Developer Salary Prediction Model

REGRESSION

TEAM 4- [GPS]

PROBLEM STATEMENT

Build a machine learning model to accurately predict developer salaries based on experience, skills, and job profile data.

1st DATA CLEANING

🔹 **Objective:**

To remove irrelevant, missing, or inconsistent data to improve model performance.

✅ **HOW WE DONE IT:**

- Dropped Irrelevant Columns Columns like LearnCode, TechList, BuyNewTool, and others not contributing to prediction were removed.
- Removed Rows with Missing Salary Rows without a valid ConvertedCompYearly (target) were dropped.
- Filtered for Full-Time Employees Only Ensured consistency by keeping only Full-time entries in Employment.

Source: Stack Overflow Developer Survey

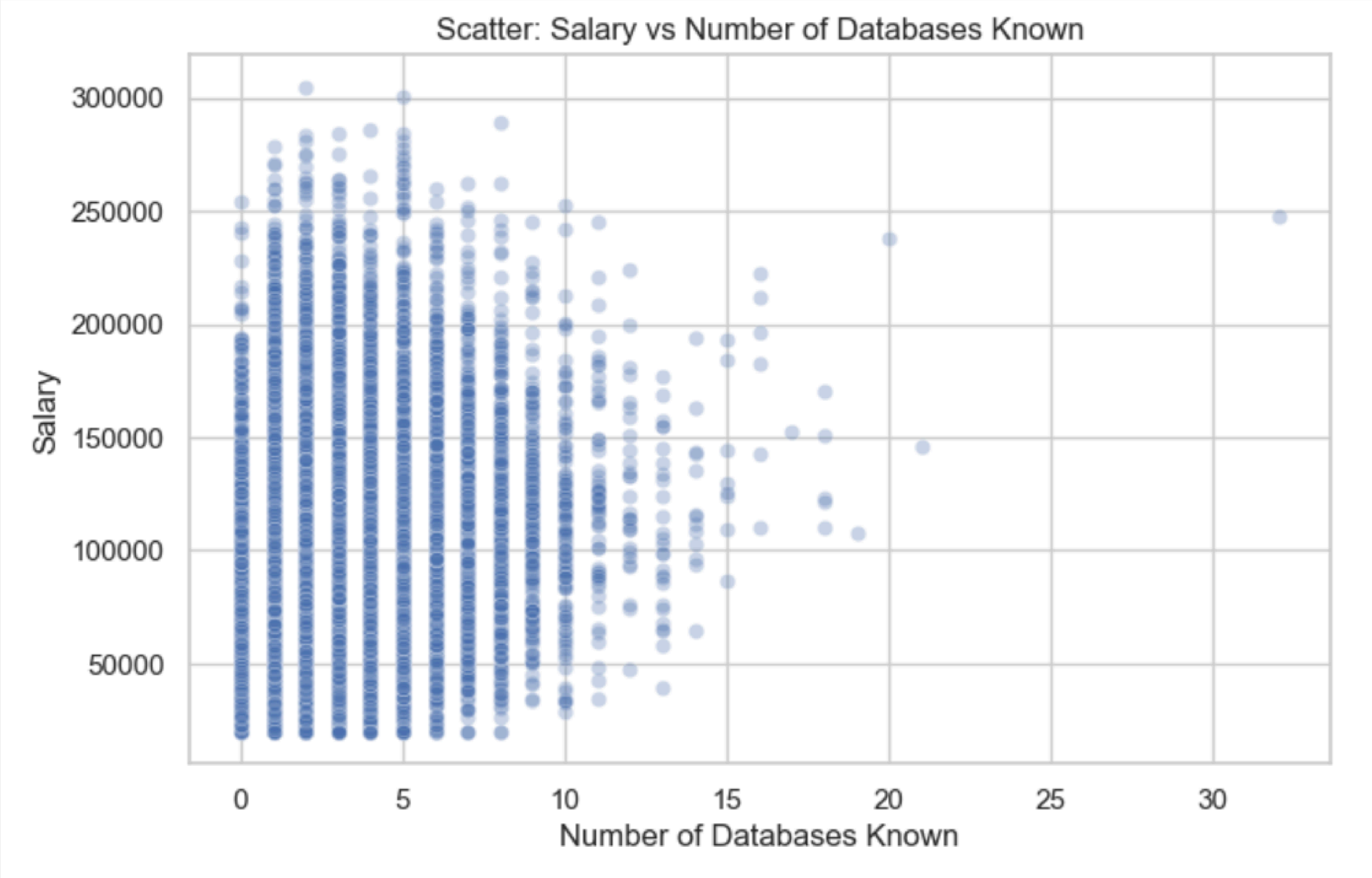
Exploratory Data Analysis

◆ Objective ◆
To understand the data distribution, identify key patterns, and detect anomalies before modeling.

✅LET US SEE,HOW WE DONE IT:

Positive Correlation:

- The graph shows an upward trend — as years of work experience increase, salary also tend to increase.
- This suggests that more experienced individuals generally earn higher salaries.



🔍 Interpretation:

- No strong correlation is visible here.
- High density around 0–10 databases.
- Salaries don’t necessarily increase with more databases known.

🔍 Interpretation:

- Strong positive correlation.
- Salaries tend to increase with more years of professional coding experience.
- Similar to general work experience, but even more tightly clustered with a clearer upward trend.

Feature Engineering

1. **Selected Relevant Features**
2. **Encoded Categorical Features**
3. **Normalized Experience Data**
4. **Grouped Rare Categories**

```
Age                             0
Employment                      6
RemoteWork                      0
EdLevel                         0
YearsCodePro                    0
DevType                        52
OrgSize                         0
Country                         0
DatabaseHaveWorkedWith          0
PlatformHaveWorkedWith       8485
WebframeHaveWorkedWith       8016
OpSysProfessional use        2314
WorkExp                     13125
Industry                        0
ConvertedCompYearly             0
NumberOfDatabasesKnown          0
NumberOfPlatformsKnown          0
NumberOfWebFrameworksKnown      0
```

```python
# Replace this list with your actual country column from the DataFrame
countries = [
    'United States of America', 'Philippines', 'United Kingdom of Great Britain and Northern Ireland',
    'Germany', 'France', 'Albania', 'Spain', 'Bangladesh', 'Switzerland', 'Lithuania', 'Serbia',
    'Netherlands', 'Australia', 'Greece', 'Norway', 'Turkey', 'Sweden', 'India', 'Poland', 'Finland',
    # ... include all your countries
]

# Function to convert country to continent
def country_to_continent(country_name):
    try:
        country_code = pc.country_name_to_country_alpha2(country_name, cn_name_format="default")
        continent_code = pc.country_alpha2_to_continent_code(country_code)
        return pc.convert_continent_code_to_continent_name(continent_code)
    except:
        return "Unknown"

# Create DataFrame
df_continents = pd.DataFrame({'Country': countries})
df_continents['Continent'] = df_continents['Country'].apply(country_to_continent)
```

Regression Models Are Used

🔍 Why is it a Regression Model?

✅ 1. Continuous Output (Target Variable)

- The output (salary) is a continuous numerical value, not a category.
- Example:
  - Predicting ₹95,000 or ₹1,50,000 — these are numerical values on a scale, not discrete classes.

✅ 2. Goal: Predict Quantitative Value

- Regression models are used when the goal is to estimate or predict "how much" or "how many".
- Here, you're predicting "how much salary" someone will earn.

1. Linear Regression

```
# 🔹 Split into training and testing sets (80% train, 20% test)
train_x, test_x, train_y, test_y = train_test_split(X, y, test_size=

# 🔹 Train model
regr = LinearRegression()
regr.fit(train_x, train_y)

# 🔹 Predict and evaluate
test_y_hat = regr.predict(test_x)
print("R2-score:", r2_score(test_y, test_y_hat))
```

```
R2-score: 0.8569365556767388
```

After training, we retrieved the coefficients and intercept:

Coefficients (regr.coef_) represent the impact each feature has on the salary.

Intercept (regr.intercept_) is the expected salary when all features are zero.

✅ Simple, Interpretable Model

- Assumes a linear relationship between features (e.g., experience, databases known) and the target (salary).
- Fits a straight line:
- Salary=β0+β1 · Experience+β2 · Coding Exp+....

🔹 Pros:

- Very fast to train.
- Easy to interpret and explain.
- Works well when relationships are linear.

🔻 Cons:

- Performs poorly with non-linear data or complex feature interactions.
- Sensitive to outliers and multicollinearity.

🌲 2. Random Forest Regressor

```python
# Train Random Forest Regressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))  # Fixed here
mae = mean_absolute_error(y_test, y_pred)

print(f"R² Score: {r2:.3f}")
print(f"RMSE: {rmse:.2f}")
print(f"MAE: {mae:.2f}")
```

```
R² Score: 0.848
RMSE: 16416.34
MAE: 13095.29
```

✅ Ensemble of Decision Trees

- Combines many decision trees trained on different data subsets.
- Each tree makes a prediction, and the final output is the average of all.

🔷 Pros:

- Handles non-linear patterns and feature interactions well.
- More robust to outliers and noise.
- Performs better than linear models on complex data.

🔻 Cons:

- Less interpretable.
- Training is slower than linear models.
- Slightly less accurate than XGBoost in many cases.

⚡ 3. XGBoost Regressor ✅)

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train XGBoost model
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=4, random_state=42)
xgb_model.fit(X_train, y_train)

# Predict
y_pred = xgb_model.predict(X_test)

# Evaluate
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"RMSE: {rmse:.2f}")
print(f"R² Score: {r2:.2f}")
```

```
RMSE: 15384.84
R² Score: 0.87
```

⚡ 3. XGBoost Regressor (Best Performa✅ Extreme Gradient Boosting

- An advanced tree-based algorithm that builds trees sequentially, learning from previous errors.

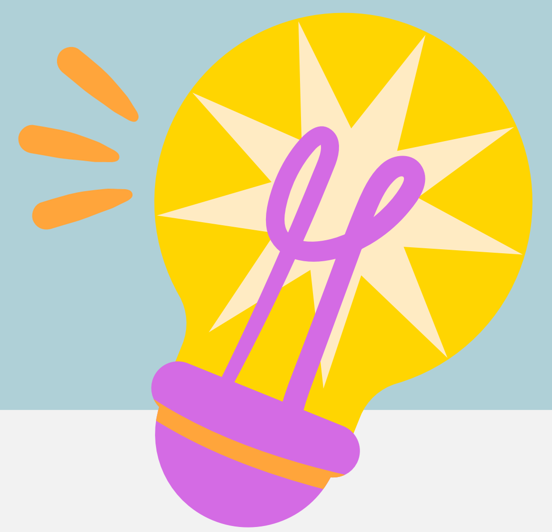
🔷 Pros:

- High performance: Often the best model in real-world regression tasks.
- Handles missing values, outliers, and feature interactions automatically.

🔻 Cons:

- More complex to tune (hyperparameters).
- Requires more computing power.

🎯 Why We Used XGBoost in Our Salary Prediction Model



🔑 Model-Specific Reasons:

1. 🏆 Best Performance on Our Data
   - Among Linear Regression, Random Forest, and XGBoost — XGBoost achieved the highest R^2 score and lowest RMSE during testing.
2. 📈 Captures Complex Patterns
   - Salary depends on non-linear interactions (e.g., coding experience may matter more after a certain threshold).
   - XGBoost automatically learns such non-linear trends.
3. 🧠 Smart Regularization
   - Helps avoid overfitting, even though salary data can be noisy or have outliers.
   - Built-in L1/L2 regularization gave our model better generalization.
4. 📊 Interpretable Feature Importance
   - We used XGBoost to analyze which features most strongly influence salary, adding insight to our predictions.

"This graph compares actual salaries with predicted ones from our Linear Regression model. The clustering around the diagonal line shows that the model captured the salary trends well, though more advanced models like XGBoost offered even better accuracy."

"Most predictions are close to actual values, proving our model is both accurate and trustworthy."

Thank you!