SALARY PREDICTION USING MACHINE LEARNING

PRESENTATION BY :

AKHILESH MAURYA

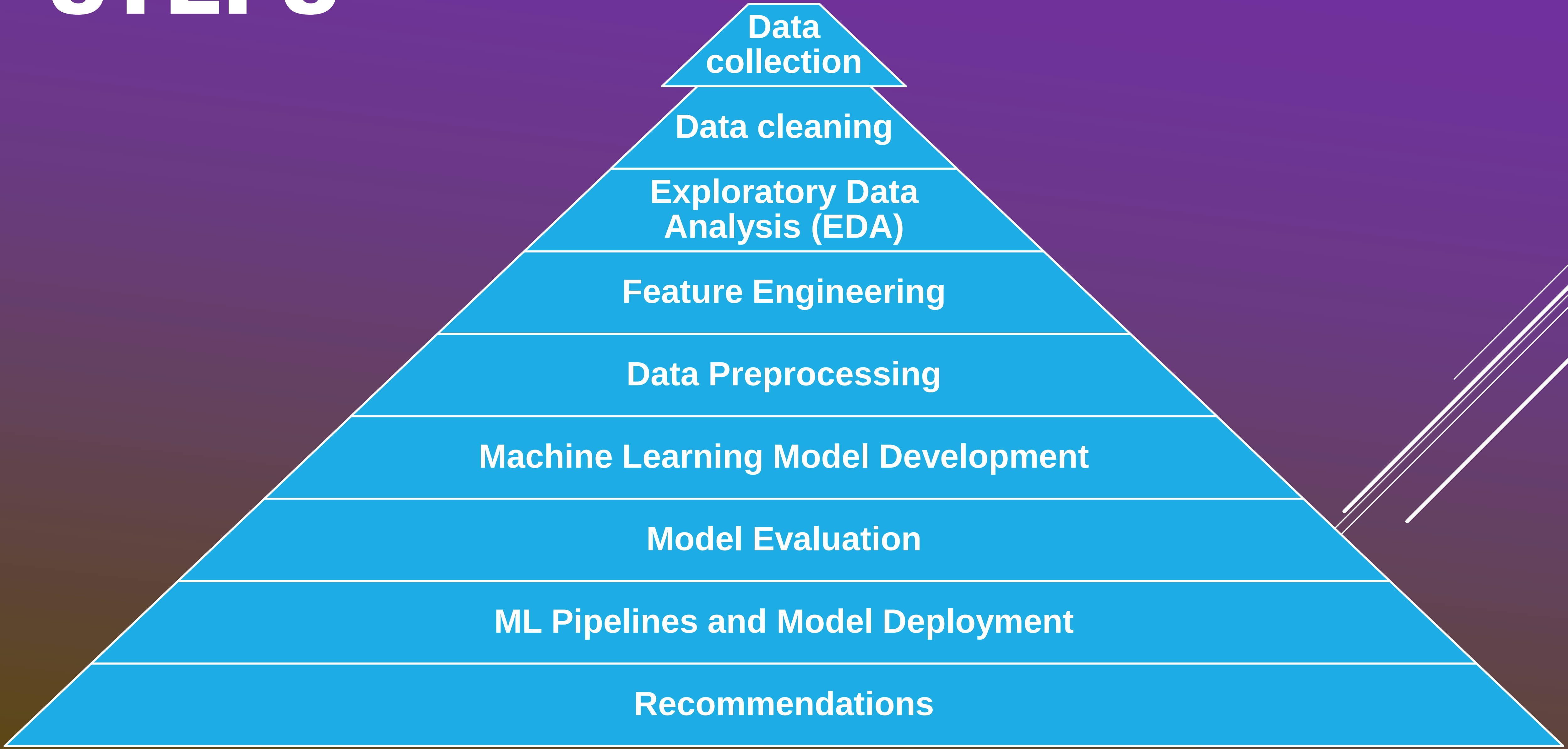
PROBLEM STATEMENTS

Salaries in the field of data professions vary widely based on factors such as experience, job role, and performance. Accurately predicting salaries for data professionals is essential for both job seekers and employers.

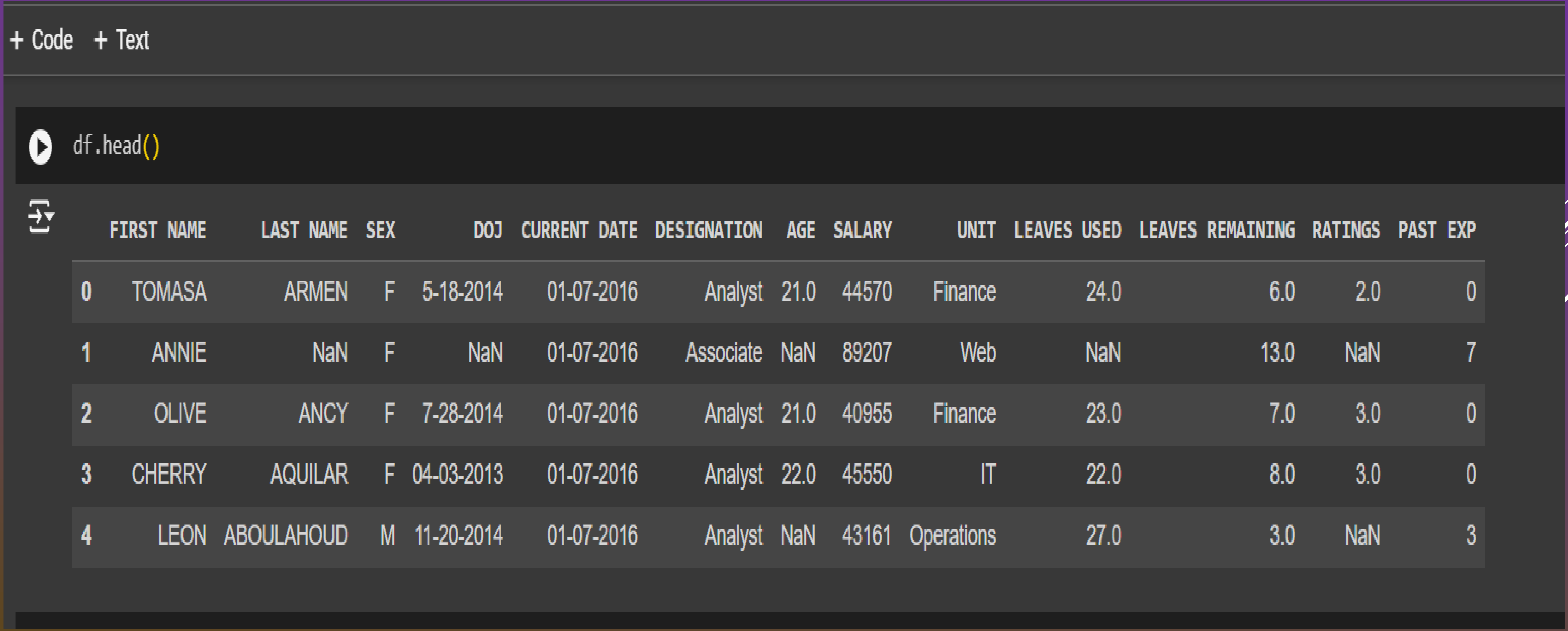
DATASET OVERVIEW

- □ `FIRST NAME`: First name
- □ `LAST NAME`: Last name
- □ `SEX`: Gender
- □ `DOJ`: Date of joining the company
- □ `CURRENT DATE`: Current date of data
- □ `DESIGNATION`: Job role/designation
- □ `AGE`: Age
- □ `SALARY`: Target variable, the salary of the data professional
- □ `UNIT`: Business unit or department
- □ `LEAVES USED`: Number of leaves used
- □ `LEAVES REMAINING`: Number of leaves remaining
- □ `RATINGS`: Ratings or performance ratings
- □ `PAST EXP`: Past work experience

STEPS


Data collection
Data cleaning
Exploratory Data Analysis (EDA)
Feature Engineering
Data Preprocessing
Machine Learning Model Development
Model Evaluation
ML Pipelines and Model Deployment
Recommendations

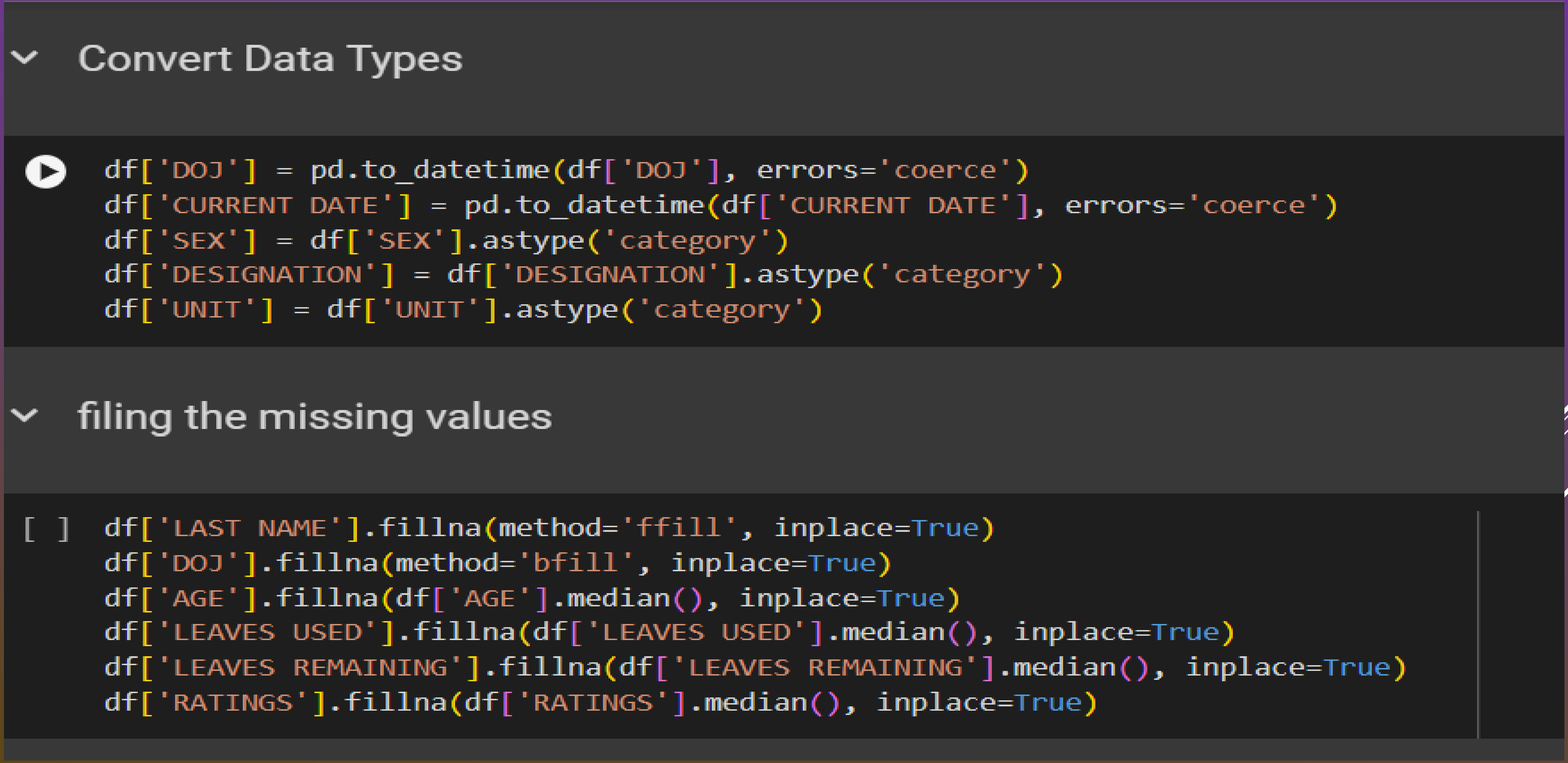
1. DATA COLLECTION



```
df.head()
```

```
   FIRST NAME   LAST NAME  SEX         DOJ  CURRENT DATE  DESIGNATION   AGE  SALARY        UNIT  LEAVES USED  LEAVES REMAINING  RATINGS  PAST EXP
0      TOMASA       ARMEN    F   5-18-2014    01-07-2016      Analyst  21.0   44570     Finance         24.0               6.0      2.0         0
1       ANNIE         NaN    F         NaN    01-07-2016    Associate   NaN   89207         Web          NaN              13.0      NaN         7
2       OLIVE        ANCY    F   7-28-2014    01-07-2016      Analyst  21.0   40955     Finance         23.0               7.0      3.0         0
3      CHERRY     AQUILAR    F  04-03-2013    01-07-2016      Analyst  22.0   45550          IT         22.0               8.0      3.0         0
4        LEON  ABOULAHOUD    M  11-20-2014    01-07-2016      Analyst   NaN   43161  Operations         27.0               3.0      NaN         3
```

2. DATA CLEANING



Convert Data Types

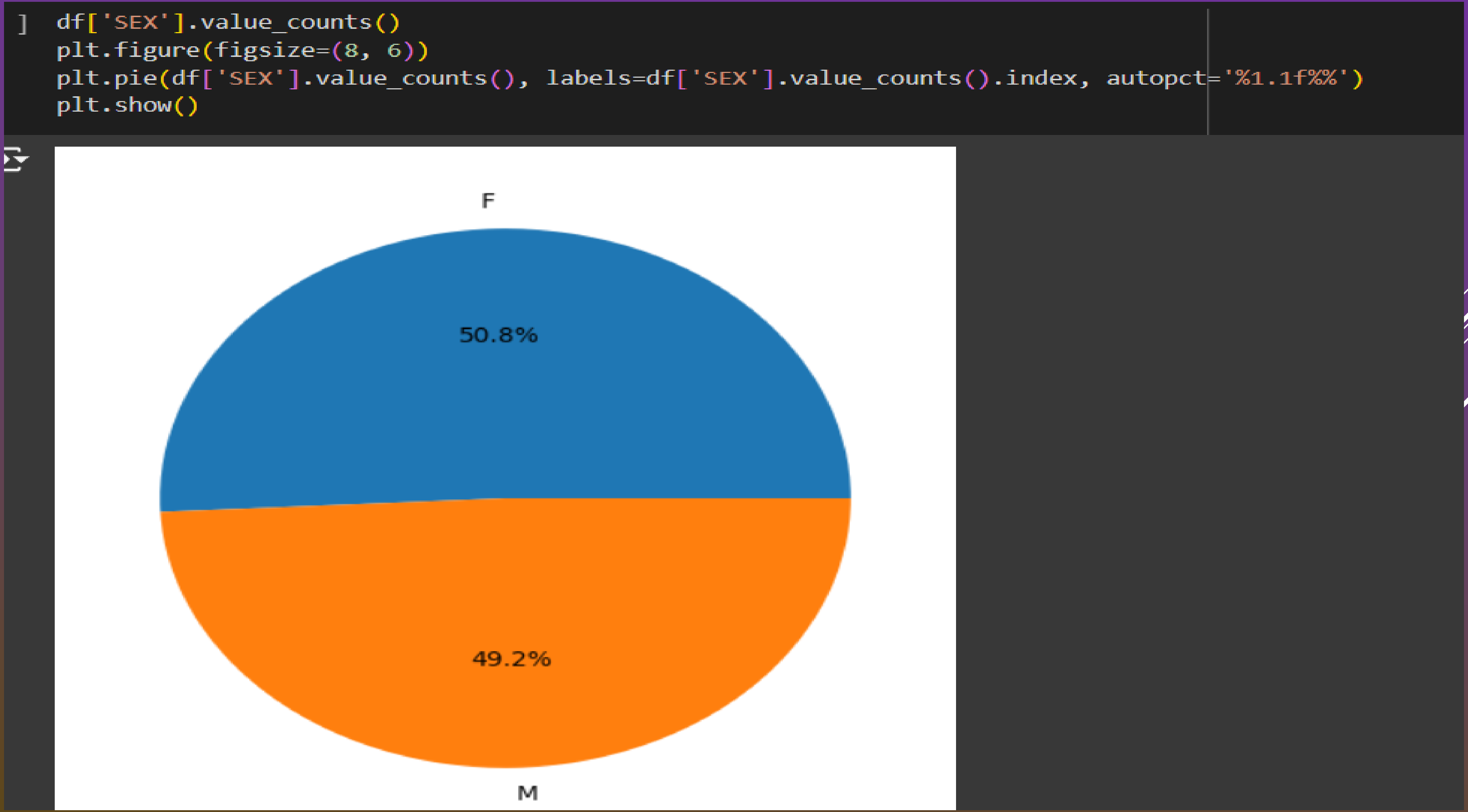
```
df['DOJ'] = pd.to_datetime(df['DOJ'], errors='coerce')
df['CURRENT DATE'] = pd.to_datetime(df['CURRENT DATE'], errors='coerce')
df['SEX'] = df['SEX'].astype('category')
df['DESIGNATION'] = df['DESIGNATION'].astype('category')
df['UNIT'] = df['UNIT'].astype('category')
```

filing the missing values

```
[ ]  df['LAST NAME'].fillna(method='ffill', inplace=True)
     df['DOJ'].fillna(method='bfill', inplace=True)
     df['AGE'].fillna(df['AGE'].median(), inplace=True)
     df['LEAVES USED'].fillna(df['LEAVES USED'].median(), inplace=True)
     df['LEAVES REMAINING'].fillna(df['LEAVES REMAINING'].median(), inplace=True)
     df['RATINGS'].fillna(df['RATINGS'].median(), inplace=True)
```

3. EXPLORATORY DATA ANALYSIS

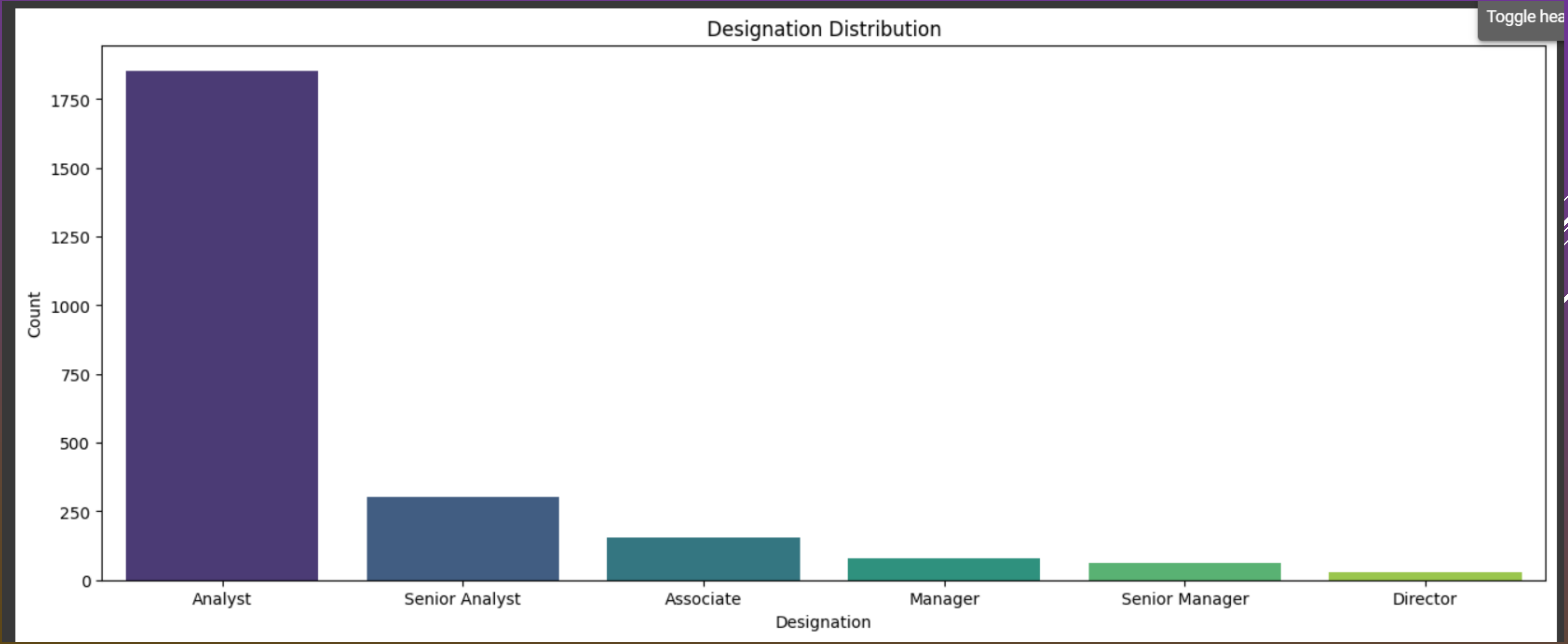
- Gender



```
df['SEX'].value_counts()
plt.figure(figsize=(8, 6))
plt.pie(df['SEX'].value_counts(), labels=df['SEX'].value_counts().index, autopct='%1.1f%%')
plt.show()
```

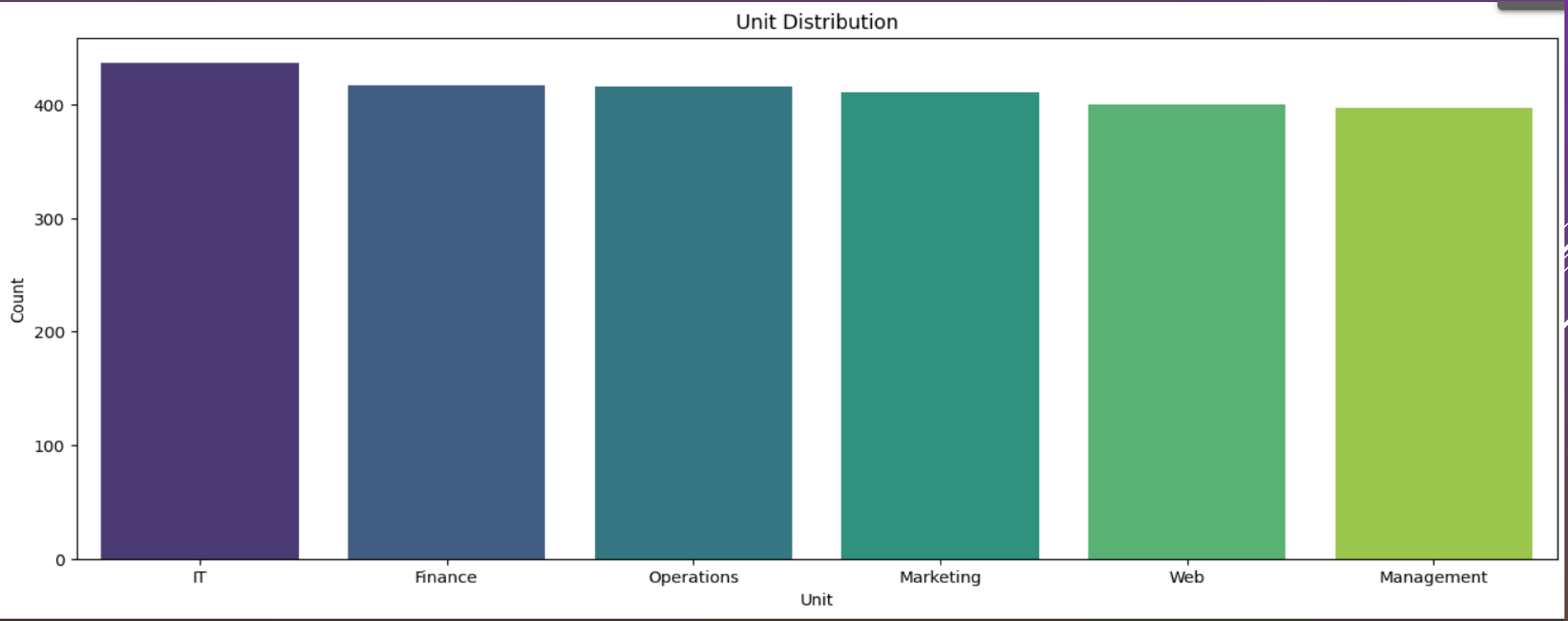
EXPLORATORY DATA ANALYSIS

- **Designation**



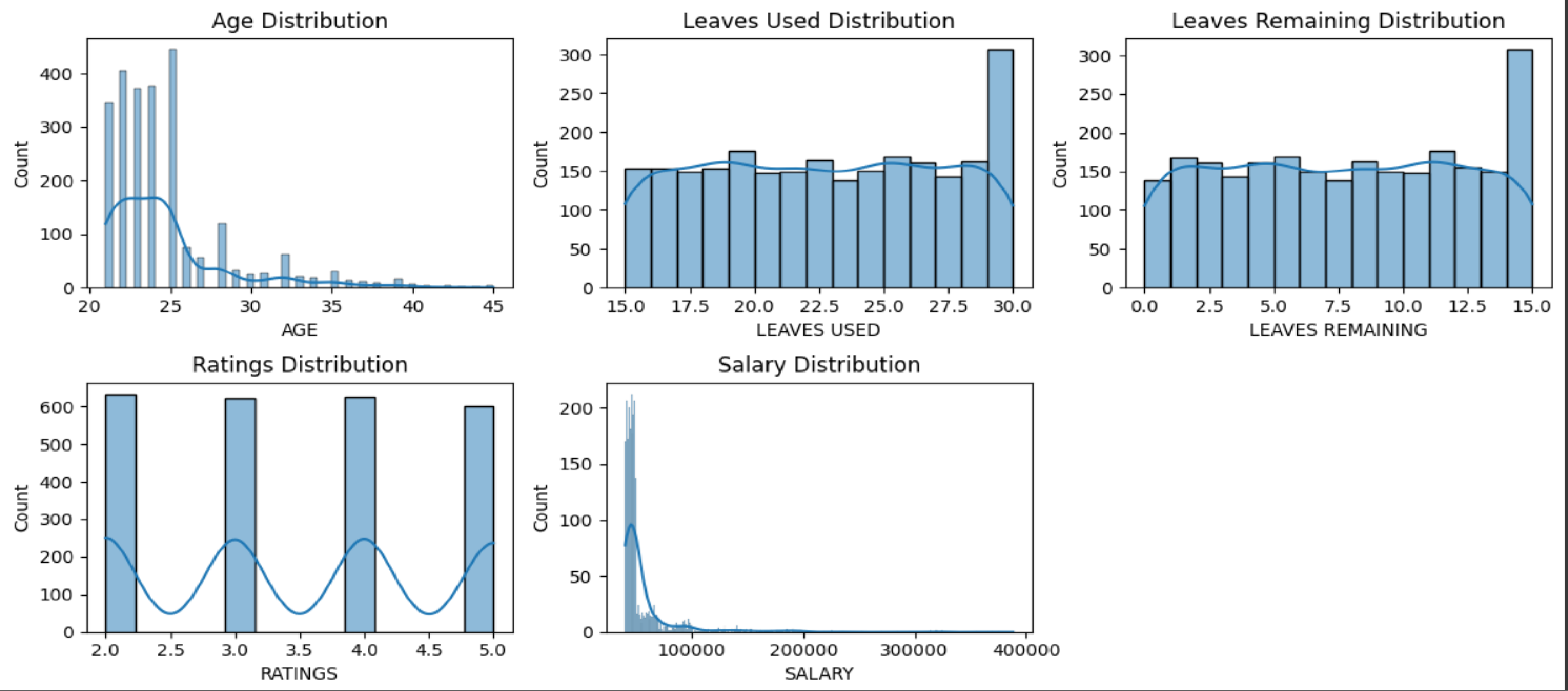
EXPLORATORY DATA ANALYSIS

- Unit



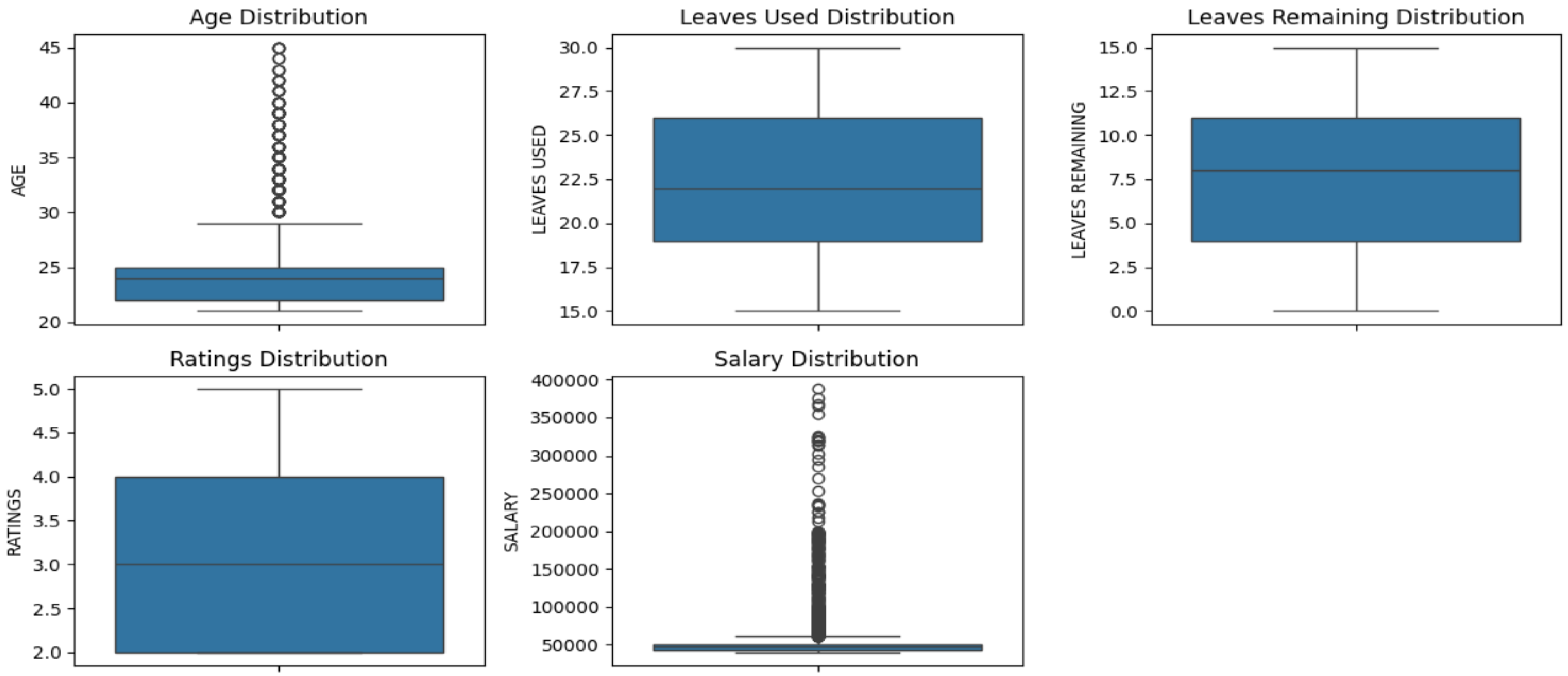
EXPLORATORY DATA ANALYSIS

Numerical columns Distribution



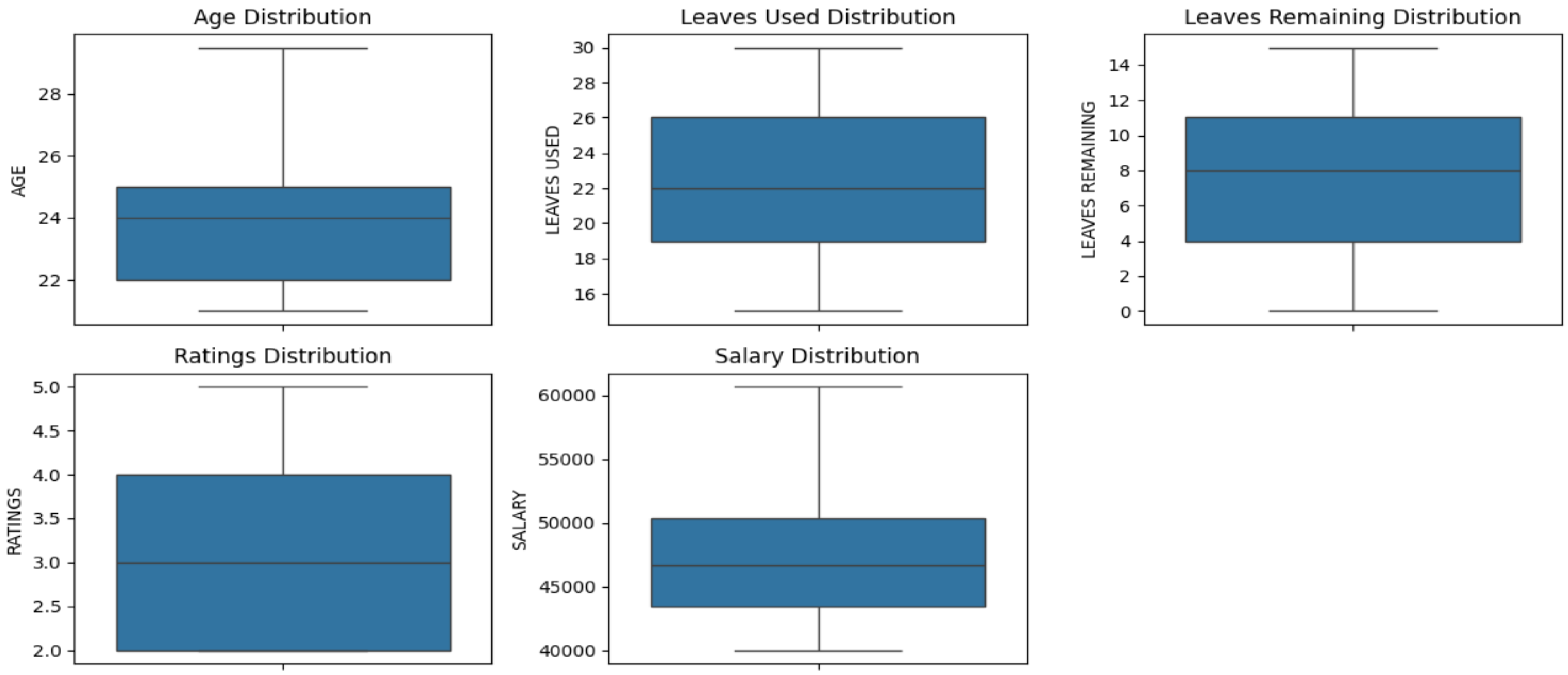
EXPLORATORY DATA ANALYSIS

Outliers



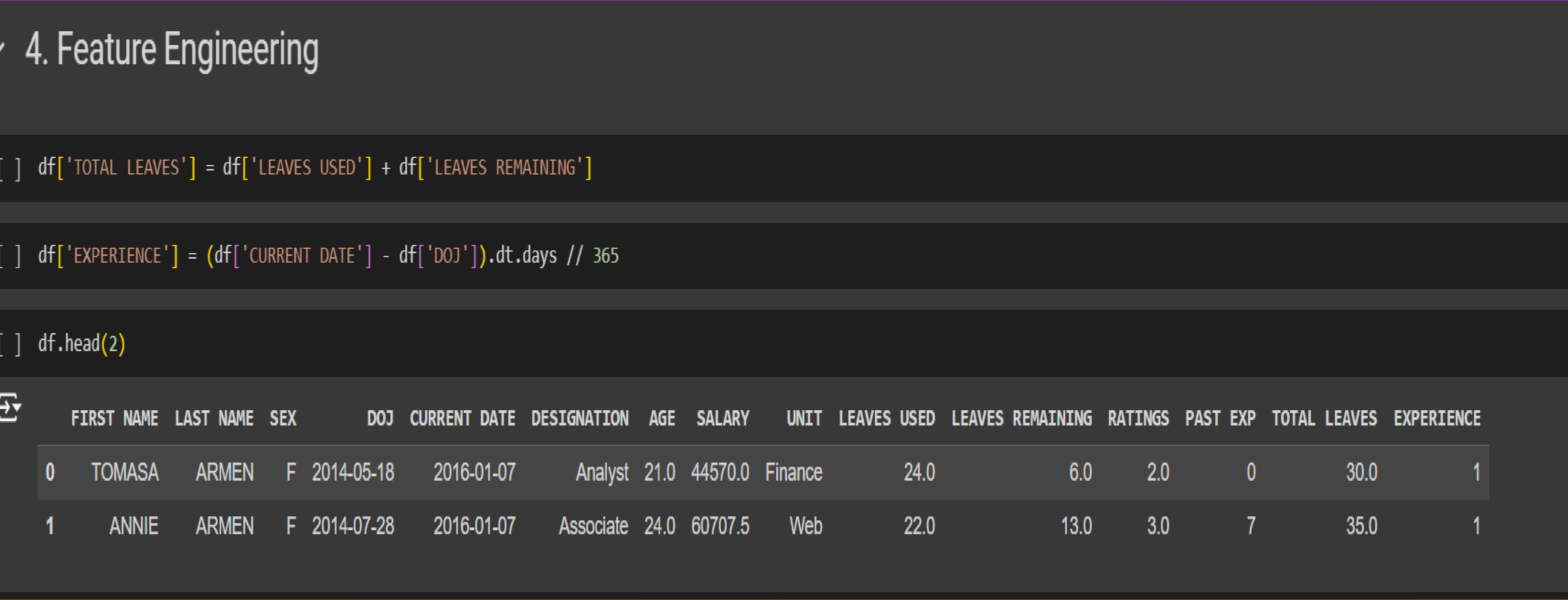
EXPLORATORY DATA ANALYSIS

Outliers Removing



4. FEATURE ENGINEERING

Creating new columns



4. Feature Engineering

```
[ ] df['TOTAL LEAVES'] = df['LEAVES USED'] + df['LEAVES REMAINING']
```

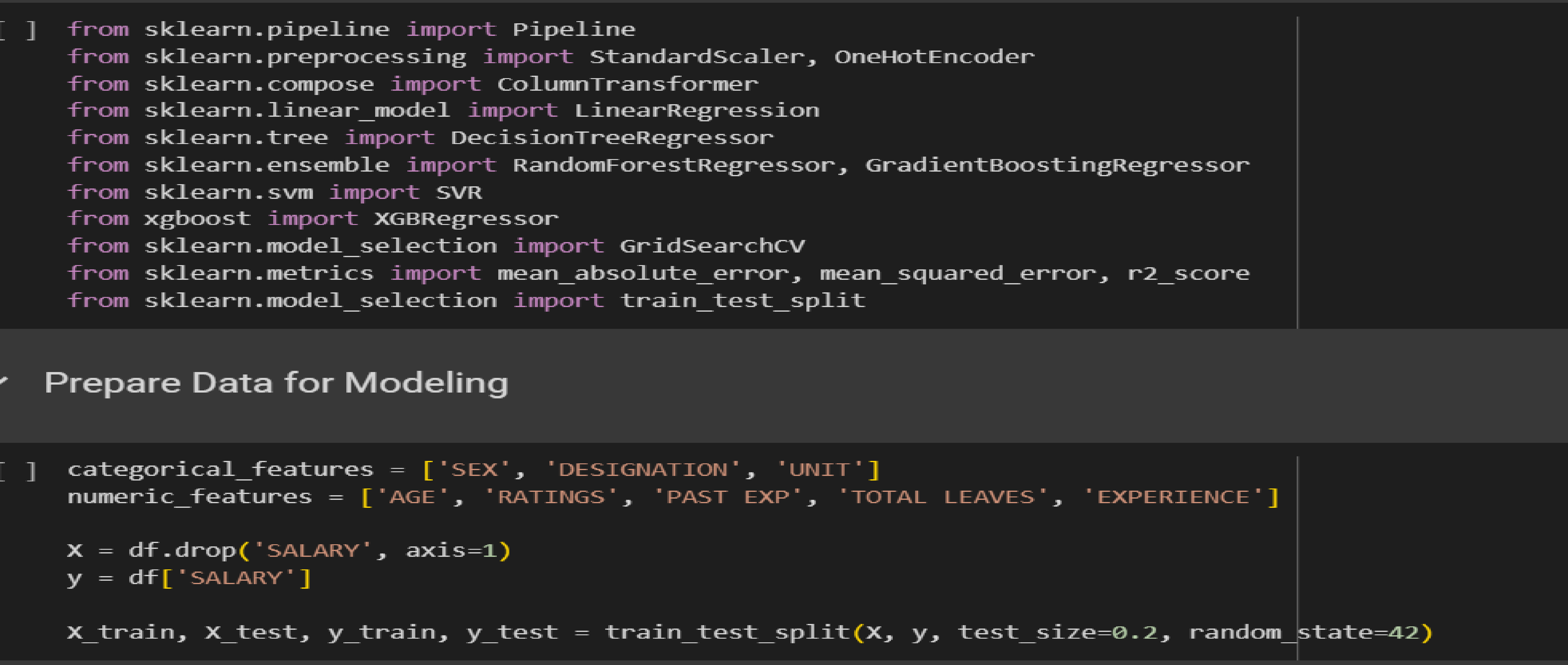
```
[ ] df['EXPERIENCE'] = (df['CURRENT DATE'] - df['DOJ']).dt.days // 365
```

```
[ ] df.head(2)
```

	FIRST NAME	LAST NAME	SEX	DOJ	CURRENT DATE	DESIGNATION	AGE	SALARY	UNIT	LEAVES USED	LEAVES REMAINING	RATINGS	PAST EXP	TOTAL LEAVES	EXPERIENCE
0	TOMASA	ARMEN	F	2014-05-18	2016-01-07	Analyst	21.0	44570.0	Finance	24.0	6.0	2.0	0	30.0	1
1	ANNIE	ARMEN	F	2014-07-28	2016-01-07	Associate	24.0	60707.5	Web	22.0	13.0	3.0	7	35.0	1

4. DATA PREPROCESSING

Creating Pipeline



```
[ ]  from sklearn.pipeline import Pipeline
     from sklearn.preprocessing import StandardScaler, OneHotEncoder
     from sklearn.compose import ColumnTransformer
     from sklearn.linear_model import LinearRegression
     from sklearn.tree import DecisionTreeRegressor
     from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
     from sklearn.svm import SVR
     from xgboost import XGBRegressor
     from sklearn.model_selection import GridSearchCV
     from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
     from sklearn.model_selection import train_test_split
```

Prepare Data for Modeling

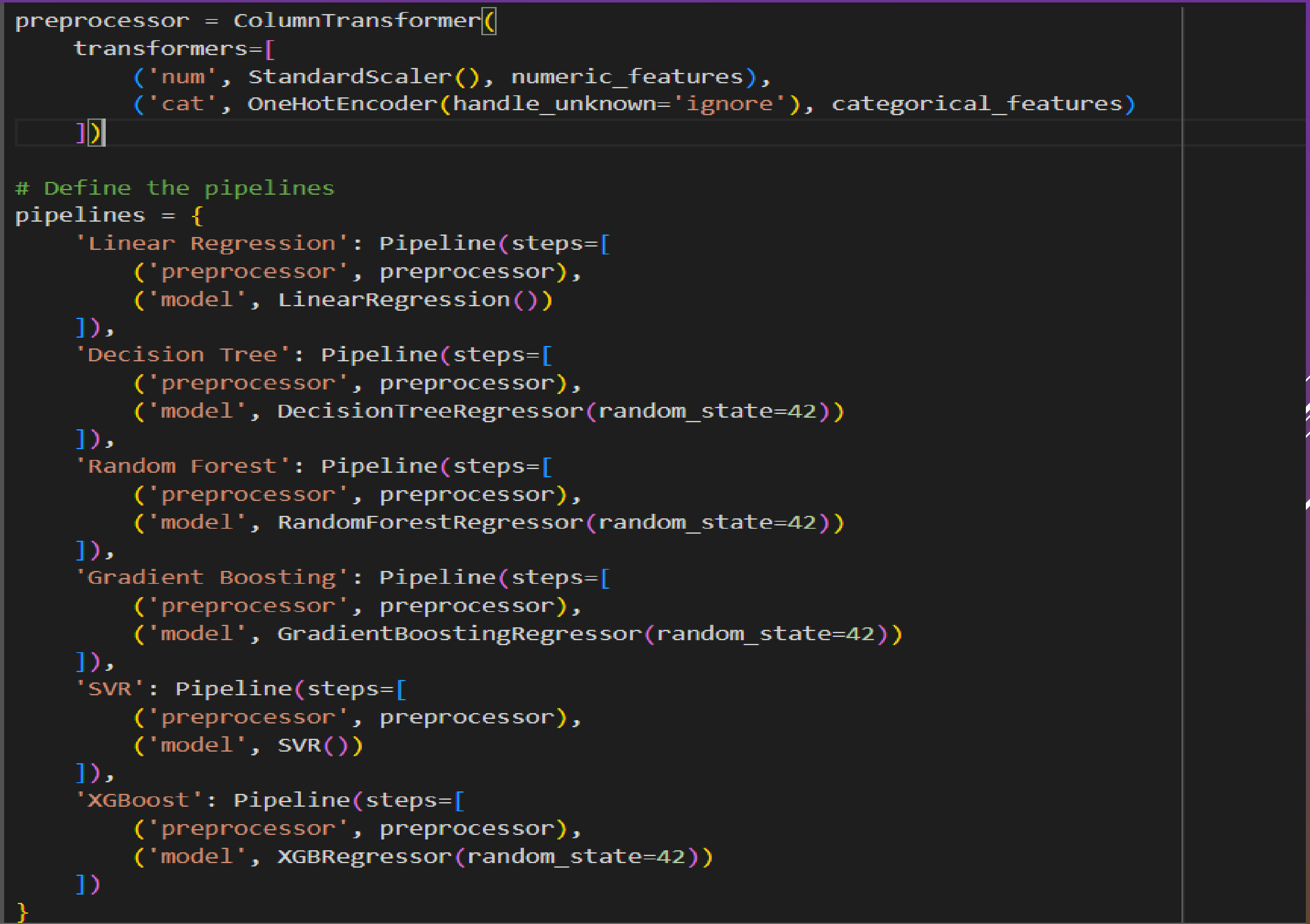
```
[ ]  categorical_features = ['SEX', 'DESIGNATION', 'UNIT']
     numeric_features = ['AGE', 'RATINGS', 'PAST EXP', 'TOTAL LEAVES', 'EXPERIENCE']

     X = df.drop('SALARY', axis=1)
     y = df['SALARY']

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

5. MACHINE LEARNING MODEL

Creating Pipeline for Multiple model

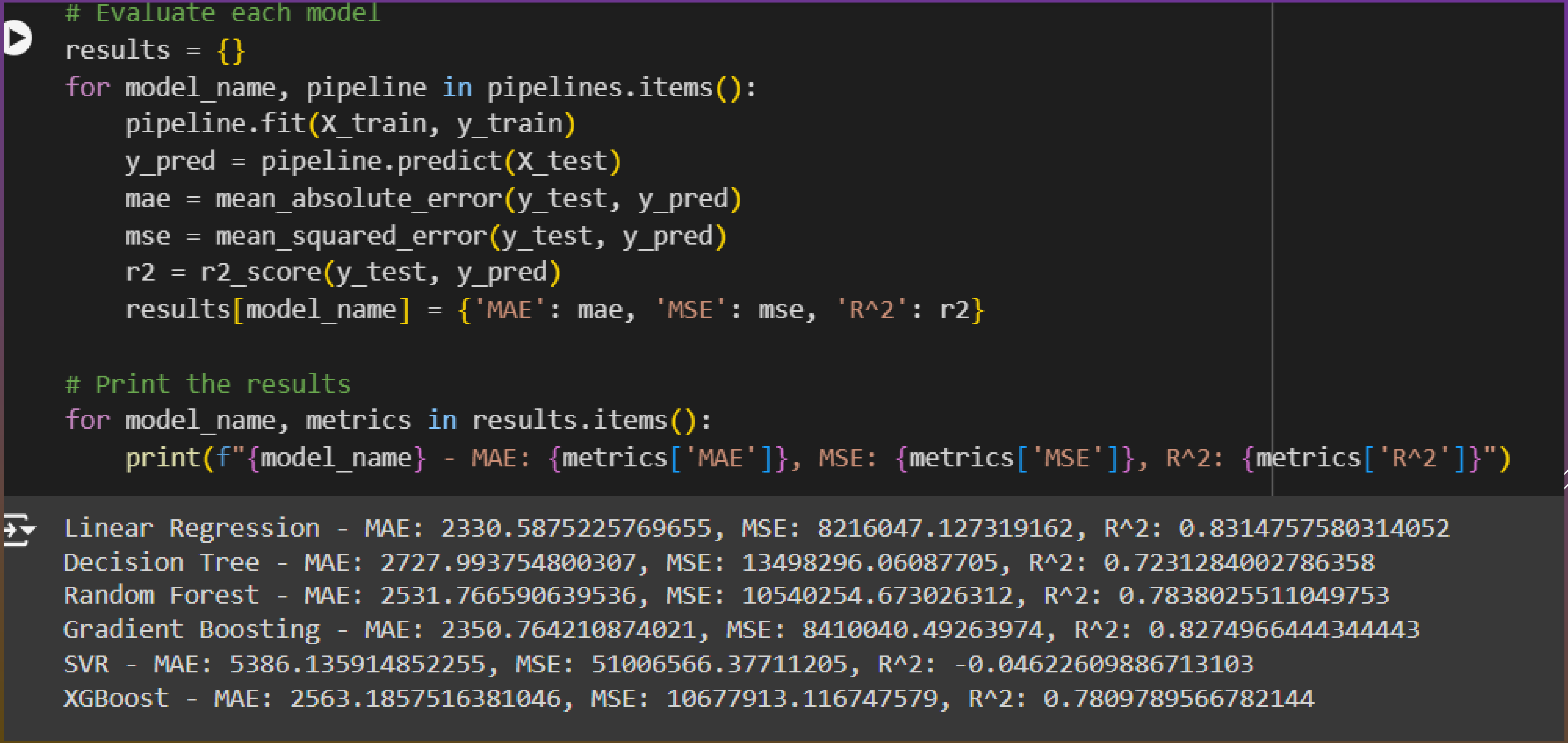


```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])

# Define the pipelines
pipelines = {
    'Linear Regression': Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('model', LinearRegression())
    ]),
    'Decision Tree': Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('model', DecisionTreeRegressor(random_state=42))
    ]),
    'Random Forest': Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('model', RandomForestRegressor(random_state=42))
    ]),
    'Gradient Boosting': Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('model', GradientBoostingRegressor(random_state=42))
    ]),
    'SVR': Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('model', SVR())
    ]),
    'XGBoost': Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('model', XGBRegressor(random_state=42))
    ])
}
```

6. MODEL EVALUATION

Checking mae, mse, r2-score



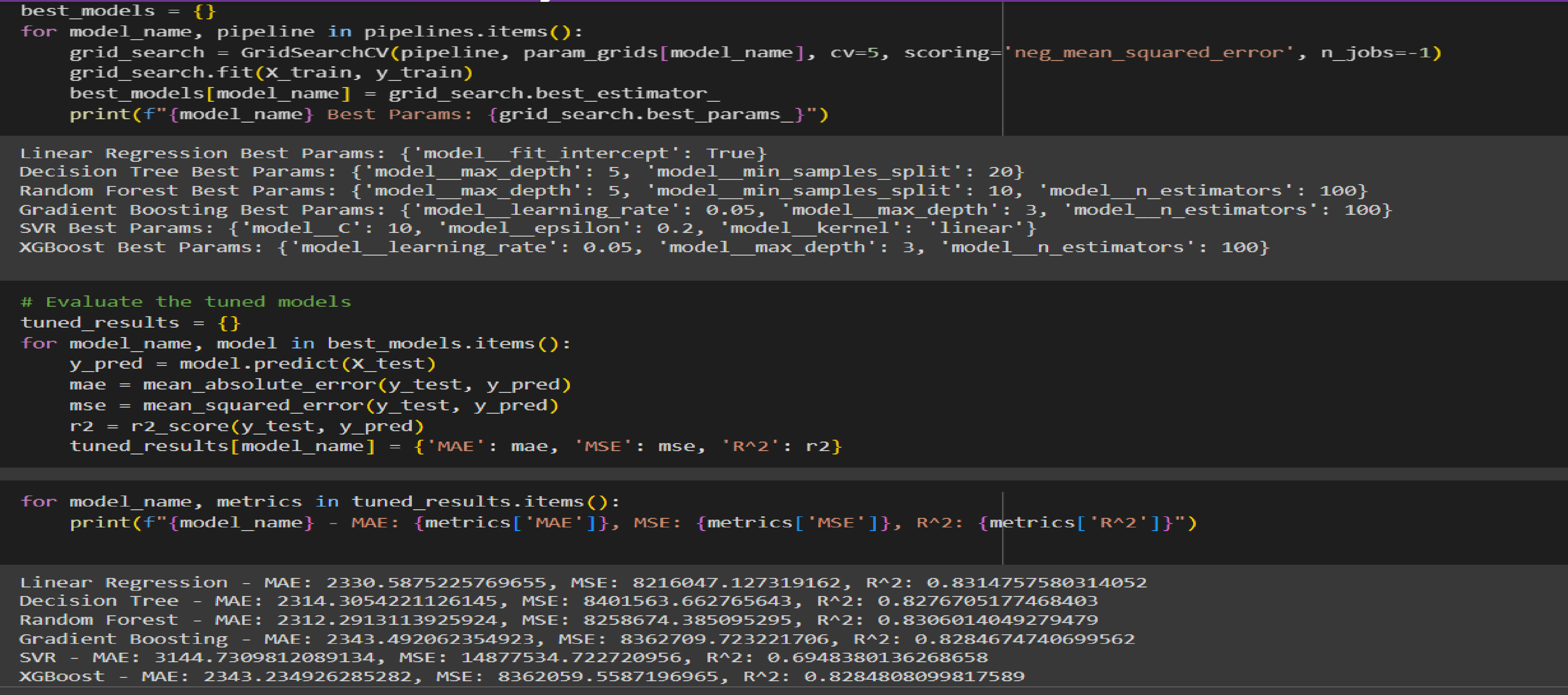
```
# Evaluate each model
results = {}
for model_name, pipeline in pipelines.items():
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results[model_name] = {'MAE': mae, 'MSE': mse, 'R^2': r2}

# Print the results
for model_name, metrics in results.items():
    print(f"{model_name} - MAE: {metrics['MAE']}, MSE: {metrics['MSE']}, R^2: {metrics['R^2']}")
```

```
Linear Regression - MAE: 2330.5875225769655, MSE: 8216047.127319162, R^2: 0.8314757580314052
Decision Tree - MAE: 2727.993754800307, MSE: 13498296.06087705, R^2: 0.7231284002786358
Random Forest - MAE: 2531.766590639536, MSE: 10540254.673026312, R^2: 0.7838025511049753
Gradient Boosting - MAE: 2350.764210874021, MSE: 8410040.49263974, R^2: 0.8274966444344443
SVR - MAE: 5386.135914852255, MSE: 51006566.37711205, R^2: -0.046226098867131O3
XGBoost - MAE: 2563.1857516381046, MSE: 10677913.116747579, R^2: 0.7809789566782144
```

6. HYPERPARAMETER TUNING

Find best model accuracy



```
best_models = {}
for model_name, pipeline in pipelines.items():
    grid_search = GridSearchCV(pipeline, param_grids[model_name], cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
    grid_search.fit(X_train, y_train)
    best_models[model_name] = grid_search.best_estimator_
    print(f"{model_name} Best Params: {grid_search.best_params_}")
```

```
Linear Regression Best Params: {'model__fit_intercept': True}
Decision Tree Best Params: {'model__max_depth': 5, 'model__min_samples_split': 20}
Random Forest Best Params: {'model__max_depth': 5, 'model__min_samples_split': 10, 'model__n_estimators': 100}
Gradient Boosting Best Params: {'model__learning_rate': 0.05, 'model__max_depth': 3, 'model__n_estimators': 100}
SVR Best Params: {'model__C': 10, 'model__epsilon': 0.2, 'model__kernel': 'linear'}
XGBoost Best Params: {'model__learning_rate': 0.05, 'model__max_depth': 3, 'model__n_estimators': 100}
```

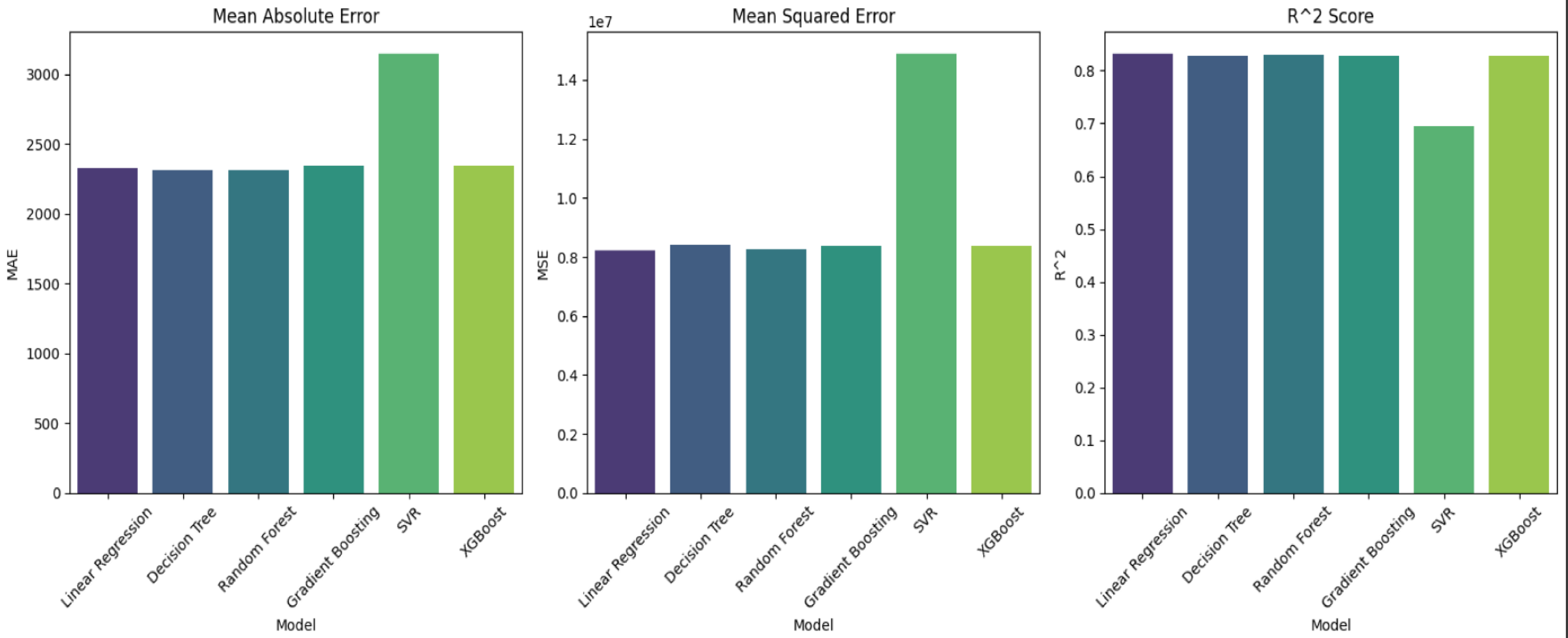
```
# Evaluate the tuned models
tuned_results = {}
for model_name, model in best_models.items():
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    tuned_results[model_name] = {'MAE': mae, 'MSE': mse, 'R^2': r2}
```

```
for model_name, metrics in tuned_results.items():
    print(f"{model_name} - MAE: {metrics['MAE']}, MSE: {metrics['MSE']}, R^2: {metrics['R^2']}")
```

```
Linear Regression - MAE: 2330.5875225769655, MSE: 8216047.127319162, R^2: 0.8314757580314052
Decision Tree - MAE: 2314.3054221126145, MSE: 8401563.662765643, R^2: 0.8276705177468403
Random Forest - MAE: 2312.2913113925924, MSE: 8258674.385095295, R^2: 0.8306014049279479
Gradient Boosting - MAE: 2343.492062354923, MSE: 8362709.723221706, R^2: 0.8284674740699562
SVR - MAE: 3144.7309812089134, MSE: 14877534.722720956, R^2: 0.6948380136268658
XGBoost - MAE: 2343.234926285282, MSE: 8362059.5587196965, R^2: 0.8284808099817589
```

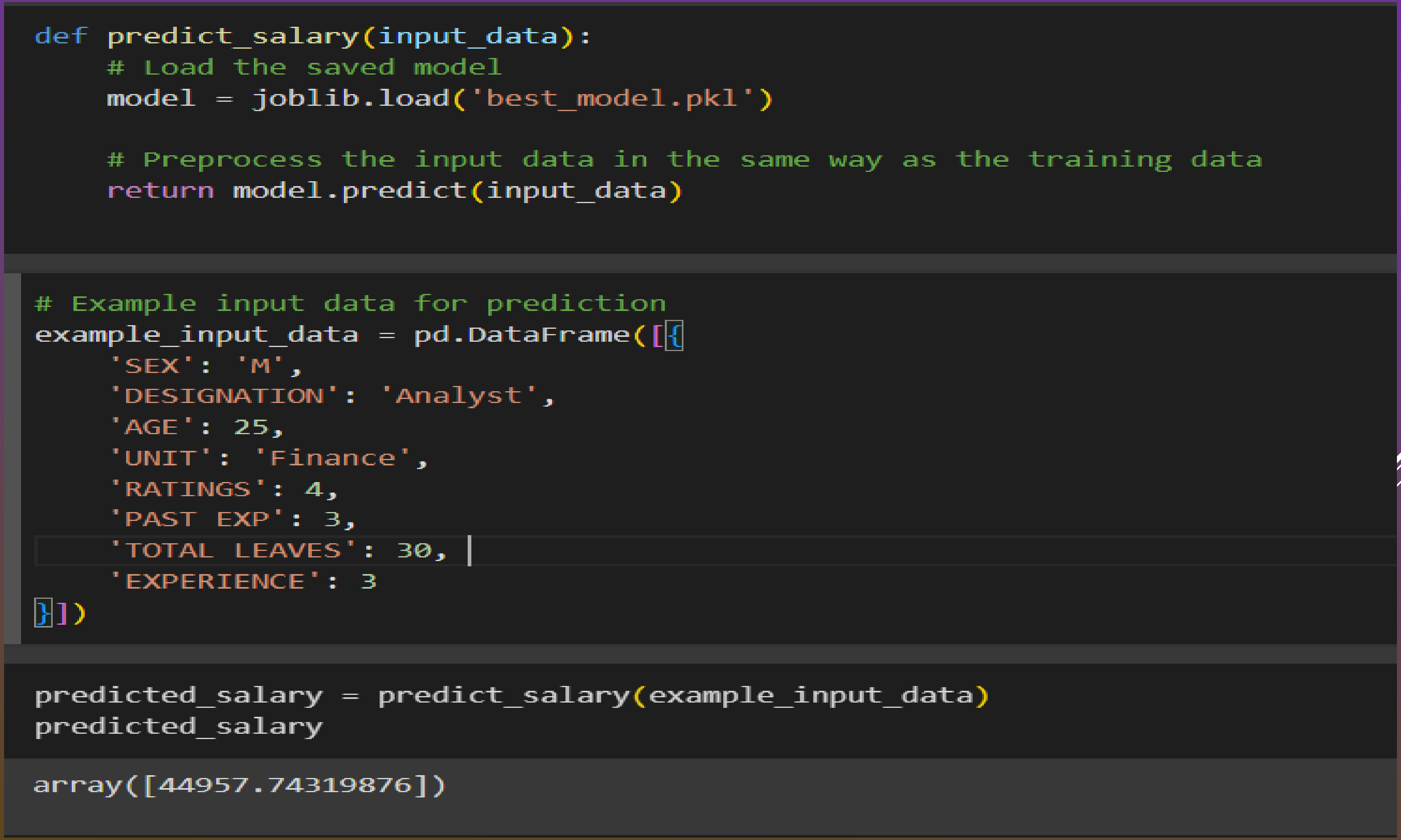
6. HYPERPARAMETER TUNING

Here can see best model in graph



7. FINAL RECOMMENDATIONS

Here can see best model in code



```
def predict_salary(input_data):
    # Load the saved model
    model = joblib.load('best_model.pkl')

    # Preprocess the input data in the same way as the training data
    return model.predict(input_data)
```

```
# Example input data for prediction
example_input_data = pd.DataFrame([{
    'SEX': 'M',
    'DESIGNATION': 'Analyst',
    'AGE': 25,
    'UNIT': 'Finance',
    'RATINGS': 4,
    'PAST EXP': 3,
    'TOTAL LEAVES': 30,
    'EXPERIENCE': 3
}])
```

```
predicted_salary = predict_salary(example_input_data)
predicted_salary
```

```
array([44957.74319876])
```

8. CONCLUSION

- Linear Regression is the best-performing model with an R^2 score of 0.8315, indicating it explains approximately 83% of the variance in salary.
- Random Forest and Gradient Boosting models also performed well with R^2 scores of 0.8306 and 0.8285 respectively.
- SVR had the lowest performance with an R^2 score of 0.6948, suggesting it is less effective for this task.
- Overall, Linear Regression is recommended for salary prediction due to its strong performance across key metrics.