Employee Attrition Prediction and Analysis Documentation

Team:

- **Amr Mohamed Hassan Osman**
- **Mohamed Tamer Mohamed Elsaid**
- **Reham Hassan AbdEljalil**
- **Hussam Ahmed Hussien**

Project Overview

This project aims to predict employee attrition (whether employees are likely to leave the company) using machine learning. By analyzing various factors such as job roles, income, tenure, and more, the project helps organizations identify at-risk employees and implement retention strategies.

Dataset

The IBM HR Analytics Employee Attrition & Performance dataset contains 1470 entries with 35 columns. It includes employee demographics, job-related factors, and performance indicators.

Libraries and Tools

- **Pandas**: For data manipulation and analysis.
- **NumPy**: For numerical computations.
- **Matplotlib/Seaborn/Plotly**: For data visualization.
- **Scikit-learn**: For machine learning models and utilities.
- **SMOTE**: For handling class imbalance.
    - Balancing the Dataset: SMOTE synthesizes new samples for the minority class by interpolating between existing samples. This increases the number of instances in the minority class, making the dataset more balanced.
    - Reduced Bias: Without addressing class imbalance, models tend to be biased toward the majority class. SMOTE helps reduce this bias, resulting in more reliable predictions for both classes.
- **XGBoost**: For gradient boosting model implementation.

Code Walkthrough

Data Loading and Initial Exploration

```
data = pd.read_csv('/content/WA_Fn-UseC_-HR-Employee-Attrition.csv')
data.head()
print("Dataset Info:")
print(data.info())
print("\nDataset Description:")
print(data.describe().T)
```

- Purpose: Load the dataset and display basic information to understand its structure and contents.

Data cleaning

```
data = data.drop(['EmployeeCount','StandardHours','Over18','EmployeeNumber'],
axis=1)
```

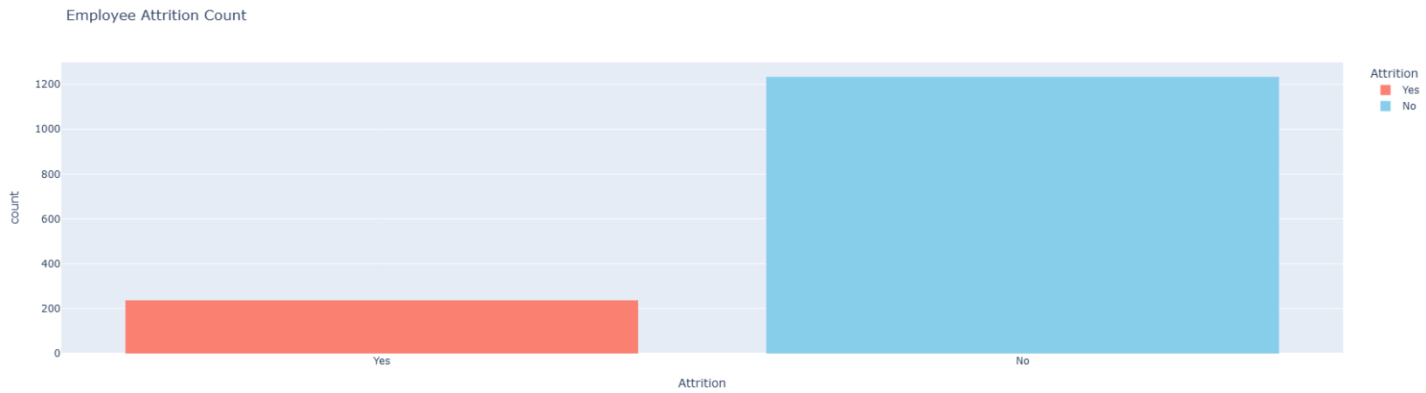
- **Purpose**: Remove unnecessary columns that do not contribute to the analysis or model.

Data Visualization

1. Employee Attrition Count

```
fig = px.histogram(data, x="Attrition", color="Attrition",
                   title="Employee Attrition Count",
                   color_discrete_sequence=['salmon', 'skyblue'])
fig.show()
```

- **Purpose**: Visualize the distribution of attrition (employees who stayed vs. left).

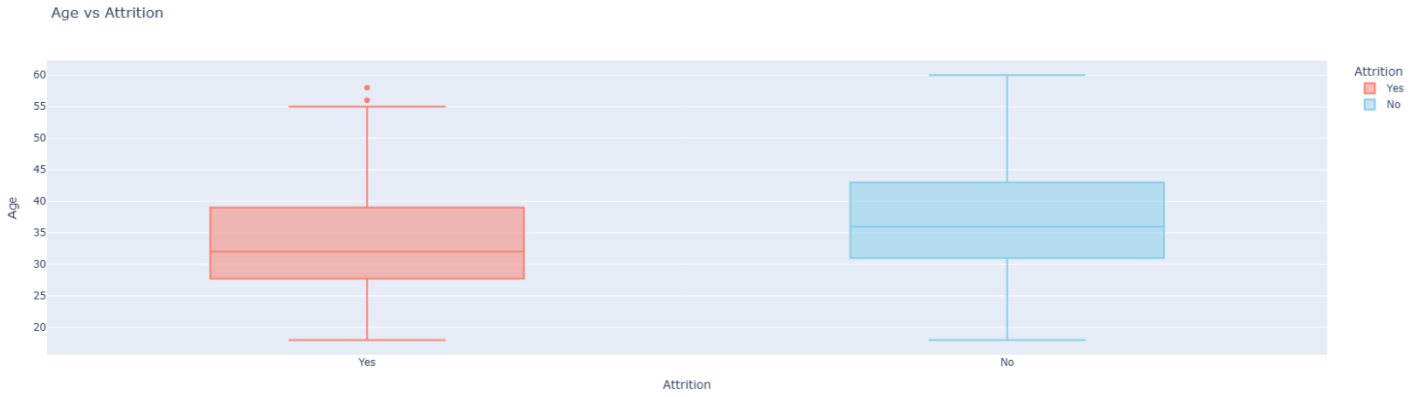


2. Numerical Features vs Attrition

```
num_cols = ['Age', 'DistanceFromHome', 'MonthlyIncome', 'NumCompaniesWorked',
            'TotalWorkingYears', 'YearsAtCompany', 'YearsInCurrentRole',
            'YearsSinceLastPromotion', 'YearsWithCurrManager']

for col in num_cols:
    fig = px.box(data, x='Attrition', y=col, color='Attrition',
                 title=f"{col} vs Attrition",
                 color_discrete_sequence=['salmon', 'skyblue'])
    fig.show()
```

- **Purpose**: Compare numerical features (e.g., age, income) between employees who stayed and those who left.

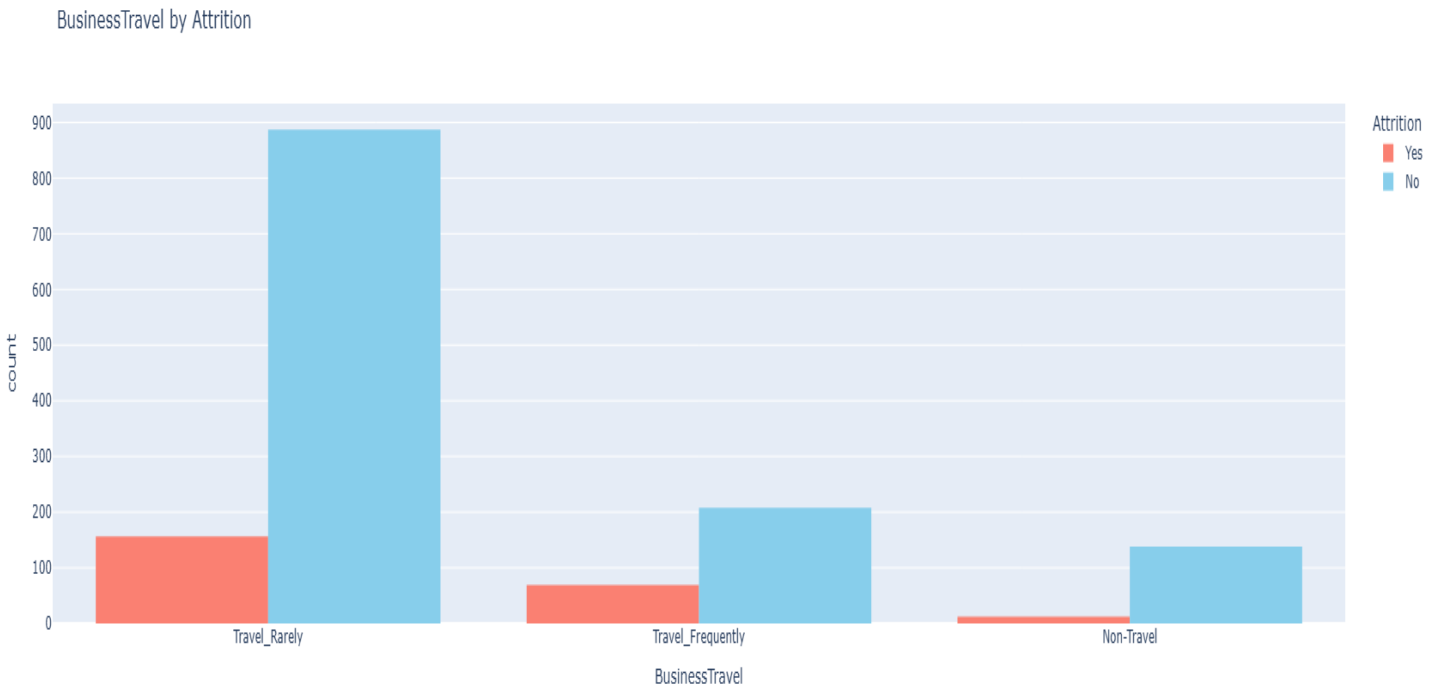


3. Categorical Features by Attrition

```
cat_cols = ['BusinessTravel', 'Department', 'EducationField', 'Gender',
            'JobRole', 'MaritalStatus', 'OverTime']

for col in cat_cols:
    fig = px.histogram(data, x=col, color='Attrition', barmode='group',
                       title=f"{col} by Attrition",
                       color_discrete_sequence=['salmon', 'skyblue'])
    fig.update_layout(xaxis={'categoryorder':'total descending'})
    fig.show()
```

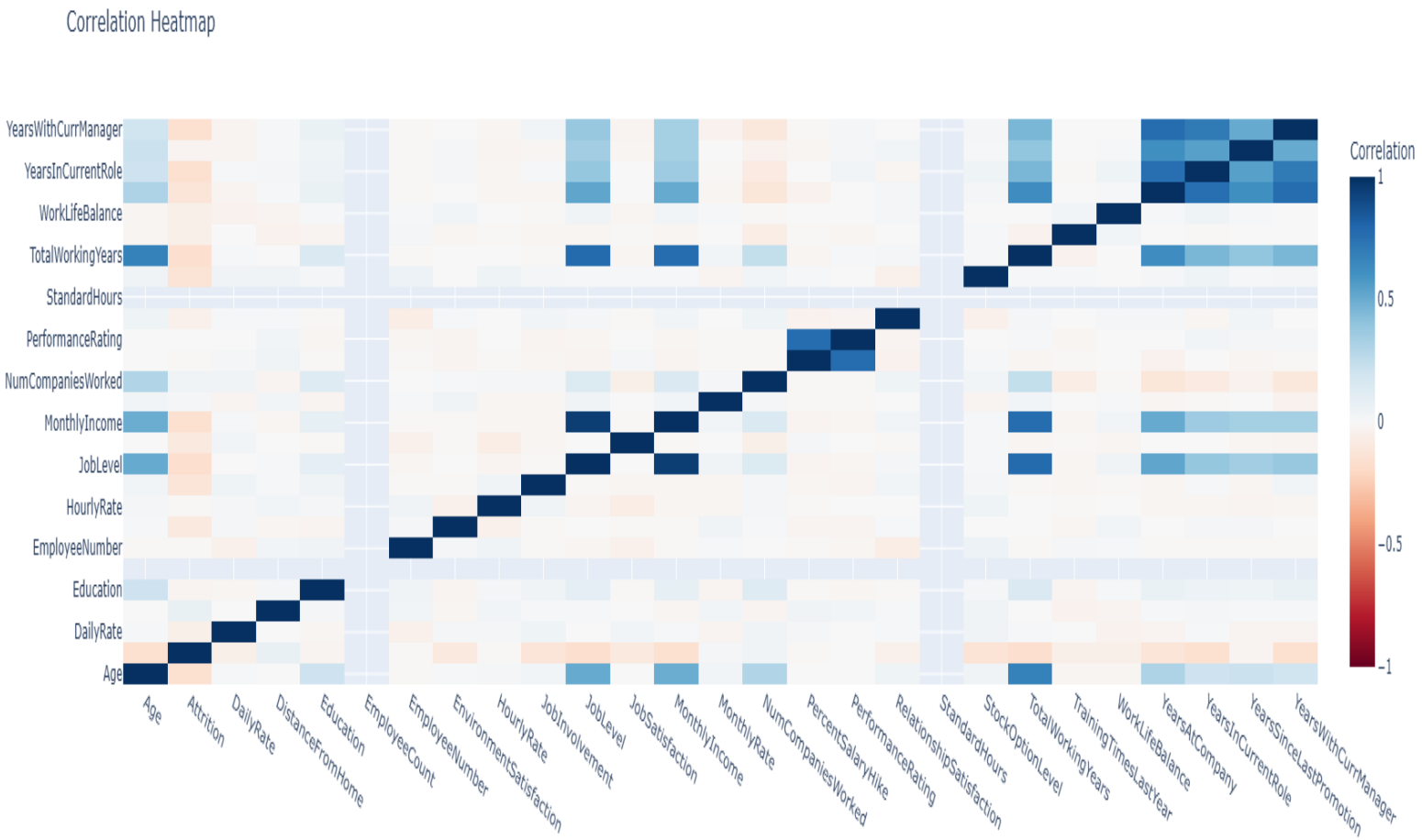
- **Purpose**: Analyze how categorical features (e.g., department, job role) relate to attrition.



4. Correlation Heatmap

```
df_corr = data.copy()
df_corr['Attrition'] = df_corr['Attrition'].map({'Yes': 1, 'No': 0})
corr = df_corr.corr(numeric_only=True).round(2)
fig = go.Figure(data=go.Heatmap(
        z=corr.values,
        x=corr.columns,
        y=corr.columns,
        colorscale='RdBu',
        zmin=-1, zmax=1,
        colorbar=dict(title="Correlation")
    ))
fig.update_layout(title='Correlation Heatmap')
fig.show()
```

- **Purpose:** Identify correlations between different features and attrition.



5. Monthly Income vs Total Working Years by Attrition

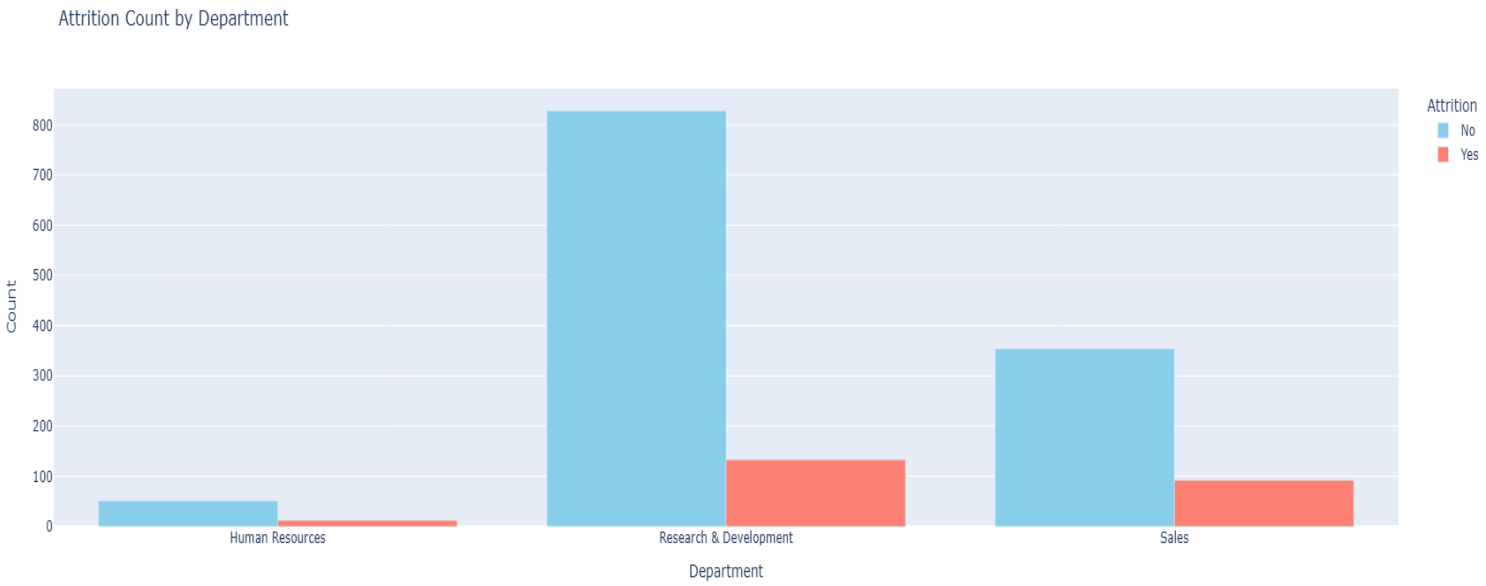
```
fig = px.scatter(data, x='TotalWorkingYears', y='MonthlyIncome',
                 color='Attrition',
                 title='Monthly Income vs Total Working Years by Attrition',
                 size='Age',
                 color_discrete_sequence=['red', 'skyblue'],
                 hover_data=['JobRole', 'JobLevel'])
fig.show()
```

- **Purpose:** Explore the relationship between income, tenure, and attrition.

6. Attrition Count by Department

```
dept_attr = data.groupby(['Department',
'Attrition']).size().reset_index(name='Count')
fig = px.bar(dept_attr, x='Department', y='Count', color='Attrition',
             barmode='group',
             title='Attrition Count by Department',
             color_discrete_sequence=['skyblue', 'salmon'])
fig.show()
```

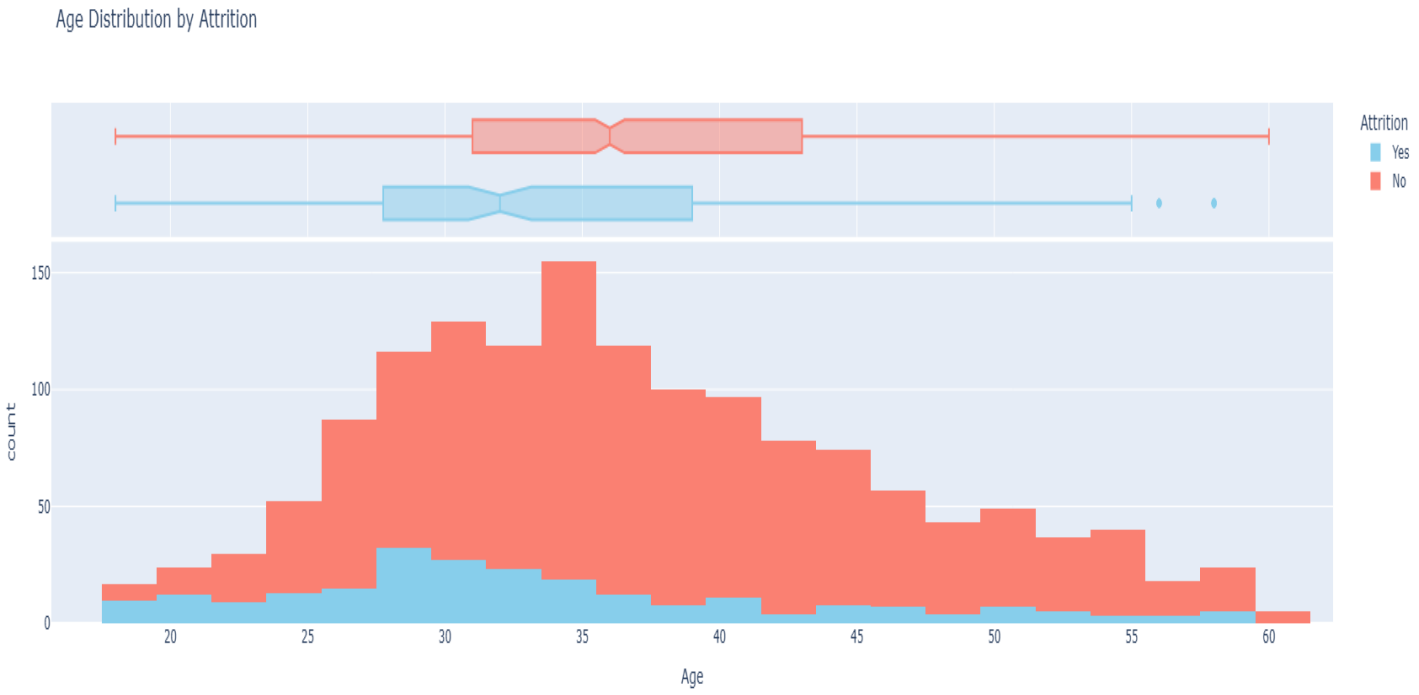
- **Purpose:** Compare attrition rates across different departments.



7. Age Distribution by Attrition

```
fig = px.histogram(data, x="Age", color="Attrition", nbins=30,
                   marginal="box",
                   title="Age Distribution by Attrition",
                   color_discrete_sequence=['skyblue', 'salmon'])
fig.show()
```

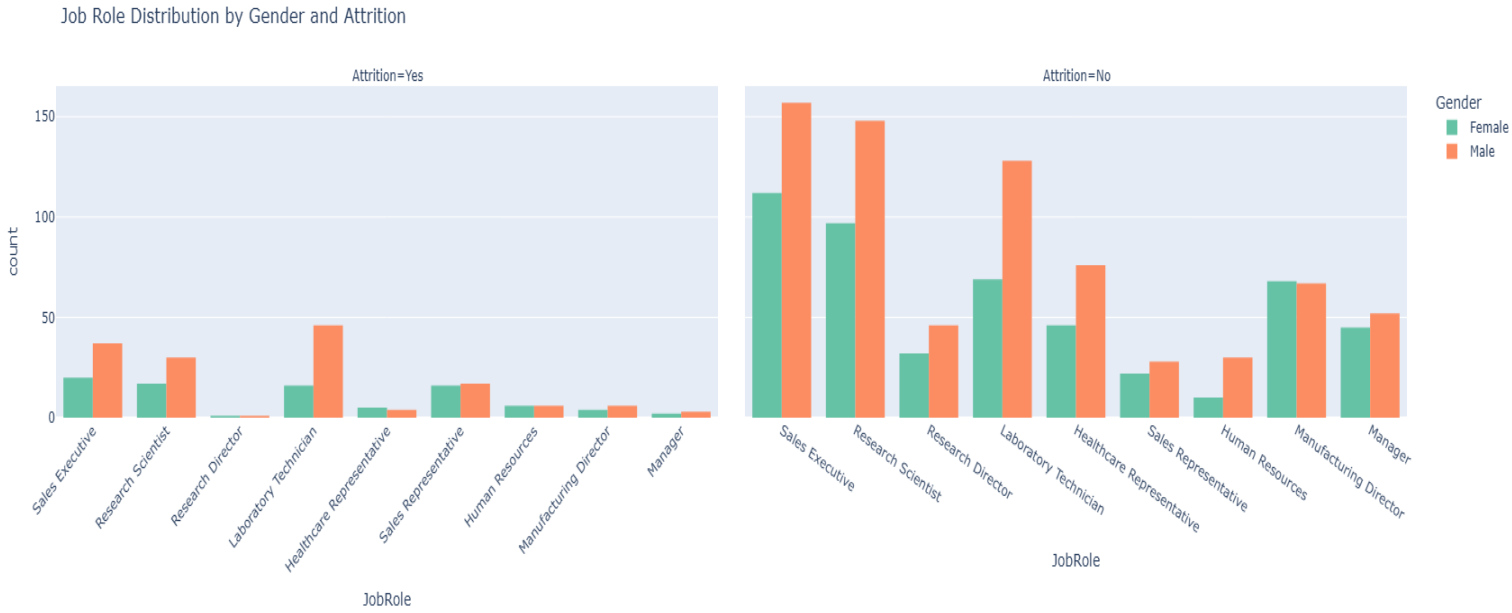
- **Purpose**: Analyze how age distribution differs between employees who stayed and those who left.



8. Job Role Distribution by Gender and Attrition

```
fig = px.histogram(data, x="JobRole", color="Gender", barmode='group',
                   facet_col="Attrition",
                   title="Job Role Distribution by Gender and Attrition",
                   color_discrete_sequence=px.colors.qualitative.Set2)
fig.update_layout(xaxis_tickangle=-45)
fig.show()
```

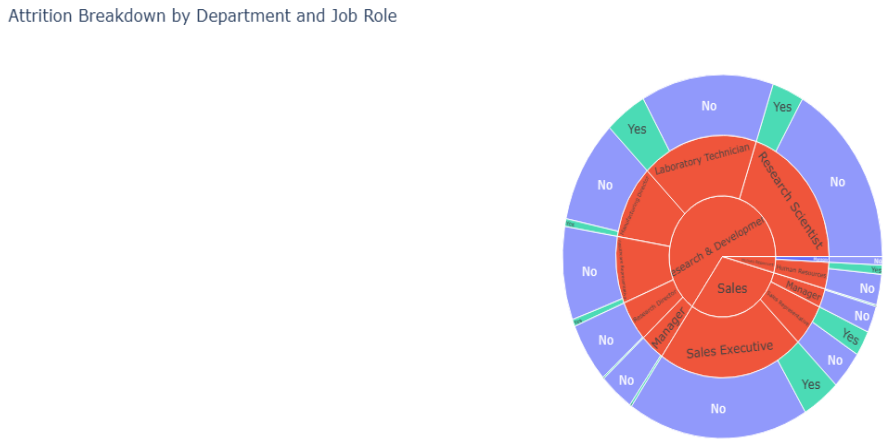
- **Purpose**: Examine job roles and gender in relation to attrition.



9. Attrition Breakdown by Department and Job Role

```
df = data.copy()
color_map = {'Yes': 'lightcoral', 'No': 'skyblue'}
df['AttritionColor'] = df['Attrition'].map(color_map)
fig = px.sunburst(df, path=['Department', 'JobRole', 'Attrition'],
                  values=None,
                  color='AttritionColor',
                  title='Attrition Breakdown by Department and Job Role')
fig.show()
```

- **Purpose**: Provide a hierarchical view of attrition by department and job role.



Feature Engineering

```
data['YearsAtCompany_to_Age'] = data['YearsAtCompany'] / data['Age']
data['JobRole_Stability'] = data['YearsInCurrentRole'] / (data['TotalWorkingYears']
+ 1)
data['PromotionRate'] = data['YearsSinceLastPromotion'] / (data['YearsAtCompany'] +
1)
```

- **Purpose:** Create new features to provide additional insights for the model.

Data Preprocessing

```
data["Attrition"] = data["Attrition"].map({"Yes": 1, "No":0})

data = pd.get_dummies(data, drop_first=True)
```

- **Purpose:** Convert categorical variables to numerical values using one-hot encoding.

Model Training and Evaluation

```
X = data.drop(columns='Attrition')
y = data['Attrition']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2,
random_state=42, stratify=y)
```

- **Purpose:** Split the dataset into training and testing sets to evaluate model performance.

Stratified K-Fold Cross-Validation with SMOTE

```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

accuracy_scores = []
f1_scores = []

for train_idx, val_idx in skf.split(X_train, y_train):
    X_tr, X_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
    y_tr, y_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

    smote = SMOTE(random_state=42)
    X_tr_resampled, y_tr_resampled = smote.fit_resample(X_tr, y_tr)

    scale_pos_weight = len(y_tr_resampled[y_tr_resampled == 0]) /
len(y_tr_resampled[y_tr_resampled == 1]) * 1.5

    xgb_model = XGBClassifier(
        n_estimators=100,
        max_depth=5,
        learning_rate=0.1,
        scale_pos_weight=scale_pos_weight,
        use_label_encoder=False,
        eval_metric='logloss',
        random_state=42
    )
    xgb_model.fit(X_tr_resampled, y_tr_resampled)

    y_pred = xgb_model.predict(X_val)
    accuracy_scores.append(accuracy_score(y_val, y_pred))
    f1_scores.append(f1_score(y_val, y_pred))

print("Average Accuracy:", np.mean(accuracy_scores))
print("Average F1 Score:", np.mean(f1_scores))
```

- **Purpose:** Train an XGBoost model using cross-validation and SMOTE to handle class imbalance. Evaluate the model using accuracy and F1 score.
- **XGBoost**: Chosen for its efficiency and effectiveness in handling tabular data, especially with class imbalance. It builds an ensemble of decision trees to improve prediction accuracy.

Final Model Evaluation

```
y_pred = xgb_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

- **Purpose**: Evaluate the final model on the test set using accuracy, confusion matrix, and classification report.