hyperparameter tuning or stacking.

Baseline Evaluations

We compared six different regression models to predict salary using the engineered dataset. Among them, CatBoost achieved the strongest results across all key metrics:

- R^2: 0.421705
- Adjusted R^2: 0.362759
- RMSE: 60,360
- MAE: 42,166


Model Performance Comparison:

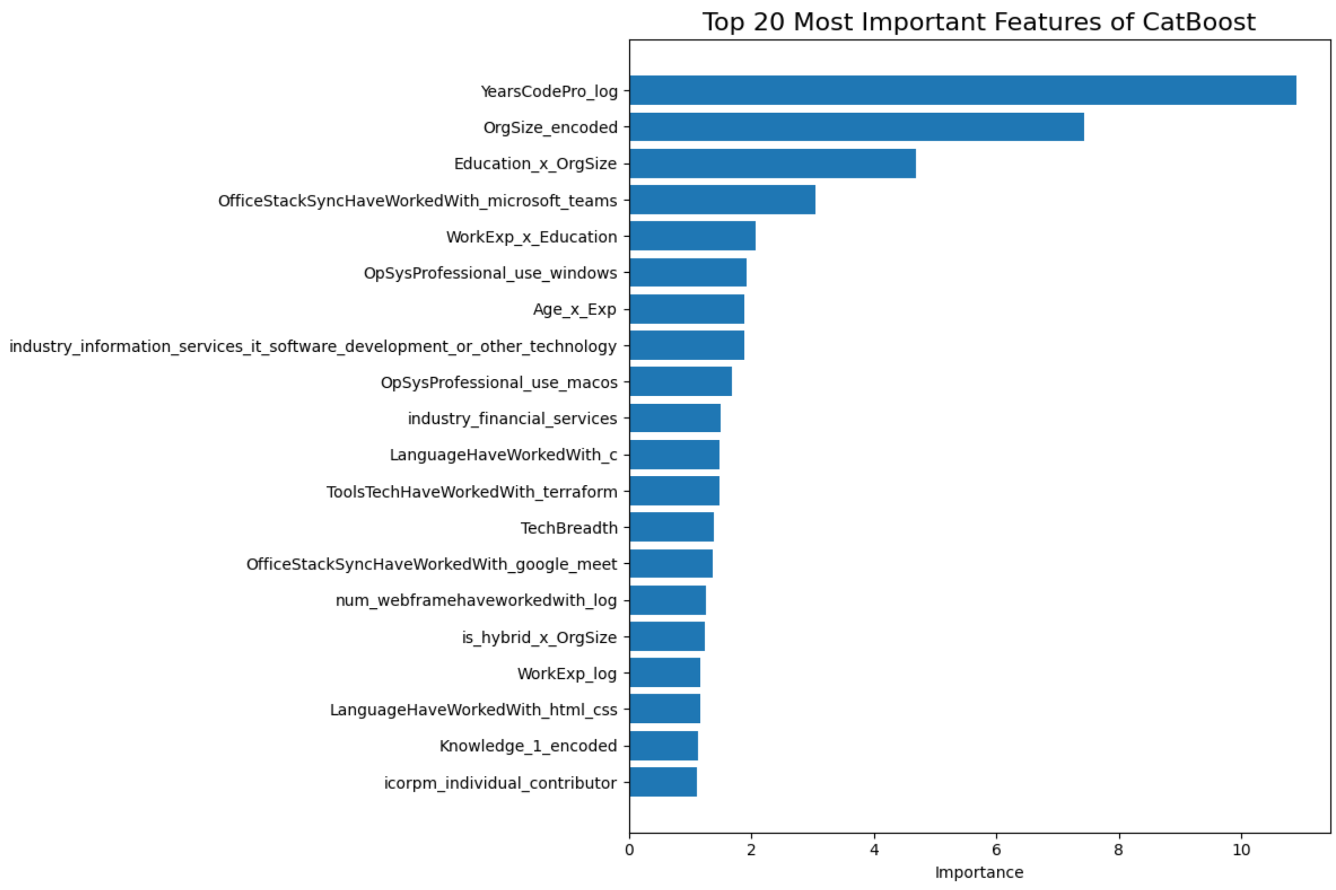
	r2	adjusted_r2	rmse	mae
CatBoost	0.421705	0.362759	60360.218991	42166.270703
Gradient Boosting	0.408133	0.347805	61064.376037	42980.688670
LightGBM	0.406770	0.346302	61134.661180	43268.753134
XGBoost	0.403138	0.342300	61321.509030	43461.884859
Random Forest	0.359580	0.294302	63519.695147	44892.294833
Lasso	0.352520	0.286523	63868.861195	44522.006430

This suggests that CatBoost captured the most variance in salary while maintaining generalization. Gradient Boosting and LightGBM followed closely, both delivering over 0.40 in R^2 with slightly higher RMSE and MAE scores compared to CatBoost.

XGBoost also performed well, but slightly under LightGBM. The tree-based ensemble methods clearly outperformed Random Forest and Lasso, with Random Forest underperforming despite its strong presence in many tabular tasks. Lasso had the lowest R^2 and the highest error metrics, which indicates that linear models may not be flexible enough for this problem.

Overall, the results show that gradient boosting frameworks (especially CatBoost) are well suited for capturing the nonlinear patterns and interactions in the dataset. The use of log-transformed features, interaction terms, and cleaned feature space contributed to improved model accuracy.

Feature Importance Analysis



The most important feature is **YearsCodePro_log**. That shows the log transform on professional coding experience worked well since the original values were skewed. It plays a key role in predicting salary.

OrgSize_encoded and the **Education × OrgSize** interaction also rank high. This means both company size and its interaction with education level have a strong link to salary. Larger companies and higher education levels often lead to better pay, and the combination captures this more effectively.

Microsoft Teams usage also appears high. This likely reflects a structured or enterprise job environment, which usually pays more. Tool usage features can capture context about the workplace.

Engineered interactions like **WorkExp × Education** and **Age × Experience** show up as useful. These combinations add value beyond what the base features provide on their own.

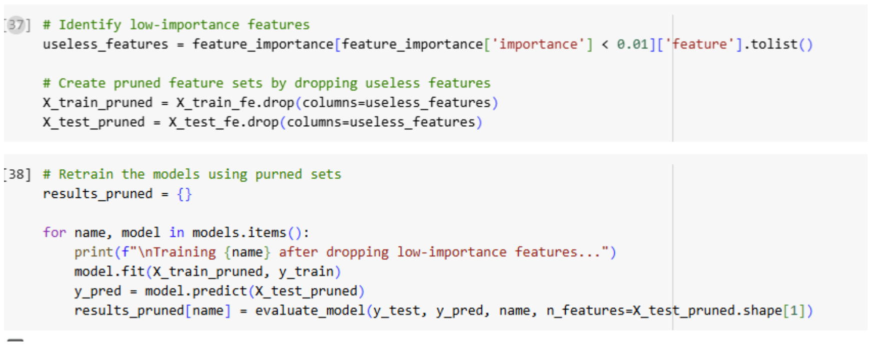
Technical signals such as experience with **C, Terraform, and HTML/CSS** still matter. These probably serve as stand-ins for specific roles like embedded systems or infrastructure.

The **is_hybrid × OrgSize** interaction suggests that hybrid roles in large companies may offer better salaries.

Features like self-assessed knowledge levels and individual contributor roles also contribute, which shows that softer signals can help improve prediction.

Although we explored a variety of additional feature combinations, some were dropped due to multicollinearity concerns or low marginal utility. We kept only those interactions and transformations that demonstrated unique and consistent predictive value.

Pruning and Retraining using Pruned sets



```
[37] # Identify low-importance features
     useless_features = feature_importance[feature_importance['importance'] < 0.01]['feature'].tolist()

     # Create pruned feature sets by dropping useless features
     X_train_pruned = X_train_fe.drop(columns=useless_features)
     X_test_pruned = X_test_fe.drop(columns=useless_features)
```

```
[38] # Retrain the models using purned sets
     results_pruned = {}

     for name, model in models.items():
         print(f"\nTraining {name} after dropping low-importance features...")
         model.fit(X_train_pruned, y_train)
         y_pred = model.predict(X_test_pruned)
         results_pruned[name] = evaluate_model(y_test, y_pred, name, n_features=X_test_pruned.shape[1])
```

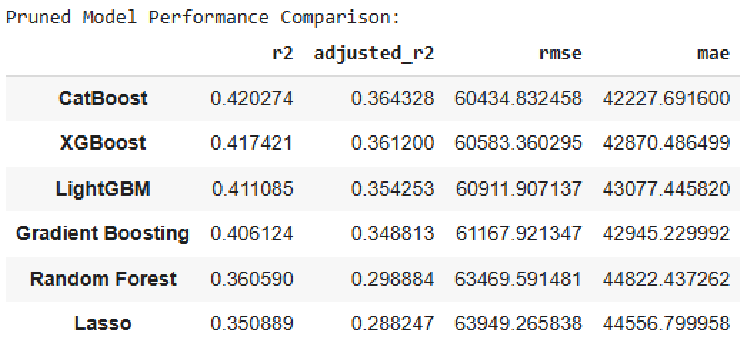
To further optimize the model and reduce noise, we pruned features based on importance scores. Specifically, we removed all features whose importance was less than 0.01, as determined from the CatBoost model. This helped eliminate variables that had minimal contribution to predictive power.

After dropping these low-importance columns, we retrained all models on the reduced feature set. This allowed us to evaluate whether simplifying the input space would lead to more generalizable results without sacrificing performance.

Summary of Model Performance After Feature Pruning

After removing low-importance features, CatBoost remained the best-performing model with an R² of 0.42 and the lowest RMSE and MAE among all models. XGBoost closely followed, showing solid performance across all metrics and slight gains from pruning. LightGBM and Gradient Boosting continued to perform well, with minimal performance changes, indicating their resilience to noise.

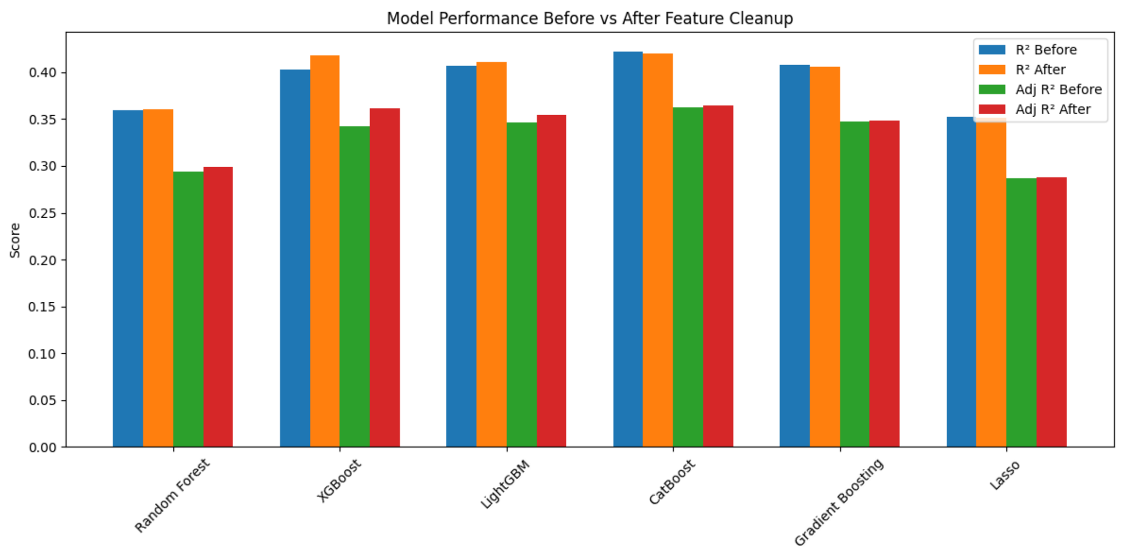
Random Forest improved slightly in adjusted R² but still trailed the boosting models in predictive accuracy. Lasso remained the weakest model, with both the lowest R² and the highest error metrics, consistent with its limitations on complex, non-linear patterns.



Pruned Model Performance Comparison:

	r2	adjusted_r2	rmse	mae
CatBoost	0.420274	0.364328	60434.832458	42227.691600
XGBoost	0.417421	0.361200	60583.360295	42870.486499
LightGBM	0.411085	0.354253	60911.907137	43077.445820
Gradient Boosting	0.406124	0.348813	61167.921347	42945.229992
Random Forest	0.360590	0.298884	63469.591481	44822.437262
Lasso	0.350889	0.288247	63949.265838	44556.799958

The grouped bar chart below further illustrates the change in performance before and after feature pruning. For each model, we compare R² and adjusted R² side by side, showing that most models retained or slightly improved their predictive power after irrelevant features were removed.



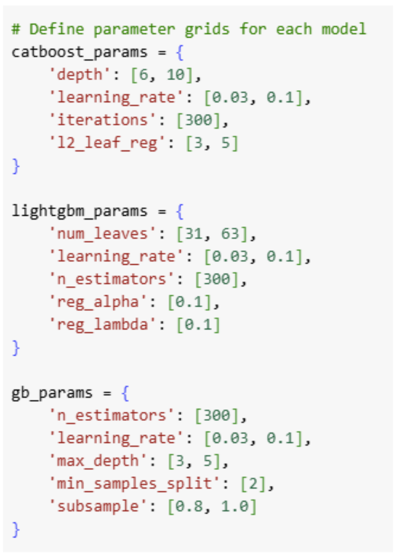
Hyperparameter Tuning for Top Models

After tuning, we compared the final model scores to assess the value of optimization. Our goal was to check whether the parameter adjustments improved performance in a measurable way.

We initially tried grid search but ran into long runtimes and memory constraints, particularly with deeper trees and larger parameter combinations. This made the process inefficient and difficult to scale across models.

To address this, we switched to randomized search. It allowed us to sample from the defined space more efficiently while still capturing the key interactions that influence performance.

The results confirmed that this approach was effective. LightGBM and CatBoost showed slight improvements in R² and error values, while Gradient Boosting also gained modestly. These refinements helped finalize our model choices with more stability and confidence.

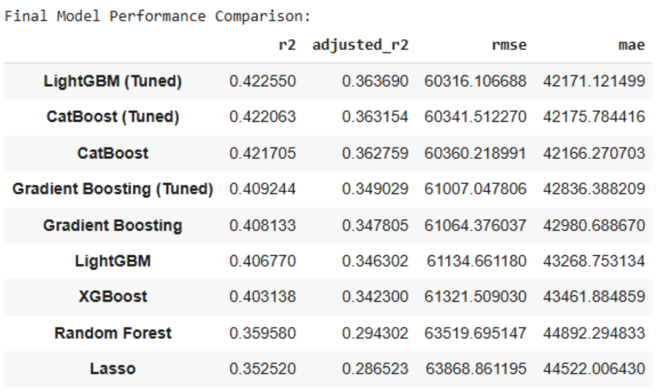


```python
# Define parameter grids for each model
catboost_params = {
    'depth': [6, 10],
    'learning_rate': [0.03, 0.1],
    'iterations': [300],
    'l2_leaf_reg': [3, 5]
}

lightgbm_params = {
    'num_leaves': [31, 63],
    'learning_rate': [0.03, 0.1],
    'n_estimators': [300],
    'reg_alpha': [0.1],
    'reg_lambda': [0.1]
}

gb_params = {
    'n_estimators': [300],
    'learning_rate': [0.03, 0.1],
    'max_depth': [3, 5],
    'min_samples_split': [2],
    'subsample': [0.8, 1.0]
}
```

Final Performance Comparison

After tuning, LightGBM came out just ahead, with the highest R² and the lowest error values overall. The improvements weren't dramatic, but they were enough to show that the tuning made a difference. CatBoost was right behind, also showing a slight boost in performance after adjusting parameters like depth and regularization.

Gradient Boosting improved too, but not by much. The changes in scores were smaller, which suggests that it may not be as sensitive to the tuning space we explored.



Final Model Performance Comparison:

	r2	adjusted_r2	rmse	mae
LightGBM (Tuned)	0.422550	0.363690	60316.106688	42171.121499
CatBoost (Tuned)	0.422063	0.363154	60341.512270	42175.784416
CatBoost	0.421705	0.362759	60360.218991	42166.270703
Gradient Boosting (Tuned)	0.409244	0.349029	61007.047806	42836.388209
Gradient Boosting	0.408133	0.347805	61064.376037	42980.688670
LightGBM	0.406770	0.346302	61134.661180	43268.753134
XGBoost	0.403138	0.342300	61321.509030	43461.884859
Random Forest	0.359580	0.294302	63519.695147	44892.294833
Lasso	0.352520	0.286523	63868.861195	44522.006430

In the end, the tuned versions of LightGBM and CatBoost stood out as the most reliable. The models were already strong before, but tuning helped make them a bit more stable and consistent. This gave us more confidence in using them as final candidates.

Future Enhancements

We've already seen meaningful improvements from advanced feature engineering, interaction terms, and tuning best models, which helped the models learn non-linear patterns more effectively. To build on this, we plan to expand the dataset beyond 2023 to reduce temporal bias and improve generalization. Incorporating additional context like historical salary trends, cost of living, or job market conditions could further strengthen predictions. We're also considering shifting from point estimates to predicting salary ranges, which would make the outputs more realistic and useful in applied settings.

Github:

https://github.com/c0927648/AML2203_Project_Salary_Prediction_StackOverflow_2023/blob/main/FinalSubmission

Resources

Software Developers, Quality Assurance Analysts, and Testers. (2025, April 18). Bureau of Labor Statistics. https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm