PREDICTIVE MODEL OF JOB PLACEMENT BASED ON SEVERAL FACTORS

CONTENT

INTRODUCTION

PROBLEM STATEMENT

Almost every students comes to a very vital juncture post high school that is getting admits from their dream university. Similar was the case for John. He was most concerned about the job placement scenario once he graduates.

The problem he faces is multifaceted: How can he accurately assess and compare the employment outcomes of different institutions? And how can these statistics reflect the true value of the education and opportunities that await him? This analysis is crucial, for it will shape his educational journey and, ultimately, his career trajectory.

RESEARCH QUESTIONS

How do different factors such as gender, streams, college name, gpa, degree, research paper, co-ops, no. of technical skills etc. affect the overall employment outcome?

IMPLICATIONS

The project's implications are profound, as it empowers students like John to make informed decisions about their education and future careers, potentially shaping their lifelong success and fulfillment.

PLAN TO ACHIEVE FROM PROJECT

The project aims to provide John with a comprehensive framework to evaluate and compare the employment outcomes of various universities accurately. Ultimately, it seeks to empower him in making informed decisions that align with his educational and career goals.

PRIMARY GOAL OF THE PROJECT

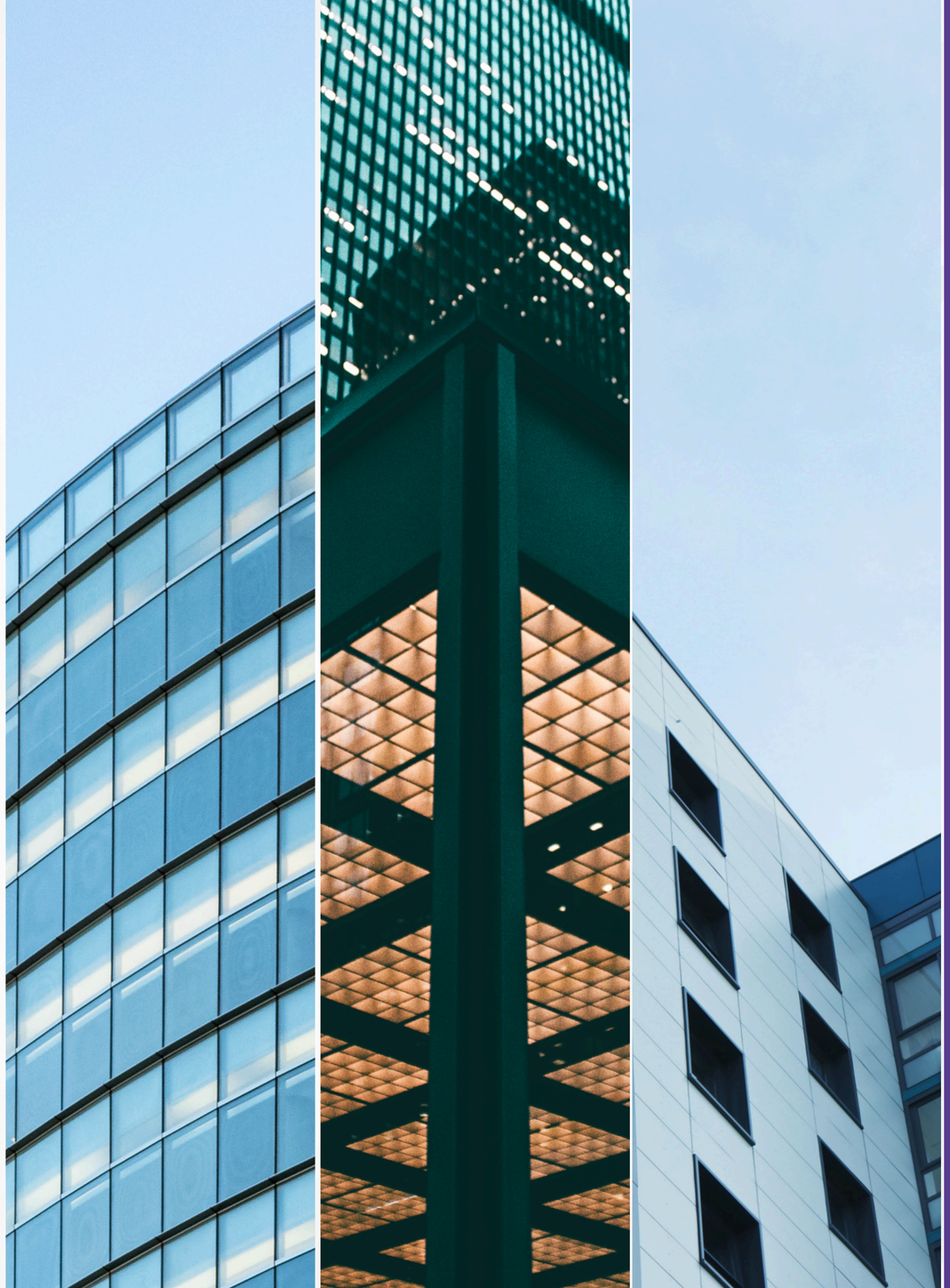
The primary goal is to develop a robust methodology for evaluating and comparing job placement outcomes among universities. This will enable informed decision-making for students like John, enhancing their educational and career prospects.

DESCRIBING DATASET

OUR DATASET

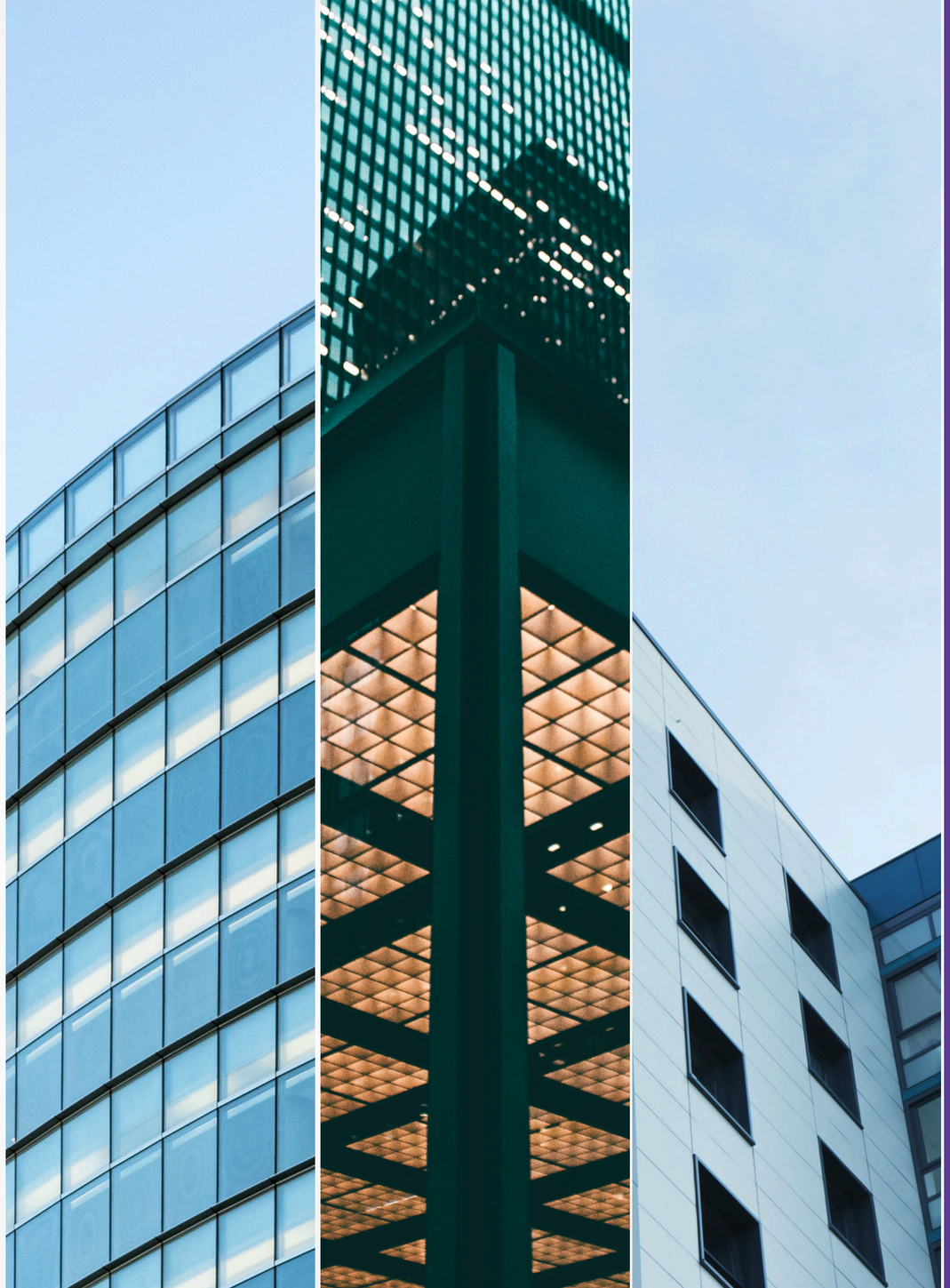
This data is derived from Kaggle. It contains 11 columns and 500+ rows.

job_placement.csv contains details about Bachelor's degree graduates from diverse American universities and their employment status. The information includes attributes such as gender, age, academic field, university affiliation, placement status, placement-associated salary, GPA, years of professional experience, research papers, co-ops and no. of technical skills.



VARIABLES OF THE DATASET

- ID
- Name
- Gender
- Age
- Degree
- Stream
- College Name
- Placement Status
- Salary
- GPA
- No. of research papers
- No. of Co-ops
- No of Technical skills



DESCRIPTIVE DATASET

NUMBER OF

ROWS & COLUMNS

```
# Shape of the data set
df.shape
```

```
(700, 14)
```

```
# No. of Rows
df.shape[0]
```

```
700
```

```
# No. of Columns
df.shape[1]
```

```
14
```

```
# No. of entries in the Data Set
df.size
```

```
9800
```

PERCENTAGE OF
MISSING VALUES COLUMNS

```
def missing (df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_values = pd.concat([missing_number, missing_percent], axis=1, keys=['Missing_Number', 'Missing_Percent'])
    return missing_values

missing(df)
```

	Missing_Number	Missing_Percent
gpa	5	0.007143
age	3	0.004286
years_of_experience	3	0.004286
Research Papers	3	0.004286
college_name	2	0.002857
Co-Ops	2	0.002857
Technical Skills	2	0.002857
id	0	0.000000
name	0	0.000000
gender	0	0.000000
degree	0	0.000000
stream	0	0.000000
placement_status	0	0.000000
salary	0	0.000000

MEAN

```python
# Calculate the mean
mean = df.mean()
print("Mean:")
print(mean)
```

```
Mean:
id                          350.500000
age                          24.410330
salary                    54772.857143
gpa                           3.802201
years_of_experience           2.177905
Research Papers               2.011478
Co-Ops                        1.992837
Technical Skills              2.924069
dtype: float64
```

MEDIAN

```python
# Calculate the median
median = df.median()
print("\nMedian:")
print(median)
```

```
Median:
id                          350.50
age                          24.00
salary                    61000.00
gpa                           3.81
years_of_experience           2.00
Research Papers               2.00
Co-Ops                        2.00
Technical Skills              3.00
dtype: float64
```

MODE

```python
# Calculate the mode
mode = df.mode().iloc[0]
print("\nMode:")
print(mode)
```

```
Mode:
id                                                1
name                                      Aiden Davis
gender                                         Female
age                                              24.0
degree                                      Bachelor's
stream                                  Computer Science
college_name           University of California--Berkeley
placement_status                                Placed
salary                                             0.0
gpa                                               3.77
years_of_experience                               3.0
Research Papers                                   3.0
Co-Ops                                            1.0
Technical Skills                                  2.0
Name: 0, dtype: object
```

VARIANCE

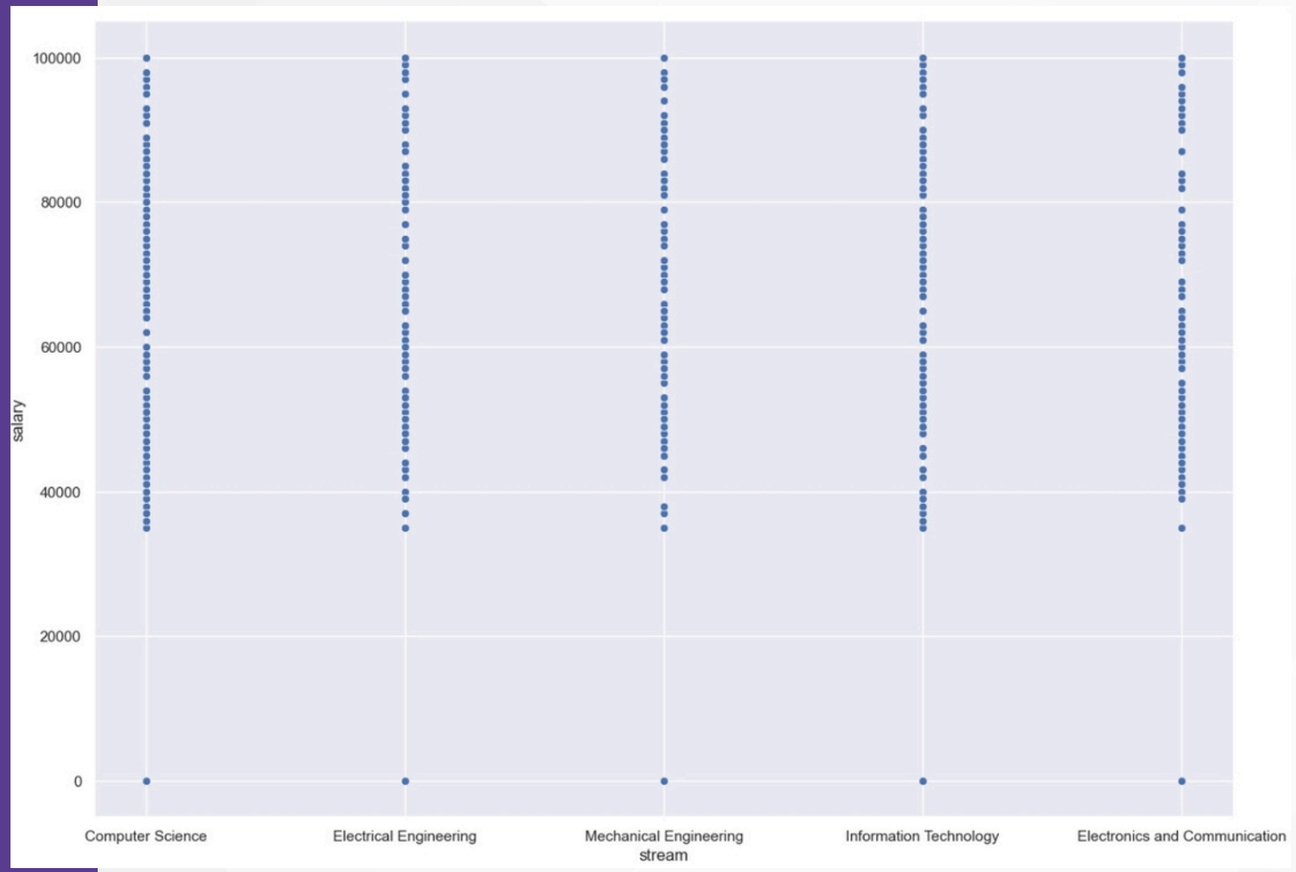
DATA VISUALIZATION

SCATTER PLOT

The scatter plot graph illustrates a positive correlation between GPA and salary, suggesting that higher academic achievement may lead to increased earning potential.

DATA VISUALIZATION

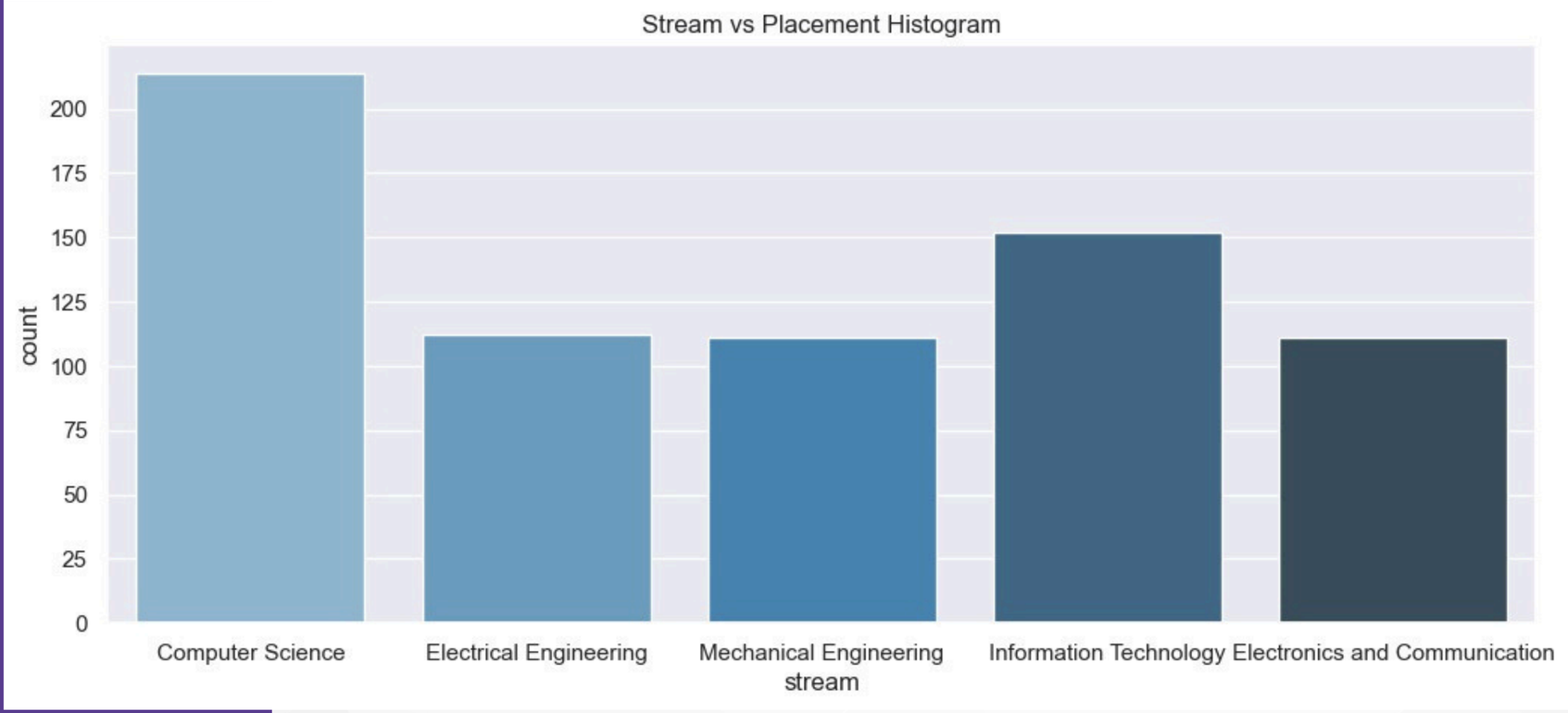
SCATTER PLOT



The graph depicts salary distributions across engineering fields, showing wide individual variances but similar averages, indicating diverse earning potential within each discipline.

DATA VISUALIZATION

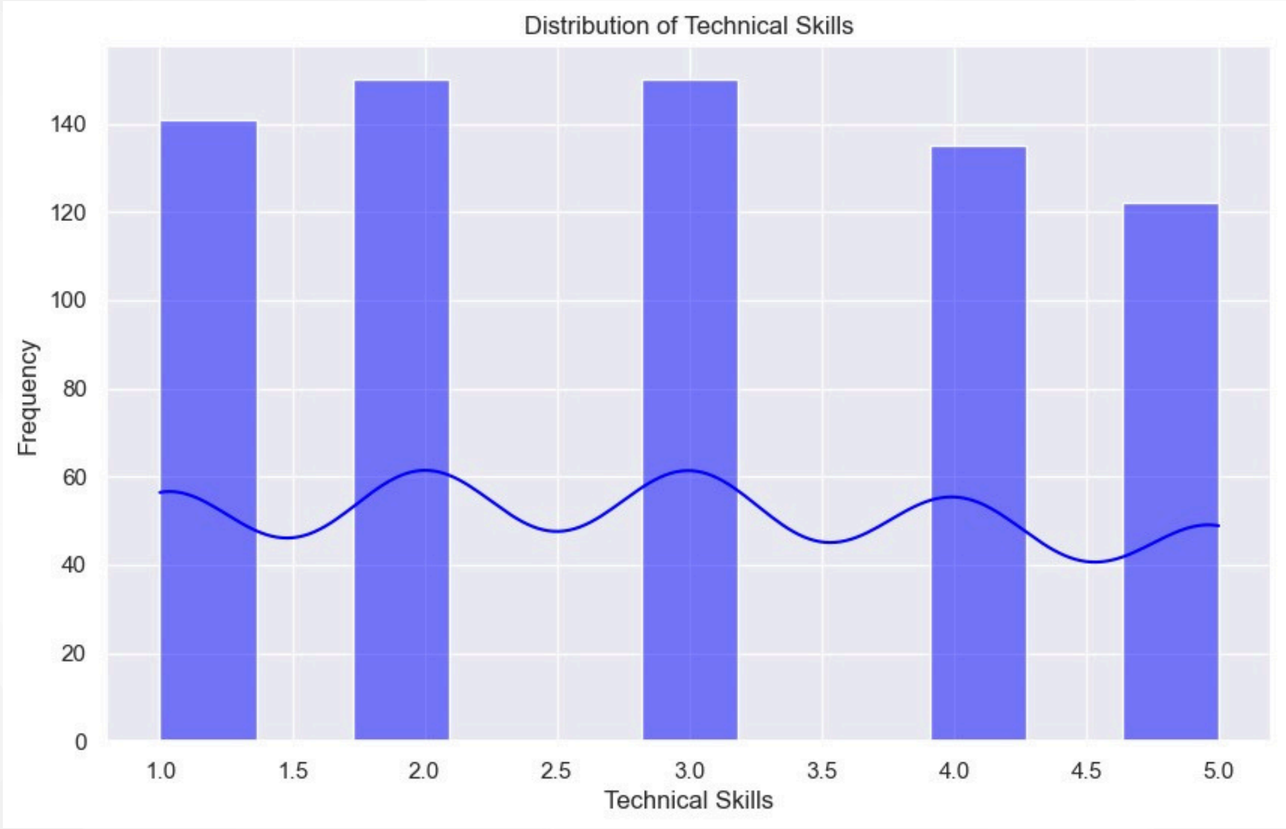
HISTOGRAM



The histogram illustrates the student distribution across engineering streams, with Computer Science being the most popular, followed by Information Technology and other fields.

DATA VISUALIZATION

FREQUENCY CHARTS



The graph visualizes the prevalence of different technical skill level. It combines a bar chart showing the actual counts or proportions of people in each skill level category with a line chart that may indicate a trend or cumulative distribution across skill levels.

CORRELATION

The heatmap reveals intricate correlations among variables like GPA and years of experience with salary, offering insights into factors influencing income. Stronger correlations suggest significant relationships that could inform career and education strategies.

DATA PREPROCESSING

VARIABLE TREATMENT

Dropping Categorical Variables with Zero variance

```
categorical_variables
```

```
['name',
 'gender',
 'degree',
 'stream',
 'college_name',
 'placement_status',
 'years_of_experience',
 'Technical Skills']
```

```
# Drop categorical variables with 0 variance
zero_variance_categorical_variables = []
for i in categorical_variables:
    if i in df.columns and len(df[i].value_counts().index) == 1:
        zero_variance_categorical_variables.append(i)

df = df.drop(zero_variance_categorical_variables, axis=1)
```

The Python code provided is a data preprocessing step used in machine learning to remove features from a dataset that have no variance, meaning they have the same value for every observation. Such features do not contribute any meaningful information for modeling. The code iterates over a list of categorical variables, checks for zero variance, and drops those columns from the DataFrame. It ensures that the remaining features are potentially useful for predictive modeling by having variability in the data.

DROPPING CATEGORICAL VARIABLES

Dropping Categorical Variables with many levels

```
#Drop high cardinality variables
high_cardinality_categorical_variables = []
categorical_variables = list(set(categorical_variables) - set(zero_variance_categorical_variables))
for i in categorical_variables:
    if len(df[i].value_counts().index)>200:
        high_cardinality_categorical_variables.append(i)
df = df.drop(high_cardinality_categorical_variables, axis = 1)
```

The code is designed to clean a pandas DataFrame by removing categorical variables that exhibit zero variance, meaning they have only one unique value across all observations. This is achieved by iterating over a list of categorical variables, identifying those with a single unique value, and then dropping these columns from the DataFrame. The process enhances the quality of the dataset by ensuring that only variables with meaningful information are retained for analysis.

IMPUTING MISSING VALUES

```
def missing (df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_values = pd.concat([missing_number, missing_percent], axis=1, keys=['Missing_Number', 'Missing_Percent'])
    return missing_values

missing(df)
```

	Missing_Number	Missing_Percent
college_name	2	0.002857
id	0	0.000000
name	0	0.000000
gender	0	0.000000
age	0	0.000000
degree	0	0.000000
stream	0	0.000000
placement_status	0	0.000000
salary	0	0.000000
gpa	0	0.000000
years_of_experience	0	0.000000
Research Papers	0	0.000000
Co-Ops	0	0.000000
Technical Skills	0	0.000000

TRANSFORMING VARIABLES

Scalling

```
from sklearn.preprocessing import StandardScaler
#Drop zero variance variables from numerical variables
numerical_variables = list(set(numerical_variables) - set(zero_variance_numerical_variables))

# Convert to numpy array
array = df[numerical_variables].values

# Create standarization instance
data_scaler = StandardScaler().fit(array)

# Standardize the numerical variables
data_rescaled = pd.DataFrame(data_scaler.transform(array), columns = numerical_variables)

df = df.drop(numerical_variables, axis = 1)

df = pd.concat([df, data_rescaled], axis=1)
```

TRANSFORMING VARIABLES

```
#Create dummy variables
#Drop variables with high variance from categorical variables
categorical_variables = list(set(categorical_variables) - set(high_cardinality_categorical_variables))

# Create dummy variables
dummy_variables = pd.get_dummies(df[categorical_variables], drop_first=True)

# Drop categorical variables from df
df = df.drop(categorical_variables, axis = 1)

# Add dummy variables to df
df = pd.concat([df, dummy_variables], axis = 1)
```

The code performs data preprocessing for machine learning. It removes numerical variables with zero variance, standardizes remaining numerical variables using StandardScaler, and then reintegrates them into the DataFrame. It also handles categorical variables by removing those with high cardinality, creating dummy variables, and merging them back. Finally, it converts the ‘salary’ column to integer type, preparing the dataset for model training with standardized numerical inputs and encoded categorical features.

FITTING MODELS

LINEAR REGRESSION

```
:
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Initialize the Linear Regression model
lreg = LinearRegression()

# Fit the model to the training data
lreg.fit(x_train, y_train)

# Predict on the training and testing sets
y_train_pred = lreg.predict(x_train)
y_test_pred = lreg.predict(x_test)

# Evaluate the model's performance on the testing set using R^2 score
print("Model Predictive Accuracy: {:.2%}".format(r2_score(y_test, y_test_pred)))
```

```
Model Predictive Accuracy: 69.14%
```

DECISION TREE REGRESSION

```
: from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
import pandas as pd

# Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.3, random_state=0)

# Initialize the Decision Tree Regressor with a random state for reproducibility
dt_regressor = DecisionTreeRegressor(random_state=0)

# Fit the model to the training data
dt_regressor.fit(X_train, Y_train)

# Predict on both training and test sets
Y_train_pred = dt_regressor.predict(X_train)
Y_test_pred = dt_regressor.predict(X_test)

# Evaluate the model's predictive accuracy using the R2 score
performance_score = dt_regressor.score(X_test, Y_test)
print(f"Model Predictive Accuracy: {performance_score:.2%}")
```

```
Model Predictive Accuracy: 35.31%
```

RANDOM FOREST REGRESSION

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
import pandas as pd

# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initializing and fitting the RandomForestRegressor with the corrected criterion parameter
Rf = RandomForestRegressor(n_estimators=15,
                           criterion='squared_error',  # Updated criterion parameter
                           random_state=20,
                           n_jobs=-1)  # Use all available processors for training

# Fitting the model to the training data
Rf.fit(x_train, y_train)

# Making predictions on both the training and testing datasets
Rf_train_pred = Rf.predict(x_train)
Rf_test_pred = Rf.predict(x_test)

# Computing and printing the R2 score to evaluate the model's performance
print("Model Predictive Accuracy: {:.2%}".format(r2_score(y_test, Rf_test_pred)))
```

```
Model Predictive Accuracy: 64.87%
```

GRADIENT BOOSTING REGRESSION

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import pandas as pd

features=list(X.columns[rfe.support_])

X = df[features]
y = df['salary']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initializing the Gradient Boosting Regressor
gbm_regressor = GradientBoostingRegressor(random_state=42)

# Fitting the model to the training data
gbm_regressor.fit(X_train, y_train)

# Making predictions on the test dataset
y_test_pred = gbm_regressor.predict(X_test)

# Evaluating the model using R2 score
r2_test_score = r2_score(y_test, y_test_pred)
print(f"Model Predictive Accuracy: {r2_test_score:.2%}")
```

```
Model Predictive Accuracy: 66.26%
```

XG BOOST REGRESSION

```
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

# Assuming 'df' is your DataFrame and 'features' contains the feature columns
X = df[features]
y = df['salary']  # Ensure 'salary' is the target variable

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Initializing the XGBoost Regressor
xgb_regressor = xgb.XGBRegressor(objective='reg:squarederror', colsample_bytree=0.9, learning_rate=0.9,
                                 max_depth=10, alpha=5, n_estimators=10, random_state=42)

# Fitting the model to the training data
xgb_regressor.fit(X_train, y_train)

# Making predictions on the test dataset
y_test_pred = xgb_regressor.predict(X_test)

# Evaluating the model using R2 score
r2_test_score = r2_score(y_test, y_test_pred)
print(f"Model Predictive Accuracy: {r2_test_score:.2%}")
```

```
Model Predictive Accuracy: 70.55%
```

BEST MODEL

Here we, applies various predictive models. The models are as follows:

- Linear Regression
- Random forest regression
- Decision tree regression
- Gradient Boosting Regression
- XG Boost Regression

As we put these different models to testing we got the following predictive accuracies from each model:

- Linear Regression **69.14%**
- Random forest regression **64.87%**
- Decision tree regression **35.31%**
- Gradient Boosting Regression **66.26%**
- XG Boost Regression **70.55%**

And as we can see that XG Boost Regression model gives the highest accuracy. **Therefore, we conclude that for this particular data set the Best Model is XG Boost Regression Model**

THANK YOU

- Arjun Malgwa
- Anmol Fernandez
- Hitankshi Jain
- Ishita Poorey
- Vaishali Patidar