Salary Prediction PROJECT

By Rana Alaa Ahmed

INTRODUCTION

In this project, we aim to predict the salaries of data professionals by leveraging a comprehensive dataset and applying various machine learning techniques. The process involves several key steps, each contributing to the overall goal of accurate salary prediction and insightful analysis of factors influencing earnings in data professions.

PROJECT OBJECTIVES

01

Exploratory Data Analysis

↓

Conduct comprehensive EDA to unveil valuable insights about data professionals' salaries.

02

Feature Engineering

↓

Create new features or transform existing ones to provide additional insights or improve model performance.

03

Data Preprocessing

↓

Prepare the data for model training by handling missing values, encoding categorical variables, and scaling or normalizing features as needed.

04

Machine Learning Model

↓

Train various machine learning regression models to predict salaries.

05

Model Evaluation

↓

Identify the model that provides the most accurate salary predictions.

06

ML Pipeline

↓

Create ML Pipeline to streamline the end-to-end machine learning process, from data preprocessing to model training.

EXPLORATORY DATA ANALYSIS

01

- Understand the dataset
- Dive into the dataset to understand its structure and content.
- Overview of data columns and types

02

- Summary statistics for numerical columns
- Conduct comprehensive EDA to unveil valuable insights about data professionals' salaries.
- Utilize data visualization and summary statistics to identify patterns and trends.

03

- Visualize and summarize key statistics
- Distribution of numerical features

```
# Display the first five rows of the DataFrame 'data'.
data.head().T
```

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

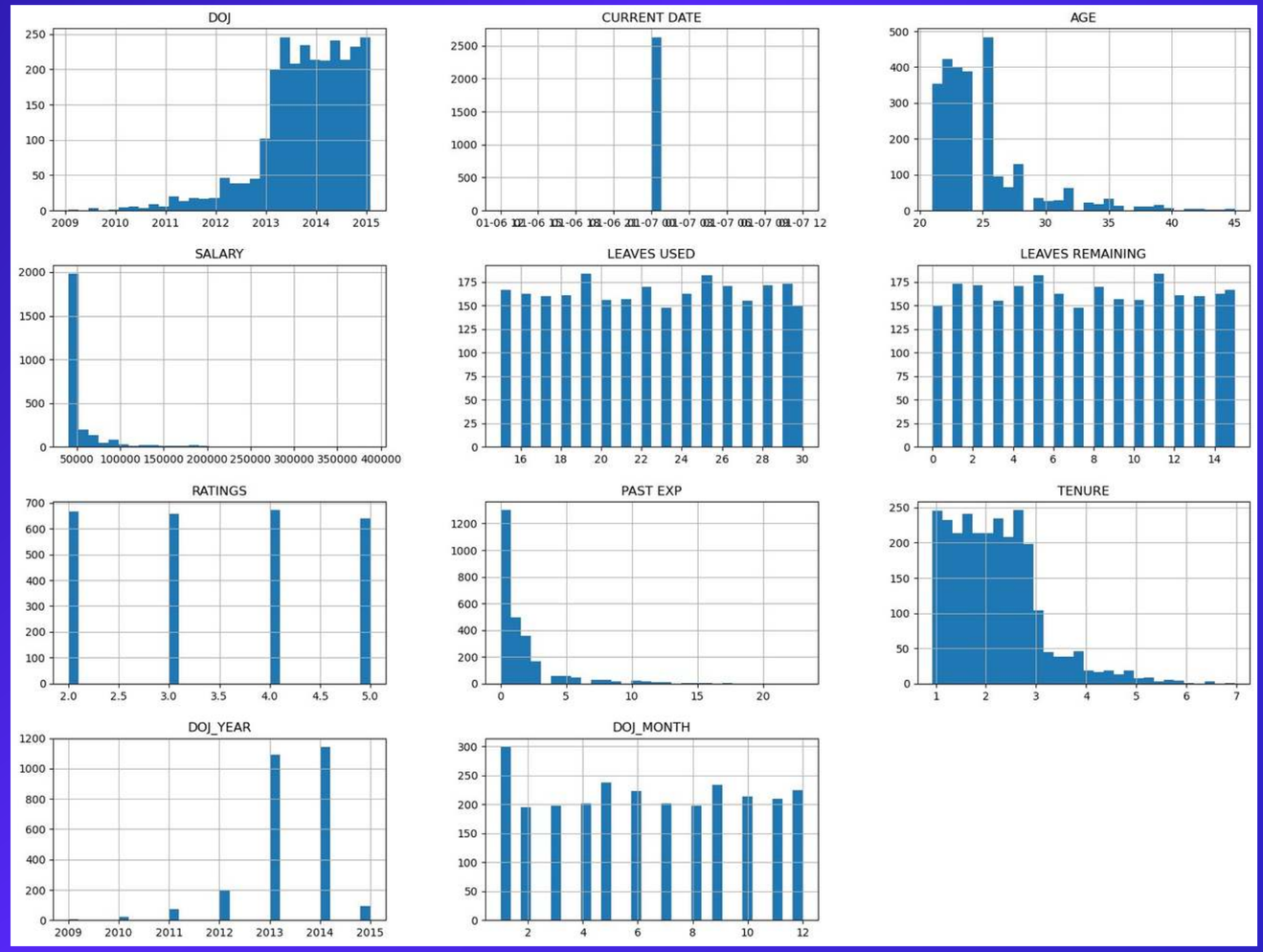
```
# Output a concise summary of the DataFrame 'data',
# including information about the index dtype and column dtypes,
# non-null values, and memory usage.
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2639 entries, 0 to 2638
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   FIRST NAME        2639 non-null   object
 1   LAST NAME         2637 non-null   object
 2   SEX               2639 non-null   object
 3   DOJ               2638 non-null   object
 4   CURRENT DATE      2639 non-null   object
 5   DESIGNATION       2639 non-null   object
 6   AGE               2636 non-null   float64
 7   SALARY            2639 non-null   int64
 8   UNIT              2639 non-null   object
 9   LEAVES USED       2636 non-null   float64
 10  LEAVES REMAINING  2637 non-null   float64
 11  RATINGS           2637 non-null   float64
 12  PAST EXP          2639 non-null   int64
dtypes: float64(4), int64(2), object(7)
memory usage: 268.2+ KB
```

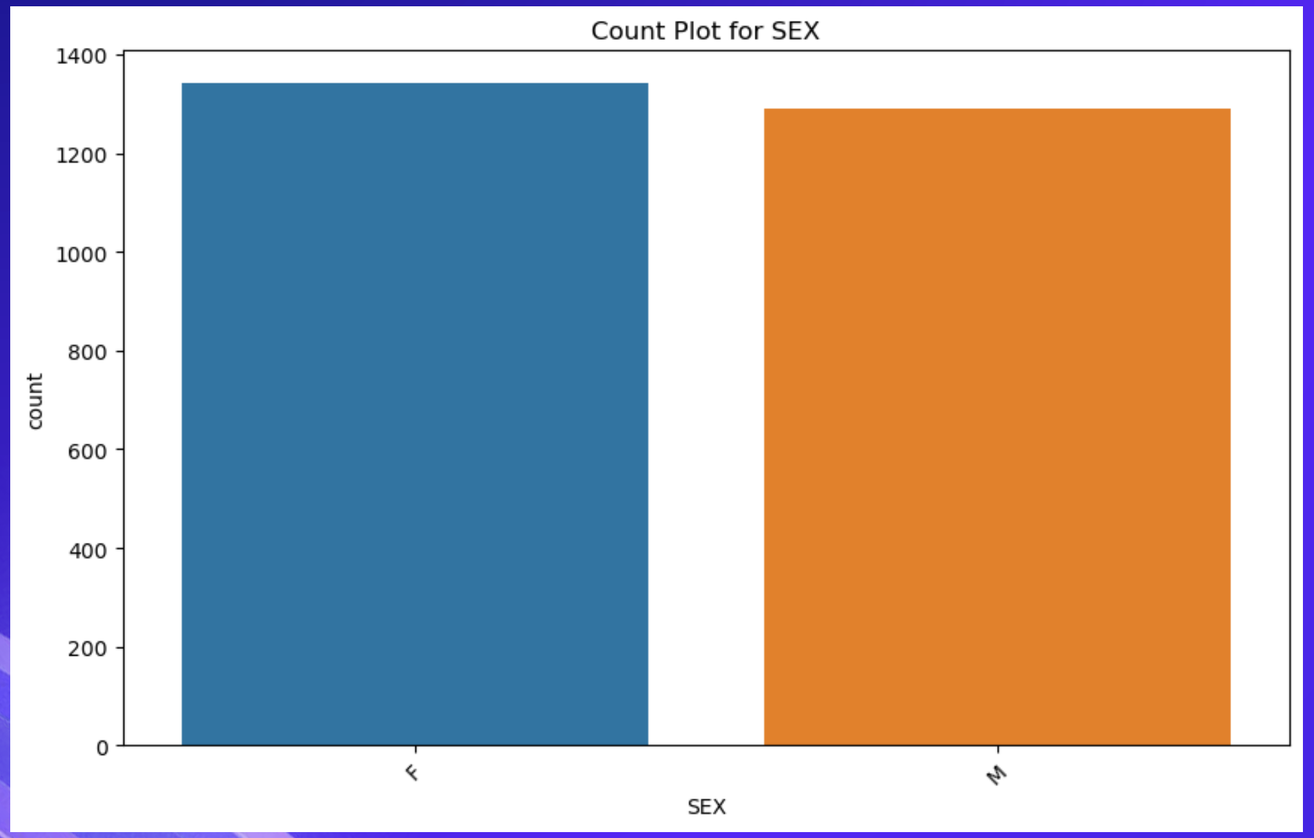
```
# Generate descriptive statistics of the numerical columns in the DataFrame 'data'
# and transpose the result for better readability.
data.describe().T
```

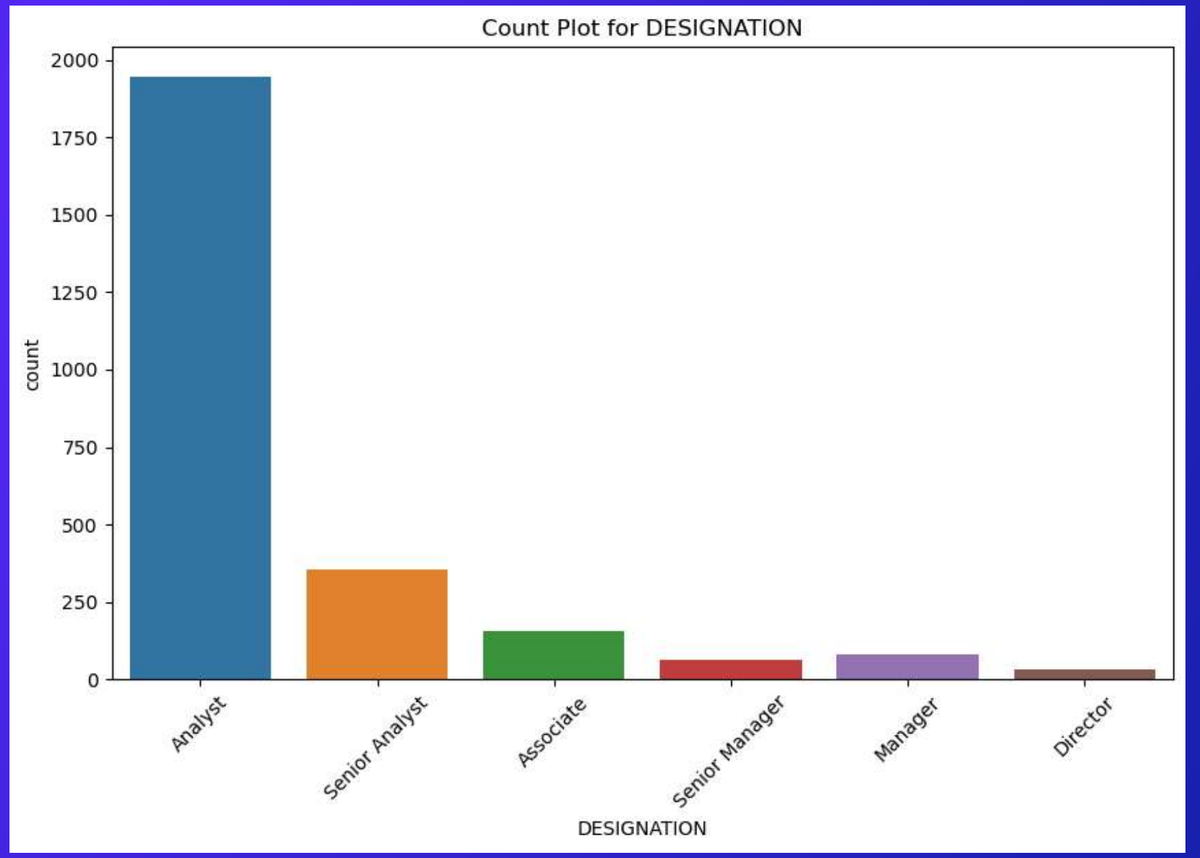
	count	mean	std	min	25%	50%	75%	max
AGE	2636.0	24.756449	3.908228	21.0	22.0	24.0	25.0	45.0
SALARY	2639.0	58136.678287	36876.956944	40001.0	43418.0	46781.0	51401.5	388112.0
LEAVES USED	2636.0	22.501517	4.604469	15.0	19.0	22.0	26.0	30.0
LEAVES REMAINING	2637.0	7.503223	4.603193	0.0	4.0	8.0	11.0	15.0
RATINGS	2637.0	3.486159	1.114933	2.0	2.0	3.0	4.0	5.0
PAST EXP	2639.0	1.566881	2.728416	0.0	0.0	1.0	2.0	23.0

```
# Display histograms for all numerical features in the DataFrame 'data'.
data.hist(bins=30, figsize=(20, 15))
plt.show()
```

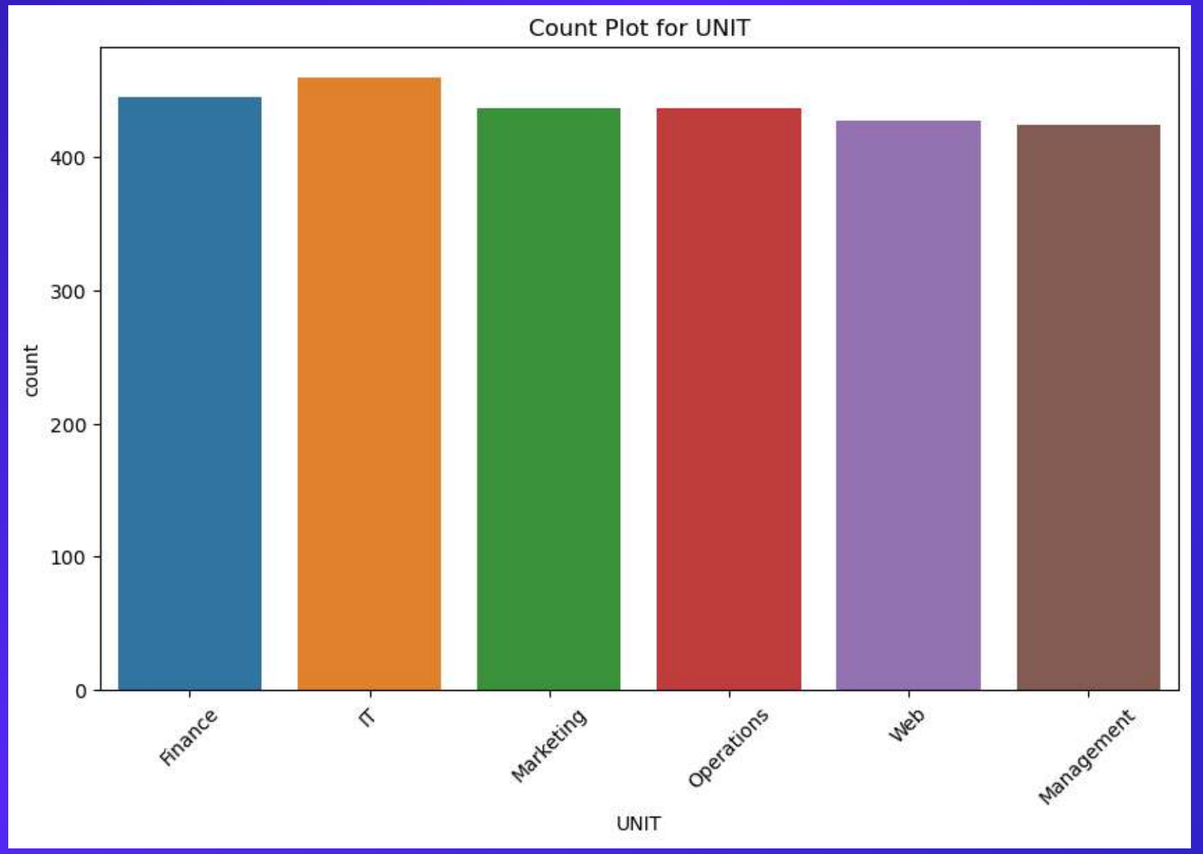


```python
# Identify categorical columns (excluding "FIRST NAME" and "LAST NAME" columns) in the DataFrame 'data'.
categorical_columns = data.select_dtypes(include=['object']).columns.tolist()

# Generate bar plots for each categorical column.
for column in categorical_columns:
    plt.figure(figsize=(10, 6))
    sns.countplot(x=column, data=data)
    plt.title(f'Count Plot for {column}')
    plt.xticks(rotation=45)
    plt.show()
```

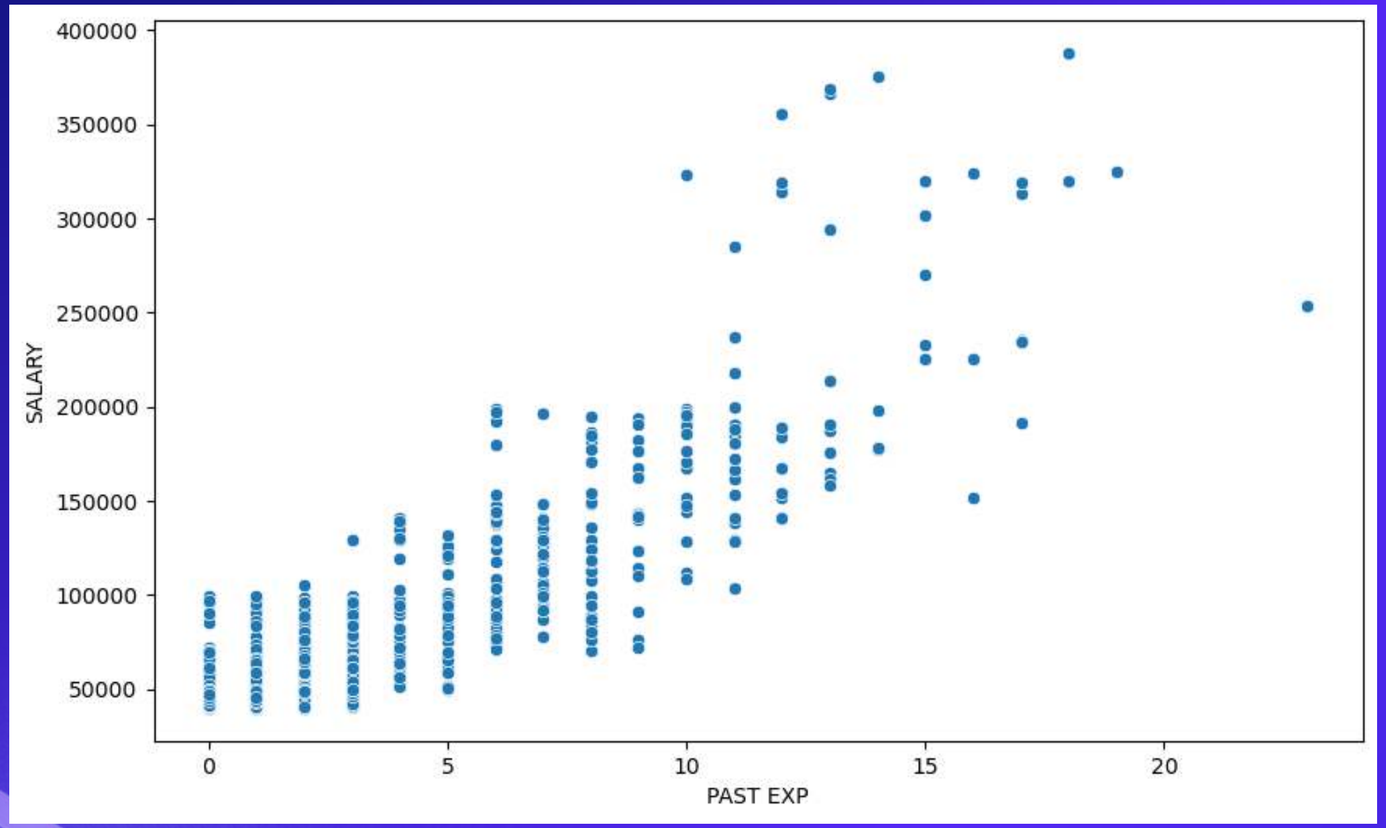




```python
# Identify categorical columns (excluding "FIRST NAME" and "LAST NAME" columns) in the DataFrame 'data'.
categorical_columns = data.select_dtypes(include=['object']).columns.tolist()

# Generate bar plots for each categorical column.
for column in categorical_columns:
    plt.figure(figsize=(10, 6))
    sns.countplot(x=column, data=data)
    plt.title(f'Count Plot for {column}')
    plt.xticks(rotation=45)
    plt.show()
```

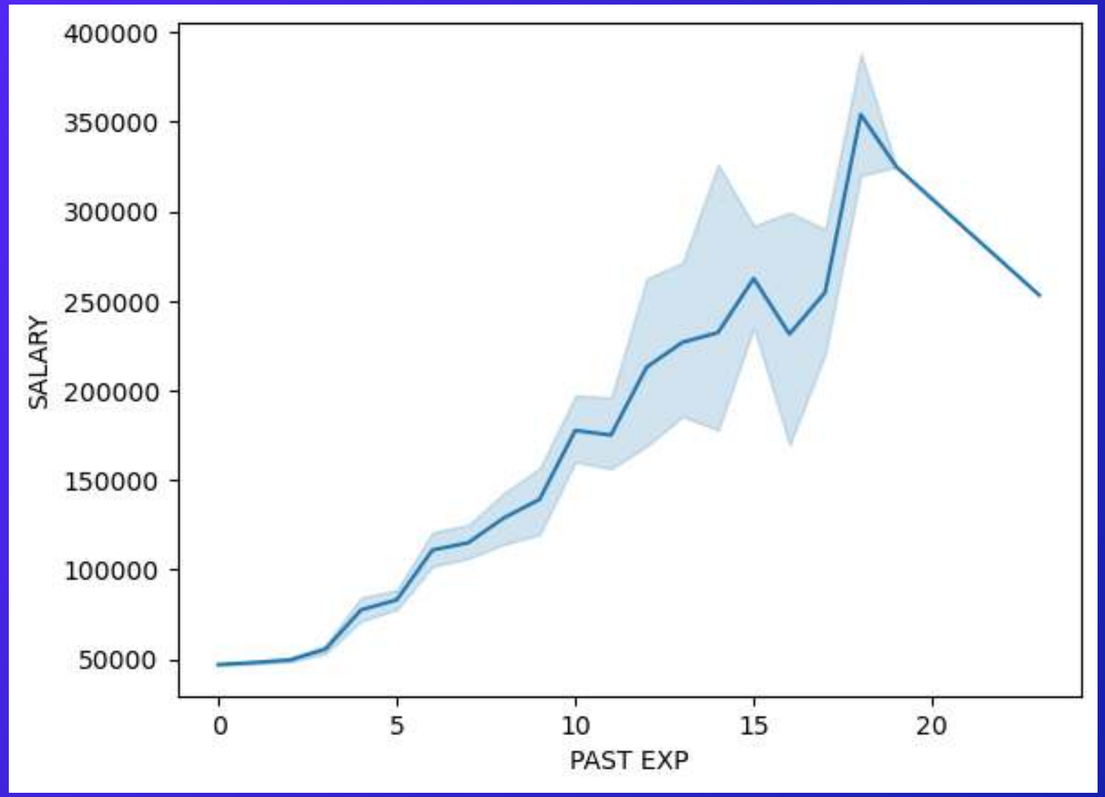


```
# Display a scatter plot to visualize the relationship between 'PAST EXP' and 'SALARY' in the DataFrame 'data'.
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PAST EXP', y='SALARY', data=data)
plt.show()
```




```
# Display a line plot to visualize the trend between 'PAST EXP' and '
sns.lineplot(x='PAST EXP', y='SALARY', data=data)
```

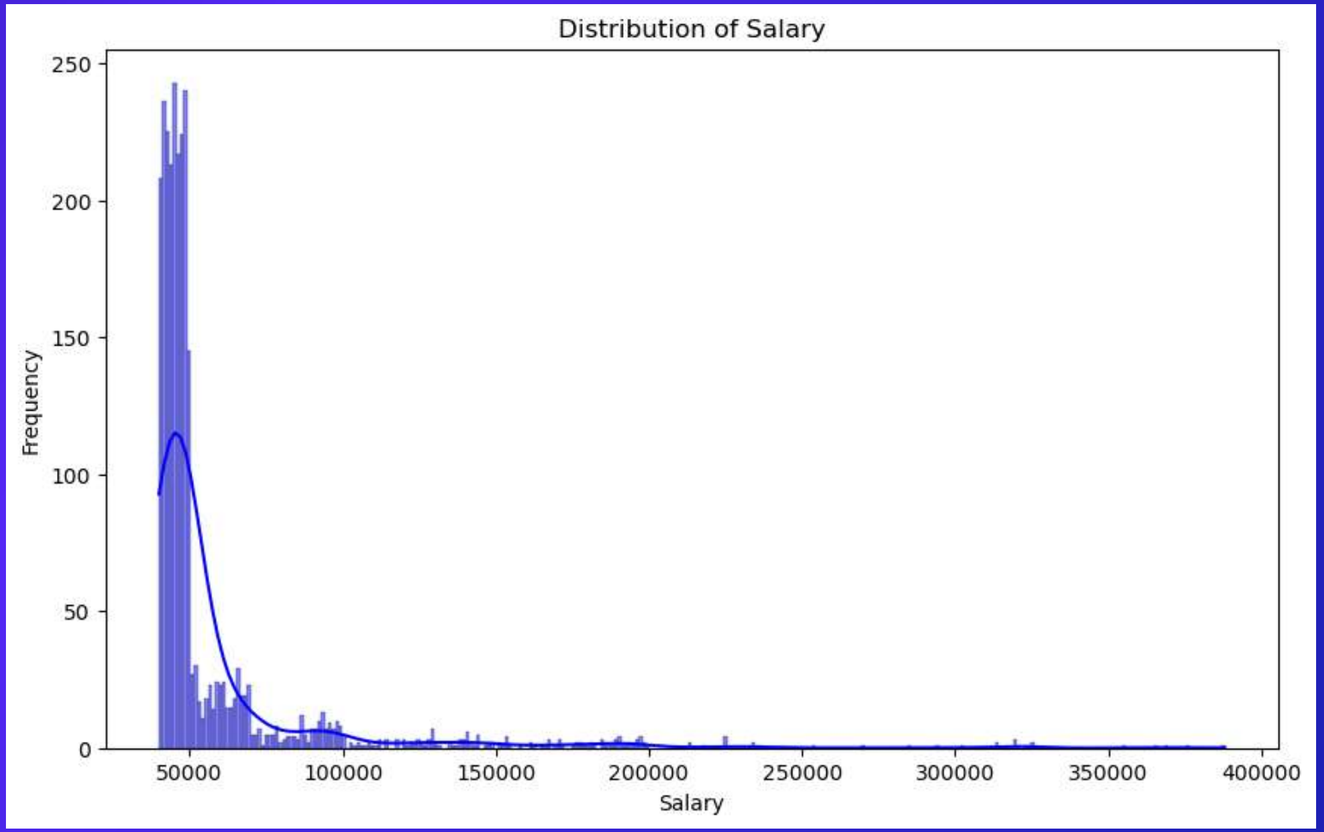
```
<Axes: xlabel='PAST EXP', ylabel='SALARY'>
```



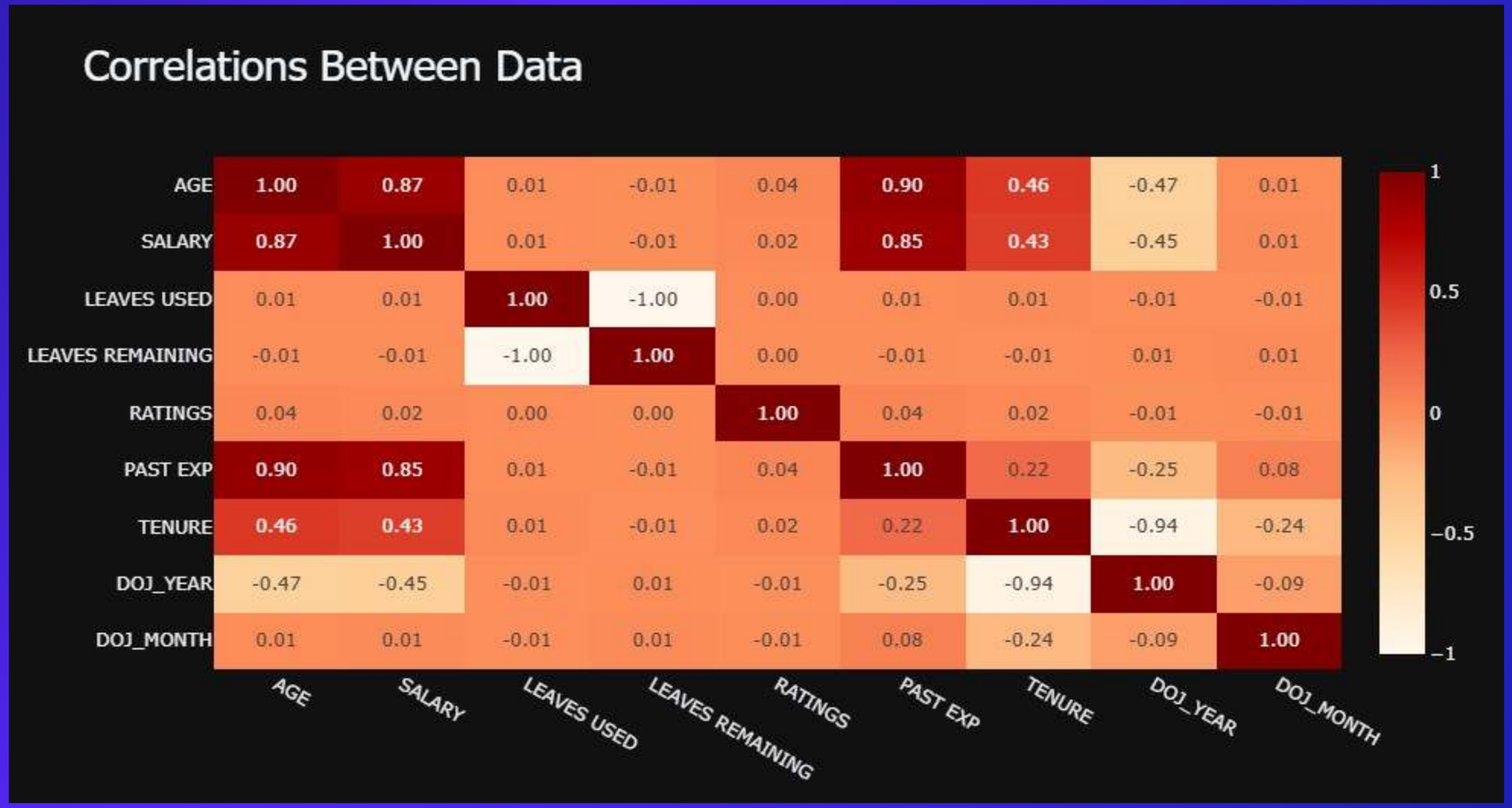
- Visualizing the relationship between past experience and salary shows whether more experienced individuals tend to have higher salaries.

```
# Calculate and visualize the distribution of the target variable 'SALARY'.
plt.figure(figsize=(10, 6))
sns.histplot(data['SALARY'], kde=True, color='blue')
plt.title('Distribution of Salary')
plt.xlabel('Salary')
plt.ylabel('Frequency')
plt.show()
```

- Understanding the overall distribution of salaries helps in identifying common salary ranges and outliers.



```python
# Calculate the correlation matrix for
# numerical columns in the DataFrame 'data'.
correlation = data.corr(numeric_only=True)

# Plot the correlation matrix using Plotly.
fig = px.imshow(
    correlation,
    template="plotly_dark",
    text_auto="0.2f",
    aspect=1,
    color_continuous_scale="orrd",
    title="Correlations Between Data"
)

# Update the layout of the plot.
fig.update_layout(
    title={
        "font": {
            "size": 28,
            "family": "tahoma"
        }
    }
)
fig
```



FEATURE ENGINEERING

01

- Create new features or transform existing ones to provide additional insights or improve model performance.
- Extracted month and year from DOJ
- Calculated tenure in years
- Created tenure buckets for categorization

```
# Experience within the Company
# Calculate the tenure of each employee, converting the result from days to years.
data['TENURE'] = (data['CURRENT DATE'] - data['DOJ']).dt.days / 365
```

- Longer tenure generally correlates with higher salary, indicating that experience within the company is rewarded.

```
# Salary Growth Over Time
# Calculate the average salary for each year of joining ('DOJ_YEAR') by grouping the data by 'DOJ_YEAR' and then computing the me
# Reset the index to make the resulting DataFrame more manageable.
data['DOJ_YEAR'] = data['DOJ'].dt.year
salary_trend = data.groupby('DOJ_YEAR')['SALARY'].mean().reset_index()
salary_trend
```

	DOJ_YEAR	SALARY
0	2009	302197.200000
1	2010	181548.125000
2	2011	121216.506667
3	2012	90680.365000
4	2013	51236.251601
5	2014	51511.858268
6	2015	54147.054348

- Salary trends over the years can reveal how the market value for data professions has changed, potentially showing an increase in average salaries for more recent hires.

```
# Calculate retention rates by categorizing tenure into buckets: 0-1 year, 1-3 years, 3-5 years, 5-10 years, and 10+ years.
data['TENURE_BUCKET'] = pd.cut(data['TENURE'], bins=[0, 1, 3, 5, 10, np.inf], labels=['0-1', '1-3', '3-5', '5-10', '10+'])
# Compute the retention rates by counting the occurrences of each tenure bucket and normalizing by the total number of records.
retention_rates = data['TENURE_BUCKET'].value_counts(normalize=True)
retention_rates
```

```
TENURE_BUCKET
1-3     0.847644
3-5     0.111702
0-1     0.029635
5-10    0.011018
10+     0.000000
Name: proportion, dtype: float64
```

- Most employees fall into specific tenure buckets (1-3 years or 3-5 years), which can inform HR strategies regarding employee retention and turnover.

```python
# Extract the month from the 'DOJ' column and store it in a new column 'DOJ_MONTH'.
data['DOJ_MONTH'] = data['DOJ'].dt.month
# Calculate the count of employees hired in each month and sort the result by month index.
hiring_trends = data['DOJ_MONTH'].value_counts().sort_index()
hiring_trends
```

```
DOJ_MONTH
1     299
2     195
3     198
4     201
5     238
6     223
7     201
8     197
9     233
10    214
11    209
12    224
Name: count, dtype: int64
```

- Identifying peak hiring periods can help in workforce planning and understanding seasonal hiring trends.

DATA PREPROCESSING

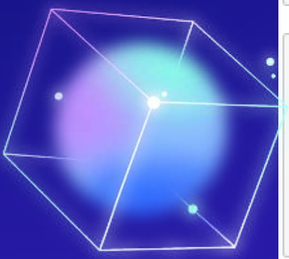
01

- Prepare the data for model training by :
  - Drop irrelevant columns
  - Handle missing values
  - Encode categorical variables
  - scale or normalize features as needed.

```
# Drop the "FIRST NAME" and "LAST NAME" columns from the DataFrame 'data' as they are not expected to provide significant insight
# and may not be relevant for analysis.
data.drop(columns="FIRST NAME", axis=1, inplace=True)
data.drop(columns="LAST NAME", axis=1, inplace=True)
```

```
# Drop rows containing any missing values from the DataFrame 'data' as they are few and are not expected to significantly affect
data.dropna(inplace=True)
```

```
# Convert the 'DOJ' and 'CURRENT DATE' columns from strings to datetime format using the specified format '%m/%d/%Y'.
data['DOJ'] = pd.to_datetime(data['DOJ'], format='%m/%d/%Y')
data['CURRENT DATE'] = pd.to_datetime(data['CURRENT DATE'], format='%m/%d/%Y')
```



```python
# Drop the "CURRENT DATE", "DOJ", "TENURE", "DOJ_YEAR", "TENURE_BUCKET", and "DOJ_MONTH" columns from the DataFrame 'data'.
data.drop(columns=["CURRENT DATE", "DOJ", "TENURE", "DOJ_YEAR", "TENURE_BUCKET", "DOJ_MONTH"], inplace=True)
```

```python
# Encode categorical columns.
categorical_columns = data.select_dtypes(include=['object']).columns
label_encoder = LabelEncoder()
for col in categorical_columns:
    data[col] = label_encoder.fit_transform(data[col])
```

```python
# Separate the target variable ('SALARY') and features
X = data.drop('SALARY', axis=1)
y = data['SALARY']
```

```python
# Split the dataset into training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

MACHINE LEARNING MODEL

01

- Train various machine learning regression models to predict salaries.
- Experiment with different algorithms such as linear regression, decision trees, random forests, and gradient boosting to identify the best-performing model.
- Evaluated model performance using R^2 scores

```
# Define the models to be used for regression
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(),
    'Gradient Boosting': GradientBoostingRegressor()
}
```

```python
# Train and evaluate each model
for name, model in models.items():
    model.fit(X_train, y_train)
    train_r2 = model.score(X_train, y_train)
    test_r2 = model.score(X_test, y_test)
    print(f"{name} -> Training R^2 score = {train_r2}")
    print(f"{name} -> Test R^2 score = {test_r2}")
```

```
Linear Regression -> Training R^2 score = 0.7826100498905039
Linear Regression -> Test R^2 score = 0.7892456017128349
Decision Tree -> Training R^2 score = 0.9991475121428263
Decision Tree -> Test R^2 score = 0.9380759126835864
Random Forest -> Training R^2 score = 0.9923339938729024
Random Forest -> Test R^2 score = 0.9711024143042276
Gradient Boosting -> Training R^2 score = 0.978272663555823
Gradient Boosting -> Test R^2 score = 0.9703771162978515
```

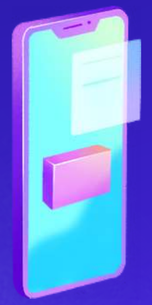
- Random Forest performed the best with the highest R^2 score, indicating it captured the variance in salary well.

MODEL EVALUATION

01

- Assess the performance of the models using appropriate evaluation metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2) score.
- Identify the model that provides the most accurate salary predictions.
- Selected Random Forest as the best model
- Visualization of predicted vs actual salaries

```
# Evaluate the best model (replace `best_model` with the actual best model from above)
best_model = RandomForestRegressor()
best_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = best_model.predict(X_test)
```



```python
# Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error = {mae}")
print(f"Mean Squared Error = {mse}")
print(f"Root Mean Squared Error = {rmse}")
print(f"R^2 Score = {r2}")
```

```
Mean Absolute Error = 3958.5752857649413
Mean Squared Error = 47110176.21020329
Root Mean Squared Error = 6863.685322784203
R^2 Score = 0.971455049766441
```

```python
# Create a DataFrame for Plotly
predicted_data = pd.DataFrame({
    'Actual_Salary': y_test,
    'Predicted_Salary': y_pred,
    'Error': y_test - y_pred
})

# Plot using Plotly Express
fig = px.scatter(
    predicted_data,
    x='Actual_Salary',
    y='Predicted_Salary',
    color='Error',
    opacity=0.8,
    title='Predicted Vs. Actual',
    template='plotly_dark',
    trendline='ols'
)

fig.update_layout(
    title={
        'font': {
            'size': 28,
            'family': 'tahoma'
        }
    }
)

fig.show()
```

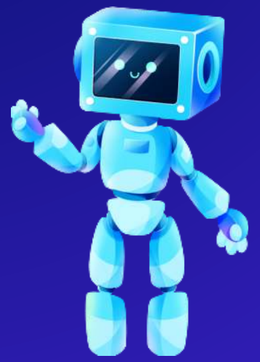
- Points close to the line of perfect prediction (where Actual Salary = Predicted Salary) indicate high model accuracy. Error Distribution: Coloring points by prediction error helps identify overestimation or underestimation. Identifying specific areas where the model performs poorly can guide further model improvement.

ML PIPELINE

01

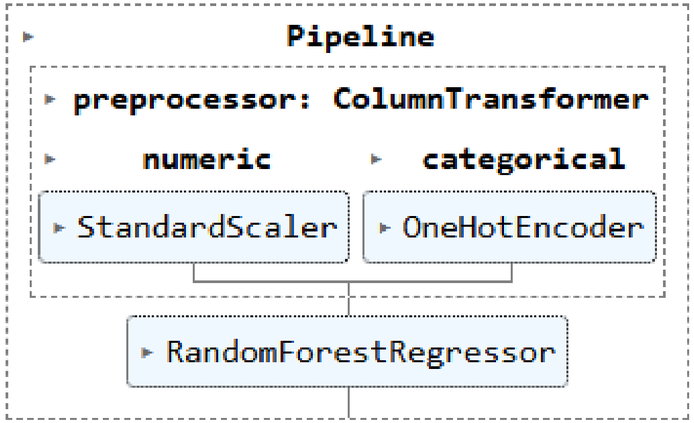
- Create ML Pipelines to streamline the end-to-end machine learning process, from data preprocessing to model training.
- Deploy a model that can generate predictions for unseen data.
- Evaluated the pipeline performance.

```
# Create a pipeline for preprocessing and modeling
numeric_features = data.select_dtypes(include=['int64', 'float64']).drop('SALARY', axis=1).columns
categorical_features = data.select_dtypes(include=['object']).columns

preprocessor = ColumnTransformer(
    transformers=[
        ('numeric', StandardScaler(), numeric_features),
        ('categorical', OneHotEncoder(), categorical_features)
    ])

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor())
])
```



```python
# Split the data
X = data.drop('SALARY', axis=1)
y = data['SALARY']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Train the pipeline
pipeline.fit(X_train, y_train)
```




```python
# Evaluate the pipeline
train_r2 = pipeline.score(X_train, y_train)
test_r2 = pipeline.score(X_test, y_test)
print(f"Pipeline -> Training R^2 score = {train_r2}")
print(f"Pipeline -> Test R^2 score = {test_r2}")
```

```
Pipeline -> Training R^2 score = 0.9808893559329463
Pipeline -> Test R^2 score = 0.9188666732237006
```

RECOMMENDATIONS

- Experience: More years of experience generally lead to higher salaries.
- Tenure: Longer tenure within a company can also positively impact salary.

- Skill Enhancement: Invest in learning new skills and obtaining certifications relevant to high-demand roles in data professions.
- Career Planning: Plan career paths that align with roles and industries offering higher salary prospects.

THANK YOU