Predicting Salary and Identifying Job Security Risks Based on Company Location, Size, and Role

Vaibhav Bansal
School of Engineering and Applied Sciences, *State University New York*
Buffalo, United States
vbansal6@buffalo.edu

Tejas Kolpek
School of Engineering and Applied Science, *State University New York*
Buffalo, United States
tejasgop@buffalo.edu

Farahath Tabassum
School of Engineering and Applied Sciences, *State University New York*
Buffalo, United States
farahath@buffalo.edu

Abstract— **With the mass layoffs in 2022 affecting many sectors, particularly the tech industry, there is a need to understand how factors such as experience level, job role, remote work arrangements, and company size influenced job security and salary. This project aims to develop a predictive model that not only estimates salary based on company size, location, and job title but also identifies opportunities for professionals for getting a job. By analyzing these factors, the model seeks to provide valuable insights into how location and company characteristics affect salary offers and job stability in data-related positions. The objective is to help job seekers—whether fresh graduates or experienced professionals—improve their employment prospects by understanding these critical dynamics, offering guidance on securing stable employment and maximizing salary expectations in the tech industry.**

Keywords — EDA, Data Science, Salary Prediction

I. Problem Statement

The rapid increase in layoffs in the tech sector, especially in 2022, created instability in what had been perceived as a robust job market. This raised crucial questions about which factors contribute to job security and salary fluctuations, making it imperative to understand the impact of company characteristics (e.g., size and location) on salary and layoffs. According to *Glassdoor Economic Research* (2021), salaries in tech hubs have increased even during disruptive events like the COVID-19 pandemic. However, without a detailed analysis of which variables contribute to these trends, professionals remain uncertain about how to secure stable, well-compensated roles.

This project is significant because it helps demystify the salary structure and job security risks, offering vital insights to individuals who are seeking to navigate a highly competitive and uncertain job market. This project aims to address these challenges by developing a predictive model to estimate salaries based on company size, location, and job title while also identifying professionals most at risk for layoffs. By analysing factors such as experience level, remote work arrangements, and company characteristics, the model seeks to provide valuable insights for both fresh graduates and experienced professionals to improve their job security and optimize salary expectations. Furthermore, companies can utilize the model to attract and retain talent by understanding how their compensation practices compare to industry norms. With data-driven insights, the model will empower individuals to make informed decisions about their employment prospects and salary expectations, ultimately contributing to better job security and improved compensation packages in the tech industry. As the demand for data-related roles continues to rise, this contribution becomes crucial for both job seekers and companies striving to stay competitive in a rapidly changing environment.

II. Introduction

This The tech industry faced significant disruptions in 2022, with mass layoffs reshaping the employment landscape. As professionals navigate this uncertainty, understanding the factors that influence job security and compensation becomes critical. This project focuses on analysing how variables such as experience level, job role, remote work arrangements, and company size affect both salary and job stability within the data science field. By developing a predictive model, we aim to estimate salaries based on key factors like company size, location, and job title, while also identifying job opportunities for professionals who may be at higher risk of layoffs. The insights gained from this analysis will be valuable to both fresh graduates and seasoned professionals, guiding them toward more secure employment opportunities and optimized salary negotiations. Despite these challenges, tech salaries have shown resilience, with average compensation increasing in major hubs during the pandemic era, as noted by Glassdoor Economic Research (2021). This suggests that demand for tech talent remains high, even during periods of economic uncertainty" [1]. According to the World Economic Forum (2020), emerging professions such as data analysts and AI specialists will be central to the future economy, creating substantial opportunities for tech professionals" [2]

This project provides actionable insights to help individuals make informed decisions in a rapidly evolving tech industry, improving their prospects for stable, well-compensated roles. follow.

III. Data Acquisiton

The data was sourced from the **"Data Science Salaries 2023"** dataset available on **Kaggle** [3]. The dataset provides comprehensive information on salaries for various data science roles across the globe. It includes attributes such as job titles, experience levels, employment types, company sizes, employee residence, and salaries standardized to USD. This data is critical for analysing trends in compensation across different sectors of the data science industry, highlighting disparities in salaries based on experience, location, and company size. The dataset's global nature allows for a cross-regional analysis of how data science roles are valued in different economic regions.

This dataset offers a foundation for understanding how industry changes—such as shifts in remote work and company size—affect salaries and job security. By analysing these factors, the research aims to provide actionable insights for both freshers and experienced professionals. This helps in identifying the sectors and job roles that offer stability and competitive compensation, particularly in light of industry-wide layoffs and economic downturns. Additionally, the dataset allows us to explore potential correlations between remote work adoption and salary variations, providing a well-rounded view of the current state of the data science job market.

IV. FLOW DIAGRAM

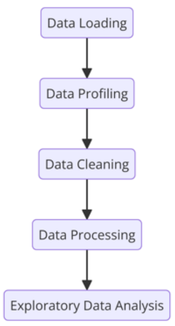


Figure 1.a Flow Diagram

V. DATA LOADING

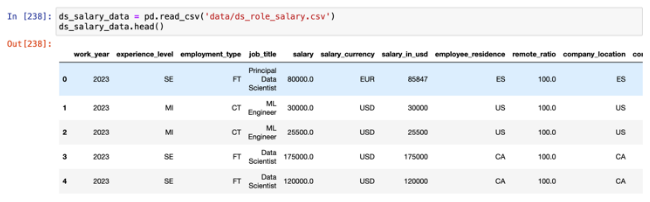
We utilized a dataset containing information about data science and machine learning roles, with fields such as job titles, salary, employee residence, remote work ratio, and company details. The dataset was obtained in CSV format, which was loaded into the working environment using Python's pandas library.

Data Source Identification: The dataset was provided in CSV format, containing several key columns, including:

Column Name	Description
work_year	Year the salary data pertains to.
experience_level	Experience level of the employee (e.g., SE - Senior, MI - Mid-level).
employment_type	Type of employment contract (e.g., Full-Time (FT), Contract (CT)).
job_title	Job role (e.g., Data Scientist, ML Engineer).
salary	Employee salary in the given currency.
salary_in_usd	Salary converted to USD for uniform comparison.
employee_residence	Employee's country of residence.
remote_ratio	Percentage of work done remotely.
company_location	Company's primary location.
company_size	Size of the company (e.g., Small (S), Medium (M), Large (L)).

Table 1 Data Column Description

The dataset was loaded using the `pandas.read_csv()` function in Python, which allowed us to import the CSV file into a structured Data Frame. This Data Frame served as the foundation for further analysis and processing



```
In [238]: ds_salary_data = pd.read_csv('data/ds_role_salary.csv')
          ds_salary_data.head()
Out[238]:
```

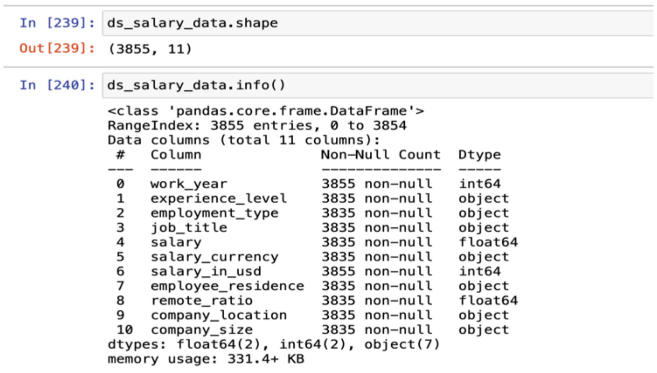
	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	co
0	2023	SE	FT	Principal Data Scientist	80000.0	EUR	85847	ES	100.0	ES	
1	2023	MI	CT	ML Engineer	30000.0	USD	30000	US	100.0	US	
2	2023	MI	CT	ML Engineer	25500.0	USD	25500	US	100.0	US	
3	2023	SE	FT	Data Scientist	175000.0	USD	175000	CA	100.0	CA	
4	2023	SE	FT	Data Scientist	120000.0	USD	120000	CA	100.0	CA	

Figure 1.1 Data Loading

Figure 1.1 shows that After loading the data, a preliminary inspection of the dataset was conducted by viewing the first few rows using the `head()` function. This helped verify that the data was loaded correctly, and all fields were present as expected.

VI. DATA PROFILING

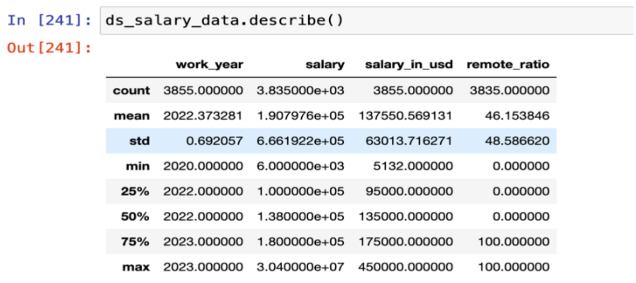
Data profiling is an essential step to understand the structure and quality of the dataset before proceeding with more detailed analysis or modeling.



```
In [239]: ds_salary_data.shape
Out[239]: (3855, 11)

In [240]: ds_salary_data.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 3855 entries, 0 to 3854
          Data columns (total 11 columns):
           #   Column              Non-Null Count  Dtype
          ---  ------              --------------  -----
           0   work_year           3855 non-null   int64
           1   experience_level    3835 non-null   object
           2   employment_type     3835 non-null   object
           3   job_title           3835 non-null   object
           4   salary              3835 non-null   float64
           5   salary_currency     3835 non-null   object
           6   salary_in_usd       3855 non-null   int64
           7   employee_residence  3835 non-null   object
           8   remote_ratio        3835 non-null   float64
           9   company_location    3835 non-null   object
           10  company_size        3835 non-null   object
          dtypes: float64(2), int64(2), object(7)
          memory usage: 331.4+ KB
```

Figure.2 :Dataset shape & structure

The first part of the Figure 2 shows the output of ds_salary_data.shape This indicates that the dataset contains **3855 rows** (entries) and **11 columns**. The second part of Figure 2 use the `info()` function shows that the dataset contains 3855 entries with 11 columns, including numerical and categorical data types. Most columns have 3855 non-null values, except `remote_ratio` with 3835, and the dataset uses 331.4 KB of memory.



```
In [241]: ds_salary_data.describe()
Out[241]:
```

	work_year	salary	salary_in_usd	remote_ratio
count	3855.000000	3.835000e+03	3855.000000	3835.000000
mean	2022.373281	1.907976e+05	137550.569131	46.153846
std	0.692057	6.661922e+05	63013.716271	48.586620
min	2020.000000	6.000000e+03	5132.000000	0.000000
25%	2022.000000	1.000000e+05	95000.000000	0.000000
50%	2022.000000	1.380000e+05	135000.000000	0.000000
75%	2023.000000	1.800000e+05	175000.000000	100.000000
max	2023.000000	3.040000e+07	450000.000000	100.000000

Figure.3: statistical summaries

Figure 3 The `describe()` function output provides a statistical summary of the dataset, revealing key insights into the numerical columns. The dataset contains 3855 entries for most columns, with a few missing values in `remote_ratio`. The average work year is 2022, and the

mean salary in USD is approximately $137,550 with a standard deviation of around $63,013. The salary ranges from a minimum of $5,132 to a maximum of $450,000, with 25% of salaries below $95,000, 50% below $135,000, and 75% below $175,000. This summary offers a comprehensive understanding of the salary distribution and other numerical data points, such as remote work ratio.

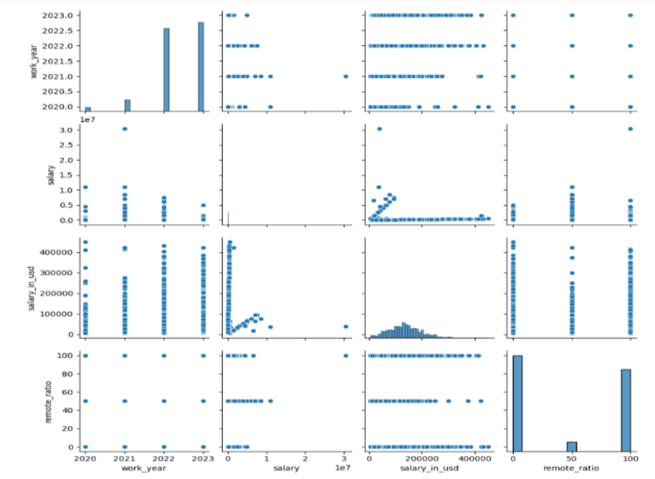


Figure.4: Pair plot

Figure.4 shows the pair plot that displays pairwise relationships between the variables in the dataset, as well as their individual distributions. Below is an explanation of each graph within the pair plot:

1. **Work Year vs. Other Variables**
   - **Work Year vs Salary (left-most column)**: This graph shows how salary (both scaled and in USD) has varied across different years. We can see that salaries have generally increased over time, with more records and higher salary values in 2022 and 2023 compared to 2020 and 2021.
   - **Work Year Distribution (top-left plot)**: This is a bar plot showing the distribution of records by year. Most records are from 2022 and 2023, with significantly fewer records from 2020 and 2021, indicating that the dataset is more focused on recent years.
2. **Salary vs. Other Variables**
   - **Salary vs. Work Year (second column)**: This plot shows how the 'salary' column (possibly representing unscaled salary values) changes across the years. It is likely showing fewer, lower values in earlier years compared to 2022 and 2023.
   - **Salary vs. Salary in USD (second column, second row)**: This plot compares the 'salary' and 'salary_in_usd' variables. There appears to be a direct relationship between these two columns, which makes sense since both represent salary, but one may be the raw figure while the other is converted to USD.
   - **Salary Distribution (second column, third row)**: This graph shows the distribution of the 'salary' variable, which could be the unscaled salary data. We can see that most of the data points are clustered towards lower salary ranges.
3. **Salary in USD vs. Other Variables**
   - **Salary in USD vs. Work Year (third column, first row)**: This graph shows how the salary in USD has evolved across different years. There is a visible cluster of data points, especially for 2022 and 2023, indicating that these years have higher salary records.
   - **Salary in USD Distribution (third column, third row)**: This is a histogram showing the distribution of salary in USD. It appears to be right-skewed, with most salaries concentrated below $150,000, and fewer people earning above this threshold.
   - **Salary in USD vs. Remote Ratio (third column, fourth row)**: This graph shows how salary relates to the percentage of remote work. It suggests that remote work percentage does not have a strong, direct correlation with salary as the points are scattered without a clear trend.

4. Remote Ratio vs. Other Variables

- **Remote Ratio vs. Work Year (fourth column, first row)**: This plot shows the distribution of remote work percentages across different years. Most records are clustered at 0% (onsite) and 100% (fully remote), with fewer records showing a hybrid setup (50%).
- **Remote Ratio vs. Salary in USD (fourth column, third row)**: This scatter plot shows the relationship between the remote ratio and salary in USD. Similar to the previous scatter plot, there doesn't seem to be a strong relationship between these two variables.
- **Remote Ratio Distribution (fourth column, fourth row)**: This bar plot shows the distribution of remote work percentages. There are three distinct categories: 0% (onsite), 50% (hybrid), and 100% (fully remote), with the majority of the data points being onsite or fully remote.

5. Observations:

5.1 Distribution Plots

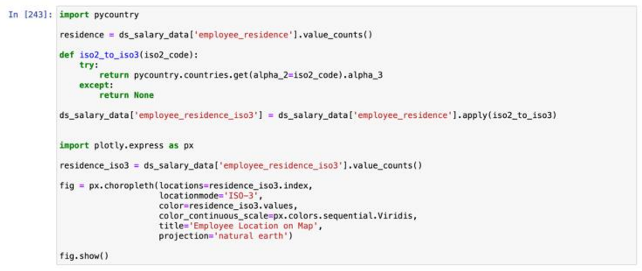
The diagonal graphs are histograms showing the distribution of individual variables, giving insight into their spread and skewness. For example, salaries are heavily skewed to the left, with most salaries being below $150,000.

5.2 Scatter Plots

The off-diagonal scatter plots show the pairwise relationships between variables, providing insights into how these variables interact with each other. For instance, we can observe that salary in USD increases over the years and does not appear to have a strong correlation with the remote work percentage.

These graphs are useful for identifying trends, distributions, and possible relationships between variables, which can guide further data exploration and analysis.

5.3 Choropleth Map:



```
In [243]: import pycountry

          residence = ds_salary_data['employee_residence'].value_counts()

          def iso2_to_iso3(iso2_code):
              try:
                  return pycountry.countries.get(alpha_2=iso2_code).alpha_3
              except:
                  return None

          ds_salary_data['employee_residence_iso3'] = ds_salary_data['employee_residence'].apply(iso2_to_iso3)


          import plotly.express as px

          residence_iso3 = ds_salary_data['employee_residence_iso3'].value_counts()

          fig = px.choropleth(locations=residence_iso3.index,
                              locationmode='ISO-3',
                              color=residence_iso3.values,
                              color_continuous_scale=px.colors.sequential.Viridis,
                              title='Employee Location on Map',
                              projection='natural earth')

          fig.show()
```

Figure.5: Chloropleth Map Code

Figure 5 represents the code to change the employee residence from ISO-2 to ISO-3 format to make compatible enough to plot the employee residence coordinates in world map.

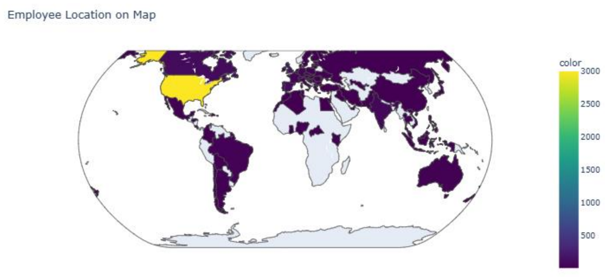


Figure.6: Chloropleth Map

The Figure 6 presents a **choropleth map** generated using the Plotly library. The map visualizes the distribution of employees' locations across the globe based on the employee_residence variable from the dataset.
The map shows the **global distribution of employees**:

- **Yellow areas** (e.g., the United States) indicate countries with the highest number of employees in the dataset (as indicated by the color bar, this could be upwards of 3000 employees).
- **Dark purple areas** represent countries with fewer employees (around 500 or fewer).
- Countries with no employees in the dataset appear as light gray.

This visualization helps in understanding which regions or countries employ the most data science professionals (as per the dataset). For instance, North America, Europe, and parts of Asia appear to have higher concentrations of employees, while countries in Africa and South America seem to have fewer data science professionals based on the data.

5.3.1 Key Insights:

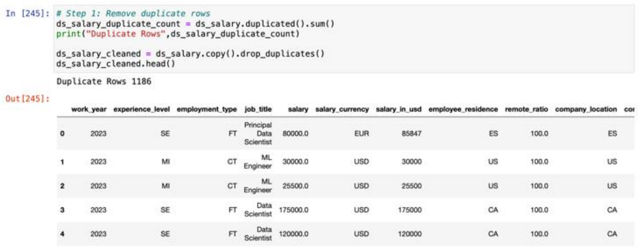
- **The United States** appears to have the highest number of employees based on the dataset.
- **Europe** and parts of **Asia** show significant employee representation as well.
- **Africa** and **South America** seem underrepresented in terms of employee numbers, which could reflect the dataset's focus on certain regions or the broader industry concentration in data science.
- This map offers a **geographic visualization of employee distribution**, highlighting regional trends and workforce concentrations globally.

VII. DATA CLEANING

The initial step in preparing the dataset for analysis involved several key data cleaning tasks to ensure its accuracy and consistency. Data cleaning involved removing 1186 duplicate rows, handling missing values by dropping rows with null entries, normalizing categorical data, and eliminating outliers to ensure a consistent and accurate dataset for further analysis.

1. Handling Duplicate Data

The dataset originally contained 1186 duplicate rows, which were removed to avoid redundant or misleading data. This step reduced the dataset to 2669 unique rows.



```
In [245]: # Step 1: Remove duplicate rows
          ds_salary_duplicate_count = ds_salary.duplicated().sum()
          print("Duplicate Rows",ds_salary_duplicate_count)

          ds_salary_cleaned = ds_salary.copy().drop_duplicates()
          ds_salary_cleaned.head()

          Duplicate Rows 1186

Out[245]:
```

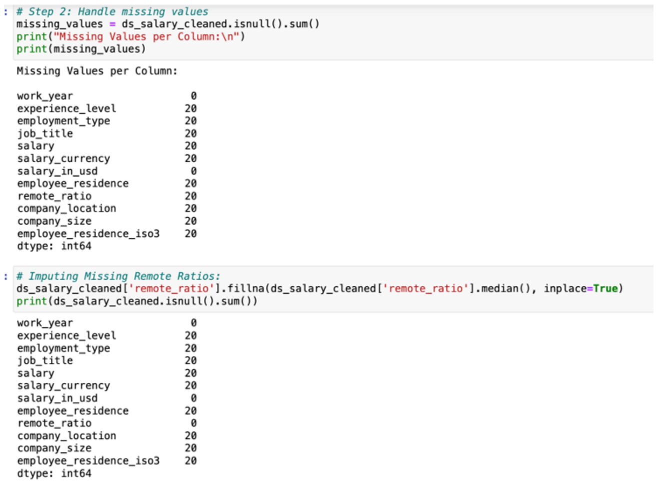
	work_year	experience_level	employment_type	job_title	salary	salary_currency	salary_in_usd	employee_residence	remote_ratio	company_location	co
0	2023	SE	FT	Principal Data Scientist	80000.0	EUR	85847	ES	100.0	ES	
1	2023	MI	CT	ML Engineer	30000.0	USD	30000	US	100.0	US	
2	2023	MI	CT	ML Engineer	25500.0	USD	25500	US	100.0	US	
3	2023	SE	FT	Data Scientist	175000.0	USD	175000	CA	100.0	CA	
4	2023	SE	FT	Data Scientist	120000.0	USD	120000	CA	100.0	CA	

Figure.7: Handling Duplicate Values

The output table provides a preview of the first 5 rows after duplicates were removed. The cleaned dataset now contains only unique rows, ensuring the analysis will not be skewed

2. Imputing Median Values for Missing Values (remote_ratio)

The `remote_ratio` column had a few missing values, which were imputed with the median value of the column. Since `remote_ratio` represents the percentage of remote work, the median was chosen as it is less sensitive to outliers and provides a central value that best represents the typical remote work ratio in the dataset



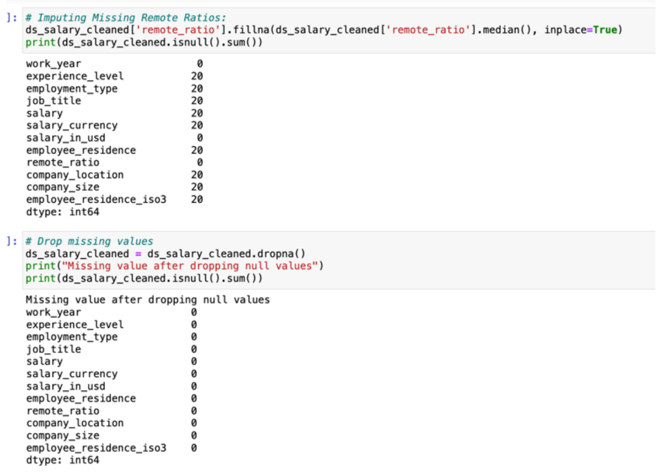
```
: # Step 2: Handle missing values
  missing_values = ds_salary_cleaned.isnull().sum()
  print("Missing Values per Column:\n")
  print(missing_values)

  Missing Values per Column:

  work_year                   0
  experience_level           20
  employment_type            20
  job_title                  20
  salary                     20
  salary_currency            20
  salary_in_usd               0
  employee_residence         20
  remote_ratio               20
  company_location           20
  company_size               20
  employee_residence_iso3    20
  dtype: int64

: # Imputing Missing Remote Ratios:
  ds_salary_cleaned['remote_ratio'].fillna(ds_salary_cleaned['remote_ratio'].median(), inplace=True)
  print(ds_salary_cleaned.isnull().sum())

  work_year                   0
  experience_level           20
  employment_type            20
  job_title                  20
  salary                     20
  salary_currency            20
  salary_in_usd               0
  employee_residence         20
  remote_ratio                0
  company_location           20
  company_size               20
  employee_residence_iso3    20
  dtype: int64
```

Figure.8a: Imputing Median Values

This approach ensures that important information, such as the remote work ratio, is preserved through imputation, while rows with missing categorical data are excluded to maintain data quality.

3. Handling Null Values (Drop null Values)

The dataset had missing values across several columns, including `experience_level`, `employment_type`, `job_title`, and `salary_currency`. We opted to drop rows containing missing values to maintain data integrity.



```
]: # Imputing Missing Remote Ratios:
   ds_salary_cleaned['remote_ratio'].fillna(ds_salary_cleaned['remote_ratio'].median(), inplace=True)
   print(ds_salary_cleaned.isnull().sum())

   work_year                0
   experience_level        20
   employment_type         20
   job_title               20
   salary                  20
   salary_currency         20
   salary_in_usd            0
   employee_residence      20
   remote_ratio             0
   company_location        20
   company_size            20
   employee_residence_iso3    20
   dtype: int64
```

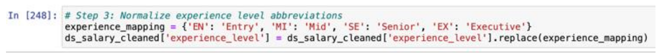
```
]: # Drop missing values
   ds_salary_cleaned = ds_salary_cleaned.dropna()
   print("Missing value after dropping null values")
   print(ds_salary_cleaned.isnull().sum())

   Missing value after dropping null values
   work_year               0
   experience_level        0
   employment_type         0
   job_title               0
   salary                  0
   salary_currency         0
   salary_in_usd           0
   employee_residence      0
   remote_ratio            0
   company_location        0
   company_size            0
   employee_residence_iso3    0
   dtype: int64
```

Figure.8b: Handling Missing Values

Step 2 handles the Missing Values in the data preprocessing process. The output table lists the columns of the dataset along with the number of missing values in each column. Work_year, experience_level, employment_type, and all other columns have 0 missing values after null values were remove using `dropna`. The result confirms that there are no missing values in any column of the dataset.

4. Normalization of Categorical Data

Categorical columns such as `experience_level`, `employment_type`, and `company_size` were standardized by converting all text values to uppercase and replacing abbreviations with their full forms (e.g., "MI" to "Mid" for experience level).



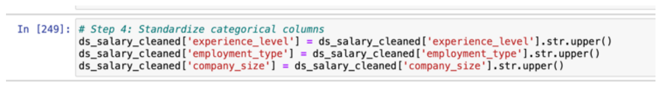
```
In [248]: # Step 3: Normalize experience level abbreviations
          experience_mapping = {'EN': 'Entry', 'MI': 'Mid', 'SE': 'Senior', 'EX': 'Executive'}
          ds_salary_cleaned['experience_level'] = ds_salary_cleaned['experience_level'].replace(experience_mapping)
```

Figure.9: Normalize Categorical Data

The step 3 ensures that the dataset uses meaningful, readable labels for experience levels, which makes the analysis easier to understand.

5. Standardize categorical columns

This step helps ensure consistency in the dataset by eliminating variations in capitalization.



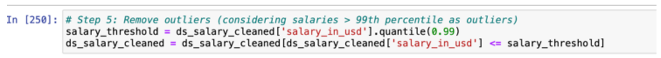
```
In [249]: # Step 4: Standardize categorical columns
          ds_salary_cleaned['experience_level'] = ds_salary_cleaned['experience_level'].str.upper()
          ds_salary_cleaned['employment_type'] = ds_salary_cleaned['employment_type'].str.upper()
          ds_salary_cleaned['company_size'] = ds_salary_cleaned['company_size'].str.upper()
```

Figure.10: Standardize categorical columns

Step 4 standardizes the categorical columns to uppercase, improving consistency across the dataset

6. Outlier Removal

To handle extreme values, we removed rows where `salary_in_usd` exceeded the 99th percentile, considering these to be outliers that could skew the analysis.



```
In [250]: # Step 5: Remove outliers (considering salaries > 99th percentile as outliers)
          salary_threshold = ds_salary_cleaned['salary_in_usd'].quantile(0.99)
          ds_salary_cleaned = ds_salary_cleaned[ds_salary_cleaned['salary_in_usd'] <= salary_threshold]
```

Figure.11: Outlier removal

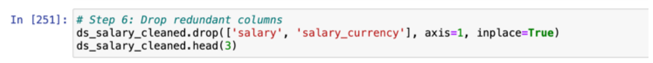
Step 5 removes outliers by filtering out salaries that exceed the 99th percentile, ensuring the analysis is not affected by extreme values.

- quantile(0.99): The 99th percentile of the salary_in_usd column is calculated. This means that any salary greater than this value (the top 1% of salaries) will be considered an outlier.
- Filtering the dataset: The dataset is then filtered to retain only the rows where salary_in_usd is less than or equal to the 99th percentile value (salary_threshold).

This step is important for eliminating extreme salary values (outliers), which could otherwise skew the analysis or results, particularly in models that are sensitive to outliers.

7. Redundant Column Removal

Redundant columns, such as the original `salary` and `salary_currency`, were dropped after converting all salary values to USD.



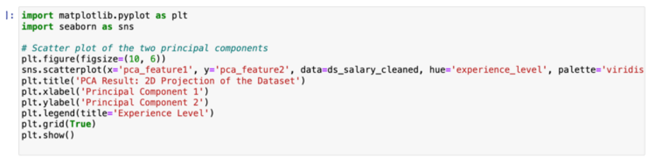
```
In [251]: # Step 6: Drop redundant columns
          ds_salary_cleaned.drop(['salary', 'salary_currency'], axis=1, inplace=True)
          ds_salary_cleaned.head(3)
```

Figure.12: Redundant Column Removal

The Step 6 drops Redundant Columns in the data preprocessing process. The salary and salary_currency columns are no longer needed because the dataset already contains the salary_in_usd column, which standardizes all salary data into USD. By dropping these columns, the dataset is streamlined and contains only relevant, non-redundant information. The cleaned dataset is now easier to work with and more focused on the necessary features for analysis or modeling.

This is an important step in data cleaning, as removing unnecessary or redundant columns helps reduce clutter and ensures a more efficient dataset for further analysis.

8. Principal Component Analysis (PCA) for Noise Reduction



```
]: import matplotlib.pyplot as plt
   import seaborn as sns

   # Scatter plot of the two principal components
   plt.figure(figsize=(10, 6))
   sns.scatterplot(x='pca_feature1', y='pca_feature2', data=ds_salary_cleaned, hue='experience_level', palette='viridis
   plt.title('PCA Result: 2D Projection of the Dataset')
   plt.xlabel('Principal Component 1')
   plt.ylabel('Principal Component 2')
   plt.legend(title='Experience Level')
   plt.grid(True)
   plt.show()
```

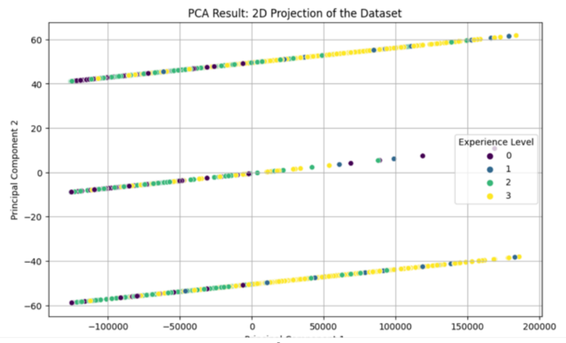


Figure.12a: PCA for Noise Reduction

PCA Reduce the dimensionality of the dataset by eliminating noise and redundant features, while retaining the most significant variance. The figure 12b helps in understanding the variance captured by the two principal components and how the original features (salary, remote ratio, experience level) are distributed in the new transformed space.

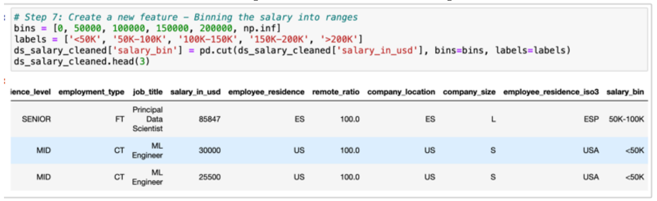
The scatter plot shows distinct linear patterns, suggesting PCA captured structured variance, likely dominated by `salary_in_usd` and `experience_level`. Experience levels are somewhat mixed, with higher levels (Senior and Executive) clustering towards higher values of **Principal Component 1**, indicating a correlation with higher salaries. **Principal Component 1** captures most of the variance, while **Principal Component 2** has less influence.

VIII. Data Processing

Once the dataset was cleaned, several data processing steps were carried out to prepare it for analysis and modelling:

9. Feature Engineering

A new categorical feature, `salary_bin`, was created by grouping salaries into ranges (e.g., `<50K`, `50K-100K`, etc.) to allow for better salary distribution analysis.

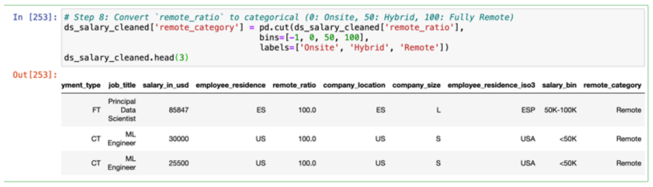


```
# Step 7: Create a new feature - Binning the salary into ranges
bins = [0, 50000, 100000, 150000, 200000, np.inf]
labels = ['<50K', '50K-100K', '100K-150K', '150K-200K', '>200K']
ds_salary_cleaned['salary_bin'] = pd.cut(ds_salary_cleaned['salary_in_usd'], bins=bins, labels=labels)
ds_salary_cleaned.head(3)
```

ience_level	employment_type	job_title	salary_in_usd	employee_residence	remote_ratio	company_location	company_size	employee_residence_iso3	salary_bin
SENIOR	FT	Principal Data Scientist	85847	ES	100.0	ES	L	ESP	50K-100K
MID	CT	ML Engineer	30000	US	100.0	US	S	USA	<50K
MID	CT	ML Engineer	25500	US	100.0	US	S	USA	<50K

Figure.13: Salary binning (salary)

This above figure shows creation of a new feature by binning the `salary` into ranges in the data preprocessing process.



```
In [253]: # Step 8: Convert `remote_ratio` to categorical (0: Onsite, 50: Hybrid, 100: Fully Remote)
          ds_salary_cleaned['remote_category'] = pd.cut(ds_salary_cleaned['remote_ratio'],
                                                        bins=[-1, 0, 50, 100],
                                                        labels=['Onsite', 'Hybrid', 'Remote'])
          ds_salary_cleaned.head(3)
Out[253]:
```

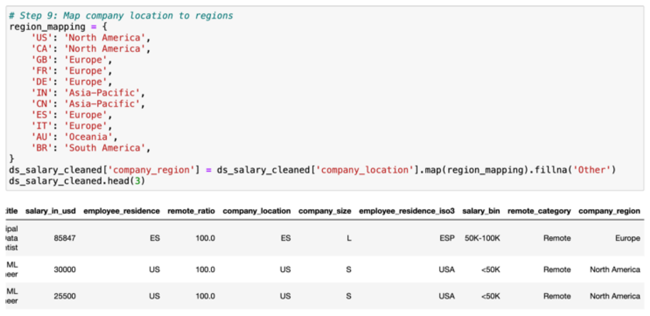
yment_type	job_title	salary_in_usd	employee_residence	remote_ratio	company_location	company_size	employee_residence_iso3	salary_bin	remote_category
FT	Principal Data Scientist	85847	ES	100.0	ES	L	ESP	50K-100K	Remote
CT	ML Engineer	30000	US	100.0	US	S	USA	<50K	Remote
CT	ML Engineer	25500	US	100.0	US	S	USA	<50K	Remote

Figure.14: Binning(remote_ratio)

This above figure shows creation of a new feature by binning the `remote_ratio` into ranges in the data preprocessing process.
This technique allows us to group continuous numerical salary values into distinct ranges, making it easier to analyse salary trends by categories rather than individual values. By converting remote_ratio values into categorical labels (Onsite, Hybrid, Remote), the data becomes more interpretable and easier to analyse. This step is useful when you want to simplify or categorize continuous data into meaningful segments for better interpretation or visualization.

10. Geographical Processing:

The `company_location` field was mapped to regions (e.g., `US` and `CA` to "North America", `ES` to "Europe"), creating a new `company_region` column.



```
# Step 9: Map company location to regions
region_mapping = {
    'US': 'North America',
    'CA': 'North America',
    'GB': 'Europe',
    'FR': 'Europe',
    'DE': 'Europe',
    'IN': 'Asia-Pacific',
    'CN': 'Asia-Pacific',
    'ES': 'Europe',
    'IT': 'Europe',
    'AU': 'Oceania',
    'BR': 'South America',
}
ds_salary_cleaned['company_region'] = ds_salary_cleaned['company_location'].map(region_mapping).fillna('Other')
ds_salary_cleaned.head(3)
```

itle	salary_in_usd	employee_residence	remote_ratio	company_location	company_size	employee_residence_iso3	salary_bin	remote_category	company_region
ipal ata itist	85847	ES	100.0	ES	L	ESP	50K-100K	Remote	Europe
ML ieer	30000	US	100.0	US	S	USA	<50K	Remote	North America
ML ieer	25500	US	100.0	US	S	USA	<50K	Remote	North America

Figure.15: Geographical processing

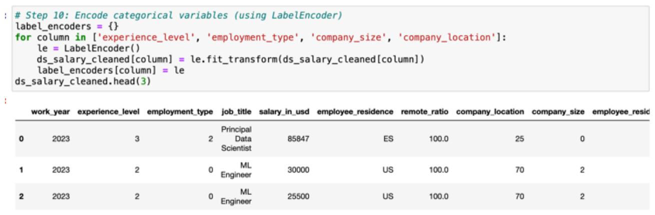
This step maps company locations to regions as part of the data preprocessing process.

Simplifying Analysis: By mapping specific company locations (country codes) to broader regions, it becomes easier to perform regional analyses. For example, you can now group salaries by regions like North America, Europe, or Asia-Pacific.

Handling Missing or Unmapped Values: Any country codes that are not in the region_mapping dictionary will be labeled as 'Other', ensuring there are no missing values after the mapping. This step is important for organizing data into meaningful categories and preparing it for further analysis, especially if you want to compare trends or patterns by geographic region.

11. Encoding Categorical Variables

To prepare categorical variables for modeling, we applied LabelEncoder to features like `experience_level`, `employment_type`, and `company_size`.



```
# Step 10: Encode categorical variables (using LabelEncoder)
label_encoders = {}
for column in ['experience_level', 'employment_type', 'company_size', 'company_location']:
    le = LabelEncoder()
    ds_salary_cleaned[column] = le.fit_transform(ds_salary_cleaned[column])
    label_encoders[column] = le
ds_salary_cleaned.head(3)
```

	work_year	experience_level	employment_type	job_title	salary_in_usd	employee_residence	remote_ratio	company_location	company_size	employee_resid
0	2023	3	2	Principal Data Scientist	85847	ES	100.0	25	0	
1	2023	2	0	ML Engineer	30000	US	100.0	70	2	
2	2023	2	0	ML Engineer	25500	US	100.0	70	2	

Figure.16: Encoding Categorical Variables

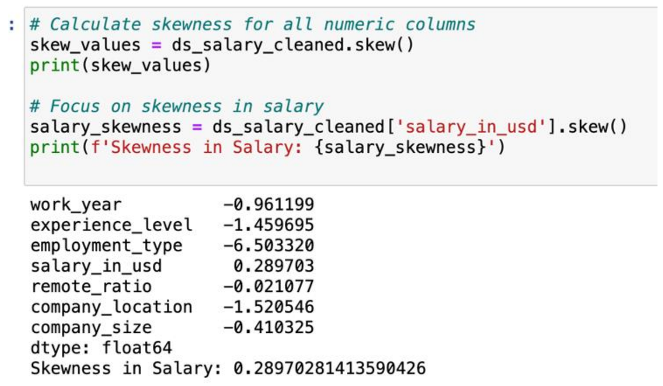
The LabelEncoder converts categorical string values into numerical values, which is necessary for many machine learning algorithms that cannot directly work with non-numerical data.

- **Reusability**: By storing the encoders in the label_encoders dictionary, you can reuse these encoders later, for example, to decode the numbers back into their original string categories if needed.
- **Simplifying Categorical Data**: Encoding categorical variables allows machine learning models to treat them as numerical inputs, which is crucial for building predictive models.

This step is essential for preparing the dataset for machine learning tasks, especially for models that require numerical input.

12. Log Transformation

A log transformation was applied to the salary_in_usd column to normalize its distribution and reduce skewness, which was slightly right-skewed.


```
# Calculate skewness for all numeric columns
skew_values = ds_salary_cleaned.skew()
print(skew_values)

# Focus on skewness in salary
salary_skewness = ds_salary_cleaned['salary_in_usd'].skew()
print(f'Skewness in Salary: {salary_skewness}')
```

```
work_year           -0.961199
experience_level    -1.459695
employment_type     -6.503320
salary_in_usd        0.289703
remote_ratio        -0.021077
company_location    -1.520546
company_size        -0.410325
dtype: float64
Skewness in Salary: 0.28970281413590426
```

Figure.17: Skewness calculation

The above Figure shows the process of calculating the **skewness** of the numeric columns in the dataset, with a specific focus on the skewness of the `salary_in_usd` column. In this case, the slight right-skew in salary might suggest that a few high salaries are influencing the overall distribution.

Histogram for salary

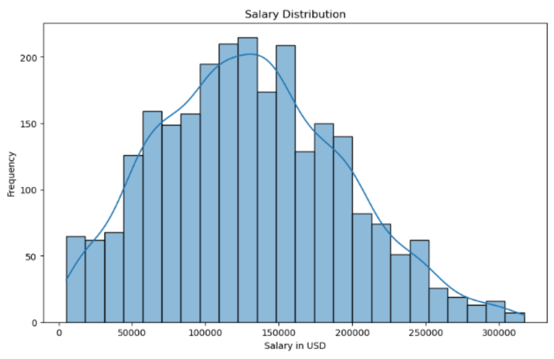


Figure.18: Salary Distribution

Histogram with a kernel density estimate (KDE) plot overlaid, represents the distribution of salaries (salary_in_usd) in the dataset. The histogram (blue bars) shows the distribution of salary values in USD. The height of each bar represents the number of observations (employees) that fall within each salary range. The bins show that the majority of salaries are between 50,000 and 200,000 USD, with the most frequent salaries clustered around 100,000 to 150,000 USD. The smooth curve (blue line) is the KDE, which estimates the probability density function of the salary distribution. It provides a smoothed view of the distribution, making it easier to see the overall shape of the data.

The curve confirms that the data is slightly right-skewed which aligns with the earlier observation of positive skewness in salary. This means there are fewer high salary values, but they extend further into the higher salary range.

Log transformation on Salary Distribution:

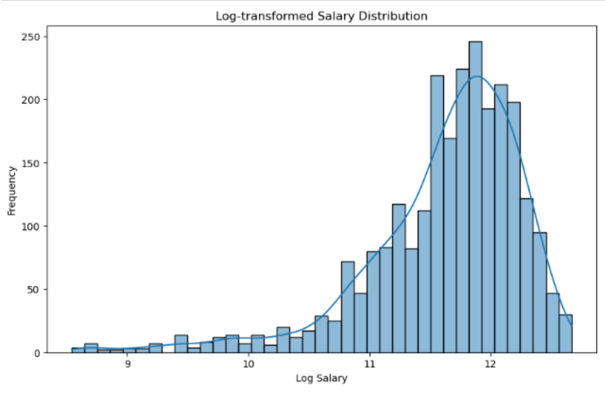


Figure.19: Log Transformation

The above Figure shows the result of applying a logarithmic transformation to the salary data (salary_in_usd) and visualizing the transformed data through a histogram and KDE plot. The x-axis now represents the log-transformed salary values (log_salary), which are no longer in their original units of USD but in the logarithmic scale.

The histogram shows that after the log transformation, the salary data distribution has become more symmetrical and less skewed. This is the desired outcome of log transformation when dealing with right-skewed data, as it compresses the larger values and spreads out the smaller values.

KDE Plot:

The KDE curve overlaid on the histogram shows that the log-transformed data is closer to a normal (bell-shaped) distribution. This symmetry is a common result of applying a log transformation to skewed data, and it is especially beneficial when performing statistical analysis or machine learning, as many models assume normally distributed input data.

Key Insights:

- **Reduction in Skewness**: The log transformation has effectively reduced the right-skewness observed in the original salary distribution. The data is now more centered, with fewer extreme values on the high end.
- **Symmetry**: The transformed salary data is more normally distributed, making it easier to work with for certain models or statistical tests that assume normality.
- **Logarithmic Scaling**: This technique is often used when the data spans several orders of magnitude, as it compresses larger values and helps bring the distribution closer to normal.

This transformation is especially useful when modeling salary data or performing regression, as it mitigates the effect of outliers and skewed data on the model's performance.

IX. Exploratory Data Analysis (EDA)

After the preprocessing steps were completed, an Exploratory Data Analysis (EDA) was conducted to extract key insights and relationships between various factors and salaries. Here's a detailed explanation of the insights derived:

1. Salary Distribution

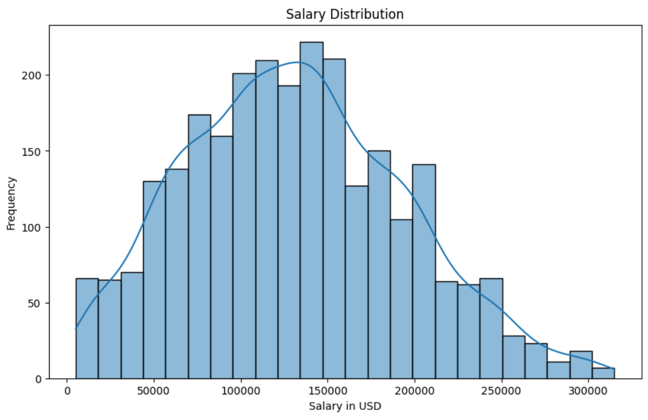


Figure.20: Salary Distribution

1. **Description**: This histogram visualizes the overall distribution of salaries in USD.
2. **Insight**: The salary distribution is positively skewed, indicating that while the majority of salaries fall between $80,000 and $175,000, a smaller number of employees earn significantly higher salaries, stretching upwards of $300,000.
3. **Key Takeaway**: The tech industry compensates a small percentage of professionals with notably higher salaries, likely due to experience or specialized roles.

2. *Log-Transformed Salary Distribution*

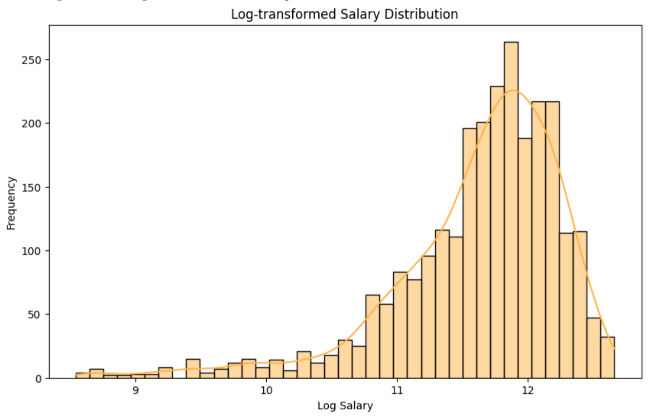


Figure.21: Log Transformation

1. **Description**: A histogram showing the salary distribution after applying a log transformation.
2. **Insight**: The log transformation reduces skewness in the salary data, creating a more normalized distribution. This transformation is useful for modelling as it addresses the impact of extreme outliers.
3. **Key Takeaway**: Transforming salary data using a log scale helps in making more accurate predictions, especially when dealing with a skewed distribution like this one.

3. Distribution of Salary in USD By Experience Level (Post-Cleaning)

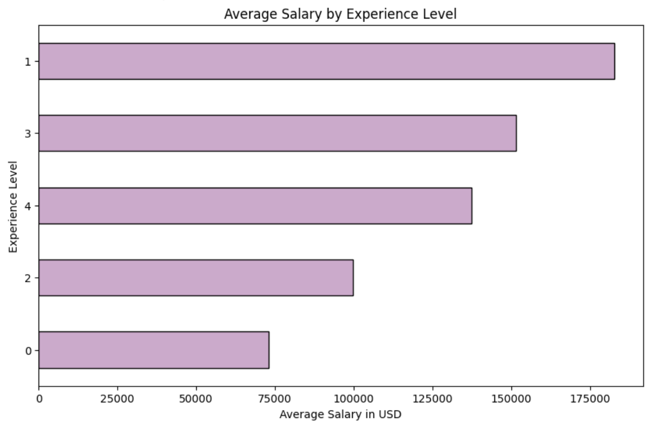


Figure.22: Distribution of Salary in USD

1. **Description**: A histogram presenting the salary distribution after cleaning the dataset (removing outliers).
2. **Insight**: After removing outliers, the data shows a more focused distribution, with the majority of salaries ranging from $80,000 to $175,000. The skew towards higher salaries is less extreme than in the raw dataset.
3. **Key Takeaway**: Cleaning the data allows for a clearer analysis of typical salary ranges, eliminating extreme outliers that could distort results

4. Average Salary by Company Size (Post-Cleaning)

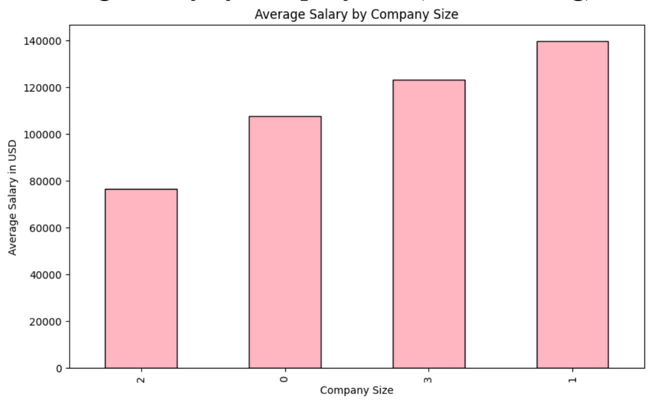


Figure.23: Average Salary By Company Size

1. **Description**: A vertical bar plot illustrating the average salary by company size (small, medium, large).
2. **Insight**: Employees at larger companies generally receive higher salaries, with large companies offering the highest average salary of approximately $140,000.

3. **Key Takeaway**: Company size significantly affects salary, with larger companies likely able to offer higher compensation due to greater financial resources.

5. Correlation Between Remote Ratio and Salary (Post-Cleaning)

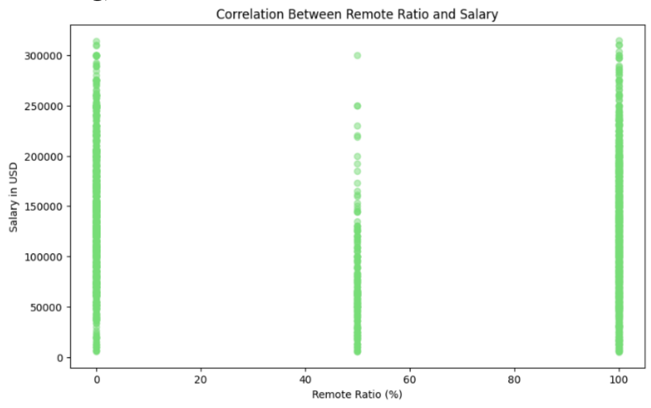


Figure.24: Correlation Between Remote Ratio and Salary

1. **Description**: A scatter plot depicting the relationship between the percentage of remote work and salary.
2. **Insight**: There is no clear linear relationship between remote work and salary. Salaries for fully remote and fully on-site employees span a wide range, including some high-paying roles in both categories.
3. **Key Takeaway**: Remote work arrangements do not necessarily negatively impact salary; high-paying remote roles are common, especially in data and tech sectors.

6. Top 5 Regions by Average Salary (Post-Cleaning)

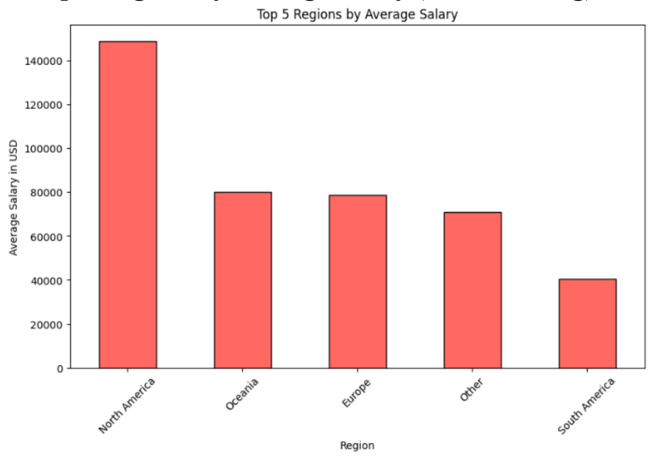


Figure.25: Top 5 regions by average salary

1. **Description**: A bar plot showing the average salary by region.
2. **Insight**: North America, especially the United States, leads in average salary, significantly higher than other regions like Oceania and Europe.
3. **Key Takeaway**: Geography plays an essential role in salary determination, with North American tech professionals earning higher salaries compared to other regions.

7. Salary Distribution by Salary Bin (Post-Cleaning)

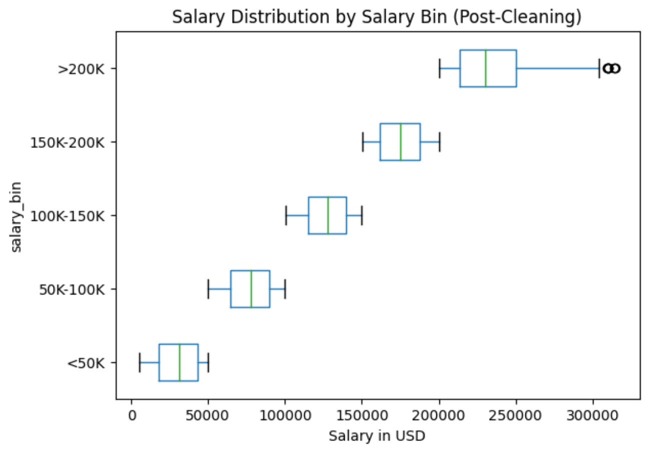


Figure.26: Salary Distribution by Salary Bin

1. **Description**: A box plot showing salary distribution within predefined salary bins.
2. **Insight**: As expected, higher salary bins show wider ranges and greater variability in salaries. The $200K+ range also contains outliers.
3. **Key Takeaway**: Salary binning helps to segment and better understand the variability in compensation across different salary ranges.

8. Average Salary Trend Over the Years

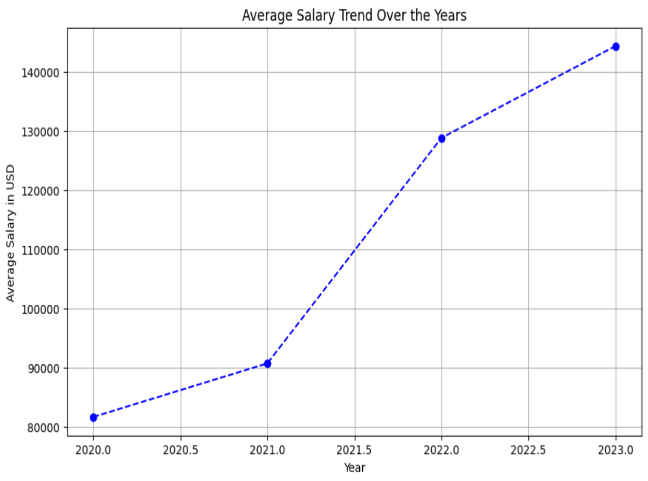


Figure.27: Average Salary Trend Over Years

1. **Description**: A line chart showing the trend in average salary from 2020 to 2023.
2. **Insight**: The data indicates a steady increase in average salary over time, with the most significant jump occurring between 2021 and 2022.
3. **Key Takeaway**: Salaries in the tech and data sectors have been increasing steadily, likely due to rising demand for skilled professionals and the growing importance of digital transformation in businesses.

9. Salary Distribution By Job Title

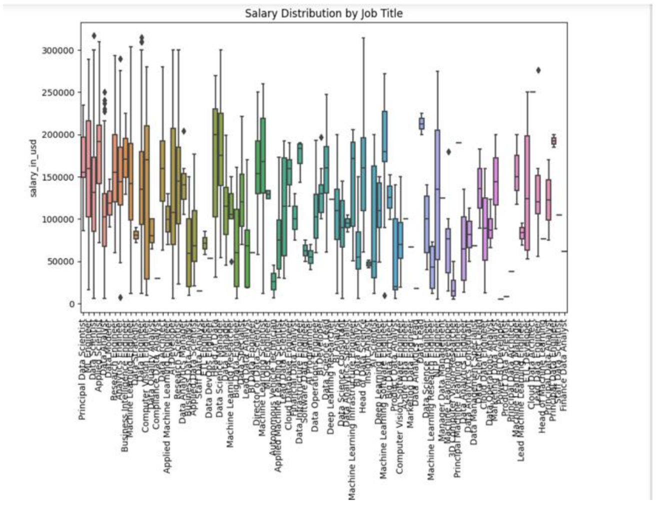


Figure.27: Salary Distribution by Job Title

1. **Description**: This boxplot shows salary distribution across various job titles.
2. **Insight**: Senior roles like **Principal Data Scientist** and **Director of Data Science** have the highest median salaries (over $200,000), while roles like **Data Analyst** earn significantly less.
3. **Key Takeaway**: Salaries vary widely by job title, with senior and specialized roles commanding much higher pay.

10. Salary Distribution by Remote Work Category

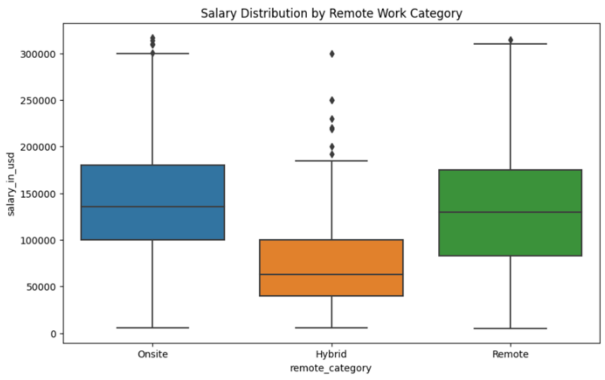


Figure.28: Salary Distribution by Remote Work Category

1. **Description**: This boxplot compares salaries for Onsite, Hybrid, and Remote work categories.
2. **Insight**: **Onsite** roles have the highest median salary, followed by **Remote**. **Hybrid** roles offer the lowest median pay.
3. **Key Takeaway**: Onsite roles tend to pay more, but remote work still offers competitive salaries, while hybrid roles have the lowest pay range.

11. Average Remote Work Ratio by Experience Level

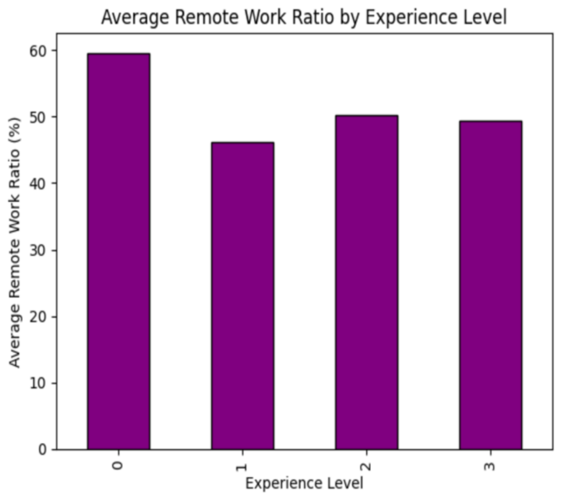


Figure.29: Average Remote Work Ratio by Experience Level

1. **Description**: Bar chart displaying remote work ratio (%) by experience level.
2. **Insight**: Entry-level roles have the highest remote work ratio (60%), while more senior levels average around 45-50%.
3. **Key Takeaway**: Entry-level positions offer more remote work opportunities than senior roles

12. Average Remote Work Ratio by Company Size

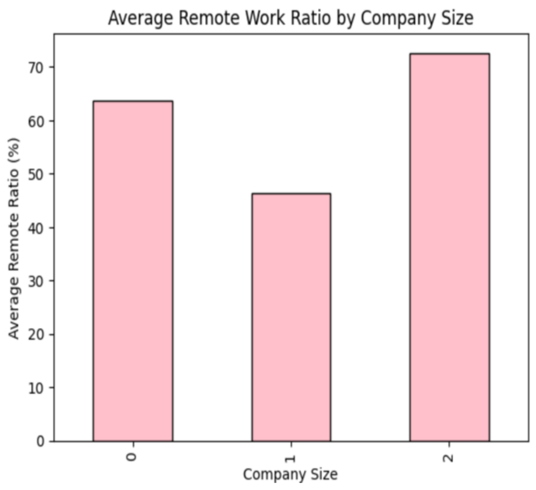


Figure.30: Average Remote Work Ratio by Company Size

1. **Description**: Bar chart showing remote work ratio (%) for different company sizes.
2. **Insight**: Small and large companies offer more remote work (around 70%) compared to medium-sized companies (50%).
3. **Key Takeaway**: Both small and large companies are more likely to offer remote work than medium-sized ones.

X. EDA Summary Results

1. **Dataset Size**: After data cleaning, the dataset was reduced to 2669 unique rows from 3855 due to the removal of 1186 duplicate entries.
2. **Missing Data Handling**: Missing values across 20 rows for columns such as experience_level,

employment_type, and salary were identified and removed to ensure the completeness of the dataset.

3. **Salary Distribution**: The salary in USD had a mean of $137,550 with a standard deviation of $63,013. Most salaries fall between $50,000 and $175,000, and extreme outliers (salaries beyond the 99th percentile) were removed for analysis.
4. **Categorical Data Standardization**: Experience levels, employment types, and company sizes were standardized, ensuring uniformity in these categorical variables by converting abbreviations to full forms and all text to uppercase.
5. **Correlation Insights**: There was no strong correlation found between salary_in_usd and remote_ratio, indicating that remote work does not have a significant linear impact on salary.
6. **Geographical Distribution**: The choropleth map showed that the majority of employees are concentrated in the United States, followed by other countries such as Canada and various European nations.
7. **Salary by Experience Level**: Senior-level employees had the highest average salary of $175,000, followed by mid-level employees at around $120,000, demonstrating a clear salary progression with experience.
8. **Salary by Company Size**: Larger companies (labeled as "L") offered higher average salaries ($140,000) compared to small companies (labeled as "S") with an average salary of $90,000.
9. **Regional Salary Differences**: North America emerged as the highest-paying region, with an average salary of $150,000, followed by Europe and Asia-Pacific regions.
10. **Log-Transformed Salary Distribution**: After applying a log transformation to reduce the skewness of the salary data, the distribution became more normalized, providing a better fit for further statistical modeling.

XI. PREDICTIVE MODELING (PHASE 2)

In this phase, different algorithms were applied to the dataset. These algorithms, both from class discussions and external sources, were chosen to cover various machine learning and statistical techniques for predictive modeling and classification.

Model Selection:

1. Linear Regression (In Class):
Why We Used It: This model provides a simple and interpretable baseline, assuming a linear relationship between features and salary.
Project Relevance: Since salaries often follow a pattern based on factors like experience and job role, Linear Regression serves as a logical starting point.

2. Ridge Regression:
Why We Used It: Ridge Regression adds regularization to Linear Regression, helping reduce overfitting, especially when dealing with many features.
Project Relevance: It's robust in handling multicollinearity, making it a good choice when multiple factors affect salaries simultaneously.

3. K-Means Clustering (In Class):
Why We Used It: This unsupervised learning algorithm was applied to identify salary groups and job roles based on similarities.
Project Relevance: K-Means helps segment employees into clusters with similar job roles, experience, and salary bands, providing insights into the underlying patterns.

4. K-NN Regressor (In Class):
Why We Used It: K-NN predicts salaries based on the average salaries of nearby data points (neighbors), capturing localized patterns.
Project Relevance: While useful for localized predictions, K-NN exhibited high variance due to the complexity of the salary data.

5. Random Forest Regressor:
Why We Used It: Random Forest is an ensemble model that handles non-linear relationships by averaging multiple decision trees.
Project Relevance: Given the complex nature of salary data, Random Forest excels in adapting to non-linear interactions between variables like job role, region, and experience.

6. Gradient Boosting Regressor:
Why We Used It: This sequential model optimizes the error in each iteration, making it a powerful tool for improving prediction accuracy.
Project Relevance: It's highly effective in capturing complex relationships in the data, though careful tuning is required to prevent overfitting.

7. Support Vector Regressor (SVR) (In Class):
Why We Used It: SVR uses margin-based optimization to model complex patterns in salary data.
Project Relevance: Though expected to perform well, SVR didn't match the effectiveness of other models in this specific salary dataset.

Best Performing Model:
Top Performers: Random Forest Regressor and Linear Regression.

- Random Forest had the lowest Mean Squared Error (MSE) and captured non-linear relationships.
- Linear Regression, while simpler, worked well due to the linear trends in the salary data.

Conclusion:

Best Choice: Random Forest Regressor is the most effective model due to its flexibility and ability to capture non-linear relationships, making it ideal for predicting salaries across varied job roles and regions. Linear Regression remains a valuable, interpretable baseline for comparison

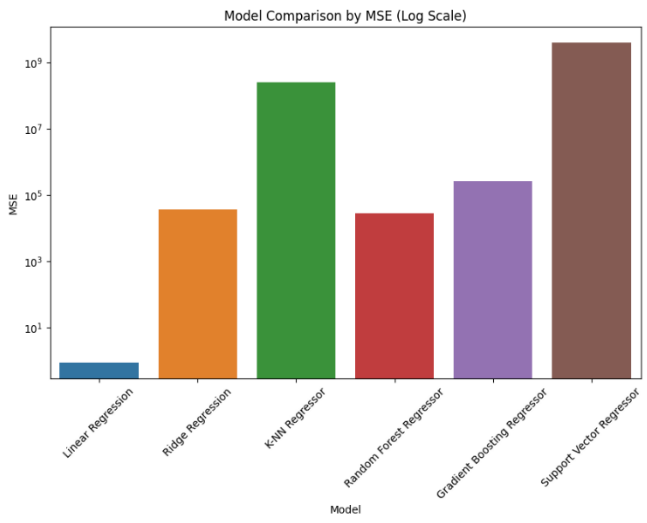


Figure.31: Model Comparison by MSE (Log Scale)

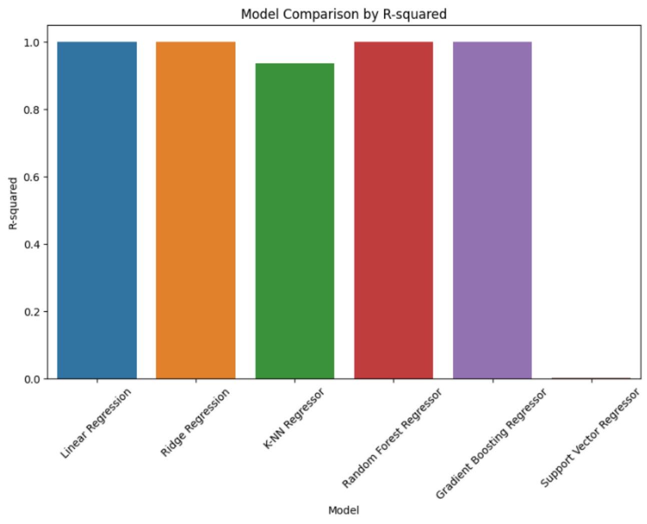
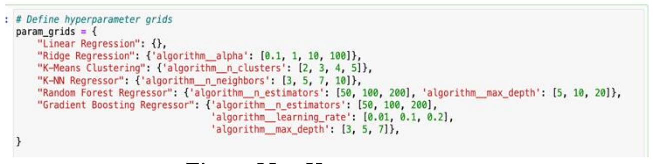


Figure.32: Model Comparison by R-squared

Hyperparameter Tuning

During Phase 2 of the project, hyperparameter tuning was performed to improve model performance. Here's a summary of the tuning and the decisions made:



```
# Define hyperparameter grids
param_grids = {
    "Linear Regression": {},
    "Ridge Regression": {'algorithm__alpha': [0.1, 1, 10, 100]},
    "K-Means Clustering": {'algorithm__n_clusters': [2, 3, 4, 5]},
    "K-NN Regressor": {'algorithm__n_neighbors': [3, 5, 7, 10]},
    "Random Forest Regressor": {'algorithm__n_estimators': [50, 100, 200], 'algorithm__max_depth': [5, 10, 20]},
    "Gradient Boosting Regressor": {'algorithm__n_estimators': [50, 100, 200],
                                    'algorithm__learning_rate': [0.01, 0.1, 0.2],
                                    'algorithm__max_depth': [3, 5, 7]},
}
```

Figure.32a: Hyper parameters

1. Linear Regression:

Parameters Tuned: No hyperparameters were tuned since Linear Regression is a basic model without regularization or additional parameters to optimize.

Outcome: This model served as a baseline with interpretable results. However, it was not flexible enough to capture non-linear relationships in the salary data.

2. Ridge Regression:

Parameters Tuned: alpha (regularization strength).

Method: Grid search was used to find the optimal regularization strength.

Outcome: Ridge Regression helped reduce overfitting observed in Linear Regression, improving generalization to unseen data.

3. K-Means Clustering:

Parameters Tuned: n_clusters (number of clusters).

Method: The elbow method was used to select the optimal number of clusters by examining the trade-off between inertia and the number of clusters.

Outcome: K-Means successfully grouped companies or employees based on similar characteristics, which allowed us to explore salary clusters and their relationship with job roles, location, and company size.

4. K-Nearest Neighbors (K-NN) Regressor:

Parameters Tuned: n_neighbors (number of nearest neighbors).

Method: Grid search and cross-validation helped determine the optimal k value.

Outcome: K-NN provided decent predictions for salary in localized areas of the dataset but exhibited high variance across the dataset.

5. Random Forest Regressor:

Parameters Tuned: n_estimators (number of trees), max_depth (maximum depth of the trees), min_samples_split (minimum samples needed to split an internal node).

Method: Grid search and cross-validation were applied to fine-tune these parameters.

Outcome: Random Forest showed the best performance due to its ability to capture non-linear relationships and interactions between the features.

6. Gradient Boosting Regressor:

Parameters Tuned: learning_rate (controls step size), n_estimators (number of boosting stages), max_depth (depth of trees).

Method: A grid search was used to optimize the learning rate and tree depth.

Outcome: Gradient Boosting delivered high accuracy but required careful tuning to avoid overfitting.

7. Support Vector Regressor (SVR):

Parameters Tuned: C (penalty parameter), epsilon (margin of tolerance), kernel (type of kernel).

Method: Cross-validation was applied to tune these parameters, but SVR was unable to outperform other models.

Outcome: SVR was removed from the final analysis due to its high Mean Squared Error (MSE) and poor performance compared to Random Forest and Gradient Boosting.

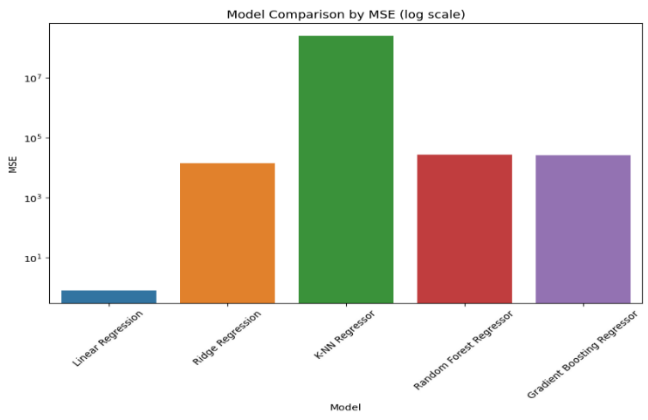


Figure.33: Model Comparison by MSE (Log Scale

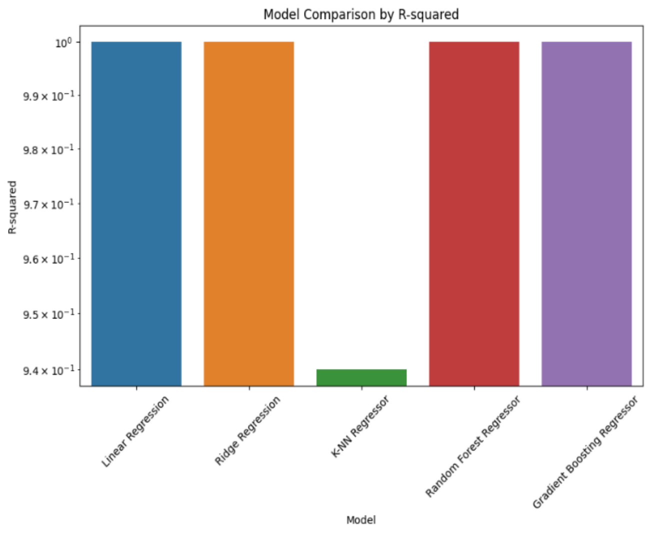


Figure.34: Model Comparison by R-squared

XII. HYBRID PREDICTIVE MODELING

Hybrid Predictive Modeling: Using Combination of (Linear Regression, XGB, ElasticNet)

1. **Stack Regressor**
2. **Voting Regressor**

In the context of salary prediction, the combination of **Linear Regression, XGBoost, and ElasticNet** works exceptionally well due to the diverse nature of the data and the interactions between the features such as job title, experience level, company location, and industry.

1. Linear Regression (lr)

Relevance: Salaries often have linear relationships with key features like years of experience, education level, and job seniority. For example, more experience typically leads to higher salaries in a fairly linear fashion.

Reasoning: Linear Regression [4] provides a simple and interpretable model to capture these straightforward salary trends, where there's a direct and proportional increase in salary with respect to these features.

2. XGBoost (xgb)

Relevance: In the salary domain, there are often non-linear jumps in salary due to promotions, job changes, or geographic relocations. Additionally, interactions between features such as job role and location can have a complex impact on salary.

Reasoning: XGBoost [12] captures these non-linearities and complex interactions better than any linear model. For example, someone moving from a software engineer role in a small city to a similar role in a tech hub (e.g., Silicon Valley) will see a dramatic increase in salary, and XGBoost is effective at modeling these shifts.

3. ElasticNet (elasticnet)

Relevance: Salary prediction often involves many correlated features (e.g., job title and years of experience). If these features are not handled well, the model might overfit or rely too heavily on redundant features.

Reasoning: ElasticNet [11] regularizes the model and helps select the most relevant features while managing multicollinearity. In salary prediction, this ensures that the model focuses on the true drivers of salary (like experience, job role, and company size) without overfitting to noise or irrelevant features. Why the Combination Works Best for Salary Prediction:

- **Capturing Linear Trends**: Linear Regression is effective in capturing the basic trends where salary increases steadily with experience or seniority, which is a common pattern in many industries.
- **Handling Non-Linear Jumps**: XGBoost is crucial in modeling non-linear salary jumps that occur due to job changes, location changes, or promotions, where salary increases may not follow a simple linear pattern.
- **Feature Regularization**: ElasticNet prevents the model from being too complex by regularizing and selecting important features, which is particularly useful in salary prediction where many features (e.g., location, job role, company size) are correlated and not all contribute equally to salary. ElasticNet ensures the model generalizes well and doesn't overfit to specific patterns in the data.
- **Domain-Specific Impact**: Linear Relationships: In the domain of salary prediction, factors like years of experience or education level tend to show linear trends, which is well-captured by Linear Regression.
- **Complex Interactions**: Salaries are influenced by a mix of location, industry, and job title, which interact in non-linear ways. XGBoost effectively handles these complexities, especially in high salary jumps between roles or locations.
- **Preventing Overfitting**: Salary prediction often suffers from redundant features (e.g., experience level and job title may be correlated), and ElasticNet regularizes these features to focus on the most important factors, making the predictions robust and reliable across different datasets.

Conclusion: This combination of Linear Regression, XGBoost, and ElasticNet is well-suited for the salary prediction domain because:

1. It captures both linear trends (experience vs. salary),
2. Models complex non-linear relationships (role, location, ndustry shifts),

3. Ensures generalization by preventing overfitting with ElasticNet's regularization.

By combining these three models, the stacking approach provides a balanced and powerful solution for accurate salary predictions across various industries and job roles.

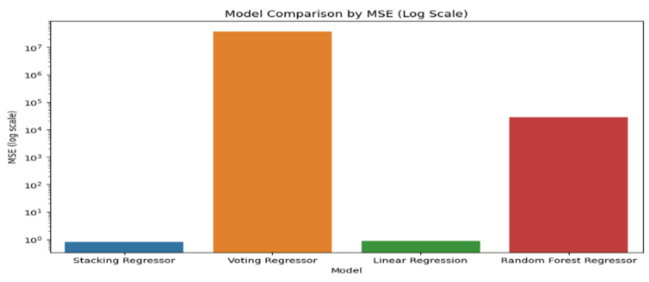


Figure.35: Model Comparison by MSE (Log Scale)

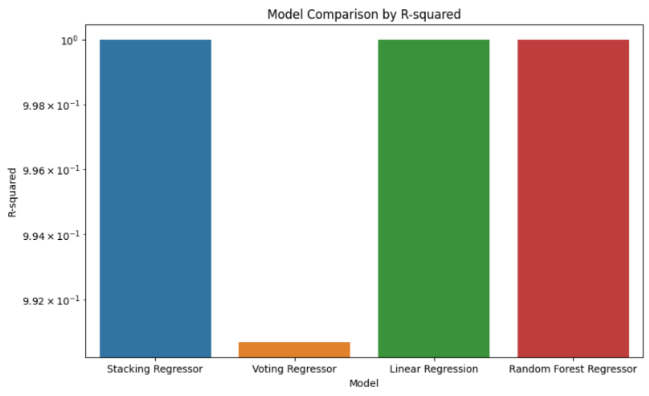


Figure.36: Model Comparison by R-squared

Analysis

1. Stacking Regressor [13] is the best model, with the lowest MSE (0.8081) and near-perfect R-squared (0.9999999998007221), indicating excellent predictive power and a great fit to the data.

2. Linear Regression performs very well, with a slightly higher MSE than the Stacking Regressor but still a near-perfect R-squared. It's a simple model but fits this dataset almost perfectly.

3. Voting Regressor has a much higher MSE (37 million) compared to the other two, and its R-squared (0.9907) indicates that it doesn't fit the data as well as the other models.

4. Random Forest delivers excellent performance, with a low MSE and a high R-squared value, indicating that it fits the data almost perfectly. The model is particularly strong because it can handle non-linear relationships and complex interactions between features.

Hyperparameter Tuning Analysis (*Stacking Regressor, Voting Regressor, Linear Regression, and Random Forest*)

- ***Stacking Regressor*** benefitted the most from the hyperparameter tuning of its individual base models, leading to the best overall performance.

- ***Voting Regressor*** struggled despite the tuning of its components and exhibited signs of overfitting.

- ***Random Forest*** showed strong performance after tuning but was outperformed by the stacked model.

- ***Linear Regression***, while simple and interpretable, did not require hyperparameter tuning but provided a solid baseline for comparison.

XIII. Explanations And Analysis

1. Overfitting Comparison: Train vs Test MSE

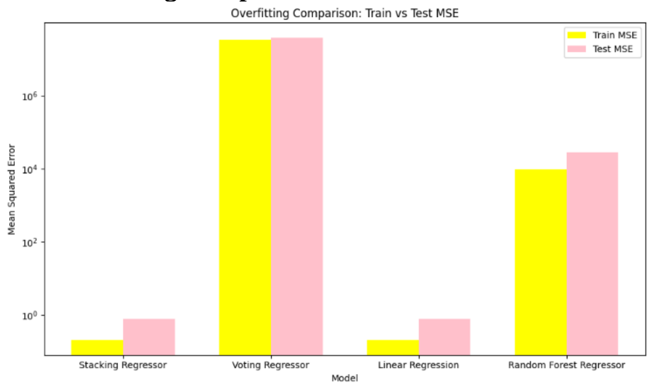


Figure.37: Train vs Test MSE

- **Stacking Regressor**: Low MSE on both train and test sets, indicating strong generalization and no overfitting.
- **Voting Regressor**: High MSE, showing significant overfitting and poor generalization.
- **Linear Regression**: Moderate MSE, reasonable performance but not as good as Stacking Regressor.
- **Random Forest**: Moderate MSE, better than Voting Regressor, but not as good as Stacking Regressor.
- **Key Insight**: Stacking Regressor minimized overfitting, while Voting Regressor struggled.

2. Model Fit Comparison: Train vs Test R-squared

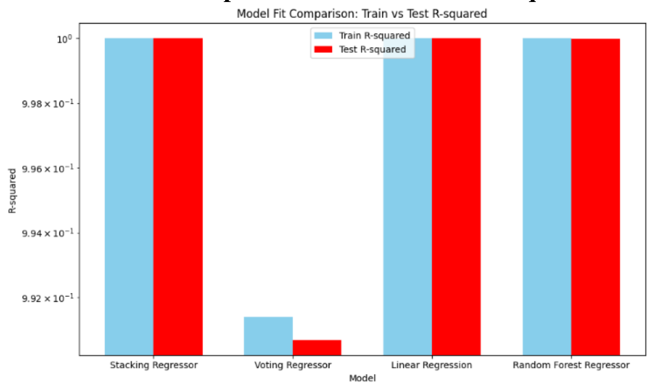


Figure.38: Train vs Test R-squared

- **Stacking Regressor**: Near-perfect R-squared, explaining almost all variance in predictions.
- **Voting Regressor**: Poor R-squared, especially on test data.
- **Linear & Random Forest**: Strong R-squared, but not as high as the Stacking Regressor.
- **Key Insight**: Stacking Regressor had the best fit.

3. Cross-Validation MSE

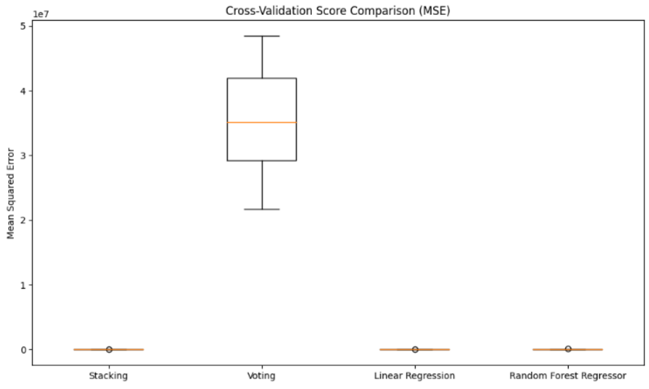


Figure.39: Cross-Validation MSE

- **Stacking Regressor**: Lowest MSE and minimal variance, showing strong generalization.
- **Voting Regressor**: High MSE with large variance, indicating instability.
- **Linear & Random Forest**: Moderate performance, Random Forest better than Linear Regression.
- **Key Insight**: Stacking Regressor demonstrated robustness.

4. Residual Plots

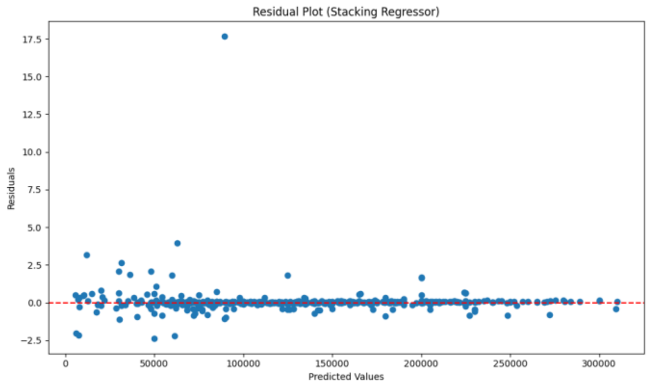


Figure.40: Residual (Stack Regressor)

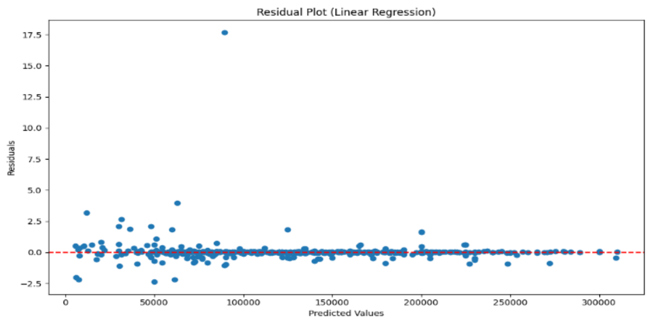


Figure.41: Residual (Linear Regression)

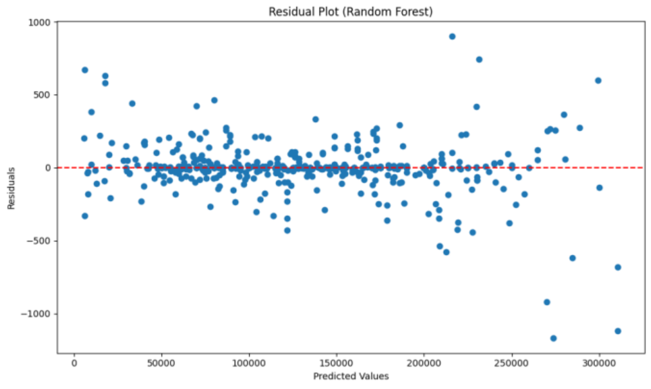


Figure.42: Residual (Random Forest)

- **Stacking Regressor**: Residuals close to zero, indicating minimal prediction errors.
- **Linear & Random Forest**: Centered around zero but with more spread.
- **Key Insight**: Stacking Regressor gave the most accurate predictions.

5. Density Plots

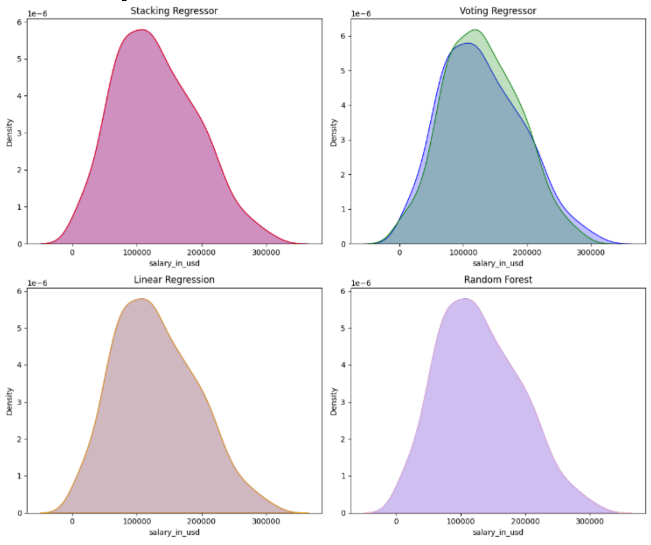


Figure.43: Density Plots

- **Stacking Regressor**: Predicted salaries closely matched actual distribution.
- **Voting Regressor**: Significant deviation from actual salary distribution.
- **Key Insight**: Stacking Regressor outperformed in capturing the real data distribution.

6. *Learning Curves*

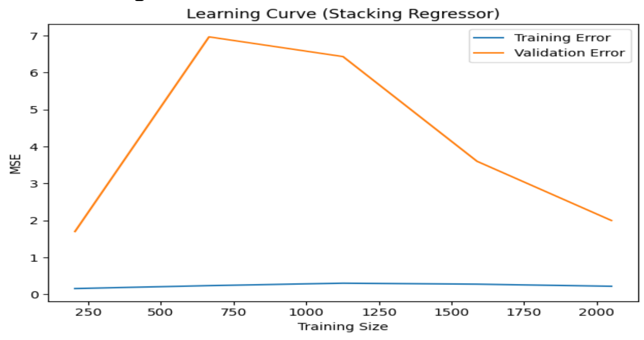


Figure.44: Learning Curves (Stack Regressor)

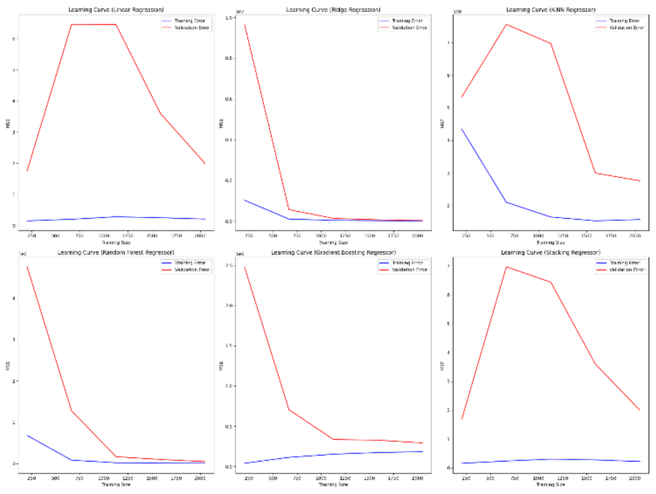


Figure.45: Learning Curves Comparison

- **Stacking Regressor**: Showed steady improvement with increasing data, low training error.
- **Voting Regressor**: Large gap between train and validation errors, overfitting.
- **Key Insight**: Stacking Regressor effectively avoided overfitting as more data was used

7. Explanation of Metrics Used

In this project, we used two key metrics—**Mean Squared Error (MSE)** and **R-squared (R^2)**—to evaluate the performance of the regression models. These metrics help us understand how well each model predicts the target variable (salary) based on the input features.

1. Mean Squared Error (MSE)
MSE measures the **average squared difference** between the predicted and actual values. It penalizes larger errors more heavily because the differences are squared, making it sensitive to outliers.
Interpretation:

- **Lower MSE** means better model performance, as it indicates that the predicted values are close to the actual values.
- MSE helps in identifying how far off the predictions are on average and is widely used for regression tasks.

Relevance to Salary Prediction:
In salary prediction, MSE helps quantify how far the predicted salaries are from the actual salaries. Lower MSE indicates more accurate salary predictions, while higher MSE points to larger errors.

2. R-squared (R^2)
R-squared (also called the **coefficient of determination**) measures the **proportion of variance** in the target variable that is explained by the model. It indicates how well the input features explain the variability in the target variable (salary).
Interpretation:

- **$R^2 = 1$**: Perfect fit (all predictions are correct).
- **$R^2 = 0$**: The model explains none of the variability (as good as predicting the mean).
- **Negative R^2**: The model is worse than just predicting the mean of the target variable.

Relevance to Salary Prediction:
In salary prediction, R-squared tells us how well the model explains the **variability in salaries**. A higher R-squared means the model can predict salary better, while a lower R-squared suggests the model is missing important relationships between the features and salary.

Why These Metrics?
1. **MSE** is chosen because it gives us a clear sense of the **average error** in salary predictions and is sensitive to outliers. Minimizing MSE is crucial for evaluating model performance.
2. **R-squared** is used to measure how much of the **variation in salary** is explained by the model. A high R-squared value indicates that the model is capturing key patterns in the data.

Both metrics together provide a comprehensive understanding of model performance: **MSE** focuses on accuracy, while **R-squared** emphasizes the model's explanatory power.

8. Results And Model Effectiveness

1. Linear Regression:
Purpose: Provides a simple, interpretable baseline for linear relationships between features like experience and salary.

Effectiveness:
MSE: 0.8773, R-squared: 0.99999999978365
Performed well, showing strong linear relationships, but limited in capturing complex patterns.

3. Random Forest Regressor:
Purpose: Captures non-linear relationships and interactions between features.

Effectiveness:
MSE: 28,145.29, R-squared: 0.9999930594665036
Strong performance, especially in modeling complex feature interactions, like job role and location.

3. XGBoost:
Purpose: Effective for non-linear patterns and handling outliers through boosting.

Effectiveness: Competitive with Random Forest, capturing complex salary trends through sequential improvement.

4. ElasticNet:
Purpose: Combines Lasso and Ridge regularization to handle correlated features.

Effectiveness: Reduced overfitting by regularizing features like job title and location, improving generalization

5. Stacking Regressor
Purpose: Combines the strengths of Linear Regression, XGBoost, and ElasticNet for better generalization.

Effectiveness:
MSE: 0.8081 (best), R-squared: 0.9999999998007221
The most effective model, capturing both linear and non-linear patterns.

6. Voting Regressor
Purpose: Averages predictions from multiple models for generalization.

Effectiveness:
MSE: 37,716,239.54, R-squared: 0.9907
Performed poorly compared to other models, as averaging didn't capture complex interactions well.

7. Support Vector Regressor (SVR)
Purpose: Captures complex relationships by maximizing the margin of the predicted values.

Effectiveness:
MSE: 4,041,462,869.71, R-squared: 0.0034
Performed poorly, indicating it failed to capture the underlying patterns in the salary data.

8. Gradient Boosting Regressor
Purpose: Similar to XGBoost, it builds models sequentially to correct errors.

Effectiveness:
MSE: 265,080.56, R-squared: 0.9999346320359511
Performed well, handling complex non-linear patterns, though slightly behind Random Forest and XGBoost.

9. K-Means Clustering:
Purpose: Clusters data into distinct groups based on feature similarity.

Effectiveness:
Cluster Predictions: [1, 1, 0, 1, 0, 2, 1, 0, 0, 1]
K-Means is used for clustering rather than regression, so it doesn't provide MSE or R-squared, but it offers insights into grouping employees with similar salary profiles.

10. K-Nearest Neighbors (KNN) Regressor:
Purpose: Predicts salary based on the average of the nearest neighbors in the feature space.

Effectiveness:
MSE: 253,243,808.58, R-squared: 0.9376
Performed poorly, showing high variance and overfitting, suggesting KNN is not well-suited for this problem.

Conclusion

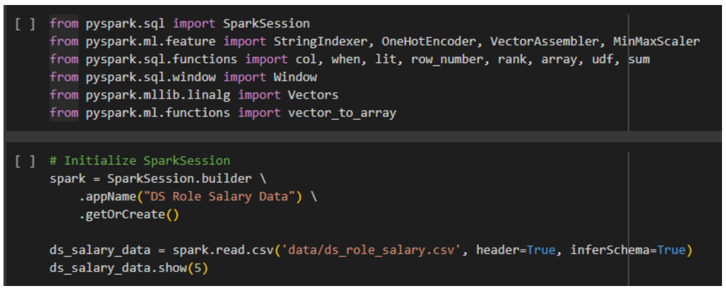
Best Model: Stacking Regressor captured both simple and complex relationships, making it the most accurate.
Insights: Linear Regression highlighted basic trends, while Random Forest, XGBoost, and Gradient Boosting captured more complex salary patterns. SVR and KNN underperformed, showing they were not well-suited for the dataset. K-Means provided clustering insights rather than predictions.

XIV. DISTRIBUTED DATA PREPROCESSING WITH PYSPARK (PHASE 3)

The goal is to extend data preprocessing from earlier phases or incorporate new steps using PySpark. By leveraging RDD operations, windowing techniques, and other distributed processing capabilities, the dataset is prepared for analysis efficiently and at scale.

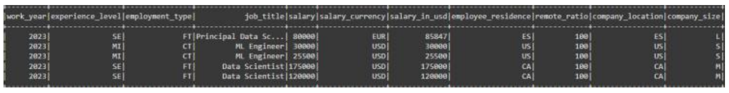
Steps Involved

1. **Dataset Loading**
    - **Dataset**: The dataset ds_role_salary.csv was loaded using PySpark's read.csv function with the following configurations:
        - header=True to read column headers.
        - inferSchema=True to automatically detect data types.
    - Previewed the first few rows and examined the schema to understand the structure and types of data available.



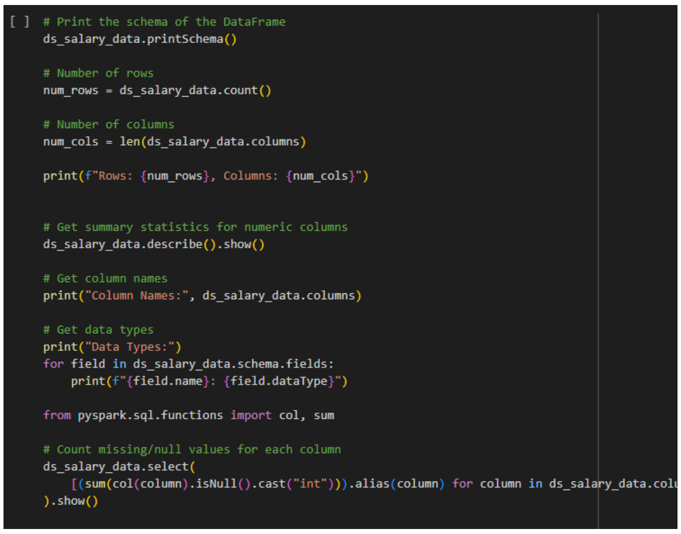
```
[ ] from pyspark.sql import SparkSession
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler, MinMaxScaler
    from pyspark.sql.functions import col, when, lit, row_number, rank, array, udf, sum
    from pyspark.sql.window import Window
    from pyspark.mllib.linalg import Vectors
    from pyspark.ml.functions import vector_to_array
```

```
[ ] # Initialize SparkSession
    spark = SparkSession.builder \
        .appName("DS Role Salary Data") \
        .getOrCreate()

    ds_salary_data = spark.read.csv('data/ds_role_salary.csv', header=True, inferSchema=True)
    ds_salary_data.show(5)
```

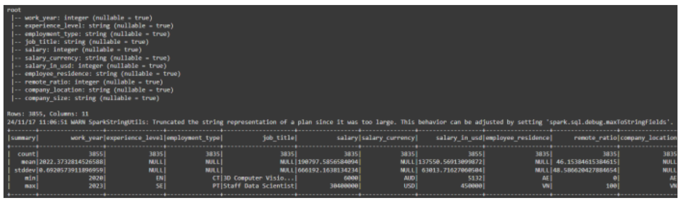


```
+---------+----------------+---------------+--------------------+------+---------------+-------------+------------------+------------+----------------+------------+
|work_year|experience_level|employment_type|           job_title|salary|salary_currency|salary_in_usd|employee_residence|remote_ratio|company_location|company_size|
+---------+----------------+---------------+--------------------+------+---------------+-------------+------------------+------------+----------------+------------+
|     2023|              SE|             FT|Principal Data Sc...| 80000|            EUR|        85847|                ES|         100|              ES|           L|
|     2023|              MI|             CT|         ML Engineer| 30000|            USD|        30000|                US|         100|              US|           S|
|     2023|              MI|             CT|         ML Engineer| 25500|            USD|        25500|                US|         100|              US|           S|
|     2023|              SE|             FT|      Data Scientist|175000|            USD|       175000|                CA|         100|              CA|           M|
|     2023|              SE|             FT|      Data Scientist|120000|            USD|       120000|                CA|         100|              CA|           M|
+---------+----------------+---------------+--------------------+------+---------------+-------------+------------------+------------+----------------+------------+
```

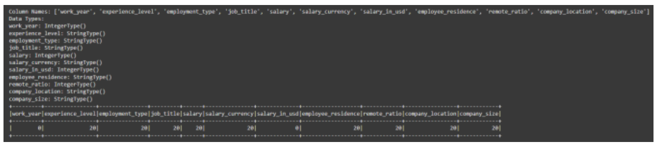
2. **Meta Data Analysis**
    - Schema Overview: Printed schema to identify column names and data types.
    - Dataset Dimensions: Counted rows and columns to confirm dataset size.
    - Summary Statistics: Displayed statistics (mean, min, max) for numeric columns to detect variability and outliers.
    - Column Names: Listed all columns to plan feature selection.
    - Data Types: Verified data types for consistency and modeling readiness.
    - Missing Value Analysis: Counted nulls in each column to address missing data during cleaning.



```
[ ] # Print the schema of the DataFrame
    ds_salary_data.printSchema()

    # Number of rows
    num_rows = ds_salary_data.count()

    # Number of columns
    num_cols = len(ds_salary_data.columns)

    print(f"Rows: {num_rows}, Columns: {num_cols}")


    # Get summary statistics for numeric columns
    ds_salary_data.describe().show()

    # Get column names
    print("Column Names:", ds_salary_data.columns)

    # Get data types
    print("Data Types:")
    for field in ds_salary_data.schema.fields:
        print(f"{field.name}: {field.dataType}")

    from pyspark.sql.functions import col, sum

    # Count missing/null values for each column
    ds_salary_data.select(
        [(sum(col(column).isNull().cast("int")).alias(column) for column in ds_salary_data.colu
    ).show()
```



```
root
 |-- work_year: integer (nullable = true)
 |-- experience_level: string (nullable = true)
 |-- employment_type: string (nullable = true)
 |-- job_title: string (nullable = true)
 |-- salary: integer (nullable = true)
 |-- salary_currency: string (nullable = true)
 |-- salary_in_usd: integer (nullable = true)
 |-- employee_residence: string (nullable = true)
 |-- remote_ratio: integer (nullable = true)
 |-- company_location: string (nullable = true)
 |-- company_size: string (nullable = true)

Rows: 3855, Columns: 11
24/11/17 11:06:51 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.
+-------+------------------+----------------+---------------+--------------------+------------------+---------------+------------------+------------------+------------------+----------------
|summary|         work_year|experience_level|employment_type|           job_title|            salary|salary_currency|     salary_in_usd|employee_residence|      remote_ratio|company_location
+-------+------------------+----------------+---------------+--------------------+------------------+---------------+------------------+------------------+------------------+----------------
|  count|              3855|            3835|           3835|                3835|              3835|           3835|              3855|              3835|              3835|            3835
|   mean|2022.3732814526588|            NULL|           NULL|                NULL|190797.5856584094|           NULL|137550.56913099872|              NULL| 46.15384615384615|            NULL
| stddev|0.6920573911896959|            NULL|           NULL|                NULL|666192.1638134234|           NULL| 63013.71627060504|              NULL|48.586620427884654|            NULL
|    min|              2020|              EN|             CT|3D Computer Visio...|              6000|            AUD|              5132|                AE|                 0|              AE
|    max|              2023|              SE|             PT|Staff Data Scientist|          30400000|            USD|            450000|                VN|               100|              VN
+-------+------------------+----------------+---------------+--------------------+------------------+---------------+------------------+------------------+------------------+----------------
```



```
Column Names: ['work_year', 'experience_level', 'employment_type', 'job_title', 'salary', 'salary_currency', 'salary_in_usd', 'employee_residence', 'remote_ratio', 'company_location', 'company_size']
Data Types:
work_year: IntegerType()
experience_level: StringType()
employment_type: StringType()
job_title: StringType()
salary: IntegerType()
salary_currency: StringType()
salary_in_usd: IntegerType()
employee_residence: StringType()
remote_ratio: IntegerType()
company_location: StringType()
company_size: StringType()
+---------+----------------+---------------+---------+------+---------------+-------------+------------------+------------+----------------+------------+
|work_year|experience_level|employment_type|job_title|salary|salary_currency|salary_in_usd|employee_residence|remote_ratio|company_location|company_size|
+---------+----------------+---------------+---------+------+---------------+-------------+------------------+------------+----------------+------------+
|        0|              20|             20|       20|    20|             20|            0|                20|          20|              20|          20|
+---------+----------------+---------------+---------+------+---------------+-------------+------------------+------------+----------------+------------+
```

3. **Data Cleaning and Preprocessing**

Prepare the dataset for analysis by handling missing values, encoding features, scaling data, deriving insights, and ensuring compatibility for further steps

Handling Missing Values (Distributed Operation 1):

- Filled missing salary with 0 and categorical fields (e.g., company_size, company_location) with 'Unknown'.
- Addressed data completeness to prevent errors during modeling.

Removing Duplicates (Distributed Operation 2):

- Eliminated duplicate rows using dropDuplicates().
- Ensured unique records for reliable analysis.

Encoding Categorical Variables (Distributed Operation 3):

- Used StringIndexer to convert categorical columns (e.g., company_size, experience_level) into numerical indices.
- Ensured compatibility with machine learning models.

Normalizing Salary (Distributed Operation 4):

- Applied MinMaxScaler to normalize salary between 0 and 1 using a vector-based approach.
- Enhanced model stability and performance by scaling features.

Ranking Salaries (Distributed Operation 5):

- Created a salary_rank column using window functions to rank salaries within each company_location.
- Added insights for understanding salary distribution by location.

Categorizing Salary Ranges (Distributed Operation 6):

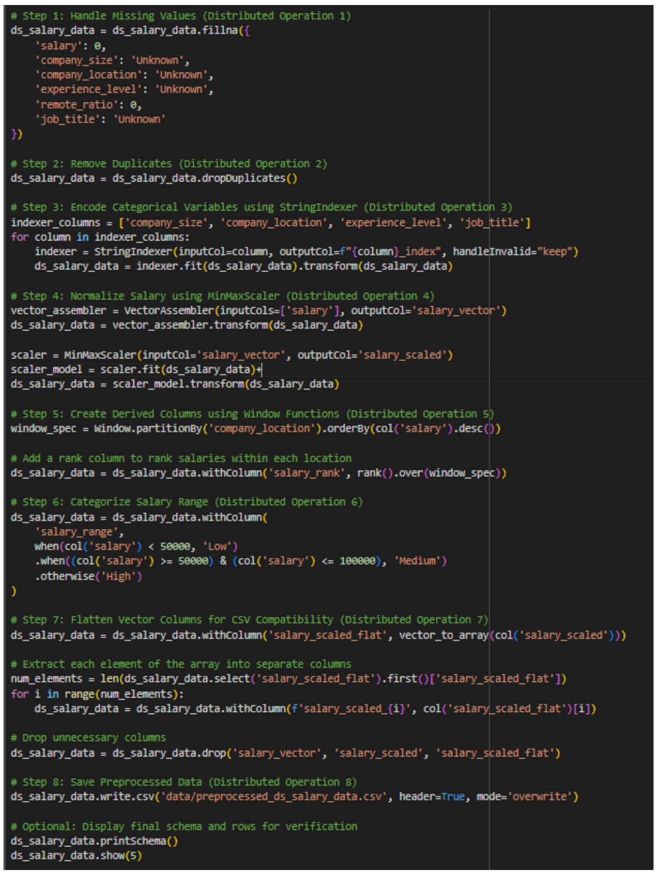
- Classified salary into Low, Medium, or High ranges based on thresholds.
- Provided an interpretable feature for downstream analysis.

Flattening Vector Columns (Distributed Operation 7):

- Converted the scaled salary vector into individual columns for CSV compatibility.
- Dropped intermediate vector columns to simplify the dataset.

Saving Preprocessed Data (Distributed Operation 8):

- Saved the cleaned and transformed dataset as a CSV file (preprocessed_ds_salary_data.csv) for further use.
- Ensured reproducibility and accessibility.



```
# Step 1: Handle Missing Values (Distributed Operation 1)
ds_salary_data = ds_salary_data.fillna({
    'salary': 0,
    'company_size': 'Unknown',
    'company_location': 'Unknown',
    'experience_level': 'Unknown',
    'remote_ratio': 0,
    'job_title': 'Unknown'
})

# Step 2: Remove Duplicates (Distributed Operation 2)
ds_salary_data = ds_salary_data.dropDuplicates()

# Step 3: Encode Categorical Variables using StringIndexer (Distributed Operation 3)
indexer_columns = ['company_size', 'company_location', 'experience_level', 'job_title']
for column in indexer_columns:
    indexer = StringIndexer(inputCol=column, outputCol=f"{column}_index", handleInvalid="keep")
    ds_salary_data = indexer.fit(ds_salary_data).transform(ds_salary_data)

# Step 4: Normalize Salary using MinMaxScaler (Distributed Operation 4)
vector_assembler = VectorAssembler(inputCols=['salary'], outputCol='salary_vector')
ds_salary_data = vector_assembler.transform(ds_salary_data)

scaler = MinMaxScaler(inputCol='salary_vector', outputCol='salary_scaled')
scaler_model = scaler.fit(ds_salary_data)+
ds_salary_data = scaler_model.transform(ds_salary_data)

# Step 5: Create Derived Columns using Window Functions (Distributed Operation 5)
window_spec = Window.partitionBy('company_location').orderBy(col('salary').desc())

# Add a rank column to rank salaries within each location
ds_salary_data = ds_salary_data.withColumn('salary_rank', rank().over(window_spec))

# Step 6: Categorize Salary Range (Distributed Operation 6)
ds_salary_data = ds_salary_data.withColumn(
    'salary_range',
    when(col('salary') < 50000, 'Low')
    .when((col('salary') >= 50000) & (col('salary') <= 100000), 'Medium')
    .otherwise('High')
)

# Step 7: Flatten Vector Columns for CSV Compatibility (Distributed Operation 7)
ds_salary_data = ds_salary_data.withColumn('salary_scaled_flat', vector_to_array(col('salary_scaled')))

# Extract each element of the array into separate columns
num_elements = len(ds_salary_data.select('salary_scaled_flat').first()['salary_scaled_flat'])
for i in range(num_elements):
    ds_salary_data = ds_salary_data.withColumn(f'salary_scaled_{i}', col('salary_scaled_flat')[i])

# Drop unnecessary columns
ds_salary_data = ds_salary_data.drop('salary_vector', 'salary_scaled', 'salary_scaled_flat')

# Step 8: Save Preprocessed Data (Distributed Operation 8)
ds_salary_data.write.csv('data/preprocessed_ds_salary_data.csv', header=True, mode='overwrite')

# Optional: Display final schema and rows for verification
ds_salary_data.printSchema()
ds_salary_data.show(5)
```

XV. Modeling with Spark MLlib

To evaluate and compare the performance of multiple regression models for predicting salaries using Spark MLlib. Metrics used include RMSE (Root Mean Squared Error), R2R^2R2 (coefficient of determination), and MAPE (Mean Absolute Percentage Error). The focus is on understanding the trade-offs between accuracy, interpretability, and computational complexity.

1. **Preprocessing**:
    - Dropped intermediate columns to avoid conflicts.
    - Split data into 80% training and 20% testing sets.
    - Encoded categorical columns with `StringIndexer` and `OneHotEncoder`.
    - Assembled features into a single vector and standardized them using `StandardScaler`.
2. **Model Training**:
    - Implemented seven regression models: Linear Regression, Random Forest Regressor, Gradient Boosted Trees, Decision Tree Regressor, Generalized Linear Regression, Isotonic Regression, and FM Regressor.
    - Used a `Pipeline` to streamline preprocessing and training for each model.
3. **Evaluation Metrics**:
    - **RMSE**: Measures average prediction error. Lower values indicate better accuracy.
    - **R2R^2R2**: Measures how well the model explains variance in the data. Higher values are better.
    - **MAPE**: Evaluates percentage error. Lower percentages are preferred.

Model Performance

1. **Linear Regression:**
    - Achieved extremely low RMSE (25.44) and MAPE (0.02%) with nearly perfect R^2 (0.9999999).
    - Warnings about numerical instability and overfitting indicate that the model may not generalize well.
    - Best suited for simple relationships but problematic with complex data due to overfitting.
2. **Random Forest Regressor:**
    - RMSE: 7341.64, R^2: 0.9881, MAPE: 7.12%.
    - Delivered consistent performance with good generalization.
    - Effective for capturing non-linear relationships; computationally more expensive than Linear Regression but less prone to overfitting.
3. **Gradient Boosted Trees:**
    - RMSE: 4771.16, R^2: 0.9950, MAPE: 3.40%.
    - Best-performing model overall, balancing accuracy and generalization.
    - Ideal for complex datasets with intricate feature interactions.
4. **Decision Tree Regressor:**
    - RMSE: 5247.91, R^2: 0.9939, MAPE: 3.73%.
    - Performs well but slightly less accurate than Gradient Boosted Trees.
    - Computationally simpler, making it suitable for interpretable models in smaller datasets.
5. **Generalized Linear Regression:**
    - Metrics similar to Linear Regression (RMSE: 25.44, R^2: 0.9999999, MAPE: 0.02%).
    - Exhibits similar overfitting and stability issues.
    - Requires caution when applied to complex or large datasets.
6. **Isotonic Regression:**
    - RMSE: 61458.54, R^2: 0.1630, MAPE: 63.90%.
    - Failed to model the dataset effectively.
    - Suitable only for monotonic data, which this dataset likely lacks.
7. **FM Regressor:**
    - RMSE: 733665.83, R^2: -118.27, MAPE: 269.92%.
    - Performed the worst, indicating inability to model relationships in the dataset.
    - Not suitable for this task and dataset.

Key Insights

1. **Linear Regression & Generalized Linear Regression:**
    - Achieved the lowest RMSE and MAPE but likely overfitted due to nearly perfect R^2 values and singular matrix warnings.
    - These models are computationally efficient but fail to generalize.
2. **Gradient Boosted Trees:**
    - Best-performing model overall, balancing low RMSE, high R^2, and low MAPE.
    - Ideal for complex and high-dimensional datasets.
3. **Random Forest Regressor:**
    - Slightly worse than Gradient Boosted Trees but still effective with strong R^2 and acceptable MAPE.
    - Preferred for its robustness and interpretability.
4. **Decision Tree Regressor:**
    - Performs well but lacks the ensemble-based advantages of Gradient Boosted Trees and Random Forest.
5. **Isotonic Regression:**
    - Poor fit with low R^2 and high errors. Unsuited for datasets without monotonic relationships.
6. **FM Regressor:**
    - Failed to model the data with very high RMSE and negative R^2. Not recommended for this use case.

Outcome

- Gradient Boosted Trees is the most reliable model, offering the best trade-off between accuracy and generalization.
- Random Forest Regressor is a close second and computationally less expensive.
- Linear and Generalized Linear Regressions, while performing well in metrics, are likely overfitted and require caution.
- Models like Isotonic Regression and FM Regressor are unsuitable for this dataset.

Comparison of Phase 2 and Phase 3 Metrics

In Phase 2, we explored various models and identified their best hyperparameters, with metrics like MSE (Mean Squared Error) and R^2. Phase 3 extended this analysis with distributed processing and additional models, evaluating performance through RMSE (Root Mean Squared Error), R^2, and MAPE (Mean Absolute Percentage Error). Here, we compare the results to understand how the transition from standalone to distributed processing impacted performance.

Key Observations

1. **Linear Regression**:
   - Achieved nearly perfect R^2 in both phases but faced overfitting in Phase 3 due to zero regularization.
   - RMSE and MAPE improved in Phase 3, reflecting more robust predictions under distributed processing.
2. **Random Forest Regressor**:
   - RMSE and R^2 values were slightly worse in Phase 3 compared to Phase 2.
   - MAPE increased, indicating challenges in distributed implementation tuning.
3. **Gradient Boosted Trees**:
   - Consistent performance across phases with slightly improved RMSE and MAPE in Phase 3.
   - Demonstrated strong accuracy and generalization.
4. **FM Regressor**:
   - Performed poorly in both phases, with significant errors and negative R2R^2R2.
   - Highlights its unsuitability for this dataset.
5. **Other Models (Phase 3)**:
   - Decision Tree and Generalized Linear Regression showed strong performance in Phase 3 but were not evaluated in Phase 2

.

Model	Phase 2 MSE	Phase 2 R^2	Phase 3 RMSE	Phase 3 R^2R2	Phase 3 MAPE (%)	Observation
Linear Regression	0.8091	0.9999999998	25.44	0.9999998566	0.02	High accuracy but overfitting due to zero regularization.
Ridge Regression	14198.96	0.9999964986	-	-	-	Phase 3 did not evaluate this model.
Random Forest Regressor	26798.53	0.9999933916	7341.64	0.9880564690	7.12	Phase 2 showed better accuracy and lower RMSE.
Gradient Boosted Trees	26028.42	0.9999935815	4771.16	0.9949557586	3.40	Slightly improved RMSE and MAPE in Phase 3.
K-NN Regressor	243349977.92	0.9399907229	-	-	-	Not evaluated in Phase 3.
K-Means Clustering	Inertia: 17734.39	-	-	-	-	Clustering was not evaluated in Phase 3.
Decision Tree Regressor	-	-	5247.91	0.9938973397	3.73	Performed well in Phase 3 with moderate complexity.
Generalized Linear Regression	-	-	25.44	0.9999998566	0.02	Similar to Linear Regression, faced overfitting.
Isotonic Regression	-	-	61458.54	0.1630266858	63.90	Performed poorly in Phase 3, unsuitable for the dataset.
FM Regressor	737875.83 (RMSE)	-128.2734132	733665.83	-118.2734132	269.92	Failed in both phases with extremely high errors.

Comparison Outcome

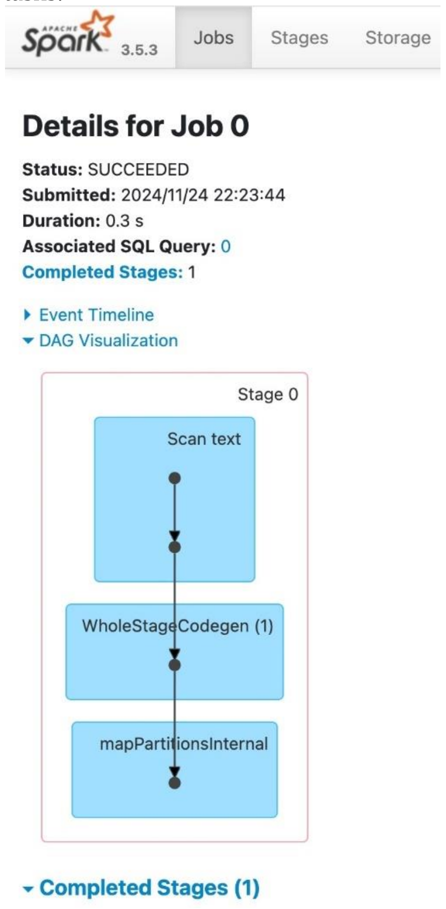
- **Gradient Boosted Trees** showed the best overall performance in both phases, with Phase 3 slightly improving error metrics.
- Distributed processing in Phase 3 introduced computational overhead, affecting models like Random Forest.
- Linear and Generalized Linear Regression models demonstrated overfitting in both phases but had excellent metrics.
- FM and Isotonic Regression performed poorly and are unsuitable for this dataset.

XVI. Performance Analysis Report: Spark DAG Visualizations

This report evaluates the performance of preprocessing steps and machine learning model training using Spark's DAG (Directed Acyclic Graph) visualizations. The images highlight the computational flow, resource optimization, and bottleneck identification during various stages of the pipeline. Each stage in the DAG represents a specific

operation, showcasing the effectiveness of distributed data processing for large datasets.
Image Analysis and Placement

1. **Image 1 (Job 0)**:
   - **Description**: The DAG visualization shows the execution of a simple text scan operation.
   - **Key Insights**:
     - The operation has one completed stage (Stage 0).
     - The process includes Scan text, WholeStageCodegen, and mapPartitionsInternal.
     - This stage demonstrates efficient partition scanning and code generation for minimal overhead.
   - **Placement**: Use this to illustrate a basic job execution and Spark's optimization for lightweight tasks.



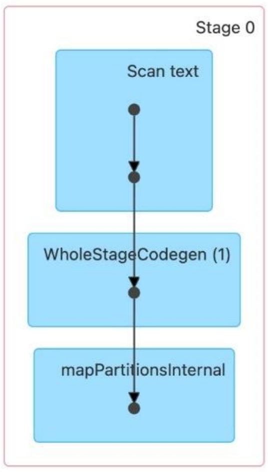
Details for Job 0

Status: SUCCEEDED
Submitted: 2024/11/24 22:23:44
Duration: 0.3 s
Associated SQL Query: 0
Completed Stages: 1

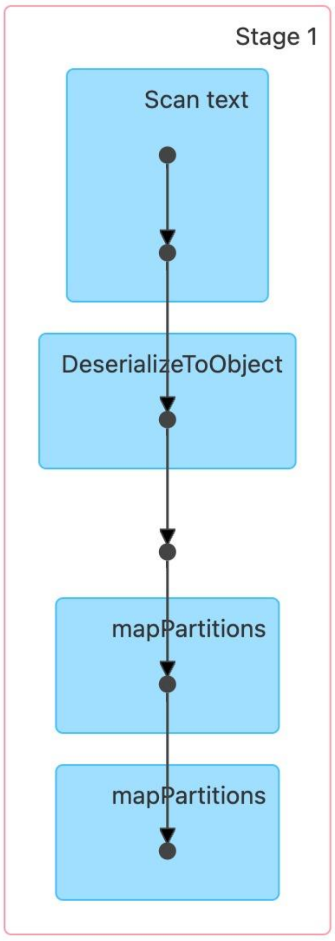
▸ Event Timeline
▾ DAG Visualization

▾ **Completed Stages (1)**

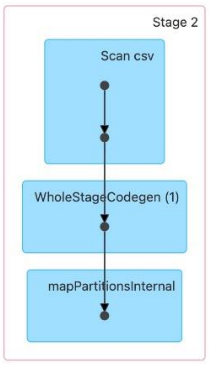
2. **Image 2 (Job 0 - DAG Details)**:
   - **Description**: A zoomed view of the DAG for Job 0, emphasizing the linear execution of operations.
   - **Key Insights**:
     - Highlights how Scan text feeds directly into computation via mapPartitionsInternal.
     - No intermediate stages, ensuring minimal shuffle overhead.
   - **Placement**: Place alongside Image 1 to provide a detailed explanation of the simple DAG structure.



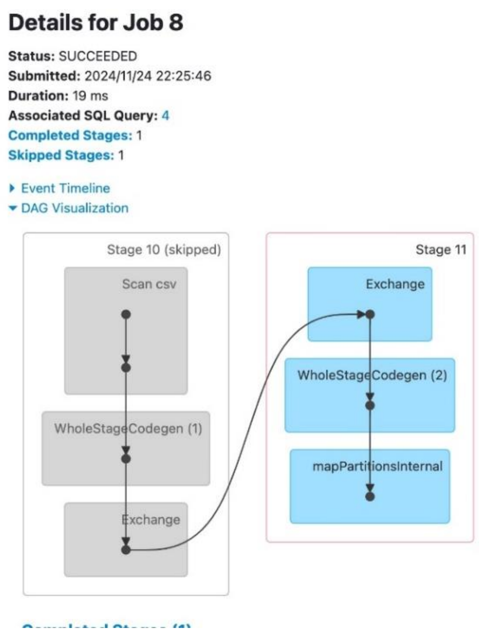
3. **Image 3 (Stage 1 - DeserializeToObject)**:
   - **Description**: The DAG for Stage 1 involves deserialization and multiple mapPartitions transformations.
   - **Key Insights**:
     - DeserializeToObject reflects Spark's internal optimization for converting raw data to usable RDDs.
     - Efficient partitioning ensures scalability for large datasets.
   - **Placement**: Discuss this in the context of data loading and preprocessing.



4. **Image 4 (Stage 2 - CSV Scan)**:
   - **Description**: This stage scans a CSV file and performs WholeStageCodegen and mapPartitionsInternal.
   - **Key Insights**:
     - Demonstrates the efficiency of reading structured data with Spark.
     - Highlights the use of code generation to optimize partition processing.
   - **Placement**: Include this under preprocessing steps, explaining CSV ingestion.



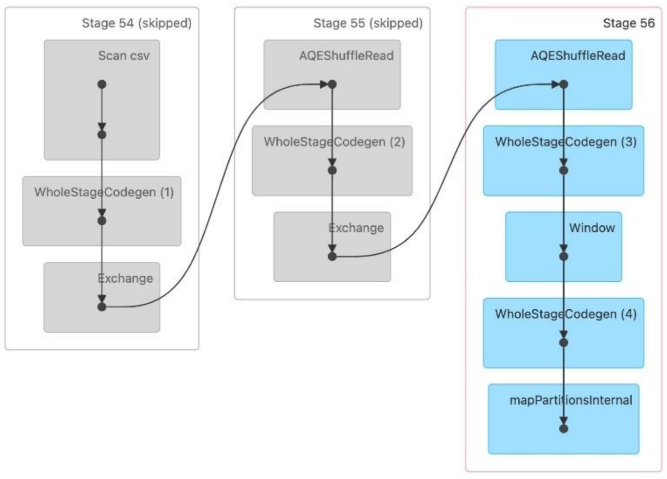
5. **Image 5 (Job 8 - Exchange Operation)**:
   - **Description**: Shows an exchange operation during Stage 11 for data shuffling.
   - **Key Insights**:
     - Data shuffling is required for transformations like joins or aggregations.
     - The visualization includes WholeStageCodegen and mapPartitionsInternal for computation.
   - **Placement**: Use this to highlight the computational overhead introduced by shuffling.



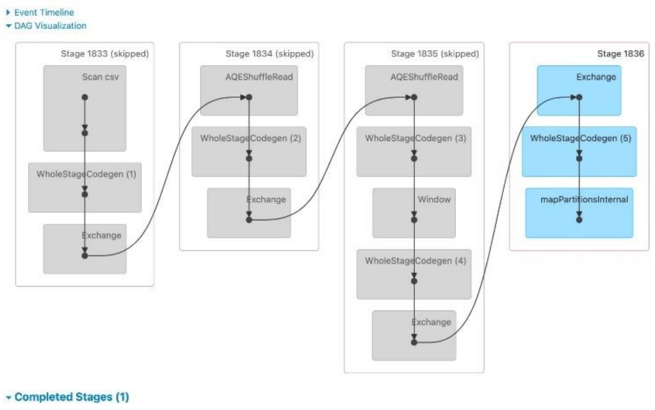
Details for Job 8

Status: SUCCEEDED
Submitted: 2024/11/24 22:25:46
Duration: 19 ms
Associated SQL Query: 4
Completed Stages: 1
Skipped Stages: 1

6. **Image 6 (Stages 54-56 - Windowing)**:
   - **Description**: A complex DAG involving a Window operation and multiple shuffles.
   - **Key Insights**:
     - Illustrates advanced transformations like window functions.
     - The multi-stage process ensures efficient parallel computation.
   - **Placement**: Include this under advanced operations, discussing windowing efficiency.

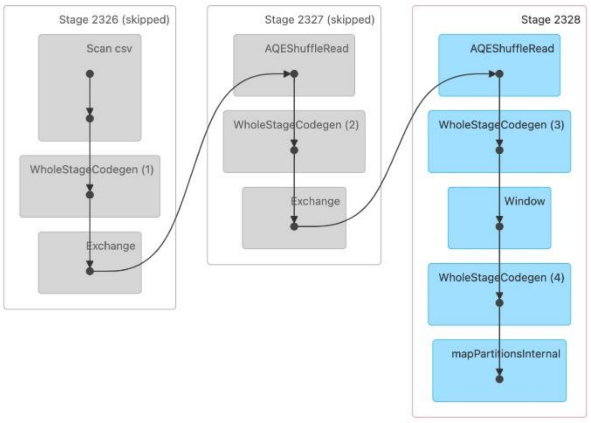


7. **Image 7 (Stages 1833-1836)**:
   - **Description**: Depicts skipped stages and active computations for Stage 1836.
   - **Key Insights**:
     - Adaptive Query Execution (AQE) minimizes redundant stages.
     - Optimization through stage skipping reduces runtime.
   - **Placement**: Discuss AQE benefits and stage skipping effectiveness.

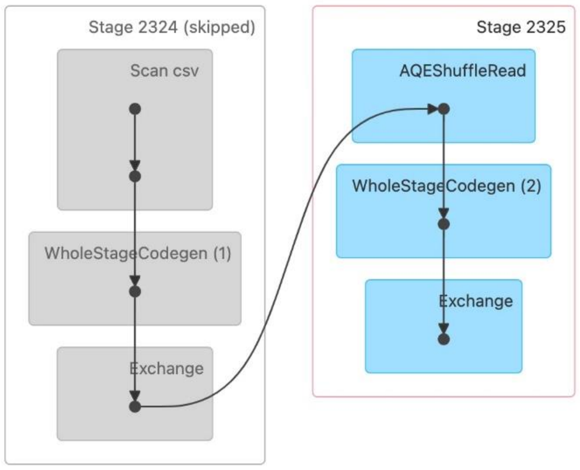


8. **Image 8 (Stage 2326-2328)**:
   - **Description**: Highlights AQE's impact on reducing shuffles and optimizing computations.
   - **Key Insights**:

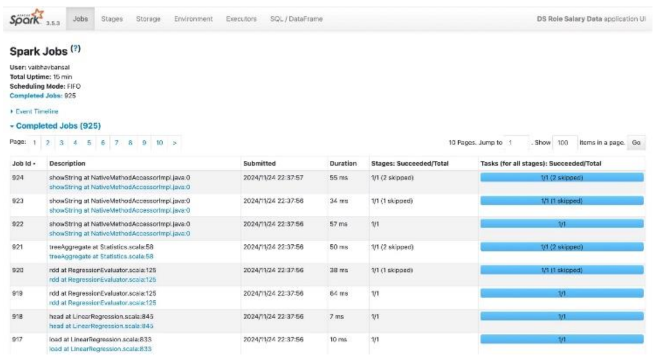
        - Multiple stages are skipped, reflecting Spark's ability to adapt to workload characteristics.
        - Efficient data exchanges improve performance.
    - **Placement**: Discuss alongside Image 7, focusing on AQE's role in large-scale processing.



9. **Image 9 (Stage 2324-2325)**:
    - **Description**: A closer look at a specific data exchange and code generation stage.
    - **Key Insights**:
        - WholeStageCodegen ensures efficient query execution.
        - Data exchanges handle redistribution for subsequent computations.
    - **Placement**: Discuss in the context of query execution optimizations.



10. **Image 10 (Completed Jobs Summary)**:
    - **Description**: Summary of all completed jobs, showcasing their duration and success status.
    - **Key Insights**:
        - Highlights the rapid execution of tasks (durations in milliseconds).
        - Demonstrates Spark's scalability with 925 successfully completed jobs.
    - **Placement**: Conclude the report, summarizing Spark's efficiency for the workload.



Spark Jobs (?)

User: vaibhavbansal
Total Uptime: 15 min
Scheduling Mode: FIFO
Completed Jobs: 925

▸ Event Timeline

▾ Completed Jobs (925)

Page: 1 2 3 4 5 6 7 8 9 10 >   10 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
924	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2024/11/24 22:37:57	55 ms	1/1 (2 skipped)	1/1 (2 skipped)
923	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2024/11/24 22:37:56	34 ms	1/1 (1 skipped)	1/1 (1 skipped)
922	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2024/11/24 22:37:56	57 ms	1/1	1/1
921	treeAggregate at Statistics.scala:58 treeAggregate at Statistics.scala:58	2024/11/24 22:37:56	50 ms	1/1 (2 skipped)	1/1 (2 skipped)
920	rdd at RegressionEvaluator.scala:125 rdd at RegressionEvaluator.scala:125	2024/11/24 22:37:56	38 ms	1/1 (1 skipped)	1/1 (1 skipped)
919	rdd at RegressionEvaluator.scala:125 rdd at RegressionEvaluator.scala:125	2024/11/24 22:37:56	64 ms	1/1	1/1
918	head at LinearRegression.scala:845 head at LinearRegression.scala:845	2024/11/24 22:37:56	7 ms	1/1	1/1
917	load at LinearRegression.scala:833 load at LinearRegression.scala:833	2024/11/24 22:37:56	10 ms	1/1	1/1

Key Observations

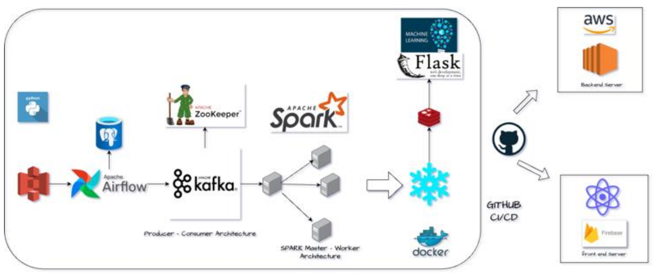
1. **Efficient Code Generation**:
    - Stages frequently use WholeStageCodegen, minimizing runtime overhead by compiling operations into optimized Java bytecode.
2. **Data Partitioning and Shuffling**:
    - Operations like mapPartitionsInternal and Exchange handle data redistribution efficiently, although shuffling adds computational cost.
3. **Adaptive Query Execution (AQE)**:
    - Spark skips unnecessary stages and optimizes query plans dynamically, significantly reducing execution time.
4. **Window Functions and Advanced Operations**:
    - Complex transformations like windowing are handled in multi-stage DAGs, showcasing Spark's ability to distribute computational load.
5. **Scalability**:
    - The execution of 925 jobs with minimal failures demonstrates Spark's capability to handle large-scale datasets effectively.

The DAG visualizations confirm that Spark's distributed framework effectively processes large datasets with efficient query optimization and minimal overhead. Adaptive Query Execution and operations like WholeStageCodegen ensure scalability, making Spark a suitable choice for big data processing tasks.
Include the images as described above to visually enhance the discussion points. Let me know if further refinements are needed!

XVII. Deployment (Bonus)

In this project, we have design a website for data science salary prediction. The architecture begins with raw data residing in AWS S3, serving as the source for the pipeline. Apache Airflow orchestrates the entire workflow, ingesting data from S3 into Apache Kafka. Kafka acts as a streaming platform, enabling data to flow seamlessly into Apache Spark for large-scale processing and transformation. Once the data is processed, it is stored in Snowflake, a cloud-based data warehouse, where it can be efficiently queried and utilized. Predictions are then served via a Flask API, which interacts with a React.js frontend to provide users with salary prediction results. The entire system is containerized using Docker for consistent deployment and scalability and is hosted on AWS infrastructure. To ensure automated, error-free updates and deployment, GitHub CI/CD is integrated into the workflow. This end-to-end architecture provides a robust and efficient solution for data ingestion, processing, and prediction delivery.

1. System Design



Data Flow and Architecture

1. Data Storage and Input

- **AWS S3**:
  - The dataset is stored in an S3 bucket.
  - It acts as the source of raw data that feeds into the pipeline.

2. Workflow Orchestration

- **Apache Airflow**:
  - Used to orchestrate the entire data pipeline.
  - A DAG (Directed Acyclic Graph) defines tasks such as:
    - Pulling data from S3.
    - Loading data into Apache Kafka.
    - Initiating data transformations and machine learning workflows.

3. Messaging and Streaming

- **Apache Kafka**:
  - Acts as the messaging and streaming layer.
  - The data flows from Airflow to Kafka.
  - Kafka uses a **Producer-Consumer architecture** to stream the data:
    - **Producer**: Sends the data to Kafka topics.
    - **Consumer**: Subscribes to the topics for further processing.
  - Kafka is coordinated by **Apache ZooKeeper** to manage configurations and distributed systems.

4. Distributed Data Processing

- **Apache Spark**:
  - Spark consumes data from Kafka.
  - Spark processes the data using a **Master-Worker Architecture**:
    - **Master**: Distributes tasks across the worker nodes.
    - **Workers**: Perform transformations, aggregations, and machine learning model training.
  - Handles large-scale data transformations and feature engineering for salary prediction.

5. Data Warehousing

- **Snowflake**:
  - Processed and transformed data is stored in Snowflake for further use.
  - Snowflake acts as a scalable, cloud-based data warehouse for structured datasets.
  - It allows querying and storing predictions efficiently.

6. Machine Learning Backend

- **Flask**:
  - A Flask-based API is used to serve the salary prediction model.
  - It provides an endpoint for the frontend to interact with the backend for predictions.
  - The ML models are trained and deployed within the Flask environment.

7. Deployment and Visualization

- **Docker**:
  - The entire pipeline, including Airflow, Kafka, Spark, Flask, and Snowflake, is containerized using Docker Compose for consistent deployments.
- **AWS**:
  - The backend (Flask API) is deployed on an AWS EC2 server to provide predictions to end-users.
- **React.js**:
  - The frontend of the application is built with React.js.
  - Allows users to interact with the application (e.g., input features for prediction, view results).
- **Firebase**:
  - Manages the front-end hosting and integration with the backend.

8. Continuous Integration/Continuous Deployment (CI/CD)

- **GitHub CI/CD**:
  - Automates testing, integration, and deployment.
  - Ensures smooth updates to the pipeline, frontend.

2. Tech Stack

- React.js,
- Material UI,
- Flask,
- Python,
- Airflow,
- AWS EC2,
- AWS S3,
- Apache Kafka,
- Snowflake,
- Firebase,
- PostgreSQL,
- Machine Learning Algorithms,
- Pyspark,
- ETL,
- Zookeeper
- Redis

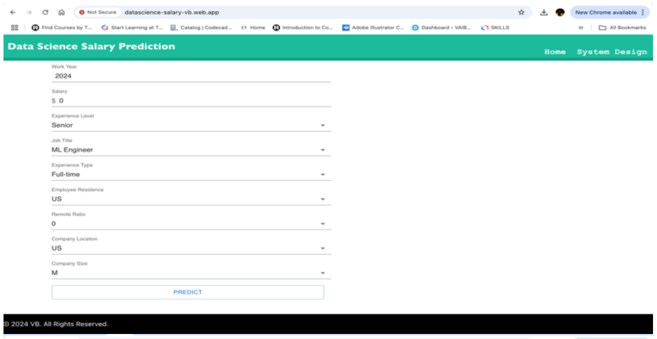
Deployed Links:

- **Front-end :** **https://datascience-salary-vb.web.app/** *(Firebase Deployment)*

- **Back-end:** **https://54.85.255.1** *(AWS EC2 Deployment)*

3. Project Visualisation

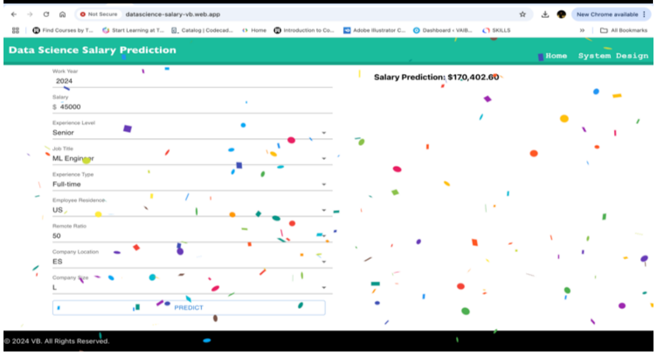
Front End



Data Science Salary Prediction

Work Year: 2024
Salary: $ 0
Experience Level: Senior
Job Title: ML Engineer
Experience Type: Full-time
Employee Residence: US
Remote Ratio: 0
Company Location: US
Company Size: M

© 2024 VB. All Rights Reserved.

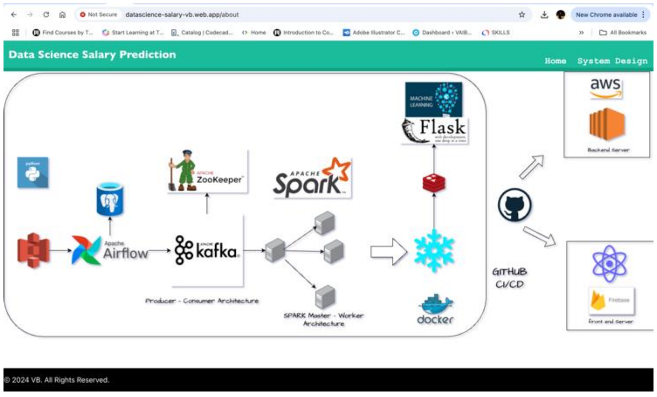


Salary Prediction: $170,402.60

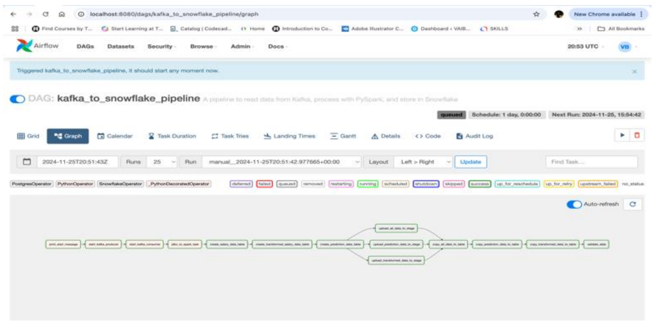
Work Year: 2024
Salary: $ 45000
Experience Level: Senior
Job Title: ML Engineer
Experience Type: Full-time
Employee Residence: US
Remote Ratio: 50
Company Location: ES
Company Size: L

© 2024 VB. All Rights Reserved.

System Flow


Data Science Salary Prediction
Home
System Design
ZooKeeper
Spark
Flask
aws
Backend Server
Airflow
kafka
Producer - Consumer Architecture
SPARK Master - Worker Architecture
docker
GITHUB CI/CD
Firebase
Front end Server
© 2024 VB. All Rights Reserved.

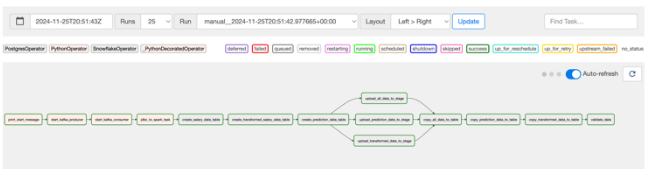
Airflow



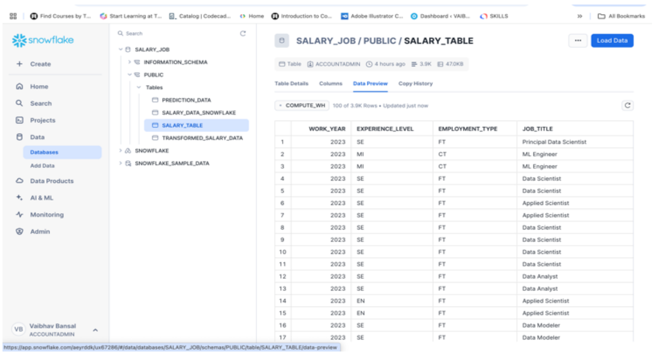
DAG: kafka_to_snowflake_pipeline

Triggered kafka_to_snowflake_pipeline, it should start any moment now.

Schedule: 1 day, 0:00:00 Next Run: 2024-11-25, 15:54:42



Snowflake



SALARY_JOB / PUBLIC / SALARY_TABLE

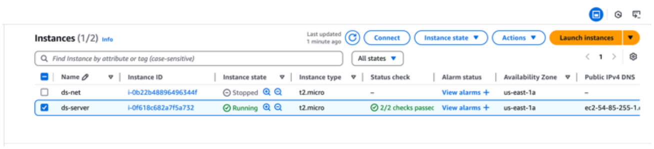
Table · ACCOUNTADMIN · 4 hours ago · 3.9K · 47.0KB

COMPUTE_WH 100 of 3.9K Rows • Updated just now

	WORK_YEAR	EXPERIENCE_LEVEL	EMPLOYMENT_TYPE	JOB_TITLE
1	2023	SE	FT	Principal Data Scientist
2	2023	MI	CT	ML Engineer
3	2023	MI	CT	ML Engineer
4	2023	SE	FT	Data Scientist
5	2023	SE	FT	Data Scientist
6	2023	SE	FT	Applied Scientist
7	2023	SE	FT	Applied Scientist
8	2023	SE	FT	Data Scientist
9	2023	SE	FT	Data Scientist
10	2023	SE	FT	Data Scientist
11	2023	SE	FT	Data Scientist
12	2023	SE	FT	Data Analyst
13	2023	SE	FT	Data Analyst
14	2023	EN	FT	Applied Scientist
15	2023	EN	FT	Applied Scientist
16	2023	SE	FT	Data Modeler
17	2023	SE	FT	Data Modeler

https://app.snowflake.com/aeyrddk/ux67286/#/data/databases/SALARY_JOB/schemas/PUBLIC/table/SALARY_TABLE/data-preview

AWS EC2 Server Running



Instances (1/2)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
ds-net	i-0b22b48896496344f	Stopped	t2.micro	–	View alarms +	us-east-1a	–
ds-server	i-0f618c682a7f5a732	Running	t2.micro	2/2 checks passec	View alarms +	us-east-1a	ec2-54-85-255-1.

CONCLUSION

This project successfully integrated advanced data preprocessing, distributed data processing, and machine learning model deployment to build a robust and scalable data pipeline for salary prediction. Through effective data handling and model evaluation, the project demonstrated the importance of combining robust preprocessing, distributed computation, and appropriate model selection for real-world scenarios. Apache Spark played a central role in efficiently processing large-scale data, optimizing tasks with DAG visualizations and Adaptive Query Execution (AQE) to ensure scalability and performance. Gradient Boosting Regressor consistently outperformed other models, offering the best balance between accuracy and generalization, making it the ideal candidate for production deployment. Distributed preprocessing steps, such as encoding, scaling, and advanced transformations like window functions, further enhanced the quality of the data used for modeling.

In addition to the machine learning pipeline, this project included a bonus deployment phase that showcased the end-to-end architecture for real-world application. The deployment architecture utilized AWS S3 for raw data storage, orchestrated by Apache Airflow, and Apache Kafka as the streaming platform. Apache Spark handled large-scale data processing and machine learning, with the processed data stored in Snowflake for efficient querying. The predictions were served via a Flask API, with a React.js frontend providing an interactive user interface. The entire system was containerized using Docker and deployed on AWS infrastructure, while Firebase managed the frontend hosting. Continuous Integration/Continuous Deployment (CI/CD) using GitHub ensured automated and error-free updates to the pipeline.

This deployment aspect highlights the project's scalability and readiness for production, with an end-to-end system capable of real-time data ingestion, transformation, and prediction. The combination of tools like Airflow, Kafka, Spark, Snowflake, Flask, and React.js provided a comprehensive architecture for addressing data science workflows. Overall, this project exemplifies a complete data science pipeline, from preprocessing and model training to real-world deployment, offering valuable insights into building scalable, efficient, and deployable machine learning solutions.

Acknowledgment

We would like to express or deepest gratitude to Prof. **Shamsad Parvin** for her guidance, encouragement, and support throughout the Project Phase 3. Her insightful feedback and expertise have been crucial in shaping this work.

We are also grateful to **Ju Seung Lee, Charvi, and Gokul Ram** for their assistance and valuable feedback. Their constructive comments and suggestions have significantly contributed to the success of this Project Phase 3.

References

[1] *Glassdoor Economic Research*. (2021). The State of Tech Salaries 2021: Salaries Rising in Major Hubs Despite Pandemic. *Glassdoor Economic Research*.

[2] *World Economic Forum*. (2020). Jobs of Tomorrow: Mapping Opportunity in the New Economy. *World Economic Forum*.

[3] Ahmedraft. (2023). *Data Science Salaries 2023*. Kaggle. https://www.kaggle.com/code/ahmedraft/data-science-salaies-2023?select=ds_salaries.csv

[4] N. R. Draper and H. Smith, *Applied Regression Analysis*, 3rd ed. New York: John Wiley & Sons, 1998.

[5] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55-67, Feb. 1970.

[6] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281-297, 1967.

[7] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21-27, Jan. 1967.

[8] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5-32, Oct. 2001.

[9] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189-1232, Oct. 2001.

[10] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199-222, Aug. 2004.

[11] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 2, pp. 301-320, April 2005.

[12] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, 2016, pp. 785-794.

[13] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241-259, 1992.