```
#1
import pandas as pd
import numpy as np
data = pd.read_csv(r"/Users/apple/Downloads/HR_comma_sep.csv")
missing_values = data.isnull().sum()
print(missing_values)
```

```
satisfaction_level       0
last_evaluation          0
number_project           0
average_montly_hours     0
time_spend_company       0
Work_accident            0
left                     0
promotion_last_5years    0
sales                    0
salary                   0
dtype: int64
```

```
#2
data.head()
data
df = pd.DataFrame(data)
df.head()
```

```
   satisfaction_level  last_evaluation  number_project
average_montly_hours  \
0                0.38             0.53               2
157
1                0.80             0.86               5
262
2                0.11             0.88               7
272
3                0.72             0.87               5
223
4                0.37             0.52               2
159

   time_spend_company  Work_accident  left  promotion_last_5years
sales  \
0                   3              0     1                      0
NaN
1                   6              0     1                      0
NaN
2                   4              0     1                      0
NaN
3                   5              0     1                      0
NaN
4                   3              0     1                      0
NaN
```

```
   salary
0     low
1  medium
2  medium
3     low
4     low
```

```
df.isna().sum()
```

```
satisfaction_level       0
last_evaluation          0
number_project           0
average_montly_hours     0
time_spend_company       0
Work_accident            0
left                     0
promotion_last_5years    0
sales                    0
salary                   0
dtype: int64
```

```
df["left"].unique()
```

```
array([], dtype=int64)
```

```
df["number_project"].unique()
```

```
array([], dtype=int64)
```

```
df.satisfaction_level.unique()
```

```
array([], dtype=float64)
```

```
df.last_evaluation.unique()
```

```
array([], dtype=float64)
```

```
df.time_spend_company.unique()
```

```
array([], dtype=int64)
```

```
df.Work_accident.unique()
```

```
array([], dtype=int64)
```

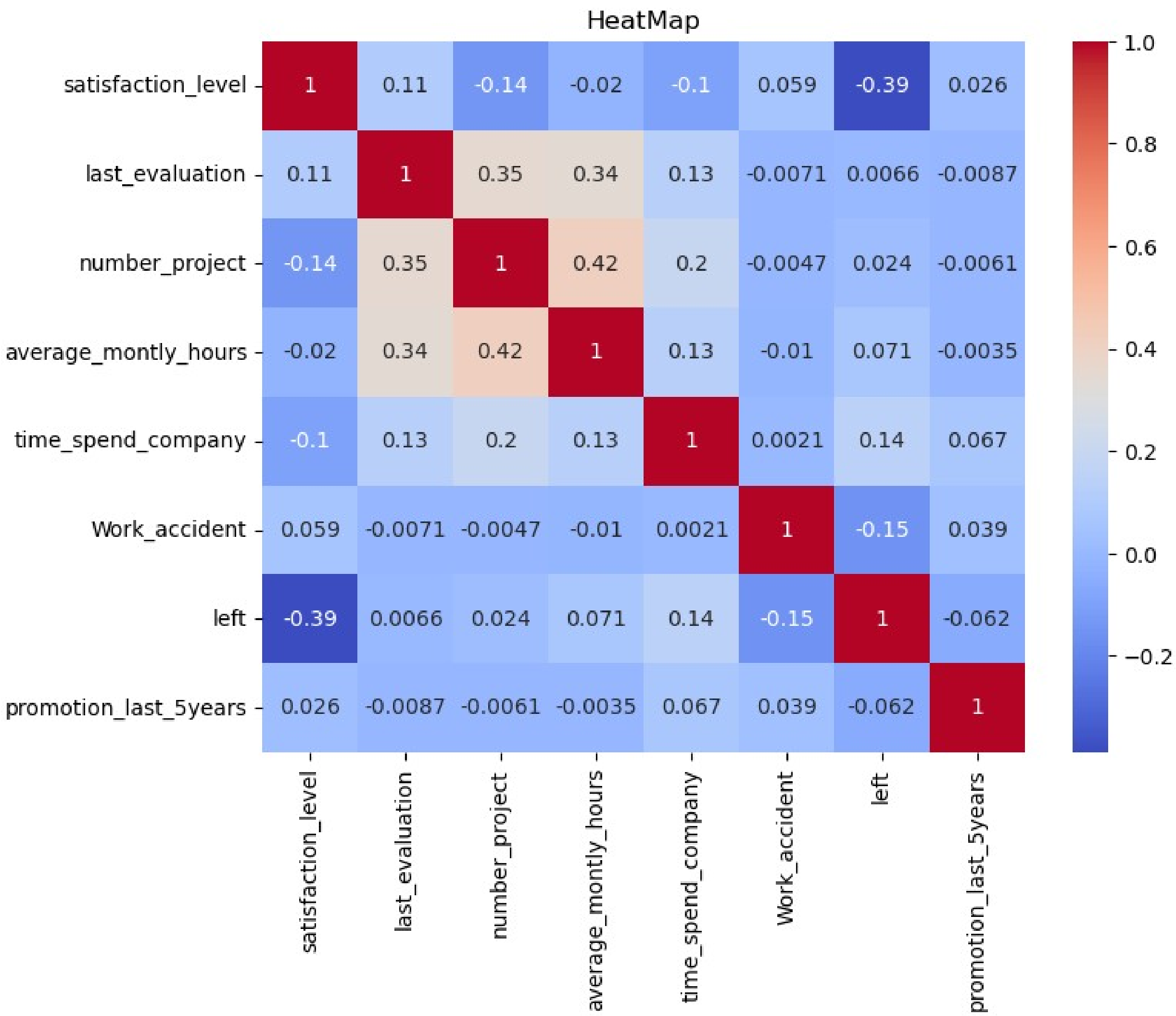
```
df.sales.unique()
```

```
array([], dtype=float64)
```

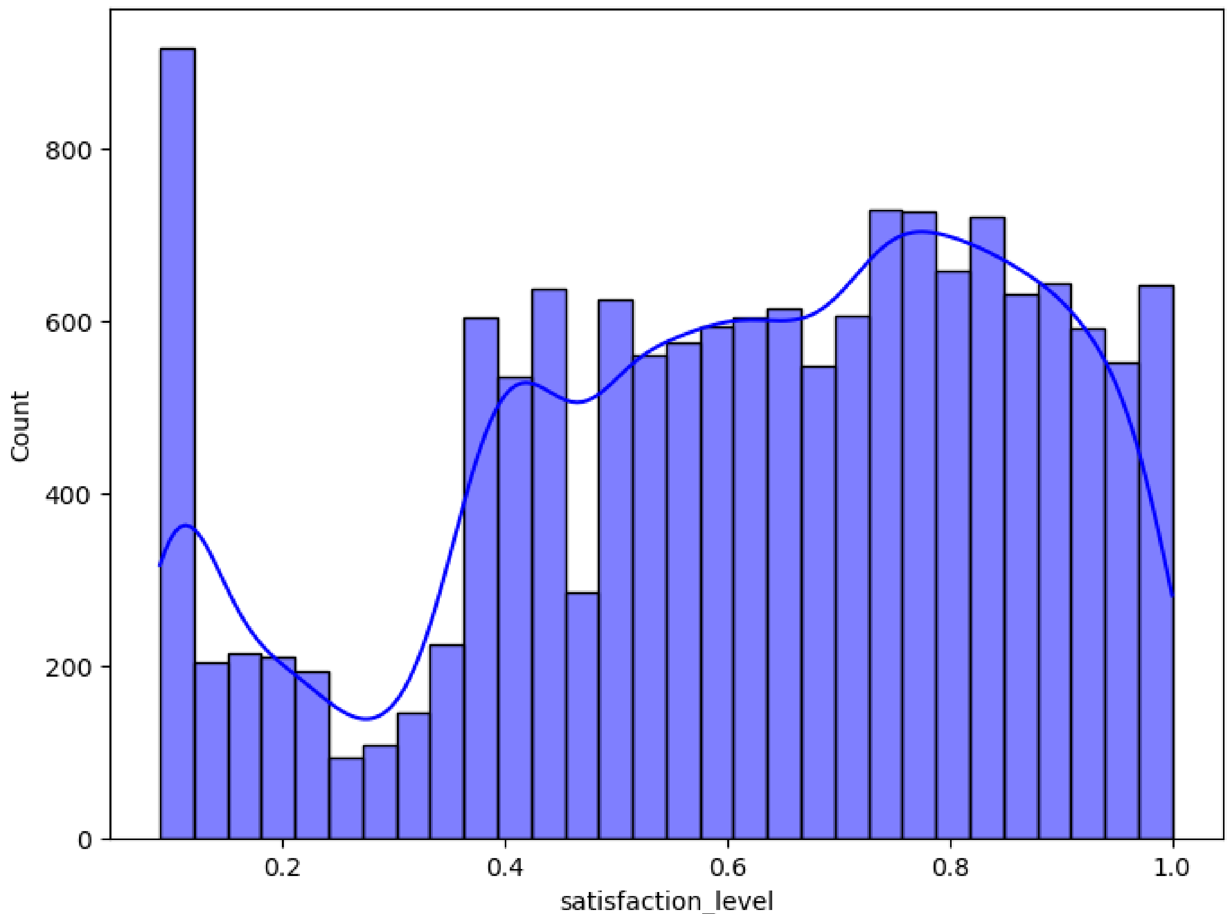
```
df.salary.unique()

array([], dtype=float64)

print(df.nunique())


Series([], dtype: float64)

print(df.dtypes)

Series([], dtype: object)

# Drop columns with only one unique value
df = df.loc[:, df.nunique() > 1]
df.corr()

Empty DataFrame
Columns: []
Index: []


import pandas as pd
import numpy as np
df = pd.read_csv(r"/Users/apple/Downloads/HR_comma_sep.csv")
numeric_df = df.select_dtypes(include=['number'])

if numeric_df.empty:
    print("No numeric data found for correlation.")
else:
    correlation_matrix = numeric_df.corr()

print(correlation_matrix)

                       satisfaction_level  last_evaluation
number_project  \
satisfaction_level               1.000000         0.105021        -
0.142970
last_evaluation                  0.105021         1.000000
0.349333
number_project                  -0.142970         0.349333
1.000000
average_montly_hours            -0.020048         0.339742
0.417211
time_spend_company              -0.100866         0.131591
0.196786
Work_accident                    0.058697        -0.007104        -
0.004741
left                            -0.388375         0.006567
0.023787
promotion_last_5years            0.025605        -0.008684        -
0.006064
```

```
                           average_montly_hours  time_spend_company  \
satisfaction_level                    -0.020048           -0.100866
last_evaluation                        0.339742            0.131591
number_project                         0.417211            0.196786
average_montly_hours                   1.000000            0.127755
time_spend_company                     0.127755            1.000000
Work_accident                         -0.010143            0.002120
left                                   0.071287            0.144822
promotion_last_5years                 -0.003544            0.067433

                           Work_accident      left  promotion_last_5years

satisfaction_level              0.058697 -0.388375               0.025605

last_evaluation                -0.007104  0.006567              -0.008684

number_project                 -0.004741  0.023787              -0.006064

average_montly_hours           -0.010143  0.071287              -0.003544

time_spend_company              0.002120  0.144822               0.067433

Work_accident                   1.000000 -0.154622               0.039245

left                           -0.154622  1.000000              -0.061788

promotion_last_5years           0.039245 -0.061788               1.000000
```

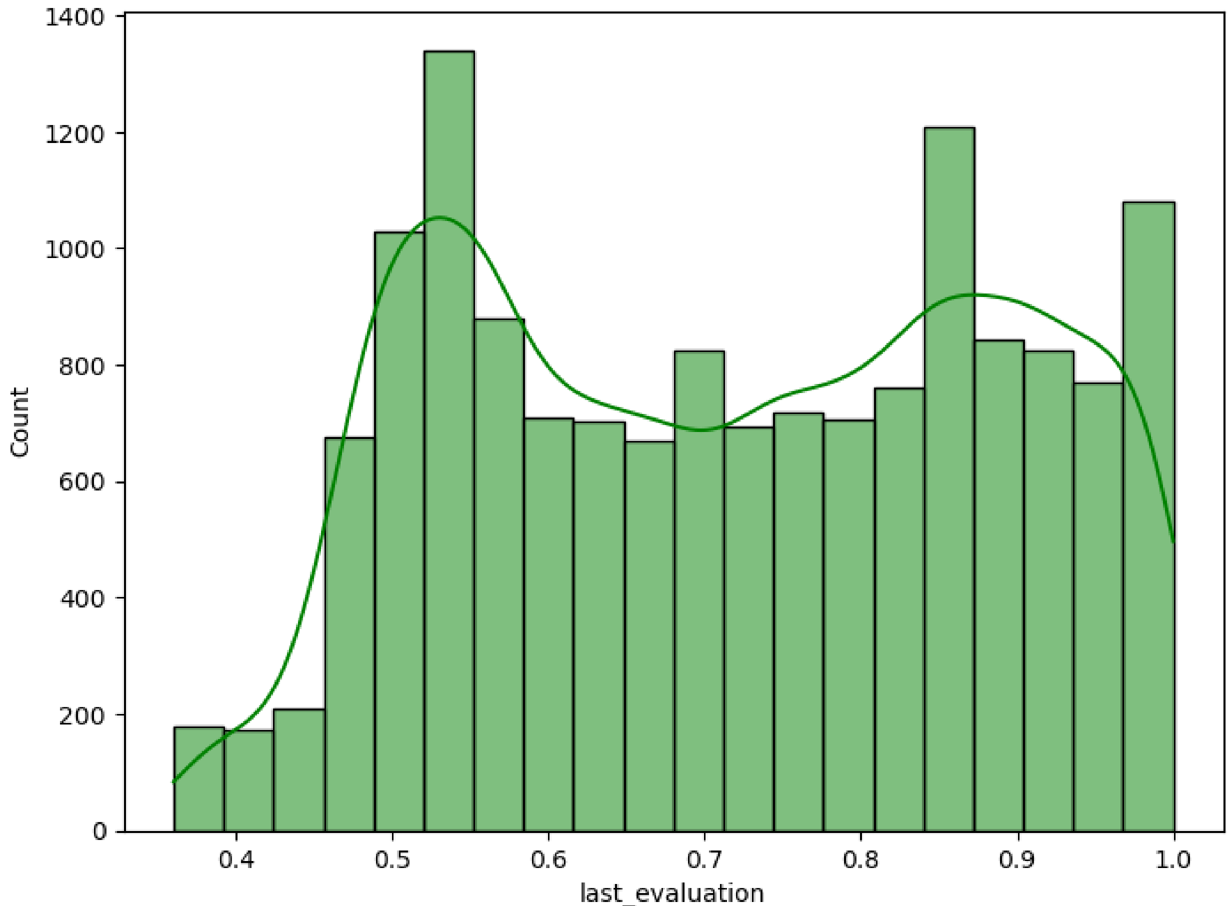
```
#heatmap
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize = (8,6))
sns.heatmap(correlation_matrix, annot = True,cmap ='coolwarm')
plt.title("HeatMap")
plt.show()
```




```
#2.2
#Employee Satisfacrion Distribution Plot

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv(r"/Users/apple/Downloads/HR_comma_sep.csv")
plt.figure(figsize =(8,6))
satisfaction_data = df['satisfaction_level']
sns.histplot(satisfaction_data, kde = True , color = 'blue' , bins =
30)

<Axes: xlabel='satisfaction_level', ylabel='Count'>
```

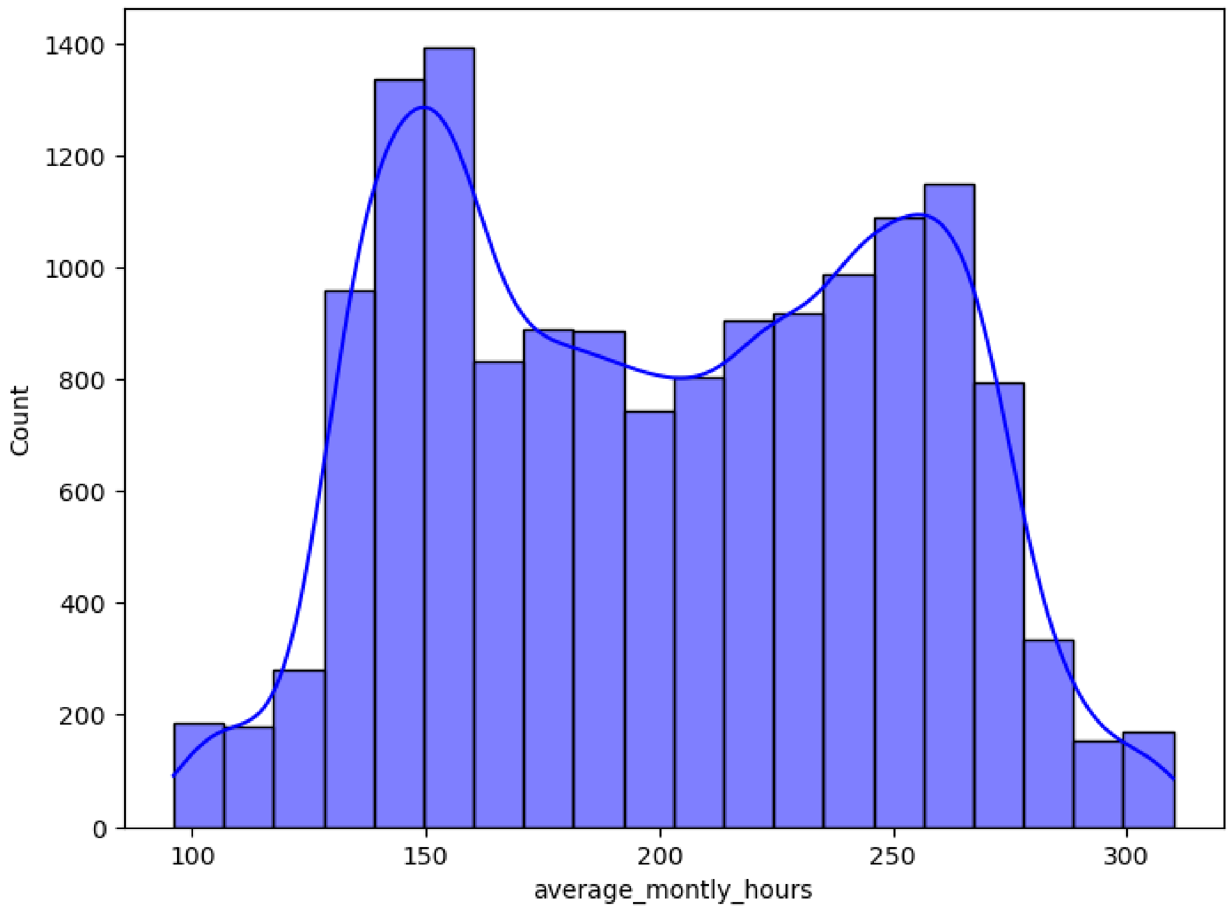



```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv(r"/Users/apple/Downloads/HR_comma_sep.csv")
plt.figure(figsize = (8,6))
last_evaluation = df['last_evaluation']
sns.histplot(last_evaluation, kde = True, color = 'green', bins = 20)

<Axes: xlabel='last_evaluation', ylabel='Count'>
```




```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv(r"/Users/apple/Downloads/HR_comma_sep.csv")
plt.figure(figsize = (8,6))
average_montly_hours = df['average_montly_hours']
sns.histplot(average_montly_hours, kde = True , color = 'blue', bins
=20)
```

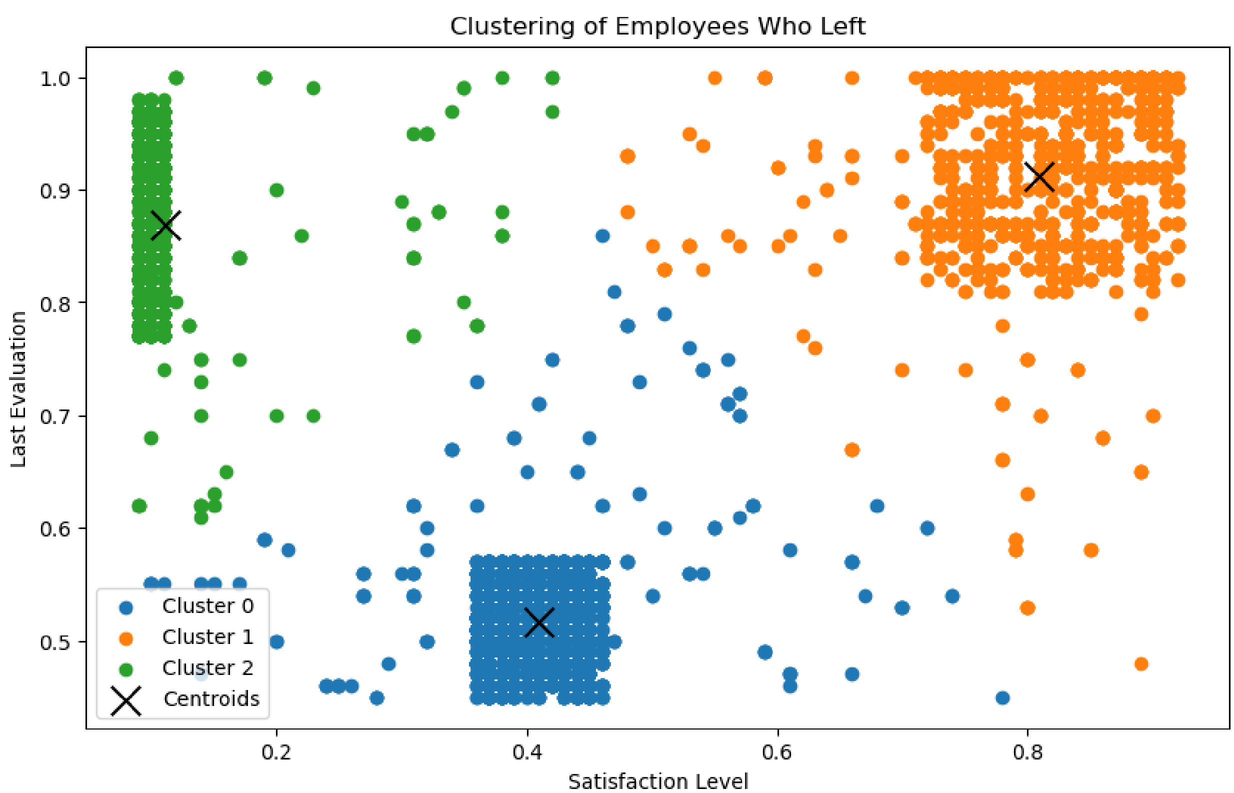
```
<Axes: xlabel='average_montly_hours', ylabel='Count'>
```




```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
data = pd.read_csv(r"/Users/apple/Downloads/HR_comma_sep.csv")
left_employees = data[data['left'] == 1]
features = left_employees[['satisfaction_level', 'last_evaluation']]

#  clustering
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(features)
left_employees['Cluster'] = clusters

# Visualize
plt.figure(figsize=(10, 6))
for cluster in range(3):
    cluster_data = left_employees[left_employees['Cluster'] ==
cluster]
    plt.scatter(
        cluster_data['satisfaction_level'],
        cluster_data['last_evaluation'],
```

```
        label=f'Cluster {cluster}'
    )

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
1],
            color='black', marker='x', s=200, label='Centroids')
plt.title('Clustering of Employees Who Left')
plt.xlabel('Satisfaction Level')
plt.ylabel('Last Evaluation')
plt.legend()
plt.show()

# Print cluster centers
print("Cluster centers:")
print(kmeans.cluster_centers_)
```

```
/var/folders/r8/5p91n_mn2hj2fl6bdg5qg7100000gn/T/
ipykernel_86878/1034483085.py:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  left_employees['Cluster'] = clusters
```



```
Cluster centers:
[[0.41014545 0.51698182]
 [0.80851586 0.91170931]
 [0.11115466 0.86930085]]
```

```python
#the cluster high satisfactioon and lov evaluation shoow inverse relationship .


import pandas as pd
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
data = pd.read_csv(r"/Users/apple/Downloads/HR_comma_sep.csv")
categorical_cols = data.select_dtypes(include=['object']).columns
numeric_cols = data.select_dtypes(include=['number']).columns

categorical_data = data[categorical_cols]
numeric_data = data[numeric_cols]

categorical_data_dummies = pd.get_dummies(categorical_data, drop_first=True)
processed_data = pd.concat([numeric_data, categorical_data_dummies], axis=1)
X = processed_data.drop('left', axis=1)  # Features (all columns except 'left')
y = processed_data['left']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
print("Class distribution before SMOTE:")
print(y_train.value_counts())
print("\nClass distribution after SMOTE:")
print(pd.Series(y_train_resampled).value_counts())
```

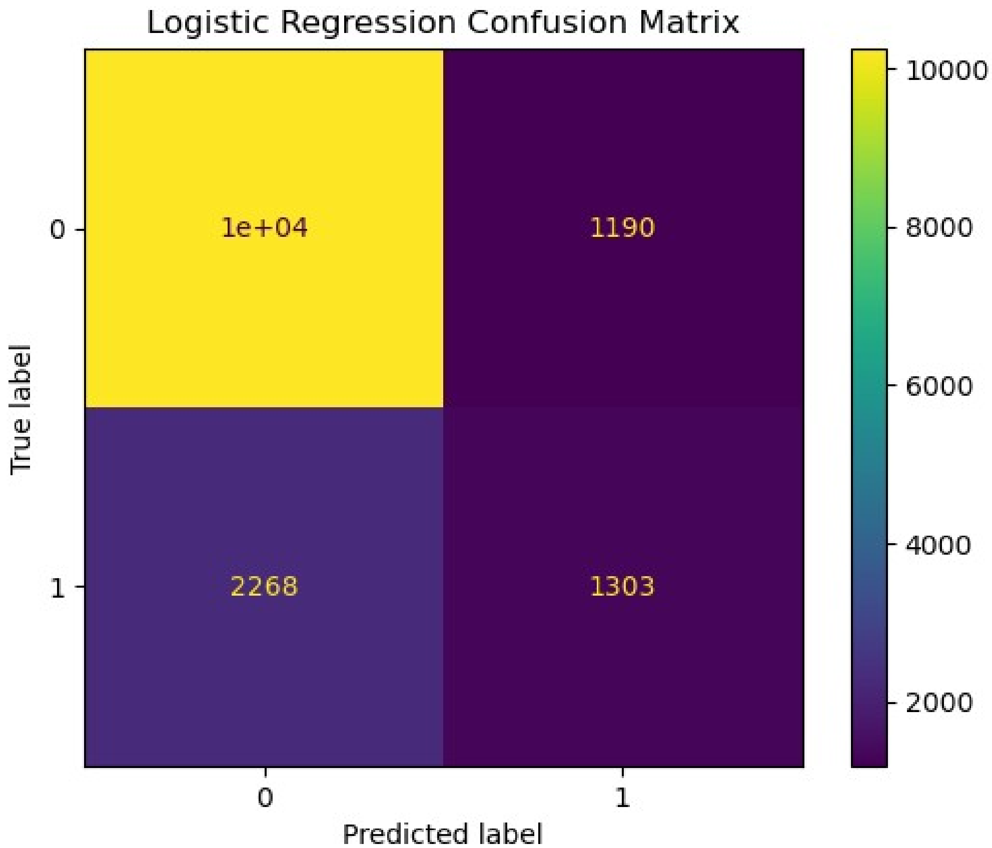
```
Class distribution before SMOTE:
left
0    7999
1    2500
Name: count, dtype: int64

Class distribution after SMOTE:
left
0    7999
1    7999
Name: count, dtype: int64
```

```python
# After introoductin smote the nuumber , we see there was a significant imbalance. It changed afer the smote was applied .
```

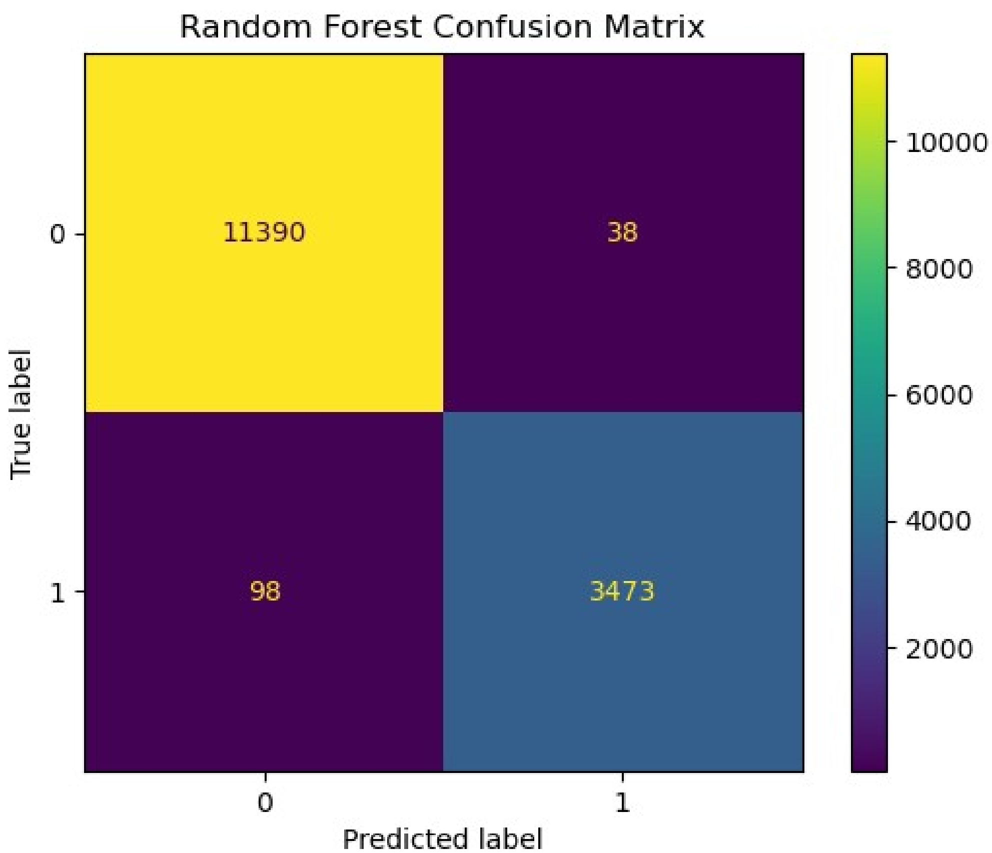
```python
#the number of minority class samples was increased.



import pandas as pd
from sklearn.model_selection import cross_val_predict
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.metrics import classification_report,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt
data = pd.read_csv(r"/Users/apple/Downloads/HR_comma_sep.csv")
X = data.drop(columns=["left"])
y = data["left"]
categorical_cols = ["sales", "salary"]
numeric_cols = [
    "satisfaction_level",
    "last_evaluation",
    "number_project",
    "average_montly_hours",
    "time_spend_company",
    "Work_accident",
    "promotion_last_5years",
]

# Preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_cols),
        ("cat", OneHotEncoder(), categorical_cols),
    ]
)

# Define models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
}

# Perform 5-fold cross-validation and generate reports
results = {}

for name, model in models.items():
    pipeline = Pipeline(steps=[("preprocessor", preprocessor),
("classifier", model)])
```

```python
    y_pred = cross_val_predict(pipeline, X, y, cv=5)
    report = classification_report(y, y_pred, output_dict=True)
    results[name] = report

    # Display classification report and confusion matrix
    print(f"\n{name} Classification Report:\n")
    print(classification_report(y, y_pred))
    ConfusionMatrixDisplay.from_predictions(y, y_pred)
    plt.title(f"{name} Confusion Matrix")
    plt.show()

# Summary of results
results_summary = {
    name: {
        "Precision (1)": metrics["1"]["precision"],
        "Recall (1)": metrics["1"]["recall"],
        "F1-score (1)": metrics["1"]["f1-score"],
        "Accuracy": metrics["accuracy"],
    }
    for name, metrics in results.items()
}

# Display the summary
print("\nModel Performance Summary:\n")
summary_df = pd.DataFrame(results_summary).T
print(summary_df)
```

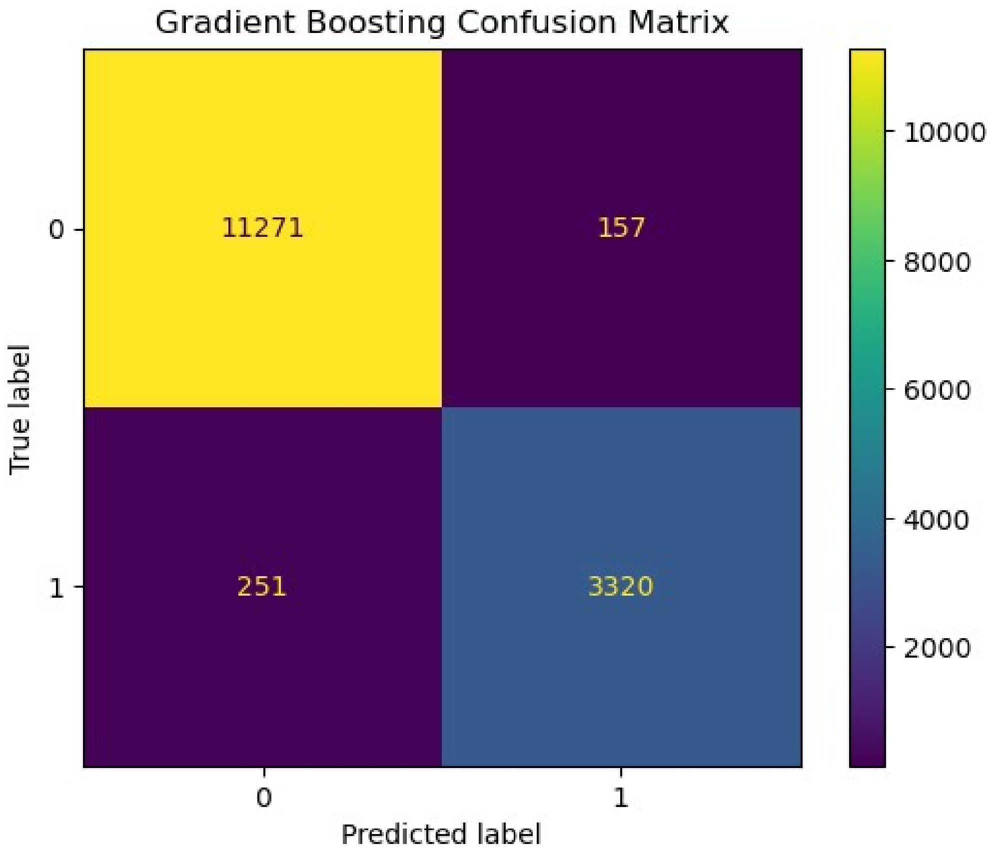
```
Logistic Regression Classification Report:

              precision    recall  f1-score   support

           0       0.82      0.90      0.86     11428
           1       0.52      0.36      0.43      3571

    accuracy                           0.77     14999
   macro avg       0.67      0.63      0.64     14999
weighted avg       0.75      0.77      0.75     14999
```




```
Random Forest Classification Report:

              precision    recall  f1-score   support

           0       0.99      1.00      0.99     11428
           1       0.99      0.97      0.98      3571

    accuracy                           0.99     14999
   macro avg       0.99      0.98      0.99     14999
weighted avg       0.99      0.99      0.99     14999
```



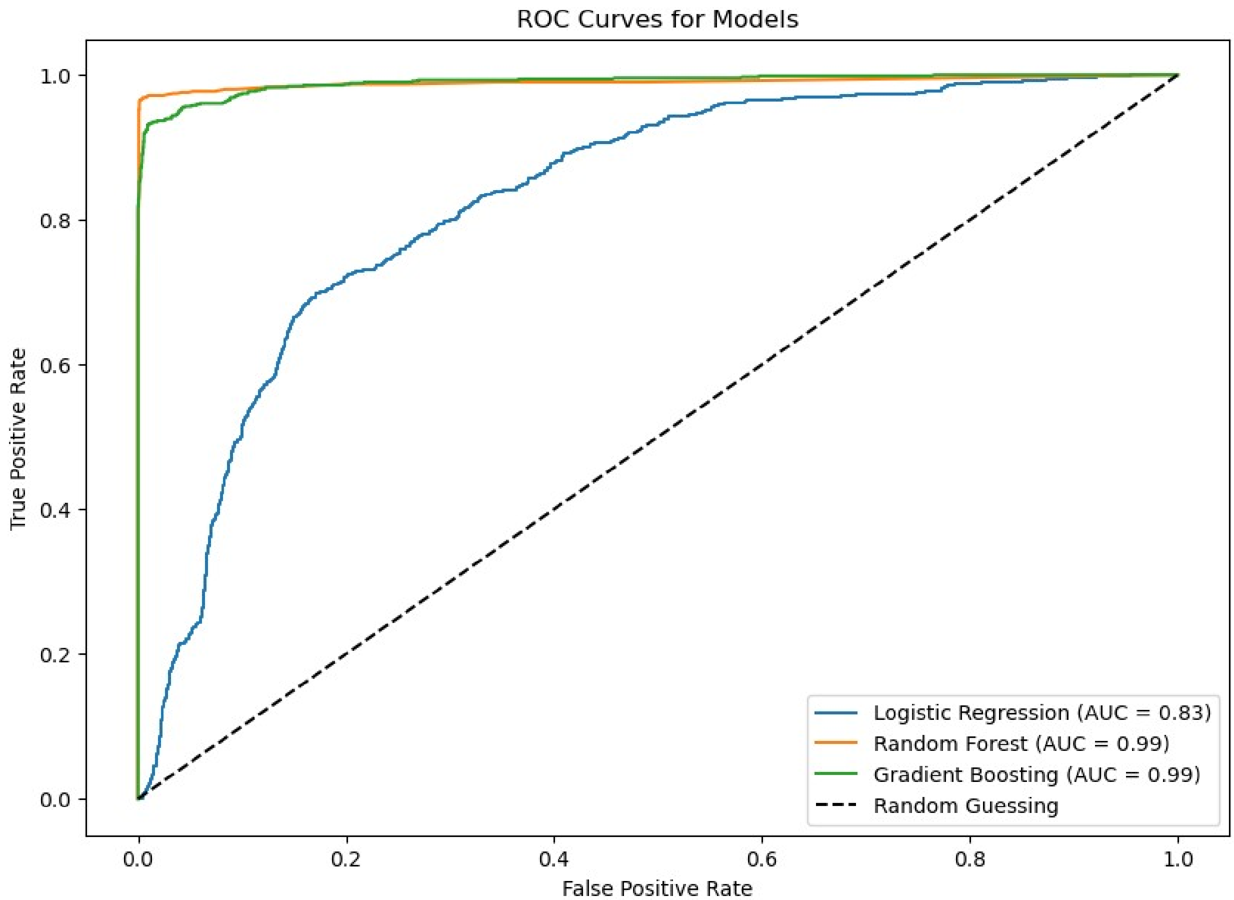

```
Gradient Boosting Classification Report:

              precision    recall  f1-score   support

           0       0.98      0.99      0.98     11428
           1       0.95      0.93      0.94      3571

    accuracy                           0.97     14999
   macro avg       0.97      0.96      0.96     14999
weighted avg       0.97      0.97      0.97     14999
```



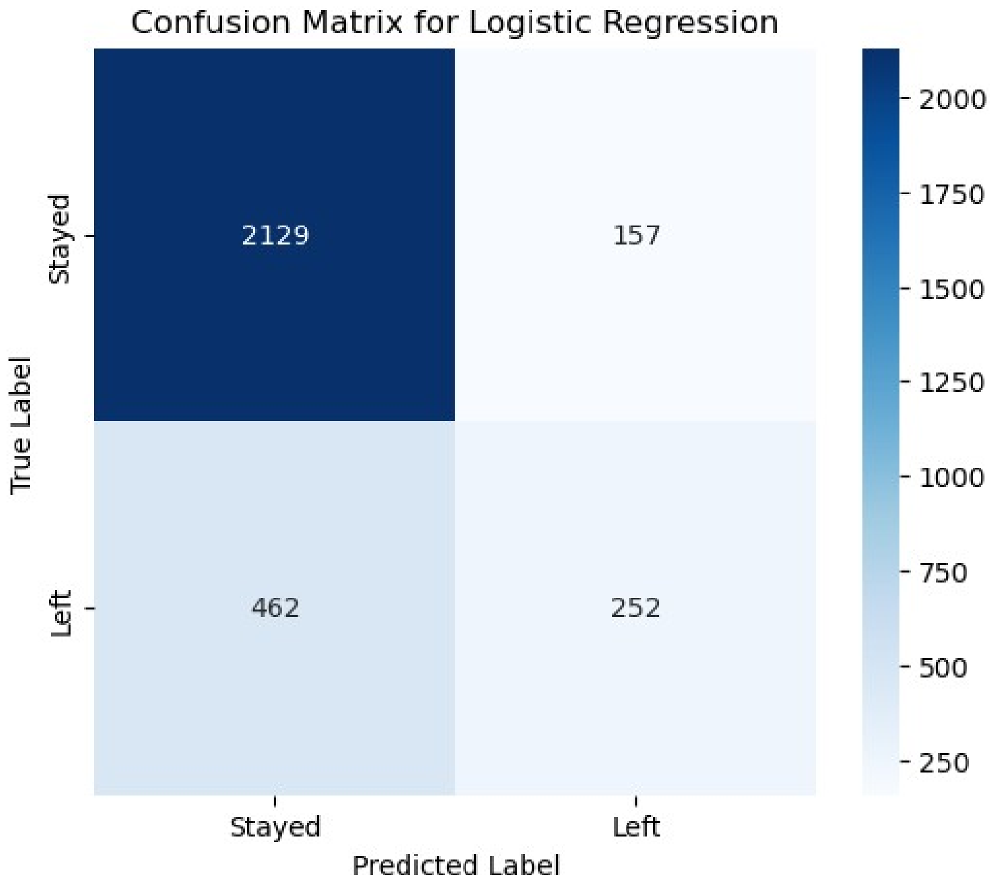

```
Model Performance Summary:

                     Precision (1)  Recall (1)  F1-score (1)  Accuracy
Logistic Regression       0.522663    0.364884      0.429749  0.769451
Random Forest             0.989177    0.972557      0.980796  0.990933
Gradient Boosting         0.954846    0.929712      0.942111  0.972798

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.metrics import roc_curve, auc, confusion_matrix,
classification_report, RocCurveDisplay
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv(r"/Users/apple/Downloads/HR_comma_sep.csv")
X = data.drop(columns=["left"])
```

```python
y = data["left"]

categorical_cols = ["sales", "salary"]
numeric_cols = [
    "satisfaction_level",
    "last_evaluation",
    "number_project",
    "average_montly_hours",
    "time_spend_company",
    "Work_accident",
    "promotion_last_5years",
]

preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_cols),
        ("cat", OneHotEncoder(), categorical_cols),
    ]
)

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

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000,
random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
}

# Train models, calculate ROC/AUC, and plot ROC curves
roc_auc_scores = {}
plt.figure(figsize=(10, 7))

for name, model in models.items():

    pipeline = Pipeline(steps=[("preprocessor", preprocessor),
("classifier", model)])


    pipeline.fit(X_train, y_train)


    y_pred_proba = pipeline.predict_proba(X_test)[:, 1]


    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    auc_score = auc(fpr, tpr)
    roc_auc_scores[name] = auc_score
```

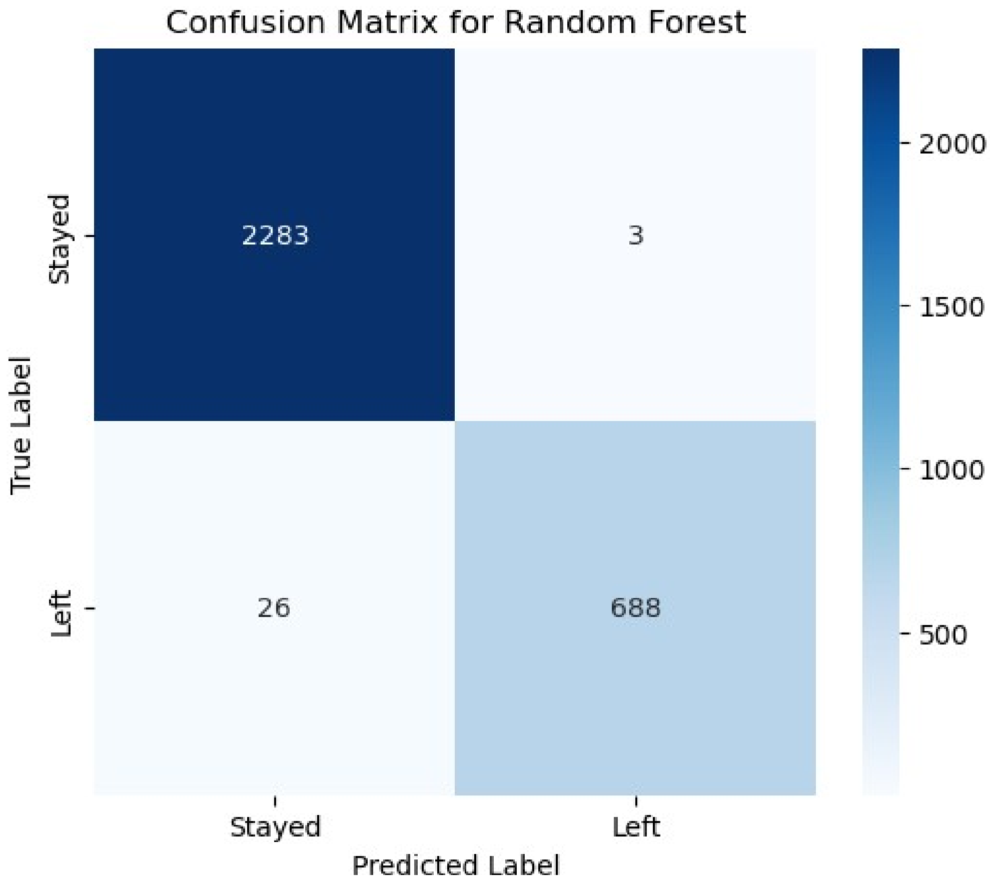
```
    plt.plot(fpr, tpr, label=f"{name} (AUC = {auc_score:.2f})")

# Plot details
plt.plot([0, 1], [0, 1], "k--", label="Random Guessing")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves for Models")
plt.legend(loc="lower right")
plt.show()

print("ROC/AUC Scores:")
for model, score in roc_auc_scores.items():
    print(f"{model}: AUC = {score:.3f}")

# Compute and Display confusion matrices and classification reports
for each model
for name, model in models.items():
    # Build a pipeline
    pipeline = Pipeline(steps=[("preprocessor", preprocessor),
("classifier", model)])

    # Train
    pipeline.fit(X_train, y_train)


    y_pred = pipeline.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)

    # Plot Confusion matrix
    plt.figure(figsize=(6, 5))
    plt.title(f"Confusion Matrix for {name}")
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=["Stayed", "Left"], yticklabels=["Stayed", "Left"])
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
    print(f"Classification Report for {name}:")
    print(classification_report(y_test, y_pred))
```



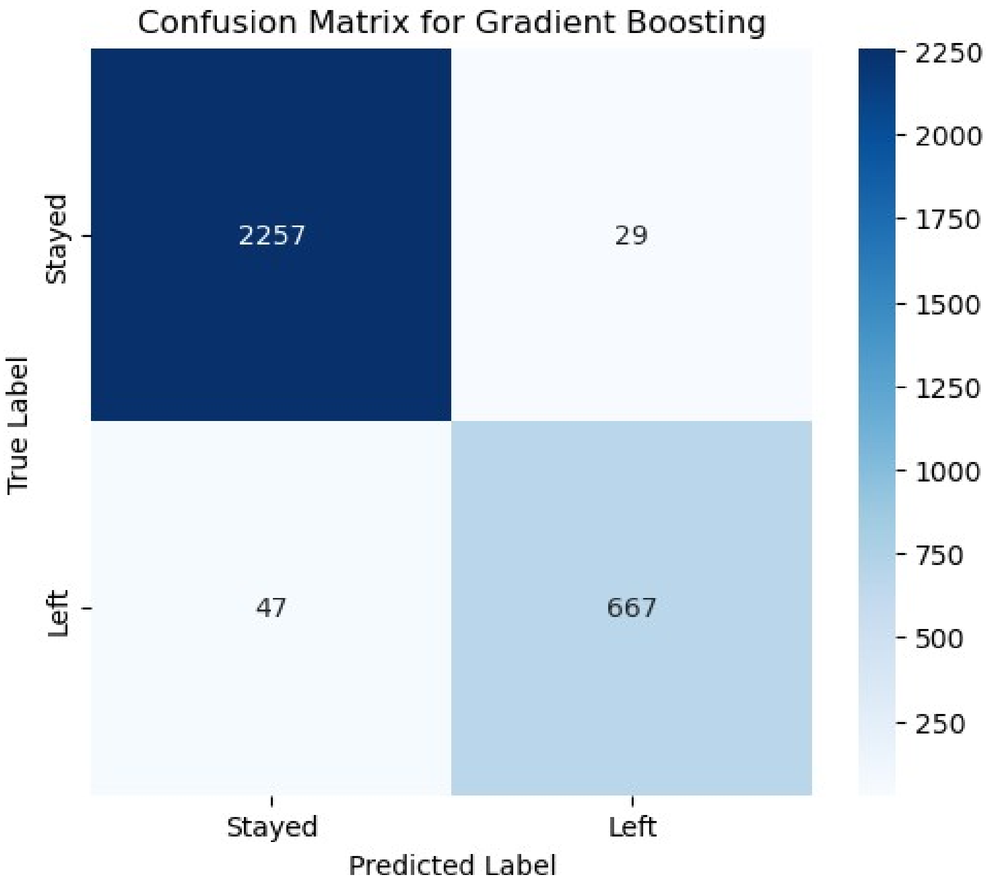

```
ROC/AUC Scores:
Logistic Regression: AUC = 0.828
Random Forest: AUC = 0.990
Gradient Boosting: AUC = 0.990
```




```
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.82      0.93      0.87      2286
           1       0.62      0.35      0.45       714

    accuracy                           0.79      3000
   macro avg       0.72      0.64      0.66      3000
weighted avg       0.77      0.79      0.77      3000
```




```
Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      2286
           1       1.00      0.96      0.98       714

    accuracy                           0.99      3000
   macro avg       0.99      0.98      0.99      3000
weighted avg       0.99      0.99      0.99      3000
```

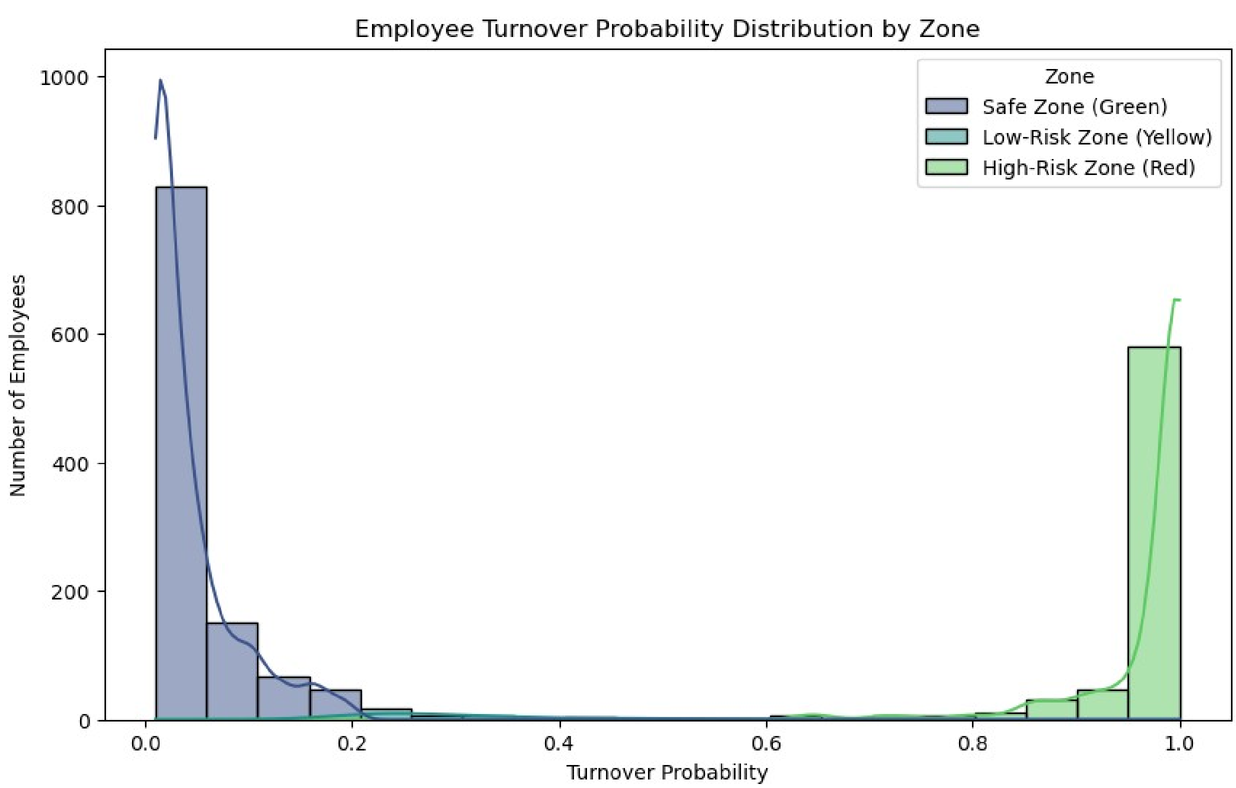



```
Classification Report for Gradient Boosting:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      2286
           1       0.96      0.93      0.95       714

    accuracy                           0.97      3000
   macro avg       0.97      0.96      0.96      3000
weighted avg       0.97      0.97      0.97      3000


#in my opinion , the random Forest achieves the best balance, with
near-perfect precision, recall, and F1-score. It minimizes both false
negative

 #Using the best model, predict the probability of employee turnover
in t
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.metrics import roc_curve, auc, confusion_matrix,
classification_report, RocCurveDisplay
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv(r"/Users/apple/Downloads/HR_comma_sep.csv")
X = data.drop(columns=["left"])
y = data["left"]
categorical_cols = ["sales", "salary"]
numeric_cols = [
    "satisfaction_level",
    "last_evaluation",
    "number_project",
    "average_montly_hours",
    "time_spend_company",
    "Work_accident",
    "promotion_last_5years",
]

preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_cols),
        ("cat", OneHotEncoder(), categorical_cols),
    ]
)

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

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000,
random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
}

best_model = RandomForestClassifier(random_state=42)
best_pipeline = Pipeline(steps=[("preprocessor", preprocessor),
("classifier", best_model)])
best_pipeline.fit(X_train, y_train)
y_pred_proba = best_pipeline.predict_proba(X_test)[:, 1]
zones = pd.DataFrame({
    "Employee": range(len(y_pred_proba)),
    "Probability": y_pred_proba
})
zones["Zone"] = pd.cut(
    zones["Probability"],
    bins=[0, 0.2, 0.6, 1],
    labels=["Safe Zone (Green)", "Low-Risk Zone (Yellow)", "High-Risk
```

```
Zone (Red)"]
)
print("Categorized Zones:")
print(zones.head())
plt.figure(figsize=(10, 6))
sns.histplot(zones, x="Probability", hue="Zone", kde=True,
palette="viridis", bins=20)
plt.title("Employee Turnover Probability Distribution by Zone")
plt.xlabel("Turnover Probability")
plt.ylabel("Number of Employees")
plt.show()
print("\nRetention Strategies:")
print("1. Safe Zone (Green): Maintain current policies; these
employees are highly satisfied.")
print("2. Low-Risk Zone (Yellow): Monitor satisfaction and engagement
levels; small interventions may prevent turnover.")
print("3. High-Risk Zone (Red): Implement targeted retention
strategies, such as personalized career growth plans or incentives.")
```

```
Categorized Zones:
   Employee  Probability                 Zone
0         0         0.04  Safe Zone (Green)
1         1         0.00                  NaN
2         2         0.03  Safe Zone (Green)
3         3         0.00                  NaN
4         4         0.00                  NaN
```



```
Retention Strategies:
1. Safe Zone (Green): Maintain current policies; these employees are
highly satisfied.
2. Low-Risk Zone (Yellow): Monitor satisfaction and engagement levels;
small interventions may prevent turnover.
3. High-Risk Zone (Red): Implement targeted retention strategies, such
as personalized career growth plans or incentives.
```