ENGINEERING GRADUATE SALARY

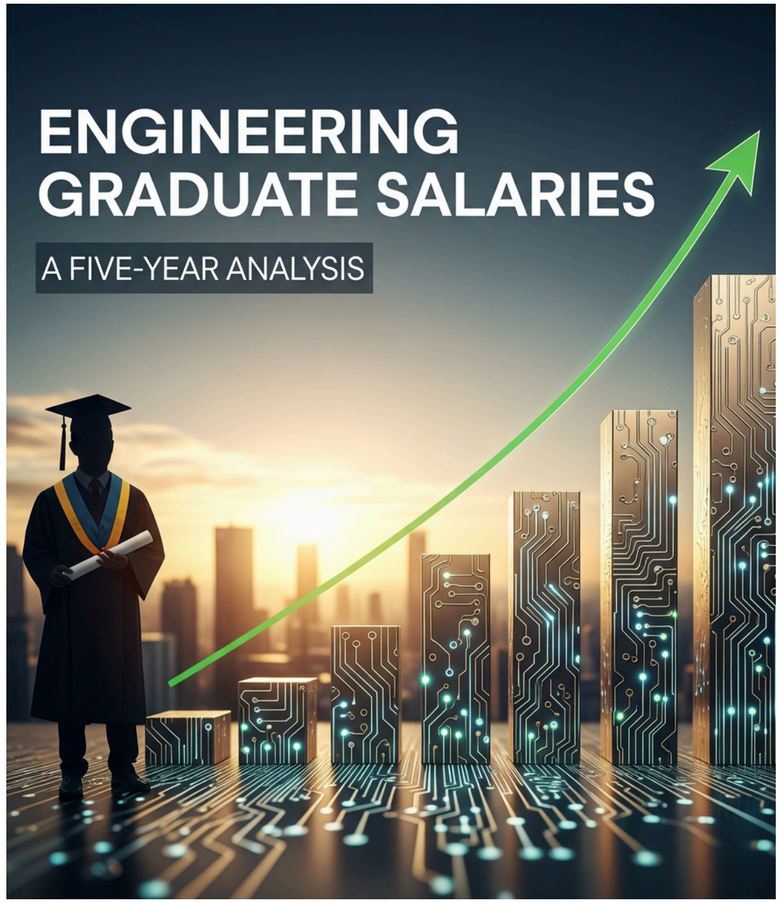
MACHINE LEARNING (CLASSIFICATION)

TEAM : 4

Team members:

M. Shiva Charan
T. Kavya
N. Aravind
E. Sai Kumar

CONTENTS



DATA DESCRIPTION

This dataset records the salaries of engineering graduates along with related attributes such as specialization, college tier, location, job role, and experience. The goal is to classify salary data into two categories: High and Low, enabling predictive analysis using machine learning.

OBJECTIVE

To build a classification model that predicts whether an engineering graduate’s salary falls into the high or low category based on their profile.

PATH

Implementing various machine learning models to find the best model , to predict the accuracy of the dataset.

Data And Data Quality Check

About The Data :

- Number of Instances: 2,998 engineering graduate records
- Number of Attributes: 33 input features + 1 output attribute (Salary)
- Target Variable: Salary – Annual CTC offered to the candidate (in INR)

Categorical Attributes:

(Columns with non-numeric values)

Gender → 2 categories (m, f)
DOB → 1633 unique birth dates
10board → 221 unique boards (most common: CBSE, 1026 students)
12board → 277 unique boards (most common: CBSE, 1039 students)
Degree → 4 categories (most common: B.Tech/B.E., 2757 students)
Specialization → 42 specializations (top: Electronics & Communication Engg, 670 students)
CollegeState → 26 states (top: Uttar Pradesh, 698 students)

Numerical Attributes

(Columns with numeric values)

Academic Info:10percentage, 12percentage, collegeGPA
GraduationYear → mostly 2012–2014
Test Scores: English, Logical, Quant Domain, ComputerProgramming, ElectronicsAndSemicon, ComputerScience, etc.
Personality Traits (Big Five)**:**conscientiousness, agreeableness, extraversion, nueroticism, openess_to_experience
Other IDs & Indicators : ID, CollegeID, CollegeTier, CollegeCityID, CollegeCityTier
Target Variable:
Salary → mean: ₹3,05,175 (range: ₹35,000 – ₹40,00,000)

```
0   Gender                  2998 non-null   object
1   10percentage            2998 non-null   float64
2   12graduation            2998 non-null   int64
3   12percentage            2998 non-null   float64
4   CollegeTier             2998 non-null   int64
5   Degree                  2998 non-null   object
6   Specialization          2998 non-null   object
7   collegeGPA              2998 non-null   float64
8   CollegeCityID           2998 non-null   int64
9   CollegeCityTier         2998 non-null   int64
10  CollegeState            2998 non-null   object
11  GraduationYear          2998 non-null   int64
12  English                 2998 non-null   int64
13  Logical                 2998 non-null   int64
14  Quant                   2998 non-null   int64
15  Domain                  2998 non-null   float64
16  ComputerProgramming     2998 non-null   int64
17  ElectronicsAndSemicon   2998 non-null   int64
18  ComputerScience         2998 non-null   int64
19  MechanicalEngg          2998 non-null   int64
20  ElectricalEngg          2998 non-null   int64
21  TelecomEngg             2998 non-null   int64
22  CivilEngg               2998 non-null   int64
23  conscientiousness       2998 non-null   float64
24  agreeableness           2998 non-null   float64
25  extraversion            2998 non-null   float64
26  nueroticism             2998 non-null   float64
27  openess_to_experience   2998 non-null   float64
28  Salary                  2998 non-null   int64
29  Age                     2998 non-null   int32
```

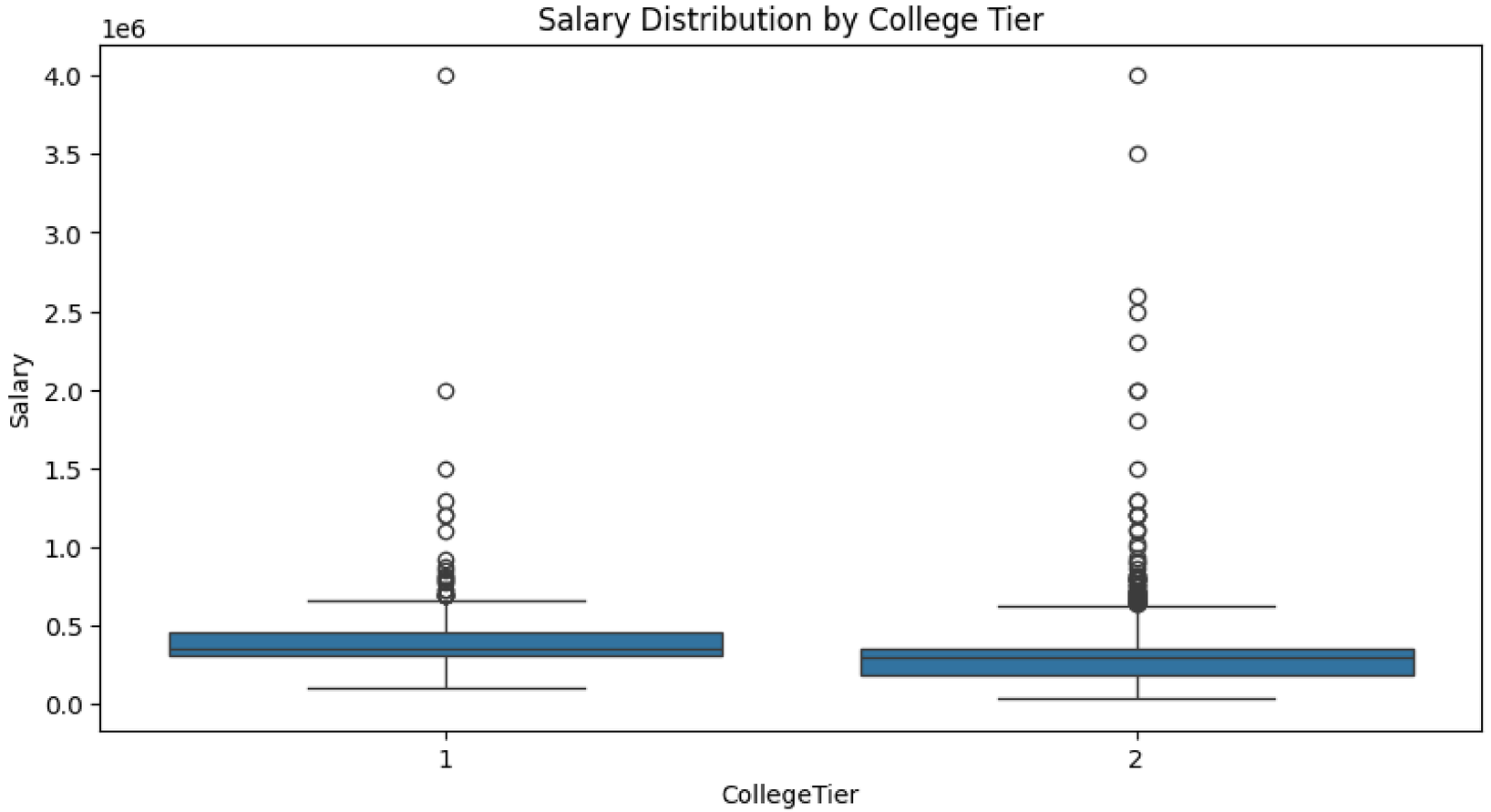
All columns have no missing values and are clearly defined with their respective data types (integer, float, or object). This schema provides a comprehensive overview of the data available for analysis.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2998 entries, 0 to 2997
Data columns (total 30 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Gender                  2998 non-null   object
 1   10percentage            2998 non-null   float64
 2   12graduation            2998 non-null   int64
 3   12percentage            2998 non-null   float64
 4   CollegeTier             2998 non-null   int64
 5   Degree                  2998 non-null   object
 6   Specialization          2998 non-null   object
 7   collegeGPA              2998 non-null   float64
 8   CollegeCityID           2998 non-null   int64
 9   CollegeCityTier         2998 non-null   int64
 10  CollegeState            2998 non-null   object
 11  GraduationYear          2998 non-null   int64
 12  English                 2998 non-null   int64
 13  Logical                 2998 non-null   int64
 14  Quant                   2998 non-null   int64
 15  Domain                  2998 non-null   float64
 16  ComputerProgramming     2998 non-null   int64
 17  ElectronicsAndSemicon   2998 non-null   int64
 18  ComputerScience         2998 non-null   int64
 19  MechanicalEngg          2998 non-null   int64
 20  ElectricalEngg          2998 non-null   int64
 21  TelecomEngg             2998 non-null   int64
 22  CivilEngg               2998 non-null   int64
 23  conscientiousness       2998 non-null   float64
 24  agreeableness           2998 non-null   float64
 25  extraversion            2998 non-null   float64
 26  nueroticism             2998 non-null   float64
 27  openess_to_experience   2998 non-null   float64
 28  Salary                  2998 non-null   int64
 29  Age                     2998 non-null   int32
dtypes: float64(9), int32(1), int64(16), object(4)
memory usage: 691.1+ KB
```

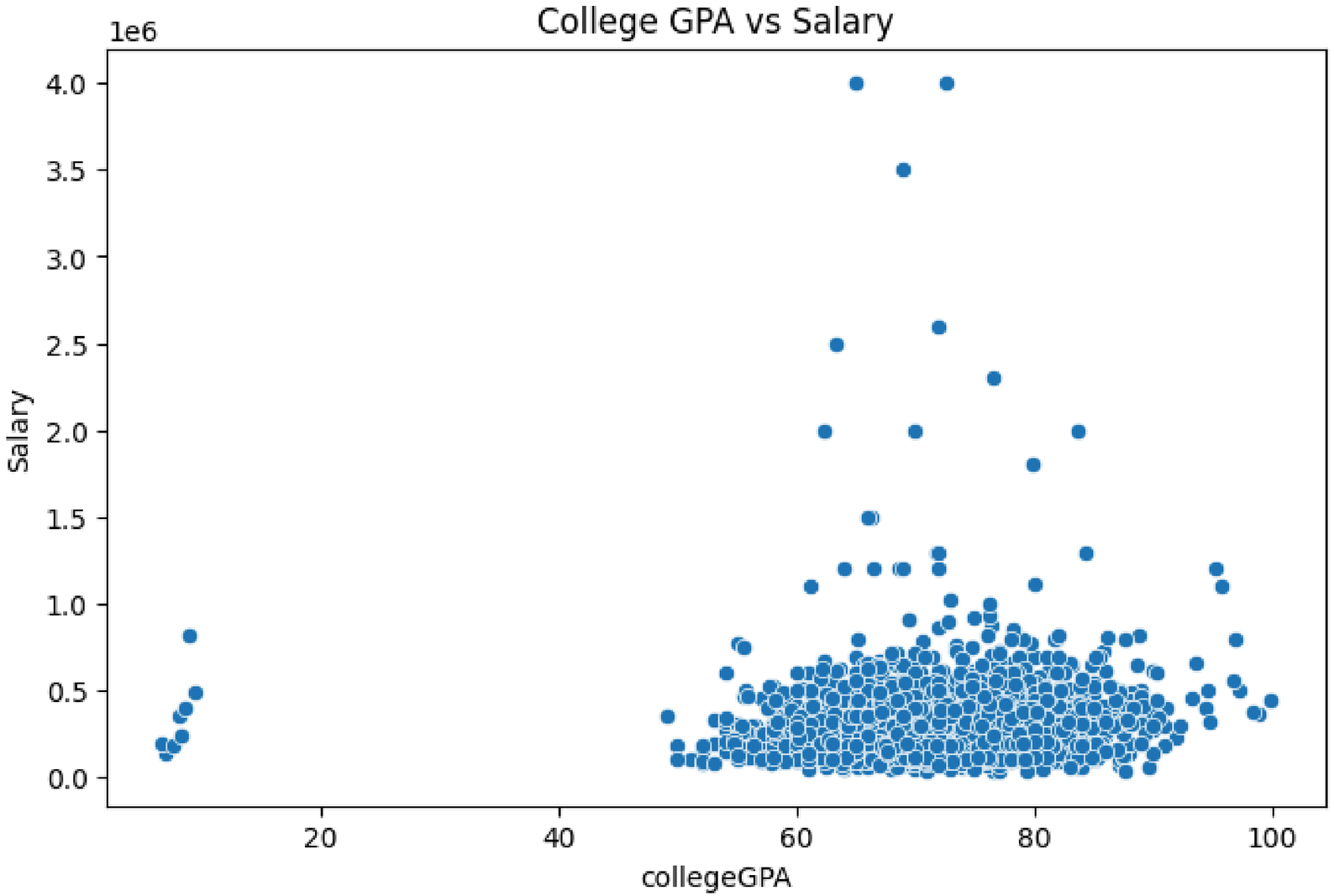
Age was calculated from the 'DOB' column, which was then dropped.Irrelevant columns like 'CollegeID', 'ID', '10board', and '12board' were removed.

One-hot encoding, a data preprocessing technique. It converts all categorical (object-type) columns in a DataFrame into numerical (dummy) columns. This is a necessary step because most machine learning algorithms require numerical input. The original categorical columns are then dropped and replaced by the new numerical ones.

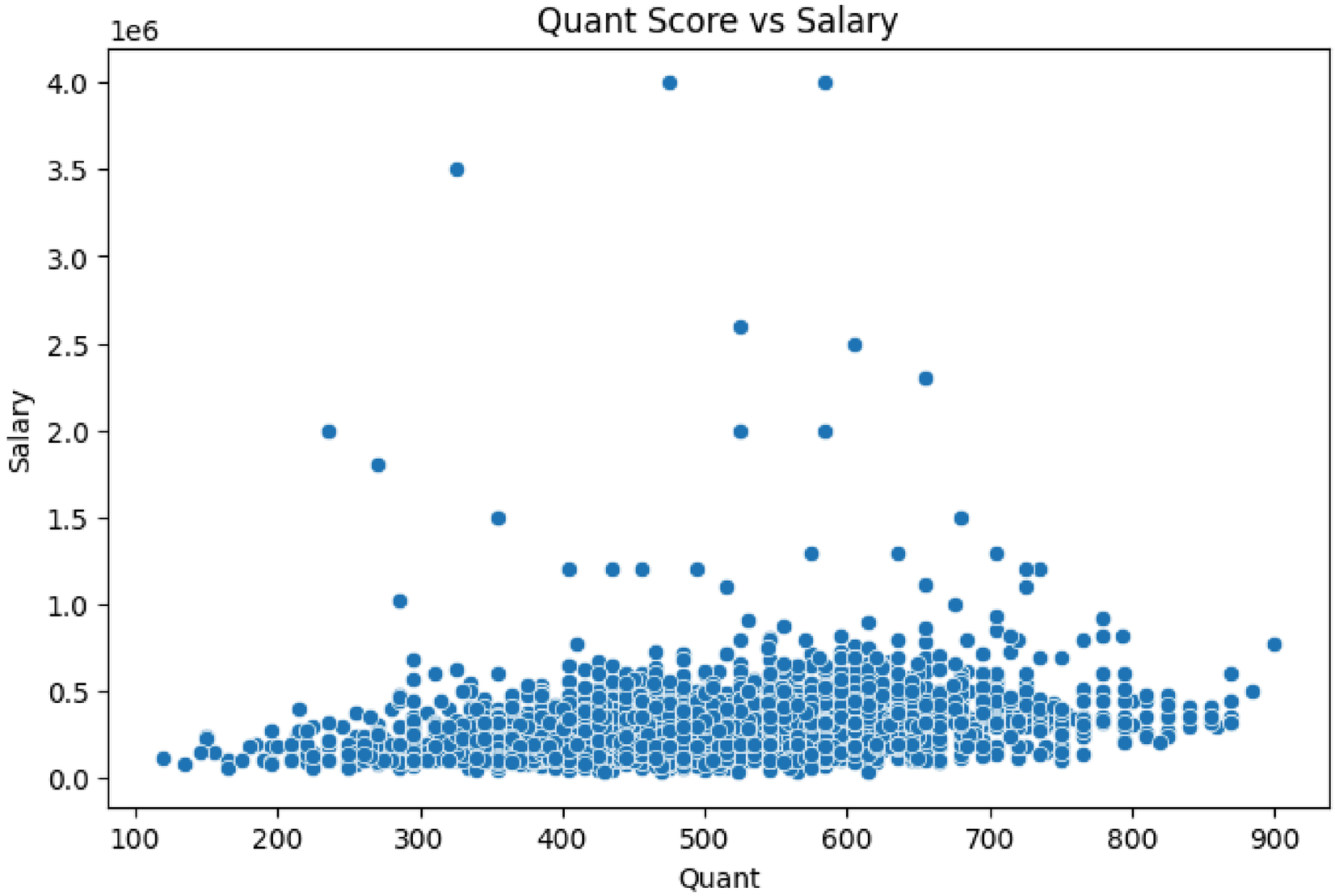
EXPLORATORY DATA ANALYSIS(EDA)



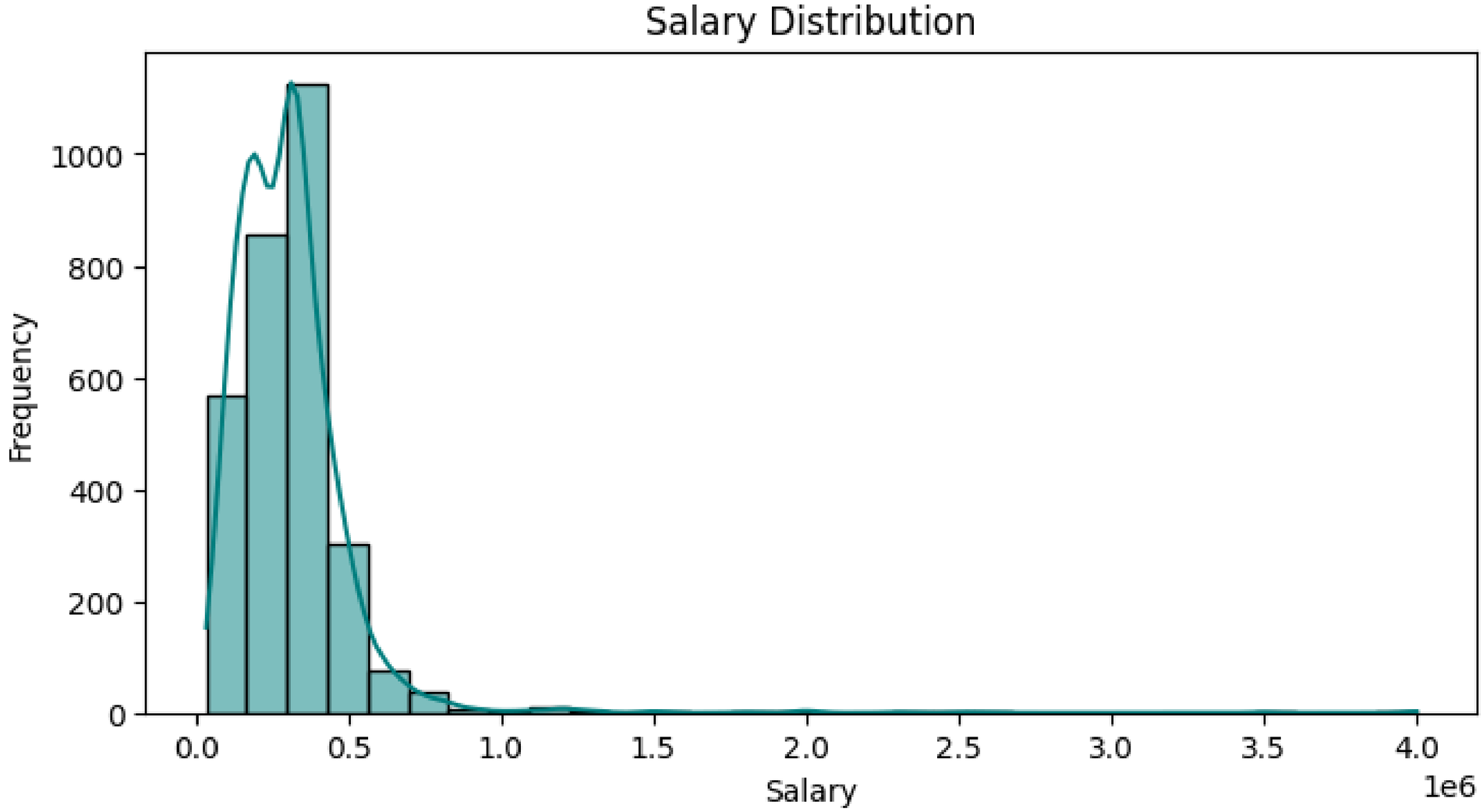
This box plot compares salary distributions between two college tiers. The key takeaway is that Tier 1 colleges have a higher median salary and a wider range of high-paying jobs compared to Tier 2 colleges. While both tiers have similar lower-end salaries, graduates from Tier 1 have a significantly better chance of earning a very high income.



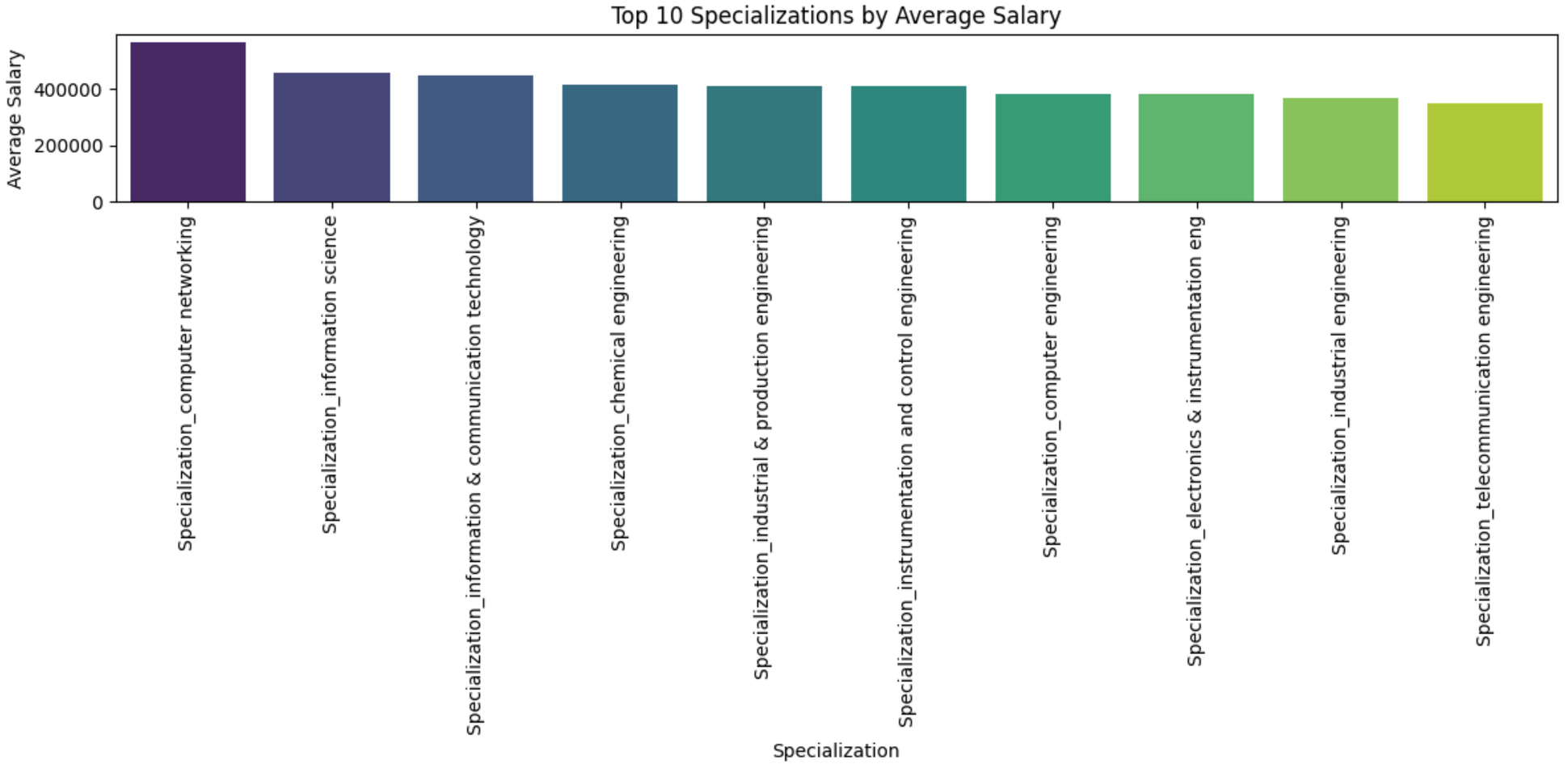
There's a weak positive correlation between College GPA and salary.
The data is heavily clustered in the GPA range of 60-90.
A few individuals with average GPAs are outliers with exceptionally high salaries.



There is a positive correlation between Quant score and salary, but the relationship is weak. A high Quant score does not guarantee a high salary, as evidenced by the wide scatter and numerous outliers.

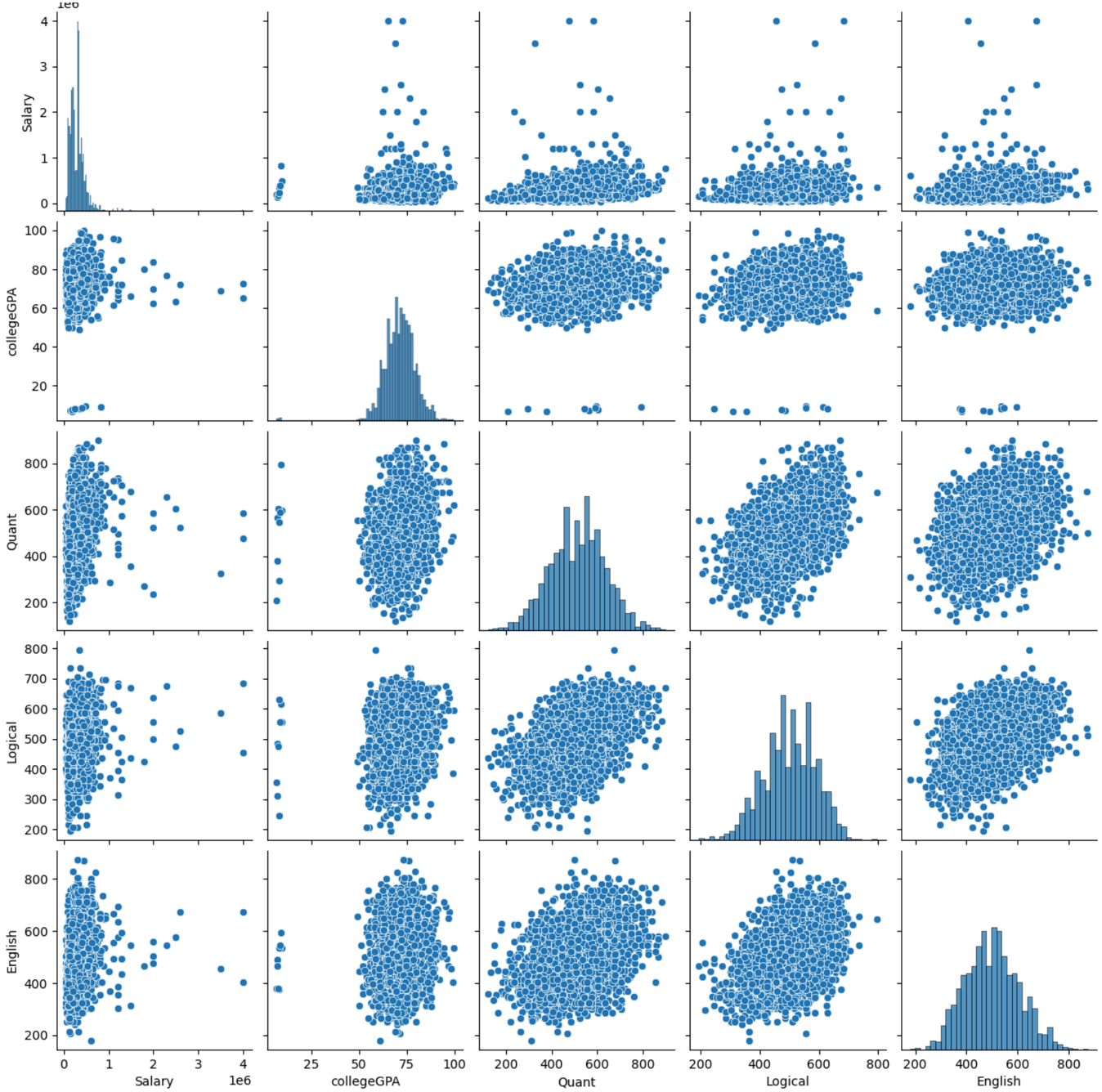


The distribution of salaries is highly right-skewed, meaning most people earn a lower salary, and a small number of individuals earn a much higher salary. The majority of salaries are concentrated between 0 and 500,000, with the frequency dropping off sharply as the salary increases.



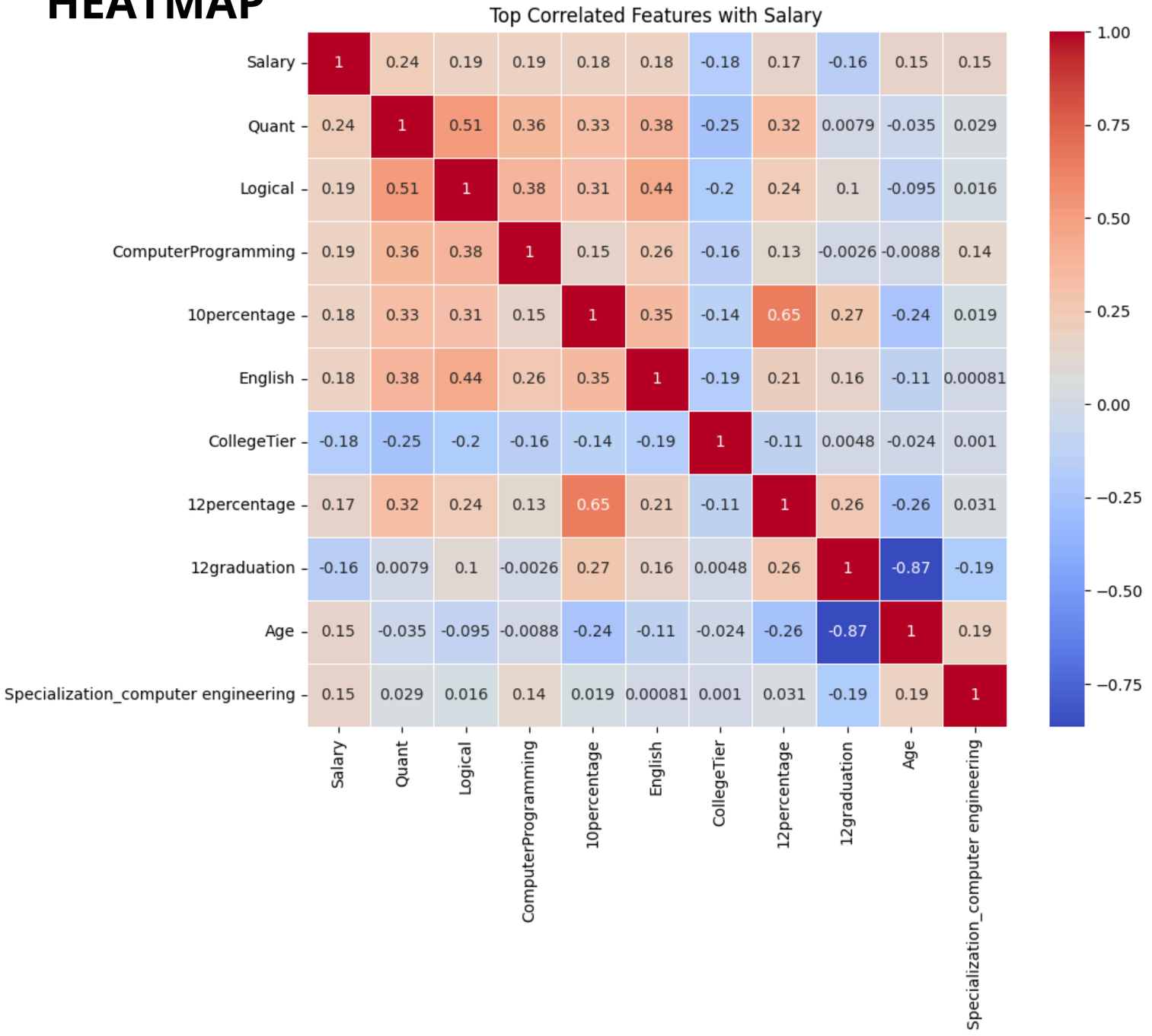
Computer networking has the highest average salary.
The top specializations are all in technology and engineering fields.
The difference in average salary across the top 10 specializations is relatively small.

Pair plot that includes the continuous columns of the data-frame :



his pair plot shows that while there is a strong positive correlation between a person's scores in Quant, Logical, and English, there is very little to no correlation between their college GPA or these test scores and their salary.

HEATMAP



This correlation heatmap displays the relationships between the various features and Salary. It shows that Quant and Logical abilities have the strongest positive correlation with Salary, while factors like Age and 12th-grade percentage show a weak negative correlation.

VIF (Variance Inflation Factor)

	variables	VIF
0	Age	554.19
1	12graduation	8738.07
2	GraduationYear	3029.53
3	CivilEngg	2631.21
4	ElectricalEngg	503.66
5	MechanicalEngg	383.82
6	Specialization_computer application	273.46
7	TelecomEngg	257.95
8	Degree_MCA	204.88
9	Specialization_information technology	172.87
10	Specialization_computer engineering	141.93
11	10percentage	137.06
12	12percentage	121.29
13	collegeGPA	119.38
14	ComputerScience	104.12
15	CollegeTier	82.78
16	ElectronicsAndSemicon	72.81
17	Logical	56.07
18	ComputerProgramming	45.99
19	English	34.81
20	Quant	34.41
21	Specialization_electrical engineering	22.44

MODEL BUILDING

LOGISTIC REGRESSION

Train-Test	Accuracy
60:40	70
70:30	71
75:25	71
80:20	**72**

K-NEAREST NEIGHBORS (KNN)

Train-Test	Accuracy
60:40	63
70:30	63
75:25	64
80:20	**65**

DECISION TREE

Train-Test	Accuracy
60:40	62
70:30	**63**
75:25	61
80:20	61

RANDOM FOREST

Train-Test	Accuracy
60:40	71
70:30	71
75:25	**72**
80:20	71

SUPPORT VECTOR MACHINE(SVM)

Train-Test	Accuracy
60:40	58
70:30	**60**
75:25	59
80:20	59

NAIVE BAYES

Train-Test	Accuracy
60:40	68
70:30	66
75:25	**69**
80:20	68

BAGGING (BOOTSTRAP AGGREGATING)

Train-Test	Accuracy
60:40	69
70:30	69
75:25	69
80:20	**70**

BOOSTING:

Gradient Boosting :

Train-Test	**n-estimators**	Accuracy
60-40	500	68
60-40	1000	69
60-40	2000	68
70-30	**500**	**71**
70-30	1000	69
70-30	2000	68
75-25	500	70
75-25	1000	69
80-20	500	70
80-20	1000	69

AdaBoost :

Train-Test	n-estimators	Accuracy
60-40	500	70
60-40	1000	70
60-40	2000	71
70-30	500	71
70-30	**1000**	**72**
70-30	2000	71
75-25	500	70
75-25	1000	71
80-20	500	71
80-20	1000	70

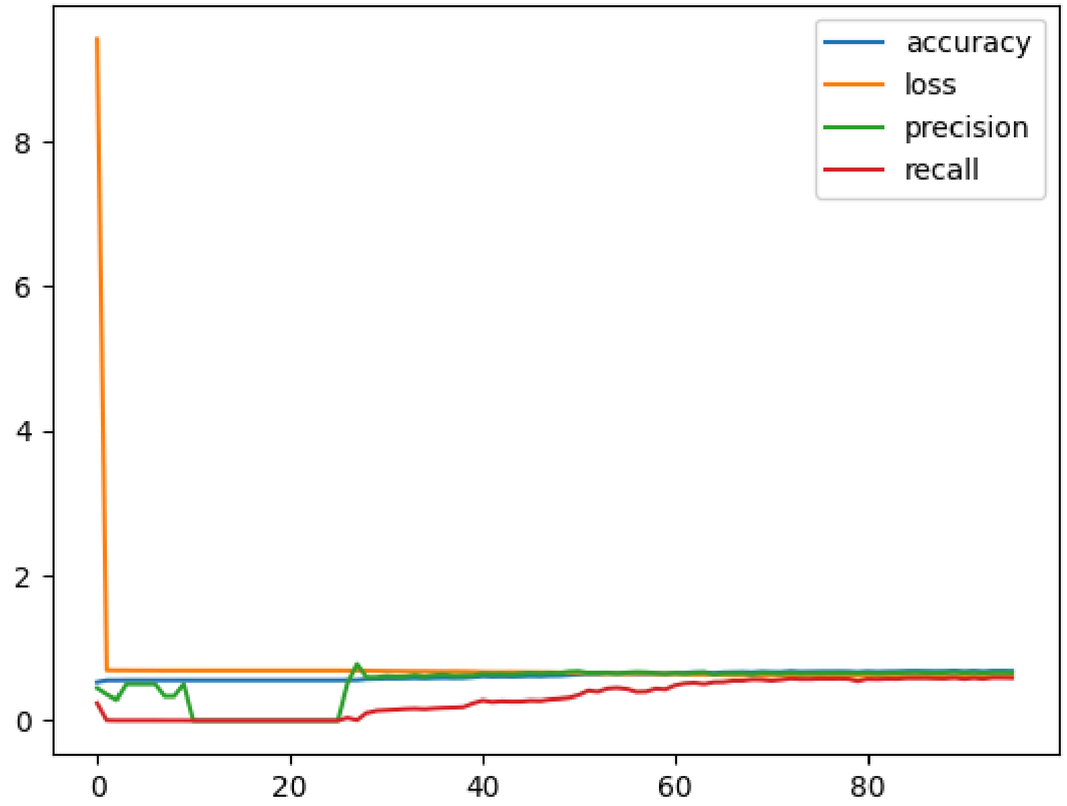
Extreme Gradient Boosting :

Train-Test	n-estimators	Accuracy
60-40	500	68
60-40	1000	70
60-40	2000	69
70-30	500	70
70-30	1000	70
70-30	2000	69
75-25	500	70
75-25	1000	70
80-20	**500**	**71**
80-20	1000	70

ANN:

Train-Test	Architecture	Epochs	Accuracy
60-40	10-7-5-1	50	60
60-40	10-7-5-1	100	52
60-40	10-7-5-1	125	63
60-40	10-7-5-1	477	69
70-30	7-7-5-1	50	69
70-30	7-7-5-1	100	65
70-30	7-7-5-1	250	70
70-30	7-7-5-1	500	69
75-25	10-7-5-1	50	61
75-25	10-7-5-1	125	68
75-25	10-7-5-1	500	65
80-20	10-7-5-1	50	52
80-20	**10-7-5-1**	**96**	**72**

ANN Plot:



Train-test	**80-20**
Optimizer	**Adam**
Architecture	**10-7-5-1**
Epochs	**96**

Comparison of the Models:

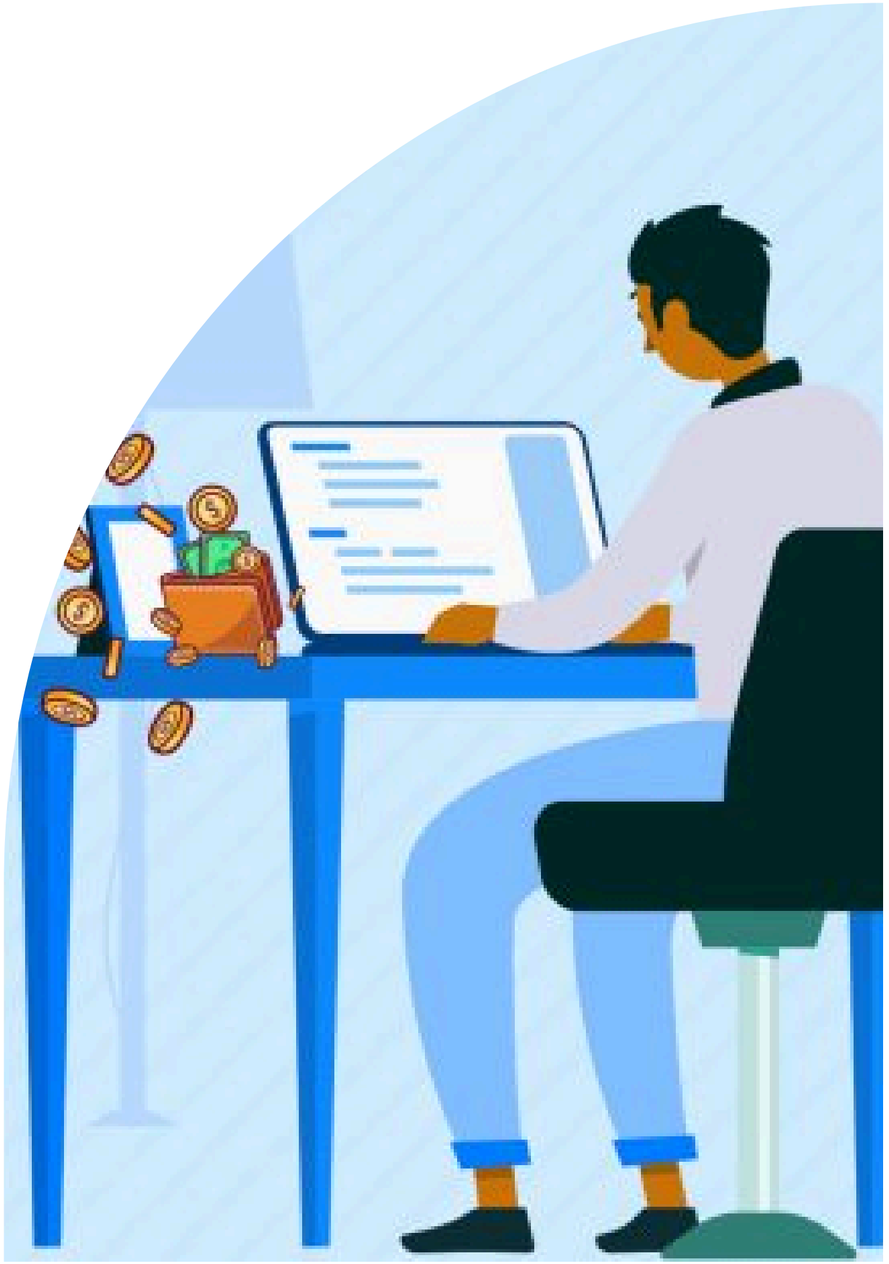
MODEL	TRAIN-TEST	ACCURACY
Logistic Regression	80-20	72.01
K Nearest Neighbor (kNN)	80-20	65
Decision Tree	70-30	63
Naïve Bayes	75-25	69
Support Vector Machine	70-30	60
Bagging	80-20	70
Gradient Boosting	70-30	71
Extreme Gradient Boosting	80-20	71
Adaptive Boosting	70-30	72
ANN	80-20	72.33
Random Forest	**75-25**	**72.51**

CONCLUSION

After applying various machine learning algorithms to the dataset, the Random Forest algorithm gave the best performance. Using a 75:25 train-test ratio, the model achieved an accuracy of 72.51%, which was the highest among the algorithms tested.

INSIGHTS

- Computer engineering and information technology specializations have the highest average salaries.
- Strong quantitative, logical, and English skills are positively correlated with higher salaries.
- The data shows variations in average salary across different specializations. Computer engineering graduates have the highest average salary among the top specializations, followed by information technology.



Thank You!

Colab Link:

https://colab.research.google.com/drive/1y4U7UQJVjMi2nVH_jxkoM-nSeRQoL6Ul?usp=sharing

Team members:
M. Shiva Charan
T. Kavya
N. Aravind
E. Sai Kumar

APPENDIX

Training and Testing :

```
X = df.drop(['Salary', 'High_Salary'], axis=1)
y = df['High_Salary']  # Target variable

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Initialize and train a simple classification model
model = LogisticRegression(max_iter=5000) # Increased max_iter for convergence
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy and F1 score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

ONE-HOT ENCODING:

```python
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# Get all object columns
object_cols = df.select_dtypes(include='object').columns.tolist()

# Apply OneHotEncoder to each object column
# Use handle_unknown='ignore' to handle potential new categories in test data
encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)

# Fit and transform the selected columns
encoded_data = encoder.fit_transform(df[object_cols])

# Create a DataFrame from the encoded data with appropriate column names
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(object_cols))

# Drop the original object columns
df = df.drop(object_cols, axis=1)

# Concatenate the original DataFrame with the new one-hot encoded DataFrame
df = pd.concat([df, encoded_df], axis=1)
```

Logistic Regression ：

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

# Step 1: Split features and target
X = df.drop(['Salary', 'High_Salary'], axis=1)
y = df['High_Salary']

# Step 2: Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Define parameter grid (try k values from 1 to 20)
param_grid = {'n_neighbors': list(range(1, 21))}

# Step 4: Initialize GridSearchCV
grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, scoring='accuracy')

# Step 5: Fit to the scaled data
grid.fit(X_scaled, y)

# Step 6: Print best results
print("Best n_neighbors:", grid.best_params_['n_neighbors'])
print("Best cross-validation accuracy:", grid.best_score_)
```

K-Nearest Neighbors :

```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

# Define features (X) and target (y)
X = df.drop(['Salary', 'High_Salary'], axis=1)  # Features
y = df['High_Salary']  # Target variable

# Initialize KNN classifier (you can still tune n_neighbors)
knn_model = KNeighborsClassifier(n_neighbors=5)

# Perform 5-fold cross-validation
cv_scores = cross_val_score(knn_model, X, y, cv=5, scoring='accuracy')

print("Cross-validation Accuracy Scores:", cv_scores)
print("Mean Cross-validation Accuracy:", cv_scores.mean())
```

Decision Tree :

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import train_test_split

# Assuming 'df' is already loaded and preprocessed with 'High_Salary' column

# Define features (X) and target (y)
X = df.drop(['Salary', 'High_Salary'], axis=1)  # Features
y = df['High_Salary']  # Target variable

# Split data into training and testing sets (using the same split as before for consistency)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Initialize and train a Decision Tree classifier with specified parameters
dt_model_tuned = DecisionTreeClassifier(max_depth=4, min_samples_leaf=1, min_samples_split=2, random_state=42)
dt_model_tuned.fit(X_train, y_train)

# Make predictions on the test set
y_pred_dt_tuned = dt_model_tuned.predict(X_test)

# Calculate accuracy and F1 score
accuracy_dt_tuned = accuracy_score(y_test, y_pred_dt_tuned)
f1_dt_tuned = f1_score(y_test, y_pred_dt_tuned)

print(f"Tuned Decision Tree Accuracy: {accuracy_dt_tuned:.2f}")
print(f"Tuned Decision Tree F1 Score: {f1_dt_tuned:.2f}")
```

Naïve Bayes :

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score

# Assuming X and y are already defined from previous steps

# Initialize Gaussian Naive Bayes classifier
gnb_model = GaussianNB()

# Perform 5-fold cross-validation without scaling
cv_scores_gnb = cross_val_score(gnb_model, X_train, y_train, cv=5, scoring='accuracy')

print("Cross-validation Accuracy Scores (Gaussian Naive Bayes without scaling):", cv_scores_gnb)
print("Mean Cross-validation Accuracy (Gaussian Naive Bayes without scaling):", cv_scores_gnb.mean())
```

Support Vector Machine:

```python
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# 1. Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 2. Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 3. Define parameter grid for tuning
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': [0.01, 0.1, 1],
    'kernel': ['rbf', 'linear']
}

# 4. Set up GridSearchCV
grid = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy')
grid.fit(X_train_scaled, y_train)

# 5. Best parameters and model
print("Best Parameters:", grid.best_params_)

# 6. Evaluate on test set
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy with Tuned SVM:", accuracy)
```

Random Forest:

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

# Assuming X and y are already defined from previous steps

# Initialize Random Forest classifier
rf_model = RandomForestClassifier(random_state=42)

# Perform 5-fold cross-validation
cv_scores = cross_val_score(rf_model, X, y, cv=5, scoring='accuracy')

print("Cross-validation Accuracy Scores:", cv_scores)
print("Mean Cross-validation Accuracy:", cv_scores.mean())
```

Booting :

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import cross_val_score

# Initialize Gradient Boosting classifier
# You might want to tune hyperparameters later
gb_model = GradientBoostingClassifier(random_state=42)

# Perform 5-fold cross-validation on scaled data
cv_scores_gb = cross_val_score(gb_model, X_train, y_train, cv=5, scoring='accuracy')

print("Cross-validation Accuracy Scores (Gradient Boosting):", cv_scores_gb)
print("Mean Cross-validation Accuracy (Gradient Boosting):", cv_scores_gb.mean())
```

XGBoost:

```
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score

# Initialize XGBoost classifier
# You might want to tune hyperparameters later for better performance
xgb_model = XGBClassifier(eval_metric='logloss', random_state=42)

# Perform 5-fold cross-validation
cv_scores_xgb = cross_val_score(xgb_model, X_train, y_train, cv=5, scoring='accuracy')

print("Cross-validation Accuracy Scores (XGBoost):", cv_scores_xgb)
print("Mean Cross-validation Accuracy (XGBoost):", cv_scores_xgb.mean())
```

AdaBoost:

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score

# Assuming X and y are already defined from previous steps

# Initialize AdaBoost classifier
adaboost_model = AdaBoostClassifier(random_state=42)

# Perform 5-fold cross-validation
cv_scores_adaboost = cross_val_score(adaboost_model, X_train, y_train, cv=5, scoring='accuracy')

print("Cross-validation Accuracy Scores (AdaBoost):", cv_scores_adaboost)
print("Mean Cross-validation Accuracy (AdaBoost):", cv_scores_adaboost.mean())
```

ANN:

```
import tensorflow as tf
```

```
# Split data into training and testing sets using df_reduced_vif
X_train, X_test, y_train, y_test = train_test_split(df_reduced_vif.drop(['Salary', 'High_Salary'], axis=1), df_reduced_vif['High_Salary'], test_size=0.3, random_state=46)
```

```
tf.random.set_seed(42)

# STEP1: Creating the model

model= tf.keras.Sequential([
                            tf.keras.layers.Dense(7, activation='relu'),
                            tf.keras.layers.Dense(7, activation='relu'),
                            tf.keras.layers.Dense(5, activation='relu'),
                            tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP2: Compiling the model

model.compile(loss= tf.keras.losses.binary_crossentropy,
              optimizer= tf.keras.optimizers.Adam(learning_rate=0.001),
              metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                        tf.keras.metrics.Precision(name='precision'),
                        tf.keras.metrics.Recall(name='recall')
              ]
              )

# Split data into training and testing sets using df_reduced_vif
X_train, X_test, y_train, y_test = train_test_split(df_reduced_vif.drop(['Salary', 'High_Salary'], axis=1), df_reduced_vif['High_Salary'], test_size=0.3, random_state=46)


# STEP1: Fit the model

history= model.fit(X_train,y_train, epochs= 250)
```

```
Epoch 1/250
63/63 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5452 - loss: 54.1259 - precision: 0.4648 - recall: 0.2625
Epoch 2/250
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5198 - loss: 1.7422 - precision: 0.4771 - recall: 0.5819
```

```
model.summary();
```

```
Model: "sequential_19"
```

Layer (type)	Output Shape	Param #
dense_96 (Dense)	(None, 7)	616
dense_97 (Dense)	(None, 7)	56
dense_98 (Dense)	(None, 5)	40
dense_99 (Dense)	(None, 1)	6

```
Total params: 2,156 (8.43 KB)
Trainable params: 718 (2.80 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,438 (5.62 KB)
```

```
model.evaluate(X_test, y_test)
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7015 - loss: 0.5591 - precision: 0.6479 - recall: 0.7498
[0.5714136958122253,
 0.6947368383407593,
 0.6553288102149963,
 0.7261306643486023]
```