Experiment Notebook

A. Project

student_name = 'Shashikanth Senthil Kumar'

student_id = '25218722'

experiment_id = '0'

business_objective:

The primary business objective of this project is to develop regression models that can accurately predict the salary of engineering students across different colleges based on their academic performance, personality traits, and specialized skills.

These predictions will be used by recruitment agencies, college career services, and companies to:

- Optimize Hiring Decisions: Companies can use the model to estimate the potential salary of a candidate and make informed decisions on salary offers during recruitment.
- Assess Talent Pools: Recruitment agencies can identify candidates who are likely to command higher salaries, allowing them to prioritize resources and tailor recruitment efforts.
- Improve Career Guidance: Colleges and career counselors can use the predictions to guide students on improving their skills or choosing specializations that may lead to higher salaries.

By accurately predicting salaries, the model aims to enhance decision-making in recruitment, talent management, and career planning, benefiting both employers and candidates.

B. Experiment Description

```
experiment_hypothesis = 'There is no hypothesis to test. This is used for data exploration and analysis and required data
```

```
experiment_expectations = 'The goal is to assess the baseline performance for this project and fix critical data quality i
```

C. Data Understanding

data_understanding_executive_summary

- Datasets:

```
Training (1803 entries), Validation (901 entries), Testing (902 entries).
```

- Content:

```
Includes personal identifiers and academic information. Remove irrelevant features like passport_number and DOB.
```

- Target Variable (Salary):

```
Training: Narrow range, minimal skew.
Validation: Broader range, slight skew towards higher salaries.
Testing: Wide range, significant right skew due to high outliers.
```

- Features:

```
CollegeGPA: Concentrated between 65 and 80 across all datasets, with a peak around 70-75.
12percentage: Mean varies (Training: 74.79%, Validation: 71.25%, Testing: 76.08%) with broad variability and high outliers.
Quant: Mean varies (Training: 518.48, Validation: 467.18, Testing: 548.21). The data is normally distributed across all datasets, with extreme values and broader variability in the Testing dataset.
```

- Considerations:

```
Ensure consistent preprocessing (e.g., remove irrelevant features, handle missing values).
Scale numerical features and encode categorical variables.
Manage outliers and consider feature engineering for better model performance.
```

- Issues:

```
Significant differences in means across datasets could impact model accuracy and generalization.
Variability in 'Quant' values differs among datasets, potentially challenging model consistency.
Testing dataset has more extreme values, which may affect model stability if not properly managed.
Risk of overfitting due to irrelevant features and privacy issues with personal data.
```

C.0 Import Packages

```
# Pandas for data handling
import pandas as pd

# Scikit Learn for ML training
import sklearn

# Altair for plotting
import altair as alt

# NumPy for numerical computations
import numpy as np

# Matplotlib for basic plotting
import matplotlib.pyplot as plt

# Ensures that Matplotlib plots are displayed inline in the notebook
%matplotlib inline

# Seaborn for statistical data visualization
import seaborn as sns


# Warnings module to suppress unwanted warnings
import warnings

# Suppress future warnings to make the output cleaner
warnings.simplefilter(action='ignore', category=FutureWarning)
```

C.1 Load Datasets

```
# Load training data
# Do not change this code

training_df = pd.read_csv("salary_training.csv")
```

```
# Load validation data
# Do not change this code

validation_df = pd.read_csv("salary_validation.csv")
```

```
# Load testing data
# Do not change this code

testing_df = pd.read_csv("salary_testing.csv")
```

C.2 Explore Datasets

```
# Training Dataset

# Display all columns of the DataFrame when printing
pd.set_option('display.max_columns', None)

# Display the first 5 rows of the training dataset
training_df.head()
```

	ID int64	Gender object	DOB object	10percentage flo...	12graduation int64	12percentage flo...	CollegeID int64	Co
0	301647	m	1989-08-21	77.33	2007	85.17	9084	
1	339001	f	1990-10-30	82	2008	75	4889	
2	1183070	m	1992-11-25	76.8	2010	87.7	5338	
3	794062	f	1993-03-15	57	2009	73	8346	
4	1088206	m	1990-06-21	77	2008	75	13424	

5 rows, 58 cols 10 / page Page 1 of 1

Insights:

- The head() method displays the first five rows of the dataset.
- The dataset contains various columns such as ID, full_name, DOB, passport_number, collegeGPA, 10percentage, Salary, etc.
- Personal identifiers like first_name, last_name, DOB, and passport_number are present, which will need to be removed before modeling.
- Academic variables such as collegeGPA, 12percentage, and Specialization are visible and will likely be important for predicting salary.

```
# Display the descriptive statistics for numerical columns in the training Data
training_df.describe()
```

	ID float64	10percentage flo...	12graduation flo...	12percentage flo...	CollegeID float64	CollegeTier float...	collegeGPA float...	Co
cou...	1803	1803	1803	1803	1803	1803	1803	
me...	721743.4664	78.28312812	2008.357737	74.7903716	5408.078203	1.92900721	71.5663117	
std	349576.5691	9.752816064	1.537590322	10.84485689	4844.573926	0.2568840466	7.94783348	
min	17126	44	2001	40	2	1	6.63	
25%	407687	72	2008	66.8	533	2	66.785	
50%	731107	79.6	2009	74.6	4450	2	72	
75%	1037145	86	2009	82.8	9020	2	76.19	
max	1297877	97	2012	98.2	18409	2	99	

8 rows, 31 cols 10 / page Page 1 of 1

> Insights

- Academic Performance:

Average 10th-grade score: 78.28%, 12th-grade: 74.79%, College GPA: 71.57.
Moderate variability in performance, with a standard deviation of around 9-10%.

- Test Scores:

Average English, Logical, and Quant scores: around 500-518.
Significant variability, with low scores around 200 and high scores reaching 800.

- Engineering Disciplines:

Wide range in disciplines like Computer Programming, with negative values suggesting missing/erroneous data.

- Personality Traits:

Mean values range from -0.21 to 1.03, with high variability, indicating diverse personality profiles.

- Salary:

Average salary: 85,715, with a small range (80,051 to 89,894), showing similar compensation across students.

```
# Display the shape of training dataset
training_df.shape
```

```
# Display information about the training dataset
training_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1803 entries, 0 to 1802
Data columns (total 58 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   ID                     1803 non-null   int64
 1   Gender                 1803 non-null   object
 2   DOB                    1803 non-null   object
 3   10percentage           1803 non-null   float64
 4   12graduation           1803 non-null   int64
 5   12percentage           1803 non-null   float64
 6   CollegeID              1803 non-null   int64
 7   CollegeTier            1803 non-null   int64
 8   Degree                 1803 non-null   object
 9   Specialization         1803 non-null   object
 10  collegeGPA             1803 non-null   float64
 11  CollegeCityID          1803 non-null   int64
 12  CollegeCityTier        1803 non-null   int64
 13  GraduationYear         1803 non-null   int64
 14  English                1803 non-null   int64
 15  Logical                1803 non-null   int64
 16  Quant                  1803 non-null   int64
 17  Domain                 1803 non-null   float64
 18  ComputerProgramming    1803 non-null   int64
 19  ElectronicsAndSemicon  1803 non-null   int64
 20  ComputerScience        1803 non-null   int64
 21  MechanicalEngg         1803 non-null   int64
 22  ElectricalEngg         1803 non-null   int64
 23  TelecomEngg            1803 non-null   int64
 24  CivilEngg              1803 non-null   int64
```

```
# Display columns of the training dataset
training_df.columns
```

Insights:

- There are 1803 entries and 58 columns in the training dataset.
- All columns except last_name have no missing values. The last_name column has one missing entry.
- Most features are either integers or floats, with some categorical columns like Specialization and Degree.
- The dataset also contains irrelevant columns (e.g., first_name, passport_number) which should be dropped to avoid data leakage.

```
# Validation Dataset

# Display the first 5 rows of the validation dataset
validation_df.head()
```

	ID int64	Gender object	DOB object	10percentage flo...	12graduation int64	12percentage flo...	CollegeID int64	Co
0	230553	m	1992-05-01	66.66	2005	80.4	64	
1	1038837	m	1990-09-21	73	2008	76	10930	
2	781196	m	1991-02-09	78.4	2009	58.4	11013	
3	661812	f	1991-09-23	71	2009	68	13040	
4	657409	m	1992-01-08	76.5	2009	84	11447	

5 rows, 58 cols 10 / page Page 1 of 1

Insights:

- The first five rows of the validation dataset show similar features as the training dataset, with personal information (e.g., first_name, passport_number) and academic-related columns (collegeGPA, 12percentage, etc.).
- The structure and content of the validation dataset align with the training set, which is important for consistent model evaluation.

```
# Display the descriptive statistics for numerical columns in the validation Data
validation_df.describe()
```

	ID float64	10percentage flo...	12graduation flo...	12percentage flo...	CollegeID float64	CollegeTier float...	collegeGPA float...	Co
cou...	599	599	599	599	599	599	599	
me...	691548.6144	74.28145242	2007.896494	71.25480801	5596.098497	1.969949917	70.09066778	
std	377658.1137	10.23675431	1.665899698	11.19574707	4903.705079	0.1708678346	8.202504668	
min	21229	43	2001	43	13	1	6.85	
25%	330563.5	67	2007	63	993	2	65	
50%	660938	75	2008	70	4439	2	70	
75%	1062176.5	82	2009	79.8	9459	2	75.18	
max	1297763	97.12	2011	97.5	17782	2	96.7	

8 rows, 31 cols 10 / page Page 1 of 1

Insights

- Academic Performance: 10th (74%) and 12th (71%) average percentages, with wide variability.
- College GPA: Average GPA is 70, mostly from Tier 2 colleges.
- Test Scores: English (464), Logical (475), and Quant (467) have large score ranges.
- Engineering Specializations: Many missing/erroneous values across different engineering fields.
- Personality Traits: Scores vary widely, centered around zero.
- Salary: Average salary is 81,917 with minimal variation.

```
# Display the shape of validation dataset
validation_df.shape
```

```
# Display information about the validation dataset
validation_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 599 entries, 0 to 598
Data columns (total 58 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   ID                     599 non-null    int64
 1   Gender                 599 non-null    object
 2   DOB                    599 non-null    object
 3   10percentage           599 non-null    float64
 4   12graduation           599 non-null    int64
 5   12percentage           599 non-null    float64
 6   CollegeID              599 non-null    int64
 7   CollegeTier            599 non-null    int64
 8   Degree                 599 non-null    object
 9   Specialization         599 non-null    object
 10  collegeGPA             599 non-null    float64
 11  CollegeCityID          599 non-null    int64
 12  CollegeCityTier        599 non-null    int64
 13  GraduationYear         599 non-null    int64
 14  English                599 non-null    int64
 15  Logical                599 non-null    int64
 16  Quant                  599 non-null    int64
 17  Domain                 599 non-null    float64
 18  ComputerProgramming    599 non-null    int64
 19  ElectronicsAndSemicon  599 non-null    int64
 20  ComputerScience        599 non-null    int64
 21  MechanicalEngg         599 non-null    int64
 22  ElectricalEngg         599 non-null    int64
 23  TelecomEngg            599 non-null    int64
 24  CivilEngg              599 non-null    int64
```

```
# Display columns of the validation dataset
validation_df.columns
```

Insights:

- The validation dataset has 901 entries and the same 58 columns as the training dataset.
- There are no missing values in the validation dataset, except for one missing value in the last_name column in the training dataset. This discrepancy should be noted for consistency between the datasets.
- The data types and structure are similar to the training set, which is necessary for fair evaluation during model experimentation.

```
#Testing Dataset

# Display the first 5 rows of the testing dataset
testing_df.head()
```

	ID int64	**Gender** object	**DOB** object	**10percentage** flo...	**12graduation** int64	**12percentage** flo...	**CollegeID** int64	Co
0	669692	m	1992-05-04	73	2009	71	9699	
1	87319	m	1986-10-27	56.4	2006	72	403	
2	962441	m	1992-08-10	88.67	2009	87.69	436	
3	110863	m	1988-08-04	85.6	2006	90.6	964	
4	1083018	m	1992-08-25	91	2010	91.2	2921	

5 rows, 58 cols 10 / page Page 1 of 1

Insights:

- The first five rows show that the testing dataset is similar in structure and content to the training and validation datasets.
- Features such as personal identifiers (passport_number, first_name, DOB) are present, indicating the need for further cleaning.
- Key academic features (collegeGPA, Specialization, Quant) are also available for salary prediction.

```
# Display the descriptive statistics for numerical columns in the testing Data
testing_df.describe()
```

	ID float64	10percentage flo...	12graduation flo...	12percentage flo...	CollegeID float64	CollegeTier float...	collegeGPA float...	Cc
cou...	596	596	596	596	596	596	596	
me...	466118.9178	79.20199664	2007.427852	76.08360738	4223.798658	1.865771812	72.76540268	
std	327308.7815	9.774413634	1.66420428	11.29006078	4293.321491	0.3411836662	8.352586476	
min	11244	46.24	1998	45.5	2	1	8.89	
25%	232168	73	2006	68	401.5	2	67.8	
50%	344506	81	2008	77	2761	2	72.7	
75%	631799	86.8	2009	84.4	7054.5	2	77.7	
max	1279987	97.76	2012	98.7	17933	2	99.93	

8 rows, 31 cols 10 / page Page 1 of 1

Insights

- Academic Performance: Average 10th grade percentage is 79.2%, 12th grade is 76.1%, and college GPA is 72.8. Wide variability in scores.
- Test Scores: Average English, Logical, and Quantitative scores are 515, 519, and 548, with high variation.
- Domain/Engineering Skills: Missing values for many engineering disciplines, reflecting data quality issues.
- Personality Traits: Scores cluster around zero, with significant variation.
- Salary: Average salary is 93,479, ranging from 79,335 to 218,749, showing a broad earning potential.

```
# Display the shape of testing dataset
testing_df.shape
```

```
# Display the information about the testing dataset
testing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 596 entries, 0 to 595
Data columns (total 58 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   ID                     596 non-null    int64
 1   Gender                 596 non-null    object
 2   DOB                    596 non-null    object
 3   10percentage           596 non-null    float64
 4   12graduation           596 non-null    int64
 5   12percentage           596 non-null    float64
 6   CollegeID              596 non-null    int64
 7   CollegeTier            596 non-null    int64
 8   Degree                 596 non-null    object
 9   Specialization         596 non-null    object
 10  collegeGPA             596 non-null    float64
 11  CollegeCityID          596 non-null    int64
 12  CollegeCityTier        596 non-null    int64
 13  GraduationYear         596 non-null    int64
 14  English                596 non-null    int64
 15  Logical                596 non-null    int64
 16  Quant                  596 non-null    int64
 17  Domain                 596 non-null    float64
 18  ComputerProgramming    596 non-null    int64
 19  ElectronicsAndSemicon  596 non-null    int64
 20  ComputerScience        596 non-null    int64
 21  MechanicalEngg         596 non-null    int64
 22  ElectricalEngg         596 non-null    int64
 23  TelecomEngg            596 non-null    int64
 24  CivilEngg              596 non-null    int64
```

```
# Display columns of the testing dataset
testing_df.columns
```

> Insights:

- The testing dataset contains 902 entries and 58 columns, matching the structure of both the training and validation datasets.
- There are no missing values in the testing dataset, but there is one missing value in the last_name column of the training dataset, which should be accounted for during data preprocessing.
- The testing data structure and column types are consistent with the training and validation datasets, ensuring reliable testing conditions.

> Considerations:

- Remove Irrelevant Features: Correctly identifies personal identifiers like passport_number, phone_number, and DOB as unnecessary and potentially harmful due to data leakage.
- Redundant Features: Columns such as first_name, last_name, company_name, and cc_number are irrelevant for salary prediction and should be dropped. Note that there is one missing value in the last_name column in the training set, which needs to be addressed.
- Key Features: Appropriately focuses on academic and personality-related features that are likely to influence salary.
- Scaling Required: Identifies the need to scale numerical features to ensure consistency for machine learning models.
- Categorical Encoding: Categorical variables like Degree and Specialization need to be encoded properly to be used in machine learning models.
- Multicollinearity Risk: Rightly points out the correlations between academic features that could cause multicollinearity and inflate model errors.

> Issues found:

- Irrelevant Data: Personal identifiers (e.g., passport_number, phone_number) are potential sources of data leakage and should be removed.
- Negative Values: Negative values in several columns need to be addressed during data cleaning.
- Overfitting Risk: Unique identifiers like passport_number can cause the model to overfit to specific instances, leading to poor generalization.
- Redundant Features: Too many irrelevant columns add complexity and noise to the model, affecting performance.
- Ethical Concerns: Including personal data raises privacy concerns, so these should be removed during preprocessing.
- Feature Engineering Needed: Additional transformations or derivation of new features may improve model accuracy.
- Missing Value in last_name: While there are no missing values in the validation and testing datasets, the training dataset has one missing value in the last_name column, which should be handled during preprocessing.

C.3 Explore Target variable

```
# Set the name of the target column
# The 'Salary' column is identified as the target variable, which we aim to predict
target_name = 'Salary'
```

```
# Display summary statistics of the target column (Salary) in training dataset
training_df[target_name].describe()
```

```
# Create a histogram to visualize the distribution of salaries in the training dataset
salary_distribution_train = alt.Chart(training_df).mark_bar().encode(
    alt.X('Salary', bin=alt.Bin(maxbins=30), title='Salary Range'),
    alt.Y('count()', title='Count')
).properties(
    title='Salary Distribution in Training Dataset'
)

salary_distribution_train.display()
```

```
# Create a Kernel Density Estimate (KDE) plot for the 'Salary' column in the training dataset
sns.kdeplot(training_df['Salary'], shade=True)
plt.title('KDE Plot of Salary in Training Dataset')
plt.xlabel('Salary')
```

> Insights:

- The salary distribution in the training dataset is tightly clustered, with a small range between the minimum and maximum values.
- There is minimal skewness, as the mean and median salaries are close to each other.
- There are no extreme outliers in the data, as the maximum salary is not far from the average.

```
# Display summary statistics of the target column (Salary) in validation dataset
validation_df[target_name].describe()
```

```
# Create a histogram to visualize the distribution of salaries in the validation dataset
salary_distribution_val = alt.Chart(validation_df).mark_bar().encode(
    alt.X('Salary', bin=alt.Bin(maxbins=30), title='Salary Range'),
    alt.Y('count()', title='Count')
).properties(
    title='Salary Distribution in Validation Dataset'
)

salary_distribution_val.display()
```

```
# Create a Kernel Density Estimate (KDE) plot for the 'Salary' column in the validation dataset
sns.kdeplot(validation_df['Salary'], shade=True)
plt.title('KDE Plot of Salary')
plt.xlabel('Salary')
```

> Insights:

- The salary distribution in the validation set is more dispersed compared to the training set, with a higher standard deviation and a wider range of values.
- The presence of higher salary values, as seen in the maximum salary of 95,621, introduces potential outliers, especially compared to the relatively compact range in the training data.
- There is a slight skew towards higher salaries, as the mean is greater than the median.

```
# Display summary statistics of the target column (Salary) in testing dataset
testing_df[target_name].describe()
```

```
# Create a histogram to visualize the distribution of salaries in the testing dataset
salary_distribution_test = alt.Chart(testing_df).mark_bar().encode(
    alt.X('Salary', bin=alt.Bin(maxbins=50), title='Salary Range'),
    alt.Y('count()', title='Count')
).properties(
    title='Salary Distribution in Testing Dataset'
)

salary_distribution_test.display()
```

```
# Create a Kernel Density Estimate (KDE) plot for the 'Salary' column in the testing dataset
sns.kdeplot(testing_df['Salary'], shade=True)
plt.title('KDE Plot of Salary')
plt.xlabel('Salary')
```

> Insights:

- Wide Range: The wide gap between the min (79,335) and max (218,749) salary values suggests a highly dispersed distribution with the potential for high-earning outliers.
- Skewed Distribution: The much higher mean salary compared to the median indicates that the distribution is right skewed due to the presence of a few extremely high salary values.
- High Salaries: The 75th percentile salary (95,621) is close to the maximum in the validation dataset, but the presence of high outliers (above 100,000 and reaching 218,749) pushes the mean up.

Considerations:

- Data Discrepancy: The significant difference in salary distributions between the training, validation, and testing datasets suggests that the datasets might have been sampled from different populations or periods. This discrepancy could affect model performance and generalizability.
- Model Training and Validation: Since the training dataset has a tight salary range while the validation and testing datasets have more dispersed distributions, you might need to adjust the model to account for this variation. Consider stratified sampling or ensuring that the model is robust across different salary ranges.
- Outliers Handling: The testing dataset shows a wide range of salaries and extreme values. These outliers could significantly influence the model's performance and should be carefully analyzed and handled during preprocessing.
- Feature Engineering: The observed skewness and distribution differences may require additional feature engineering to improve model accuracy. For instance, transforming salary data or incorporating features that capture the variability in salary might be beneficial.
- Evaluation Metrics: Given the different distributions, ensure that the evaluation metrics used for the model (such as RMSE) are suitable for assessing performance across the varied salary ranges.

> Issues found:

- Discrepancy in Range: The stark difference in salary ranges among the datasets (e.g., the high salaries in the testing set compared to the training and validation sets) can lead to challenges in model training and validation.
- Potential Outliers: The presence of potential outliers, particularly in the testing dataset, can skew the model's predictions. These outliers could affect the accuracy and reliability of the model.
- Skewed Distributions: The skewness in the salary distribution, especially in the testing dataset, may lead to biased predictions. This issue needs addressing to ensure fair and accurate salary predictions across different data splits.
- Generalization: The model trained on a more compact distribution in the training set might struggle to generalize well to the more dispersed distributions in validation and testing. This could result in poor performance when applied to new data.

C.4 Explore Variables of Interest

C.4.a Feature CollegeGPA

```
# Describing the 'collegeGPA' column in the training dataset
training_df['collegeGPA'].describe()
```

```
# Create a bar chart to visualize the distribution of College GPA in the training dataset
collegeGPA_distribution_train = alt.Chart(training_df).mark_bar().encode(
    alt.X('collegeGPA', bin=alt.Bin(maxbins=30), title='College GPA Range'),
    alt.Y('count()', title='Count')
).properties(
    title='College GPA Distribution in Training Dataset'
)
# Display the GPA distribution chart for the training dataset
collegeGPA_distribution_train.display()
```

```
# Display a scatter plot of Salary vs collegeGPA in training dataset
plt.scatter( training_df['Salary'], training_df['collegeGPA'], c= 'orange')
plt.title('Salary vs collegeGPA in training dataset')
plt.ylabel('collegeGPA')
plt.xlabel('Salary')
```

> Insights:

- The College GPA distribution in the training dataset is mostly concentrated between 65 and 80, with a significant number of values falling between 70 and 75.
- There are few entries with GPAs above 85 or below 60, indicating a narrow spread of academic performance.
- No GPA values exist between 6.63 and 50, indicating that lower GPAs are almost absent.
- The distribution appears somewhat symmetrical, with most students scoring within a small GPA range, reflecting uniform academic performance across the training dataset.

```
# Describing the 'collegeGPA' column in the validation dataset
validation_df['collegeGPA'].describe()
```

```
# Create a bar chart to visualize the distribution of College GPA in the validation dataset
collegeGPA_distribution_val = alt.Chart(validation_df).mark_bar().encode(
    alt.X('collegeGPA', bin=alt.Bin(maxbins=30), title='College GPA Range'),
    alt.Y('count()', title='Count')
).properties(
    title='College GPA Distribution in Validation Dataset'
)
# Display the GPA distribution chart for the validation dataset
collegeGPA_distribution_val.display()
```

```
# Display a scatter plot of Salary vs collegeGPA in validation dataset
plt.scatter(validation_df['Salary'],validation_df['collegeGPA'],  c= 'orange')
plt.title('Salary vs collegeGPA in validation dataset')
plt.ylabel('collegeGPA')
plt.xlabel('Salary')
```

> Insights:

- The College GPA distribution in the validation dataset is concentrated between 65 and 80, with most values clustered between 65 and 75, suggesting a peak around this range.
- There are very few entries with GPAs above 85 or below 60.
- No GPA values exist between 8 and 50, indicating the absence of lower GPAs in this dataset.
- The distribution appears symmetrical, with the majority of students scoring within a narrow GPA range, reflecting uniform academic performance in the validation sample.

```
# Describing the 'collegeGPA' column in the testing dataset
testing_df['collegeGPA'].describe()
```

```
# Create a bar chart to visualize the distribution of College GPA in the testing dataset
collegeGPA_distribution_test = alt.Chart(testing_df).mark_bar().encode(
    alt.X('collegeGPA', bin=alt.Bin(maxbins=30), title='College GPA Range'),
    alt.Y('count()', title='Count')
).properties(
    title='College GPA Distribution in Testing Dataset'
)
# Display the GPA distribution chart for the testing dataset
collegeGPA_distribution_test.display()
```

```
# Display a scatter plot of Salary vs collegeGPA in testing dataset
plt.scatter(testing_df['Salary'],testing_df['collegeGPA'],  c= 'orange')
plt.title('Salary vs collegeGPA in testing dataset')
plt.ylabel('collegeGPA')
plt.xlabel('Salary')
```

> Insights:

- The College GPA distribution in the testing dataset is heavily concentrated between 65 and 80, with most values falling between 70 and 75, indicating a peak around this range.
- The dataset has very few entries having a GPA above 85 or below 60.
- No GPA values exist between 8 and 50, suggesting that the variable has a slight left-skew distribution and we see that lower GPAs are almost absent in this sample.
- The distribution is symmetrical, with a majority of students scoring within a narrow GPA range, suggesting uniformity in academic performance for the group.

> Considerations:

- Distribution Similarity Across Datasets:

The GPA distribution is consistent across training, validation, and testing datasets, with most GPAs concentrated between 65 and 80, peaking between 70 and 75.

- Symmetry in Distribution:

The distribution of GPAs is relatively symmetrical in all datasets, indicating that most students have similar academic performance levels.

- Scaling Requirements:

As collegeGPA is a continuous numerical variable, it may need to be normalized or standardized to ensure better performance of machine learning models, especially for algorithms sensitive to scale.

- Potential Feature Engineering:

Given the concentration of GPA values in a specific range, binning the GPAs into categories (e.g., low, medium, high) or creating interaction terms with other academic features may improve model performance.

- Model Generalization:

The narrow range of GPA values could affect the model's ability to generalize to students with very low or very high GPAs. Special care should be taken when training models to ensure they don't become too focused on this narrow band.

> Issues found:

- Lack of Low GPA Representation:

There is an absence of GPA values between 8 and 50 across all datasets, indicating that the model may struggle to predict salaries for students with lower GPAs due to this underrepresentation.

- Outliers:

The presence of outliers, with some GPAs as low as 6.63 and as high as 99, may impact model performance. These values may need to be treated (e.g., removing, clipping, or scaling) to avoid skewing model results.

- Potential Overfitting to the Common GPA Range:

With most GPAs concentrated between 65 and 80, the model might overfit to this range and struggle to predict accurately for students outside of this range (either very high or very low GPAs).

- Slight Left-Skew:

The distribution shows a slight left-skew, with most students having relatively high GPAs. This skew might require correction or adjustment during feature preprocessing.

C.4.b Feature 12percentage

```
# Describing the '12percentage' column in the training dataset
training_df['12percentage'].describe()
```

```
# Create a bar chart to visualize the distribution of 12th-grade percentage in the training dataset
percentage_distribution_train = alt.Chart(training_df).mark_bar().encode(
    alt.X('12percentage', bin=alt.Bin(maxbins=30), title='12th Grade Percentage Range'),
    alt.Y('count()', title='Count')
).properties(
    title='12th Grade Percentage Distribution in Training Dataset'
)
# Display the 12th-grade percentage distribution chart for the training dataset
percentage_distribution_train.display()
```

```
# Display a scatter plot of Salary vs collegeGPA in training dataset
plt.scatter(training_df['Salary'],training_df['12percentage'],  c = 'green')
plt.title('Salary vs 12percentage in training dataset')
plt.ylabel('12percentage')
plt.xlabel('Salary')
```

> Insights:

- Mean & Median: Average percentage is ~74.79%, with a median close to this, indicating a fairly balanced distribution.
- Spread: Standard deviation is 10.84%, with percentages ranging from 40% to 98.2%, showing significant variability.
- Percentiles: Lower 25% have ≤66.8%, upper 25% have ≥82.8%, highlighting a spread between quartiles.
- Distribution: The bar chart shows a relatively uniform distribution, with a slight concentration in the middle range.
- Outliers: Few extreme low (40%) and high (98.2%) values may need further investigation.

```
# Describing the '12percentage' column in the validation dataset
validation_df['12percentage'].describe()
```

```
# Create a bar chart to visualize the distribution of 12th-grade percentage in the validation dataset
percentage_distribution_val = alt.Chart(validation_df).mark_bar().encode(
    alt.X('12percentage', bin=alt.Bin(maxbins=30), title='12th Grade Percentage Range'),
    alt.Y('count()', title='Count')
).properties(
    title='12th Grade Percentage Distribution in Validation Dataset'
)
# Display the 12th-grade percentage distribution chart for the validation dataset
percentage_distribution_val.display()
```

```
# Display a scatter plot of Salary vs collegeGPA in validation dataset
plt.scatter(validation_df['Salary'],validation_df['12percentage'],  c = 'green')
plt.title('Salary vs 12percentage in validation dataset')
plt.ylabel('12percentage')
plt.xlabel('Salary')
```

> Insights:

- Mean & Median: The average percentage is ~71.25%, with a median of 70%, indicating a slightly lower central tendency compared to the training dataset.
- Spread: The standard deviation is 11.20%, with percentages ranging from 43% to 97.5%, showing a broad variability similar to the training set.
- Percentiles: Lower 25% have ≤63%, upper 25% have ≥79.8%, indicating a spread between quartiles with a slight shift towards lower percentages compared to the training dataset.
- Distribution: The bar chart shows a distribution with a concentration around the middle range, similar to the training dataset but with a lower mean.
- Outliers: Extreme values are present in both low (43%) and high (97.5%) ends, similar to the training dataset.

```
# Describing the '12percentage' column in the testing dataset
testing_df['12percentage'].describe()
```

```
# Create a bar chart to visualize the distribution of 12th-grade percentage in the testing dataset
percentage_distribution_test = alt.Chart(testing_df).mark_bar().encode(
    alt.X('12percentage', bin=alt.Bin(maxbins=30), title='12th Grade Percentage Range'),
    alt.Y('count()', title='Count')
).properties(
    title='12th Grade Percentage Distribution in Testing Dataset'
)
# Display the 12th-grade percentage distribution chart for the testing dataset
percentage_distribution_test.display()
```

```
# Display a scatter plot of Salary vs collegeGPA in testing dataset
plt.scatter(testing_df['Salary'],testing_df['12percentage'],  c = 'green')
plt.title('Salary vs 12percentage in testing dataset')
plt.ylabel('12percentage')
plt.xlabel('Salary')
```

> Insights:

- Mean & Median: The average percentage is ~76.08%, with a median of 77%, indicating a slightly higher central tendency compared to both the training and validation datasets.
- Spread: The standard deviation is 11.29%, with percentages ranging from 45.5% to 98.7%, showing broad variability similar to the other datasets.
- Percentiles: Lower 25% have ≤68%, upper 25% have ≥84.4%, suggesting a wider range at the higher end compared to the validation dataset.
- Distribution: The bar chart shows a concentration around the middle range, with a higher mean compared to the training and validation datasets.
- Outliers: Extreme values are present at both ends (45.5% and 98.7%), consistent with the other datasets.

> Considerations:

- Central Tendency: Testing dataset has a higher average (76.08%) and median (77%) compared to the training (74.79%, 74.60%) and validation datasets (71.25%, 70.00%).
- Spread and Variability: Standard deviations are similar across datasets (10.84% to 11.29%), indicating consistent variability. Percentages range broadly (40% to 98.7%).
- Percentile Distribution: Testing dataset shows higher upper quartile values (84.4%) compared to training (82.8%) and validation datasets (79.8%).
- Distribution Shape: All datasets have a concentration around the middle range, with the testing dataset skewing higher.
- Outliers: Extreme values are present in all datasets (low ~40-45.5%, high ~98-98.7%).

> Issues found:

- Mean Differences: The mean 12th-grade percentage varies across datasets, with the testing dataset showing a notably higher average. This may impact model performance if the test data significantly differs from the training and validation data.
- Consistent Variability: Despite similar standard deviations, the slight differences in mean and median could indicate variations in the distribution of data that might affect model generalization.
- Potential Bias:The differences in central tendency and percentiles suggest potential shifts in data distribution, which could bias the model if not properly accounted for.
- Outliers: Extreme values in all datasets may affect model training and evaluation. Handling these outliers through preprocessing or robust modeling techniques might be necessary.

C.4.c Feature Quant

```
# Describing the 'quant' column in the training dataset
training_df['Quant'].describe()
```

```
#Create a bar chart to visualize the distribution of quant scores in the training dataset
quant_distribution_train = alt.Chart(training_df).mark_bar().encode(
    alt.X('Quant', bin=alt.Bin(maxbins=30), title='Quantitative Score Range'),
    alt.Y('count()', title='Count') ).properties(
         title='Quantitative Score Distribution in Training Dataset' )
#Display the quant score distribution chart for the training dataset
quant_distribution_train.display()
```

```
# Display a scatter plot of Salary vs collegeGPA in training dataset
plt.scatter(training_df['Salary'],training_df['Quant'],  c = 'red')
plt.title('Salary vs Quant in training dataset')
plt.ylabel('Quant')
plt.xlabel('Salary')
```

> Insights:

- Mean & Median: The average is ~518.48, with a median of 524, indicating a balanced central tendency.
- Spread: Standard deviation is 119.51, with values ranging from 145 to 870, showing significant variability.
- Percentiles: Lower 25% have ≤440, upper 25% have ≥605, indicating a spread between quartiles with no major shifts.
- Distribution: The data is normally distributed with no skewness, and the mean and median are close, suggesting symmetry.
- Outliers: Extreme values are present at both ends (145 and 870), but the overall distribution is balanced and normal.

```
# Describing the 'quant' column in the validation dataset
validation_df['Quant'].describe()
```

```
#Create a bar chart to visualize the distribution of quant scores in the validation dataset
quant_distribution_val = alt.Chart(validation_df).mark_bar().encode(
    alt.X('Quant', bin=alt.Bin(maxbins=30), title='Quantitative Score Range'),
    alt.Y('count()', title='Count') ).properties(
        title='Quantitative Score Distribution in Validation Dataset' )
#Display the quant score distribution chart for the validation dataset
quant_distribution_val.display()
```

```
# Display a scatter plot of Salary vs collegeGPA in validation dataset
plt.scatter(validation_df['Salary'],validation_df['Quant'],  c = 'red')
plt.title('Salary vs Quant in validation dataset')
plt.ylabel('Quant')
plt.xlabel('Salary')
```

> Insights:

- Mean & Median: The average is ~467.18, with a median of 470, indicating a balanced central tendency.
- Spread: Standard deviation is 114.79, with values ranging from 135 to 795, showing significant variability.
- Percentiles: Lower 25% have ≤387.5, upper 25% have ≥549.5, indicating a spread between quartiles with no major shifts.
- Distribution: The data is normally distributed, similar to the training dataset, with no skewness.
- Outliers: Extreme values are present at both ends (135 and 795), but the distribution remains balanced and normal.

```
# Describing the 'quant' column in the testing dataset
testing_df['Quant'].describe()
```

```
# Create a bar chart to visualize the distribution of quant scores in the testing dataset
quant_distribution_test = alt.Chart(testing_df).mark_bar().encode(
    alt.X('Quant', bin=alt.Bin(maxbins=30), title='Quantitative Score Range'),
    alt.Y('count()', title='Count') ).properties(
     title='Quantitative Score Distribution in Testing Dataset' )
#Display the quant score distribution chart for the testing dataset
quant_distribution_test.display()
```

```
# Display a scatter plot of Salary vs collegeGPA in testing dataset
plt.scatter(testing_df['Salary'],testing_df['Quant'],  c = 'red')
plt.title('Salary vs Quant in testing dataset')
plt.ylabel('Quant')
plt.xlabel('Salary')
```

> Insights:

- Mean & Median: The average is ~548.21, with a median of 560, showing a higher central tendency compared to the training and validation datasets.
- Spread: Standard deviation is 123.57, with values ranging from 120 to 900, indicating a broad variability.
- Percentiles: Lower 25% have ≤465, upper 25% have ≥626.25, suggesting a wider spread at the higher end compared to the other datasets.
- Distribution: The data is normally distributed, similar to the training and validation datasets, with no significant skewness.
- Outliers: Extreme values are present at both ends (120 and 900), but the distribution remains balanced and normal.

> Considerations:

Central Tendency:

- Validation: Lower mean (467.18) than training (518.48).0
- Testing: Higher mean (548.21) than both training and validation. This can affect model performance if data distributions differ significantly.

Spread and Variability:

- Validation: Slightly less variability compared to training.
- Testing: Greater variability and a wider range, which may impact model generalization.

Percentile Distribution:

- Testing: Shows a broader range at the higher end, potentially affecting model predictions if the training data doesn't include similar extremes.

> Issues found:

- Mean Differences: Significant differences in means across datasets could impact model accuracy and generalization.
- Inconsistent Spread: Variability in quant values differs among datasets, which could challenge model consistency.
- Extreme Values: Testing dataset has more extreme values, which may affect model stability if not addressed properly.

D. Feature Selection

feature_selection_executive_summary :

In this feature selection process, we aimed to identify and retain the most relevant features for predicting the target variable, Salary, while eliminating irrelevant, redundant, or sensitive information.

Two key approaches were used:

- Domain Knowledge Selection: Based on domain expertise and ethical considerations, we removed personal information and features that could lead to overfitting or data leakage, such as personal identifiers (e.g., passport_number, phone_number) and contact details. These features are irrelevant for salary prediction and could violate data privacy regulations. The remaining features focus on academic performance, professional qualifications, and cognitive scores, which are more likely to influence salary outcomes.
- Correlation Analysis: We performed a statistical correlation analysis to determine the strength of relationships between numerical features and the target variable, Salary. Features with strong positive correlations, such as Quant, 10percentage, 12percentage, and English, were prioritized for retention. On the other hand, features with weak and negative correlations, such as CollegeTier, CollegeCityTier, and GraduationYear, were deemed less significant for prediction and were excluded from the final model.

This methodical feature selection process ensures that the dataset includes only the most relevant variables for predicting salary, improving model performance and interpretability while adhering to privacy standards.

Final Selected Features:

The final set of features retained for modeling includes:
Gender,
10percentage,
12percentage,
Degree,
Specialization,
collegeGPA,
English,
Logical,
Quant,
Domain,
ComputerProgramming,
ElectronicsAndSemicon,
ComputerScience,
MechanicalEngg,
ElectricalEngg,
TelecomEngg,
CivilEngg,
Personality Test Scores (conscientiousness, agreeableness, extraversion, neuroticism, openness_to_experience),
Salary (Target variable).

D.1 Approach 1 : Domain Knowledge Selection

> Rationale:

- Privacy and Ethical Concerns: Personal information columns such as passport_number, phone_number, and email are removed to address privacy and ethical concerns, ensuring the dataset complies with data protection regulations.
- Avoid Overfitting and Data Leakage: Personal identifiers and contact details are not useful for salary prediction and could lead to overfitting or data leakage. Including such information might inadvertently allow the model to memorize specific details rather than learning generalizable patterns.
- Relevance: By focusing on features that are directly related to academic and professional attributes, the model is more likely to identify factors that genuinely influence salary, improving its predictive performance and generalizability.

```
# Display columns of the training dataset
training_df.columns
```

```
# Drop unnecessary or irrelevant features from the training, validation, and testing datasets
features_to_drop = [
    'ID','Gender', 'DOB', 'CollegeID', 'CollegeCityID','profile', 'locale', 'prefix', 'first_name', 'last_name','full_name
    'passport_number','passport_expiry', 'phone_number', 'email', 'secondary_address','building_number', 'street_name', 's
    'city', 'postcode','state_abbr', 'address', 'cc_number', 'cc_expiry', 'cc_security_code','company_name', 'company_suff
]

# Drop the specified features from all datasets
training_df = training_df.drop(columns=features_to_drop)
validation_df = validation_df.drop(columns=features_to_drop)
testing_df = testing_df.drop(columns=features_to_drop)
```

> Results:

- The cleaned dataset now includes features that are pertinent to salary prediction, such as academic performance (collegeGPA, 12percentage), professional qualifications (Degree, Specialization), and test scores (Logical, Quant).
- Removing irrelevant, bias and sensitive features results in a more focused dataset, reducing noise and the risk of model overfitting.
- The final set of features is expected to enhance the model's ability to predict salaries accurately while adhering to privacy standards and ethical considerations.

D.2 Approach 2 Correlation Analysis

> Rationale:

- This approach leverages statistical correlation to assess the strength of the relationships between numerical features and the target variable, Salary.
- By analyzing how closely features correlate with Salary, we can prioritize features with stronger correlations, as these are more likely to influence the outcome.
- Features with weaker or negative correlations may be excluded as they contribute less to the prediction of Salary, improving the model's performance and interpretability by focusing on the most relevant variables.

```
# Identify numeric columns for correlation analysis
numeric_cols = [col for col in training_df.columns if training_df[col].dtype != 'object']

# Compute the correlation matrix for numeric features in the training dataset
correlation_matrix = training_df[numeric_cols].corr()
```

```
# Plot a heatmap of the correlation matrix to visualize relationships between numeric features
plt.figure(figsize=(20,10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap of Numeric Features in Training Data")
```

```
# Get the correlation of all features with the 'Salary' feature, sorted in descending order
salary_correlation = correlation_matrix['Salary'].sort_values(ascending=False)
salary_correlation
```

> Results:

- The correlation analysis identified features such as Quant, 10percentage, 12percentage, and English as having a strong positive correlation with Salary. These variables are likely important predictors of salary outcomes.
- Conversely, features such as agreeableness, civilEngg, GraduationYear, and others showed weak or negative correlations with Salary.
- However, only features like CollegeTier, CollegeCityTier, 12graduation, and GraduationYear will be removed, as they exhibit little predictive power. The other features, including personality test scores and Specialization, could still play a role in salary determination by recruiters and will be retained for the model.

D.3 Final Selection of Features

```
# List of features selected for further analysis or modeling
features_list = ['10percentage', '12percentage','Degree', 'Specialization', 'collegeGPA',
                 'English', 'Logical', 'Quant', 'Domain','ComputerProgramming', 'ElectronicsAndSemicon',
                 'ComputerScience','MechanicalEngg', 'ElectricalEngg', 'TelecomEngg', 'CivilEngg','conscientiousness',
                 'agreeableness', 'extraversion', 'nueroticism','openess_to_experience', 'Salary']
```

E. Data Cleaning

data_cleaning_executive_summary

1. Copying Datasets:

- Datasets were copied to ensure original data remained unchanged during cleaning.

2. Fixing Missing Values:

- Rationale: Missing values can skew predictions and require handling for data integrity.
- Results: No missing values were found in any dataset, so no imputation was needed.

3. Fixing Data Inconsistencies:

- Rationale: Inconsistent data can affect analysis accuracy. Standardization is crucial.
- Results: Categorical entries were standardized across all datasets, ensuring uniformity.

4. Fixing Duplicated Values:

- Rationale: Duplicates can bias analysis. Removing them maintains dataset integrity.
- Results: No duplicated values were found in any dataset.

E.1 Copy Datasets

```
# Create copy of datasets
# Do not change this code

training_df_clean = training_df[features_list].copy()
validation_df_clean = validation_df[features_list].copy()
testing_df_clean = testing_df[features_list].copy()
```

E.2 Fixing Missing Values

> Rationale:

- Missing values can lead to biased or inaccurate model predictions.
- Imputation or removal is necessary to ensure the integrity of the dataset.

```
 # Display the count of missing values in the training dataset
testing_df_clean.isna().sum()
```

```
 # Display the count of missing values in the validation dataset
validation_df_clean.isna().sum()
```

```
 # Display the count of missing values in the testing dataset
testing_df_clean.isna().sum()
```

> Results:

- All three datasets did not have any missing values, so no imputation was required during the data cleaning process.

E.3 Fixing Data Inconsistencies

> Rationale:

- Inconsistent data entries can skew results and negatively impact model performance.
- Standardizing and correcting inconsistencies is crucial to ensure uniformity in categorical variables.
- Inconsistent values like -1 can indicate invalid or missing data and should be replaced with appropriate substitutes, such as 0 where necessary.

```
# The function converts text to lowercase and removes leading/trailing whitespace
def standardize_text(df, text_columns):
    for col in text_columns:
        df[col] = df[col].str.lower().str.strip()
    return df
# Identify columns with text data (categorical variables) that need standardization
text_columns = [col for col in training_df_clean.columns if training_df_clean[col].dtype == 'object']

# Apply standardization to text columns in all datasets
training_df_clean = standardize_text(training_df_clean, text_columns)
validation_df_clean = standardize_text(validation_df_clean, text_columns)
testing_df_clean = standardize_text(testing_df_clean, text_columns)
```

```
# Replace inconsistent values (-1) with 0 in all datasets
training_df_clean.replace(-1,0,inplace=True)
validation_df_clean.replace(-1,0,inplace=True)
testing_df_clean.replace(-1,0,inplace=True)
```

> Results:

- Data inconsistencies were resolved by standardizing categorical entries (e.g., converting text fields to uniform case) in the training, validation, and testing datasets.
- Inconsistent values of -1 were successfully replaced with 0 across all datasets, ensuring uniform and clean categorical fields, as well as corrected numerical data.

E.4 Fixing Duplicated Values

> Rationale:

- Duplicated values in a dataset can lead to biased analysis and inaccurate model performance by over-representing certain records.
- Identifying and removing duplicates ensures that each data point contributes only once, maintaining the integrity of the dataset.

```
# Display the Count of duplicated rows in the training dataset
training_df_clean.duplicated().sum()
```

```
# Display the Count of duplicated rows in the validation dataset
validation_df_clean.duplicated().sum()
```

```
# Display the Count of duplicated rows in the testing dataset
testing_df_clean.duplicated().sum()
```

> Results:

- No duplicated values were found in any of the datasets.

F. Feature Engineering

feature_engineering_executive_summary

- Feature engineering focused on creating new interaction terms and categorizing variables to enhance model performance and uncover relationships between features that might otherwise go unnoticed.
- Specifically, the interaction terms aim to capture the compounded effects of academic performance across various stages, while the categorical transformation of GPA simplifies the interpretation of its relationship with the target variable (salary).
- These engineered features, added to the training, validation, and testing datasets, are designed to enrich the predictive power of the model by leveraging combined and categorized factors that influence salary potential.

F.1 Copy Datasets

```
# Create copy of datasets
# Do not change this code

training_df_eng = training_df_clean.copy()
validation_df_eng = validation_df_clean.copy()
testing_df_eng = testing_df_clean.copy()
```

F.2 New Feature GPA_Quant_Interaction

> Rationale:

- The interaction between college GPA and quantitative skills is designed to capture how the combination of academic performance and quantitative aptitude influences salary predictions.
- High academic scores coupled with strong quantitative skills might have a more pronounced effect on a candidate's salary potential than considering either feature in isolation.
- By multiplying these two factors, the model can better capture the interaction and its potential impact.

```
# Create interaction feature 'GPA_Quant_Interaction' by multiplying 'collegeGPA' with 'Quant'
training_df_eng['GPA_Quant_Interaction'] = training_df_eng['collegeGPA'] * training_df_eng['Quant']
validation_df_eng['GPA_Quant_Interaction'] = validation_df_eng['collegeGPA'] * validation_df_eng['Quant']
testing_df_eng['GPA_Quant_Interaction'] = testing_df_eng['collegeGPA'] * testing_df_eng['Quant']
```

> Results:

- The new feature GPA_Quant_Interaction was successfully added to the training, validation, and testing datasets.
- This interaction provides a combined measure of academic success and quantitative proficiency, potentially enhancing the model's ability to predict salary based on the joint effect of these two attributes.

F.3 New Feature 10_12_Percentage_Interaction

> Rationale:

- Creating an interaction term between the 10percentage and 12percentage features allows us to capture any multiplicative relationship between these two academic performance metrics.
- By multiplying the percentages from 10th and 12th grades, we aim to assess whether a combination of strong (or weak) performances across both levels has a compounded impact on the target variable, such as salary.
- This interaction feature may reveal patterns that are not apparent when looking at the two percentages individually, and could be an indicator of consistent academic performance across time.

```
# Create interaction feature '10_12_Percentage_Interaction' by multiplying '10percentage' with '12percentage'
training_df_eng['10_12_Percentage_Interaction'] = training_df_eng['10percentage'] * training_df_eng['12percentage']
validation_df_eng['10_12_Percentage_Interaction'] = validation_df_eng['10percentage'] * validation_df_eng['12percentage']
testing_df_eng['10_12_Percentage_Interaction'] = testing_df_eng['10percentage'] * testing_df_eng['12percentage']
```

> Results:

- The interaction term 10_12_Interaction was successfully created by multiplying the 10th-grade percentage with the 12th-grade percentage.
- This new feature has been added to the training, validation, and testing datasets.
- It provides additional insight into the relationship between early academic performance and the target variable, offering the potential to improve model performance by accounting for the combined effect of these two metrics.

F.4 New Feature GPA_Category

> Rationale:

- Categorizing GPA into distinct groups (e.g., "Low," "Medium," "High") allows us to simplify the relationship between GPA and salary.
- By transforming a continuous variable like collegeGPA into a categorical feature, we can potentially uncover patterns that may not be evident when analyzing GPA as a raw numerical value.
- This approach can be particularly useful when there are thresholds in the GPA distribution that correspond to different salary outcomes, making it easier to interpret and analyze.

```
# Define a function to categorize GPA into 'Low', 'Medium', or 'High' based on specified thresholds
def categorize_gpa(gpa):
    if gpa < 50:
        return 'Low'
    elif 50 <= gpa < 70:
        return 'Medium'
    else:
        return 'High'

# Apply the 'categorize_gpa' function to the 'collegeGPA' column to create a new categorical feature 'GPA_Category'
training_df_eng['GPA_Category'] = training_df_eng['collegeGPA'].apply(categorize_gpa)
validation_df_eng['GPA_Category'] = validation_df_eng['collegeGPA'].apply(categorize_gpa)
testing_df_eng['GPA_Category'] = testing_df_eng['collegeGPA'].apply(categorize_gpa)
```

> Results:

- The GPA values in the dataset were successfully categorized into three distinct groups: "Low" for GPA below 50, "Medium" for GPA between 50 and 70, and "High" for GPA above 70.
- This new categorical feature, "GPA Category," was added to the training, validation, and testing datasets.
- It provides a new way to interpret the academic performance of individuals, allowing the model to capture trends based on GPA groupings that may impact salary predictions.

G. Data Preparation for Modeling

modeling_preparation_executive_summary

In preparation for modeling, the data was preprocessed through a series of transformations to ensure that it was suitable for machine learning algorithms.

- First, the datasets were split into training, validation, and test sets, with the target variable separated.
- Next, categorical features were label encoded to convert them into numerical values, making them compatible with machine learning models.
- Following this, the features were standardized to have a mean of 0 and a standard deviation of 1, ensuring all variables were on a similar scale.
- Finally, the standardized data was converted back into DataFrames to preserve column names and metadata, improving readability and facilitating further analysis.

These steps ensure that the data is appropriately transformed, consistent, and ready for modeling, leading to more effective and accurate predictions.

G.1 Split Datasets

```
# Create copy of datasets
# Do not change this code

X_train = training_df_eng.copy()
X_val = validation_df_eng.copy()
X_test = testing_df_eng.copy()

y_train = X_train.pop(target_name)
y_val = X_val.pop(target_name)
y_test = X_test.pop(target_name)
```

G.2 Data Transformation: Label Encoding of Categorical Features

> Rationale:

- Label encoding converts categorical variables into numerical values, making them suitable for machine learning algorithms that require numerical input.
- This transformation allows the model to process categorical features effectively and can improve model interpretability and performance.

```
# Import LabelEncoder from sklearn
from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder for transforming categorical features
le = LabelEncoder()

# Identify categorical features in the training dataset
categorical_features = [col for col in X_train.columns if X_train[col].dtype == 'object']

# Apply Label Encoding to categorical features in training, validation, and test datasets
for i in categorical_features:
    X_train[i] = le.fit_transform(X_train[i])
    X_val[i] = le.fit_transform(X_val[i])
    X_test[i] = le.fit_transform(X_test[i])
```

> Results:

- The Categorical features were successful encoded and converted to numeric values in all datasets.
- Encoding was consistently maintained across training, validation, and test sets.

G.3 Data Transformation: Standardization of Features

> Rationale:

- Standardizing features ensures that all input variables are on a similar scale, which can improve the performance and convergence of many machine learning algorithms.
- This step is particularly important for algorithms sensitive to feature scaling, such as those based on distance metrics or gradient-based methods.

```
# Import StandardScaler from sklearn
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler for feature scaling
scaler = StandardScaler()

# Scale features of the training, validation, and test datasets to have zero mean and unit variance
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

> Results:

- The features were successfully standardized, resulting in each feature having a mean of 0 and a standard deviation of 1.
- This transformation ensures that features contribute equally to the model, improving the effectiveness of distance-based and gradient-based algorithms.

G.4 Data Transformation: Conversion to DataFrame Post-Scaling

> Rationale:

Converting the scaled arrays back to DataFrames:

- Enhances Readability: DataFrames provide a clear, structured format with column names.
- Facilitates Analysis: Easier integration with tools and libraries that expect DataFrames.
- Preserves Metadata: Retains column names and data structure for subsequent processing.

```
# Convert the scaled arrays back to DataFrames with the original column names for consistency
X_train= pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_val = pd.DataFrame(X_val_scaled, columns=X_val.columns)
X_test = pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

> Results:

After the conversion:

- X_train: The training data is now in DataFrame format with scaled features and original column names.
- X_val: The validation data is similarly converted to a DataFrame with scaled features, matching the structure of the training data.
- X_test: The test data is also converted to a DataFrame, ensuring consistency with the training and validation sets in terms of feature names and structure.

H. Save Datasets

```
# Save training set
# Do not change this code

X_train.to_csv('X_train.csv', index=False)
y_train.to_csv('y_train.csv', index=False)
```

```
# Save validation set
# Do not change this code

X_val.to_csv('X_val.csv', index=False)
y_val.to_csv('y_val.csv', index=False)
```

```
# Save testing set
# Do not change this code

X_test.to_csv('X_test.csv', index=False)
y_test.to_csv('y_test.csv', index=False)
```

I. Assess Baseline Model

baseline_model_executive_summary :

The baseline model serves as a fundamental benchmark by predicting the central value (mean) of the training target for all instances in the training, validation, and test datasets. This model provides a reference point to evaluate the performance of more advanced predictive models.

Performance Metrics:

- Training Set RMSE: 2461.76
- Validation Set RMSE: 6053.32
- Test Set RMSE: 15439.72

These RMSE values highlight the prediction errors of the simplest model, setting a baseline for comparison with more sophisticated models.

I.1 Simulate Predictions with Baseline Model

> Rationale:

The purpose of the baseline model is to provide a simple benchmark by predicting the central tendency of the target variable (e.g., mean or median). This approach helps in evaluating how much better (or worse) a complex machine learning model performs in comparison.

```
# Define central value for baseline model
# This central value (mean of the training target) will be used as the prediction for all instances

y_central = y_train.mean()
```

```
# Generate predictions for training, validation, and test datasets using the baseline model

train_preds = [y_central] * len(y_train)
val_preds = [y_central] * len(y_val)
test_preds = [y_central] * len(y_test)
```

I.2 Selection of Performance Metrics

> Rationale:

- Performance metrics allow us to quantify how well the model is performing.
- For regression tasks, common metrics like Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) provide insights into the magnitude of prediction errors. As given in the assignment requirements we will use RMSE as performance Metrics.
- These metrics are crucial for comparing the baseline with more sophisticated models.

```
# Import mean squared error function for performance evaluation

from sklearn.metrics import mean_squared_error
```

I.2 Baseline Model Performance

```
# Calculate the Root Mean Squared Error (RMSE) for the baseline model on training, validation, and test datasets

baseline_rmse_train = mean_squared_error(y_train, train_preds, squared=False)

baseline_rmse_val = mean_squared_error(y_val, val_preds, squared=False)

baseline_rmse_test = mean_squared_error(y_test, test_preds, squared=False)
```

```
# Print the RMSE values to evaluate the baseline model's performance
print(baseline_rmse_train)
print(baseline_rmse_val)
print(baseline_rmse_test)
```

```
2461.755232794408
6053.321938989372
15439.716269435767
```

Prediction vs. Actual plot

```
plt.figure(figsize=(18, 6))

# Prediction vs Actual for Training Set
plt.subplot(1, 3, 1)
plt.scatter(y_train, train_preds, color='blue')
plt.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], 'r--', lw=2)  # Diagonal line
plt.xlabel('Actual Values (Train)')
plt.ylabel('Predicted Values')
plt.title('Prediction vs. Actual for Training Set (Baseline Model)')

# Prediction vs Actual for Validation Set
plt.subplot(1, 3, 2)
plt.scatter(y_val, val_preds, color='green')
plt.plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()], 'r--', lw=2)  # Diagonal line
plt.xlabel('Actual Values (Validation)')
plt.ylabel('Predicted Values')
plt.title('Prediction vs. Actual for Validation Set (Baseline Model)')

# Prediction vs Actual for Test Set
plt.subplot(1, 3, 3)
plt.scatter(y_test, test_preds, color='red')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)  # Diagonal line
plt.xlabel('Actual Values (Test)')
plt.ylabel('Predicted Values')
plt.title('Prediction vs. Actual for Test Set (Baseline Model)')

plt.tight_layout()
```

> Results:

The baseline model performance metrics provide a benchmark for evaluating the effectiveness of more complex models.

- Training Set RMSE: 2461.755232794408
- Validation Set RMSE: 6053.321938989372
- Test Set RMSE: 15439.716269435767

These results highlight the prediction error when using a simple central value, forming the foundation for comparison with future models.