Marriage Proposal Acceptance or Rejection Prediction using Machine Learning

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
df = pd.read_csv("marriage_proposal.csv")
```

In [4]:
```
df
```

Out[4]:

	Height	Age	Income	RomanticGestureScore	CompatibilityScore	CommunicationScore	DistanceKM	Response	AgeCategory
0	156	59	7977	3	1	1	45	1	Senior
1	169	32	5842	0	1	5	46	1	Middle-aged
2	178	42	17638	2	5	5	13	0	Middle-aged
3	164	78	8793	0	0	7	52	0	Senior
4	160	35	15262	6	0	0	9	1	Middle-aged
...	...	...	...	...	...	...	...	...	...
9995	162	76	12311	4	1	5	75	1	Senior
9996	162	75	6459	7	9	0	52	1	Senior
9997	166	70	9231	9	4	6	33	0	Senior
9998	176	78	12656	8	9	5	25	1	Senior
9999	156	68	5812	0	9	4	14	1	Senior

10000 rows × 9 columns

In [5]:
```
df.shape
```

Out[5]:
```
(10000, 9)
```

In [6]:
```
df.columns
```

Out[6]:
```
Index(['Height', 'Age', 'Income', 'RomanticGestureScore', 'CompatibilityScore',
       'CommunicationScore', 'DistanceKM', 'Response', 'AgeCategory'],
      dtype='object')
```

In [7]:
```
df.duplicated().sum()
```

Out[7]:
```
0
```

In [8]:
```python
df.isnull().sum()
```

Out[8]:
```
Height                    0
Age                       0
Income                    0
RomanticGestureScore      0
CompatibilityScore        0
CommunicationScore        0
DistanceKM                0
Response                  0
AgeCategory             153
dtype: int64
```

In [9]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   Height                10000 non-null  int64 
 1   Age                   10000 non-null  int64 
 2   Income                10000 non-null  int64 
 3   RomanticGestureScore  10000 non-null  int64 
 4   CompatibilityScore    10000 non-null  int64 
 5   CommunicationScore    10000 non-null  int64 
 6   DistanceKM            10000 non-null  int64 
 7   Response              10000 non-null  int64 
 8   AgeCategory           9847 non-null   object
dtypes: int64(8), object(1)
memory usage: 703.2+ KB
```

In [10]:
```python
df.describe()
```

Out[10]:

	Height	Age	Income	RomanticGestureScore	CompatibilityScore	CommunicationScore	DistanceKM
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	165.170500	49.878300	12441.388000	4.965000	4.589000	4.543400	49.879600
std	8.907635	17.599059	4310.645672	3.140376	2.859702	2.870564	28.598155
min	150.000000	20.000000	5000.000000	0.000000	0.000000	0.000000	1.000000
25%	157.000000	35.000000	8684.750000	2.000000	2.000000	2.000000	25.000000
50%	165.000000	50.000000	12432.000000	5.000000	5.000000	5.000000	50.000000
75%	173.000000	65.000000	16113.250000	8.000000	7.000000	7.000000	75.000000
max	180.000000	80.000000	19999.000000	10.000000	9.000000	9.000000	99.000000

In [11]:
```python
df.nunique()
```

Out[11]:
```
Height                    31
Age                       61
Income                  7307
RomanticGestureScore      11
CompatibilityScore        10
CommunicationScore        10
DistanceKM                99
Response                   2
AgeCategory                3
dtype: int64
```

In [12]:
```python
object_columns = df.select_dtypes(include=['object']).columns
print("Object type columns:")
print(object_columns)

numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns
print("\nNumerical type columns:")
print(numerical_columns)
```

```
Object type columns:
Index(['AgeCategory'], dtype='object')

Numerical type columns:
Index(['Height', 'Age', 'Income', 'RomanticGestureScore', 'CompatibilityScore',
       'CommunicationScore', 'DistanceKM', 'Response'],
      dtype='object')
```

In [13]:
```python
def classify_features(df):
    categorical_features = []
    non_categorical_features = []
    discrete_features = []
```

```python
    continuous_features = []

    for column in df.columns:
        if df[column].dtype == 'object':
            if df[column].nunique() < 10:
                categorical_features.append(column)
            else:
                non_categorical_features.append(column)
        elif df[column].dtype in ['int64', 'float64']:
            if df[column].nunique() < 10:
                discrete_features.append(column)
            else:
                continuous_features.append(column)

    return categorical_features, non_categorical_features, discrete_features, continuous_features
```

In [14]:
```python
categorical, non_categorical, discrete, continuous = classify_features(df)
```

In [15]:
```python
print("Categorical Features:", categorical)
print("Non-Categorical Features:", non_categorical)
print("Discrete Features:", discrete)
print("Continuous Features:", continuous)
```

```
Categorical Features: ['AgeCategory']
Non-Categorical Features: []
Discrete Features: ['Response']
Continuous Features: ['Height', 'Age', 'Income', 'RomanticGestureScore', 'CompatibilityScore', 'CommunicationSco
re', 'DistanceKM']
```

In [16]:
```python
null_counts = df.isnull().sum()
null_columns = null_counts[null_counts > 0].index.tolist()
```

In [17]:
```python
null_counts
```

Out[17]:
```
Height                    0
Age                       0
Income                    0
RomanticGestureScore      0
CompatibilityScore        0
CommunicationScore        0
DistanceKM                0
Response                  0
AgeCategory             153
dtype: int64
```

In [18]:
```python
null_columns
```

Out[18]:
```
['AgeCategory']
```

In [19]:
```python
for column in null_columns:
    null_count = null_counts[column]
    print(f"Column '{column}' has {null_count} null values.")
```

```
Column 'AgeCategory' has 153 null values.
```

In [20]:
```python
total_rows = len(df)
null_percentage = (null_counts / total_rows) * 100
```

In [21]:
```python
null_df = pd.DataFrame({
    'Column': null_counts.index,
    'Null Count': null_counts.values,
    'Null Percentage': null_percentage.values
})
```

In [22]:
```python
null_df = null_df.sort_values(by='Null Count', ascending=False)
```

In [23]:
```python
null_df
```

Out[23]:

	Column	Null Count	Null Percentage
8	AgeCategory	153	1.53
0	Height	0	0.00
1	Age	0	0.00
2	Income	0	0.00
3	RomanticGestureScore	0	0.00
4	CompatibilityScore	0	0.00
5	CommunicationScore	0	0.00
6	DistanceKM	0	0.00
7	Response	0	0.00

In [24]:
```
df['AgeCategory'] = df['AgeCategory'].fillna('Not Available')
```

In [25]:
```
df.isnull().sum()
```

Out[25]:
```
Height                  0
Age                     0
Income                  0
RomanticGestureScore    0
CompatibilityScore      0
CommunicationScore      0
DistanceKM              0
Response                0
AgeCategory             0
dtype: int64
```

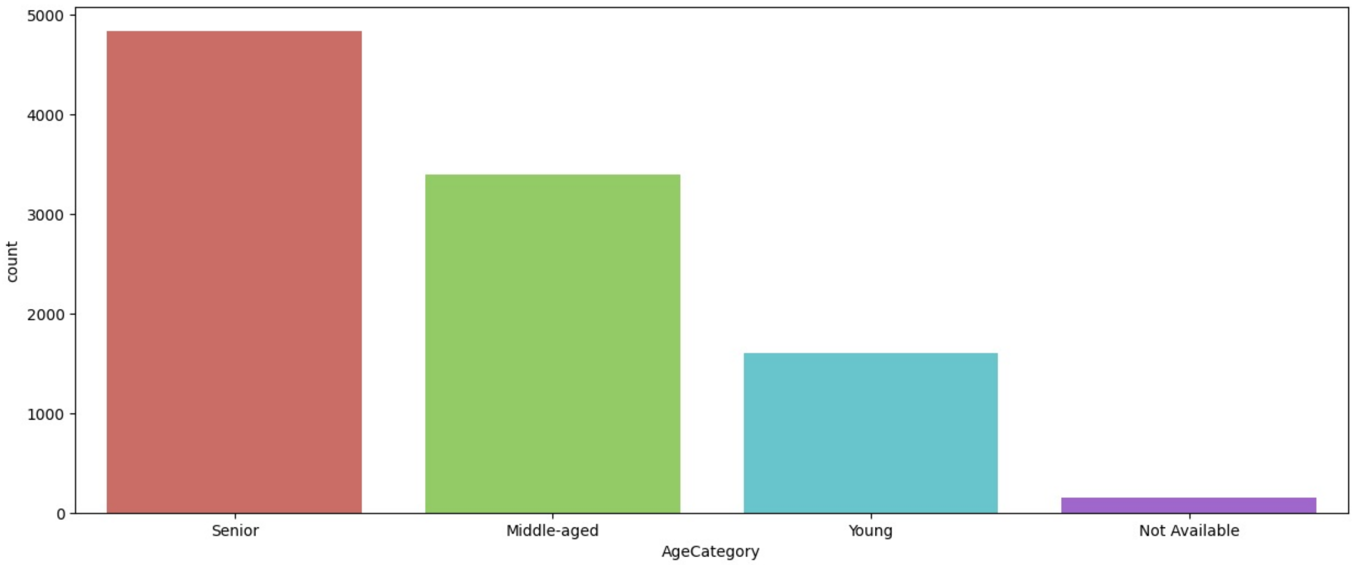
In [26]:
```
df['AgeCategory'].unique()
```

Out[26]:
```
array(['Senior', 'Middle-aged', 'Young', 'Not Available'], dtype=object)
```

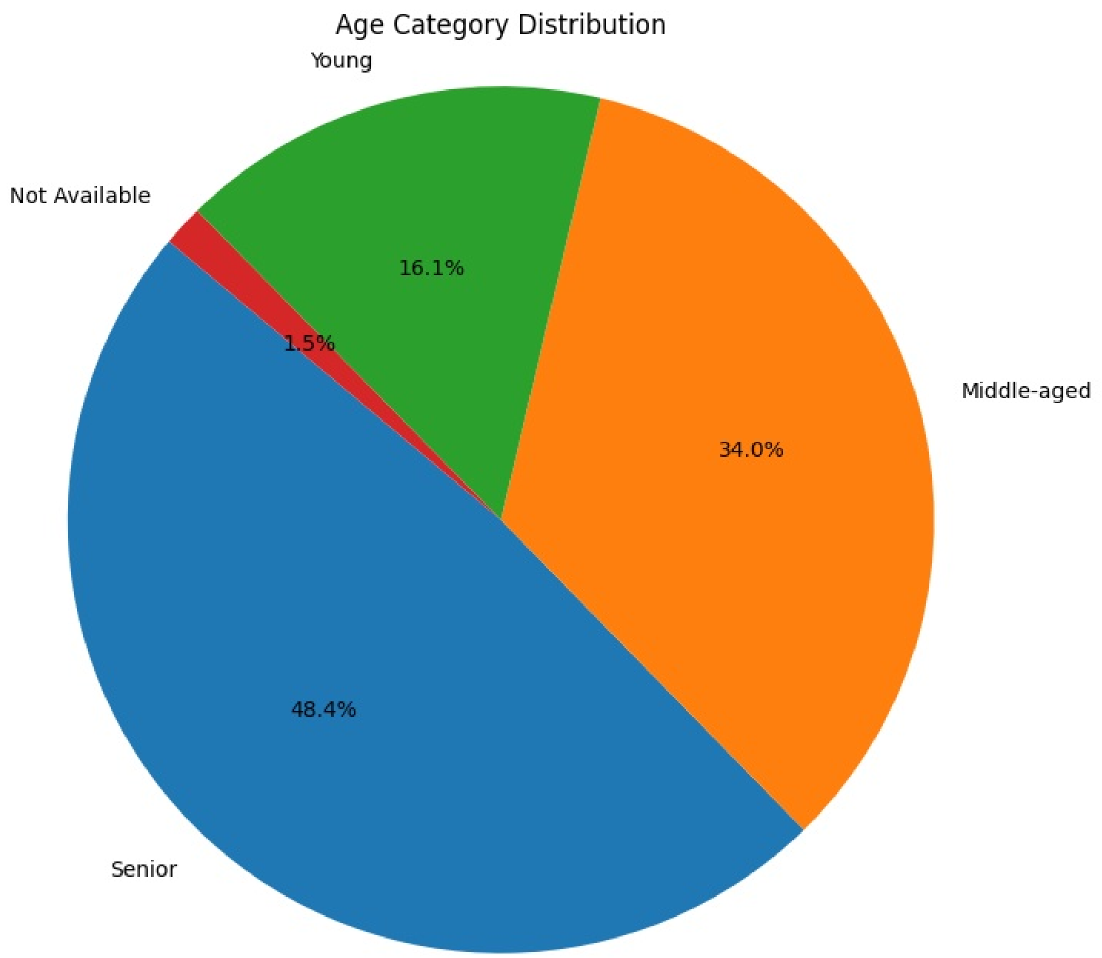
In [27]:
```
df['AgeCategory'].value_counts()
```

Out[27]:
```
AgeCategory
Senior           4843
Middle-aged      3399
Young            1605
Not Available     153
Name: count, dtype: int64
```

In [28]:
```
plt.figure(figsize=(15,6))
sns.countplot(x = 'AgeCategory', data = df, palette='hls')
plt.show()
```



In [29]:
```
age_counts = df['AgeCategory'].value_counts()

plt.figure(figsize=(8, 8))
plt.pie(age_counts, labels=age_counts.index, autopct='%1.1f%%', startangle=140)
plt.axis('equal')
plt.title('Age Category Distribution')
plt.show()
```



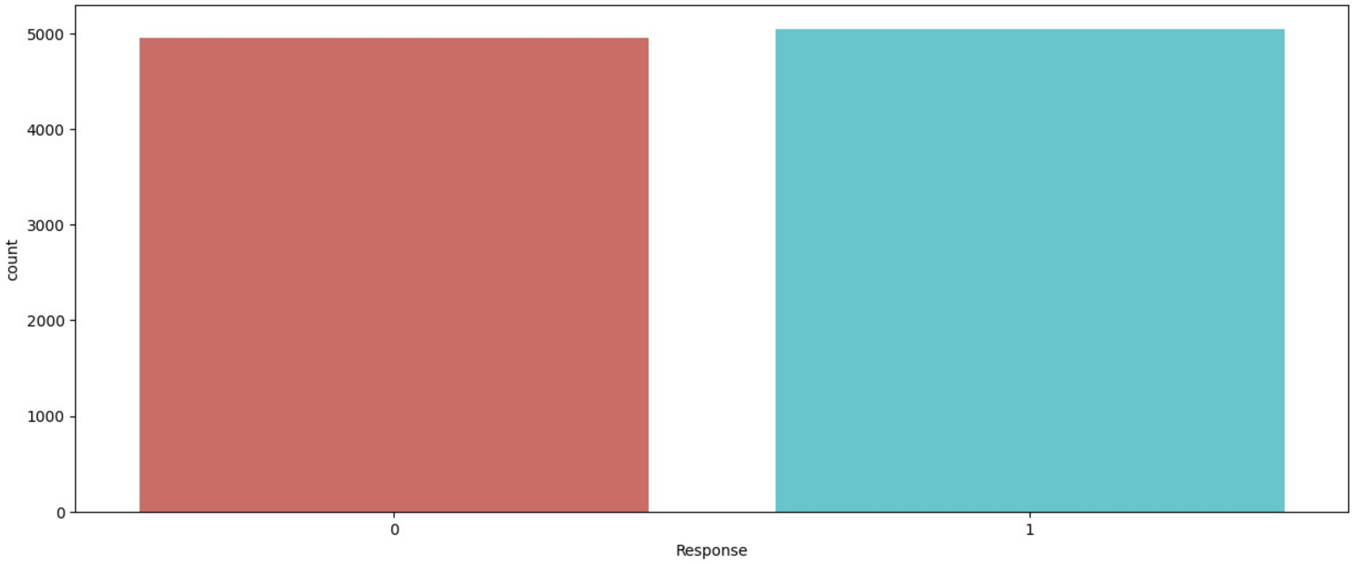
In [30]:
```
df['Response'].unique()
```

Out[30]:
```
array([1, 0], dtype=int64)
```

In [31]:
```
df['Response'].value_counts()
```

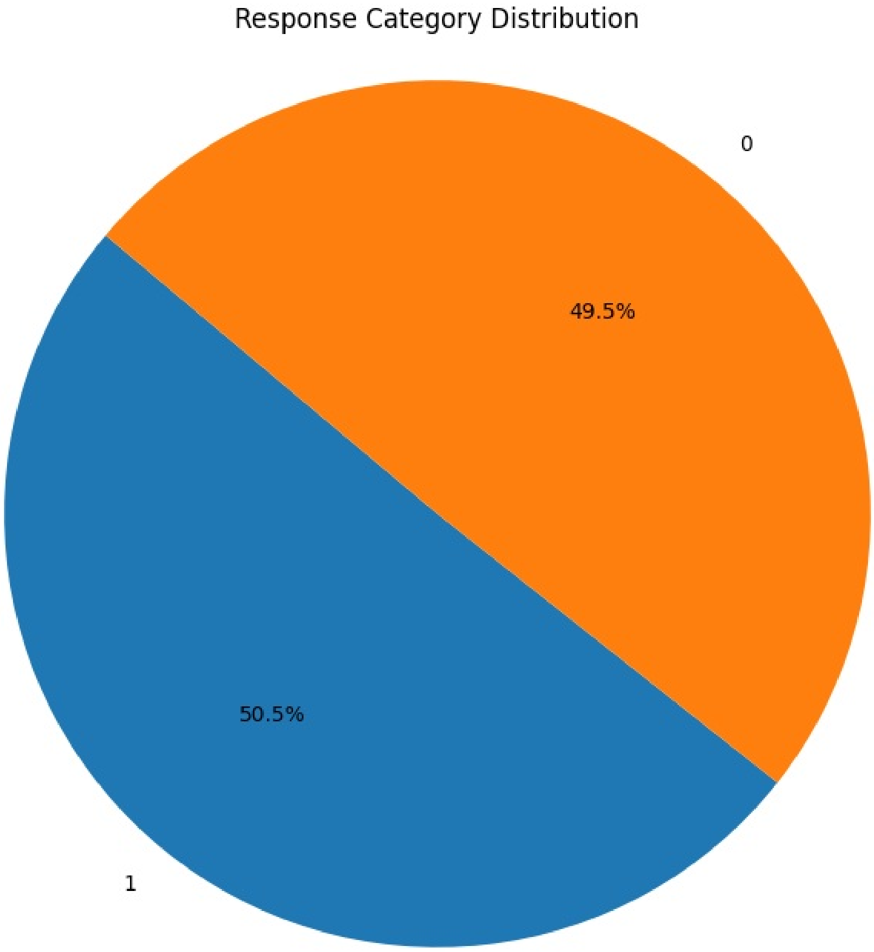
Out[31]:
```
Response
1    5047
0    4953
Name: count, dtype: int64
```

In [32]:
```
plt.figure(figsize=(15,6))
sns.countplot(x = 'Response', data = df, palette='hls')
plt.show()
```

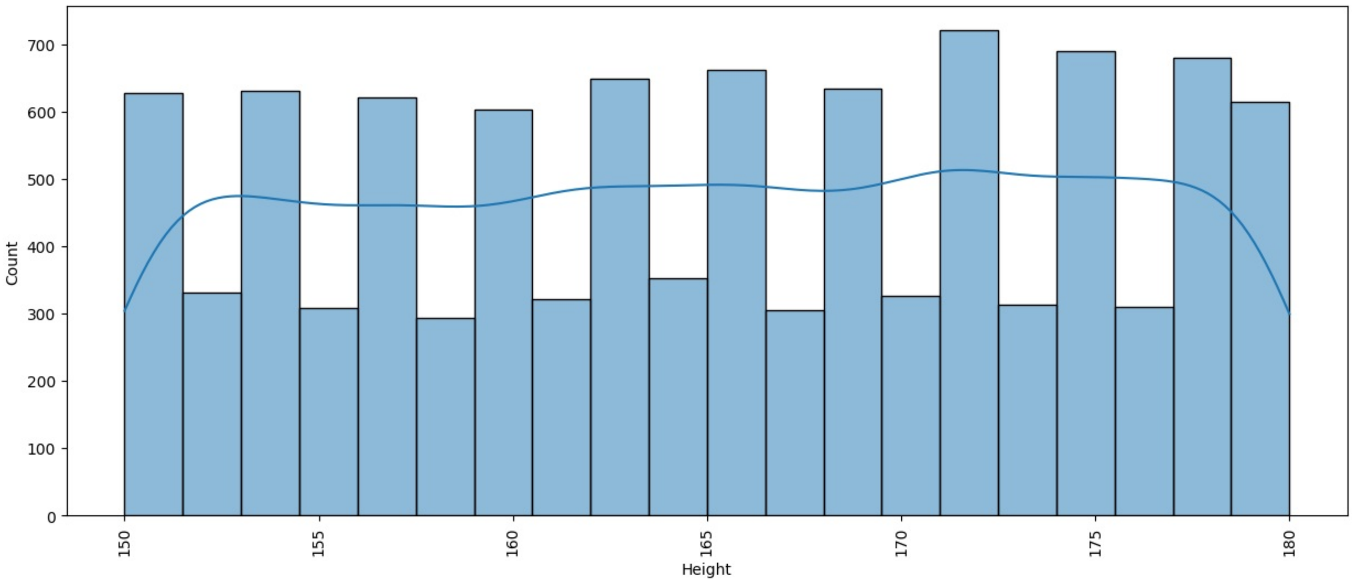


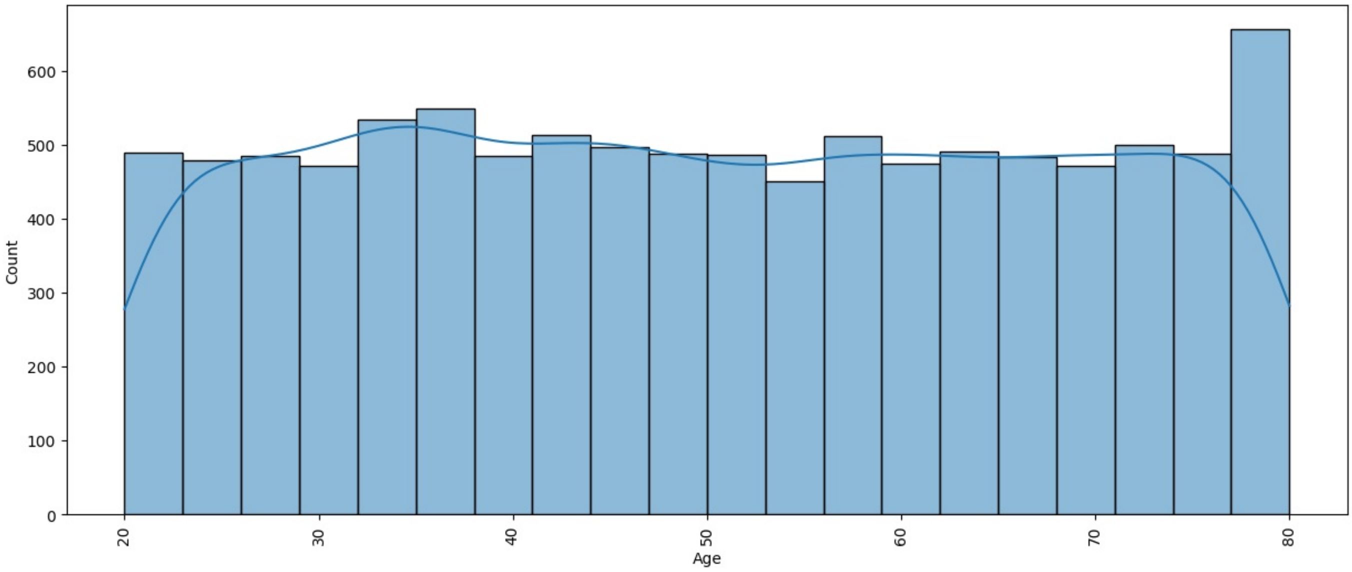
In [33]:
```
response_counts = df['Response'].value_counts()

plt.figure(figsize=(8, 8))
plt.pie(response_counts, labels=response_counts.index, autopct='%1.1f%%', startangle=140)
plt.axis('equal')
plt.title('Response Category Distribution')
```

```
plt.show()
```

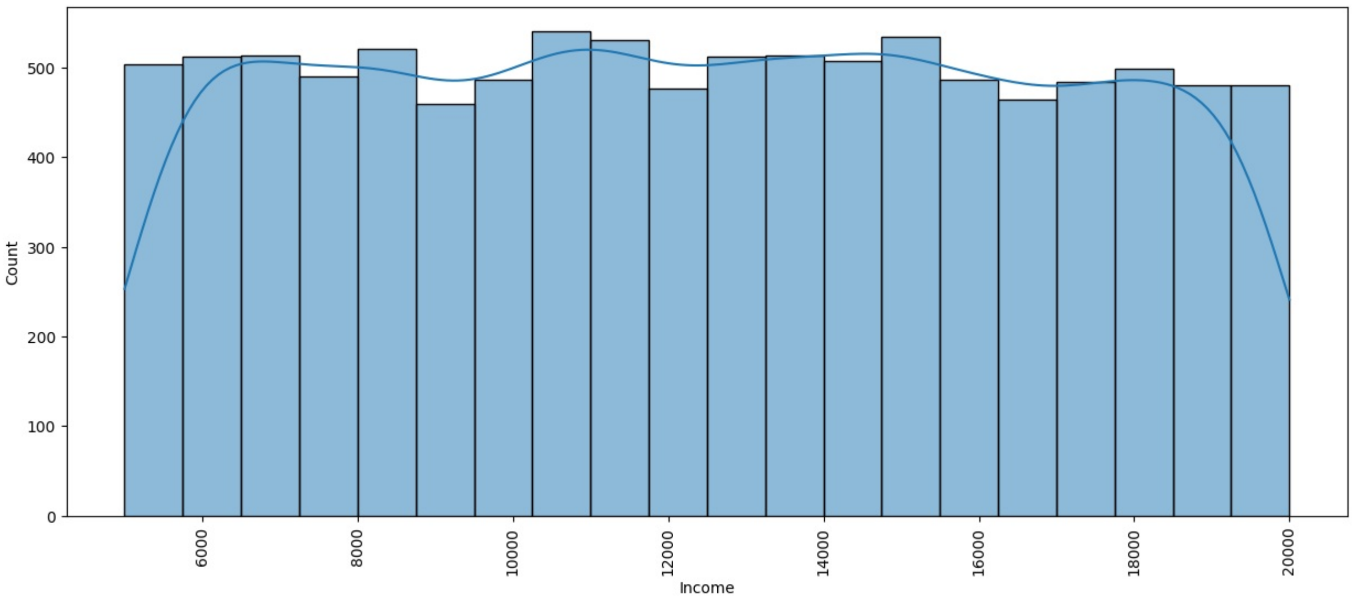


In [34]:
```
for i in continuous:
    plt.figure(figsize=(15,6))
    sns.histplot(df[i], bins = 20, kde = True, palette='hls')
    plt.xticks(rotation = 90)
    plt.show()
```

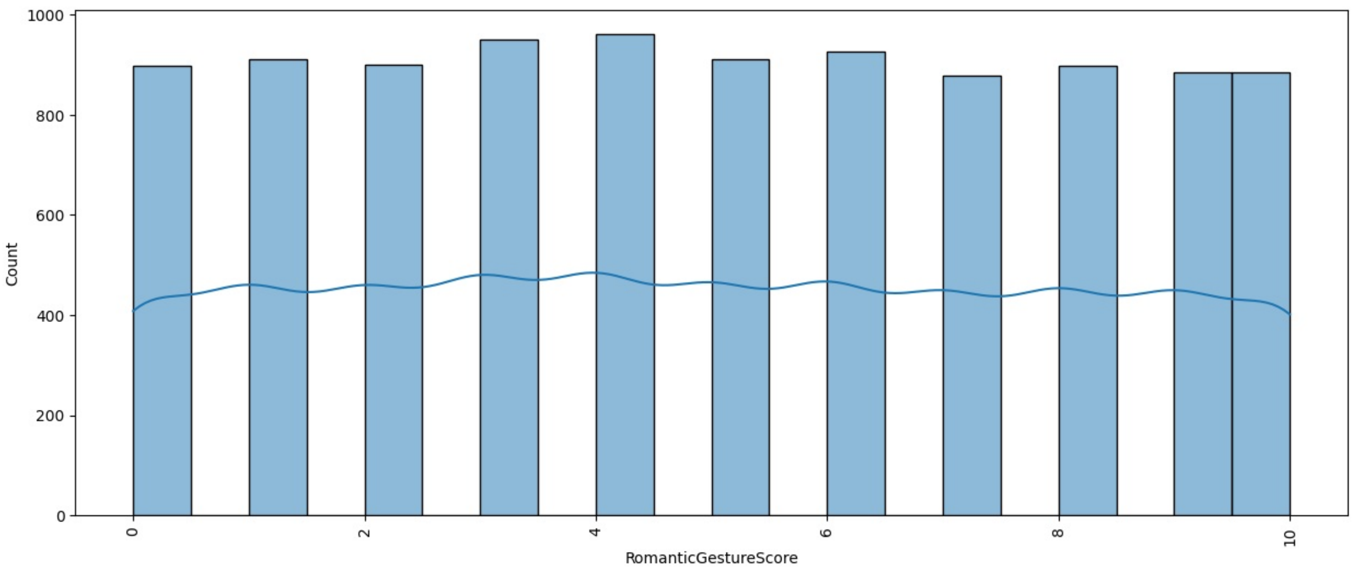




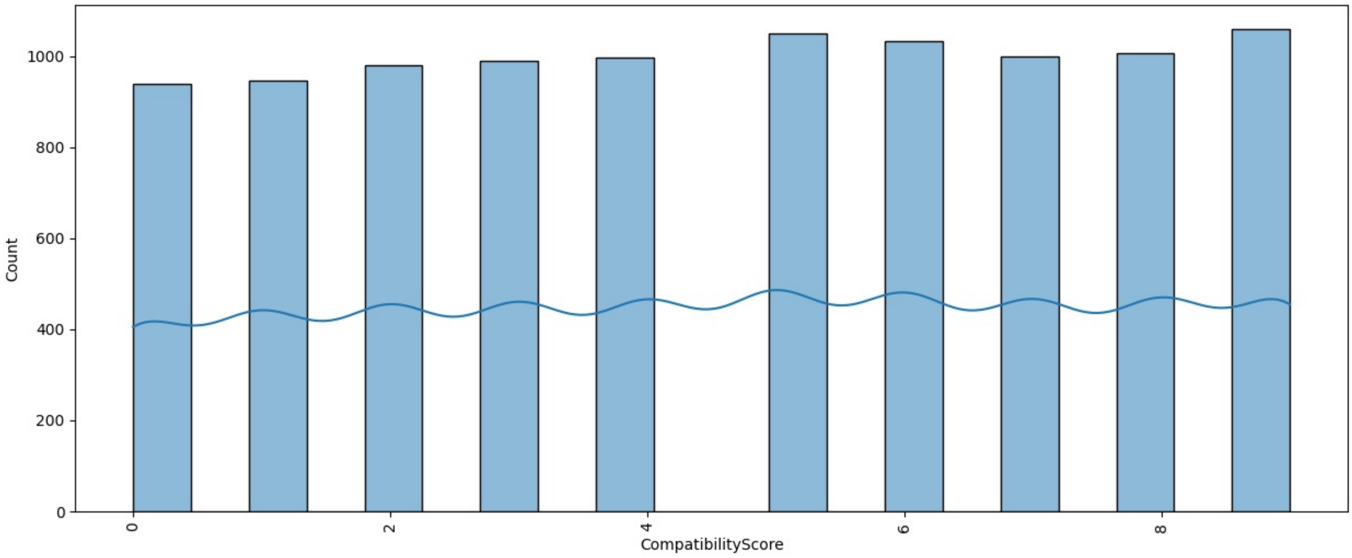
Count
Age

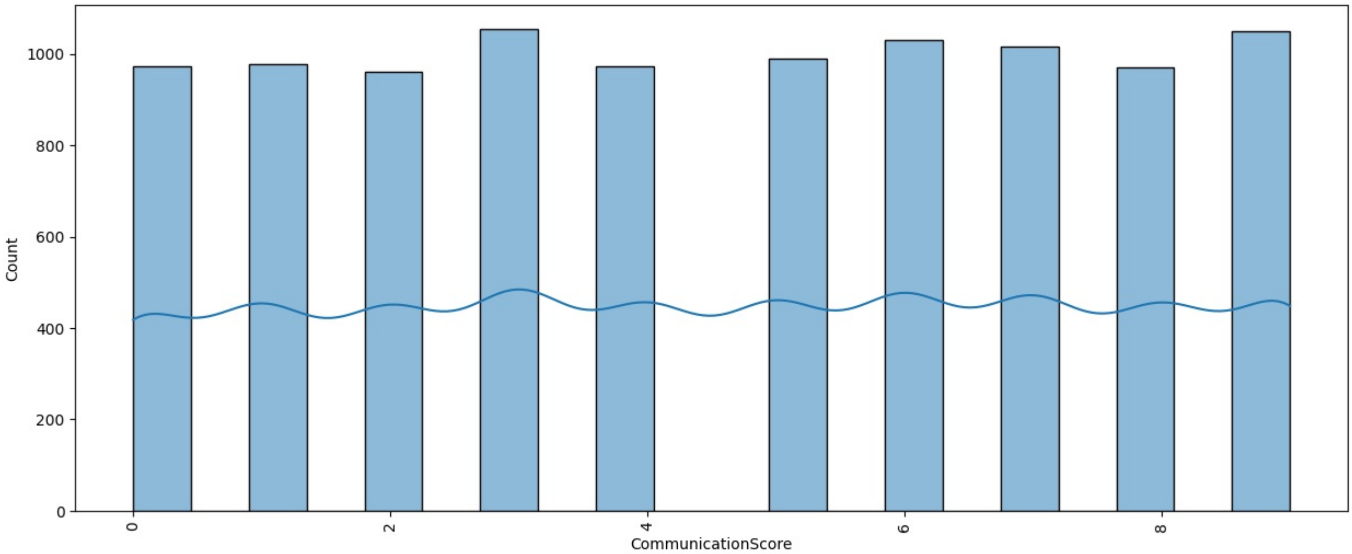


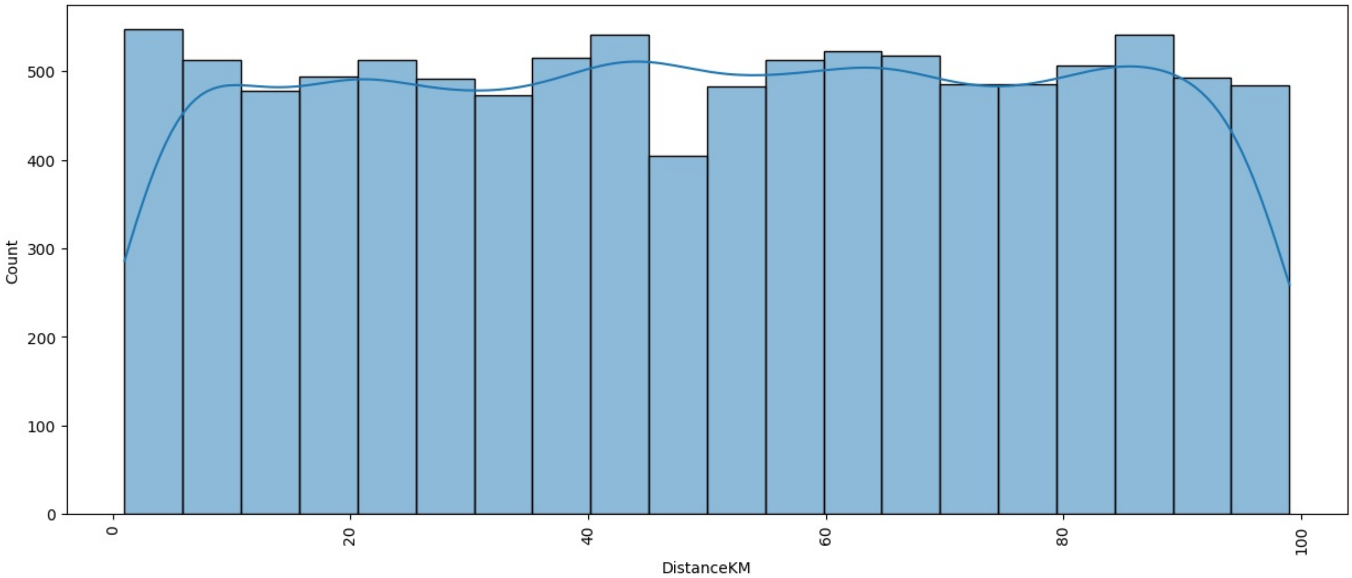
Count
Income



Count
RomanticGestureScore

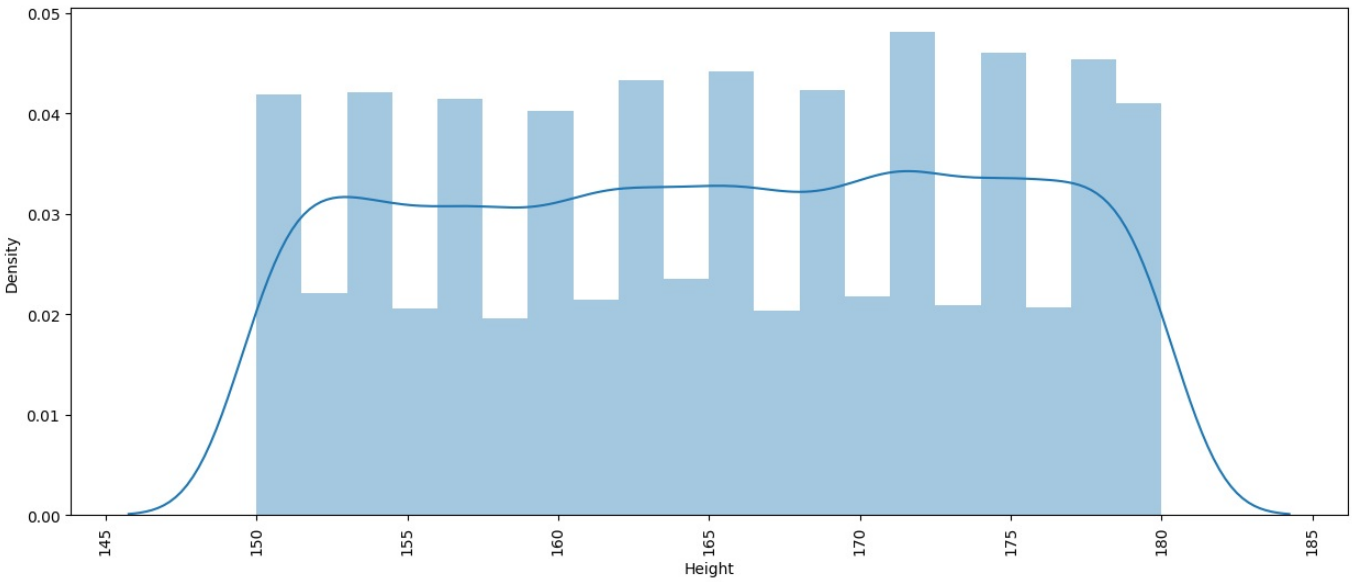


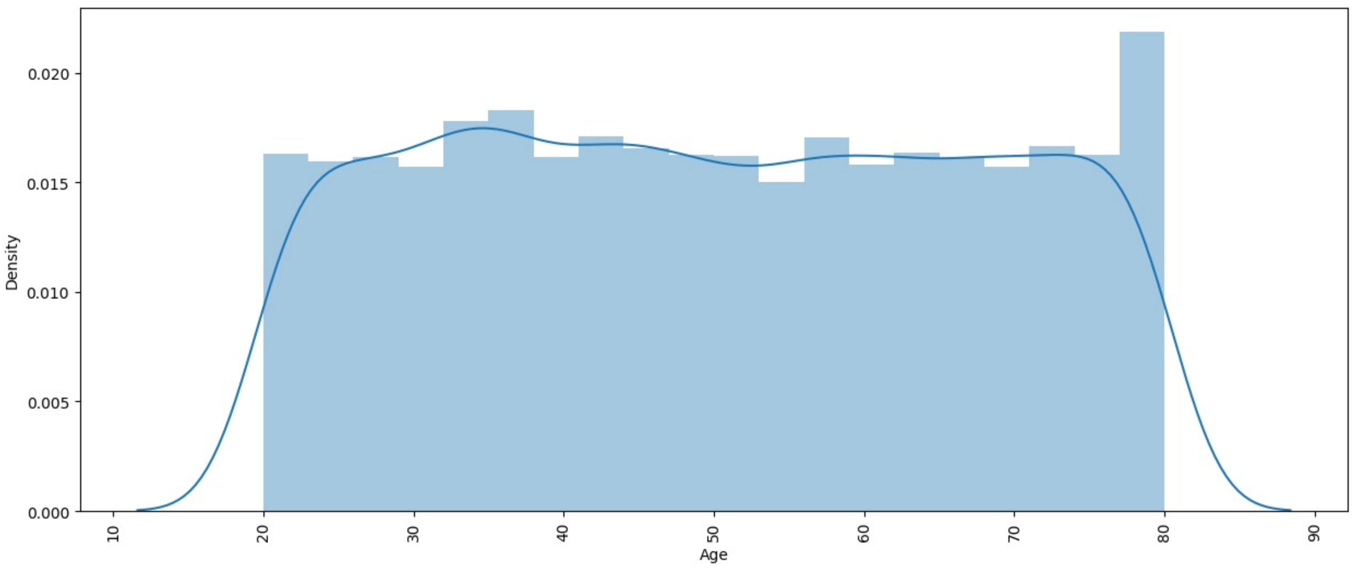


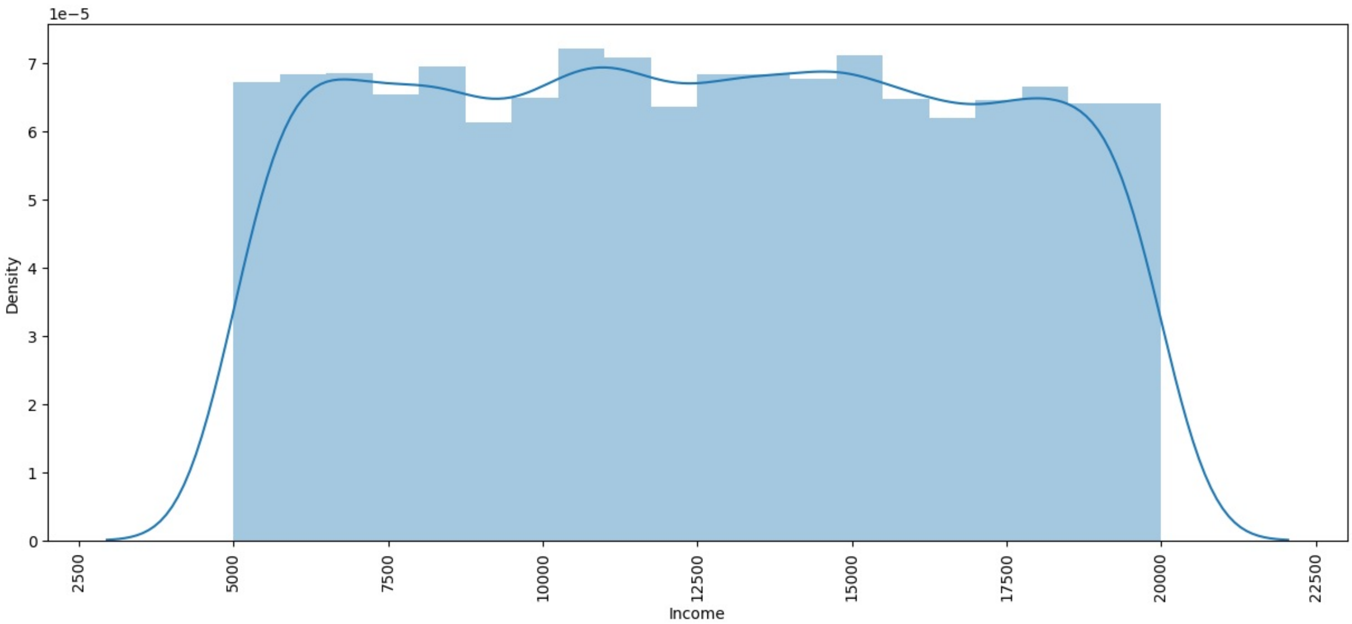


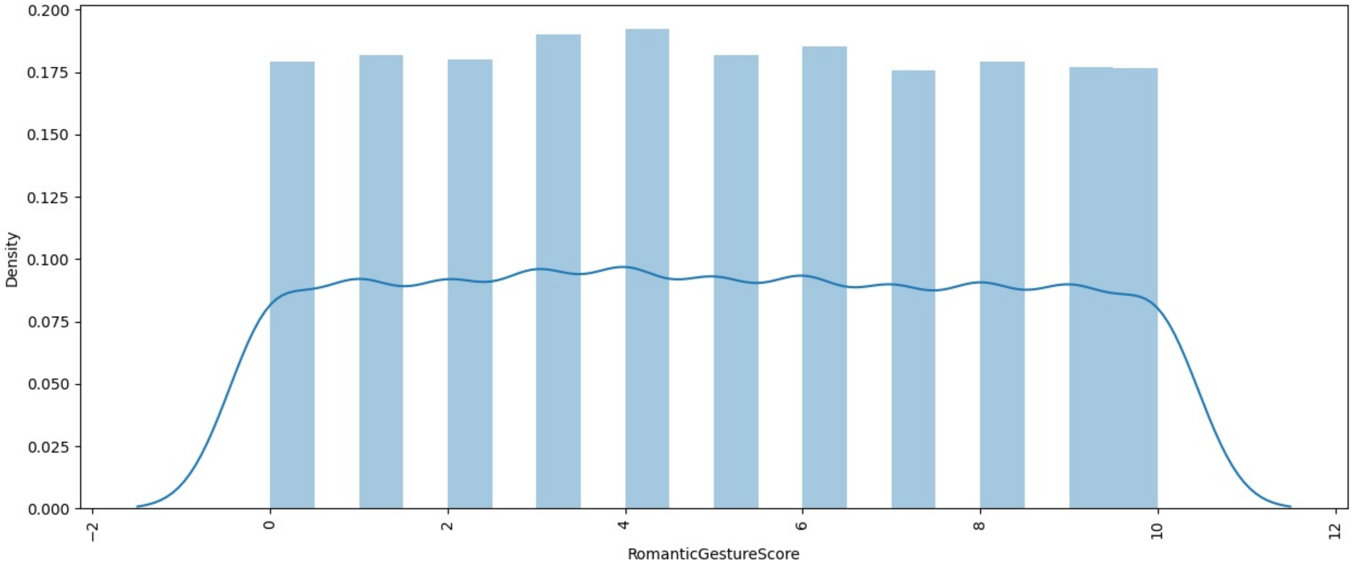
In [35]:

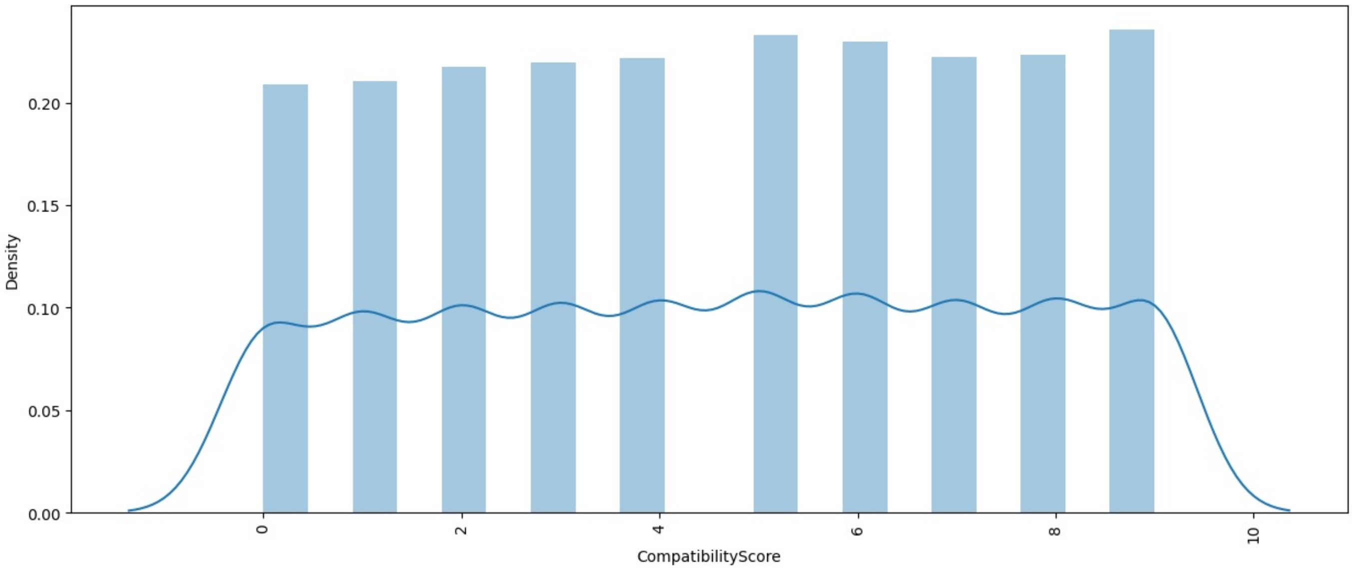
```
for i in continuous:
    plt.figure(figsize=(15,6))
    sns.distplot(df[i], bins = 20, kde = True)
    plt.xticks(rotation = 90)
    plt.show()
```

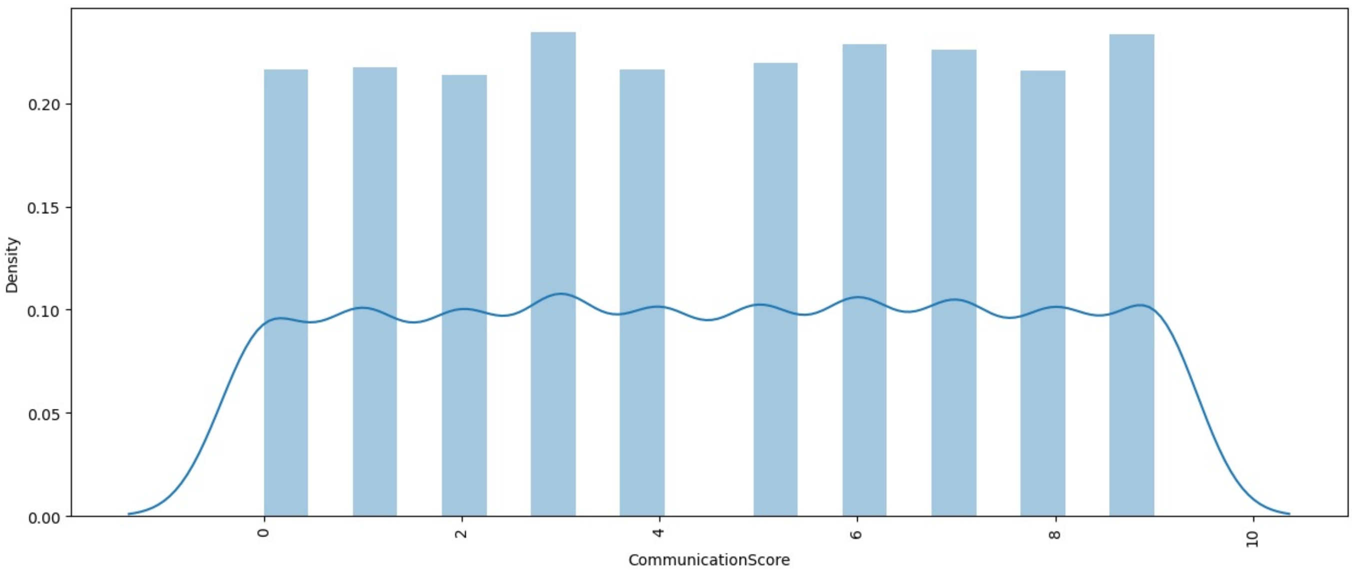

Density
Height
0.00
0.01
0.02
0.03
0.04
0.05
145
150
155
160
165
170
175
180
185

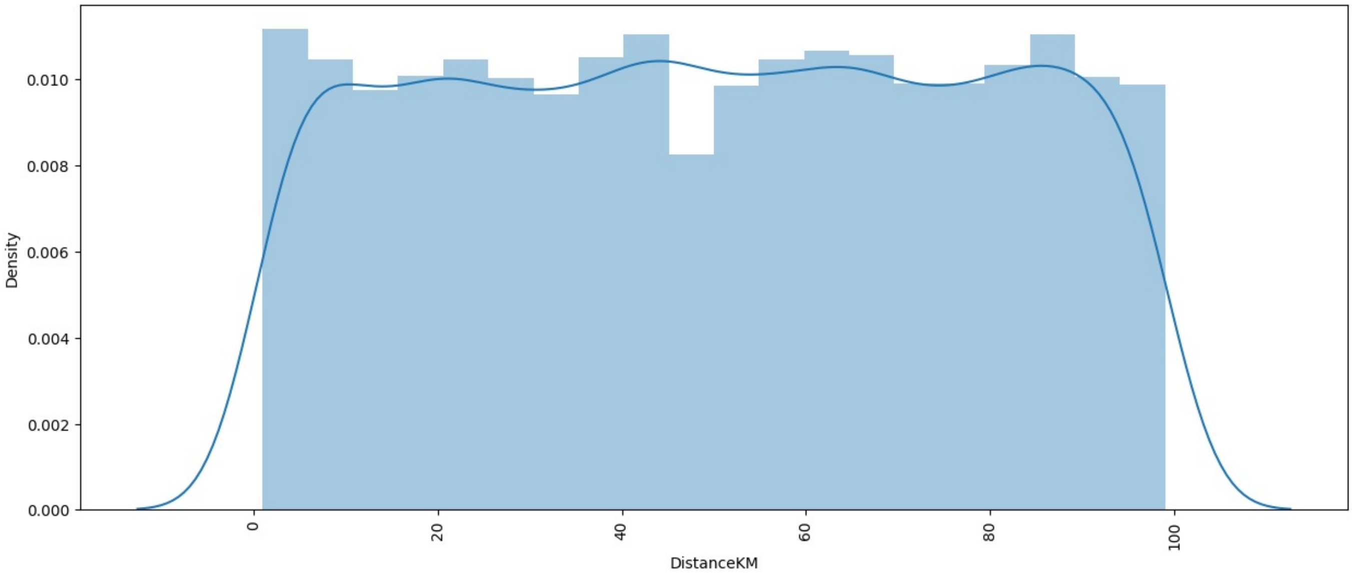

Density
Age
0.000
0.005
0.010
0.015
0.020
10
20
30
40
50
60
70
80
90


1e−5
Density
Income
0
1
2
3
4
5
6
7
2500
5000
7500
10000
12500
15000
17500
20000
22500

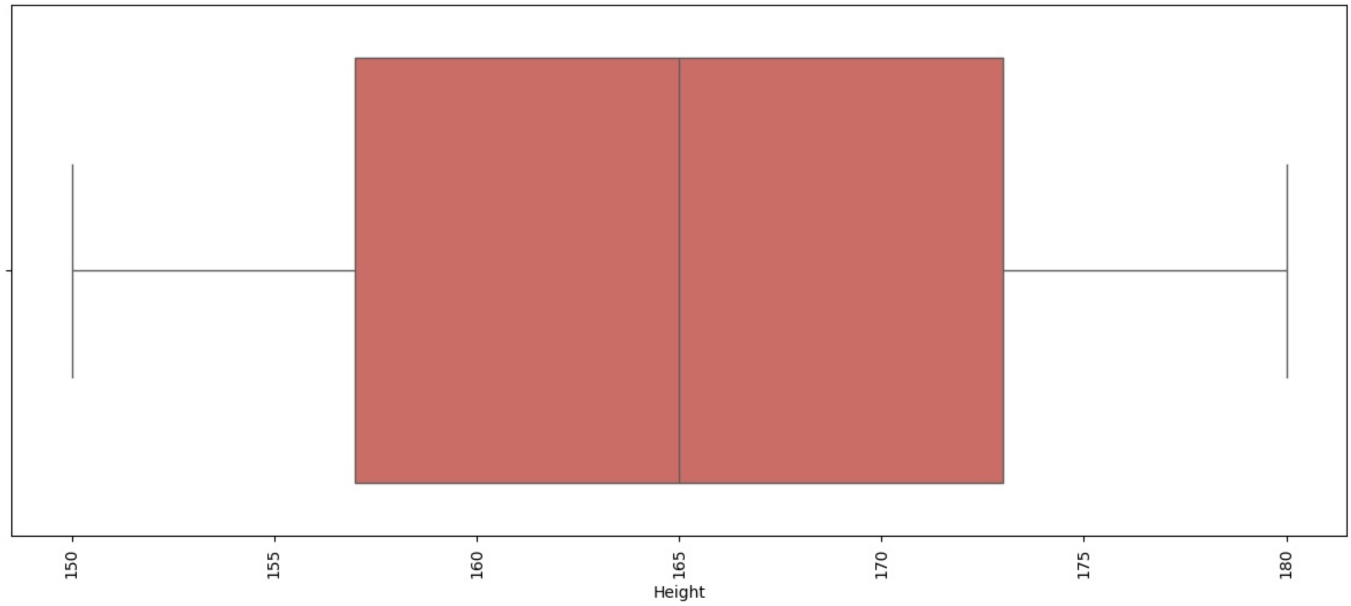

0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Density
−2
0
2
4
6
8
10
12
RomanticGestureScore

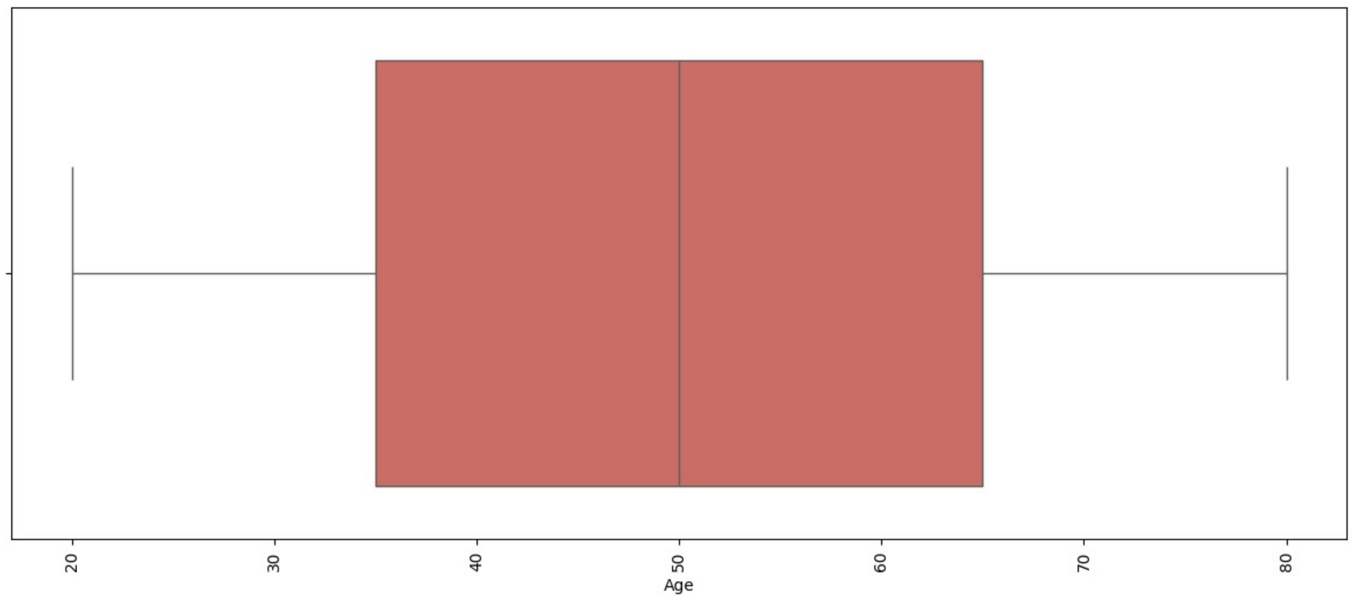

0.20
0.15
0.10
0.05
0.00
Density
0
2
4
6
8
10
CompatibilityScore

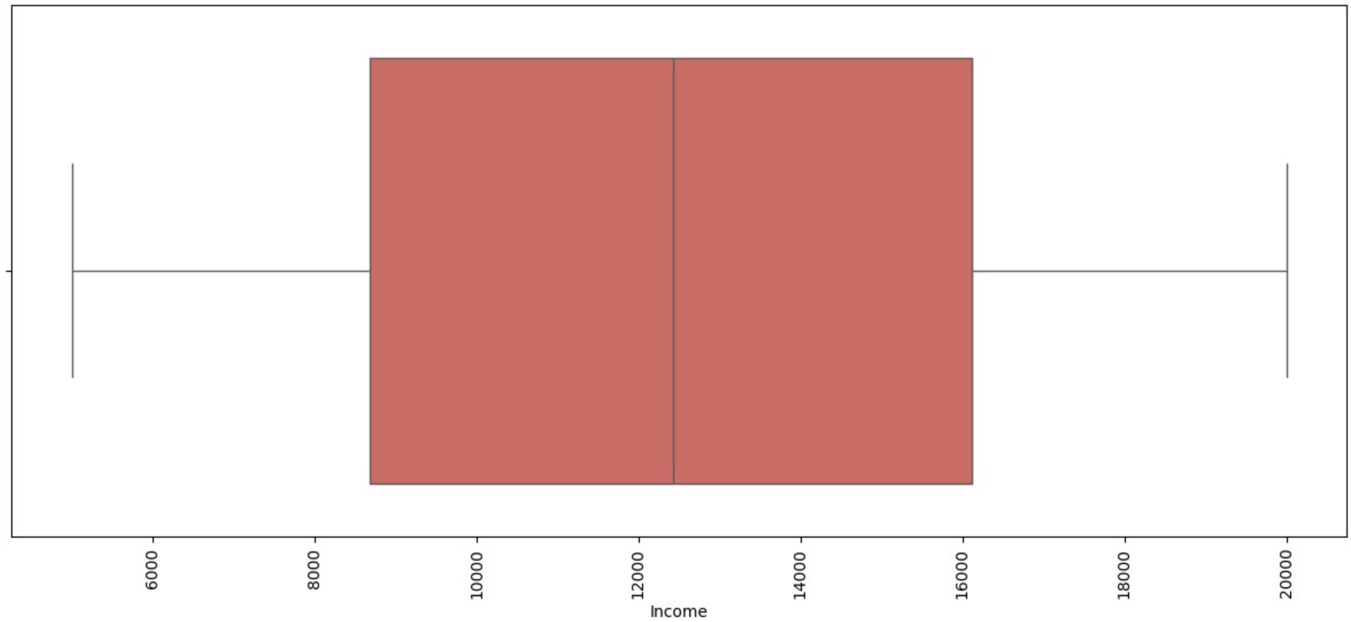

0.20
0.15
0.10
0.05
0.00
Density
0
2
4
6
8
10
CommunicationScore

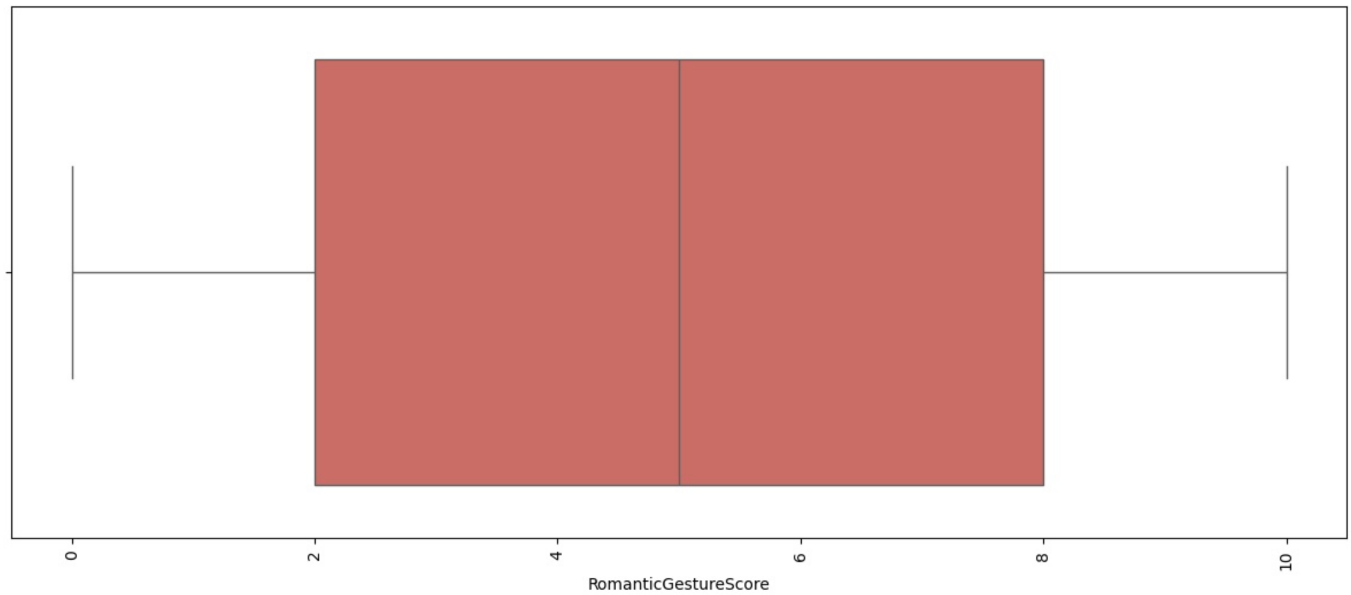


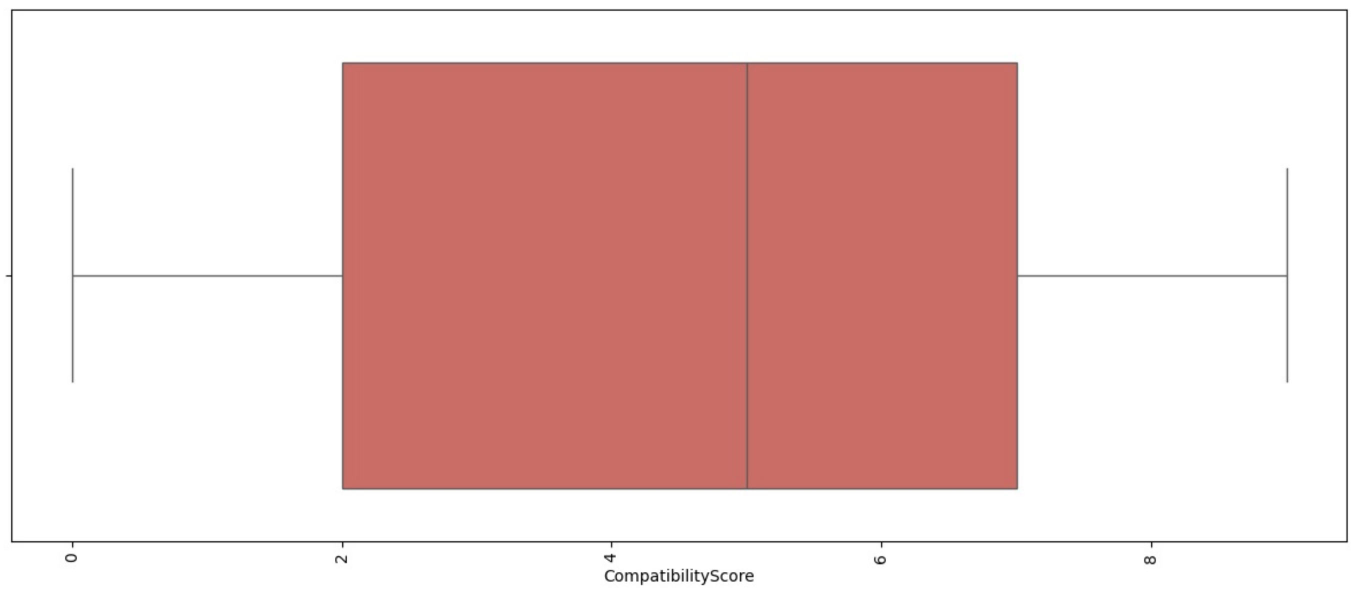
In [36]:
```
for i in continuous:
    plt.figure(figsize=(15, 6))
    sns.boxplot(x=i, data=df, palette='hls')
    plt.xticks(rotation=90)
    plt.show()
```

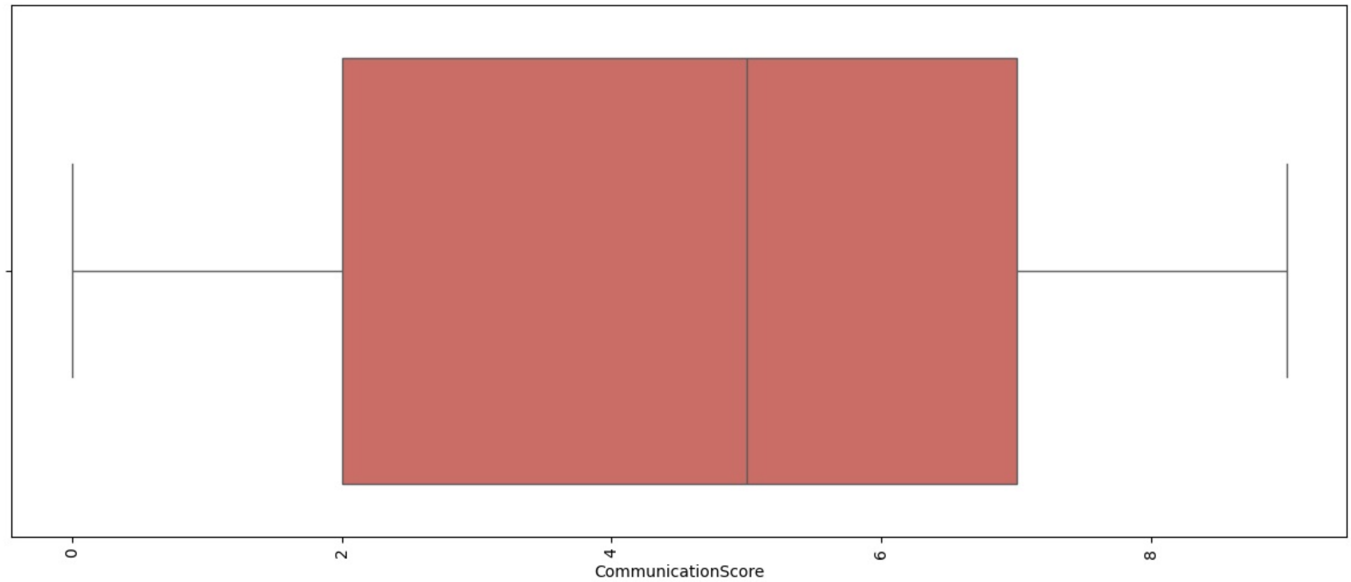


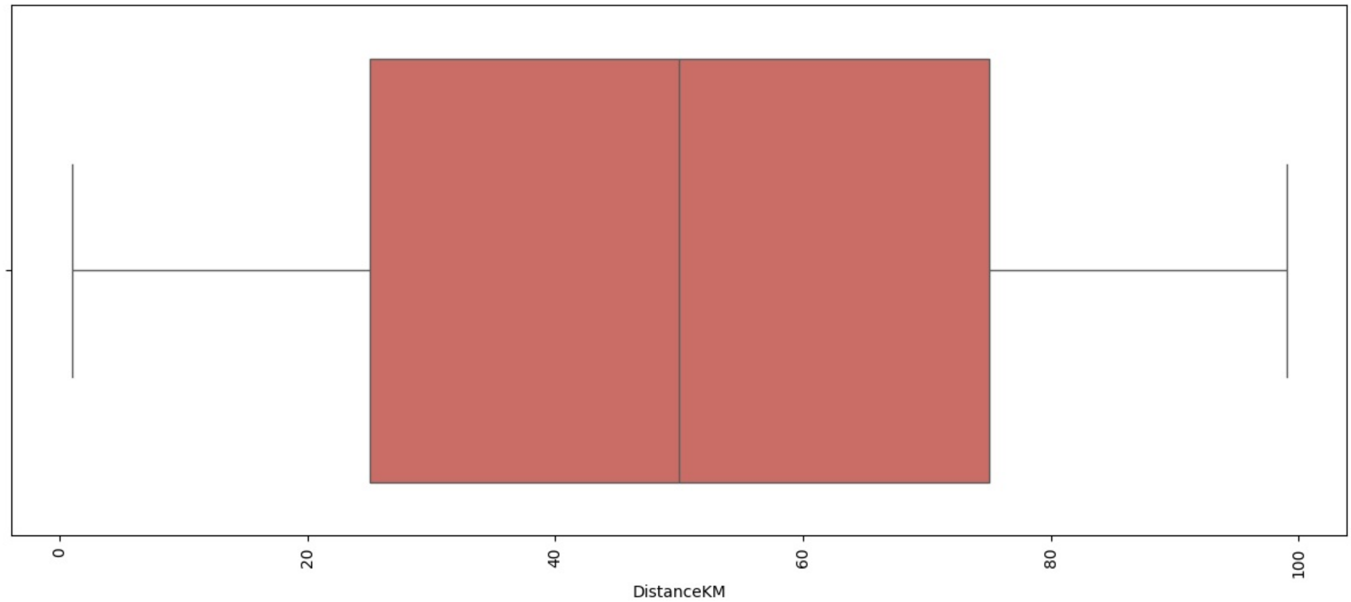



6000
8000
10000
12000
14000
16000
18000
20000
Income

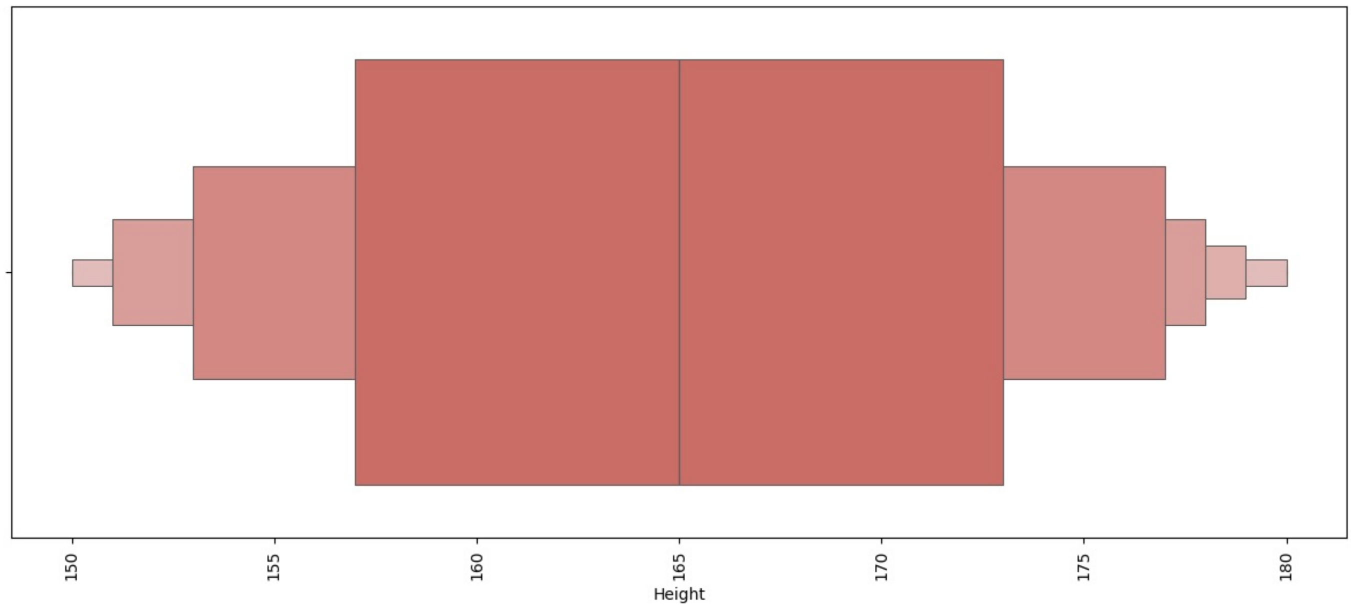

0
2
4
6
8
10
RomanticGestureScore

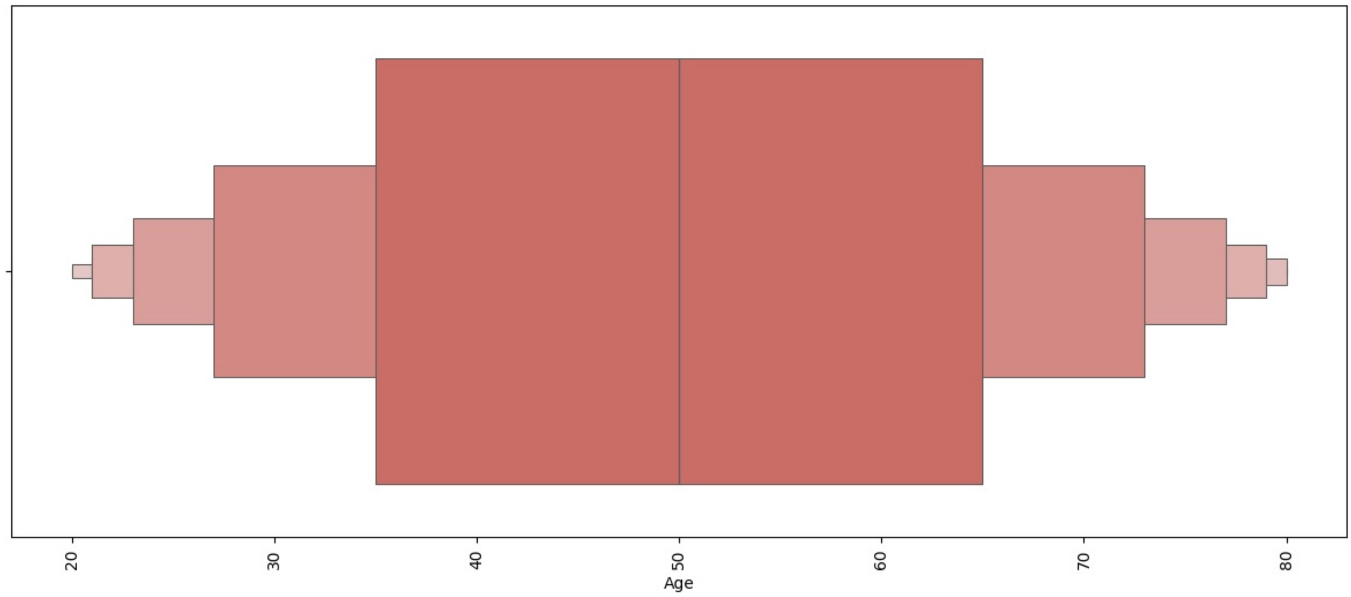

0
2
4
6
8
CompatibilityScore



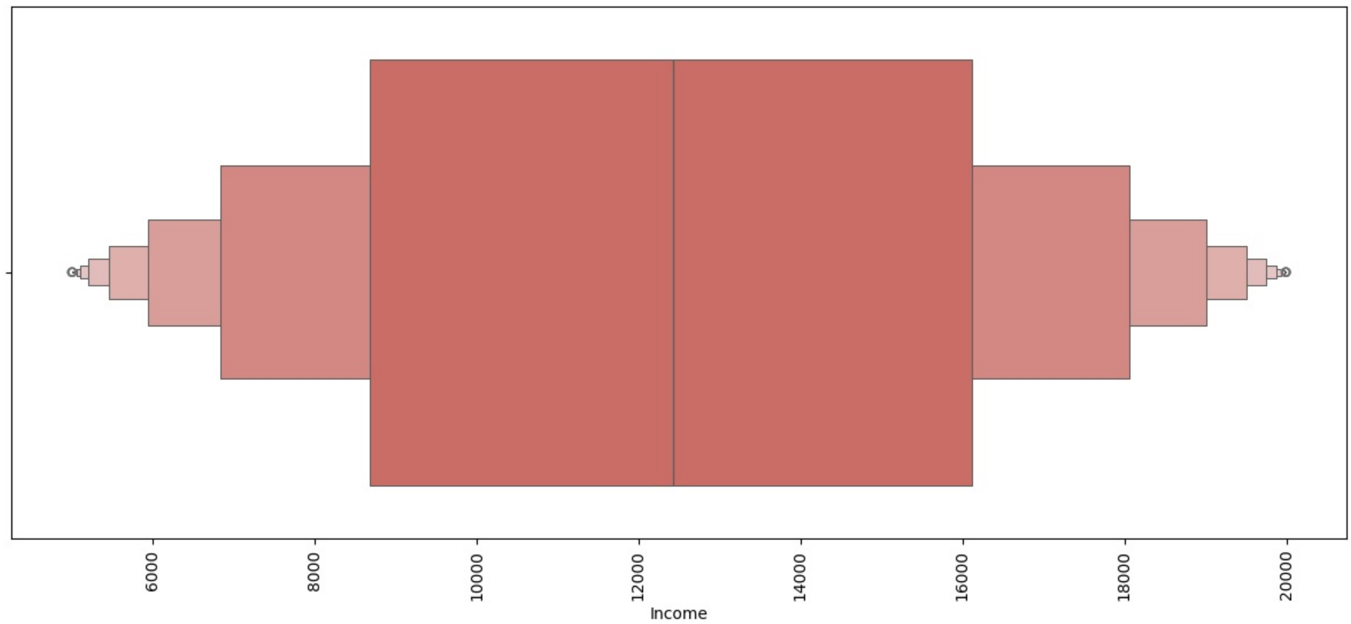


In [37]:
```
for i in continuous:
    plt.figure(figsize=(15, 6))
    sns.boxenplot(x=i, data=df, palette='hls')
    plt.xticks(rotation=90)
    plt.show()
```

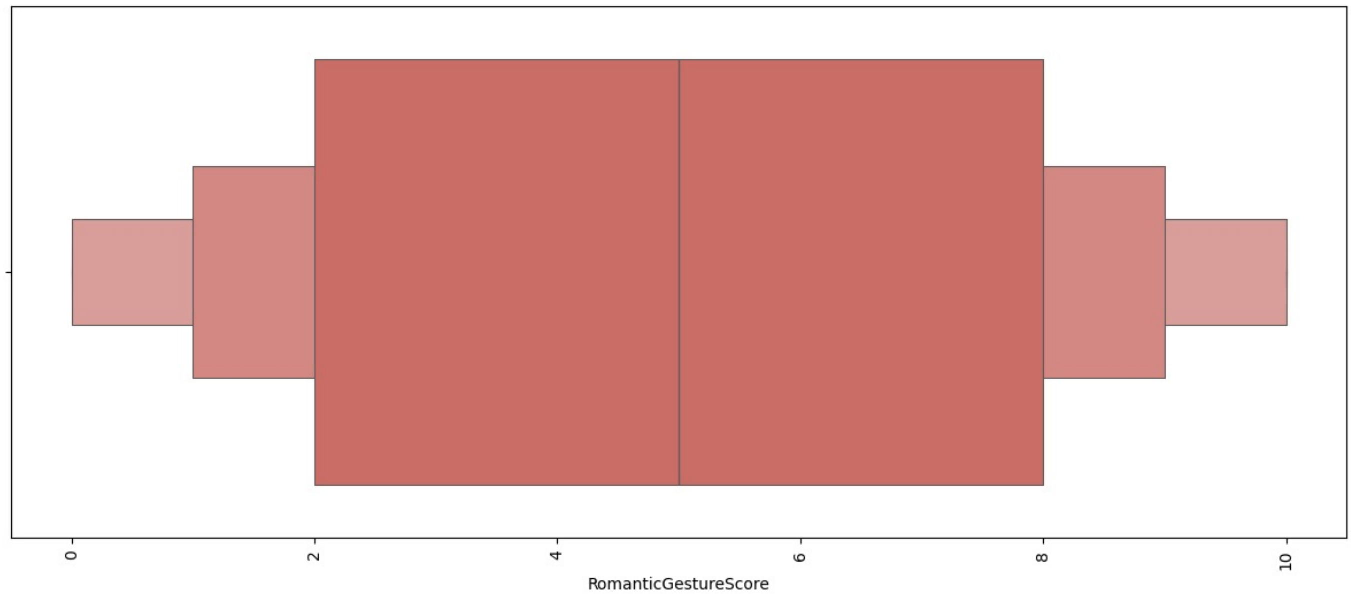




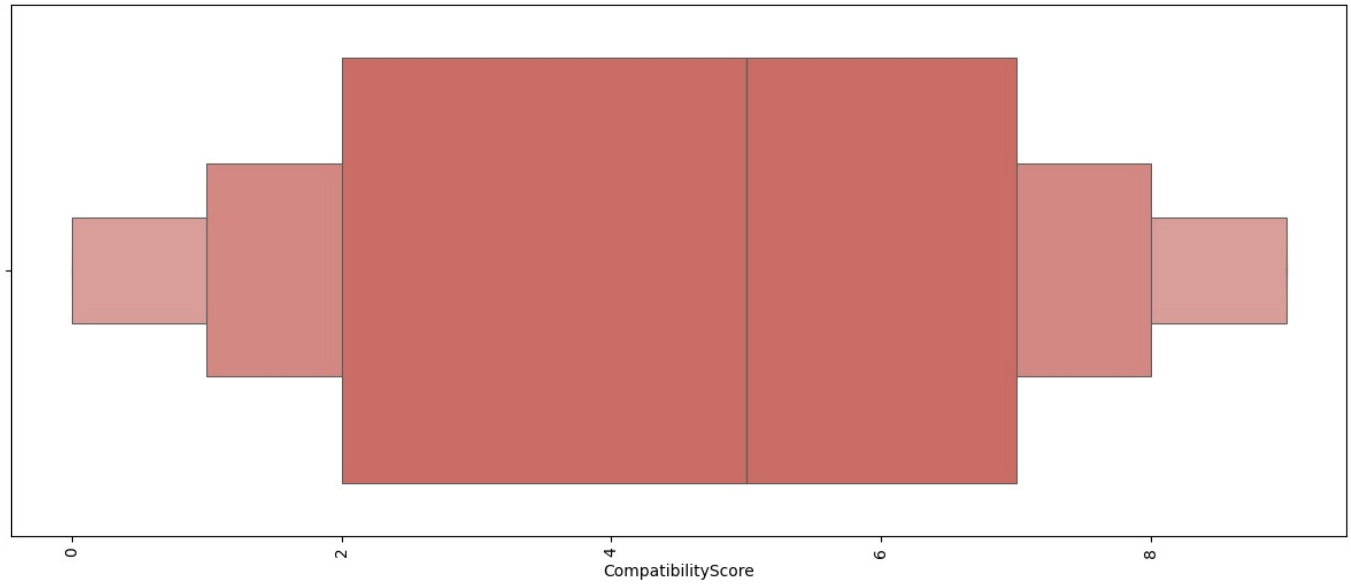
20
30
40
50
60
70
80
Age

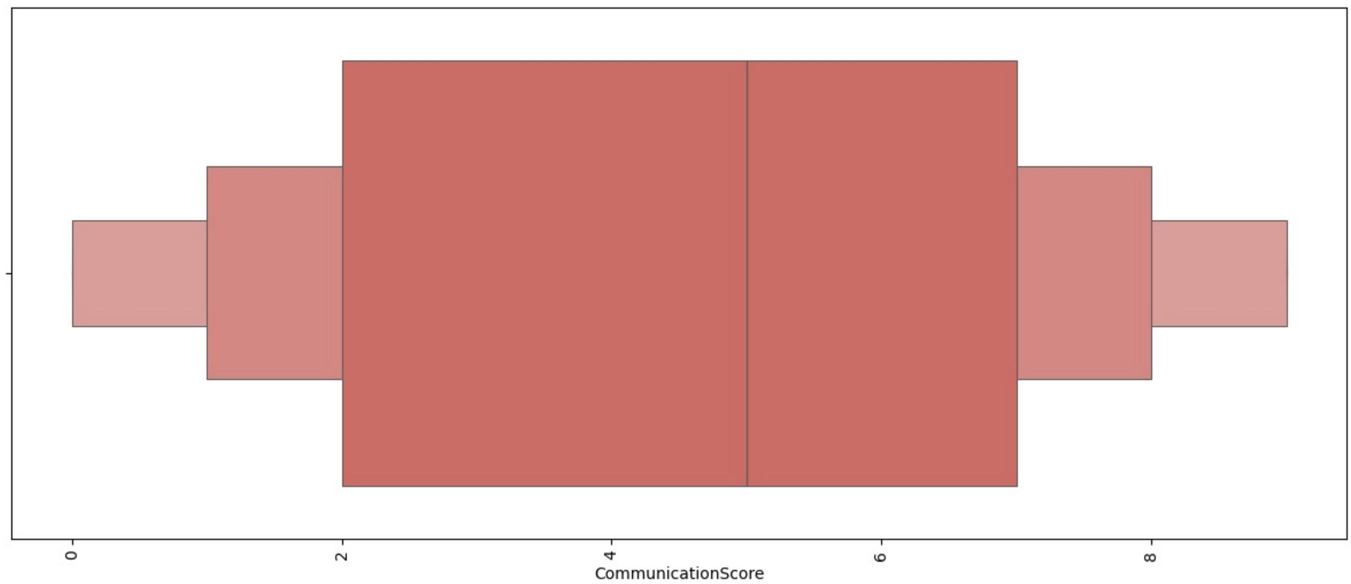


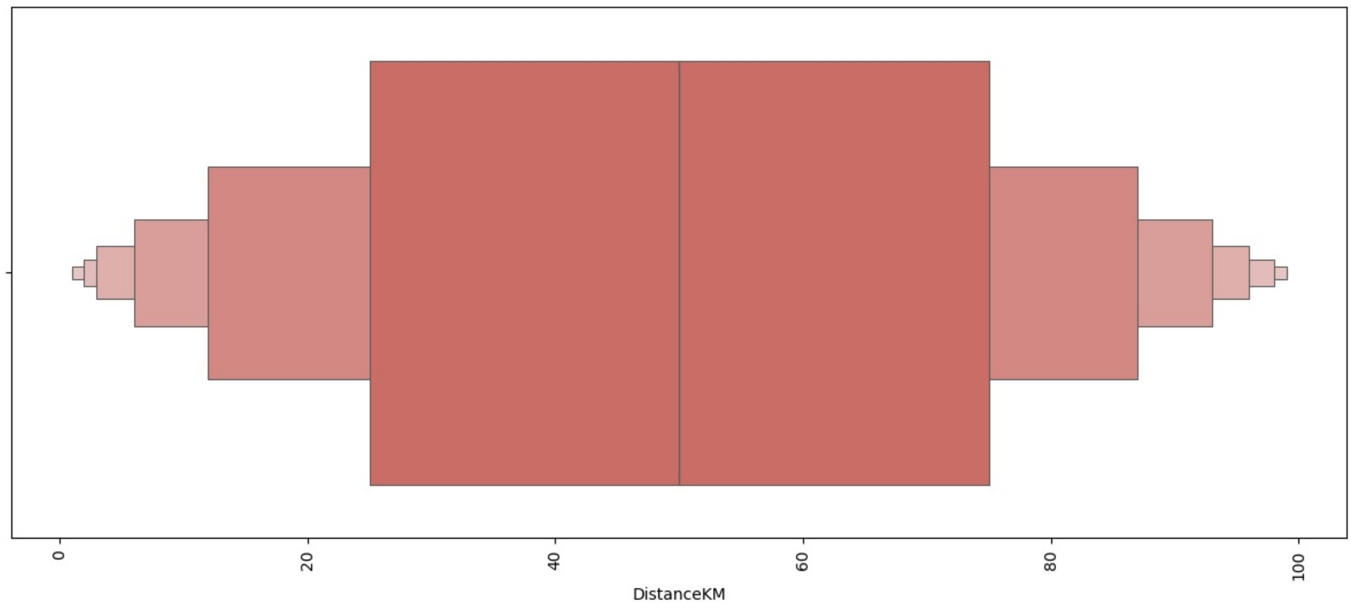
6000
8000
10000
12000
14000
16000
18000
20000
Income



0
2
4
6
8
10
RomanticGestureScore

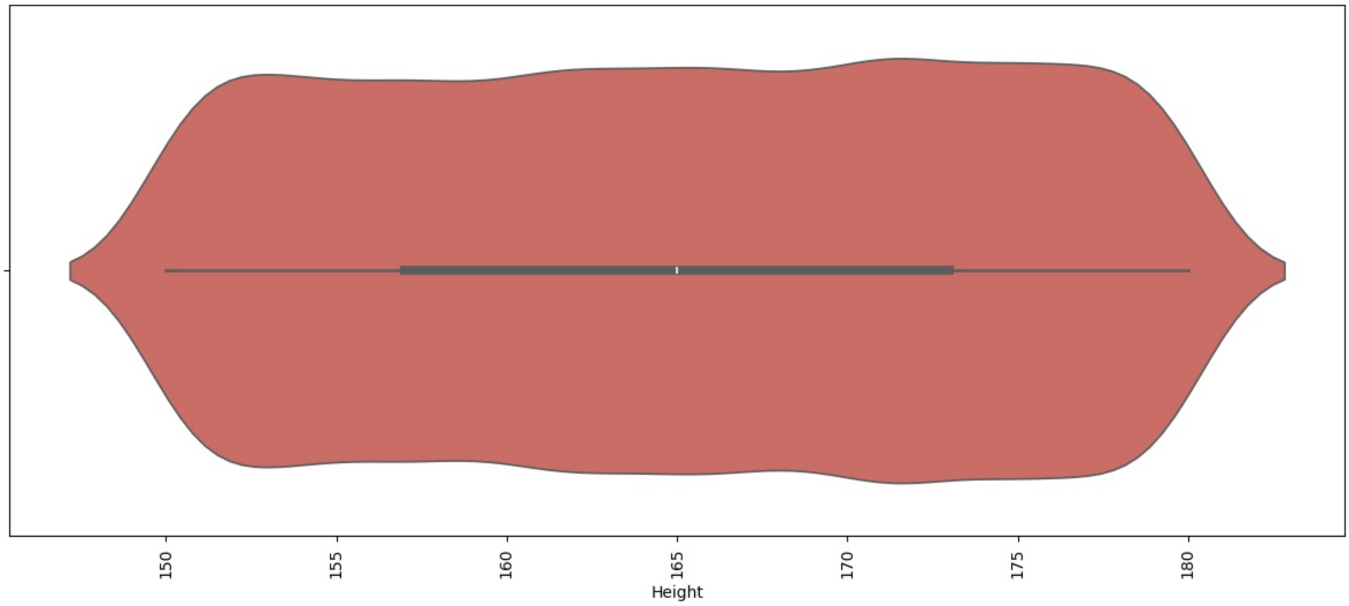


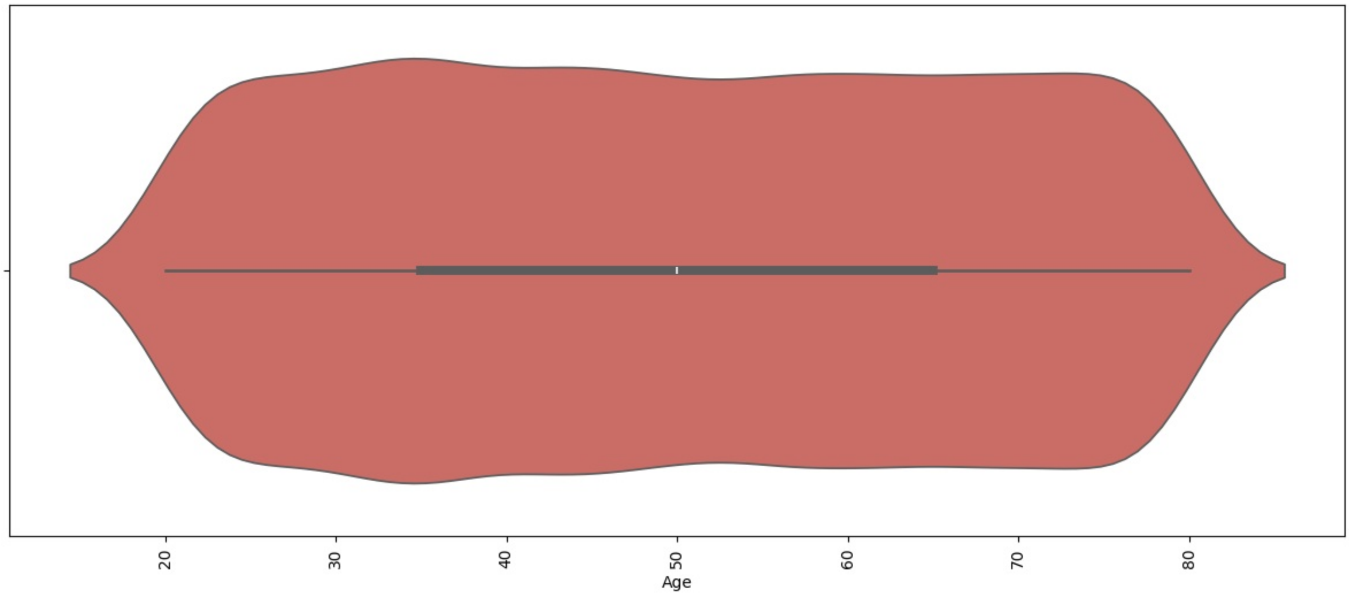


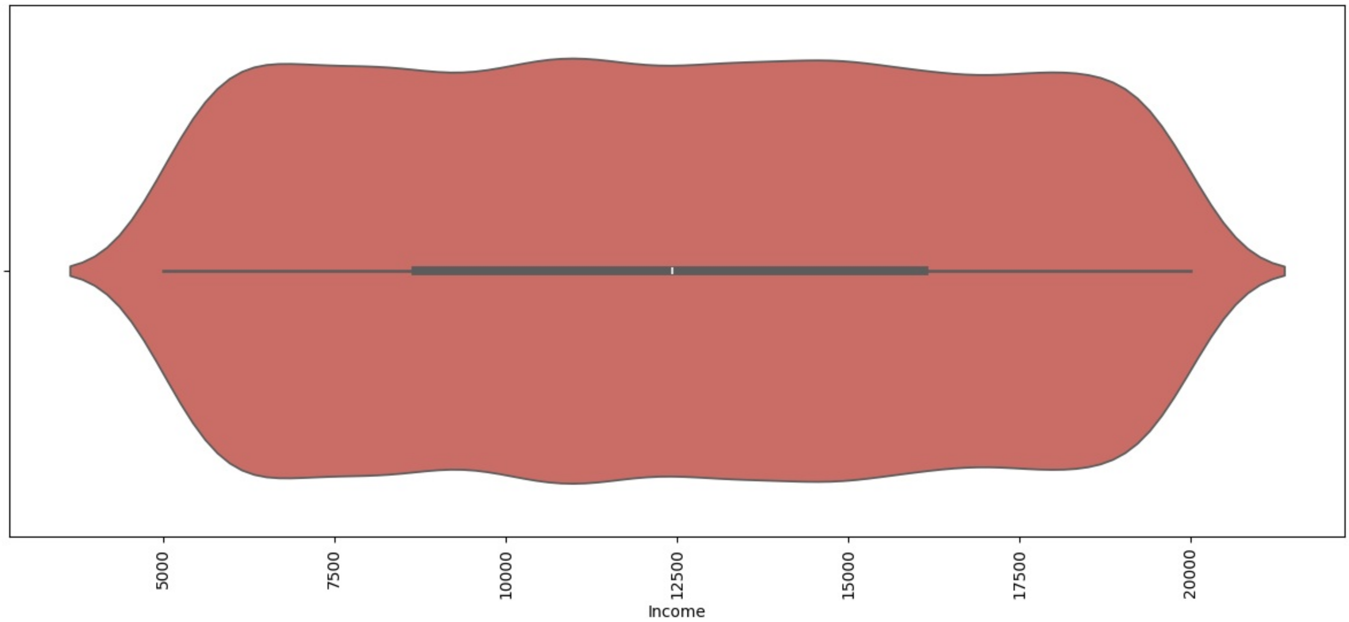


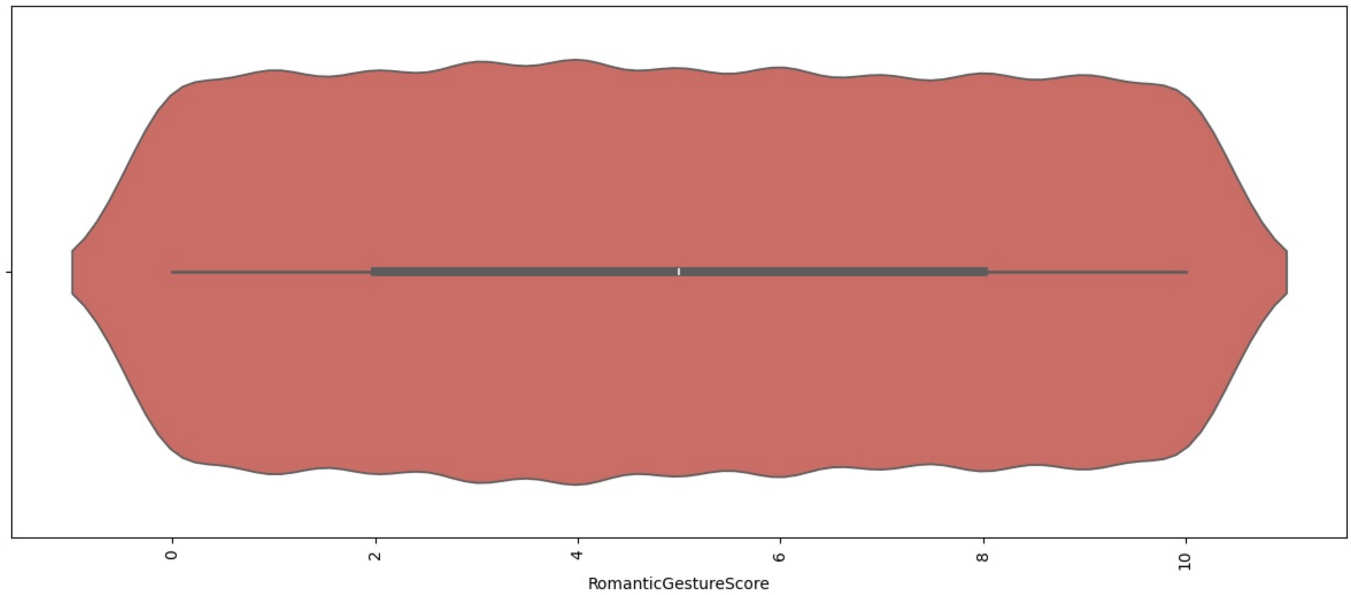
In [38]:
```
for i in continuous:
```

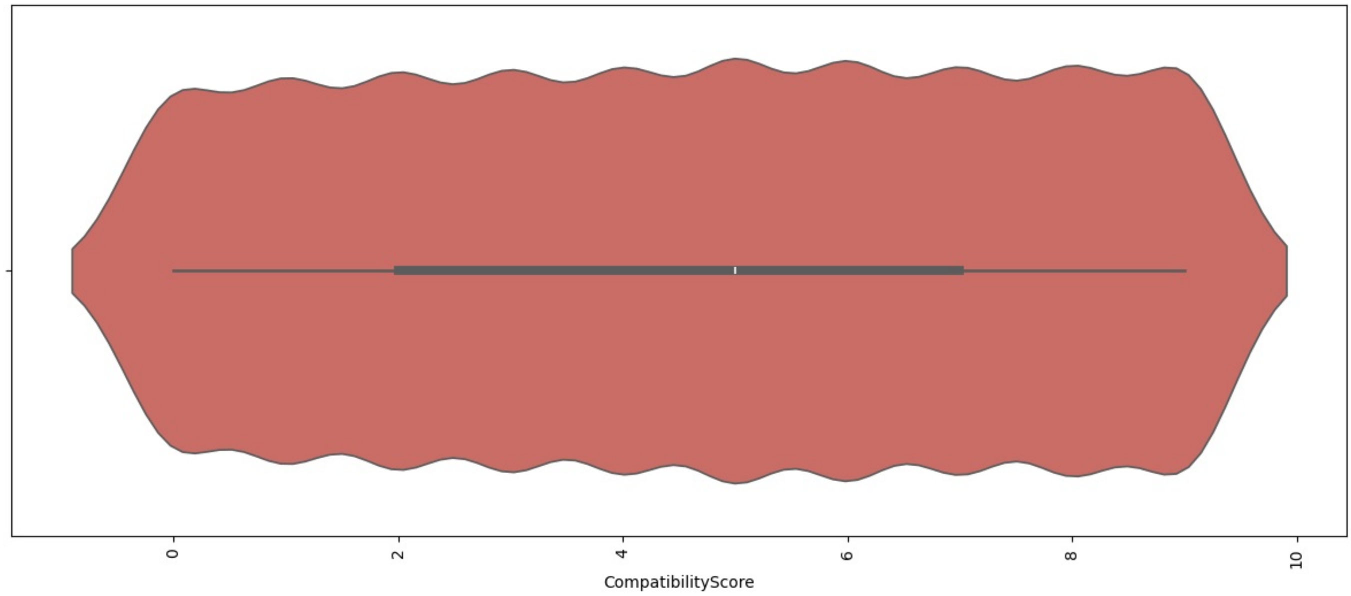
```
    plt.figure(figsize=(15, 6))
    sns.violinplot(x=i, data=df, palette='hls')
    plt.xticks(rotation=90)
    plt.show()
```

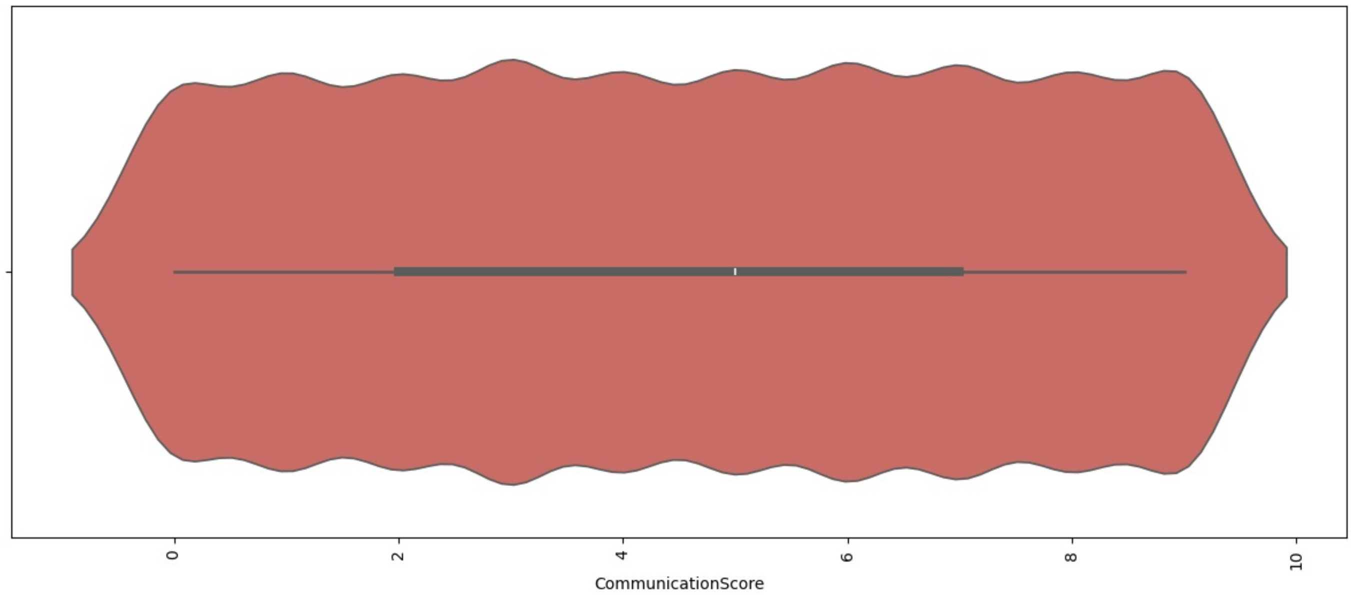


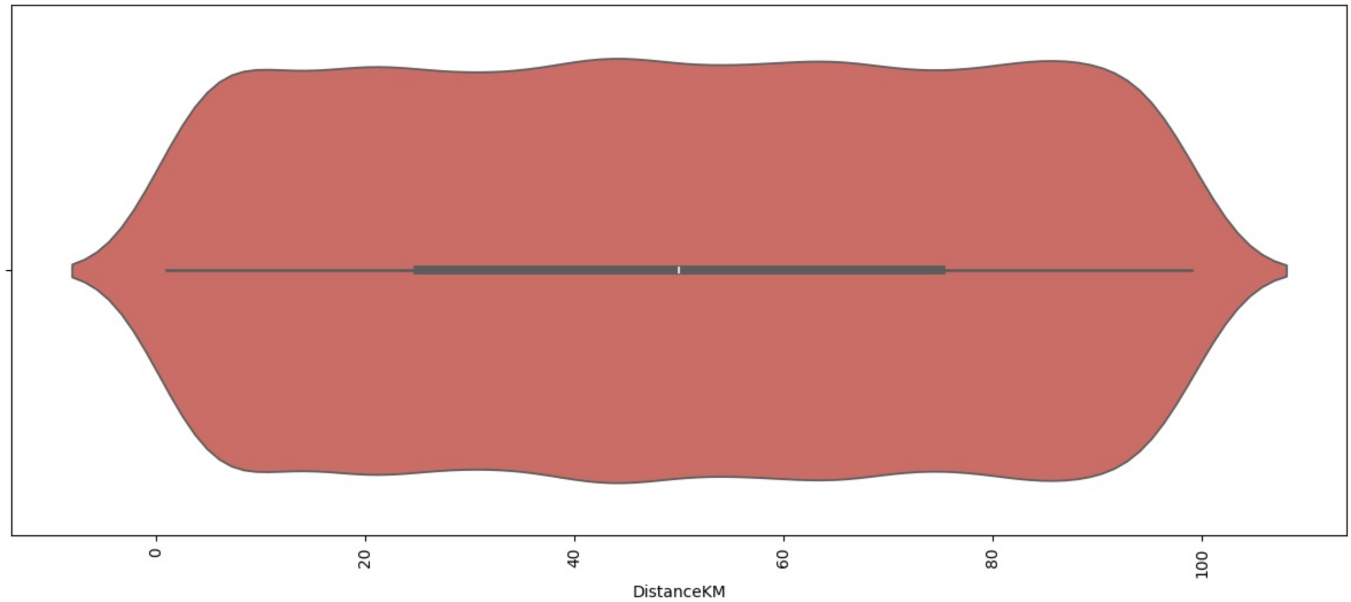




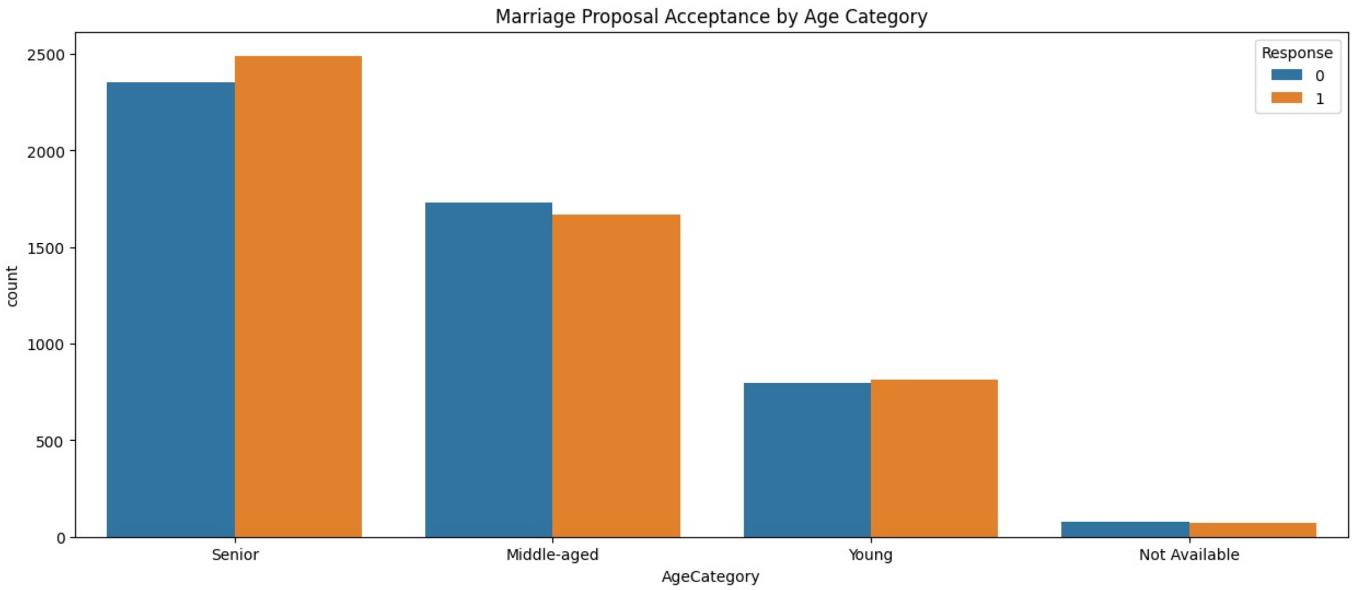

0
2
4
6
8
10
RomanticGestureScore


0
2
4
6
8
10
CompatibilityScore

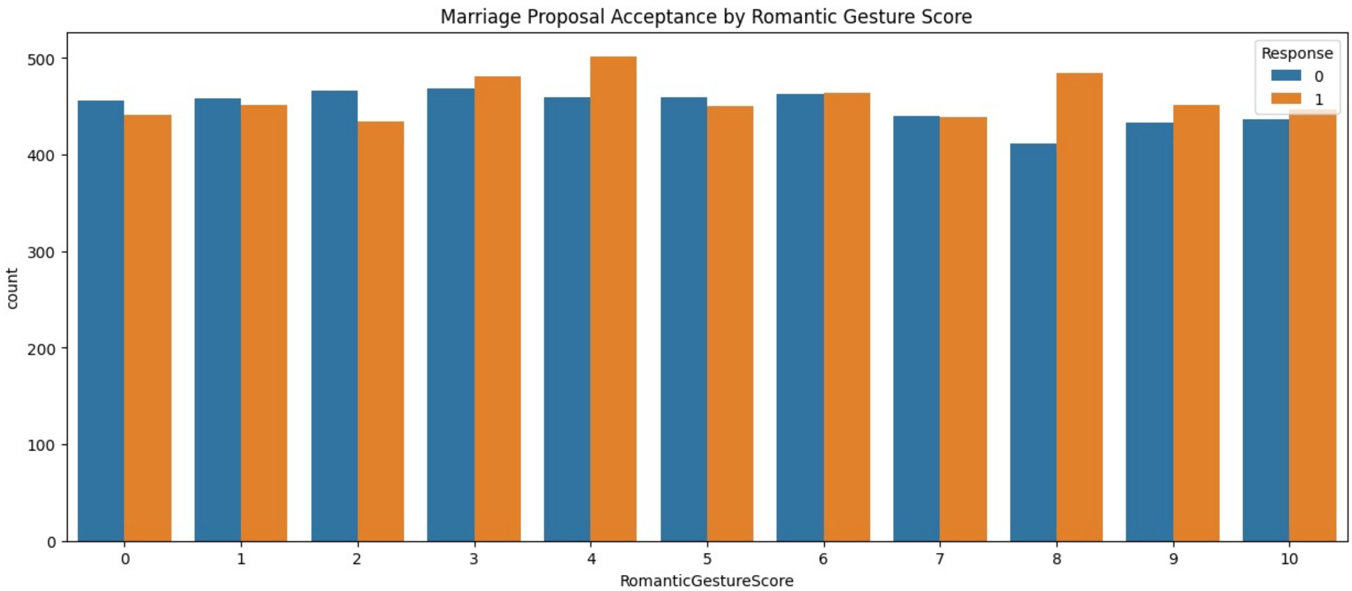

0
2
4
6
8
10
CommunicationScore



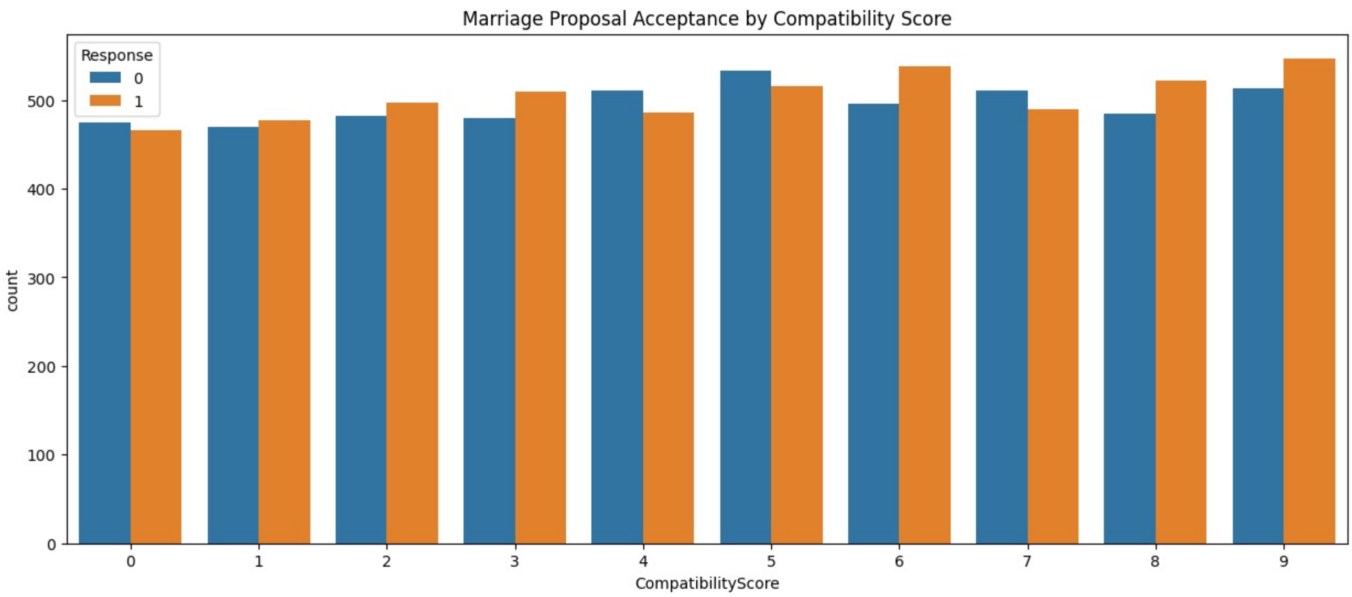
In [39]:
```
plt.figure(figsize=(15, 6))
sns.countplot(x='AgeCategory', hue='Response', data=df)
plt.title('Marriage Proposal Acceptance by Age Category')
plt.show()
```



In [40]:
```
plt.figure(figsize=(15, 6))
sns.countplot(x = 'RomanticGestureScore', hue='Response', data=df)
plt.title('Marriage Proposal Acceptance by Romantic Gesture Score')
plt.show()
```




```
In [41]: plt.figure(figsize=(15, 6))
sns.countplot(x = 'CompatibilityScore', hue='Response', data=df)
plt.title('Marriage Proposal Acceptance by Compatibility Score')
plt.show()
```




```
In [42]: pivot_table = pd.pivot_table(df, values='Income', index='AgeCategory', columns='Response', aggfunc='mean')
```

```
In [43]: pivot_table
```

Out[43]:

Response	0	1
AgeCategory		
Middle-aged	12470.907407	12440.625374
Not Available	12060.632911	11876.189189
Senior	12397.293367	12465.092734
Young	12375.469773	12588.330456

```
In [44]: pivot_table = pd.pivot_table(df, values='Income', index='AgeCategory', columns='Response', aggfunc='max')
```

```
In [45]: pivot_table
```

Out[45]:

Response	0	1
AgeCategory		
Middle-aged	19992	19999
Not Available	19656	19906
Senior	19985	19991
Young	19990	19963

In [46]:
```
pivot_table = pd.pivot_table(df, values='Income', index='AgeCategory', columns='Response', aggfunc='min')
```

In [47]:
```
pivot_table
```

Out[47]:

Response	0	1
AgeCategory		
Middle-aged	5012	5018
Not Available	5068	5098
Senior	5000	5008
Young	5019	5004

In [48]:
```
cross_tab = pd.crosstab(df['AgeCategory'], df['Response'])
```

In [49]:
```
cross_tab
```

Out[49]:

Response	0	1
AgeCategory		
Middle-aged	1728	1671
Not Available	79	74
Senior	2352	2491
Young	794	811

In [50]:
```
pivot_table = pd.pivot_table(df,
                             values=['RomanticGestureScore', 'CompatibilityScore'],
                             index='AgeCategory',
                             columns='Response',
                             aggfunc= 'mean',
                             margins=True,
                             fill_value=0)
```

In [51]:
```
pivot_table
```

Out[51]:

	CompatibilityScore			RomanticGestureScore		
Response	0	1	All	0	1	All
AgeCategory						
Middle-aged	4.552662	4.675643	4.613122	4.968171	5.081388	5.023831
Not Available	4.126582	4.891892	4.496732	4.670886	4.810811	4.738562
Senior	4.575255	4.606985	4.591575	4.861395	4.999599	4.932480
Young	4.614610	4.464858	4.538941	4.994962	4.926017	4.960125
All	4.566525	4.611056	4.589000	4.917020	5.012086	4.965000

In [52]:
```
cross_tab = pd.crosstab(df['AgeCategory'], [df['Response'], df['RomanticGestureScore'], df['CompatibilityScore'
                        normalize='index', margins=True)
```

In [53]:
```
cross_tab
```

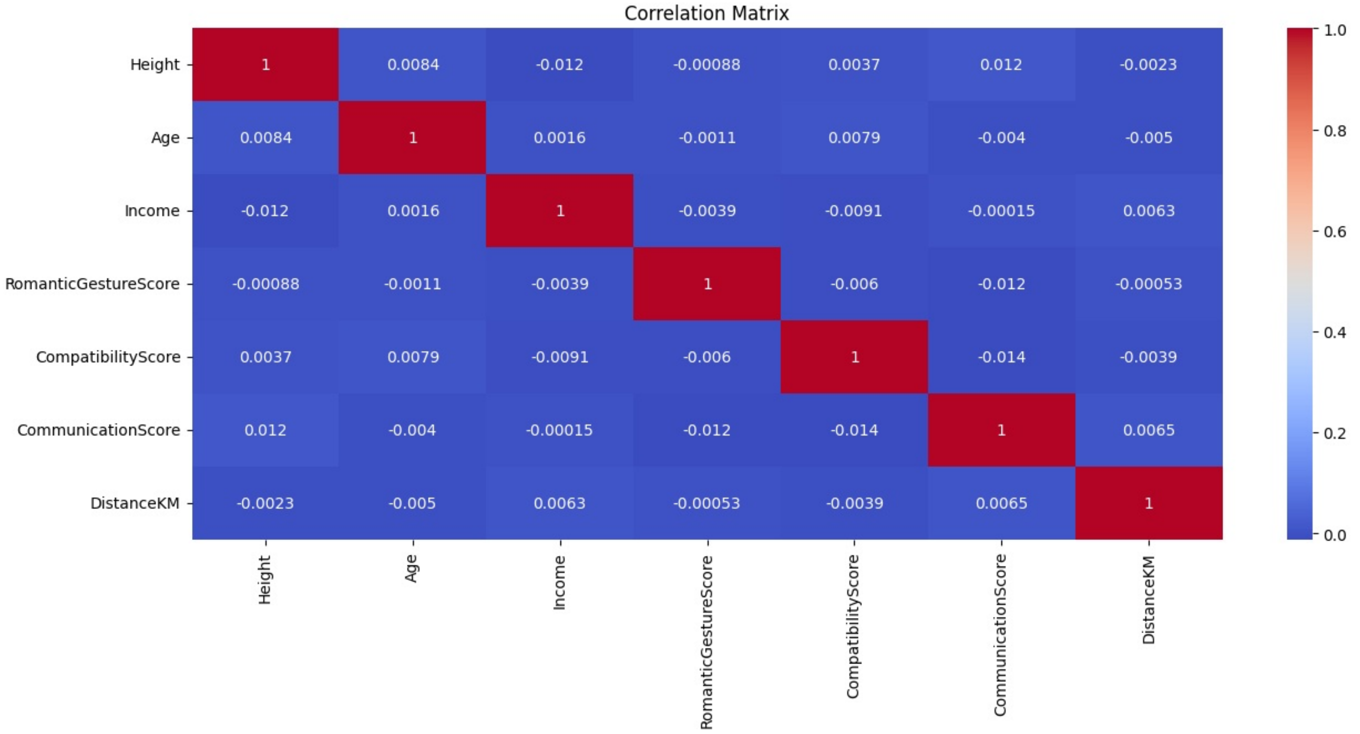
Out[53]:

Response										0	...	
RomanticGestureScore										0	...	
CompatibilityScore	0	1	2	3	4	5	6	7	8	9	...	
AgeCategory												
Middle-aged	0.002059	0.004413	0.005296	0.002942	0.004119	0.004413	0.005296	0.004413	0.003825	0.004707	...	0.0
Not Available	0.013072	0.006536	0.000000	0.000000	0.013072	0.006536	0.000000	0.006536	0.000000	0.000000	...	0.0
Senior	0.005988	0.003510	0.004336	0.007020	0.004749	0.004543	0.004336	0.005162	0.004956	0.005369	...	0.0
Young	0.003738	0.003115	0.002492	0.003115	0.006231	0.003738	0.003738	0.005607	0.006231	0.003115	...	0.0
All	0.004400	0.003800	0.004300	0.004900	0.004900	0.004400	0.004500	0.005000	0.004700	0.004700	...	0.0

5 rows × 220 columns

In [54]:
```
plt.figure(figsize=(15, 6))
correlation_matrix = df[['Height', 'Age', 'Income', 'RomanticGestureScore', 'CompatibilityScore', 'Communicatio
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
```

```python
plt.title('Correlation Matrix')
plt.show()
```



In [55]:
```python
df['AgeCategory'] = df['AgeCategory'].astype('category')

df_new = pd.get_dummies(df, columns=['AgeCategory'], drop_first=True)
```

In [56]:
```python
df_new = df_new.astype(int)
```

In [57]:
```python
df_new
```

Out[57]:

	Height	Age	Income	RomanticGestureScore	CompatibilityScore	CommunicationScore	DistanceKM	Response	AgeCategory_Avai
0	156	59	7977	3	1	1	45	1	
1	169	32	5842	0	1	5	46	1	
2	178	42	17638	2	5	5	13	0	
3	164	78	8793	0	0	7	52	0	
4	160	35	15262	6	0	0	9	1	
...	...	...	...	...	...	...	...	...	
9995	162	76	12311	4	1	5	75	1	
9996	162	75	6459	7	9	0	52	1	
9997	166	70	9231	9	4	6	33	0	
9998	176	78	12656	8	9	5	25	1	
9999	156	68	5812	0	9	4	14	1	

10000 rows × 11 columns

In [58]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

In [59]:
```python
X = df_new.drop('Response', axis=1)
y = df_new['Response']
```

In [60]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify = y, random_state=42)
```

In [61]:
```python
log_reg_model = LogisticRegression()
log_reg_model.fit(X_train, y_train)
log_reg_pred = log_reg_model.predict(X_test)
log_reg_accuracy = accuracy_score(y_test, log_reg_pred)
print("Logistic Regression Accuracy:", log_reg_accuracy)
```

```
Logistic Regression Accuracy: 0.488
```

In [62]:
```
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_pred)
print("Random Forest Accuracy:", rf_accuracy)
```

```
Random Forest Accuracy: 0.5245
```

In [63]:
```
svm_model = SVC()
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_pred)
print("SVM Accuracy:", svm_accuracy)
```

```
SVM Accuracy: 0.4825
```

In [64]:
```
from sklearn.preprocessing import StandardScaler
```

In [65]:
```
non_discrete_features = ['Height', 'Age', 'Income', 'RomanticGestureScore', 'CompatibilityScore', 'Communicatio
```

In [66]:
```
scaler = StandardScaler()

df_new[non_discrete_features] = scaler.fit_transform(df_new[non_discrete_features])
```

In [67]:
```
X = df_new.drop('Response', axis=1)
y = df_new['Response']
```

In [68]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify = y, random_state=42)
```

In [69]:
```
svm_model = SVC()
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_pred)
print("SVM Accuracy:", svm_accuracy)
```

```
SVM Accuracy: 0.486
```

In [70]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [71]:
```
dt_classifier = DecisionTreeClassifier(random_state=42)
```

In [72]:
```
dt_classifier.fit(X_train, y_train)
```

Out[72]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [73]:
```
dt_pred = dt_classifier.predict(X_test)

dt_accuracy = accuracy_score(y_test, dt_pred)
print("Decision Tree Accuracy:", dt_accuracy)
```

```
Decision Tree Accuracy: 0.514
```

In [74]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In [75]:
```
degree = 2

selected_features = ['Height', 'Age', 'Income', 'RomanticGestureScore',
                     'CompatibilityScore', 'CommunicationScore', 'DistanceKM']
```

In [76]:
```
X_selected = df[selected_features]
```

In [77]:
```
poly = PolynomialFeatures(degree=degree, include_bias=False)
```

In [78]:
```
poly_features = poly.fit_transform(X_selected)
```

In [79]:
```
poly_feature_names = poly.get_feature_names_out(input_features=selected_features)

df_poly = pd.DataFrame(poly_features, columns=poly_feature_names)

df_poly = pd.concat([df[['Response']], df_poly], axis=1)
```

In [80]:
```
df_poly
```

Out[80]:

	Response	Height	Age	Income	RomanticGestureScore	CompatibilityScore	CommunicationScore	DistanceKM	Height^2
0	1	156.0	59.0	7977.0	3.0	1.0	1.0	45.0	24336.0
1	1	169.0	32.0	5842.0	0.0	1.0	5.0	46.0	28561.0
2	0	178.0	42.0	17638.0	2.0	5.0	5.0	13.0	31684.0
3	0	164.0	78.0	8793.0	0.0	0.0	7.0	52.0	26896.0
4	1	160.0	35.0	15262.0	6.0	0.0	0.0	9.0	25600.0
...	...	...	...	...	...	...	...	...	...
9995	1	162.0	76.0	12311.0	4.0	1.0	5.0	75.0	26244.0
9996	1	162.0	75.0	6459.0	7.0	9.0	0.0	52.0	26244.0
9997	0	166.0	70.0	9231.0	9.0	4.0	6.0	33.0	27556.0
9998	1	176.0	78.0	12656.0	8.0	9.0	5.0	25.0	30976.0
9999	1	156.0	68.0	5812.0	0.0	9.0	4.0	14.0	24336.0

10000 rows × 36 columns

In [81]:
```
df_poly.columns
```

Out[81]:
```
Index(['Response', 'Height', 'Age', 'Income', 'RomanticGestureScore',
       'CompatibilityScore', 'CommunicationScore', 'DistanceKM', 'Height^2',
       'Height Age', 'Height Income', 'Height RomanticGestureScore',
       'Height CompatibilityScore', 'Height CommunicationScore',
       'Height DistanceKM', 'Age^2', 'Age Income', 'Age RomanticGestureScore',
       'Age CompatibilityScore', 'Age CommunicationScore', 'Age DistanceKM',
       'Income^2', 'Income RomanticGestureScore', 'Income CompatibilityScore',
       'Income CommunicationScore', 'Income DistanceKM',
       'RomanticGestureScore^2', 'RomanticGestureScore CompatibilityScore',
       'RomanticGestureScore CommunicationScore',
       'RomanticGestureScore DistanceKM', 'CompatibilityScore^2',
       'CompatibilityScore CommunicationScore',
       'CompatibilityScore DistanceKM', 'CommunicationScore^2',
       'CommunicationScore DistanceKM', 'DistanceKM^2'],
      dtype='object')
```

In [82]:
```
from sklearn.feature_selection import SelectFromModel
```

In [83]:
```
X = df_poly.drop(columns=['Response'])
y = df_poly['Response']
```

In [84]:
```
clf = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [85]:
```
selector = SelectFromModel(clf)
selector.fit(X, y)
```

Out[85]:

▸ **SelectFromModel**

▸ **estimator: RandomForestClassifier**

▸ RandomForestClassifier

In [86]:
```
selected_feature_indices = selector.get_support(indices=True)
```

In [87]:
```
selected_features = X.columns[selected_feature_indices]
```

In [88]:
```
X_selected = X[selected_features]
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)
```

In [89]:
```
X_selected = X[selected_features]
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)

model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[89]:

▾ LogisticRegression

LogisticRegression()

In [90]:
```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.4965
```