Introduction

In the absence of proprietary data from WPH, our strategy involves leveraging a surrogate dataset from a comparable corporate environment. This approach will allow us to showcase potential analytical solutions that could be adapted for WPH.

The dataset in use is sourced from the IBM HR Analytics Employee Attrition & Performance dataset available on Kaggle. It provides a rich simulation for our analysis, offering insights that could inform the development of WPH-specific models.

For a comprehensive overview of the dataset, please refer to: IBM HR Analytics Employee Attrition & Performance (https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset/data)

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

# Load the dataset
data = pd.read_csv('IBM HR Employee Attrition.csv')

# Identifying columns where all values are the same
constant_columns = [col for col in data.columns if data[col].nunique() == 1]

# Display constant columns
print(constant_columns)
print(data.shape)
```

```
['EmployeeCount', 'Over18', 'StandardHours']
(1470, 35)
```

Prior to delving into the nuances of data analysis, it is crucial to refine the dataset by removing features that offer no discernible variation. Features with uniform values across all entries fail to contribute to the predictive power of models and are hence redundant. In our current dataset, the following columns exhibit this characteristic:

'EmployeeCount' 'Over18' 'StandardHours'

These columns will be excluded from the dataset to ensure that our subsequent analyses and models are influenced only by variables that provide meaningful information.

In [2]:
```
data_cleaned = data.drop(columns=constant_columns + ['EmployeeNumber'])

# Display the first few rows of the cleaned dataset to confirm changes
display(data_cleaned.head())
print(data_cleaned.shape)
#A total of 4 columns will be dropped
```

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

5 rows × 31 columns

```
(1470, 31)
```

In [3]:
```python
# Check for null values in all columns
null_values = data_cleaned.isnull().sum()
print(null_values)

# Performing value_counts on each column
value_counts_summary = {column: data_cleaned[column].value_counts() for column in data_cleaned.columns}
print(value_counts_summary)
```

```
Travel_Rarely        1043
Travel_Frequently     277
Non-Travel            150
Name: count, dtype: int64, 'DailyRate': DailyRate
691     6
408     5
530     5
1329    5
1082    5
       ..
650     1
279     1
316     1
314     1
628     1
Name: count, Length: 886, dtype: int64, 'Department': Department
Research & Development    961
Sales                     446
Human Resources            63
Name: count, dtype: int64, 'DistanceFromHome': DistanceFromHome
```

The dataset has no missing values and inconsistencies. We can proceed on the next steps of our anlaysis.

Comprehensive Analytical Framework

The analysis will be meticulously segmented into five distinct sections, each tailored to undergo a dedicated process of data preprocessing, dataset partitioning, model development, and rigorous evaluation. This structured segmentation is conceived to address specific focal points of the analysis:

1. **Predicting Employee Attrition:** This segment aims to model and understand the factors influencing employee turnover, which is paramount in assessing the alignment between WPH's objectives and the aspirations of potential talents, particularly in the context of internal recruitment.
2. **Predicting Relationship Satisfaction:** In this section, we forecast the level of employees' contentment with their workplace relationships, a facet that significantly impacts their collaboration and team dynamics.
3. **Predicting Job Satisfaction:** This analysis will delve into the factors contributing to employees' job contentment, a key indicator of their engagement and overall job performance.
4. **Predicting Work-Life Balance:** We will explore the elements that balance professional responsibilities with personal life, an essential aspect of a healthy work environment that fosters employee well-being.
5. **Predicting Job Roles:** The final segment focuses on identifying the most suitable job roles for employees based on a set of influential variables. This will involve an in-depth feature analysis and elimination process to pinpoint the key attributes that best define the alignment of job roles with employee skills and competencies.

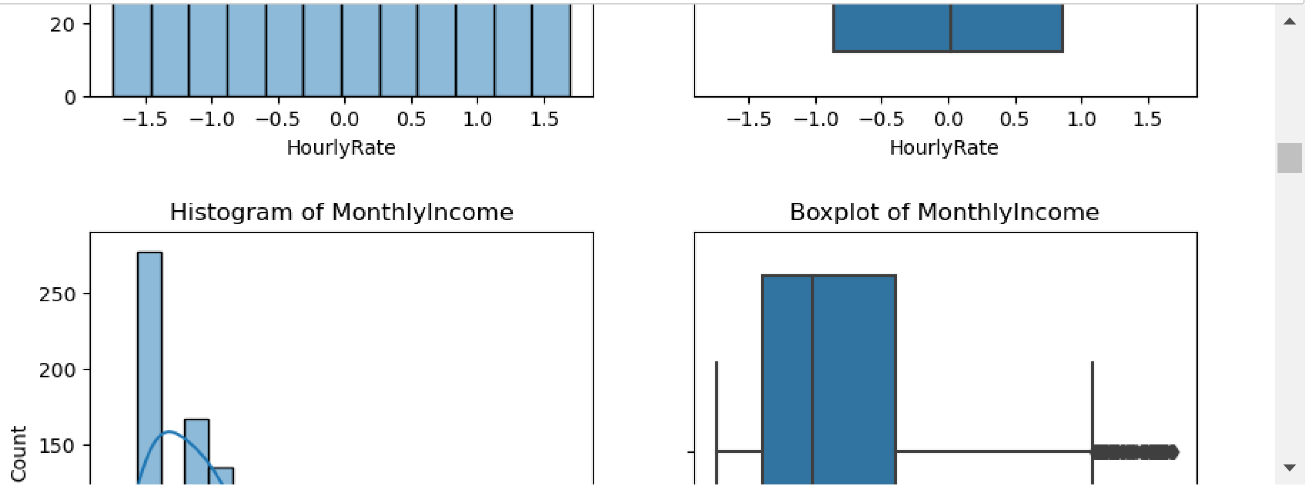
Each section will not only contribute to a holistic understanding of employee dynamics within the organization but will also serve as a strategic tool in evaluating the soft skills and capabilities of talents. A well-balanced work life, coupled with job satisfaction, are indicators of an employee's potential and adaptability, as underscored by recent studies Susanto et al., 2022 (https://consensus.app/papers/worklife-balance-satisfaction-performance-smes-susanto/9156c1754a74591e911a0d6449df09a7). The in-depth examination in section five will further illuminate the pathways to optimal job role assignments, ensuring a harmonious and productive workplace.

Section 1: Predicting Employee Attrition

In [4]:
```python
# Spitting the dataset
from sklearn.model_selection import train_test_split

# Define the target variable 'y' and features 'X'
y = data_cleaned['Attrition']
X = data_cleaned.drop('Attrition', axis=1)

# Splitting the dataset into training and testing sets, stratified based on the 'Attrition' column
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Displaying the shapes of the training and testing sets
(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Out[4]:
```
((1176, 30), (294, 30), (1176,), (294,))
```

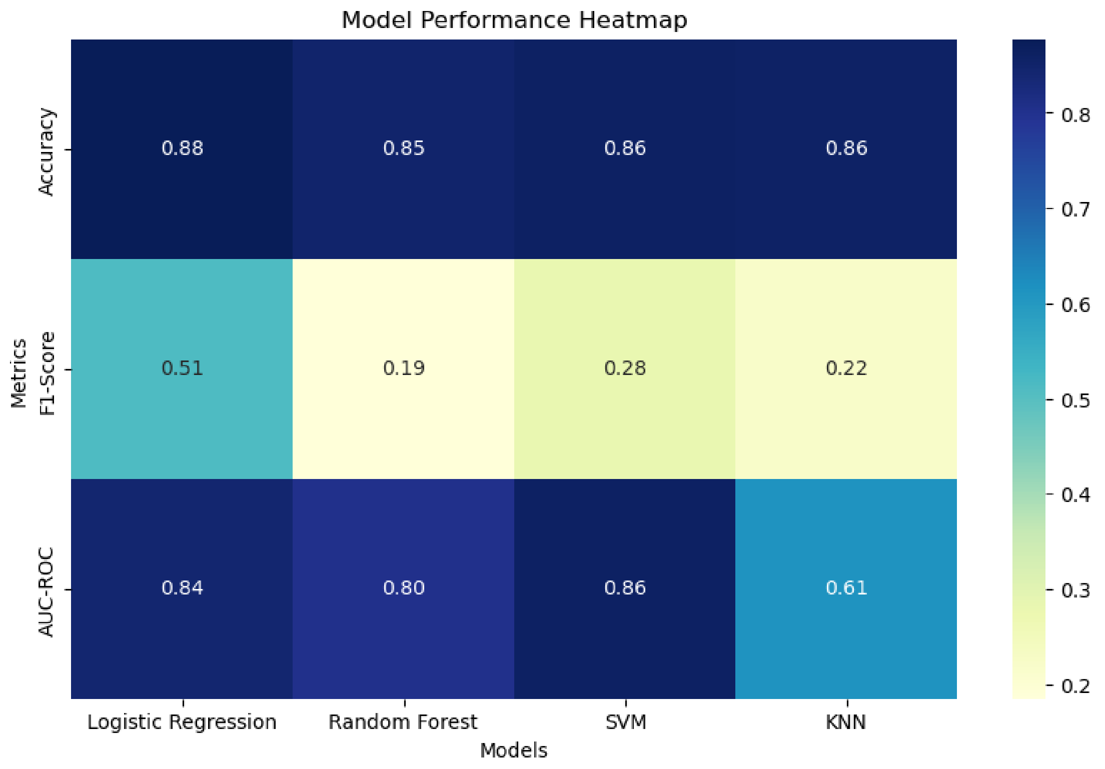
In [5]:
```python
# Standardizing numerical vairables and one hot encoding categorical variables
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
import matplotlib.pyplot as plt
import seaborn as sns

# List of numerical and categorical columns
numerical_cols = ['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate',
                  'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
                  'PercentSalaryHike', 'TotalWorkingYears', 'TrainingTimesLastYear',
                  'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
                  'YearsWithCurrManager']
categorical_cols = ['BusinessTravel', 'Department', 'Education', 'EducationField',
                    'EnvironmentSatisfaction', 'Gender', 'JobInvolvement', 'JobLevel',
                    'JobRole', 'JobSatisfaction', 'MaritalStatus', 'PerformanceRating',
                    'RelationshipSatisfaction', 'StockOptionLevel', 'WorkLifeBalance','OverTime']

# Create a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Fit and transform the training data
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

# Convert processed data back to DataFrame for visualization
column_names = preprocessor.named_transformers_['num'].get_feature_names_out(numerical_cols)
column_names = np.append(column_names, preprocessor.named_transformers_['cat'].get_feature_names_out())
X_train_processed_df = pd.DataFrame(X_train_processed, columns=column_names)

# Visualization of Numerical Variables (after standardization but before removing outliers)
for col in column_names[:len(numerical_cols)]:
    plt.figure(figsize=(10, 4))

    # Histogram
    plt.subplot(1, 2, 1)
    sns.histplot(X_train_processed_df[col], kde=True)
    plt.title(f'Histogram of {col}')

    # Boxplot
    plt.subplot(1, 2, 2)
    sns.boxplot(x=X_train_processed_df[col])
    plt.title(f'Boxplot of {col}')

    plt.show()
```



We will train and evaluate models using Logistic Regression, Random Forest Classifier, SVM, and KNN on data that includes outliers. This approach aims to assess the impact of outliers on each model, leveraging Random Forest's known robustness to outliers as a benchmark for comparison. Through this method, we will determine the significance of outliers on model performance based on specific metrics.

In [6]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
from sklearn.preprocessing import LabelEncoder

# Encode the labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Initialize the models
log_reg = LogisticRegression(max_iter=10000)  # Increased max_iter
rf_clf = RandomForestClassifier(n_estimators=500)
svm_clf = SVC(probability=True)  # Set probability=True for AUC-ROC
knn_clf = KNeighborsClassifier()

# Train the models
log_reg.fit(X_train_processed, y_train_encoded)
rf_clf.fit(X_train_processed, y_train_encoded)
svm_clf.fit(X_train_processed, y_train_encoded)
knn_clf.fit(X_train_processed, y_train_encoded)

models = [log_reg, rf_clf, svm_clf, knn_clf]
model_names = ["Logistic Regression", "Random Forest", "SVM", "KNN"]
performance_metrics = {"Accuracy": accuracy_score, "F1-Score": f1_score, "AUC-ROC": roc_auc_score}
results = {name: [] for name in model_names}

for model, name in zip(models, model_names):
    predictions = model.predict(X_test_processed)
    probabilities = model.predict_proba(X_test_processed)[:, 1]
    for metric_name, metric_func in performance_metrics.items():
        if metric_name == "AUC-ROC":
            score = metric_func(y_test_encoded, probabilities)
        else:
            score = metric_func(y_test_encoded, predictions)
        results[name].append(score)

# Convert results to DataFrame
results_df = pd.DataFrame(results, index=performance_metrics.keys())
```

In [7]:
```
plt.figure(figsize=(10, 6))
sns.heatmap(results_df, annot=True, cmap="YlGnBu", fmt=".2f")
plt.title("Model Performance Heatmap")
plt.ylabel("Metrics")
plt.xlabel("Models")
plt.show()
```



Model Performance Analysis

Upon visual inspection of the performance heatmap for four machine learning models, we derive the following insights:

Accuracy:

- Logistic Regression and SVM exhibit the highest accuracy (0.88 - 0.86), indicating their superior capability in correct predictions of both classes.
- Random Forest follows closely with an accuracy of ~0.84, while KNN matches this but may be less reliable due to its lower AUC-ROC score.

F1-Score:

- Logistic Regression leads with an F1-score of 0.51, suggesting it has the best balance between precision and recall among the models.
- The other models have notably lower F1-scores, with Random Forest at 0.18, SVM at 0.28, and KNN at 0.22, which could indicate a less effective balance in classification performance for positive and negative instances.

AUC-ROC:

- SVM shows the best AUC-ROC score at 0.86, with Logistic Regression close behind at 0.84, both indicating strong discriminative abilities between class probabilities.
- Random Forest's AUC-ROC score is 0.81, which is respectable but less than the leading SVM and Logistic Regression.
- KNN lags with an AUC-ROC score of 0.61, implying a comparatively weaker capability in distinguishing between the classes.

Conclusions:

- **Logistic Regression** demonstrates robustness across all metrics, with the highest F1-score, indicative of its strong predictive performance and class balance.
- **SVM** shows competitive accuracy and the best AUC-ROC, suggesting excellent class separation capabilities.
- **Random Forest** and **KNN**, while exhibiting decent accuracy, fall short in terms of F1-score and AUC-ROC, signaling potential limitations in their precision, recall, or both.

Insights:

- The addition of the `OverTime` feature appears to have influenced the models' capabilities, especially boosting the performance of Logistic Regression in terms of F1-score.
- Logistic Regression's performance, strong across all metrics, may indicate that the linearity assumption holds well for this dataset, and that features including `OverTime` contribute significantly to the prediction of the target variable.
- Given SVM's performance, especially in AUC-ROC, it is evident that the margin-based approach of SVM is effective for this particular problem, even in the presence of potential outliers and class imbalance.
- The lower performance of KNN in AUC-ROC could be attributed to its sensitivity to feature scales and distribution, reinforcing the importance of feature selection and preprocessing.

Note: To ensure these models' robustness, further validation such as cross-validation should be employed. Hyperparameter tuning could potentially enhance model performance. In addition, the business context and the cost of different types of classification errors should be carefully

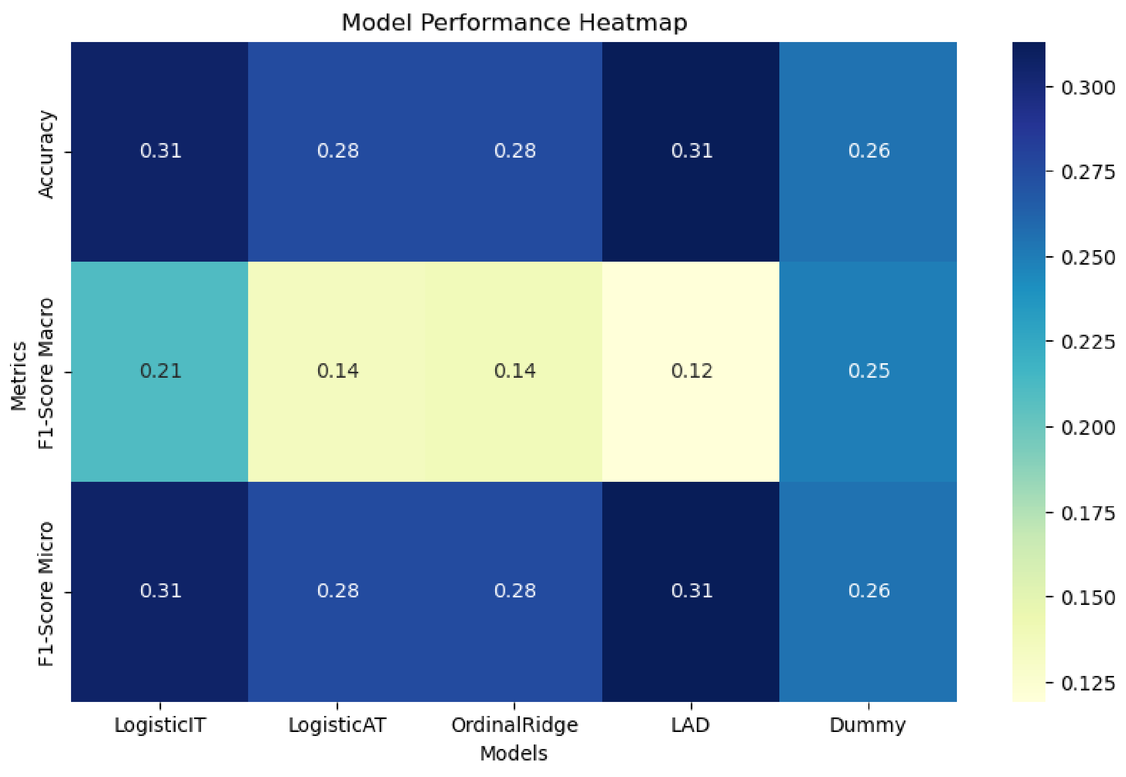
Section 2: Predicting Employee Relationship Satisfaction

In [8]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Define the target variable 'y' and features 'X'
y = data_cleaned['RelationshipSatisfaction']
X = data_cleaned.drop(['RelationshipSatisfaction'], axis=1)

# Numerical and categorical columns as provided
numerical_cols = ['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate',
                  'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
                  'PercentSalaryHike', 'TotalWorkingYears', 'TrainingTimesLastYear',
                  'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
                  'YearsWithCurrManager']
categorical_cols = ['BusinessTravel', 'Department', 'Education', 'EducationField',
                    'EnvironmentSatisfaction', 'Gender', 'JobInvolvement', 'JobLevel',
                    'JobRole', 'JobSatisfaction', 'MaritalStatus', 'PerformanceRating',
                    'Attrition', 'StockOptionLevel', 'WorkLifeBalance', 'OverTime']

# Split the dataset into training and testing sets with stratification
X_train_rel, X_test_rel, y_train_rel, y_test_rel = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Create a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Fit and transform the training data
X_train_rel_processed = preprocessor.fit_transform(X_train_rel)

# Transform the test data
X_test_rel_processed = preprocessor.transform(X_test_rel)
```

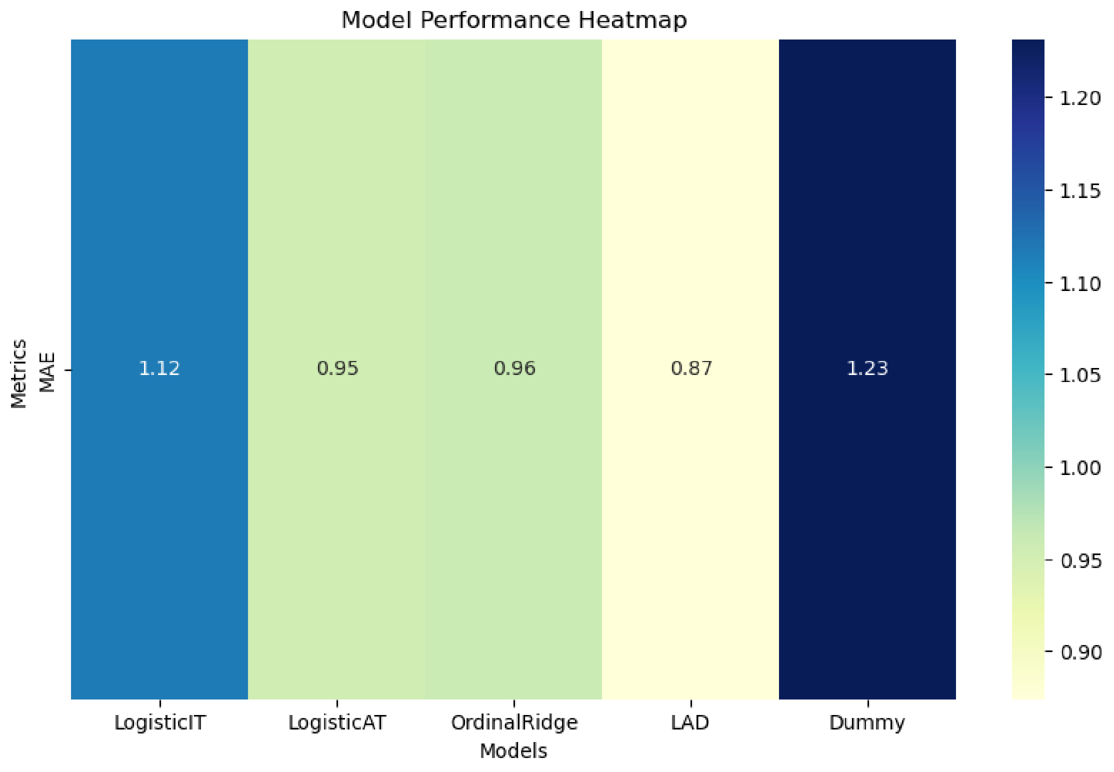
Since Relationship Satisfaction is of ordinal type of data, instead of using classifier models, we will be using ordinal regression models, for that cases we will be using the "mord" library to handle such cases. We will be using MAE, F1 score micro and macro as our metrics. To evaluate the performance we will be comparing our trained model to a dummy model which will give random predictions as compared to the models we have trained.

In [9]:
```python
import mord
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, f1_score, accuracy_score
from sklearn.dummy import DummyClassifier

models = {
    'LogisticIT': mord.LogisticIT(alpha=1.0),
    'LogisticAT': mord.LogisticAT(alpha=1.0),
    'OrdinalRidge': mord.OrdinalRidge(alpha=1.0),
    'LAD': mord.LAD(dual=True),
    'Dummy':DummyClassifier(strategy='uniform', random_state=42)
}

for name, model in models.items():
    model.fit(X_train_rel_processed, y_train_rel)


accuracy_f1_results = {}
mae_results = {}

for name, model in models.items():
    predictions = model.predict(X_test_rel_processed)
    accuracy_f1_results[name] = {
        'Accuracy': accuracy_score(y_test_rel, predictions),
        'F1-Score Macro': f1_score(y_test_rel, predictions, average='macro'),
        'F1-Score Micro': f1_score(y_test_rel, predictions, average='micro')
    }
    mae_results[name] = {
        'MAE': mean_absolute_error(y_test_rel, predictions)
    }

# Convert the dictionaries to dataframes
accuracy_f1_df = pd.DataFrame(accuracy_f1_results)
mae_df = pd.DataFrame(mae_results)
```

In [10]:
```python
# Visualize the accuracy_f1_results in a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(accuracy_f1_df, annot=True, cmap="YlGnBu", fmt=".2f")
plt.title("Model Performance Heatmap")
plt.ylabel("Metrics")
plt.xlabel("Models")
plt.show()
```



```
In [11]: # Visualize the mae_results in a heatmap
         plt.figure(figsize=(10, 6))
         sns.heatmap(mae_df, annot=True, cmap="YlGnBu", fmt=".2f")
         plt.title("Model Performance Heatmap")
         plt.ylabel("Metrics")
         plt.xlabel("Models")
         plt.show()
```



Upon reviewing the initial model performances, it has become evident that the predictive accuracy does not surpass the baseline of random chance in class differentiation. This indicates an opportunity for us to reallocate our analytical efforts towards other key areas of the dataset that may yield more actionable insights.

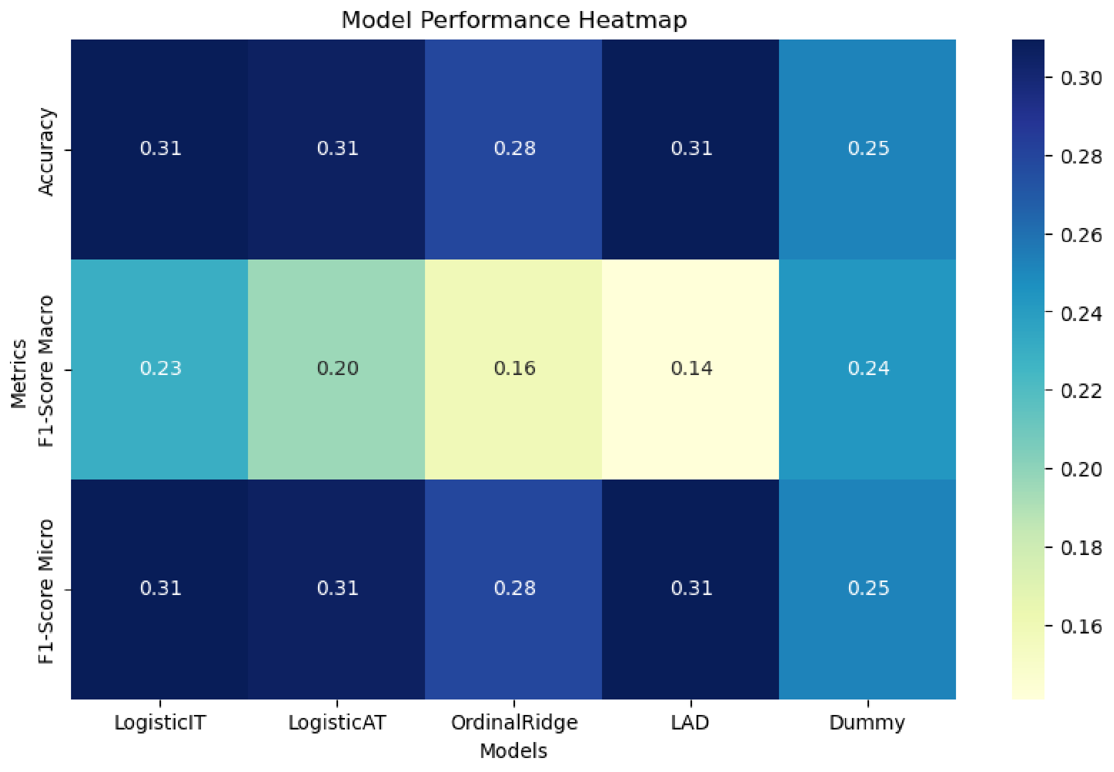
We will proceed to concentrate on developing models to predict other crucial variables that could be instrumental in understanding the dynamics within the organization.

Section 3: Predicting Job Satisfaction

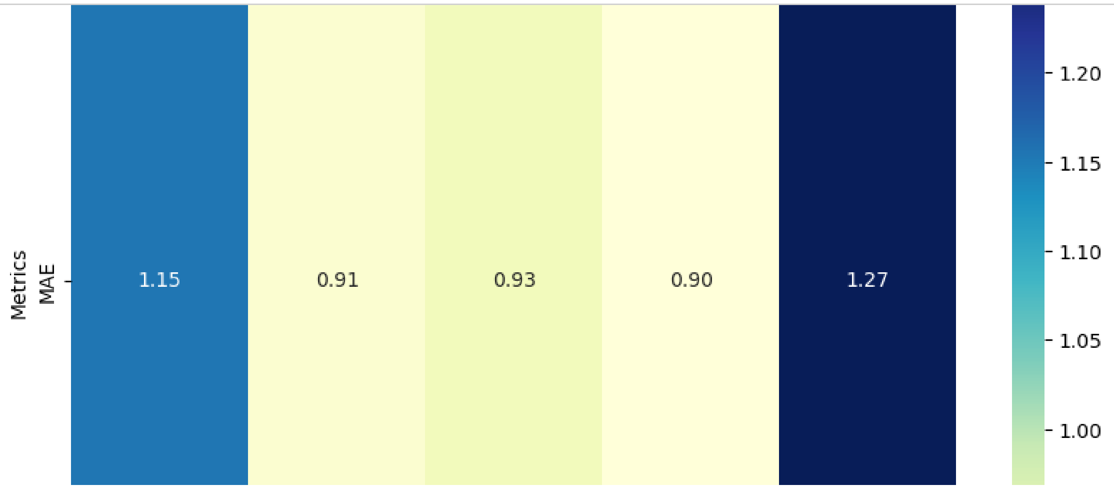
In [12]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Define the target variable 'y' and features 'X'
y = data_cleaned['JobSatisfaction']
X = data_cleaned.drop(['JobSatisfaction'], axis=1)

# Numerical and categorical columns as provided
numerical_cols = ['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate',
                  'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
                  'PercentSalaryHike', 'TotalWorkingYears', 'TrainingTimesLastYear',
                  'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
                  'YearsWithCurrManager']
categorical_cols = ['BusinessTravel', 'Department', 'Education', 'EducationField',
                    'EnvironmentSatisfaction', 'Gender', 'JobInvolvement', 'JobLevel',
                    'JobRole', 'RelationshipSatisfaction', 'MaritalStatus', 'PerformanceRating',
                    'Attrition', 'StockOptionLevel', 'WorkLifeBalance', 'OverTime']

# Split the dataset into training and testing sets with stratification
X_train_rel, X_test_rel, y_train_rel, y_test_rel = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Create a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Fit and transform the training data
X_train_rel_processed = preprocessor.fit_transform(X_train_rel)

# Transform the test data
X_test_rel_processed = preprocessor.transform(X_test_rel)
```

In [13]:
```
models = {
    'LogisticIT': mord.LogisticIT(alpha=1.0),
    'LogisticAT': mord.LogisticAT(alpha=1.0),
    'OrdinalRidge': mord.OrdinalRidge(alpha=1.0),
    'LAD': mord.LAD(dual=True),
    'Dummy':DummyClassifier(strategy='uniform', random_state=42)
}

for name, model in models.items():
    model.fit(X_train_rel_processed, y_train_rel)


accuracy_f1_results = {}
mae_results = {}

for name, model in models.items():
    predictions = model.predict(X_test_rel_processed)
    accuracy_f1_results[name] = {
        'Accuracy': accuracy_score(y_test_rel, predictions),
        'F1-Score Macro': f1_score(y_test_rel, predictions, average='macro'),
        'F1-Score Micro': f1_score(y_test_rel, predictions, average='micro')
    }
    mae_results[name] = {
        'MAE': mean_absolute_error(y_test_rel, predictions)
    }

# Convert the dictionaries to dataframes
accuracy_f1_df = pd.DataFrame(accuracy_f1_results)
mae_df = pd.DataFrame(mae_results)
```

In [14]:
```python
# Visualize the accuracy_f1_results in a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(accuracy_f1_df, annot=True, cmap="YlGnBu", fmt=".2f")
plt.title("Model Performance Heatmap")
plt.ylabel("Metrics")
plt.xlabel("Models")
plt.show()
```



In [15]:
```python
# Visualize the mae_results in a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(mae_df, annot=True, cmap="YlGnBu", fmt=".2f")
plt.title("Model Performance Heatmap")
plt.ylabel("Metrics")
plt.xlabel("Models")
plt.show()
```



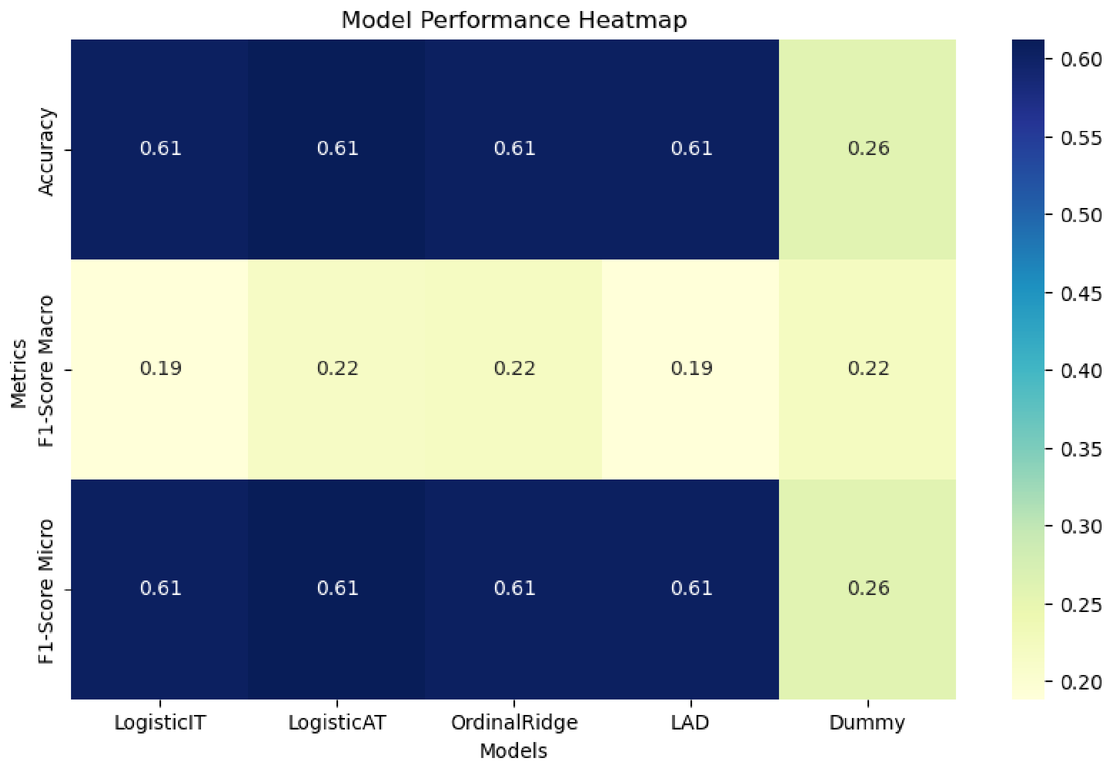
The visualizations indicate that the models trained to predict job satisfaction are not substantially outperforming a model that guesses randomly when it comes to distinguishing between classes. Despite efforts in feature engineering and hyperparameter tuning, the results have not shown significant improvement. Consequently, we will proceed to analyze the prediction of work-life balance.

Section 4: Predicting Work Life Balance

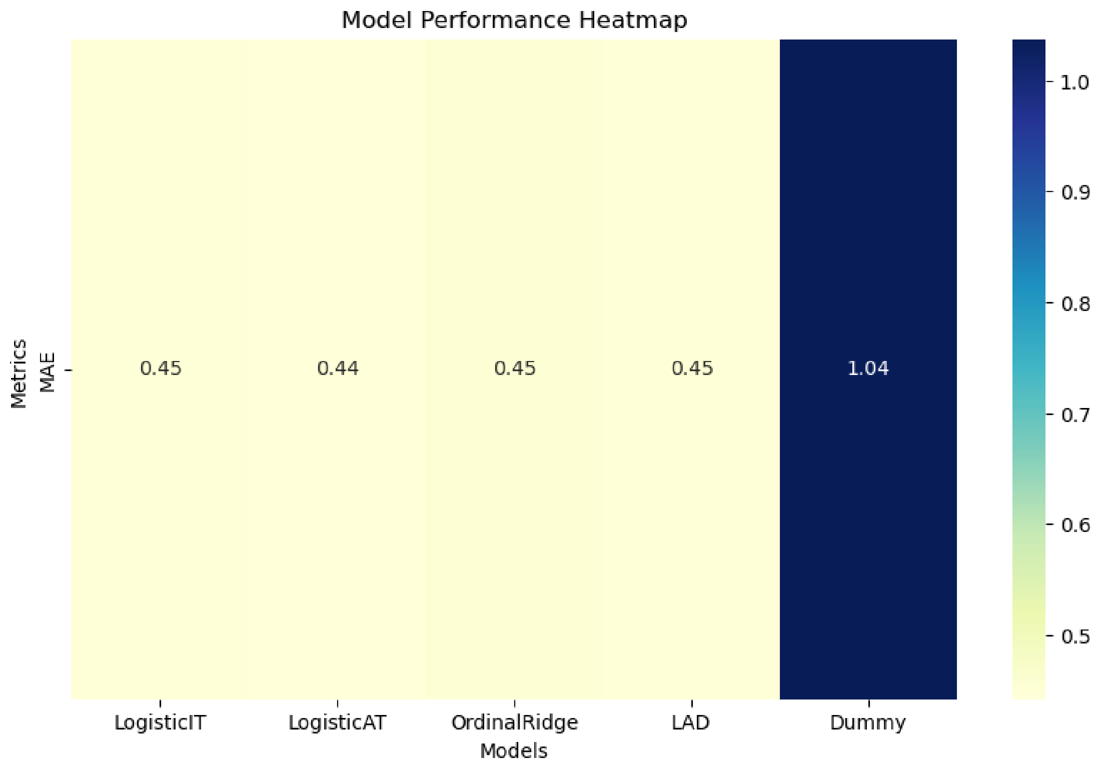
In [16]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Define the target variable 'y' and features 'X'
y = data_cleaned['WorkLifeBalance']
X = data_cleaned.drop(['WorkLifeBalance'], axis=1)

# Numerical and categorical columns as provided
numerical_cols = ['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate',
                  'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
                  'PercentSalaryHike', 'TotalWorkingYears', 'TrainingTimesLastYear',
                  'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
                  'YearsWithCurrManager']
categorical_cols = ['BusinessTravel', 'Department', 'Education', 'EducationField',
                    'EnvironmentSatisfaction', 'Gender', 'JobInvolvement', 'JobLevel',
                    'JobRole', 'RelationshipSatisfaction', 'MaritalStatus', 'PerformanceRating',
                    'Attrition', 'StockOptionLevel', 'JobSatisfaction', 'OverTime']

# Split the dataset into training and testing sets with stratification
X_train_rel, X_test_rel, y_train_rel, y_test_rel = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Create a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Fit and transform the training data
X_train_rel_processed = preprocessor.fit_transform(X_train_rel)

# Transform the test data
X_test_rel_processed = preprocessor.transform(X_test_rel)
```

In [17]:
```python
models = {
    'LogisticIT': mord.LogisticIT(alpha=1.0),
    'LogisticAT': mord.LogisticAT(alpha=1.0),
    'OrdinalRidge': mord.OrdinalRidge(alpha=1.0),
    'LAD': mord.LAD(dual=True),
    'Dummy':DummyClassifier(strategy='uniform', random_state=42)
}

for name, model in models.items():
    model.fit(X_train_rel_processed, y_train_rel)


accuracy_f1_results = {}
mae_results = {}

for name, model in models.items():
    predictions = model.predict(X_test_rel_processed)
    accuracy_f1_results[name] = {
        'Accuracy': accuracy_score(y_test_rel, predictions),
        'F1-Score Macro': f1_score(y_test_rel, predictions, average='macro'),
        'F1-Score Micro': f1_score(y_test_rel, predictions, average='micro')
    }
    mae_results[name] = {
        'MAE': mean_absolute_error(y_test_rel, predictions)
    }

# Convert the dictionaries to dataframes
accuracy_f1_df = pd.DataFrame(accuracy_f1_results)
mae_df = pd.DataFrame(mae_results)
```

```
In [18]: # Visualize the accuracy_f1_results in a heatmap
         plt.figure(figsize=(10, 6))
         sns.heatmap(accuracy_f1_df, annot=True, cmap="YlGnBu", fmt=".2f")
         plt.title("Model Performance Heatmap")
         plt.ylabel("Metrics")
         plt.xlabel("Models")
         plt.show()
```



```
In [19]: # Visualize the mae_results in a heatmap
         plt.figure(figsize=(10, 6))
         sns.heatmap(mae_df, annot=True, cmap="YlGnBu", fmt=".2f")
         plt.title("Model Performance Heatmap")
         plt.ylabel("Metrics")
         plt.xlabel("Models")
         plt.show()
```



```
The provided visualization suggests that our predictive models significantly outperform random chance in
distinguishing between classes. This enhancement in model performance bolsters our ability to accurately estimate
employees' work-life balance.
```

Section 5: Predicting Job Roles

In this section, we will focus on developing a predictive model to identify the most suitable job roles for talent within the company, aligning employee strengths with organizational needs.

In [20]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
import numpy as np
import pandas as pd

# Define the target variable 'y' and features 'X'
y = data_cleaned['JobRole']
X = data_cleaned.drop('JobRole', axis=1)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# List of numerical and categorical columns
numerical_cols = ['Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate',
                  'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
                  'PercentSalaryHike', 'TotalWorkingYears', 'TrainingTimesLastYear',
                  'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
                  'YearsWithCurrManager']
categorical_cols = ['BusinessTravel', 'Department', 'Education', 'EducationField',
                    'EnvironmentSatisfaction', 'Gender', 'JobInvolvement', 'JobLevel',
                    'JobSatisfaction', 'MaritalStatus', 'PerformanceRating',
                    'RelationshipSatisfaction', 'StockOptionLevel', 'WorkLifeBalance','OverTime']

# Select only the specified numerical and categorical columns
X_train = X_train[numerical_cols + categorical_cols]
X_test = X_test[numerical_cols + categorical_cols]

# Create a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Fit and transform the training data
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

# Get feature names from one hot encoder and numerical features
ohe_feature_names = preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_cols)
feature_names = np.concatenate([numerical_cols, ohe_feature_names])

# Encode the labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

In [21]:
```
from sklearn.feature_selection import RFECV

# Initialize RFE with a logistic regression estimator
rfe = RFECV(estimator=LogisticRegression(max_iter=10000), step=1, cv=5, scoring='accuracy')

# Fit RFE
rfe.fit(X_train_processed, y_train_encoded)

# Get the support array (True if a feature is selected, False otherwise)
selected_features = rfe.support_

# Get the ranking of features
ranking_features = rfe.ranking_

# Print the features selected by RFE
print("Features selected by RFE:")
for i, (feature, selected, rank) in enumerate(zip(feature_names, selected_features, ranking_features)):
    if selected:
        print(f"Rank {rank}: {feature}")
```

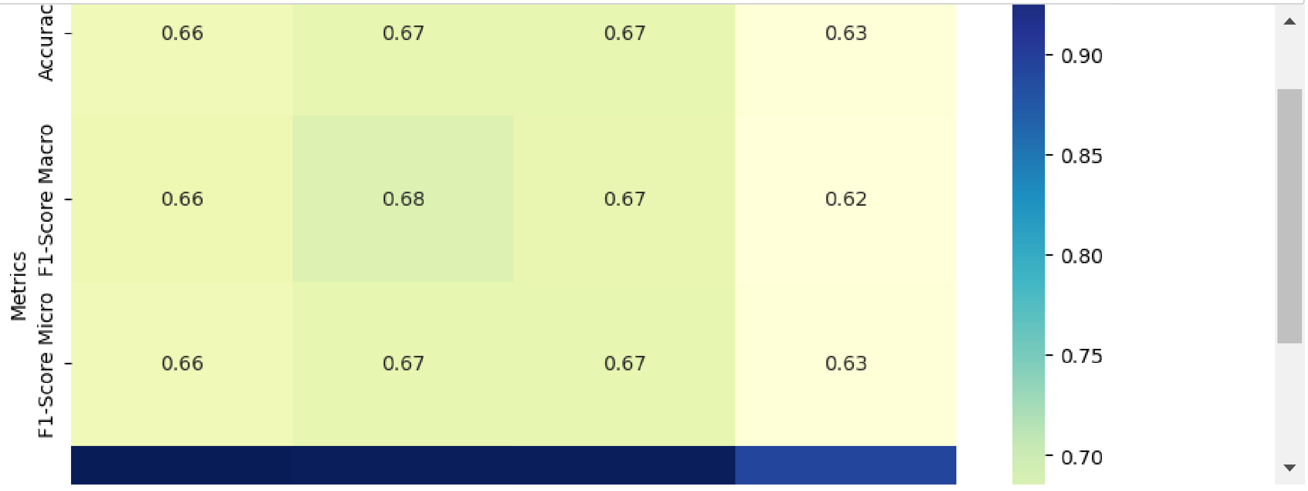
```
Features selected by RFE:
Rank 1: MonthlyIncome
Rank 1: TotalWorkingYears
Rank 1: Department_Human Resources
Rank 1: Department_Research & Development
Rank 1: Department_Sales
Rank 1: Education_5
Rank 1: EducationField_Technical Degree
Rank 1: JobLevel_1
Rank 1: JobLevel_2
Rank 1: JobLevel_3
Rank 1: JobLevel_4
Rank 1: PerformanceRating_3
Rank 1: WorkLifeBalance_1
```

For effective model training aimed at predicting job roles, we will concentrate on the most relevant features identified through prior results and enhance the feature set with additional variables informed by domain expertise. This targeted approach seeks to refine the predictive accuracy by focusing on significant predictors.

In [22]:
```
# List of numerical and categorical columns
numerical_cols = ['Education', 'JobLevel', 'MonthlyIncome', 'TotalWorkingYears' ,'PerformanceRating', 'WorkLifeBalance'
categorical_cols = ['Department', 'EducationField']

# Select only the specified numerical and categorical columns
X_train = X_train[numerical_cols + categorical_cols]
X_test = X_test[numerical_cols + categorical_cols]

# Create a column transformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Fit and transform the training data
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

# Get feature names from one hot encoder and numerical features
ohe_feature_names = preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_cols)
feature_names = np.concatenate([numerical_cols, ohe_feature_names])

# Encode the labels
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)
```

In [23]:
```
from sklearn.preprocessing import label_binarize

# Initialize the models
log_reg = LogisticRegression(max_iter=10000)  # Increased max_iter
rf_clf = RandomForestClassifier(n_estimators=500)
svm_clf = SVC(probability=True)  # Set probability=True for AUC-ROC
knn_clf = KNeighborsClassifier()

# Train the models
log_reg.fit(X_train_processed, y_train_encoded)
rf_clf.fit(X_train_processed, y_train_encoded)
svm_clf.fit(X_train_processed, y_train_encoded)
knn_clf.fit(X_train_processed, y_train_encoded)


models = [log_reg, rf_clf, svm_clf, knn_clf]
model_names = ["Logistic Regression", "Random Forest", "SVM", "KNN"]
performance_metrics = {
    "Accuracy": accuracy_score,
    "F1-Score Macro": lambda y_true, y_pred: f1_score(y_true, y_pred, average='macro'),
    "F1-Score Micro": lambda y_true, y_pred: f1_score(y_true, y_pred, average='micro'),
    # Use a One-vs-Rest approach for multiclass AUC-ROC
    "AUC-ROC": lambda y_true, y_prob: roc_auc_score(label_binarize(y_true, classes=sorted(set(y_true))), y_prob, multi_
}
results = {name: [] for name in model_names}

for model, name in zip(models, model_names):
    predictions = model.predict(X_test_processed)
    probabilities = model.predict_proba(X_test_processed)
    for metric_name, metric_func in performance_metrics.items():
        if metric_name == "AUC-ROC":
            score = metric_func(y_test_encoded, probabilities)
        else:
            score = metric_func(y_test_encoded, predictions)
        results[name].append(score)

# Convert results to DataFrame
results_df = pd.DataFrame(results, index=performance_metrics.keys())
```

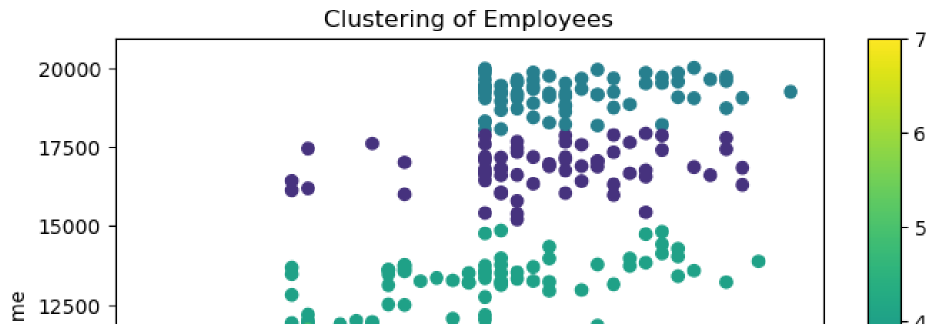
In [24]:
```
# Visualize the accuracy_f1_results in a heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(results_df, annot=True, cmap="YlGnBu", fmt=".2f")
plt.title("Model Performance Heatmap")
plt.ylabel("Metrics")
plt.xlabel("Models")
plt.show()
```



The visualization indicates that while the accuracy for multi-class classification is modest, the AUC-ROC scores are promising, suggesting that the models are capable of effectively distinguishing between classes across varying thresholds. This provides a solid foundation for further investigation into clustering approaches, which may yield insightful segments of the dataset corresponding to the most suitable job roles for each talent or employee. Moving forward with unsupervised machine learning models could uncover meaningful patterns and groupings that support talent allocation and internal mobility strategies.

In [42]:
```python
from kmodes.kprototypes import KPrototypes
from sklearn.metrics import silhouette_score
from sklearn.metrics.cluster import adjusted_rand_score

# Assuming 'data_cleaned' is your DataFrame
X = data_cleaned

# Define numerical and categorical features
numerical_features = ['MonthlyIncome', 'TotalWorkingYears']
categorical_features = ['Department', 'Education', 'EducationField', 'JobLevel', 'PerformanceRating', 'WorkLifeBalance'

# One-hot encoding categorical variables
X_encoded = pd.get_dummies(X[categorical_features])

# Combine numerical and one-hot encoded categorical data
X_combined = pd.concat([X[numerical_features], X_encoded], axis=1)

# Indices of categorical columns in the combined DataFrame
categorical_indices = list(range(len(numerical_features), len(X_combined.columns)))

# Convert DataFrame to numpy array for K-Prototypes
X_array = X_combined.values

# Apply K-Prototypes clustering
n_clusters = 8  # Total Job Roles that exist
kproto = KPrototypes(n_clusters=n_clusters, init='Cao', n_init=6, verbose=1, random_state=42)
clusters = kproto.fit_predict(X_array, categorical=categorical_indices)

# Evaluate the clustering using silhouette score
silhouette_avg = silhouette_score(X_array, clusters, metric='euclidean')
print(f'Silhouette Score: {silhouette_avg}')
print('ARI Score: {:.5f}'.format(adjusted_rand_score(X["JobRole"],clusters)))

# Add the cluster information to the original dataframe for visualization
data_cleaned['Cluster'] = clusters

# Visualize the clustering
plt.figure(figsize=(8, 6))
plt.scatter(data_cleaned['TotalWorkingYears'], data_cleaned['MonthlyIncome'], c=data_cleaned['Cluster'], cmap='viridis'
plt.title('Clustering of Employees')
plt.xlabel('Total Working Years')
plt.ylabel('Monthly Income')
plt.colorbar(label='Cluster')
plt.show()
```

```
Run: 10, iteration: 32/100, moves: 5, ncost: 498171581.4112663
Run: 10, iteration: 33/100, moves: 4, ncost: 498136790.77350676
Run: 10, iteration: 34/100, moves: 0, ncost: 498136790.77350676
Best run was number 6
Silhouette Score: 0.6044889182392996
ARI Score: 0.19561
```



We used KPrototypes model since we are trying to cluster our dataset based on both numerical and categorical variables which is more effective compared to other models such as K-Means, K-Modes, and DBScan. The model scored about 0.60 in terms of silhouette score which suggest that on average, samples are appropriately placed in their own clusters, and the clusters are well separated.To compare the actual labeling by Job Roles we also measure the ARI score that has a value of 0.20 which considered low, indicating that there is a slight agreement between the clustering and the true labels, but it is not strong. This suggests that while the clusters are separated to some extent (as indicated by the silhouette score), they do not match well with the true labels that you have for your data.

Conclusions:

1. **Employee Attrition Prediction:** The ability to predict employee turnover is pivotal for assessing organizational fit and retaining talent. Machine learning models, particularly Logistic Regression and SVM, have demonstrated commendable predictive capabilities in this regard.
2. **Employee Relationship Satisfaction:** Although the initial models did not outperform the random chance in predicting relationship satisfaction, there is potential for further model refinement or the application of alternative analytical strategies.
3. **Job Satisfaction Prediction:** The models did not significantly outperform a random-chance predictor in distinguishing between levels of job satisfaction. This suggests a need for more nuanced feature engineering or potentially different modeling techniques to capture the complexities of job satisfaction factors.
4. **Work-Life Balance Prediction:** Models showed a substantial improvement in predicting work-life balance, indicating the effectiveness of the chosen features and the potential utility of the models for estimating this aspect of employee well-being.
5. **Job Role Prediction:** The modest accuracy in predicting job roles, coupled with promising AUC-ROC scores, implies that models can effectively distinguish between different job roles at various thresholds. This presents an opportunity for clustering and unsupervised learning approaches to refine job role alignment strategies.
6. **Clustering for Job Role Alignment:** The use of KPrototypes clustering yielded a moderate silhouette score, suggesting adequately distinct clusters. However, the ARI score indicates that the clusters do not align closely with the actual job roles, signaling room for refinement in the clustering approach.

Recommendations:

1. **Modeling Strategy Optimization:** For future model developments, consider a mixed approach that combines the robustness of ensemble methods with the precision of algorithms like SVM and Logistic Regression.
2. **Feature Engineering:** Prioritize domain-informed feature engineering to enhance model performance. Utilize techniques like Recursive Feature Elimination to identify and retain the most predictive features.
3. **Model Evaluation:** Extend model evaluation beyond conventional metrics. Implement cross-validation strategies and consider the business implications of model errors.
4. **Unsupervised Learning Exploration:** Investigate unsupervised learning techniques, like clustering, to uncover intrinsic groupings within the workforce that align with job roles and employee strengths.
5. **Domain Expert Integration:** Involve domain experts in the iterative cycle of model refinement to ensure the inclusion of nuanced factors that may influence model outcomes.
6. **Data Quality and Collection:** As data is central to the modeling process, invest in data collection and cleaning strategies to improve the quality and relevance of the dataset for future analyses.