In [151]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, LabelEncoder, OneHo
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

Travel Insurance - DataSet1

In [152]:
```
data1 = pd.read_csv("TravelInsurancePrediction.csv")
data1 = data1.drop("Unnamed: 0",axis=1)
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1987 entries, 0 to 1986
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Age                  1987 non-null   int64 
 1   Employment Type      1987 non-null   object
 2   GraduateOrNot        1987 non-null   object
 3   AnnualIncome         1987 non-null   int64 
 4   FamilyMembers        1987 non-null   int64 
 5   ChronicDiseases      1987 non-null   int64 
 6   FrequentFlyer        1987 non-null   object
 7   EverTravelledAbroad  1987 non-null   object
 8   TravelInsurance      1987 non-null   int64 
dtypes: int64(5), object(4)
memory usage: 139.8+ KB
```

In [153]:
```
data1.head()
```

Out[153]:

	Age	Employment Type	GraduateOrNot	AnnualIncome	FamilyMembers	ChronicDiseases	Frequ
0	31	Government Sector	Yes	400000	6	1	
1	31	Private Sector/Self Employed	Yes	1250000	7	0	
2	34	Private Sector/Self Employed	Yes	500000	4	1	
3	28	Private Sector/Self Employed	Yes	700000	3	1	
4	28	Private Sector/Self Employed	Yes	700000	8	1	

In [154]:
```
data1.describe()
```

Out[154]:

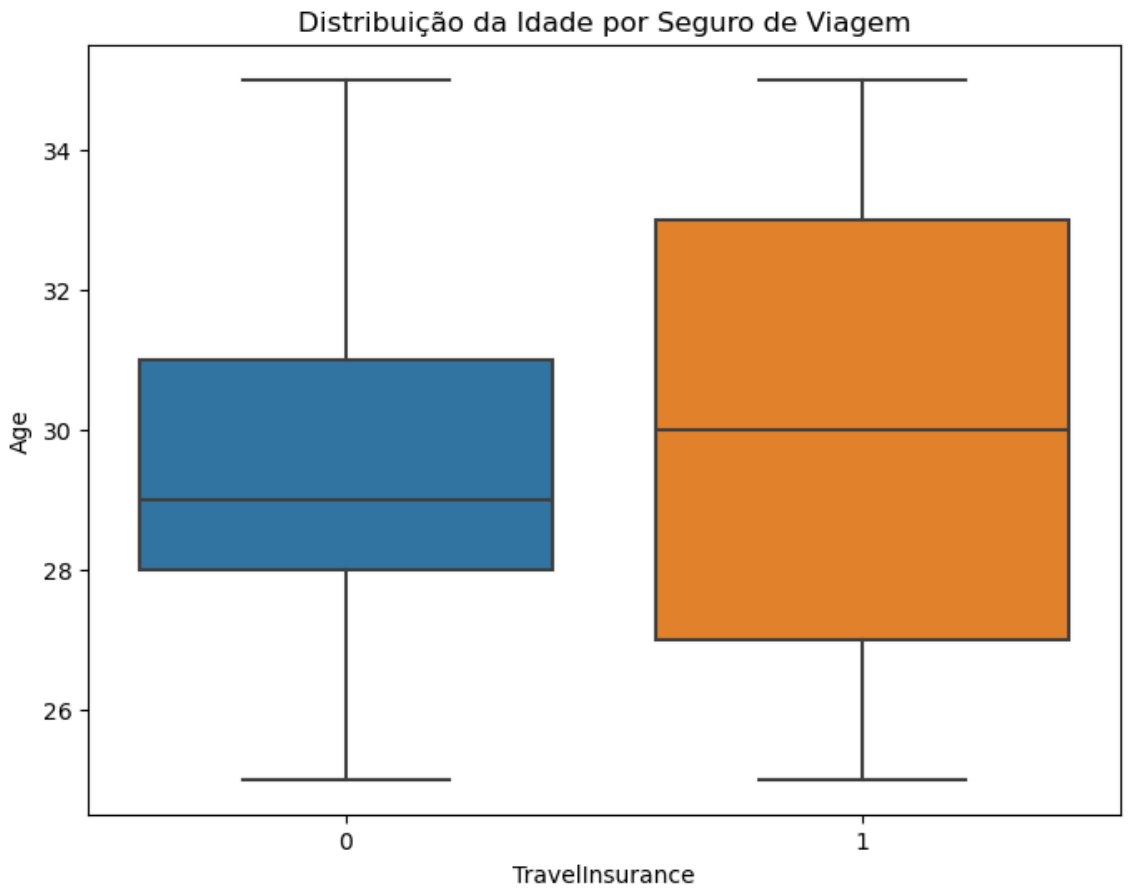
	Age	AnnualIncome	FamilyMembers	ChronicDiseases	TravelInsurance
count	1987.000000	1.987000e+03	1987.000000	1987.000000	1987.000000
mean	29.650226	9.327630e+05	4.752894	0.277806	0.357323
std	2.913308	3.768557e+05	1.609650	0.448030	0.479332
min	25.000000	3.000000e+05	2.000000	0.000000	0.000000
25%	28.000000	6.000000e+05	4.000000	0.000000	0.000000
50%	29.000000	9.000000e+05	5.000000	0.000000	0.000000
75%	32.000000	1.250000e+06	6.000000	1.000000	1.000000
max	35.000000	1.800000e+06	9.000000	1.000000	1.000000

In [155]:
```
data1.isnull().sum()
```

Out[155]:
```
Age                  0
Employment Type      0
GraduateOrNot        0
AnnualIncome         0
FamilyMembers        0
ChronicDiseases      0
FrequentFlyer        0
EverTravelledAbroad  0
TravelInsurance      0
dtype: int64
```

In [156]:
```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [157]:
```
# Boxplot para a coluna 'Age'
plt.figure(figsize=(8, 6))
sns.boxplot(x='TravelInsurance', y='Age', data=data1)
plt.title('Distribuição da Idade por Seguro de Viagem')
plt.show()
```



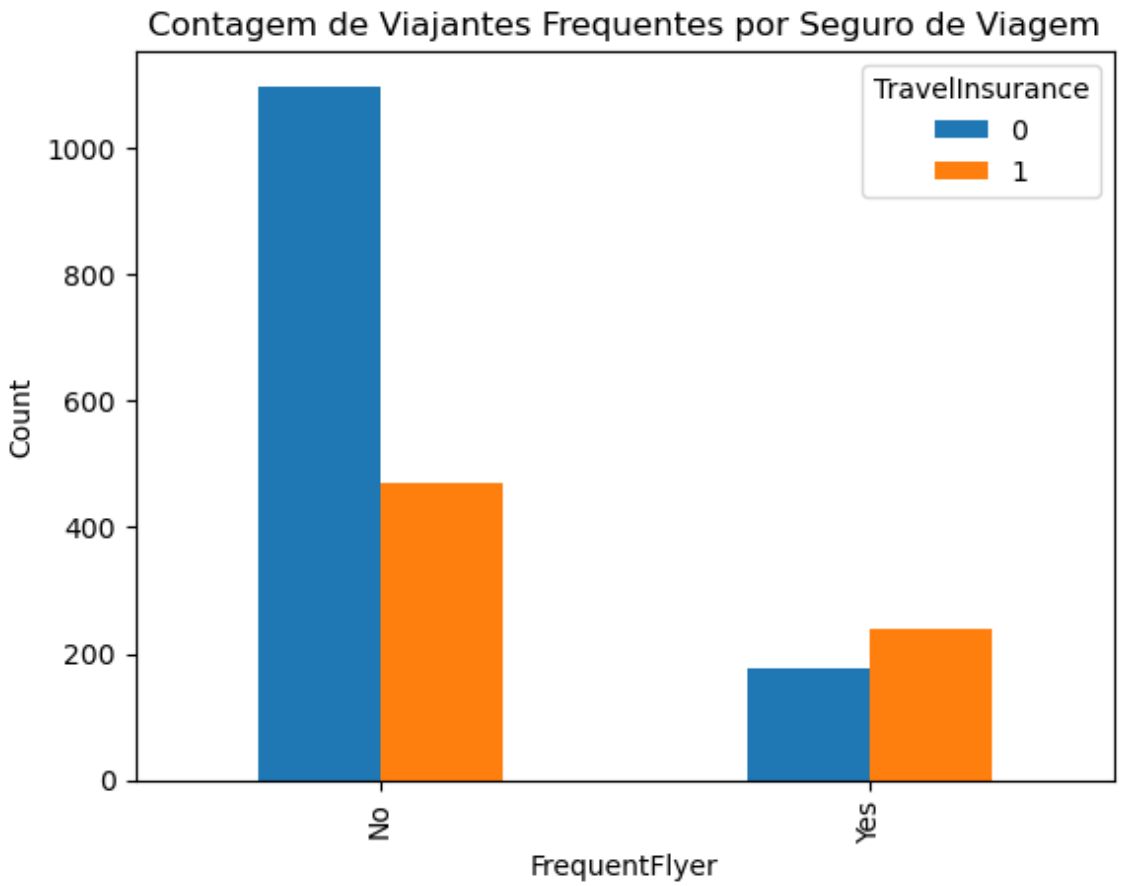
Interpretation of the boxplot:

boxplot graph is a standardized way of displaying the distribution of data based on a five-number summary : minimum, first quartile (Q1), median, third quartile (Q3), and maximum. As we can see in the `.describe()` method. This visualization helps to compare the age distributions between individuals with and without travel insurance.

- The median age of individuals with travel insurance (1) is higher than those without (0).
- The age distribution for those with travel insurance is wider than for those without, which suggests more variability in the ages of insured individuals.
- The ages for both categories are skewed towards a younger population, with medians below the midpoint of the age range (assuming an usual adult age range of 18-60 for travel insurance relevance).

In [158]:
```
# Countplot para a coluna 'FrequentFlyer'
plt.figure(figsize=(8, 6))
# Count the occurrences
counts = data1.groupby(['FrequentFlyer', 'TravelInsurance']).size().u
# Plot
counts.plot(kind='bar', stacked=False)
plt.title('Contagem de Viajantes Frequentes por Seguro de Viagem')
plt.xlabel('FrequentFlyer')
plt.ylabel('Count')
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



Interpretation

- There are significantly more non-frequent flyers than frequent flyers
- Among the non-frequent flyers, a greater number do not have travel insurance compared to those who do.
- For frequent flyers, the distribution between insured and uninsured individuals is more even, but it still appears that slightly fewer have travel insurance than those who do not.
- Relative to their group sizes, frequent flyers seem more likely to have travel insurance than non-frequent flyers.

This visualization helps to understand how travel insurance ownership is distributed among frequent and non-frequent flyers. It suggests that being a frequent flyer might be associated with a higher likelihood of having travel insurance.

Data Preparation:

In [159]:
```
X = data1.drop('TravelInsurance', axis=1)
y = data1['TravelInsurance']
```

In [160]:
```
# converting categorical variables into numerical values
X_encoded = pd.get_dummies(X, columns=['Employment Type', 'GraduateOr
```

In [161]:
```
# StandardScaler to normalize the data
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X_encoded)
```

In [162]:
```
# The dataset is split into training and testing
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y,
```

Models

In [163]:
```
# Decision Tree
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)

# KNN
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train, y_train)
knn_pred = knn_model.predict(X_test)
```

Model Evaluation

- Both models are evaluated on the test data using accuracy and a confusion matrix.

In [164]:
```
# Acurácia e Matriz de Confusão para Decision Tree
dt_accuracy = accuracy_score(y_test, dt_pred)
dt_conf_matrix = confusion_matrix(y_test, dt_pred)
print("Acurácia Decision Tree:", dt_accuracy)
print("Matriz de Confusão Decision Tree:")
print(dt_conf_matrix)

# Acurácia e Matriz de Confusão para KNN
knn_accuracy = accuracy_score(y_test, knn_pred)
knn_conf_matrix = confusion_matrix(y_test, knn_pred)
print("\nAcurácia KNN:", knn_accuracy)
print("Matriz de Confusão KNN:")
print(knn_conf_matrix)
```

```
Acurácia Decision Tree: 0.8090452261306532
Matriz de Confusão Decision Tree:
[[230  27]
 [ 49  92]]

Acurácia KNN: 0.7763819095477387
Matriz de Confusão KNN:
[[227  30]
 [ 59  82]]
```

- Decision Tree Classifier:
  - Accuracy: 80.9%
  - Confusion Matrix: 230 true negatives (correctly predicted no insurance), 27 false positives (incorrectly predicted insurance), 49 false negatives (incorrectly predicted no insurance), and 92 true positives (correctly predicted insurance).
- K-Nearest Neighbors (KNN) Classifier:
  - Accuracy: 77.6%
  - Confusion Matrix: 227 true negatives, 30 false positives, 59 false negatives, and 82 true positives.

Results:

- The Decision Tree model performs better than the KNN model in terms of accuracy.
- The Decision Tree has fewer false negatives but more false positives compared to the KNN

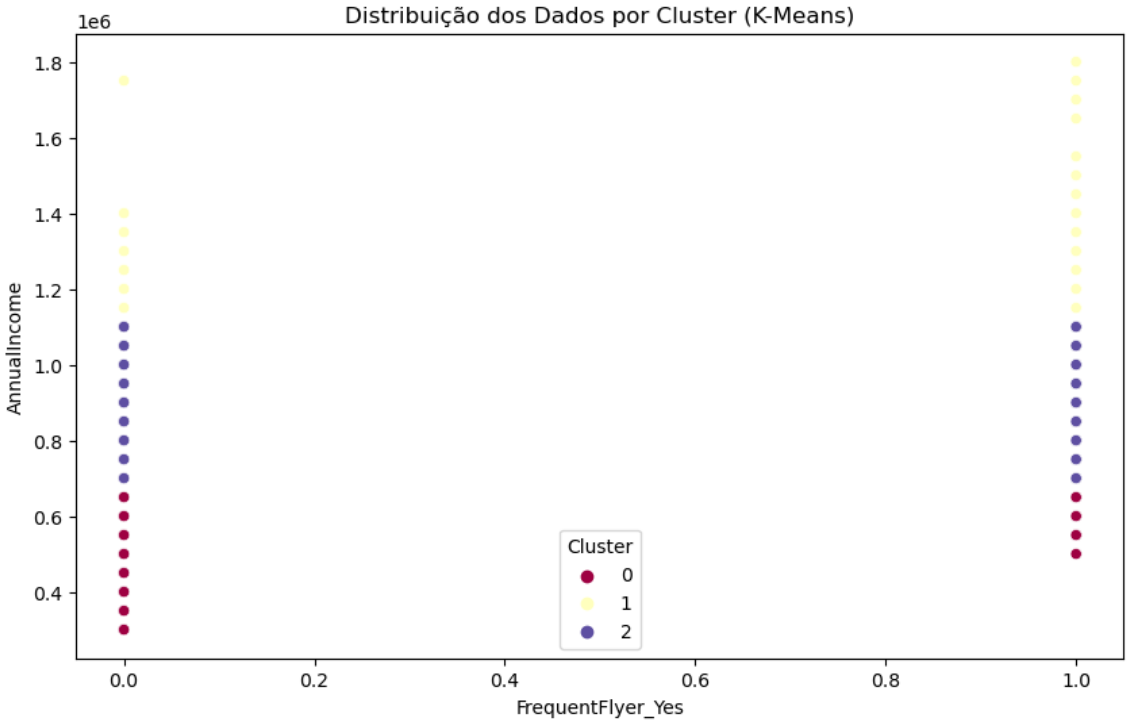
In [165]:
```
X_kmeans = X_encoded[["AnnualIncome","FrequentFlyer_Yes"]].copy()
kmeans = KMeans(n_clusters=3, random_state=0, n_init=10)

kmeans.fit(X_kmeans)
kmeans.cluster_centers_
kmeans.labels_
```

Out[165]:
```
array([0, 1, 0, ..., 1, 2, 0], dtype=int32)
```

The output `array([0, 1, 0, ..., 1, 2, 0], dtype=int32)` represents the cluster assignments for each observation in the dataset. Each number corresponds to the cluster that the k-means algorithm has assigned a particular data point to, based on the closest centroid.

To visualize it we will plot the distribution of data points based on, 'AnnualIncome' (on the y-axis) against 'FrequentFlyer_Yes' (on the x-axis), with data points colored according to the cluster they belong to (0, 1, or 2), as explained above.

In [166]:
```
X_kmeans['Cluster'] = kmeans.labels_

plt.figure(figsize=(10, 6))
sns.scatterplot(y='AnnualIncome', x='FrequentFlyer_Yes', hue='Cluste
plt.title('Distribuição dos Dados por Cluster (K-Means)')
plt.show()
```

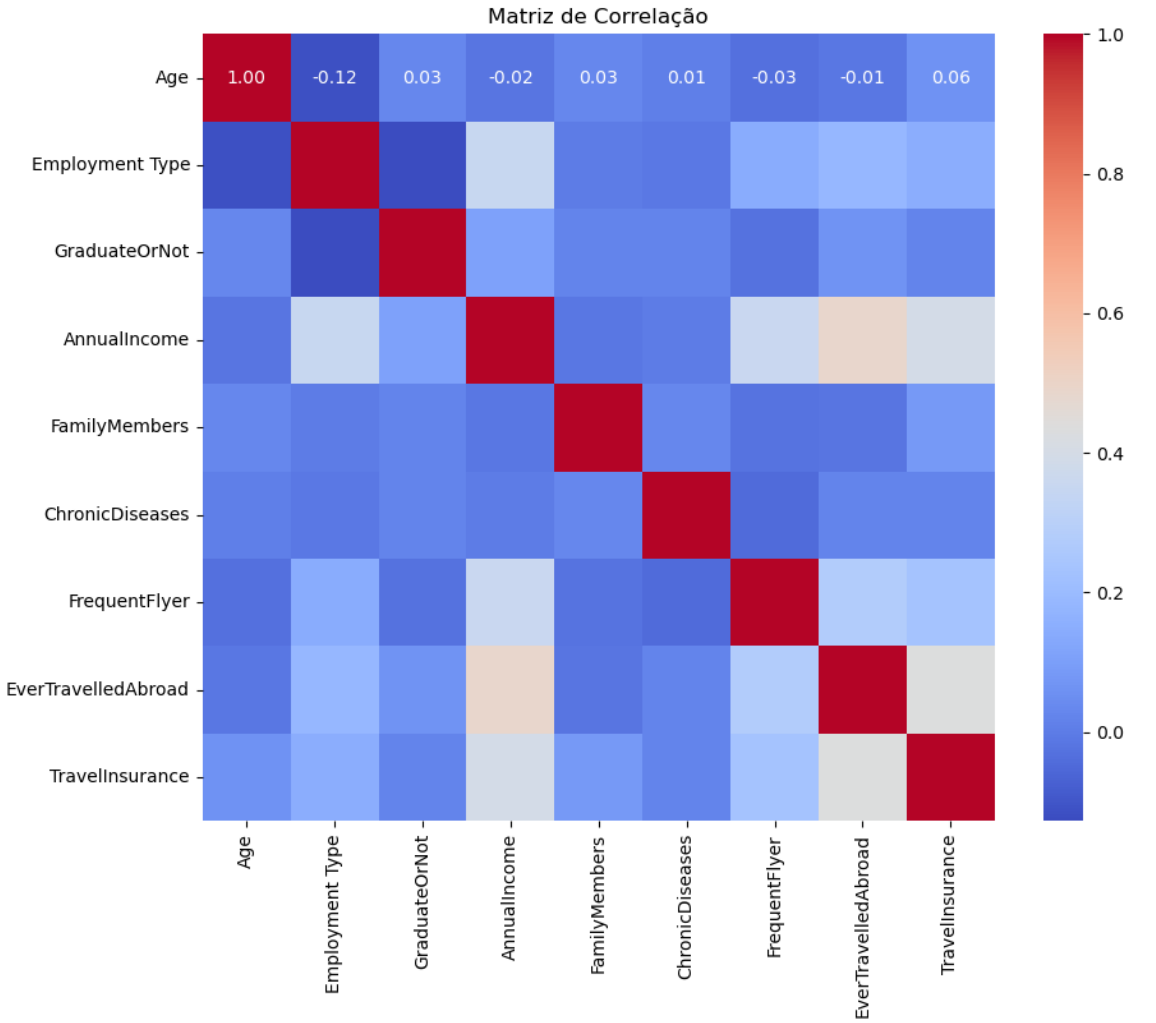


- There is a vertical line of data points at `FrequentFlyer_Yes = 0` and another at `FrequentFlyer_Yes = 1`, which indicates that the `FrequentFlyer_Yes` status does not have a varying degree but is rather a categorical status.
- Interestingly, the `FrequentFlyer_Yes` variable does not seem to significantly affect the clustering, as the clusters are distributed along the entire range of the `AnnualIncome` for both frequent and non-frequent flyers.

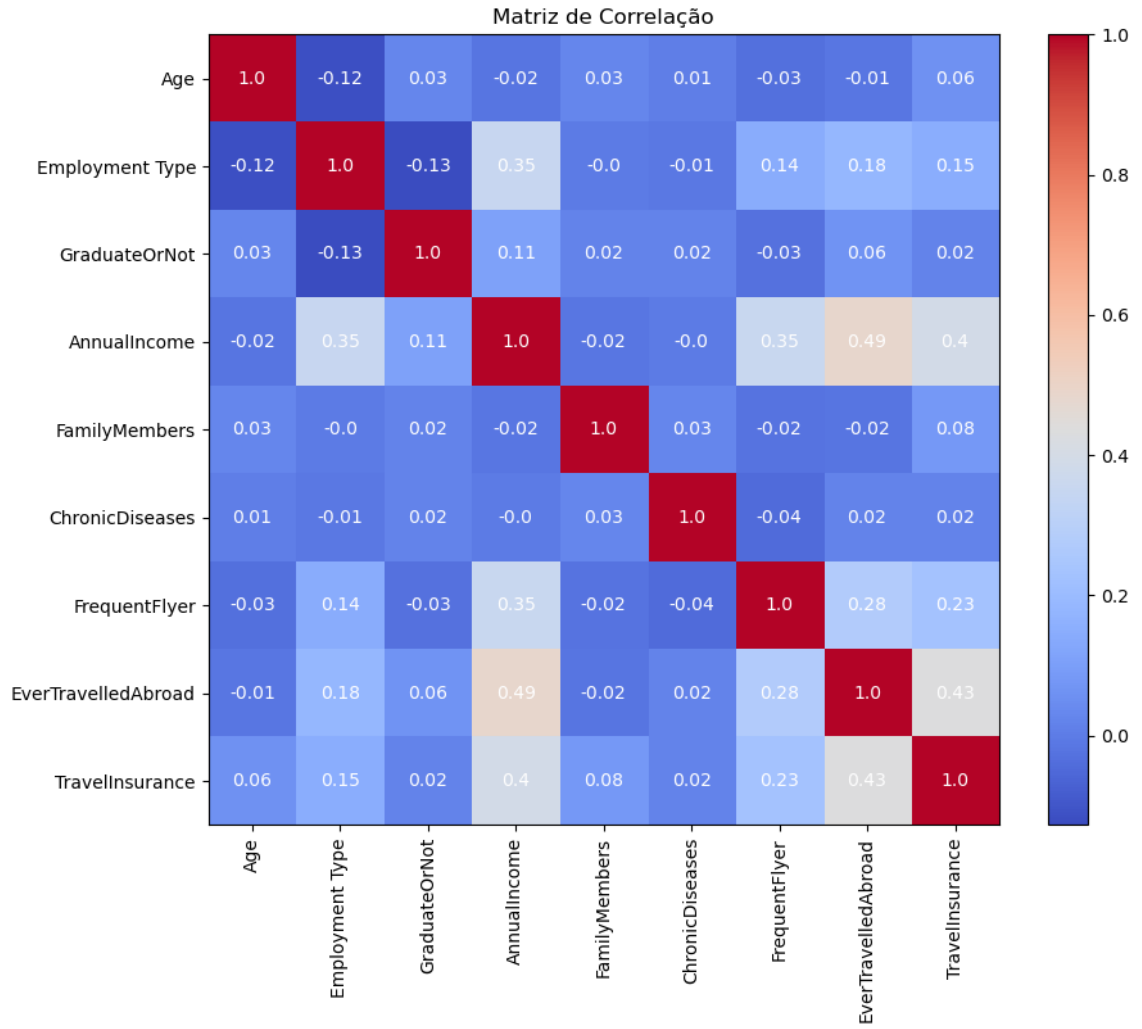
This indicates that `AnnualIncome` is likely the more dominant feature in determining the clustering, while `FrequentFlyer` status, although used in the clustering, does not differentiate the clusters on its own. This could suggest that income level is a more significant factor than frequent flyer status in this particular segmentation.

In [167]:
```
# Transformação da classe para discreto
le = LabelEncoder()
data1['TravelInsurance'] = le.fit_transform(data1['TravelInsurance'])
data1['Employment Type'] = le.fit_transform(data1['Employment Type'])
data1['GraduateOrNot'] = le.fit_transform(data1['GraduateOrNot'])
data1['FrequentFlyer'] = le.fit_transform(data1['FrequentFlyer'])
data1['EverTravelledAbroad'] = le.fit_transform(data1['EverTravelledA
```

In [168]:
```
# Correlação entre as variáveis
correlation_matrix = data1.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f
plt.title('Matriz de Correlação')
plt.show()
```



In [169]:
```
correlation_matrix = data1.corr()
plt.figure(figsize=(10, 8))
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='none')
plt.colorbar()
plt.xticks(range(len(correlation_matrix)), correlation_matrix.columns
plt.yticks(range(len(correlation_matrix)), correlation_matrix.columns
# Annotating the heatmap
for i in range(len(correlation_matrix.columns)):
    for j in range(len(correlation_matrix.columns)):
        text = plt.text(j, i, round(correlation_matrix.iloc[i, j], 2)
                        ha="center", va="center", color="w")
plt.title('Matriz de Correlação')
plt.show()
```



This is a **heatmap** that represents the **correlation matrix** of different variables in our dataset.

Interpretation

- The diagonal from the top-left to the bottom-right is filled with red squares indicating a perfect correlation of 1, as it's the correlation of each variable with itself.
- Cells in red represent a positive correlation, where the correlation coefficient is closer to 1. This means that as one variable increases, the other variable tends to also increase.
- Cells in blue represent a negative correlation, where the correlation coefficient is closer to -1. This indicates that as one variable increases, the other variable tends to

decrease.
- Cells in white or neutral color represent no or very little correlation, where the correlation coefficient is closer to 0. This indicates that the variables do not have a strong linear relationship.

Insights and Patterns:

- `AnnualIncome` has a moderate positive correlation (0.35) with `EverTravelledAbroad`. This could imply that individuals with higher annual incomes are more likely to have traveled abroad.
- `TravelInsurance` also shows a moderate positive correlation with `AnnualIncome` (0.4) and `EverTravelledAbroad` (0.43). This suggests that people who have higher incomes or those who have traveled abroad are more likely to purchase travel insurance.
- `Employment Type` has a positive correlation with `AnnualIncome` (0.35) and `FrequentFlyer` (0.14), although the correlation is not very strong. This might indicate that the type of employment could have some relation to income level and flying frequency.

Student Stress Factors - Dataset2

In [170]:
```
data2 = pd.read_csv("StressLevelDataset.csv")
data2.describe()
```

Out[170]:

	anxiety_level	self_esteem	mental_health_history	depression	headache	blood_pre
count	1100.000000	1100.000000	1100.000000	1100.000000	1100.000000	1100.0
mean	11.063636	17.777273	0.492727	12.555455	2.508182	2.1
std	6.117558	8.944599	0.500175	7.727008	1.409356	0.8
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.0
25%	6.000000	11.000000	0.000000	6.000000	1.000000	1.0
50%	11.000000	19.000000	0.000000	12.000000	3.000000	2.0
75%	16.000000	26.000000	1.000000	19.000000	3.000000	3.0
max	21.000000	30.000000	1.000000	27.000000	5.000000	3.0

8 rows × 21 columns

In [171]:
```
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1100 entries, 0 to 1099
Data columns (total 21 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   anxiety_level                 1100 non-null   int64
 1   self_esteem                   1100 non-null   int64
 2   mental_health_history         1100 non-null   int64
 3   depression                    1100 non-null   int64
 4   headache                      1100 non-null   int64
 5   blood_pressure                1100 non-null   int64
 6   sleep_quality                 1100 non-null   int64
 7   breathing_problem             1100 non-null   int64
 8   noise_level                   1100 non-null   int64
 9   living_conditions             1100 non-null   int64
 10  safety                        1100 non-null   int64
 11  basic_needs                   1100 non-null   int64
 12  academic_performance          1100 non-null   int64
 13  study_load                    1100 non-null   int64
 14  teacher_student_relationship  1100 non-null   int64
 15  future_career_concerns        1100 non-null   int64
 16  social_support                1100 non-null   int64
 17  peer_pressure                 1100 non-null   int64
 18  extracurricular_activities    1100 non-null   int64
 19  bullying                      1100 non-null   int64
 20  stress_level                  1100 non-null   int64
dtypes: int64(21)
memory usage: 180.6 KB
```

In [172]:
```
data1.isnull().sum()
```

Out[172]:
```
Age                    0
Employment Type        0
GraduateOrNot          0
AnnualIncome           0
FamilyMembers          0
ChronicDiseases        0
FrequentFlyer          0
EverTravelledAbroad    0
TravelInsurance        0
dtype: int64
```

In [173]:
```
data2.head(5)
```

Out[173]:

	anxiety_level	self_esteem	mental_health_history	depression	headache	blood_pressure
0	14	20	0	11	2	1
1	15	8	1	15	5	3
2	12	18	1	14	2	1
3	16	12	1	15	4	3
4	16	28	0	7	2	3

5 rows × 21 columns

Data Preparation

In [174]:
```
X = data2.drop('stress_level', axis=1)
y = data2['stress_level']
```

`stress_level` is the target variable I'm trying to predict, and X contains the features.

In [175]:
```
#Normalize data
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)
```

Now we split the dataset into a training set and a test set. The `test_size=0.2` means that 20% of the data is used for testing

In [176]:
```
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y,
```

Models

In [177]:
```
# Decision Tree model
dt_model = DecisionTreeClassifier(criterion='gini', splitter='best',r
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)

# KNN model
knn_model = KNeighborsClassifier(n_neighbors=3, weights='uniform', me
knn_model.fit(X_train, y_train)
knn_pred = knn_model.predict(X_test)
```

So far we prepared the dataset, normalized it, splitted it into training and test sets, and then trains and tests two different models, **Decision Tree** and **KNN** to predict stress levels.

Model Evaluation

- Both models are evaluated on the test data using accuracy and a confusion matrix.

In [178]:
```
# Acurácia e Matriz de Confusão para Decision Tree
dt_accuracy = accuracy_score(y_test, dt_pred)
dt_conf_matrix = confusion_matrix(y_test, dt_pred)
print("Acurácia Decision Tree:", dt_accuracy)
print("Matriz de Confusão Decision Tree:")
print(dt_conf_matrix)

# Acurácia e Matriz de Confusão para KNN
knn_accuracy = accuracy_score(y_test, knn_pred)
knn_conf_matrix = confusion_matrix(y_test, knn_pred)
print("\nAcurácia KNN:", knn_accuracy)
print("Matriz de Confusão KNN:")
print(knn_conf_matrix)
```

```
Acurácia Decision Tree: 0.8863636363636364
Matriz de Confusão Decision Tree:
[[68  5  3]
 [ 2 66  5]
 [ 5  5 61]]

Acurácia KNN: 0.8454545454545455
Matriz de Confusão KNN:
[[64  6  6]
 [ 6 63  4]
 [ 8  4 59]]
```

- Decision Tree Classifier:
  - Accuracy: 88.64%
  - Confusion Matrix:
    - Class 1: 68 true negatives (correctly predicted no insurance), 5 false positives (incorrectly predicted insurance).
    - Class 2: 2 false positives (incorrectly predicted insurance), 66 true negatives (correctly predicted no insurance), 5 incorrectly predicted as Class 3.
    - Class 3: 5 incorrectly predicted as Class 1, 5 incorrectly predicted as Class 2, 61 correctly predicted as Class 3 (True Positive).
- K-Nearest Neighbors (KNN) Classifier:
  - Accuracy: 84.55%
  - Confusion Matrix:
    - Class 1: 64 correctly predicted as Class 1 (True Positive), 6 incorrectly predicted as Class 2, 6 incorrectly predicted as Class 3.
    - Class 2: 6 incorrectly predicted as Class 1, 63 correctly predicted as Class 2 (True Positive), 4 incorrectly predicted as Class 3.
    - Class 3: 8 incorrectly predicted as Class 1, 4 incorrectly predicted as Class 2, 59 correctly predicted as Class 3 (True Positive).

Analysis

- The Decision Tree model has a higher accuracy (about 88.64%) compared to the KNN model (about 84.55%). This suggests that, for this particular dataset and problem, the Decision Tree model might be a better predictor of stress levels.
- The confusion matrices provide a more detailed view, showing how each class was predicted versus its actual value.

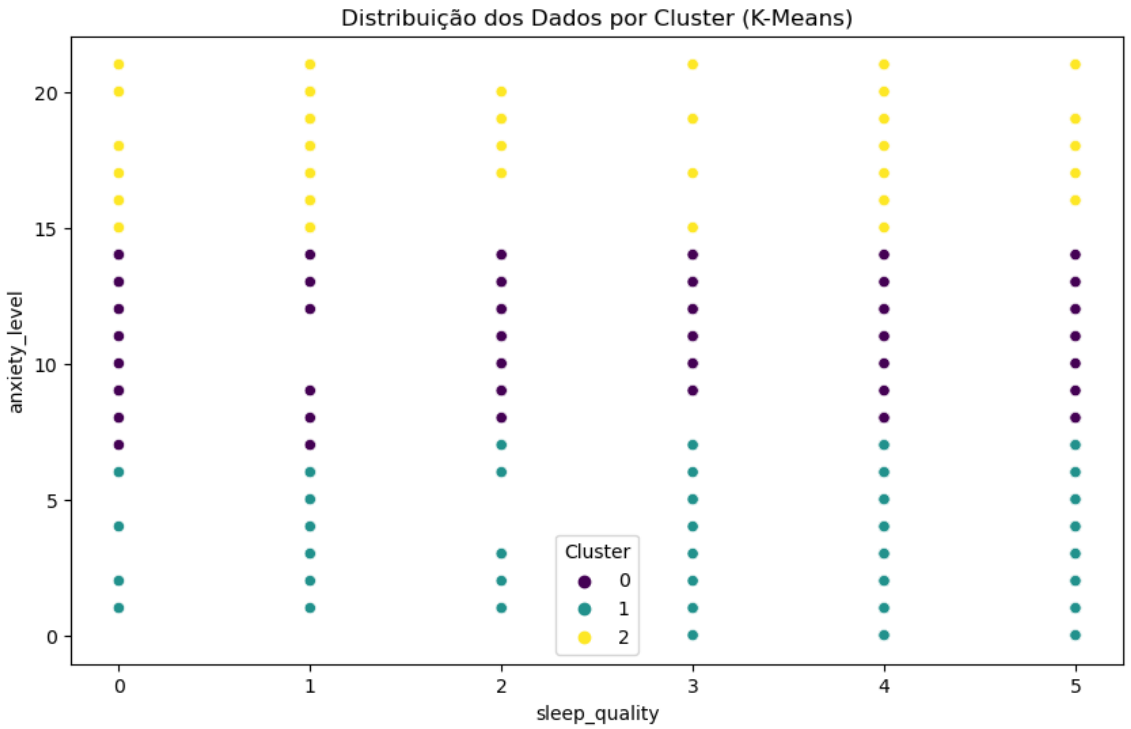
In [179]:
```
X_kmeans = data2[["anxiety_level","sleep_quality"]].copy()

kmeans = KMeans(n_clusters=3, random_state=0)

kmeans.fit(X_kmeans)
kmeans.cluster_centers_
kmeans.labels_
```

Out[179]:
```
array([0, 2, 0, ..., 1, 2, 2], dtype=int32)
```

The output `array([0, 2, 0, ..., 1, 2, 2], dtype=int32)` represents the cluster assignments for each observation in the dataset. Each number corresponds to the cluster that the k-means algorithm has assigned a particular data point to, based on the closest centroid.

To visualize it we will plot the distribution of data points based on, 'anxiety_level' (on the y-axis) against 'sleep_quality' (on the x-axis), with data points colored according to the cluster they belong to (0, 1, or 2), as explained above.

In [183]:
```
X_kmeans['Cluster'] = kmeans.labels_

plt.figure(figsize=(10, 6))
sns.scatterplot(y='anxiety_level', x='sleep_quality', hue='Cluster',
plt.title('Distribuição dos Dados por Cluster (K-Means)')
plt.show()
```



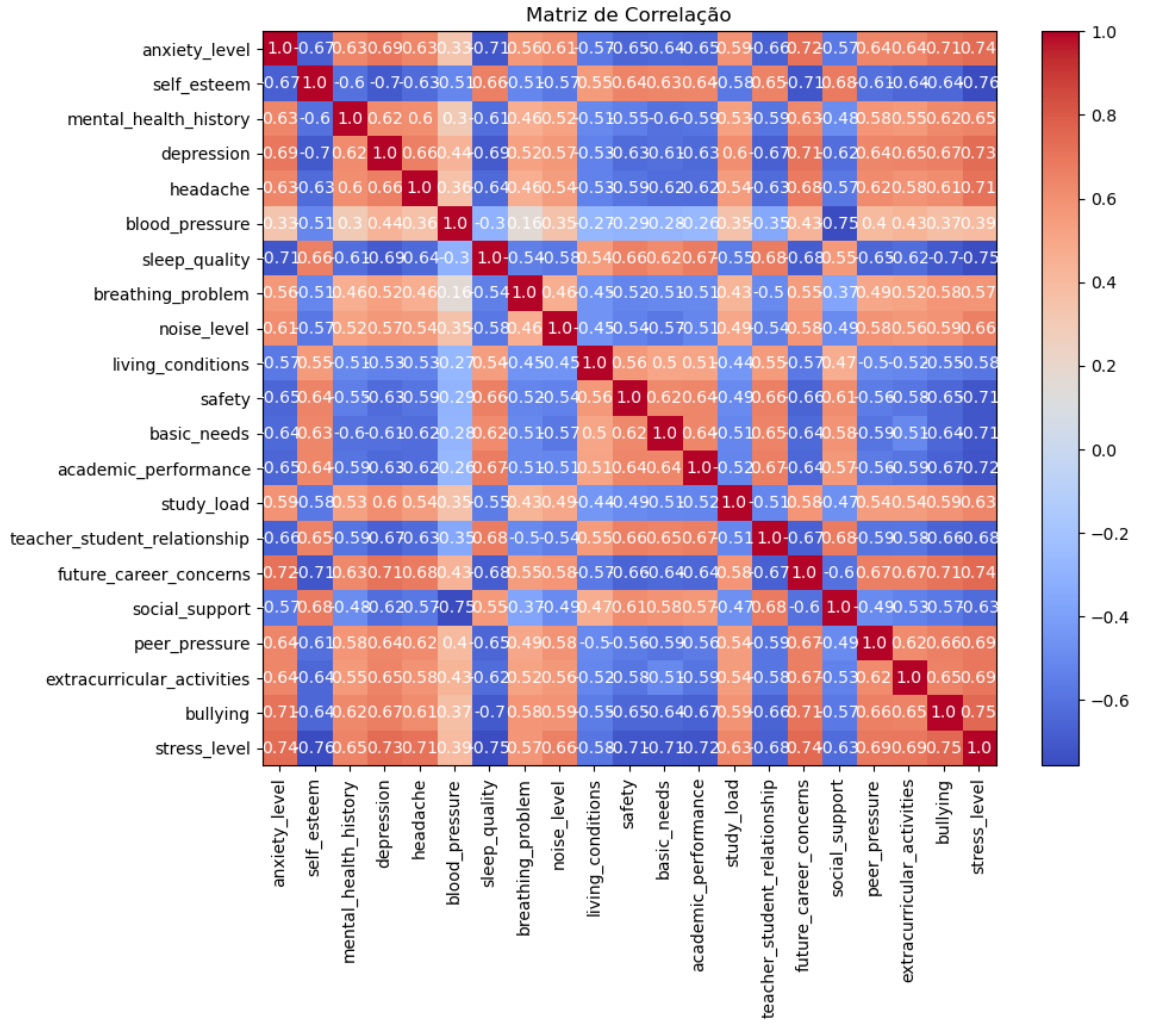
- **Green Dots**: This cluster is characterized by low anxiety levels and higher sleep quality. It seems to represent a group that is less stressed or anxious and generally sleeps well.
- **Yellow Dots**: Individuals in this cluster have higher anxiety levels. Interestingly, their sleep quality varies across the spectrum, which might indicate that while anxiety is high, its impact on sleep is not consistent for everyone in this group.
- **Purple Dots**: This cluster groups individuals with moderate levels of anxiety and sleep

quality. It's the middle ground between the two extremes, possibly indicating a balanced state or an average stress level.

Results

- There's a clear separation between the clusters based on anxiety levels. This separation is less distinct with sleep quality, particularly between Clusters 1 and 2, which suggests that anxiety level is a stronger clustering feature in our context.
- `Anxiety` seems to be a significant predictor for clustering, which might imply that it's a critical factor in stress-related research and interventions.
- The distribution of `sleep quality` in `high-anxiety` individuals (Cluster 2) could indicate that interventions to improve sleep might need to be tailored differently for individuals within the same anxiety level.
- Aiming at reducing stress, individuals in Cluster 0 might benefit from anxiety-reduction strategies, while those in Cluster 2 might be a reference group for stress management practices.

In [181]:
```
correlation_matrix = data2.corr()

# Setting the figure size
plt.figure(figsize=(10, 8))

# Creating the heatmap
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='none')
plt.colorbar()

# Setting the tick marks with the column names
plt.xticks(range(len(correlation_matrix)), correlation_matrix.columns
plt.yticks(range(len(correlation_matrix)), correlation_matrix.columns

# Annotating the heatmap with correlation coefficients
for i in range(len(correlation_matrix.columns)):
    for j in range(len(correlation_matrix.columns)):
        text = plt.text(j, i, round(correlation_matrix.iloc[i, j], 2)
                        ha="center", va="center", color="w")

# Adding the title and showing the plot
plt.title('Matriz de Correlação')
plt.show()
```



Results

- `stress_level` has high positive correlations with `anxiety_level` ,

`mental_health_history`, `depression`, `bullying`, `future_career_concerns`, and `headache`. This suggests that as these factors increase, the stress level tends to increase as well, which is intuitive given the nature of these variables.

- There are moderate to high positive correlations within several factors such as between `depression` and `anxiety_level`, `mental_health_history`, `bullying`, and `future_career_concerns`. This indicates that these factors tend to increase together.

In [ ]: