Employee Attrition

Step 1: Data Exploration

1. Read the dataset and calculate the number of rows and columns.
2. Check for duplicate lines in the data.
3. Calculate the percentage of missing values for each column.
4. Generate descriptive statistics for each column.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore') # ignore warnings.
```

```
df=pd.read_csv("/HR_Employee_Attrition_Data1.csv")
df.shape
```

```
(1470, 35)
```

```
pd.set_option('display.max_columns', None)
df.head(5)
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumbe
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	

```
df.columns
```

```
Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
```

```
 9   EmployeeNumber            1470 non-null   int64 
 10  EnvironmentSatisfaction   1470 non-null   int64 
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64 
 13  JobInvolvement            1470 non-null   int64 
 14  JobLevel                  1470 non-null   int64 
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64 
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64 
 19  MonthlyRate               1470 non-null   int64 
 20  NumCompaniesWorked        1470 non-null   int64 
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64 
 24  PerformanceRating         1470 non-null   int64 
 25  RelationshipSatisfaction  1470 non-null   int64 
 26  StandardHours             1470 non-null   int64 
 27  StockOptionLevel          1470 non-null   int64 
 28  TotalWorkingYears         1470 non-null   int64 
 29  TrainingTimesLastYear     1470 non-null   int64 
 30  WorkLifeBalance           1470 non-null   int64 
 31  YearsAtCompany            1470 non-null   int64 
 32  YearsInCurrentRole        1470 non-null   int64 
 33  YearsSinceLastPromotion   1470 non-null   int64 
 34  YearsWithCurrManager      1470 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

```
df.duplicated().sum()
```

```
0
```

```
# Calculate the percentage of missing values for each column
missing_percentages = (df.isnull().sum() / len(df)) * 100

# Create a DataFrame to display missing percentages
missing_data = pd.DataFrame({'Column': missing_percentages.index, 'MissingPercentage': missing_percentages.values})

# Sort the DataFrame by missing percentage in descending order
missing_data.sort_values(by='MissingPercentage', ascending=False)
```

	Column	MissingPercentage
0	Age	0.0
26	StandardHours	0.0
20	NumCompaniesWorked	0.0
21	Over18	0.0
22	OverTime	0.0
23	PercentSalaryHike	0.0
24	PerformanceRating	0.0
25	RelationshipSatisfaction	0.0
27	StockOptionLevel	0.0
18	MonthlyIncome	0.0
28	TotalWorkingYears	0.0
29	TrainingTimesLastYear	0.0
30	WorkLifeBalance	0.0
31	YearsAtCompany	0.0
32	YearsInCurrentRole	0.0
33	YearsSinceLastPromotion	0.0
19	MonthlyRate	0.0
17	MaritalStatus	0.0
1	Attrition	0.0
8	EmployeeCount	0.0
2	BusinessTravel	0.0
3	DailyRate	0.0
4	Department	0.0
5	DistanceFromHome	0.0
6	Education	0.0
7	EducationField	0.0
9	EmployeeNumber	0.0
16	JobSatisfaction	0.0
10	EnvironmentSatisfaction	0.0
11	Gender	0.0
12	HourlyRate	0.0
13	JobInvolvement	0.0
14	JobLevel	0.0
15	JobRole	0.0
34	YearsWithCurrManager	0.0

```
df.nunique()
```

	0
Age	43
Attrition	2
BusinessTravel	3
DailyRate	886
Department	3
DistanceFromHome	29
Education	5
EducationField	6
EmployeeCount	1
EmployeeNumber	1470
EnvironmentSatisfaction	4
Gender	2
HourlyRate	71
JobInvolvement	4
JobLevel	5
JobRole	9
JobSatisfaction	4
MaritalStatus	3
MonthlyIncome	1349
MonthlyRate	1427
NumCompaniesWorked	10
Over18	1
OverTime	2
PercentSalaryHike	15
PerformanceRating	2
RelationshipSatisfaction	4
StandardHours	1
StockOptionLevel	4
TotalWorkingYears	40
TrainingTimesLastYear	7
WorkLifeBalance	4
YearsAtCompany	37
YearsInCurrentRole	19
YearsSinceLastPromotion	16
YearsWithCurrManager	18

Exclude features with only one value for `EmployeeCount`, `Over18`, and `StandardHours`.

```
df.drop(columns=['EmployeeCount', 'Over18', 'StandardHours'], inplace=True)
df.shape
```

```
(1470, 32)
```

```
df.describe()
```

	Age	DailyRate	DistanceFromHome	Education	EmployeeNumber	EnvironmentSatisfaction	HourlyRate	JobInvolvement
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000
mean	36.923810	802.485714	9.192517	2.912925	1024.865306	2.721769	65.891156	2.729932
std	9.135373	403.509100	8.106864	1.024165	602.024335	1.093082	20.329428	0.711561
min	18.000000	102.000000	1.000000	1.000000	1.000000	1.000000	30.000000	1.000000
25%	30.000000	465.000000	2.000000	2.000000	491.250000	2.000000	48.000000	2.000000
50%	36.000000	802.000000	7.000000	3.000000	1020.500000	3.000000	66.000000	3.000000
75%	43.000000	1157.000000	14.000000	4.000000	1555.750000	4.000000	83.750000	3.000000
max	60.000000	1499.000000	29.000000	5.000000	2068.000000	4.000000	100.000000	4.000000

```
df1=df.copy()
```

```
# Unencoding Categorical Features
col = ['EnvironmentSatisfaction','JobInvolvement','JobSatisfaction','RelationshipSatisfaction']

for i in df['Education']:
    df['Education'].replace({1:'Below College',2:'College',3:'Bachelor',4:'Master', 5:'Doctor'},
                   inplace = True)

for i in df['PerformanceRating']:
    df['PerformanceRating'].replace({1:'Low', 2:'Good',3:'Excellent',4:'Outstanding'},
                                inplace = True)

for i in df['WorkLifeBalance']:
    df['WorkLifeBalance'].replace({1: 'Bad', 2:'Good', 3:'Better', 4:'Best'},
                                inplace = True)

for i in df[col]:
    df[i].replace({1:'Low', 2:'Medium',3:'High', 4:'Very High'},
               inplace = True)
# Checking new values for decoded attributes
df.head(10)
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeNumber	EnvironmentS
0	41	Yes	Travel_Rarely	1102	Sales	1	College	Life Sciences	1	
1	49	No	Travel_Frequently	279	Research & Development	8	Below College	Life Sciences	2	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	College	Other	4	
3	33	No	Travel_Frequently	1392	Research & Development	3	Master	Life Sciences	5	
4	27	No	Travel_Rarely	591	Research & Development	2	Below College	Medical	7	
5	32	No	Travel_Frequently	1005	Research & Development	2	College	Life Sciences	8	
6	59	No	Travel_Rarely	1324	Research & Development	3	Bachelor	Medical	10	
7	30	No	Travel_Rarely	1358	Research & Development	24	Below College	Life Sciences	11	
8	38	No	Travel_Frequently	216	Research & Development	23	Bachelor	Life Sciences	12	
9	36	No	Travel_Rarely	1299	Research & Development	27	Bachelor	Medical	13	

Step 2: Data Analysis and Visualization

Now, let's address each business question and perform the necessary analysis for each one.I will be creating visualizations and interpreting them to answer these questions.

1) Attrition Analysis & Demographic Information about Employees:

2) Employee Satisfaction and Work Environment:

3) Performance and Growth Opportunities

4) Additional Factors and Analysis:

How many employees left the company?

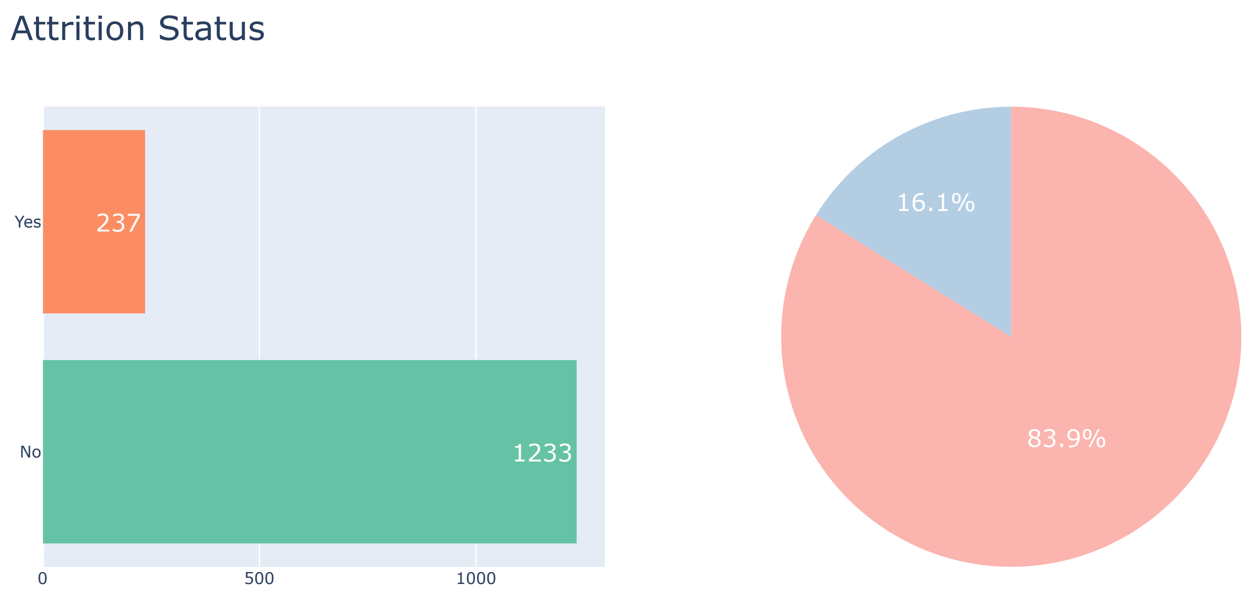
```
# Calculate attrition counts
attrition_counts = df['Attrition'].value_counts()
print(attrition_counts)
```

```
Attrition
No     1233
Yes     237
Name: count, dtype: int64
```

Attrition Rate:

What is the overall Attrition Rate in the company?

```
# Create subplots with a bar chart and a pie chart side by side
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'bar'}, {'type':'pie'}]])

# Add horizontal bar chart for Attrition
fig.add_trace(
    go.Bar(y=attrition_counts.index, x=attrition_counts.values, orientation='h',
           marker=dict(color=px.colors.qualitative.Set2), showlegend=False,
           text=attrition_counts.values, textposition='auto', textfont=dict(size=18, color='white')),
    row=1, col=1
)

# Add pie chart for Attrition
fig.add_trace(
    go.Pie(labels=attrition_counts.index, values=attrition_counts.values,
           marker=dict(colors=px.colors.qualitative.Pastel1),
           textfont=dict(size=18, color='white')),
    row=1, col=2
)

# Update layout
fig.update_layout(
    title_text="Attrition Status",
    title_font=dict(size=25)
)

# Show the figure
fig.show()
```



Around `16.12%` of the workforce has left the company, emphasizing the need to analyze and enhance retention strategies

How many employees of each gender are currently in the company?

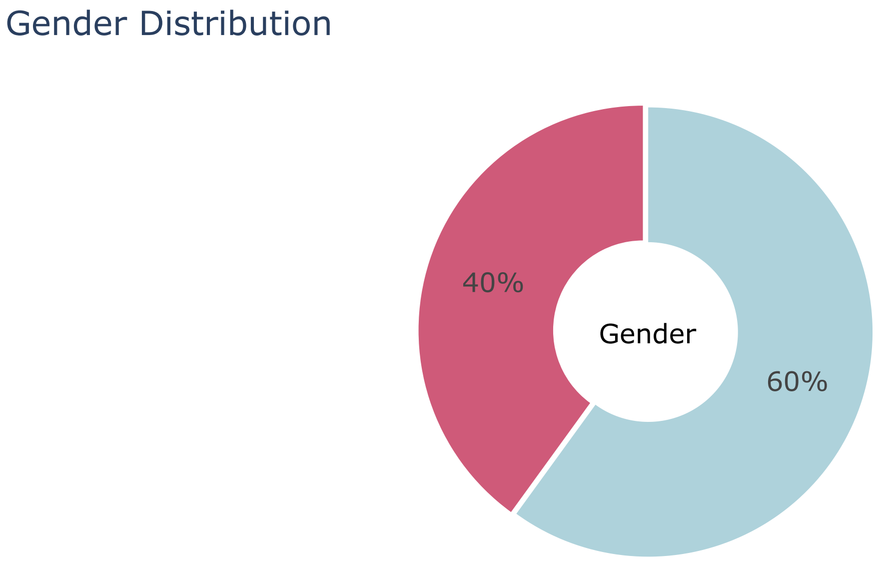
```
gender_counts = df['Gender'].value_counts()
print(gender_counts)
```

```
Gender
Male      882
Female    588
Name: count, dtype: int64
```

The dataset displays a gender distribution imbalance, with 882 males and 588 females, which could impact analysis, decision-making, and representation considerations within the dataset's context.

What is the gender ratio among our employees ?

```
# Values and labels for the pie chart
values = gender_counts.values
labels = gender_counts.index

# Custom colors
colors = ['#AED2DB', '#CF5A79']

# Create the pie chart
fig7 = go.Figure(data=go.Pie(values=values,
                             labels=labels, hole=0.4,
                             pull=[0, 0.025],
                             marker_colors=colors))

# Update the hover and text info
fig7.update_traces(hoverinfo='label+percent',
                   textinfo='percent', textfont_size=20)

# Add annotation for the year
fig7.add_annotation(x=0.5, y=0.5,
                    text='Gender',
                    font=dict(size=20, family='Verdana', color='black'),
                    showarrow=False)

# Update the layout
fig7.update_layout(title_text='Gender Distribution',
                   title_font=dict(size=25, family='Verdana'))

# Show the pie chart
fig7.show()
```



What percentage of employees who left the company are male, and what percentage are female?

How Is Attrition Affected by Gender?

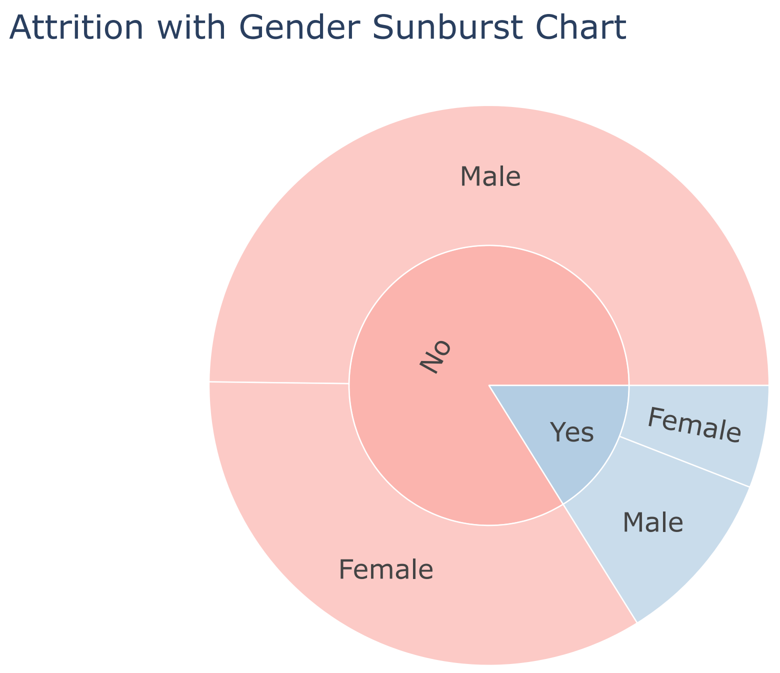
```
def calculate_percentage_cross_tab(df, x):
    # Create the cross-tabulation
    cross_tab = pd.crosstab(df[x], df['Attrition'])

    # Convert counts to percentages
    percentage_cross_tab = cross_tab.apply(lambda row: row / row.sum() * 100, axis=1)

    # Round the percentages to two decimal places
    rounded_percentage_cross_tab = percentage_cross_tab.round(2)

    return rounded_percentage_cross_tab
```

```
calculate_percentage_cross_tab(df, 'Gender')
```

Attrition	No	Yes
Gender		
Female	85.20	14.80
Male	82.99	17.01

```
# Create a DataFrame with churn and gender counts
churn_gender_counts = df.groupby(['Attrition', 'Gender']).size().reset_index(name='Count')

# Calculate percentages
churn_gender_counts['Percentage'] = churn_gender_counts['Count'] / churn_gender_counts['Count'].sum() * 100

# Create a sunburst chart using Plotly
fig = px.sunburst(churn_gender_counts,
                  path=['Attrition', 'Gender'],
                  values='Count',
                  title="Attrition with Gender Sunburst Chart",
                  color='Attrition',
                  color_discrete_sequence=px.colors.qualitative.Pastel1,
                  labels={'Attrition': 'Status'},
                  custom_data=['Percentage'])

fig.update_layout(title_font=dict(size=25, family='Verdana'),
                  width=800, height=600)

fig.update_traces(hovertemplate='<b>%{label}</b><br>Count: %{value}<br>Percentage: %{customdata[0]:.2f}%',
                  textfont=dict(size=20, family='Verdana'))
```

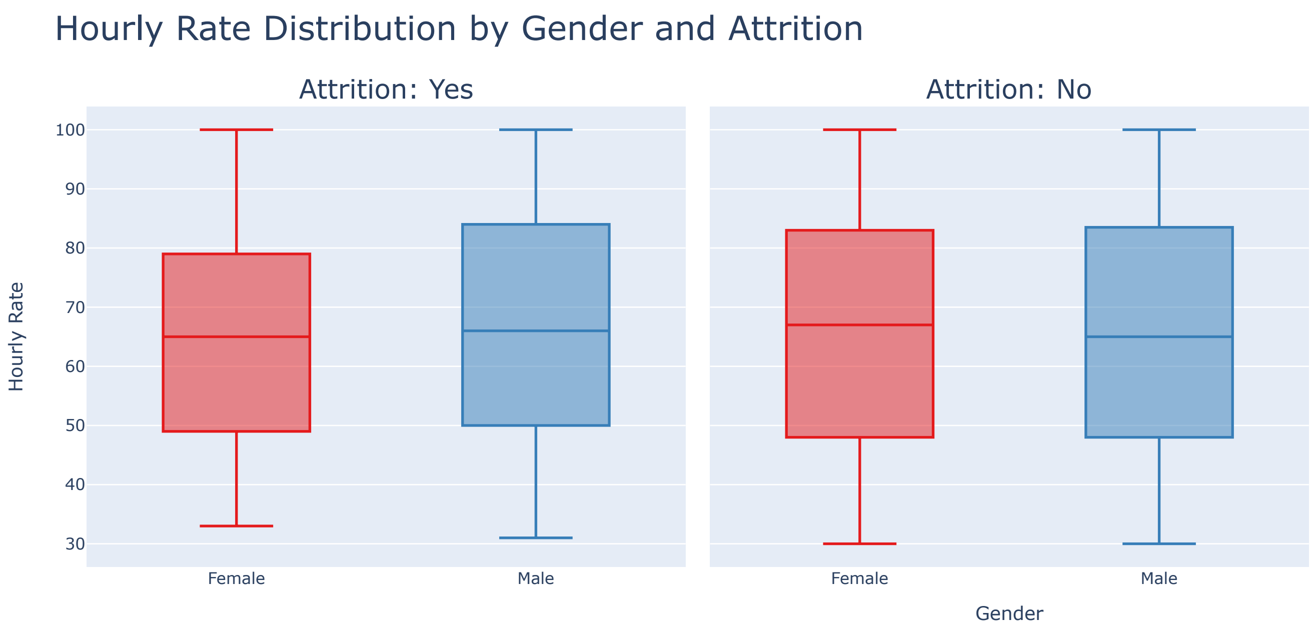
```
fig.show()
```



The attrition rates differ slightly based on gender among the employees. The data shows that the attrition rate for female employees is approximately 14.80%, while for male employees, it's around 17.01% compared to the corresponding overall attrition rate in the dataset (16.12%). This suggests that gender might play a minor role in influencing attrition, but other factors are likely more influential in driving employee turnover.

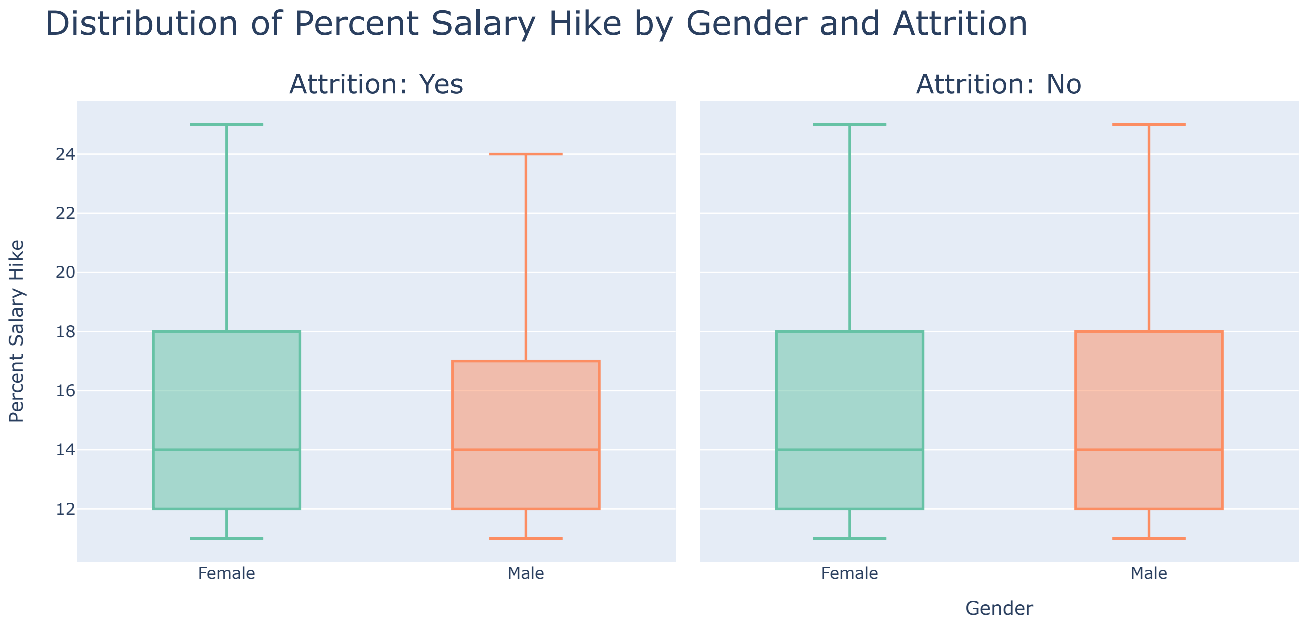
How does the hourly rate distribution vary by gender and attrition?

```
# Create a box plot using Plotly Express
fig = px.box(
    data_frame=df,
    x="Gender",
    y="HourlyRate",
    color="Gender",
    facet_col="Attrition",
    title="Hourly Rate Distribution by Gender and Attrition",
    labels={"Gender": "Gender", "HourlyRate": "Hourly Rate"},
    color_discrete_sequence=px.colors.qualitative.Set1
)

# Update facet labels
facet_col_labels = {
    "Yes": "Attrition: Yes",
    "No": "Attrition: No"
}

for i, label in enumerate(facet_col_labels.values()):
    fig.layout.annotations[i].text = label

# Update layout with font properties
fig.update_layout(
    legend_title="Gender",
    xaxis_title=None,
    yaxis_title="Hourly Rate",
    margin=dict(t=100),  # Adjust top margin to accommodate facet labels
    title_font=dict(size=25, family='Verdana'),
)

# Update font properties of annotations
for annotation in fig.layout.annotations:
    annotation.font = dict(size=20, family='Verdana')

# Show the plot
fig.show()
```



Our analysis reveals a notable pattern: when attrition is marked as "Yes," men tend to receive higher compensation, whereas in cases where attrition is labeled as "No," male employees receive lower compensation. This data-driven insight underscores that among the individuals who have left our company, females, on average, had a lower median hourly rate compared to their male counterparts.

```
# Create the box plot
fig = px.box(
    data_frame=df,
    x="Gender",
    y="PercentSalaryHike",
    color="Gender",
    facet_col="Attrition",
    title="Distribution of Percent Salary Hike by Gender and Attrition",
    labels={"Gender": "Gender", "PercentSalaryHike": "Percent Salary Hike"},
    color_discrete_sequence=px.colors.qualitative.Set2
)

# Update facet labels
facet_col_labels = {
    "Yes": "Attrition: Yes",
    "No": "Attrition: No"
}

for i, label in enumerate(facet_col_labels.values()):
    fig.layout.annotations[i].text = label

# Update layout with font properties
fig.update_layout(
    legend_title="Gender",
    xaxis_title=None,
    yaxis_title="Percent Salary Hike",
    margin=dict(t=100),  # Adjust top margin to accommodate facet labels
    title_font=dict(size=25, family='Verdana'),
)

# Update font properties of annotations
for annotation in fig.layout.annotations:
    annotation.font = dict(size=20, family='Verdana')

# Show the plot
fig.show()
```



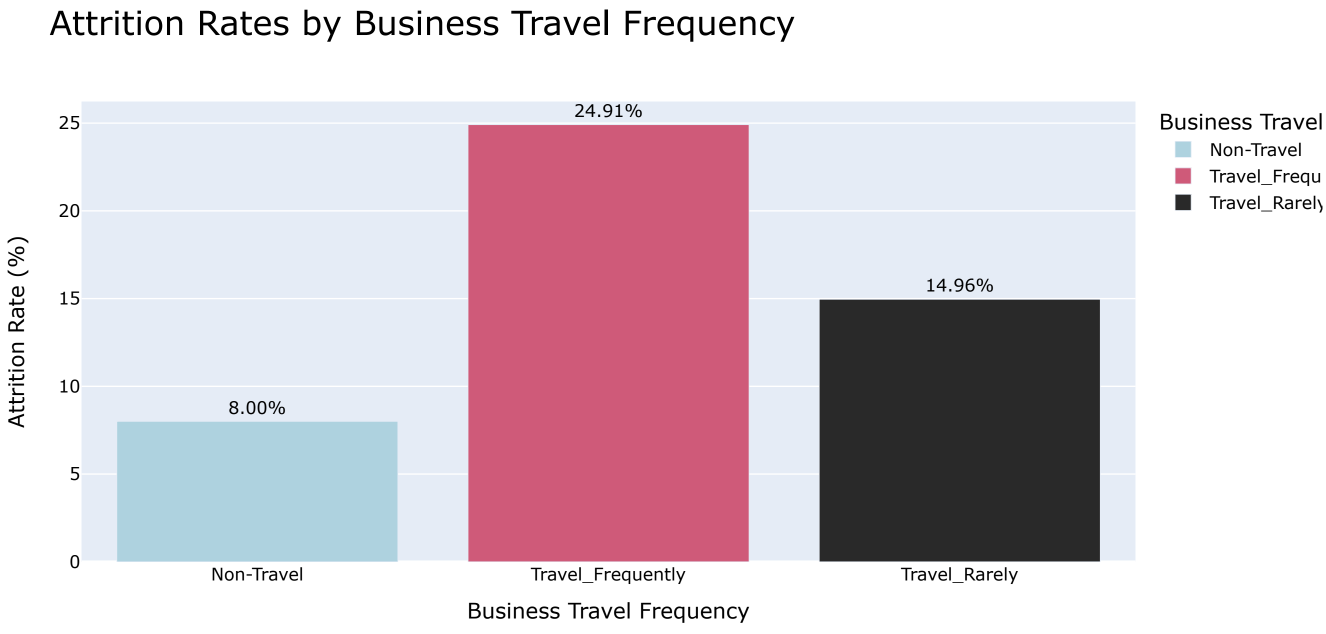
Interestingly, among those who left the company, it appears that the male employees had a lower average percent salary hike compared to their female counterparts. However, when looking at the current employees, both male and female employees have fairly similar average percent salary hikes. This suggests that there might have been a gender-based discrepancy in salary adjustments among the departed employees, but the situation has since been rectified for the existing workforce.

How is Attrition Affected by business travels?

```
calculate_percentage_cross_tab(df, 'BusinessTravel')
```

Attrition	No	Yes
BusinessTravel		
Non-Travel	92.00	8.00
Travel_Frequently	75.09	24.91
Travel_Rarely	85.04	14.96

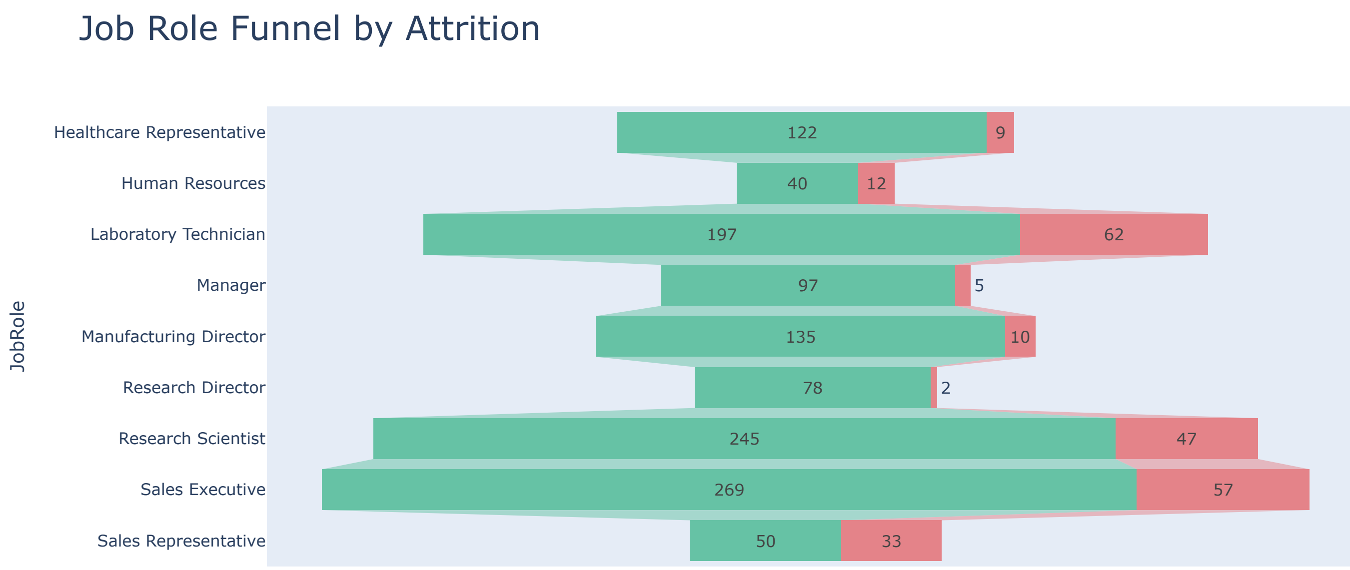
```
# Group data by BusinessTravel and Attrition, and calculate attrition rates
attrition_by_travel = df.groupby(['BusinessTravel', 'Attrition']).size().unstack()
attrition_by_travel['Attrition Rate'] = attrition_by_travel['Yes'] / (attrition_by_travel['Yes'] + attrition_by_travel['No']) * 100

# Reset index for plotting
attrition_by_travel = attrition_by_travel.reset_index()

# Create a bar plot using Plotly with specified title and font styles
fig = px.bar(attrition_by_travel, x='BusinessTravel', y='Attrition Rate',
             color='BusinessTravel', text='Attrition Rate',
             title='Attrition Rates by Business Travel Frequency',
             labels={'BusinessTravel': 'Business Travel Frequency', 'Attrition Rate': 'Attrition Rate (%)'},
             color_discrete_sequence=['#AED2DF', '#CF5A79', '#292929'])

# Add data labels to the bars
fig.update_traces(texttemplate='%{text:.2f}%', textposition='outside')

# Apply title and font styles
fig.update_layout(
    title=dict(text='Attrition Rates by Business Travel Frequency',
               font=dict(size=25, family='Verdana')),
    font=dict(size=13, family='Verdana', color='black')
)

# Show the plot
fig.show()
```



The data highlights that Non-Travel employees have the lowest attrition rate (8%), while Travel_Frequently employees face the highest rate (24.91%). This disparity indicates a potential link between frequent business travel and increased turnover. To address this, the company could introduce measures like improving work-life balance for frequent travelers, offering extra support, and addressing challenges tied to extensive travel. Retention efforts should particularly target Travel_Frequently employees.

Which job roles did the majority of those who departed the company hold?

```
# Calculate value counts for JobRole column
jobrole_attrition_counts = df.groupby(['JobRole', 'Attrition']).size().reset_index(name='Count')

# Define custom color scales for each category
colors = {'Yes': '#E48389', 'No': '#66C2A5'}

# Create funnel charts with the specified design
fig_jobrole_attrition = px.funnel(jobrole_attrition_counts,
                                  x='Count',
                                  y='JobRole',
                                  color='Attrition',
                                  color_discrete_map=colors,
                                  title='Job Role Funnel by Attrition')

# Apply design changes
fig_jobrole_attrition.update_layout(
    title_font=dict(size=25, family='Verdana'))

# Show the chart
fig_jobrole_attrition.show()
```



The majority of employees experiencing attrition belong to roles like laboratory technicians, sales executives, or research scientists. This observation highlights the significance of understanding the reasons behind attrition in these specific job roles and implementing targeted measures to improve retention.

Step 3: Data Preparation & Modeling

```
df.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeNumber	EnvironmentS
0	41	Yes	Travel_Rarely	1102	Sales	1	College	Life Sciences	1	
1	49	No	Travel_Frequently	279	Research & Development	8	Below College	Life Sciences	2	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	College	Other	4	
3	33	No	Travel_Frequently	1392	Research & Development	3	Master	Life Sciences	5	
4	27	No	Travel_Rarely	591	Research & Development	2	Below College	Medical	7	

```
df1.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeNumber	EnvironmentS
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	2	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	4	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	5	
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	7	

```
df1= df1.drop(['EmployeeNumber'], axis=1)
```

Imports and Data Preparation

```
#Transform categorical values to the binary values using the dictionary function
df1['Attrition'] = df1['Attrition'].replace({'No': 0, 'Yes': 1})
print(f"Attribute: {df1['Attrition'].unique(), df1['Attrition'].dtype}")
```

```
Attribute: (array([1, 0]), dtype('int64'))
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from imblearn.over_sampling import SMOTE

# Separate features and target
X = df1.drop("Attrition", axis=1)
y = df1["Attrition"]

# Encode categorical columns
nominal_cols = ["BusinessTravel", "Department", "EducationField", "Gender", "JobRole", "MaritalStatus", "OverTime"]

# One-hot encode nominal columns
encoder = OneHotEncoder(drop="first", sparse=False)
X_encoded = pd.concat([X.drop(nominal_cols, axis=1),
                      pd.DataFrame(encoder.fit_transform(X[nominal_cols]))], axis=1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Oversampling using SMOTE
oversampler = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = oversampler.fit_resample(X_train, y_train)
```

Show hidden output

Next steps: Explain error

Hyperparameter Tuning for Random Forest

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

# Hyperparameter tuning for Random Forest using GridSearchCV
rf_params = {
    "n_estimators": [100, 200, 300],
    "max_depth": [None, 10, 20],
    "min_samples_split": [2, 5, 10]
}
rf_grid = GridSearchCV(RandomForestClassifier(random_state=42), param_grid=rf_params, cv=5)
rf_grid.fit(X_train_resampled, y_train_resampled)
rf_best = rf_grid.best_estimator_
```

Hyperparameter Tuning for CatBoost

```
from catboost import CatBoostClassifier

# Hyperparameter tuning for CatBoost using GridSearchCV
cat_params = {
    "iterations": [100, 200, 300],
    "depth": [6, 8, 10],
    "learning_rate": [0.1, 0.2, 0.3]
}
cat_grid = GridSearchCV(CatBoostClassifier(random_state=42, verbose=0), param_grid=cat_params, cv=5)
cat_grid.fit(X_train_resampled, y_train_resampled)
cat_best = cat_grid.best_estimator_
```

Training and Evaluating SVM Classifier

```
from sklearn.svm import SVC

# Train and evaluate SVM classifier
svm_classifier = SVC(kernel='rbf', class_weight='balanced', random_state=42)
svm_classifier.fit(X_train_resampled, y_train_resampled)
```

Training and Evaluating MLP Classifier

```
from sklearn.neural_network import MLPClassifier

# Train and evaluate MLP classifier
mlp_classifier = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=500, random_state=42)
mlp_classifier.fit(X_train_resampled, y_train_resampled)
mlp_predictions = mlp_classifier.predict(X_test)
```

Training and Evaluating Logistic Regression Classifier

```
from sklearn.linear_model import LogisticRegression

# Train and evaluate Logistic Regression classifier
lr_classifier = LogisticRegression(class_weight='balanced', random_state=42)
lr_classifier.fit(X_train_resampled, y_train_resampled)
lr_predictions = lr_classifier.predict(X_test)
```

Training and Evaluating XGBoost Classifier

```
from xgboost import XGBClassifier

# Train and evaluate XGBoost classifier
xgb_classifier = XGBClassifier(random_state=42)
xgb_classifier.fit(X_train_resampled, y_train_resampled)
xgb_predictions = xgb_classifier.predict(X_test)
```

Print Classification Reports for All Classifiers

```
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier

# Print classification reports for all classifiers
classifiers = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest (Tuned)": rf_best,
    "CatBoost (Tuned)": cat_best,
    "SVM": svm_classifier,
    "MLP": mlp_classifier,
    "Logistic Regression": lr_classifier,
    "XGBoost": xgb_classifier
}

for name, classifier in classifiers.items():
    classifier.fit(X_train_resampled, y_train_resampled)
    predictions = classifier.predict(X_test)
    print(f"Classifier: {name}")
    print(classification_report(y_test, predictions))
    print("="*50)
```

```
Classifier: Decision Tree

              precision    recall  f1-score   support

           0       0.90      0.86      0.88       255
           1       0.29      0.38      0.33        39
```