This Is My 30. Employee Promotion Prediction Project

Import Required Libraries

In [1]:
```
# Data handling
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Preprocessing & splitting
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import make_pipeline,Pipeline
from sklearn.impute import SimpleImputer
from sklearn.utils import resample

# Evaluation metrics
from sklearn.metrics import (accuracy_score,precision_score,recall_score,f1_scor
    classification_report,confusion_matrix,auc,roc_auc_score, roc_curve
)

# Regression Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier

# XGBoost
from xgboost import XGBClassifier

# Warnings
import warnings
warnings.filterwarnings("ignore")

# Save the model
import joblib
```

Load Dataset

In [2]:
```
data = pd.read_csv("Employees Promotion.csv")
data.head()
```

Out[2]:

	EmployeeID	Department	Region_Employment	Education Level	Gender	Recruitment Channel	NO
0	65438	Sales & Marketing	7	Master's & above	f	sourcing	
1	65141	Operations	22	Bachelor's	m	other	
2	7513	Sales & Marketing	19	Bachelor's	m	sourcing	
3	2542	Sales & Marketing	23	Bachelor's	m	other	
4	48945	Technology	26	Bachelor's	m	other	

Data Preprocessing

In [3]:
```
data.shape
```

Out[3]:
```
(54808, 13)
```

In [4]:
```
data.duplicated().sum()
```

Out[4]:
```
0
```

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54808 entries, 0 to 54807
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   EmployeeID            54808 non-null  int64  
 1   Department            54808 non-null  object 
 2   Region_Employment     54808 non-null  int64  
 3   Education Level       52399 non-null  object 
 4   Gender                54808 non-null  object 
 5   Recruitment Channel   44404 non-null  object 
 6   NO_Trainings_LstYear  54808 non-null  int64  
 7   Age                   54268 non-null  float64
 8   previous_year_rating  50684 non-null  float64
 9   Service Length        54808 non-null  int64  
 10  Awards                54808 non-null  object 
 11  Avg_Training_Score    52248 non-null  float64
 12  Is Promoted           54808 non-null  object 
dtypes: float64(3), int64(4), object(6)
memory usage: 5.4+ MB
```

In [6]:
```
data.isna().sum()
```

Out[6]:
```
EmployeeID                 0
Department                 0
Region_Employment          0
Education Level         2409
Gender                     0
Recruitment Channel    10404
NO_Trainings_LstYear       0
Age                      540
previous_year_rating    4124
Service Length             0
Awards                     0
Avg_Training_Score      2560
Is Promoted                0
dtype: int64
```

In [7]:
```
a = pd.DataFrame()
a['columns'] = list(data.columns)
a['nulls'] = list(data.isna().sum())

for i in a[a['nulls']>500]['columns'].values:
    data.drop(i,axis=1,inplace=True)
```

In [8]:
```
data.isna().sum()
```

Out[8]:
```
EmployeeID              0
Department              0
Region_Employment       0
Gender                  0
NO_Trainings_LstYear    0
Service Length          0
Awards                  0
Is Promoted             0
dtype: int64
```

In [9]:
```
data.describe()
```

Out[9]:

	EmployeeID	Region_Employment	NO_Trainings_LstYear	Service Length
count	54808.000000	54808.000000	54808.000000	54808.000000
mean	39195.830627	14.195045	1.253011	5.865512
std	22586.581449	10.086273	0.609264	4.265094
min	1.000000	1.000000	1.000000	1.000000
25%	19669.750000	4.000000	1.000000	3.000000
50%	39225.500000	13.000000	1.000000	5.000000
75%	58730.500000	22.000000	1.000000	7.000000
max	78298.000000	34.000000	10.000000	37.000000

In [10]:
```
cat = data.select_dtypes(include='object')
for i in cat.columns:
    print(f"{i}")
    print(f"{cat[i].nunique()}")
```

```
Department
10
Gender
2
Awards
2
Is Promoted
2
```

In [11]:
```
for i in cat.columns:
    print(f"{i}")
    print(f"{cat[i].unique()}")
```

```
Department
['Sales & Marketing' 'Operations' 'Technology' 'Analytics' 'R&D'
 'Procurement' 'Finance' 'HR' 'Legal' '-']
Gender
['f' 'm']
Awards
['NO' 'YES']
Is Promoted
['NO' 'YES']
```

In [12]:
```
data = data.drop(data[data['Department'] == '-'].index)

print(data['Department'].value_counts())
```

```
Department
Sales & Marketing    16773
Operations           11304
Procurement           7117
Technology            7113
Analytics             5330
Finance               2525
HR                    2411
Legal                 1035
R&D                    995
Name: count, dtype: int64
```

Convert to imbalance dataset to balance dataset

In [13]:
```
data['Is Promoted'].value_counts()
```

Out[13]:
```
Is Promoted
NO     49954
YES     4649
Name: count, dtype: int64
```

In [14]:
```
max_count = data['Is Promoted'].value_counts().max()
max_count
```

Out[14]:
```
49954
```

In [15]:
```
upsampled_list = []
for cls in data['Is Promoted'].unique():
    cls_data = data[data['Is Promoted'] == cls]
    cls_upsampled = resample(
```

```
        cls_data,
        replace=True,
        n_samples=max_count,
        random_state=42
    )
    upsampled_list.append(cls_upsampled)

data_balanced = pd.concat(upsampled_list)
data_balanced = data_balanced.sample(frac=1, random_state=42).reset_index(drop=T

data =data_balanced

data['Is Promoted'].value_counts()
```

Out[15]:
```
Is Promoted
NO     49954
YES    49954
Name: count, dtype: int64
```

Encoding

In [16]:
```
data['Gender'] = data['Gender'].map({'f':0,'m':1})
data['Awards'] = data['Awards'].map({'NO':0,'YES':1})
data['Is Promoted'] = data['Is Promoted'].map({'NO':0,'YES':1})
```

In [17]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99908 entries, 0 to 99907
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   EmployeeID            99908 non-null  int64
 1   Department            99908 non-null  object
 2   Region_Employment     99908 non-null  int64
 3   Gender                99908 non-null  int64
 4   NO_Trainings_LstYear  99908 non-null  int64
 5   Service Length        99908 non-null  int64
 6   Awards                99908 non-null  int64
 7   Is Promoted           99908 non-null  int64
dtypes: int64(7), object(1)
memory usage: 6.1+ MB
```

In [18]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data['Department'] = le.fit_transform(data['Department'])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99908 entries, 0 to 99907
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   EmployeeID            99908 non-null  int64
 1   Department            99908 non-null  int32
 2   Region_Employment     99908 non-null  int64
 3   Gender                99908 non-null  int64
 4   NO_Trainings_LstYear  99908 non-null  int64
 5   Service Length        99908 non-null  int64
 6   Awards                99908 non-null  int64
 7   Is Promoted           99908 non-null  int64
dtypes: int32(1), int64(7)
memory usage: 5.7 MB
```

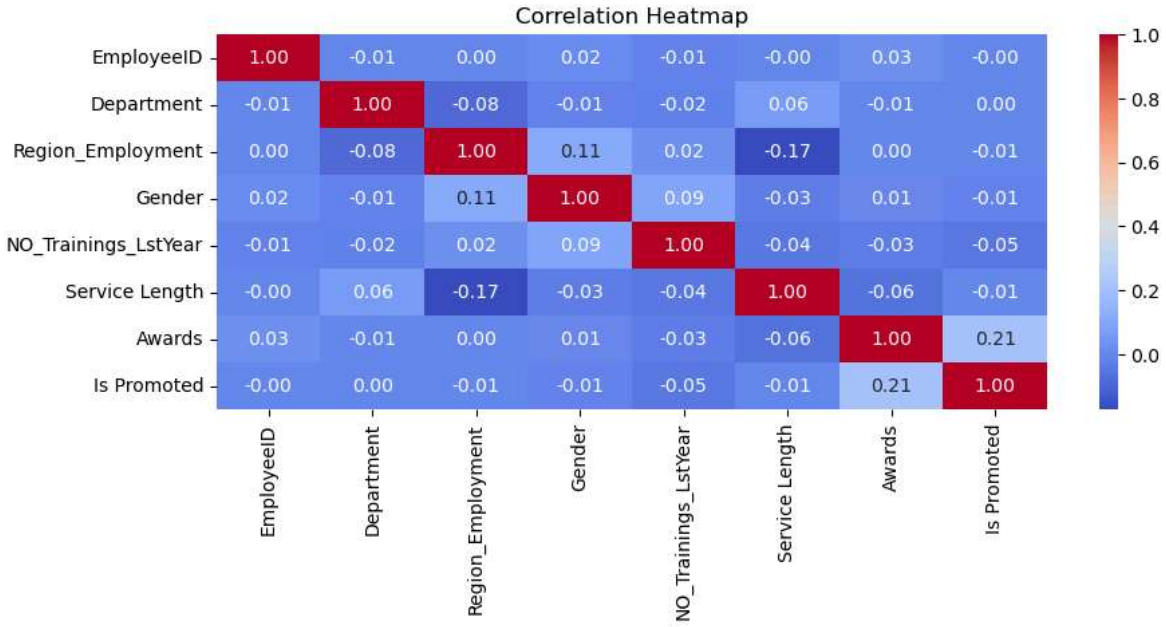
I use sample 20000 Dataset

In [19]:
```
data = data.sample(n=20000, replace=True, random_state=42)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20000 entries, 15795 to 59426
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   EmployeeID            20000 non-null  int64
 1   Department            20000 non-null  int32
 2   Region_Employment     20000 non-null  int64
 3   Gender                20000 non-null  int64
 4   NO_Trainings_LstYear  20000 non-null  int64
 5   Service Length        20000 non-null  int64
 6   Awards                20000 non-null  int64
 7   Is Promoted           20000 non-null  int64
dtypes: int32(1), int64(7)
memory usage: 1.3 MB
```

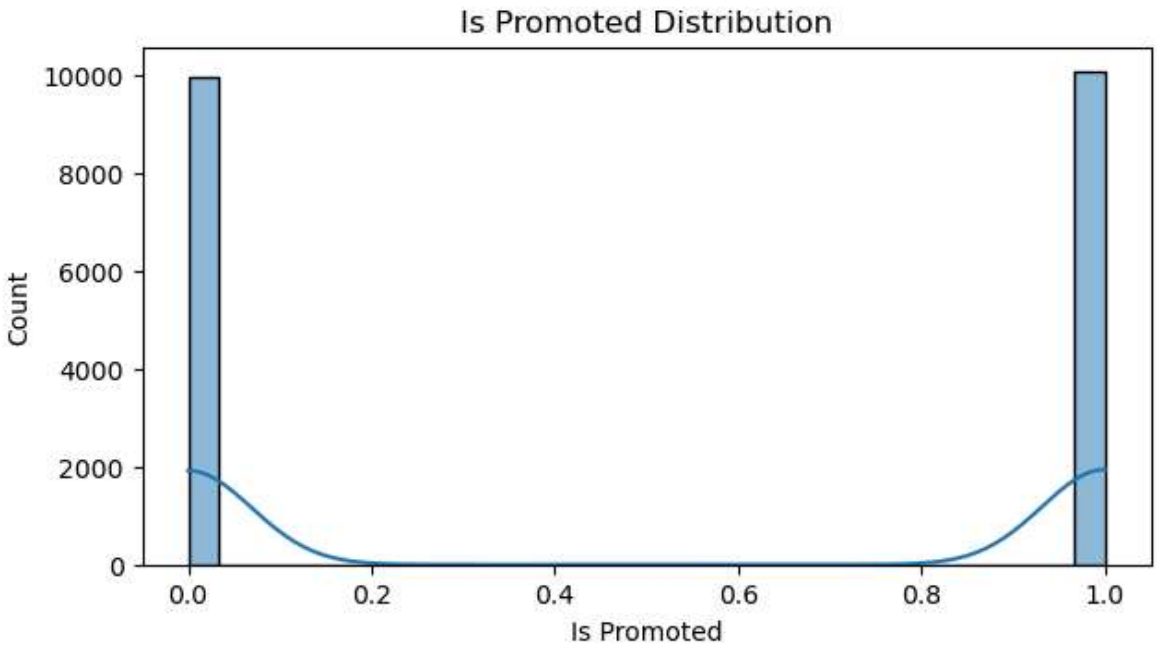
Heatmap

In [20]:
```
plt.figure(figsize=(34,8))
plt.subplot(2,3,2)
sns.heatmap(data.corr(), annot=True, cmap='coolwarm',fmt='.2f')
plt.title('Correlation Heatmap')
plt.show();
```



Is Promoted Distribution

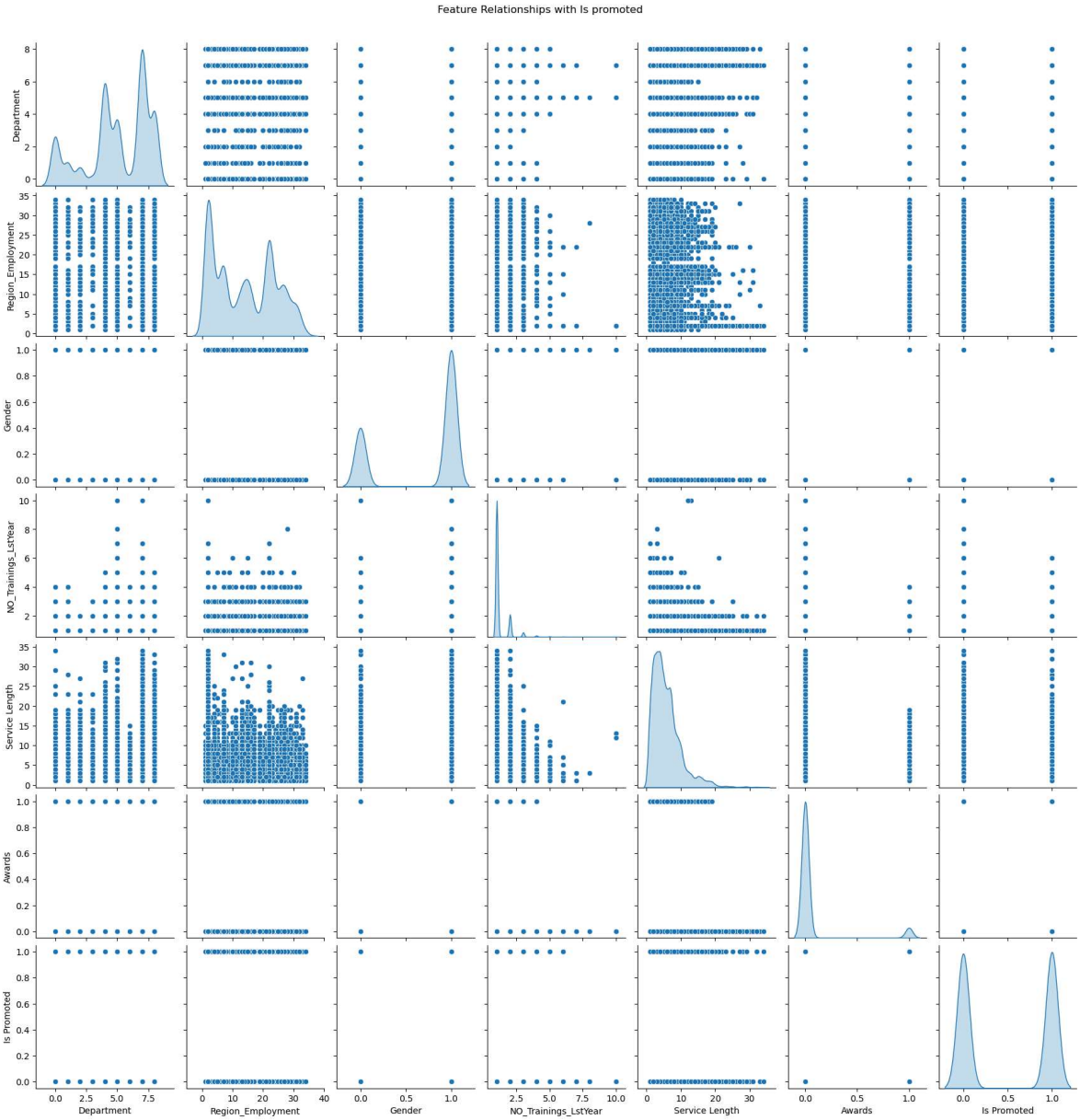
In [21]:
```
plt.figure(figsize=(24,8))
plt.subplot(2,3,1)
sns.histplot(data['Is Promoted'], kde=True, bins=30)
plt.title('Is Promoted Distribution')
plt.show();
```



Feature Relationships with Is promoted

In [28]:
```
plt.figure(figsize=(14,8))
sns.pairplot(data[['Department', 'Region_Employment', 'Gender',
       'NO_Trainings_LstYear', 'Service Length', 'Awards', 'Is Promoted']], diag
plt.suptitle("Feature Relationships with Is promoted", y=1.02)
plt.show();
```

```
<Figure size 1400x800 with 0 Axes>
```



Splitting for Target and Feature into Train Test Split

In [22]:
```
X = data.drop(['Is Promoted','EmployeeID'],axis=1)
y = data['Is Promoted']

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

Preprocessing pipeline

In [23]:
```
preprocessor = Pipeline([
    ('imputer',SimpleImputer(strategy='median')),
    ('scaler',StandardScaler())
])
```

Define models & parameter grids for classification

In [24]:
```
models = {
    # -------------------------------
    "Logistic Regression": {
        "model": LogisticRegression(solver='lbfgs', max_iter=500, random_state=4
        "param_grid": {
            "C": [0.1, 1, 10],
            "penalty": ['l2']
        }
    },
    # -------------------------------
    "Decision Tree": {
        "model": DecisionTreeClassifier(random_state=42),
        "param_grid": {
            "max_depth": [3, 5, 10, None],
            "min_samples_split": [2, 5, 10],
            "min_samples_leaf": [1, 2, 4]
        }
    },
    # -------------------------------
    "Random Forest": {
        "model": RandomForestClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2]
        }
    },
    # -------------------------------
    "Gradient Boosting": {
        "model": GradientBoostingClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "learning_rate": [0.01, 0.1, 0.2],
            "max_depth": [3, 5]
        }
    },
    # -------------------------------
    "AdaBoost": {
        "model": AdaBoostClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [50, 100, 200],
            "learning_rate": [0.01, 0.1, 1.0]
        }
    },
    # -------------------------------
    "Extra Trees": {
        "model": ExtraTreesClassifier(random_state=42),
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1, 2]
        }
```

```
    },
    # ------------------------------
    "K-Nearest Neighbors": {
        "model": KNeighborsClassifier(),
        "param_grid": {
            "n_neighbors": [3, 5, 7, 9],
            "weights": ["uniform", "distance"],
            "p": [1, 2]  # Manhattan (L1) or Euclidean (L2)
        }
    },
    # ------------------------------
    "MLP Classifier": {
        "model": MLPClassifier(random_state=42),
        "param_grid": {
            "hidden_layer_sizes": [(50,), (100,)],
            "activation": ["relu", "tanh"],
            "solver": ["adam"],
            "max_iter": [200]
        }
    },
    # ------------------------------
    "XGBoost": {
        "model": XGBClassifier(use_label_encoder=False, eval_metric="logloss", r
        "param_grid": {
            "n_estimators": [100, 200],
            "max_depth": [3, 5, 7],
            "learning_rate": [0.01, 0.1, 0.2],
            "subsample": [0.8, 1.0],
            "colsample_bytree": [0.8, 1.0]
        }
    }
}

print(f" Total Models Loaded: {len(models)}")
```

```
Total Models Loaded: 9
```

Model Evaluation Run GridSearchCV for each classification

In [25]:

```
results = []
best_score = -float('inf')
best_model = None
best_name = None

# Run GridSearchCV for each classification model
for name, mp in models.items():
    print(f"\n Running GridSearchCV for {name}...")
    grid = GridSearchCV(mp['model'], param_grid=mp['param_grid'], cv=5, scoring=
    grid.fit(X_train, y_train)

    y_pred = grid.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted')
    rec = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
```

```
    results.append({
        'Model': name,
        'Best Params': grid.best_params_,
        'Accuracy': acc,
        'Precision': prec,
        'Recall': rec,
        'F1 Score': f1
    })

    print(f"{name} Best CV Score (Accuracy): {grid.best_score_:.4f}")
    print(f"{name} Best Params: {grid.best_params_}")
    print(f"{name} Classification Report:\n{classification_report(y_test, y_pred


    if grid.best_score_ > best_score:
        best_score = grid.best_score_
        best_model = grid.best_estimator_
        best_name = name

# Output summary
results_df = pd.DataFrame(results)
print("\n Summary Results:")
print(results_df)

print(f"\n Best Model: {best_name}")
print(f" Best CV Score (Accuracy): {best_score:.4f}")
```

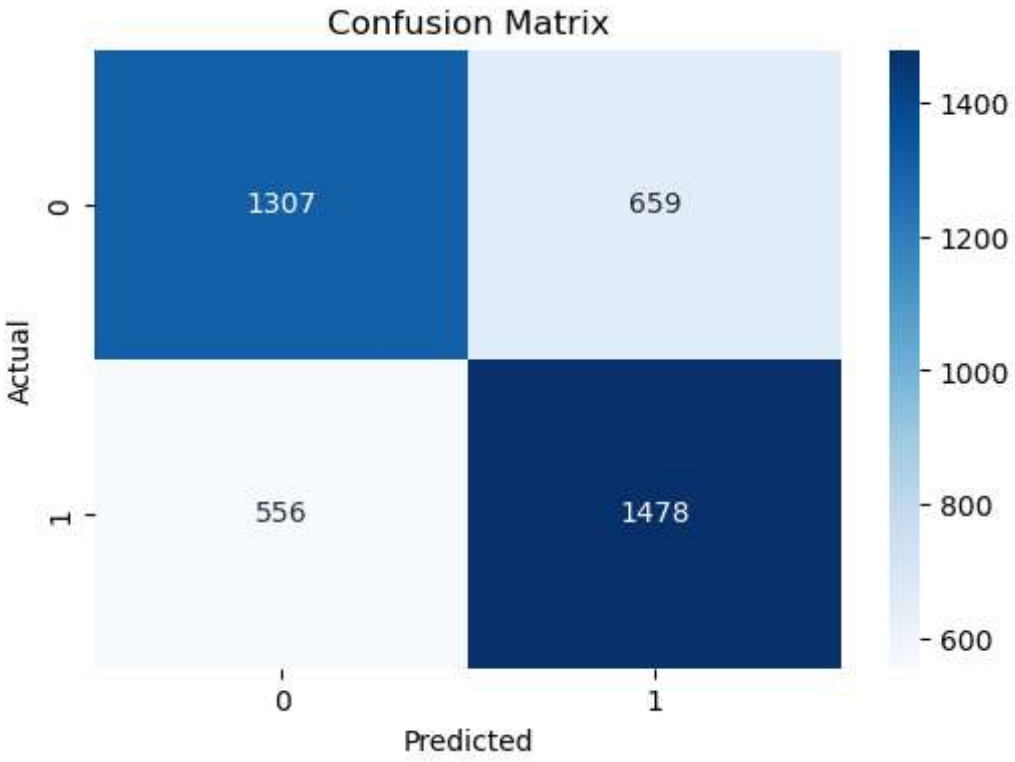
```
 Running GridSearchCV for Logistic Regression...
Logistic Regression Best CV Score (Accuracy): 0.5496
Logistic Regression Best Params: {'C': 1, 'penalty': 'l2'}
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.52      0.97      0.68      1966
           1       0.83      0.14      0.24      2034

    accuracy                           0.55      4000
   macro avg       0.68      0.56      0.46      4000
weighted avg       0.68      0.55      0.46      4000


 Running GridSearchCV for Decision Tree...
Decision Tree Best CV Score (Accuracy): 0.6959
Decision Tree Best Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_sample
s_split': 2}
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.69      0.70      1966
           1       0.71      0.72      0.71      2034

    accuracy                           0.71      4000
   macro avg       0.71      0.71      0.71      4000
weighted avg       0.71      0.71      0.71      4000


 Running GridSearchCV for Random Forest...
Random Forest Best CV Score (Accuracy): 0.7043
Random Forest Best Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_sample
s_split': 2, 'n_estimators': 200}
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.67      0.70      1966
           1       0.70      0.77      0.74      2034

    accuracy                           0.72      4000
   macro avg       0.72      0.72      0.72      4000
weighted avg       0.72      0.72      0.72      4000


 Running GridSearchCV for Gradient Boosting...
Gradient Boosting Best CV Score (Accuracy): 0.6593
Gradient Boosting Best Params: {'learning_rate': 0.2, 'max_depth': 5, 'n_estimato
rs': 200}
Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.66      0.66      1966
           1       0.68      0.68      0.68      2034

    accuracy                           0.67      4000
   macro avg       0.67      0.67      0.67      4000
weighted avg       0.67      0.67      0.67      4000


 Running GridSearchCV for AdaBoost...
```

```
AdaBoost Best CV Score (Accuracy): 0.5952
AdaBoost Best Params: {'learning_rate': 1.0, 'n_estimators': 200}
AdaBoost Classification Report:
              precision    recall  f1-score   support

           0       0.58      0.67      0.62      1966
           1       0.62      0.53      0.57      2034

    accuracy                           0.60      4000
   macro avg       0.60      0.60      0.60      4000
weighted avg       0.60      0.60      0.60      4000


 Running GridSearchCV for Extra Trees...
Extra Trees Best CV Score (Accuracy): 0.7094
Extra Trees Best Params: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_sp
lit': 5, 'n_estimators': 200}
Extra Trees Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.69      0.71      1966
           1       0.72      0.75      0.73      2034

    accuracy                           0.72      4000
   macro avg       0.72      0.72      0.72      4000
weighted avg       0.72      0.72      0.72      4000


 Running GridSearchCV for K-Nearest Neighbors...
K-Nearest Neighbors Best CV Score (Accuracy): 0.6875
K-Nearest Neighbors Best Params: {'n_neighbors': 9, 'p': 1, 'weights': 'distanc
e'}
K-Nearest Neighbors Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.69      0.70      1966
           1       0.71      0.72      0.71      2034

    accuracy                           0.70      4000
   macro avg       0.70      0.70      0.70      4000
weighted avg       0.70      0.70      0.70      4000


 Running GridSearchCV for MLP Classifier...
MLP Classifier Best CV Score (Accuracy): 0.5799
MLP Classifier Best Params: {'activation': 'tanh', 'hidden_layer_sizes': (100,),
'max_iter': 200, 'solver': 'adam'}
MLP Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.55      0.79      0.65      1966
           1       0.65      0.38      0.48      2034

    accuracy                           0.58      4000
   macro avg       0.60      0.58      0.56      4000
weighted avg       0.60      0.58      0.56      4000


 Running GridSearchCV for XGBoost...
XGBoost Best CV Score (Accuracy): 0.6808
```

```
XGBoost Best Params: {'colsample_bytree': 1.0, 'learning_rate': 0.2, 'max_depth':
7, 'n_estimators': 200, 'subsample': 0.8}
XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.66      0.68      1966
           1       0.69      0.73      0.71      2034

    accuracy                           0.70      4000
   macro avg       0.70      0.70      0.70      4000
weighted avg       0.70      0.70      0.70      4000


 Summary Results:
                 Model                                          Best Params  \
0  Logistic Regression                           {'C': 1, 'penalty': 'l2'}
1        Decision Tree  {'max_depth': None, 'min_samples_leaf': 1, 'mi...
2        Random Forest  {'max_depth': None, 'min_samples_leaf': 1, 'mi...
3    Gradient Boosting  {'learning_rate': 0.2, 'max_depth': 5, 'n_esti...
4             AdaBoost        {'learning_rate': 1.0, 'n_estimators': 200}
5          Extra Trees  {'max_depth': 20, 'min_samples_leaf': 1, 'min_...
6  K-Nearest Neighbors  {'n_neighbors': 9, 'p': 1, 'weights': 'distance'}
7       MLP Classifier  {'activation': 'tanh', 'hidden_layer_sizes': (...
8              XGBoost  {'colsample_bytree': 1.0, 'learning_rate': 0.2...

   Accuracy  Precision   Recall  F1 Score
0   0.54800   0.677657  0.54800  0.455323
1   0.70600   0.706004  0.70600  0.705891
2   0.71875   0.720197  0.71875  0.717924
3   0.67050   0.670480  0.67050  0.670487
4   0.59850   0.602030  0.59850  0.596583
5   0.72250   0.722771  0.72250  0.722235
6   0.70475   0.704719  0.70475  0.704711
7   0.58075   0.601276  0.58075  0.563146
8   0.69625   0.696506  0.69625  0.695915

 Best Model: Extra Trees
 Best CV Score (Accuracy): 0.7094
```

In [27]:

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=set(y_test), ytic
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



Save The Model

In [29]:
```
feature_columns = X.columns.to_list()
joblib.dump(feature_columns,'Employee Promotion Prediction.joblib')
```

Out[29]:
```
['Employee Promotion Prediction.joblib']
```

In [30]:
```
joblib.dump(best_model,'bestmodel Employee Promotion Prediction.joblib')
```

Out[30]:
```
['bestmodel Employee Promotion Prediction.joblib']
```

If you have any suggestions, please DM me.

Even a small message from you can make a big impact on my career

I am Arun