IMPORT LIBRARIES AND DATASET

In [6]:

```
import plotly.express as px
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from math import pi
import string
import re
import tensorflow as tf
from surprise import Dataset, Reader
from sklearn.model_selection import train_test_split
from surprise import KNNBasic
from surprise.accuracy import rmse
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from surprise import Dataset, Reader, KNNBasic
from surprise.model_selection import train_test_split
from surprise.accuracy import rmse
import researchpy as rp
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
from nltk.corpus import stopwords
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import LabelEncoder
from scipy.stats import chi2_contingency
```

```
from matplotlib.colors import Normalize
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
import warnings
import plotly.graph_objects as go
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score
import statsmodels.api as sm
import squarify
import math
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.impute import SimpleImputer, MissingIndicator
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
import numpy as np
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
# Read the Excel file into a DataFrame
data= pd.read_excel(r"C:\Users\Administrator\Desktop\DATA.xlsx")
# display the dataframe
data.head()
```

Out[6]:

	user ID	Name	Age	Gender	Academic Background	Field of Study	Skills	Industry_Interest	Job_Type_Interest	Location_Interest	Salary_Expectation_(in_USD)	d
0	986206	John Smith	28	Male	Bachelor's degree	Computer Science 2	Java, Python, Data Structures	Technology	Full-time	New York	65000	
1	769632	Jane Doe	42	Female	Master's degree	Business	Finance, Accounting, Microsoft Excel	Finance	Contract	London	85000	
2	981314	David Lee	35	Male	bachelor"s degree	NaN	Sales, Customer Service, Communication	Retail	Part-time	Chicago	30000	
3	962892	Sarah Johnson	27	Female	Associate's degree	Nursing	Patient Care, Medical Terminology	Healthcare	Full-time	Los Angeles	45000	
4	967782	Michael Williams	46	Male	Bachelor's degree	Marketing	Digital Marketing, Social Media	Marketing	Freelance	Toronto	70000	

In [7]:

```
data.shape
```

Out[7]: (999, 12)

In [8]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999 entries, 0 to 998
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
```

In [8]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999 entries, 0 to 998
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   user ID                    999 non-null    int64 
 1   Name                       996 non-null    object
 2   Age                        999 non-null    int64 
 3   Gender                     927 non-null    object
 4   Academic Background        999 non-null    object
 5   Field of Study             963 non-null    object
 6   Skills                     999 non-null    object
 7   Industry_Interest          999 non-null    object
 8   Job_Type_Interest          999 non-null    object
 9   Location_Interest          995 non-null    object
 10  Salary_Expectation_(in_USD)  999 non-null    int64 
 11  desired_company            999 non-null    object
dtypes: int64(3), object(9)
memory usage: 93.8+ KB
```

DESCRIPTIVE STATISTICS

In [9]:
```python
# Get descriptive statistics
descriptive_stats = data.describe()

# Display the descriptive statistics
print(descriptive_stats)
```

```
             user ID         Age  Salary_Expectation_(in_USD)
count     999.000000  999.000000                   999.000000
mean   931435.157157   30.625626                 65385.385385
std    158544.732526    9.187644                 28965.576740
min       382.000000    0.000000                 20000.000000
25%    962133.000000   24.000000                 45000.000000
50%    972695.000000   30.000000                 65000.000000
```

CHECKING FOR MISSING VALUES

In [10]:
```
# Check for missing values in the dataset
missing_values = data.isnull().sum()

# Display the count of missing values for each column
print(missing_values)
```

```
user ID                        0
Name                           3
Age                            0
Gender                        72
Academic Background            0
Field of Study                36
Skills                         0
Industry_Interest              0
Job_Type_Interest              0
Location_Interest              4
Salary_Expectation_(in_USD)    0
desired_company                0
dtype: int64
```

In [11]:
```
# Impute missing values for numerical columns with the mean
numerical_columns = data.select_dtypes(include='number').columns
data[numerical_columns] = data[numerical_columns].fillna(data[numerical_columns].mean())

# Impute missing values for categorical columns with the most frequent value
categorical_columns = data.select_dtypes(include='object').columns
data[categorical_columns] = data[categorical_columns].fillna(data[categorical_columns].mode().iloc[0])
```

In [12]:
```
# Check for missing values in the dataset
missing_values = data.isnull().sum()

# Display the count of missing values for each column
print(missing_values)
```

In [12]:

```python
# Check for missing values in the dataset
missing_values = data.isnull().sum()

# Display the count of missing values for each column
print(missing_values)
```

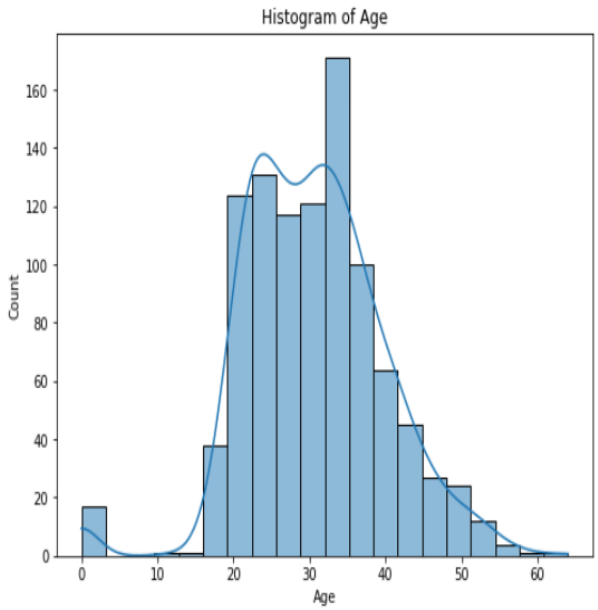
```
user ID                       0
Name                          0
Age                           0
Gender                        0
Academic Background           0
Field of Study                0
Skills                        0
Industry_Interest             0
Job_Type_Interest             0
Location_Interest             0
Salary_Expectation_(in_USD)   0
desired_company               0
dtype: int64
```

In [13]:

```python
# Plot histogram of Age
plt.figure(figsize=(8, 6))
sns.histplot(data['Age'], bins=20, kde=True)
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Histogram of Age')
plt.show()
```

In [13]:

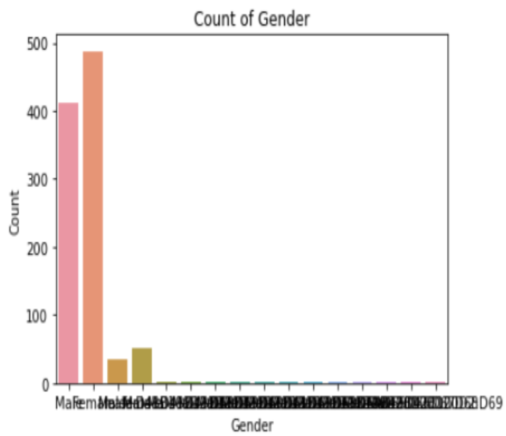
```python
# Plot histogram of Age
plt.figure(figsize=(8, 6))
sns.histplot(data['Age'], bins=20, kde=True)
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Histogram of Age')
plt.show()
```



In [14]:

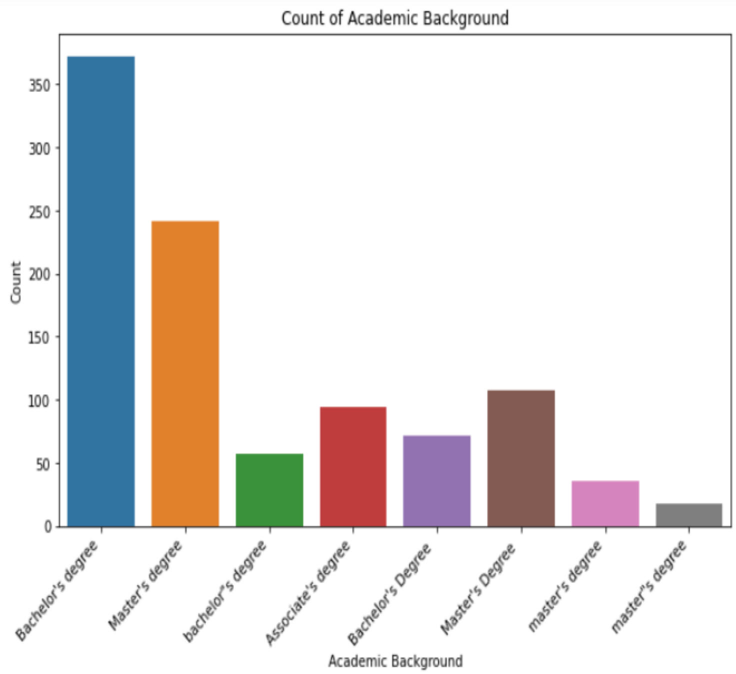
```
# Plot count of Gender
plt.figure(figsize=(6, 4))
sns.countplot(data['Gender'])
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Count of Gender')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```



In [15]:

```
# Plot count of Academic Background
plt.figure(figsize=(10, 6))
sns.countplot(data['Academic Background'])
plt.xlabel('Academic Background')
plt.ylabel('Count')
plt.title('Count of Academic Background')
plt.xticks(rotation=45, ha='right')
plt.show()
```



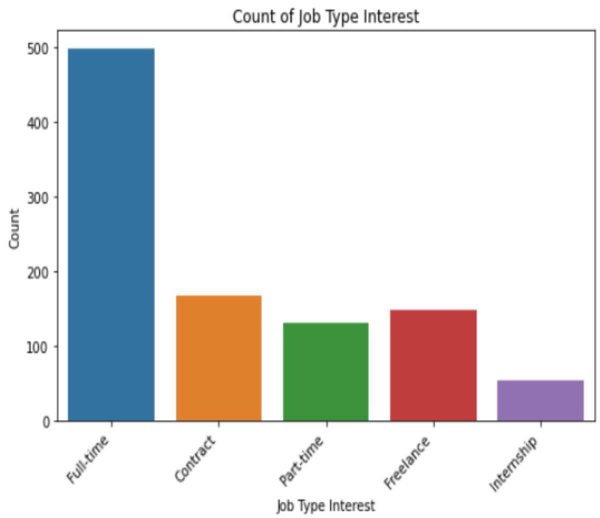
In [17]:

```
# Plot count of Job Type Interest
plt.figure(figsize=(8, 5))
sns.countplot(data['Job_Type_Interest'])
plt.xlabel('Job Type Interest')
plt.ylabel('Count')
plt.title('Count of Job Type Interest')
plt.xticks(rotation=45, ha='right')
plt.show()
```

In [17]:

```python
# Plot count of Job Type Interest
plt.figure(figsize=(8, 5))
sns.countplot(data['Job_Type_Interest'])
plt.xlabel('Job Type Interest')
plt.ylabel('Count')
plt.title('Count of Job Type Interest')
plt.xticks(rotation=45, ha='right')
plt.show()
```

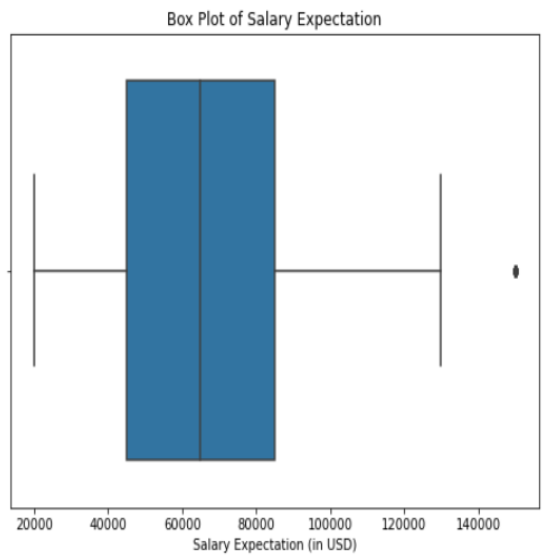
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```



In [19]:

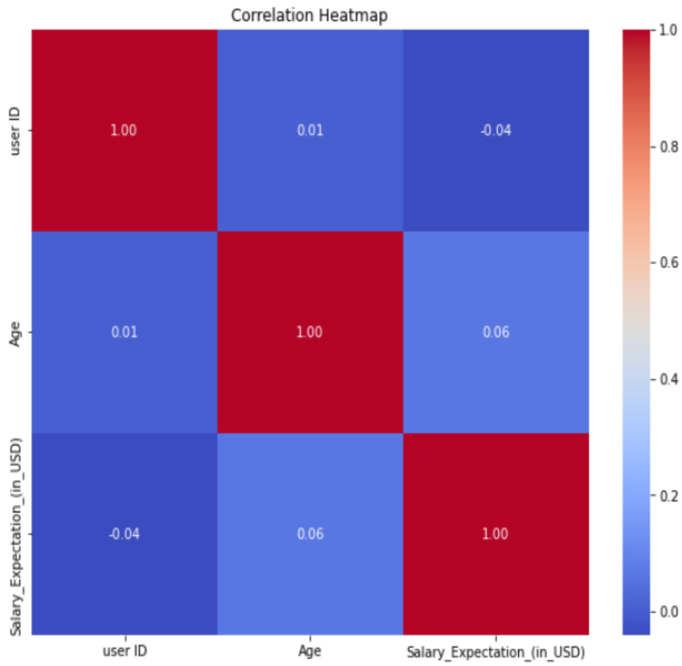
```
# Plot box plot of Salary Expectation
plt.figure(figsize=(8, 6))
sns.boxplot(data['Salary_Expectation_(in_USD)'])
plt.xlabel('Salary Expectation (in USD)')
plt.title('Box Plot of Salary Expectation')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```



In [20]:

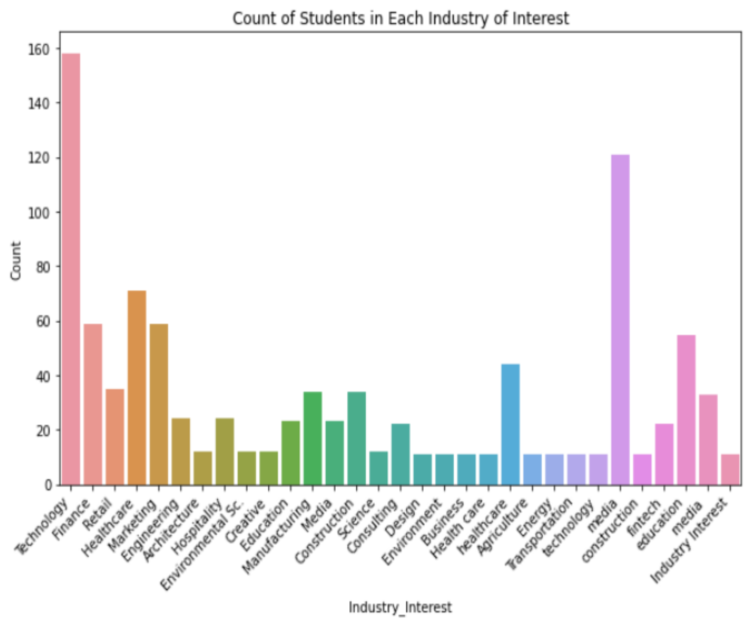
```
# Plot correlation heatmap for numerical columns
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



In [22]:

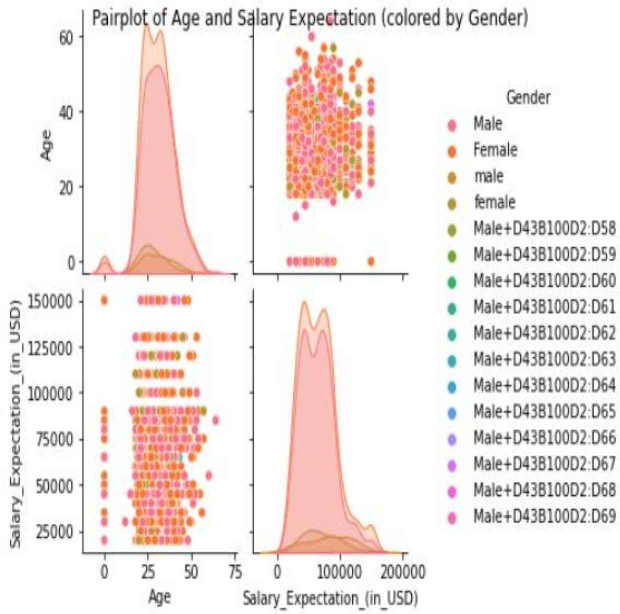
```
plt.figure(figsize=(10, 6))
sns.countplot(data['Industry_Interest'])
plt.xlabel('Industry_Interest')
plt.ylabel('Count')
plt.title('Count of Students in Each Industry of Interest')
plt.xticks(rotation=45, ha='right')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword a
rg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```



In [24]:

```
sns.pairplot(data, vars=['Age', 'Salary_Expectation_(in_USD)'], hue='Gender')
plt.suptitle('Pairplot of Age and Salary Expectation (colored by Gender)')
plt.show()
```

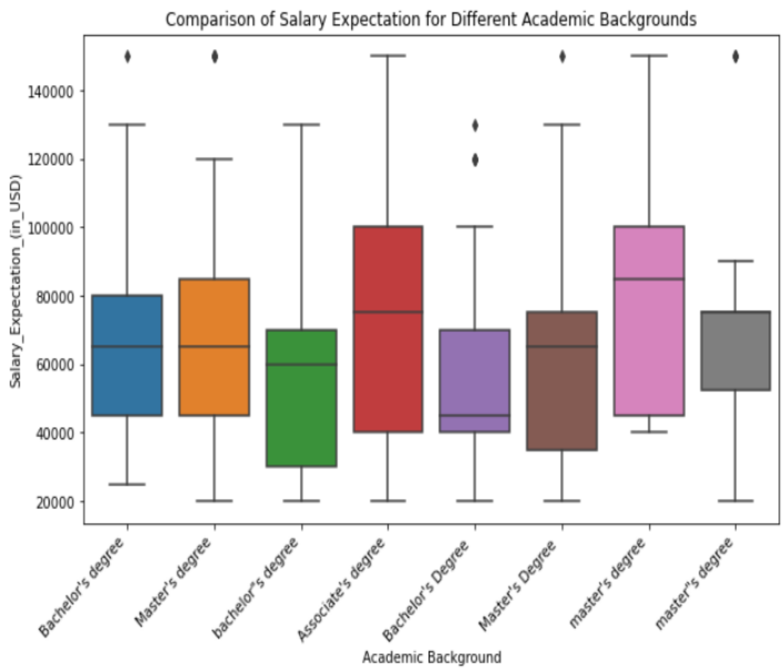


In [27]:

```
plt.figure(figsize=(10, 6))
sns.violinplot(x='Job_Type_Interest', y='Salary_Expectation_(in_USD)', data=data)
plt.xlabel('Job Type Interest')
plt.ylabel('Salary_Expectation_(in_USD)')
plt.title('Distribution of Salary Expectation across Job Types')
plt.xticks(rotation=45, ha='right')
plt.show()
```

In [30]:

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Academic Background', y='Salary_Expectation_(in_USD)', data=data)
plt.xlabel('Academic Background')
plt.ylabel('Salary_Expectation_(in_USD)')
plt.title('Comparison of Salary Expectation for Different Academic Backgrounds')
plt.xticks(rotation=45, ha='right')
plt.show()
```



In [32]:

```
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Age', y='Salary_Expectation_(in_USD)', data=data, hue='Gender')
plt.xlabel('Age')
plt.ylabel('Salary_Expectation_(in_USD)')
plt.title('Scatter Plot of Age vs. Salary Expectation (colored by Gender)')
plt.show()
```

In [34]:

```python
#One-Hot Encoding
one_hot_encoded_data = pd.get_dummies(data, columns=['Academic Background', 'Field of Study', 'Industry_Interest', 'Job_Type


label_encoder = LabelEncoder()
data['Gender_encoded'] = label_encoder.fit_transform(data['Gender'])

# Display the encoded datasets
print("One-Hot Encoded Data:")
print(one_hot_encoded_data.head())

print("\nLabel Encoded Data:")
print(data[['Gender', 'Gender_encoded']].head())
```

```
One-Hot Encoded Data:
   user ID              Name  Age  Gender  \
0   986206        John Smith   28    Male
1   769632          Jane Doe   42  Female
2   981314         David Lee   35    Male
3   962892     Sarah Johnson   27  Female
4   967782  Michael Williams   46    Male

                                Skills  Salary_Expectation_(in_USD)  \
0          Java, Python, Data Structures                        65000
1     Finance, Accounting, Microsoft Excel                        85000
2  Sales, Customer Service, Communication                        30000
3       Patient Care, Medical Terminology                        45000
4         Digital Marketing, Social Media                        70000

   Academic Background_Associate's degree  \
0                                       0
1                                       0
2                                       0
3                                       1
4                                       0

   Academic Background_Bachelor's Degree  \
```

In [36]:

```python
#Extracting First Name from Name
data['First Name'] = data['Name'].str.split().str[0]

# Feature 2: Converting 'Age' to Age Group
def get_age_group(age):
    if age < 25:
        return 'Young'
    elif age >= 25 and age < 40:
        return 'Middle-aged'
    else:
        return 'Senior'

data['Age Group'] = data['Age'].apply(get_age_group)

# Feature 3: Counting the number of skills each student has
data['Number of Skills'] = data['Skills'].apply(lambda x: len(x.split(',')))

# Feature 4: Encoding 'Full-time' as 1 and 'Part-time' as 0 for Job Type Interest
data['Job_Type_Interest'] = data['Job_Type_Interest'].apply(lambda x: 1 if x == 'Full-time' else 0)

# Feature 5: Encoding 'Male' as 1 and 'Female' as 0 for Gender
data['Gender'] = data['Gender'].apply(lambda x: 1 if x == 'Male' else 0)

# Display the updated dataset with engineered features
print(data.head())
```

```
       Field of Study                              Skills  \
0  Computer Science 2          Java, Python, Data Structures
1            Business      Finance, Accounting, Microsoft Excel
2    Computer Science  Sales, Customer Service, Communication
3             Nursing        Patient Care, Medical Terminology
4           Marketing         Digital Marketing, Social Media

  Industry_Interest  Job_Type_Interest Location_Interest  \
0        Technology                  1          New York
1           Finance                  0            London
2            Retail                  0           Chicago
3        Healthcare                  1       Los Angeles
4         Marketing                  0           Toronto
```

In [37]:

```python
# Prepare the feature matrix X and target vector y
X = data[['Age', 'Gender']]  # Select relevant features
y = data['Job_Type_Interest']  # Target variable

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print("KNN Classifier Accuracy:", accuracy)
```

```
KNN Classifier Accuracy: 0.525
```

In [45]:

```python
# Import required libraries
from surprise import Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import KNNBasic
from surprise.accuracy import rmse

# Prepare the data for Surprise library
reader = Reader(rating_scale=(0, 1))
surprise_data = Dataset.load_from_df(data[['user ID', 'Salary_Expectation_(in_USD)', 'Age']], reader)

# Split the data into training and testing sets (80% train, 20% test)
trainset, testset = train_test_split(surprise_data, test_size=0.2, random_state=42)

# Initialize and train the KNNBasic collaborative filtering model
knn_collaborative = KNNBasic(sim_options={'user_based': True})
knn_collaborative.fit(trainset)

# Make predictions on the test set
predictions = knn_collaborative.test(testset)
```

In [45]:

```python
# Import required libraries
from surprise import Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import KNNBasic
from surprise.accuracy import rmse

# Prepare the data for Surprise library
reader = Reader(rating_scale=(0, 1))
surprise_data = Dataset.load_from_df(data[['user ID', 'Salary_Expectation_(in_USD)', 'Age']], reader)

# Split the data into training and testing sets (80% train, 20% test)
trainset, testset = train_test_split(surprise_data, test_size=0.2, random_state=42)

# Initialize and train the KNNBasic collaborative filtering model
knn_collaborative = KNNBasic(sim_options={'user_based': True})
knn_collaborative.fit(trainset)

# Make predictions on the test set
predictions = knn_collaborative.test(testset)

# Evaluate the model's performance (Root Mean Squared Error, RMSE)
rmse_score = rmse(predictions)
print("Collaborative Filtering RMSE:", rmse_score)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 31.6624
Collaborative Filtering RMSE: 31.662438314191785
```

In [49]:

```python
# Import required libraries
from surprise import Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import KNNBasic
from surprise.accuracy import rmse

# Prepare the data for Surprise library
reader = Reader(rating_scale=(0, 1))
surprise_data = Dataset.load_from_df(data[['user ID', 'Salary_Expectation_(in_USD)', 'Age']], reader)

# Split the data into training and testing sets (80% train, 20% test)
```

```python
# Split the data into training and testing sets (80% train, 20% test)
trainset, testset = train_test_split(surprise_data, test_size=0.2, random_state=42)

# Initialize and train the KNNBasic collaborative filtering model
knn_collaborative = KNNBasic(sim_options={'user_based': True})
knn_collaborative.fit(trainset)

# Make predictions on the test set
predictions = knn_collaborative.test(testset)

# Evaluate the model's performance (Root Mean Squared Error, RMSE)
rmse_score = rmse(predictions)
print("Collaborative Filtering RMSE:", rmse_score)
```

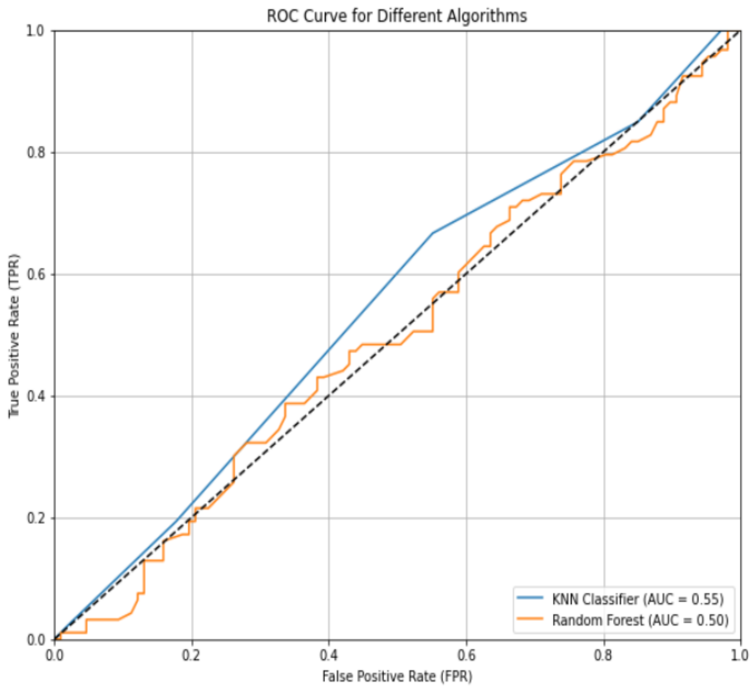
```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 31.6624
Collaborative Filtering RMSE: 31.662438314191785
```

In [51]:

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc

# Prepare the feature matrix X and target vector y
X = data[['Age', 'Gender', 'Number of Skills']]  # Select relevant features
y = data['Job_Type_Interest']  # Target variable

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the feature data for better training performance (not necessary for some algorithms)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize models for different algorithms
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train_scaled, y_train)
```

```python
# Initialize models for different algorithms
knn_classifier = KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X_train_scaled, y_train)

# Random Forest
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train_scaled, y_train)

# Calculate ROC curves and AUC for each model
models = [knn_classifier, rf_classifier]
model_names = ['KNN Classifier', 'Random Forest']

plt.figure(figsize=(10, 8))

for model, name in zip(models, model_names):
    y_pred_prob = model.predict_proba(X_test_scaled)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
    auc_score = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{name} (AUC = {auc_score:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Different Algorithms')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```

```
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve for Different Algorithms')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



In [52]:

```
accuracy = accuracy_score(y_test, y_pred_prob.round())  # Calculate accuracy using y_pred_prob
report = classification_report(y_test, knn_classifier.predict(X_test_scaled))  # Using the KNN classifier

print(f"--- {name} ---")
print("Accuracy:", accuracy)
print("Classification Report:")
print(report)
print("------------------------------------")
```

```
--- Random Forest ---
Accuracy: 0.485
Classification Report:
              precision    recall  f1-score   support

           0       0.61      0.45      0.52       107
           1       0.51      0.67      0.58        93

    accuracy                           0.55       200
   macro avg       0.56      0.56      0.55       200
weighted avg       0.56      0.55      0.55       200

------------------------------------
```

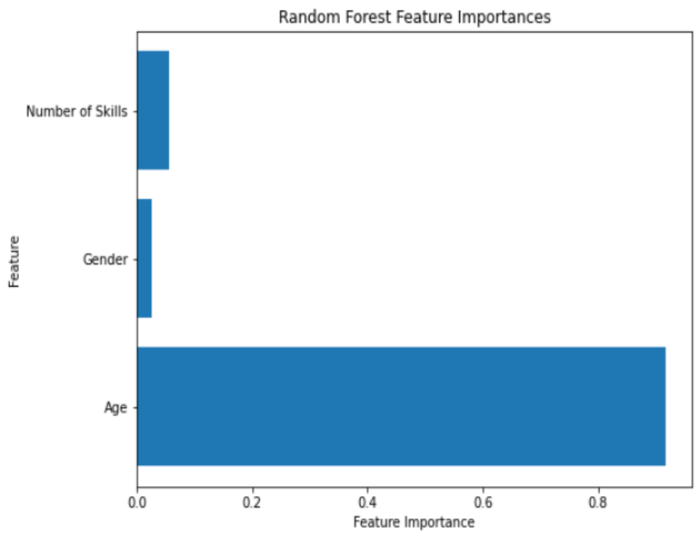
In [53]:

```
# Get feature importances from the trained Random Forest classifier
feature_importances = rf_classifier.feature_importances_

# Print feature importances
print("Feature Importances:")
for feature, importance in zip(X.columns, feature_importances):
    print(f"{feature}: {importance}")
```

```
Feature Importances:
Age: 0.9170870182907975
Gender: 0.025868912623061253
Number of Skills: 0.05704406908614126
```

In [54]:

```
# Get feature importances from the trained Random Forest classifier
feature_importances = rf_classifier.feature_importances_

# Plot feature importances in a bar plot
feature_names = X.columns
plt.figure(figsize=(8, 6))
plt.barh(feature_names, feature_importances)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Random Forest Feature Importances')
plt.show()
```



In [57]:

```
import joblib
import os
from sklearn.preprocessing import OneHotEncoder, StandardScaler

# Create the directory
output_directory = 'student_job_recommendation_system/'
os.makedirs(output_directory, exist_ok=True)

# Assuming you have defined the 'academic_data' variable before this point
# Convert categorical variables to numerical using one-hot encoding
encoder = OneHotEncoder()
academic_encoded = encoder.fit_transform(data).toarray()

# Scale the data for better clustering performance
scaler = StandardScaler()
academic_scaled = scaler.fit_transform(academic_encoded)

# Save the trained machine learning models
# Assuming you already have the trained models: knn_classifier, knn_collaborative, bayesian_model, model, rf_classifier
joblib.dump(knn_classifier, 'student_job_recommendation_system/knn_classifier_model.pkl')
joblib.dump(rf_classifier, 'student_job_recommendation_system/random_forest_model.pkl')
```

Out[57]:

```
['student_job_recommendation_system/random_forest_model.pkl']
```