In [5]:
```
import os
import pandas as pd
```

In [7]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import ttest_ind, skew, kurtosis, pearsonr, kruskal, f_oneway
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from factor_analyzer import FactorAnalyzer
from statsmodels.formula.api import ols
import statsmodels.api as sm
```

In [9]:
```
os.chdir('/Users/rosebui/desktop/Data science project/')
```

In [11]:
```
df = pd.read_csv('HR_Data1.csv')
```

In [13]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 39 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          3000 non-null   int64
 1   FirstName                   3000 non-null   object
 2   LastName                    3000 non-null   object
 3   StartDate                   3000 non-null   object
 4   ExitDate                    1533 non-null   object
 5   Title                       3000 non-null   object
 6   Supervisor                  3000 non-null   object
 7   ADEmail                     3000 non-null   object
 8   BusinessUnit                3000 non-null   object
 9   EmployeeStatus              3000 non-null   object
 10  EmployeeType                3000 non-null   object
 11  PayZone                     3000 non-null   object
 12  EmployeeClassificationType  3000 non-null   object
 13  TerminationType             3000 non-null   object
 14  TerminationDescription      1533 non-null   object
 15  DepartmentType              3000 non-null   object
 16  Division                    3000 non-null   object
 17  DOB                         3000 non-null   object
 18  State                       3000 non-null   object
 19  JobFunctionDescription      3000 non-null   object
 20  GenderCode                  3000 non-null   object
 21  LocationCode                3000 non-null   int64
 22  RaceDesc                    3000 non-null   object
 23  MaritalDesc                 3000 non-null   object
 24  Performance Score           3000 non-null   object
 25  Current Employee Rating     3000 non-null   int64
 26  Employee ID                 3000 non-null   int64
 27  Survey Date                 3000 non-null   object
 28  Engagement Score            3000 non-null   int64
 29  Satisfaction Score          3000 non-null   int64
 30  Work-Life Balance Score     3000 non-null   int64
 31  Training Date               3000 non-null   object
 32  Training Program Name       3000 non-null   object
 33  Training Type               3000 non-null   object
 34  Training Outcome            3000 non-null   object
 35  Location                    3000 non-null   object
 36  Trainer                     3000 non-null   object
 37  Training Duration(Days)     3000 non-null   int64
 38  Training Cost               3000 non-null   float64
dtypes: float64(1), int64(8), object(30)
memory usage: 914.2+ KB
```

In [25]:

```
date_cols = ["StartDate", "ExitDate", "DOB", "Survey Date", "Training Date"]
for col in date_cols:
    df[col] = pd.to_datetime(df[col], errors="coerce")
```

In [29]:

```
df["ExitDate"].fillna(pd.NaT, inplace=True)
df.duplicated().sum()
```

```
/var/folders/xq/2xryw3gd7g7bx8y6s09b93780000gn/T/ipykernel_53975/1902752859.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df["ExitDate"].fillna(pd.NaT, inplace=True)
```

Out[29]: 0

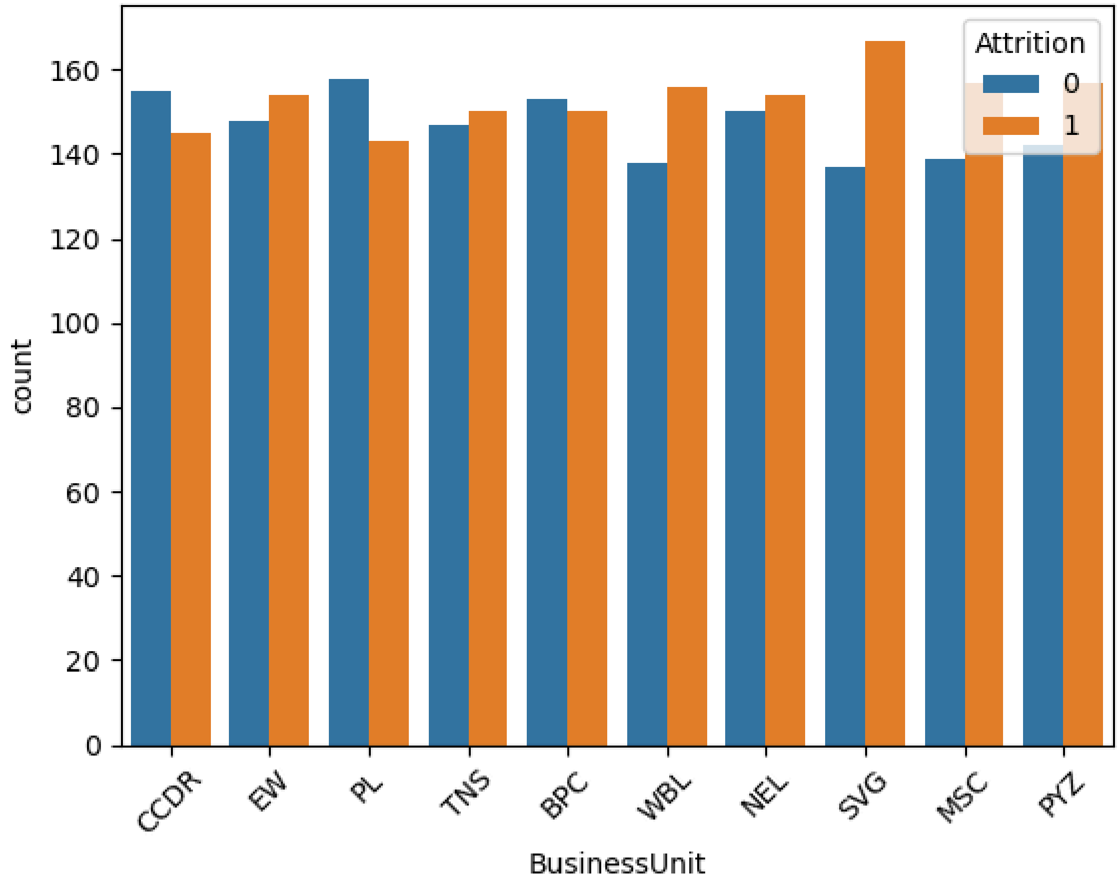
In [31]:
```python
df["TenureDays"] = (df["ExitDate"].fillna(pd.Timestamp.today()) - df["StartDate"]).dt
df["TenureYears"] = df["TenureDays"] / 365
```

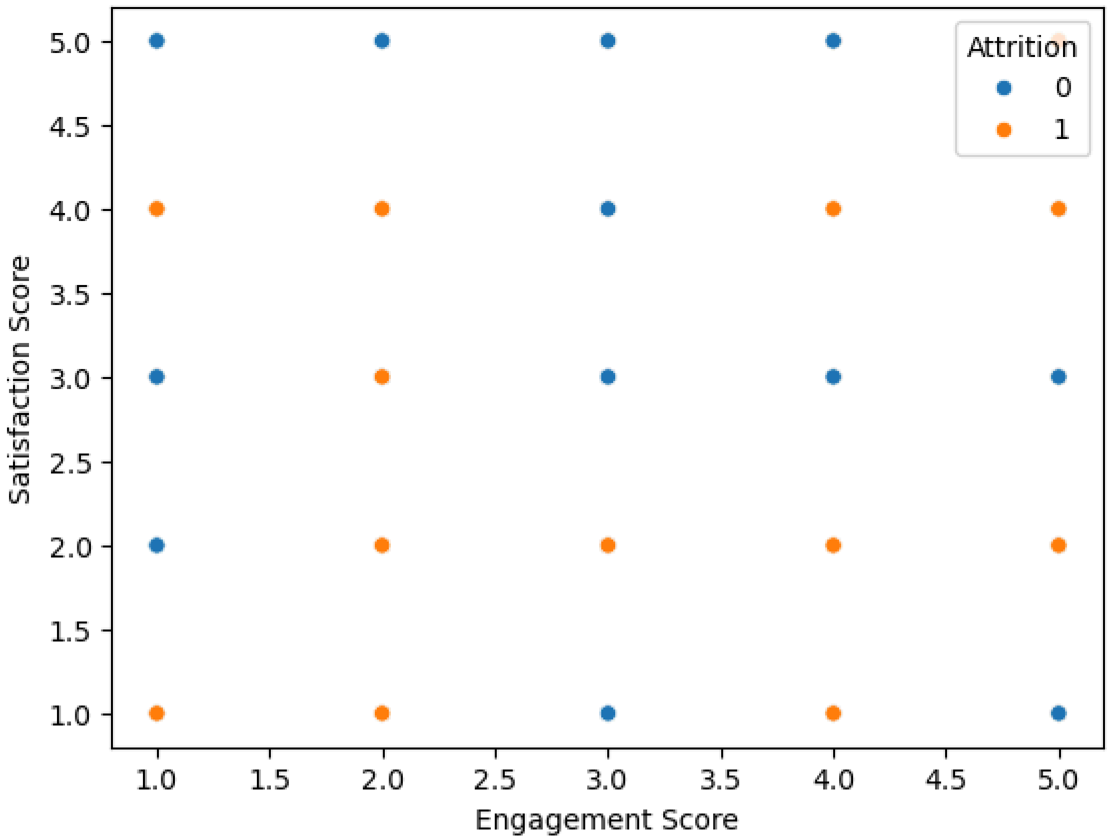
In [33]:
```python
df["Age"] = (pd.Timestamp.today() - df["DOB"]).dt.days // 365
```

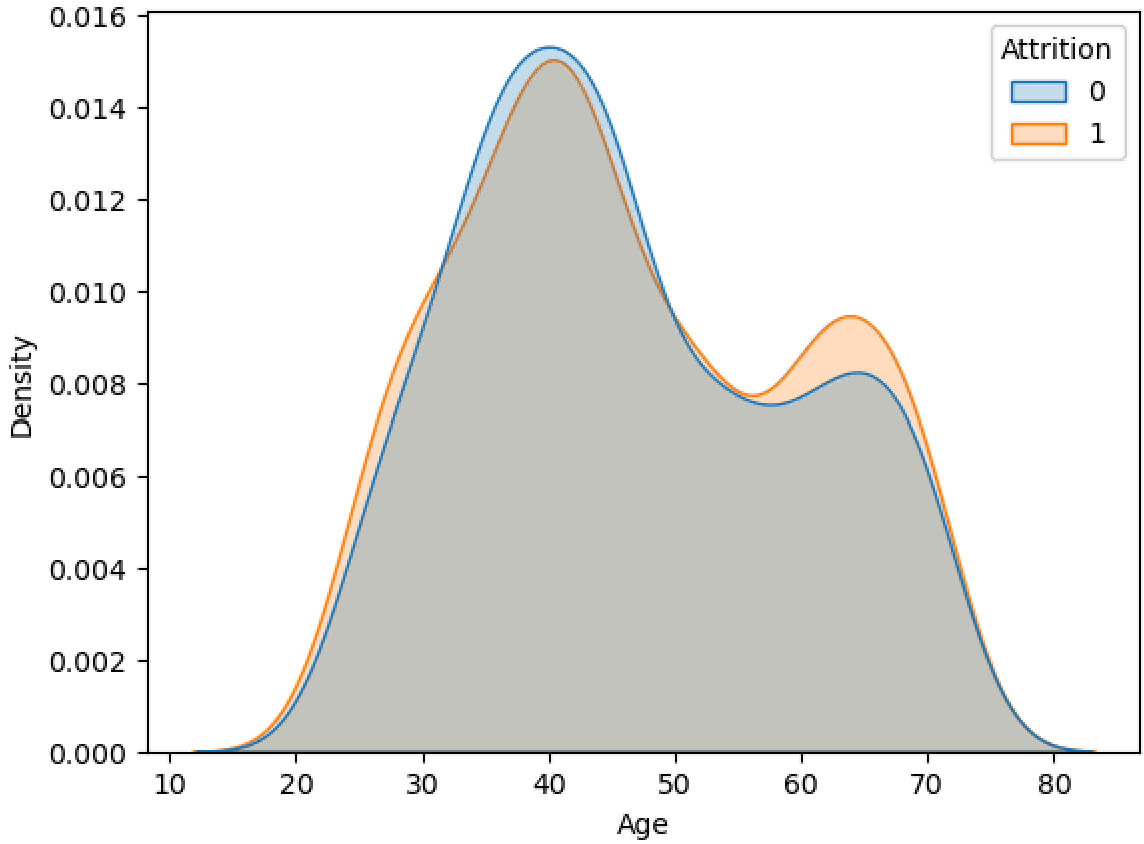
In [35]:
```python
df["Attrition"] = df["ExitDate"].notnull().astype(int)
```

In [37]:
```python
df["TrainingCostPerDay"] = df["Training Cost"] / df["Training Duration(Days)"]
```

In [39]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Attrition by BusinessUnit
sns.countplot(data=df, x="BusinessUnit", hue="Attrition")
plt.xticks(rotation=45)
plt.show()

# Engagement vs Satisfaction
sns.scatterplot(data=df, x="Engagement Score", y="Satisfaction Score", hue="Attrition
plt.show()

# Age distribution by attrition
sns.kdeplot(data=df, x="Age", hue="Attrition", fill=True)
plt.show()
```


Attrition
0
1
count
BusinessUnit
CCDR
EW
PL
TNS
BPC
WBL
NEL
SVG
MSC
PYZ


Attrition
0
1
Satisfaction Score
Engagement Score



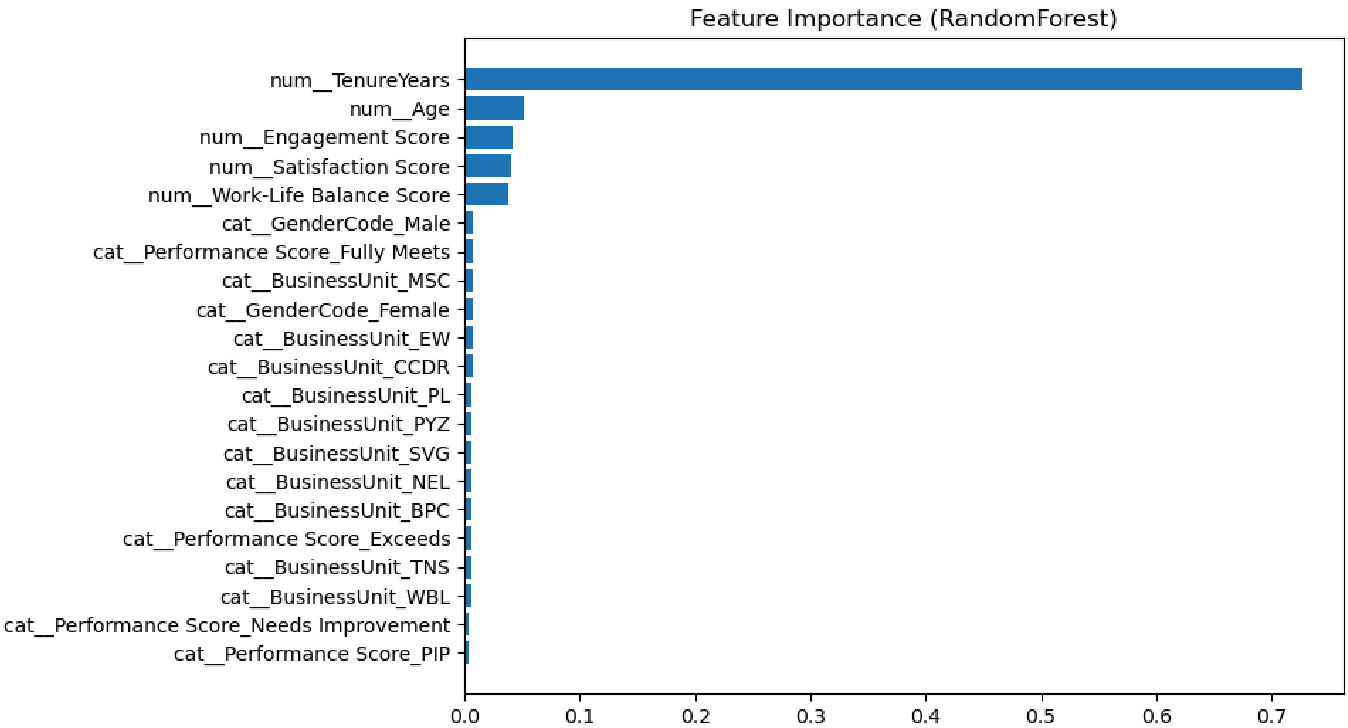
In [41]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# target
y = df["Attrition"]

# select features (example subset)
X = df[["Age", "TenureYears", "Engagement Score", "Satisfaction Score",
        "Work-Life Balance Score", "Performance Score", "GenderCode", "BusinessUnit"]
```

In [43]:
```
# categorical & numerical
cat_features = ["GenderCode", "BusinessUnit", "Performance Score"]
num_features = ["Age", "TenureYears", "Engagement Score", "Satisfaction Score", "Work

preprocessor = ColumnTransformer([
    ("cat", OneHotEncoder(handle_unknown="ignore"), cat_features),
    ("num", StandardScaler(), num_features)
])

# pipeline
model = Pipeline([
    ("preprocess", preprocessor),
    ("clf", RandomForestClassifier(random_state=42))
])

# train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,

# fit
model.fit(X_train, y_train)

# evaluate
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.94      0.86       293
           1       0.93      0.78      0.85       307

    accuracy                           0.85       600
   macro avg       0.86      0.86      0.85       600
weighted avg       0.87      0.85      0.85       600
```

In [51]:

```
import matplotlib.pyplot as plt
import pandas as pd

# Get feature names after preprocessing
feature_names = model.named_steps["preprocess"].get_feature_names_out()

# Get feature importance from RandomForest
importances = model.named_steps["clf"].feature_importances_

# Put into DataFrame
feat_imp = pd.DataFrame({
    "Feature": feature_names,
    "Importance": importances
}).sort_values("Importance", ascending=True)

# Plot
plt.figure(figsize=(8,6))
plt.barh(feat_imp["Feature"], feat_imp["Importance"])
plt.title("Feature Importance (RandomForest)")
plt.show()
```

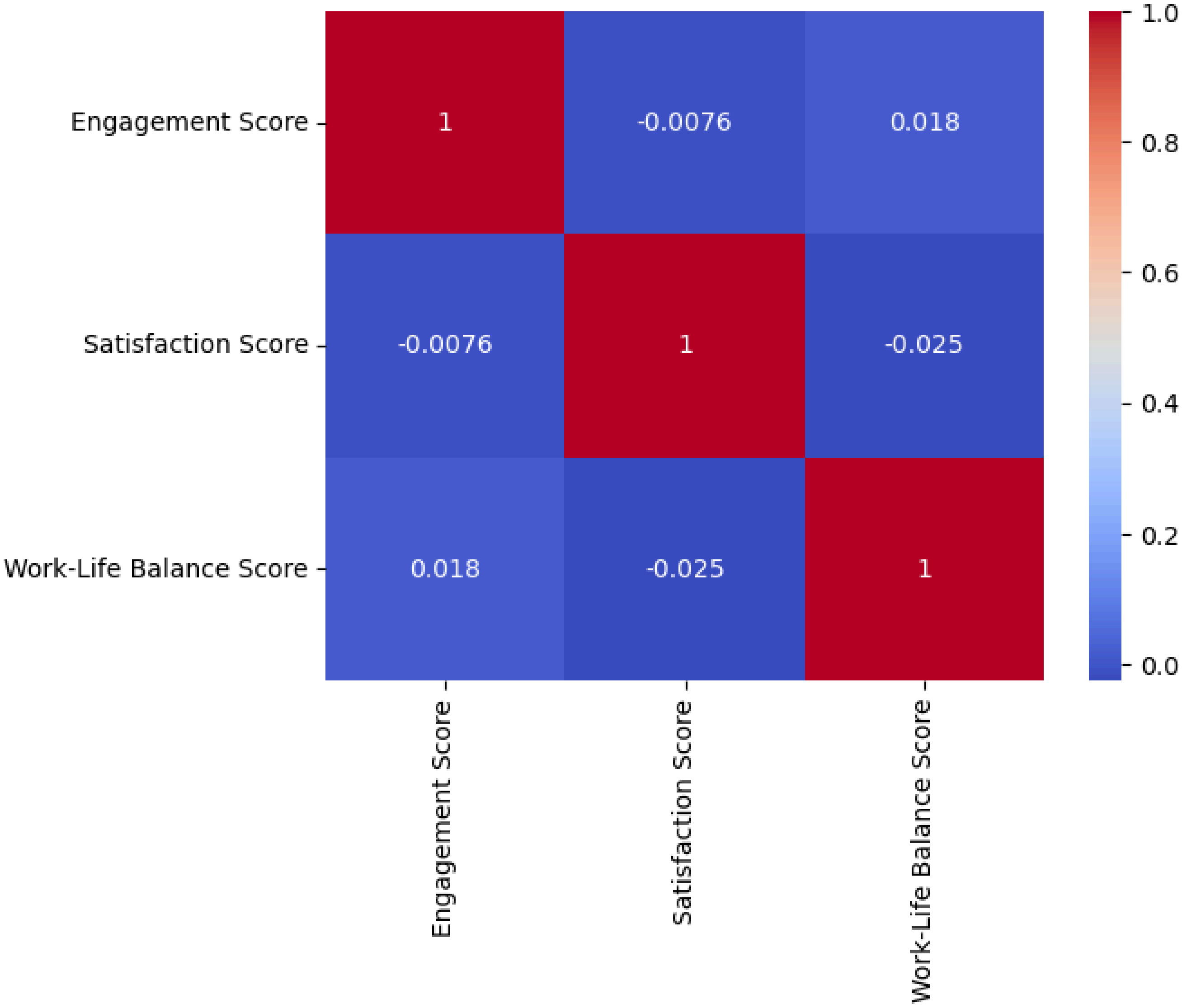


In [53]:

```
# Attrition rate
attrition_rate = df["Attrition"].mean() * 100

# Attrition by department
dept_attrition = df.groupby("DepartmentType")["Attrition"].mean().sort_values() * 100
```

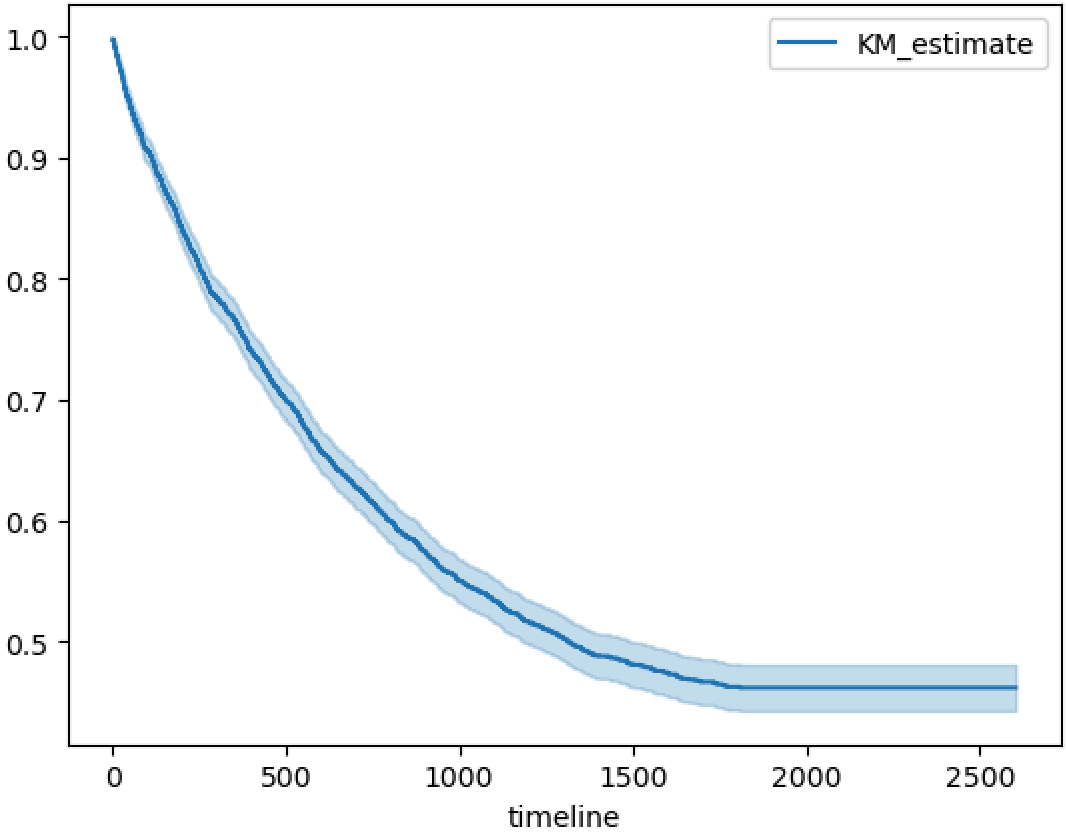
In [55]:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(df[["Engagement Score", "Satisfaction Score", "Work-Life Balance Score"]]
plt.show()
```

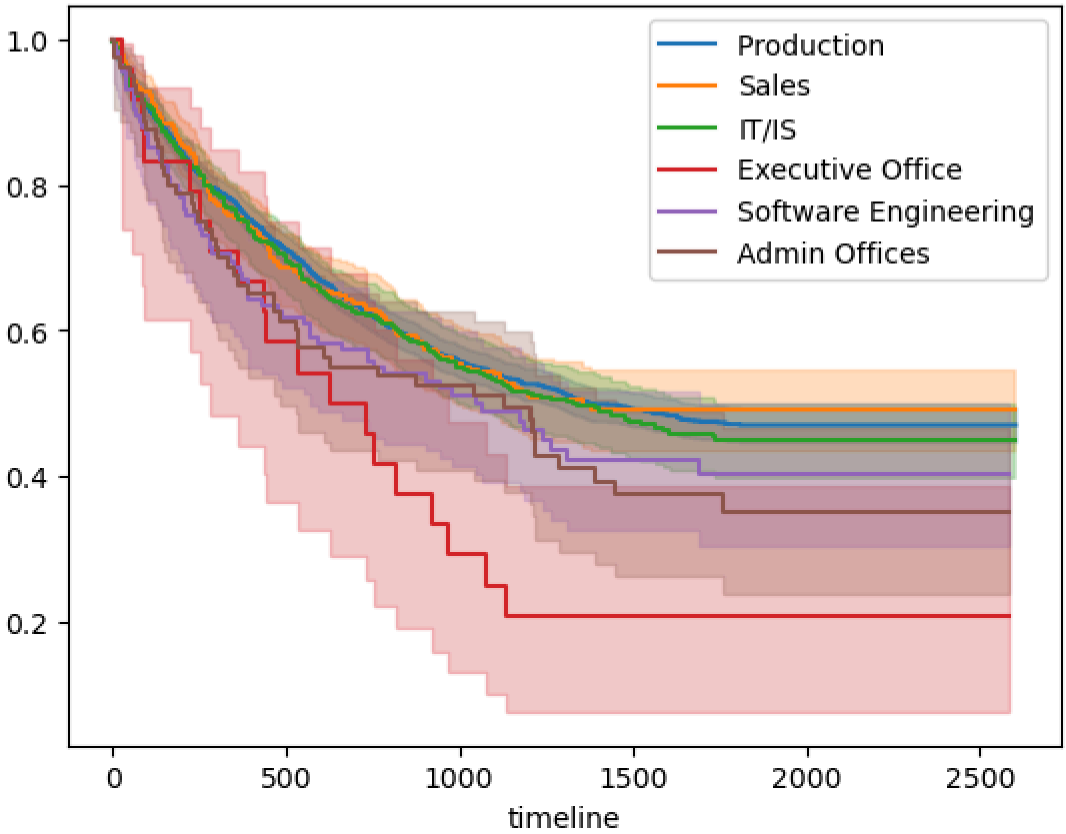


In [59]:
```
from lifelines import KaplanMeierFitter

kmf = KaplanMeierFitter()
kmf.fit(df["TenureDays"], event_observed=df["Attrition"])
kmf.plot_survival_function()
```

Out[59]:
```
<Axes: xlabel='timeline'>
```



In [83]:
```
for dept in df["DepartmentType"].unique():
    kmf.fit(df[df["DepartmentType"]==dept]["TenureDays"],
            event_observed=df[df["DepartmentType"]==dept]["Attrition"],
            label=dept)
    kmf.plot_survival_function()
```

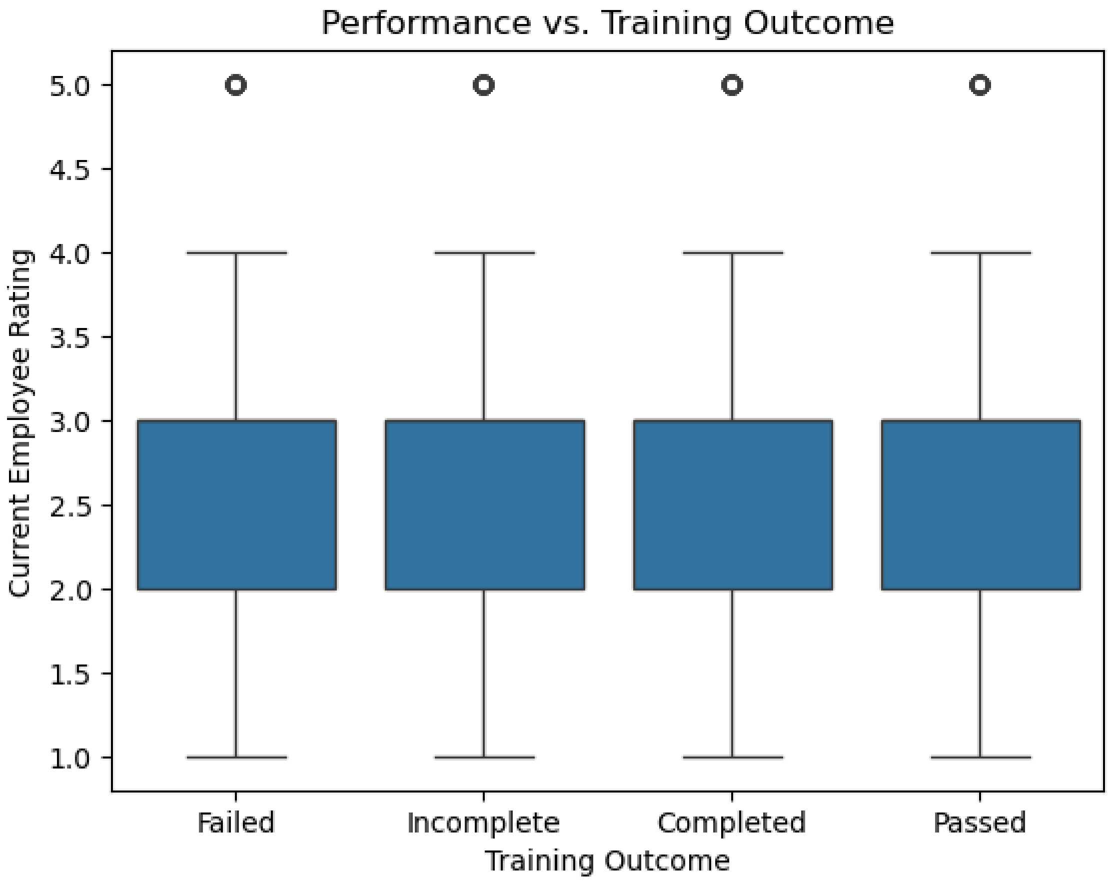


In [61]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

X = df[["Engagement Score", "Satisfaction Score", "Work-Life Balance Score"]]
X_scaled = StandardScaler().fit_transform(X)
```

```
kmeans = KMeans(n_clusters=3, random_state=42)
df["Cluster"] = kmeans.fit_predict(X_scaled)
```

In [63]:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(data=df, x="Training Outcome", y="Current Employee Rating")
plt.title("Performance vs. Training Outcome")
plt.show()

# Average performance by training outcome
df.groupby("Training Outcome")["Current Employee Rating"].mean()
```



Out[63]:

```
Training Outcome
Completed     3.003896
Failed        2.939944
Incomplete    2.989677
Passed        2.939107
Name: Current Employee Rating, dtype: float64
```

In [65]:

```
sns.scatterplot(data=df, x="Training Cost", y="Engagement Score", hue="Training Outco
plt.title("Training Cost vs Engagement Score")
plt.show()

sns.scatterplot(data=df, x="Training Cost", y="Satisfaction Score", hue="Training Out
plt.title("Training Cost vs Satisfaction Score")
plt.show()
```

Training ROI Analysis: This analysis evaluates whether training programs improve employee outcomes, including performance, satisfaction, and attrition. We also explore the relationship between training cost and ROI.

Step 1: Aggregate Training Outcomes: We calculate the average performance, satisfaction, attrition rate, and cost per training program. Then we create a simple **ROI Index** as:

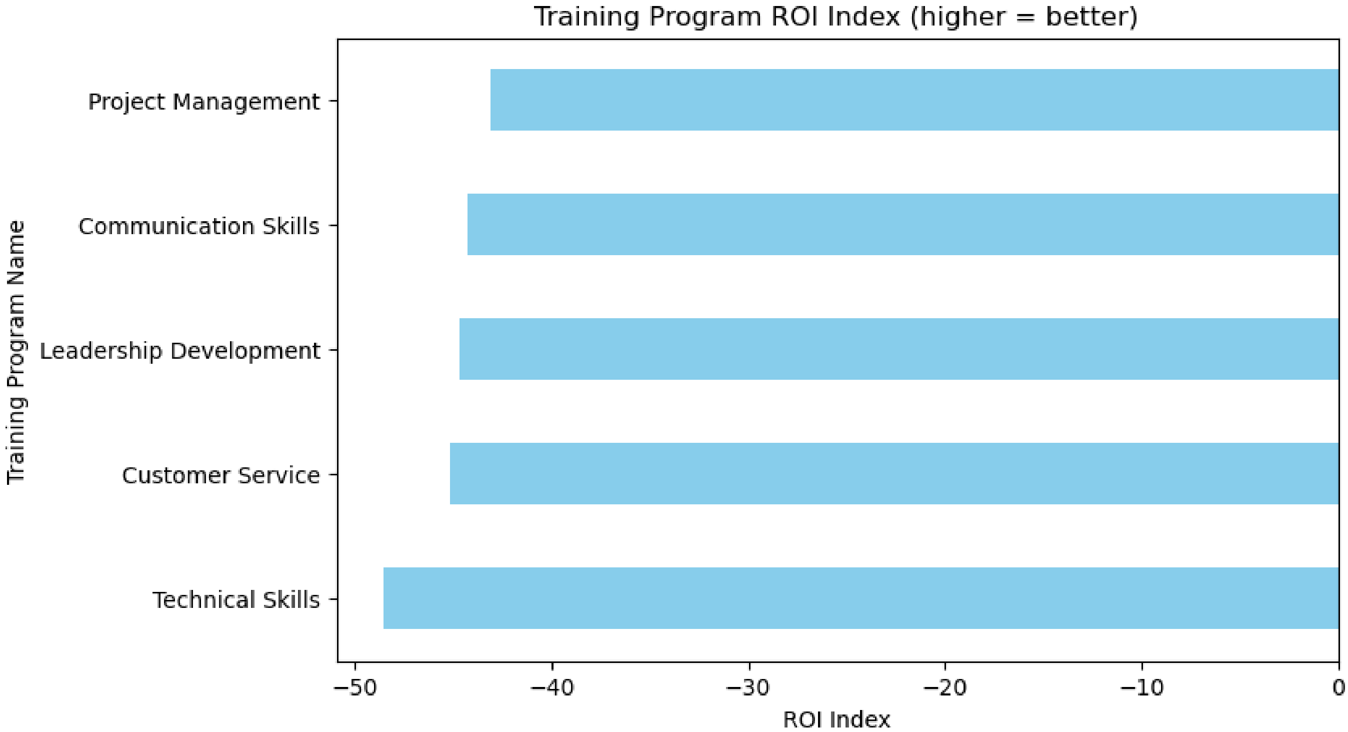
ROI Index = (Performance + Satisfaction) - (Attrition × 100)

In [67]:
```python
# Define ROI as performance gain + satisfaction gain - attrition rate
training_roi = df.groupby("Training Program Name").agg({
    "Current Employee Rating": "mean",
    "Satisfaction Score": "mean",
    "Attrition": "mean",
    "Training Cost": "mean"
})

# Create a simple ROI metric
training_roi["ROI Index"] = (training_roi["Current Employee Rating"] + training_roi["

training_roi.sort_values("ROI Index", ascending=False)
```

Out[67]:

Training Program Name	Current Employee Rating	Satisfaction Score	Attrition	Training Cost	ROI Index
Project Management	2.954023	3.001642	0.490969	563.732627	-43.141215
Communication Skills	2.982169	2.968796	0.502229	542.382229	-44.271917
Leadership Development	2.949477	3.054007	0.506969	564.289251	-44.693380
Customer Service	2.969912	3.046018	0.511504	567.389451	-45.134513
Technical Skills	2.987910	3.050086	0.545769	557.983782	-48.538860

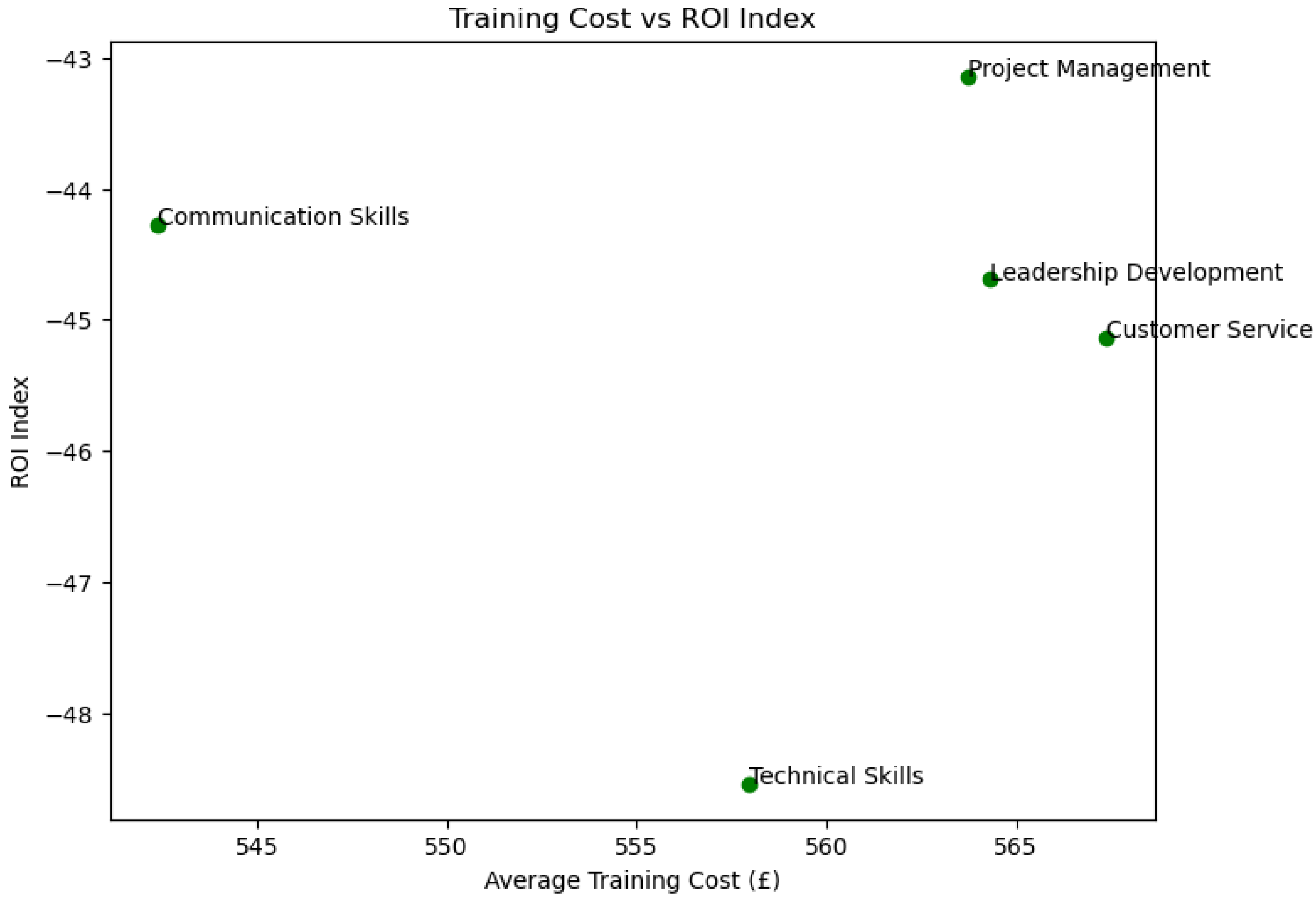
In [69]:
```python
import matplotlib.pyplot as plt

training_roi["ROI Index"].sort_values().plot(kind="barh", figsize=(8,5), color="skybl
plt.title("Training Program ROI Index (higher = better)")
plt.xlabel("ROI Index")
plt.show()
```



In [71]:
```python
plt.figure(figsize=(8,6))
plt.scatter(training_roi["Training Cost"], training_roi["ROI Index"], color="green")
for i, txt in enumerate(training_roi.index):
    plt.annotate(txt, (training_roi["Training Cost"][i], training_roi["ROI Index"][i]
plt.title("Training Cost vs ROI Index")
```

```
plt.xlabel("Average Training Cost (£)")
plt.ylabel("ROI Index")
plt.show()
```

```
/var/folders/xq/2xryw3gd7g7bx8y6s09b93780000gn/T/ipykernel_53975/1964518319.py:4: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  plt.annotate(txt, (training_roi["Training Cost"][i], training_roi["ROI Index"][i]))
```



In [ ]: