Projeto FINAL de LPAA - Machine Learning - LEANDRO DANTAS LIMA (059.323.894-00)

In [1]:
```
# importando de bibliotecas
import pandas as pd
from pandas import DataFrame
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import SMOTE
from IPython.display import display, HTML
from scipy import stats
from scipy.stats import f_oneway
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LinearRegression, LogisticRegression, LassoCV
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import cross_val_score, train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, PolynomialFeatures, StandardScaler
from sklearn.svm import SVC, SVR
import geopandas
import os
import plotly.express as px
import plotly.graph_objects as go
import plotly.offline as pyo
import scipy.stats as stats
import warnings
```

In [2]:
```
pip install --upgrade scikit-learn imbalanced-learn
```

```
Defaulting to user installation because normal site-packages is not writeableNote: you may need to restart the kernel to use updated packages.

Requirement already satisfied: scikit-learn in c:\users\ldl\appdata\roaming\python\python311\site-packages (1.4.1.post1)
Requirement already satisfied: imbalanced-learn in c:\users\ldl\appdata\roaming\python\python311\site-packages (0.12.0)
Requirement already satisfied: numpy<2.0,>=1.19.5 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn) (1.24.3)
Requirement already satisfied: scipy>=1.6.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn) (1.11.1)
Requirement already satisfied: joblib>=1.2.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
```

In [3]:
```
# desativar mensagens de warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [4]:
```
# importando o dataset "Average Time Spent By A User On Social Media" para análise
df = pd.read_csv("dummy_data.csv", sep=",", on_bad_lines='skip', low_memory=False)
```

In [5]:
```
# criando uma cópia do dataset para manter o backup do original
df_copy = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índices
```

In [6]:
```
# mostrando as primeiras linhas para entender os dados
print("Primeiras linhas do dataframe:")
df_copy.head()
```

```
Primeiras linhas do dataframe:
```

Out [6]:

	age	gender	time_spent	platform	interests	location	demographics	profession	income	indebt	isHomeOwner	Owns_Car
0	56	male	3	Instagram	Sports	United Kingdom	Urban	Software Engineer	19774	True	False	False
1	46	female	2	Facebook	Travel	United Kingdom	Urban	Student	10564	True	True	True
2	32	male	8	Instagram	Sports	Australia	Sub_Urban	Marketer Manager	13258	False	False	False
3	60	non-binary	5	Instagram	Travel	United Kingdom	Urban	Student	12500	False	True	False
4	25	male	1	Instagram	Lifestlye	Australia	Urban	Software Engineer	14566	False	True	True

In [7]:
```
# mostrando as propriedades do df
df_copy.shape
```

Out [7]: (1000, 12)

In [8]:
```
# mostrando os tipos de dados --> quando não consegue definir, classifica como object
df_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   age           1000 non-null   int64 
 1   gender        1000 non-null   object
 2   time_spent    1000 non-null   int64 
 3   platform      1000 non-null   object
 4   interests     1000 non-null   object
 5   location      1000 non-null   object
 6   demographics  1000 non-null   object
 7   profession    1000 non-null   object
 8   income        1000 non-null   int64 
 9   indebt        1000 non-null   bool  
 10  isHomeOwner   1000 non-null   bool  
 11  Owns_Car      1000 non-null   bool  
dtypes: bool(3), int64(3), object(6)
memory usage: 73.4+ KB
```

In [9]:
```
# Estatísticas descritivas
print("\nEstatísticas descritivas:")
df_copy.describe()
```

```
Estatísticas descritivas:
```

Out [9]:

	age	time_spent	income
count	1000.000000	1000.000000	1000.000000
mean	40.986000	5.029000	15014.823000
std	13.497852	2.537834	2958.628221
min	18.000000	1.000000	10012.000000
25%	29.000000	3.000000	12402.250000
50%	42.000000	5.000000	14904.500000
75%	52.000000	7.000000	17674.250000
max	64.000000	9.000000	19980.000000

In [10]:
```
# conferindo e contando se há valores ausentes no df
print("\nValores nulos no dataframe:")
print(df_copy.isnull().sum())
```

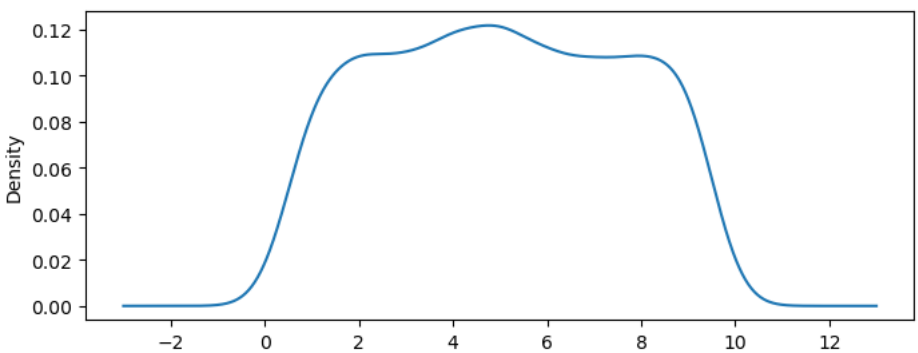
```
Valores nulos no dataframe:
age             0
gender          0
time_spent      0
platform        0
interests       0
location        0
demographics    0
profession      0
income          0
indebt          0
isHomeOwner     0
Owns_Car        0
dtype: int64
```

In [11]:
```
# conferindo se há dados duplicados
df_copy[df_copy.duplicated()].count().sum()
```

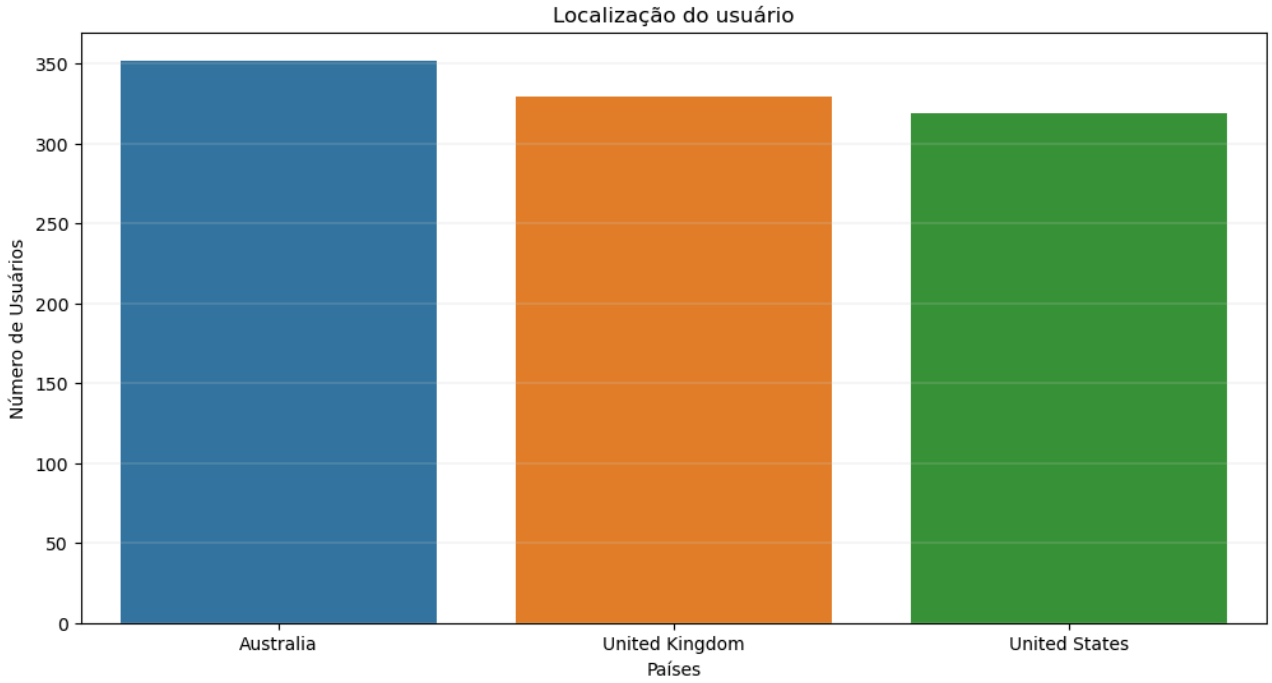
Out [11]: 0

In [12]:
```
# gráfico KDE (Kernel Density Function)
df_copy['time_spent'].plot.kde(subplots = True, figsize = (8,3))
```

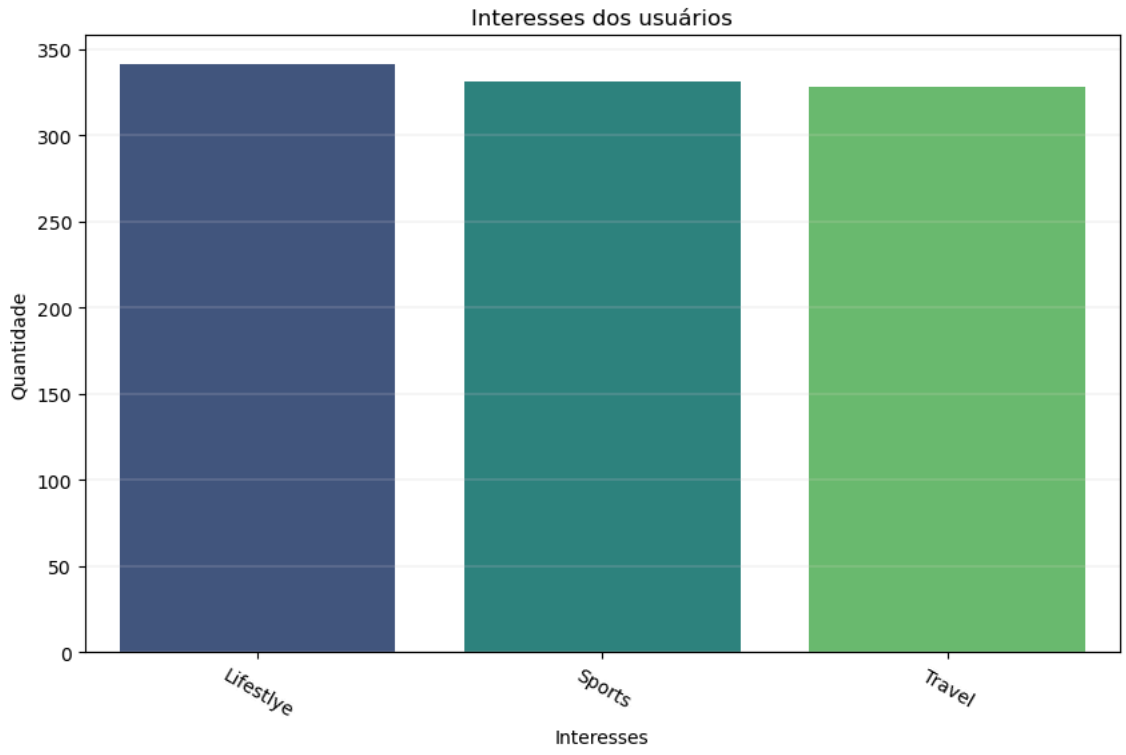
Out [12]: array([<Axes: ylabel='Density'>], dtype=object)



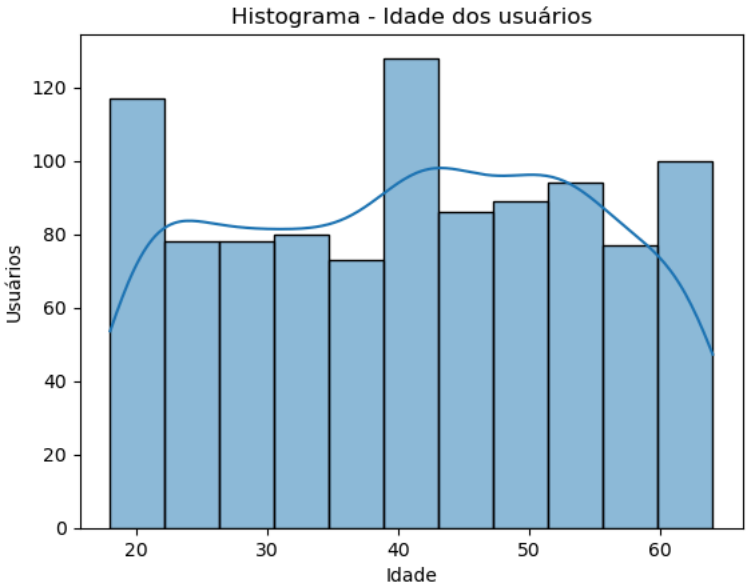
In [13]:
```
# Gráfico de Barras dos usuários por países
plt.figure(figsize=(12, 6))
plt.grid(color='lightgrey', linestyle='-', linewidth=0.25)
country = df_copy['location'].value_counts()
sns.barplot(x=country.index, y=country.values)
plt.xlabel('Países')
plt.ylabel('Número de Usuários')
plt.title('Localização do usuário')
plt.savefig("country.png") # salvando avistamentos por países
plt.show()
```



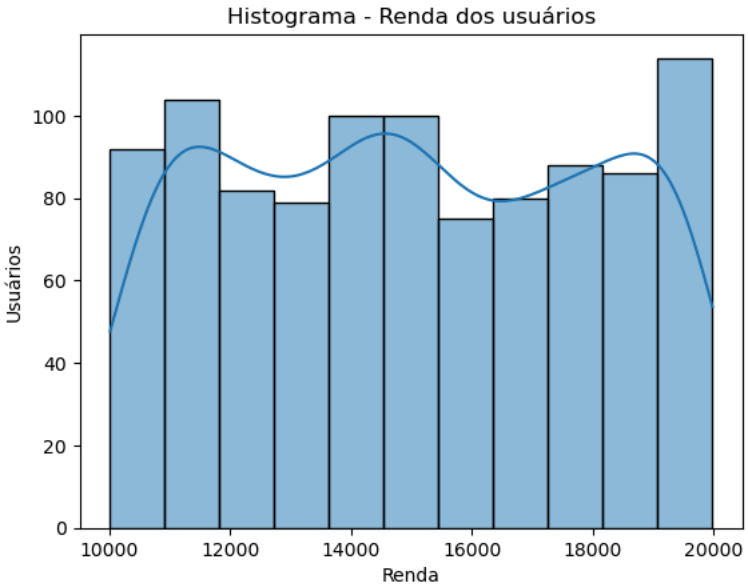
In [14]:
```
# gráfico de barras - interesses
plt.figure(figsize=(10,6))
plt.grid(color='lightgrey', linestyle='-', linewidth=0.25)
interests = df_copy['interests'].value_counts()
sns.barplot(x=interests.index, y=interests.values, palette='viridis')
plt.xlabel('Interesses')
plt.xticks(rotation = -30)
plt.ylabel('Quantidade')
plt.title('Interesses dos usuários')
plt.savefig("interests.png") # salvando os interesses
plt.show()
```



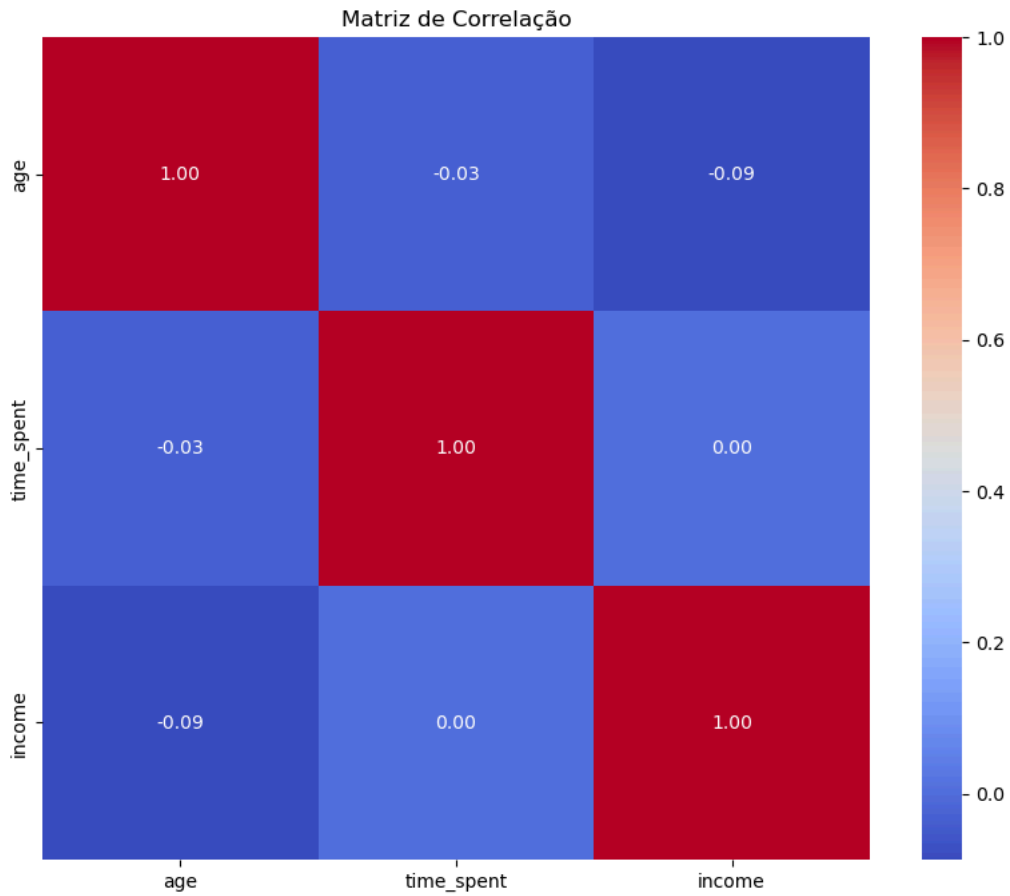
In [15]:
```
# histograma - representação gráfica da idade
sns.histplot(df_copy['age'],kde=True)
plt.title('Histograma - Idade dos usuários')
plt.xlabel('Idade')
plt.ylabel('Usuários')
plt.show()
```



In [16]:
```
# histograma - renda dos usuários
sns.histplot(df_copy['income'],kde=True)
plt.title('Histograma - Renda dos usuários')
plt.xlabel('Renda')
plt.ylabel('Usuários')
plt.show()
```



In [17]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índices

# Calculando a matriz de correlação
correlation_matrix = df_copy[['age', 'time_spent', 'income']].corr()

# Visualizando a matriz de correlação como um mapa de calor
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Matriz de Correlação')
plt.show()
```



In [18]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy2 = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índice
```

In [19]:
```
# calculando ano de nascimento
df_copy2["birth_year"] = 2024 - df_copy2["age"]

# Definindo gerações
bins = [1945, 1964, 1980, 1996, 2012]
labels = ["Baby Boomers", "Generation X", "Millennials", "Generation Z"]

# categorizando de acordo com a geração
df_copy2["generation"] = pd.cut(df_copy2["birth_year"], bins=bins, labels=labels, right=False)
```

In [20]:
```
# descrição do novo dataframe
df_copy2.describe()
```

Out [20]:

	age	time_spent	income	birth_year
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	40.986000	5.029000	15014.823000	1983.014000
std	13.497852	2.537834	2958.628221	13.497852
min	18.000000	1.000000	10012.000000	1960.000000
25%	29.000000	3.000000	12402.250000	1972.000000
50%	42.000000	5.000000	14904.500000	1982.000000
75%	52.000000	7.000000	17674.250000	1995.000000
max	64.000000	9.000000	19980.000000	2006.000000

In [21]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy3 = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índice
```

In [22]:
```
# selecionando variáveis categóricas
categorical_features = df_copy3.select_dtypes(include=[np.object_, "category"])
for column in categorical_features:
    print(f"Value counts for {column}:")
    print(df_copy3[column].value_counts())
    print("\n")
```

```
Value counts for gender:
gender
male          337
non-binary    332
female        331
Name: count, dtype: int64
```

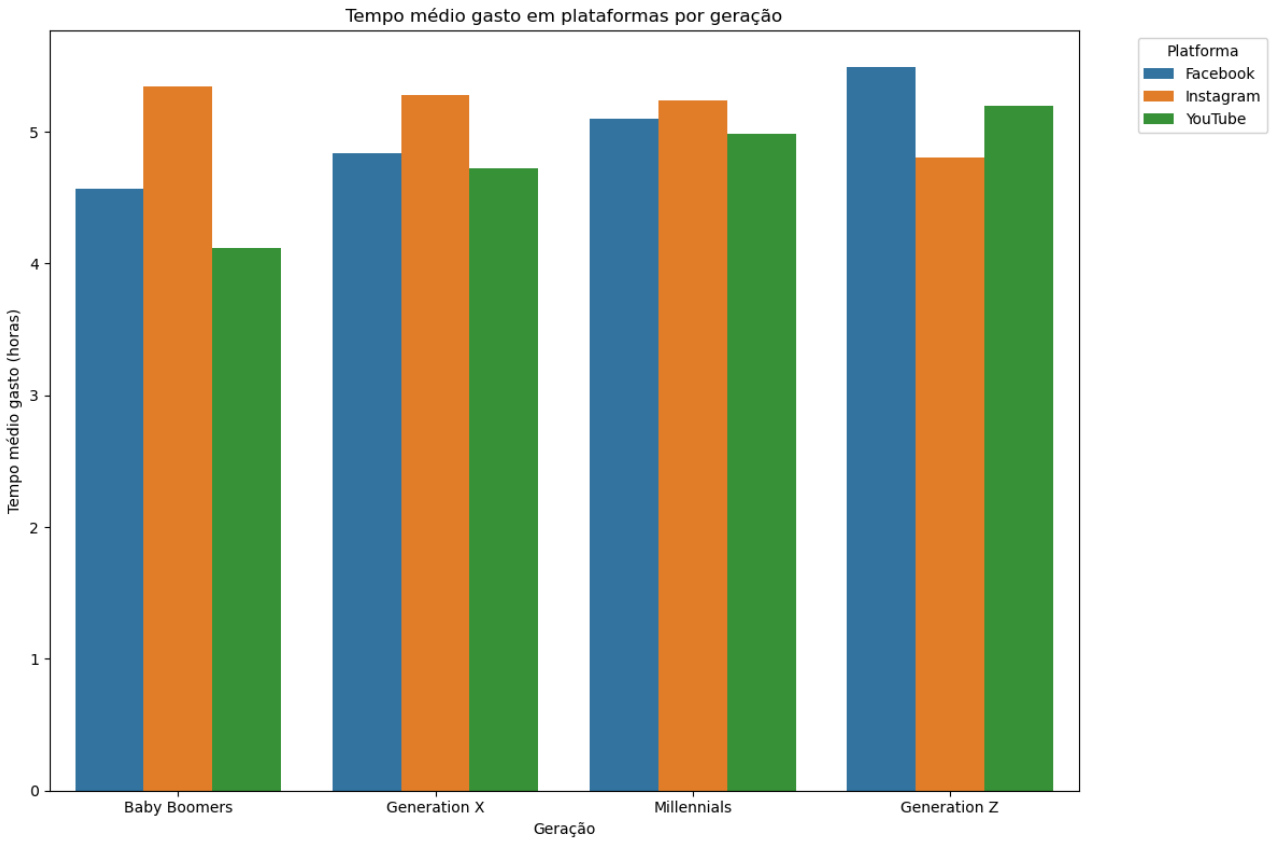
```
Value counts for platform:
platform
Instagram    363
YouTube      330
Facebook     307
Name: count, dtype: int64


Value counts for interests:
interests
Lifestlye    341
Sports       331
Travel       328
Name: count, dtype: int64


Value counts for location:
location
Australia         352
United Kingdom    329
United States     319
Name: count, dtype: int64


Value counts for demographics:
demographics
Rural        340
Sub_Urban    335
Urban        325
Name: count, dtype: int64


Value counts for profession:
profession
Marketer Manager     355
Software Engineer    336
Student              309
Name: count, dtype: int64
```

In [23]:
```python
# selecionando variáveis booleanas
boolean_features = df_copy3.select_dtypes(include=['bool'])
for column in boolean_features:
    print(f"Value counts for {column}:")
    print(df_copy3[column].value_counts())
    print("\n")
```

```
Value counts for indebt:
indebt
False    503
True     497
Name: count, dtype: int64


Value counts for isHomeOwner:
isHomeOwner
True     508
False    492
Name: count, dtype: int64


Value counts for Owns_Car:
Owns_Car
True     539
False    461
Name: count, dtype: int64
```

In [24]:
```python
# Tempo médio gasto em plataformas específicas de mídia social por geração
generation_socialmedia_avgtime = df_copy2.groupby(by=["generation", "platform"]).agg({"time_spent":"mean"}).re
```

In [25]:
```python
# Gráfico de barras do tempo médio gasto nas redes sociais por geração
plt.figure(figsize=(12, 8))
sns.barplot(data=generation_socialmedia_avgtime, x='generation', y='time_spent', hue='platform')
plt.xticks(rotation=0)
plt.title('Tempo médio gasto em plataformas por geração')
plt.xlabel('Geração')
plt.ylabel('Tempo médio gasto (horas)')
plt.legend(title='Platforma', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



In [26]:
```
plt.rcParams["figure.autolayout"] = True
```

In [27]:
```
df_copy2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   age           1000 non-null   int64
 1   gender        1000 non-null   object
 2   time_spent    1000 non-null   int64
 3   platform      1000 non-null   object
 4   interests     1000 non-null   object
 5   location      1000 non-null   object
 6   demographics  1000 non-null   object
 7   profession    1000 non-null   object
 8   income        1000 non-null   int64
 9   indebt        1000 non-null   bool
 10  isHomeOwner   1000 non-null   bool
 11  Owns_Car      1000 non-null   bool
 12  birth_year    1000 non-null   int64
 13  generation    1000 non-null   category
dtypes: bool(3), category(1), int64(4), object(6)
memory usage: 82.4+ KB
```

In [28]:
```
# definir sementes para reprodutibilidade
np.random.seed(0)
```

In [29]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy4 = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índice
```

In [30]:
```
# modelo de regressão para prever se uma pessoa está endividada
# Pré-processando os dados - Selecionando recursos e alvo para o modelo de regressão
X = df_copy4.drop(['income', 'indebt', 'isHomeOwner', 'Owns_Car'], axis=1)
y = df_copy4['income']

# Tratamento de variáveis categóricas por OneHotEncoding
categorical_features = ['gender', 'platform', 'interests', 'location', 'demographics', 'profession']
one_hot_encoder = OneHotEncoder(handle_unknown='ignore')

# Criando um transformador de coluna para aplicar transformações nas respectivas colunas
preprocessor = ColumnTransformer(transformers=[
    ('cat', one_hot_encoder, categorical_features)
], remainder='passthrough')

# Dividindo o conjunto de dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Criando um pipeline de regressão
regression_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
```

```
    ('regressor', LinearRegression())
])

# Treinando o modelo
regression_pipeline.fit(X_train, y_train)

# Previsão no conjunto de testes
y_pred = regression_pipeline.predict(X_test)

# Avaliando o modelo
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

mse, rmse, r2
# As métricas de desempenho do modelo de regressão são as seguintes:
# Erro Quadrático Médio (MSE);
# Raiz do erro quadrático médio (RMSE);
# Pontuação R^2.
```

Out [30]: (9038213.85259044, 3006.362229105209, -0.021950593979750854)

In [31]:
```
# modelo de classificação para prever se uma pessoa está endividada

# Selecionando recursos e destino para o modelo de classificação
X_classification = df_copy4.drop(['income', 'indebt'], axis=1) # Excluding 'income' as it's not a target here
y_classification = df_copy4['indebt']

# Dividindo o conjunto de dados em conjuntos de treinamento e teste para classificação
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(X_classification, y_classification, test_size=0.2,

# Criando um pipeline de classificação
classification_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000))
])

# Treinando o modelo
classification_pipeline.fit(X_train_c, y_train_c)

# Previsão no conjunto de testes
y_pred_c = classification_pipeline.predict(X_test_c)

# avaliando o modelo
accuracy = accuracy_score(y_test_c, y_pred_c)
conf_matrix = confusion_matrix(y_test_c, y_pred_c)

accuracy, conf_matrix
# As métricas de desempenho do modelo de classificação são as seguintes:
# Precisão (acurácia);
# Matriz de confusão:
# Verdadeiros Negativos;
# Falsos Positivos;
# Falsos Negativos;
# Verdadeiros Positivos.
```

Out [31]:
```
(0.485,
 array([[55, 41],
        [62, 42]], dtype=int64))
```

In [32]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy6 = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índice
```

In [33]:
```
# criando função de classificação do tempo de uso
def screen_time(number):
    if number>6:
        return "Extreme"
    elif number>4:
        return "High"
    elif number>2 :
        return "Moderate"
    else :
        return "Normal"
```

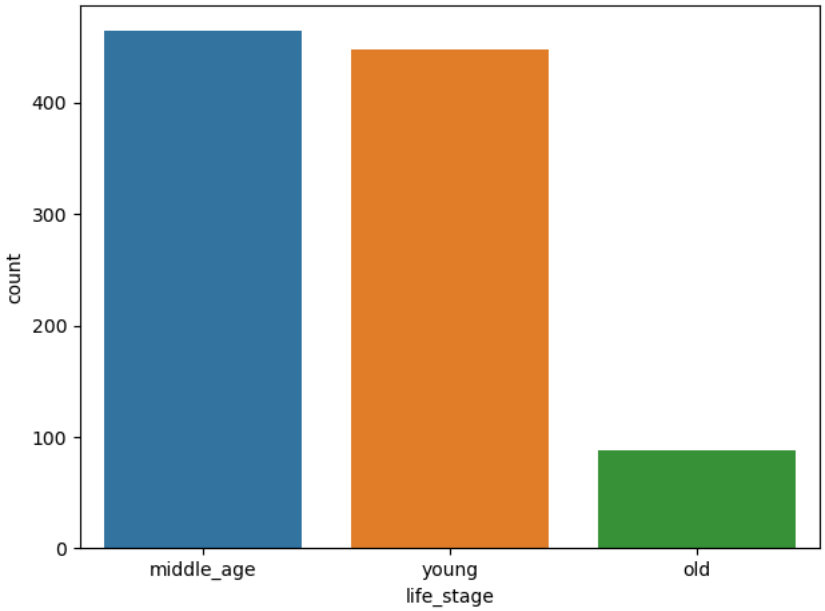
In [34]:
```
# criando função de classificação quanto à idade
def life_stage(age):
    if age > 60:
        return "old"
```

```
    elif age>=40:
        return "middle_age"
    elif age >= 18:
        return "young"
    else:
        return "teenage"
```

In [35]:
```
df_copy6['life_stage']= df_copy6['age'].apply(life_stage)
df_copy6['screen_time'] = df_copy6['time_spent'].apply(screen_time)
```

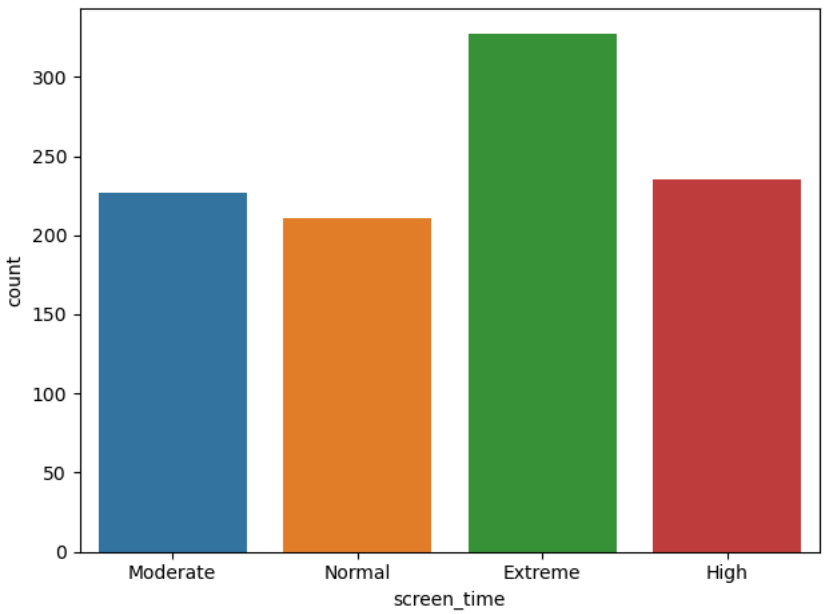
In [36]:
```
# gráfico de barras de classificação quanto à idade
sns.countplot(x= df_copy6['life_stage'])
```

Out [36]: <Axes: xlabel='life_stage', ylabel='count'>



In [37]:
```
# gráfico de barras de classificação quanto à idade
sns.countplot(x= df_copy6['screen_time'])
```

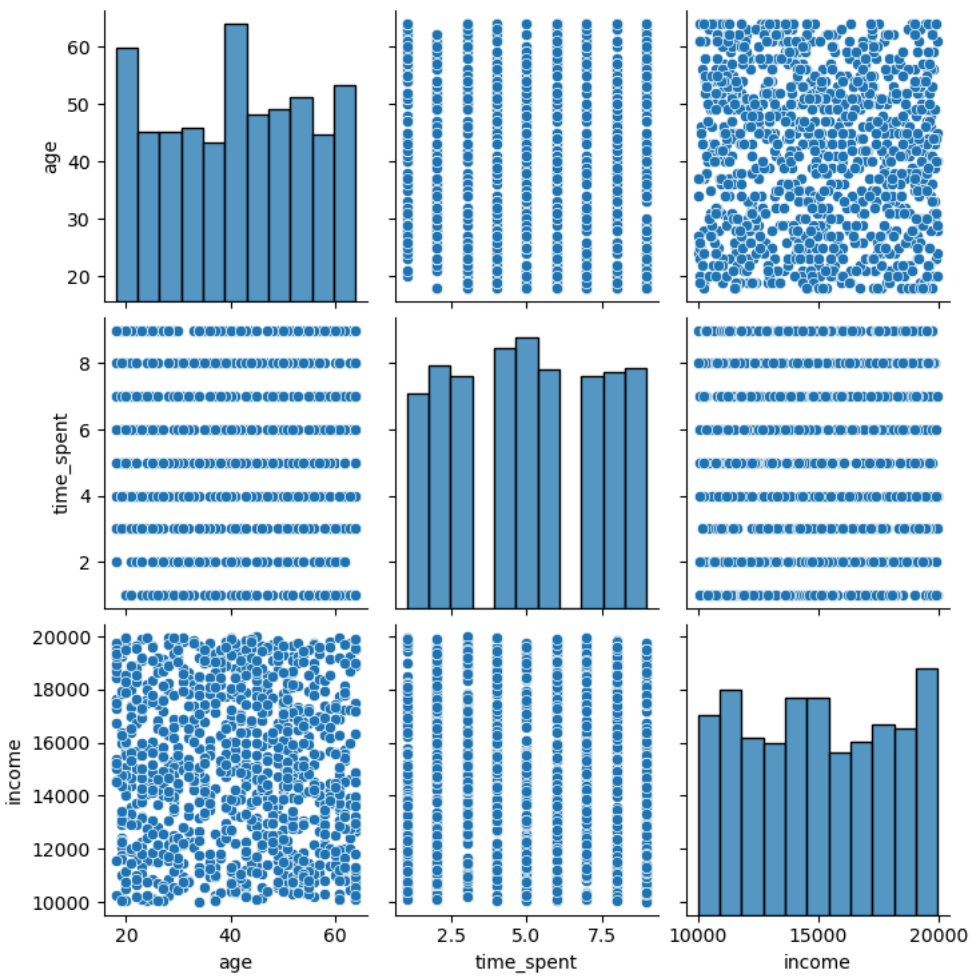
Out [37]: <Axes: xlabel='screen_time', ylabel='count'>



In [38]:
```
# gráfico de distribuição bivariada de pares
sns.pairplot(df_copy6[['age','time_spent','income']])
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:

The figure layout has changed to tight
```

Out [38]: <seaborn.axisgrid.PairGrid at 0x277cfc3ff50>

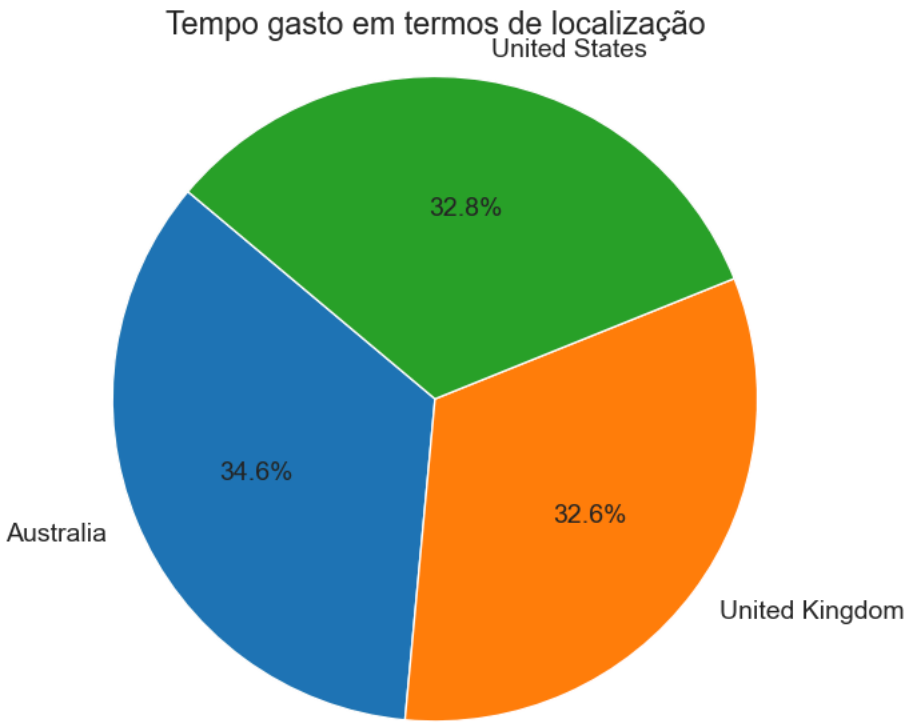


In [39]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy7 = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índice
```

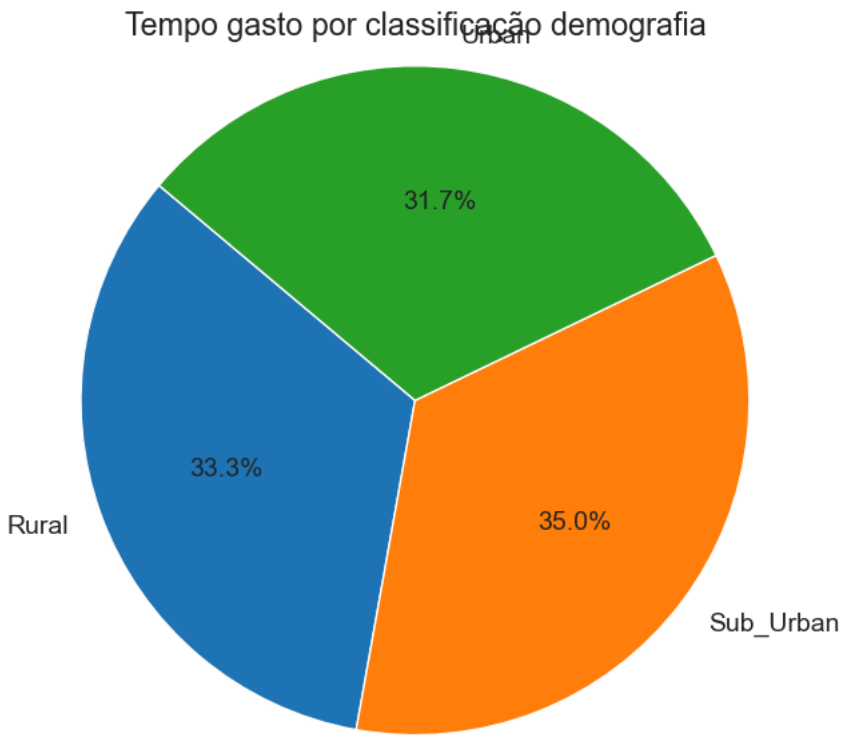
In [40]:
```
# criando estilo para os gráficos com o matplotlib
sns.set_style('darkgrid')
matplotlib.rcParams['font.size']=14
matplotlib.rcParams['figure.figsize']=(9,5)
matplotlib.rcParams['figure.facecolor']='#00000000'
```

In [ ]:
```

```

In [41]:
```
# Análise exploratória de dados
avg_time_on_sm=df_copy7.groupby(by=['platform']).agg({'time_spent':'mean'}).reset_index()
gender_wise=df_copy7.groupby(by=['gender']).agg({'time_spent':'mean'}).reset_index()
location_wise=df_copy7.groupby(by=['location']).agg({'time_spent':'mean'}).reset_index()

labels=['Australia','United Kingdom','United States']
sizes=[5.218750,4.908815,4.943574]
plt.figure(figsize=(7, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title('Tempo gasto em termos de localização')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.show()
```

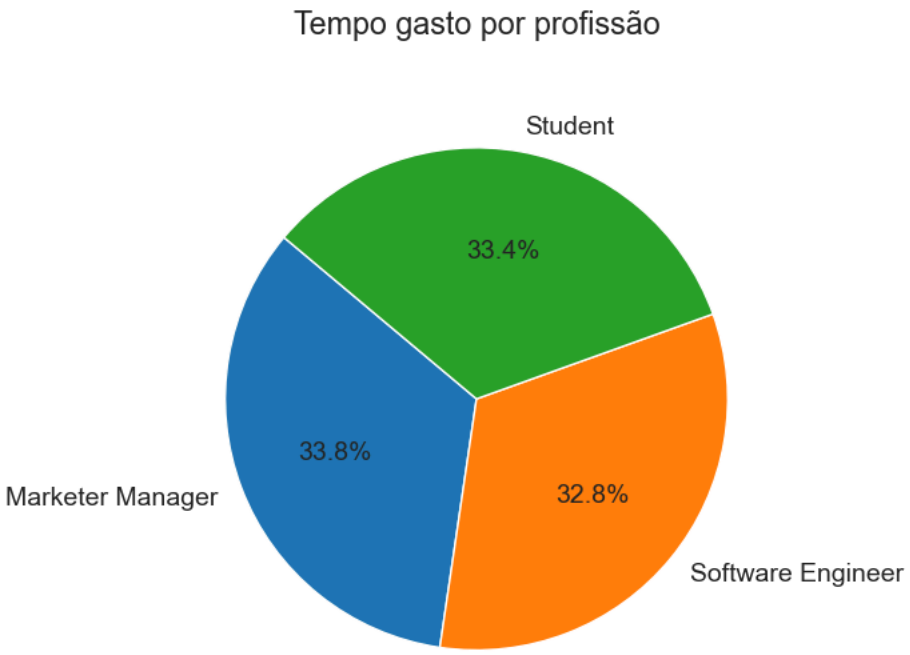


In [42]:
```
demographics_wise=df_copy7.groupby(by=['demographics']).agg({'time_spent':'mean'}).reset_index()
labels=['Rural','Sub_Urban','Urban']
sizes=[5.020588,5.271642,4.787692]
plt.figure(figsize=(7, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title('Tempo gasto por classificação demografia')
plt.axis('equal')
plt.show()
```



In [ ]:
```
```

In [43]:
```
profession_wise=df_copy7.groupby(by=['profession']).agg({'time_spent':'mean'}).reset_index()
labels=['Marketer Manager','Software Engineer','Student']
sizes=[5.095775,4.949405,5.038835]
plt.figure(figsize=(7, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title('Tempo gasto por profissão')
plt.axis('equal')
plt.show()
```

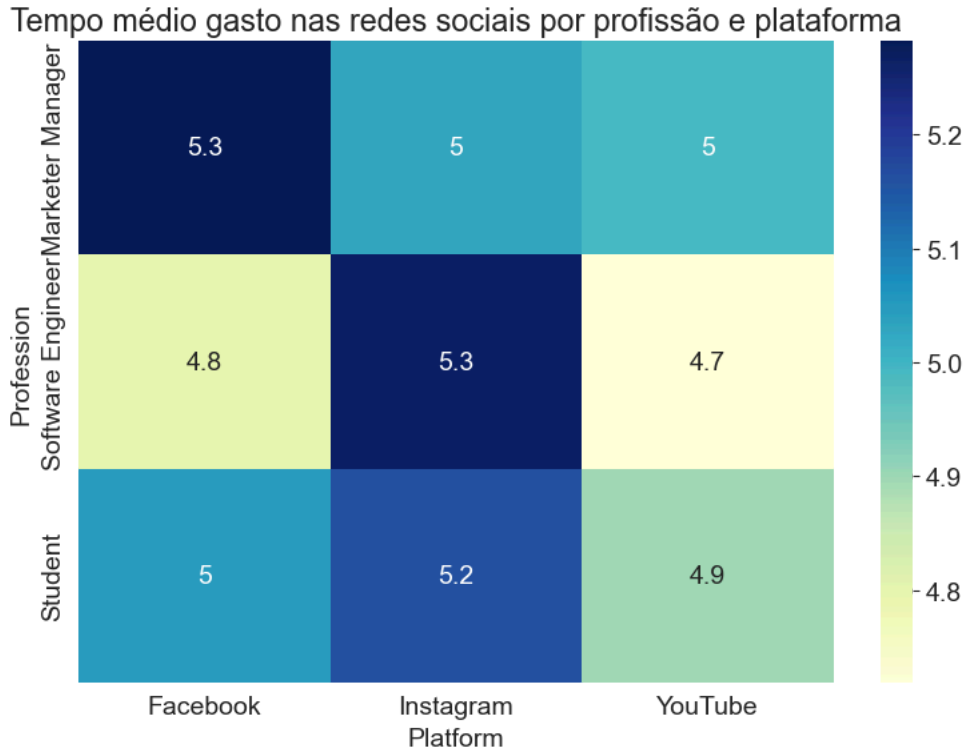


In [ ]:

```
```

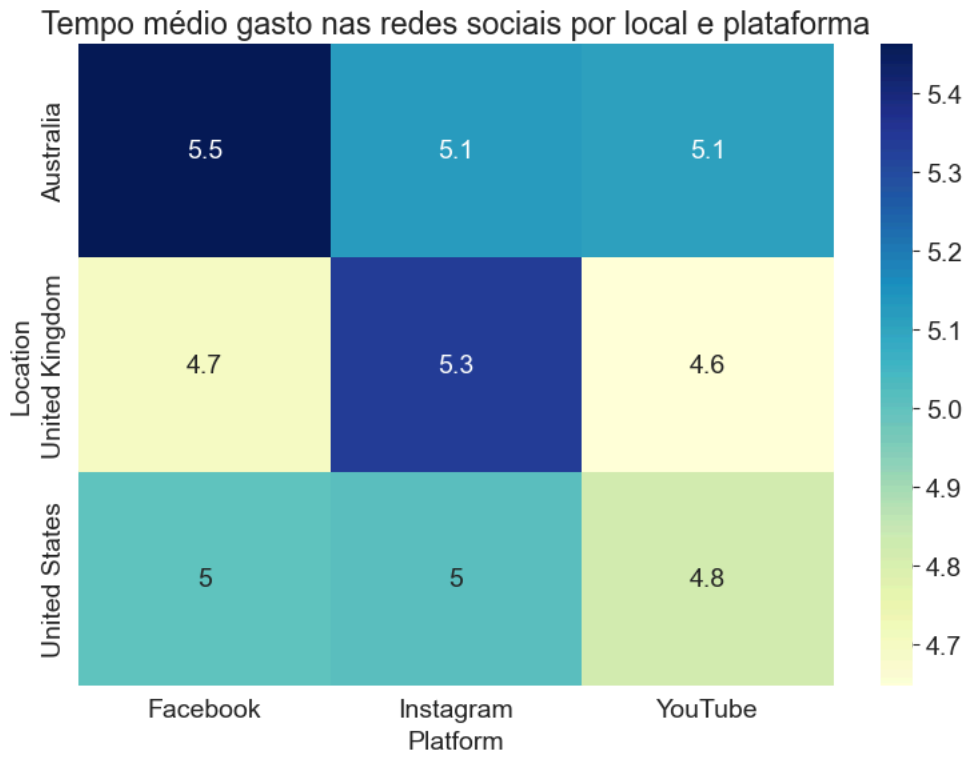
In [44]:

```
professionals_avg_time=df_copy7.groupby(by=['profession','platform']).agg({'time_spent':'mean'}).reset_index()
plt.figure(figsize=(8, 6))
heatmap = sns.heatmap(professionals_avg_time.pivot(index='profession', columns='platform', values='time_spent'
plt.title('Tempo médio gasto nas redes sociais por profissão e plataforma')
plt.xlabel('Platform')
plt.ylabel('Profession')
plt.show()
```

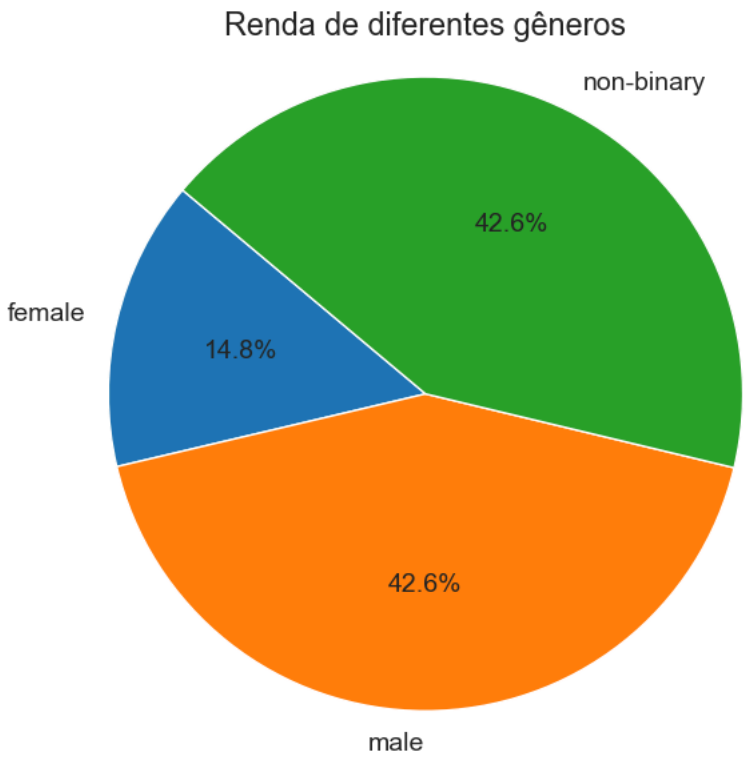


In [45]:

```
diff_loc=df_copy7.groupby(by=['location','platform']).agg({'time_spent':'mean'}).reset_index()
plt.figure(figsize=(8, 6))
heatmap = sns.heatmap(diff_loc.pivot(index='location', columns='platform', values='time_spent'), annot=True, cr
plt.title('Tempo médio gasto nas redes sociais por local e plataforma')
plt.xlabel('Platform')
plt.ylabel('Location')
plt.show()
```



In [46]:
```
labels=['female','male','non-binary']
sizes=[5185.770393,14919.620178,14941.027108]
plt.figure(figsize=(7, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title('Renda de diferentes gêneros')
plt.axis('equal')
plt.show()
```



In [47]:
```
correlation_homeowner = df_copy7['time_spent'].corr(df_copy7['isHomeOwner'])
correlation_car_owner = df_copy7['time_spent'].corr(df_copy7['Owns_Car'])

print(f"Correlação entre tempo gasto e propriedade de casa própria: {correlation_homeowner}")
print(f"Correlação entre tempo gasto e propriedade de carro: {correlation_car_owner}")
```

```
Correlação entre tempo gasto e propriedade de casa própria: 0.029388389343285318
Correlação entre tempo gasto e propriedade de carro: -0.020270983067538714
```

In [48]:
```
# A maioria das pessoas possui um carro depois dos 40 anos;
# A maioria dos usuários passa a maior parte do tempo no Instagram;
# Entre os diferentes gêneros, as mulheres passam mais tempo nas redes sociais;
# Entre diferentes locais a maioria dos usuários pertence à Austrália;
# Entre diferentes Dados demográficos a maioria dos usuários pertence a Sub_Urban;
# Entre diferentes profissões a maioria dos usuários são Gerente de mercado;
```

```
# Os gerentes de mercado passam a maior parte do tempo no Facebook, enquanto os engenheiros de software e estu
# Na Austrália, as pessoas passam a maior parte do tempo no Facebook, enquanto nos EUA e no Reino Unido as pes
# Os jovens adultos do sexo feminino passam a maior parte do tempo nas redes sociais;
# O rendimento dos homens é superior ao rendimento das mulheres;
# Não existe uma relação clara entre o tempo gasto nas redes sociais e a posse de casa ou carro.
```

In [49]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy8 = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índice
```

In [50]:
```
# tempo médio gasto pelos profissionais nas plataformas
profession_platform=pd.DataFrame(df_copy8.groupby('profession')['platform'].value_counts().sort_values(ascendi
profession_time=pd.DataFrame(df_copy8.groupby('profession')['time_spent'].mean())
all_information=pd.merge(profession_platform,profession_time,on='profession')
all_information.style.background_gradient(cmap='ocean')
```

Out [50]:

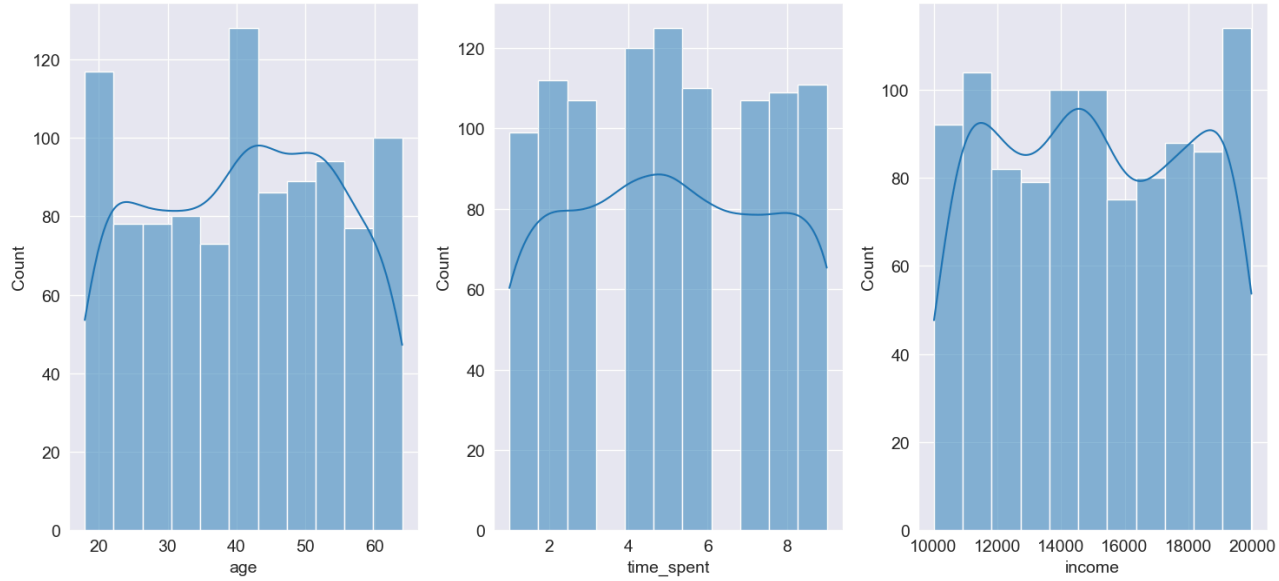
profession	Facebook	Instagram	YouTube	time_spent
Marketer Manager	110	128	117	5.095775
Software Engineer	94	128	114	4.949405
Student	103	107	99	5.038835

In [51]:
```
# gráfico de "rosca" relacionando as plataformas com as categorias
demographics = ['age', 'gender', 'profession']

for demographic in demographics:
    platform_info = df_copy8.groupby(demographic)['platform'].value_counts().unstack()
    fig = px.pie(platform_info,
                 names=platform_info.columns,
                 title=f'Platform Preference by {demographic.capitalize()}',
                 labels={'platform':'Platform'},color_discrete_sequence=px.colors.sequential.RdBu,hole=.3)
    fig.show()
    pyo.iplot(fig)
```

In [52]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy9 = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índices
```

In [53]:
```
# gráfico de barras Tempo médio gasto em plataformas de mídia social por local
fig=px.bar(df_copy9.groupby(by=['location','platform']).agg({'time_spent':'mean'}).reset_index(),
           x='location',y='time_spent',color='platform',barmode='group',
           title="Tempo médio gasto em plataformas de mídia social por local")
fig.show(render='iframe')
```

In [54]:
```
# histogramas da idade, tempo de uso e renda
plt.figure(figsize=(15,7))
plt.subplot(1,3,1)
sns.histplot(df_copy9.age, kde=True)
plt.subplot(1,3,2)
sns.histplot(df_copy9.time_spent, kde=True)
plt.subplot(1,3,3)
sns.histplot(df_copy9.income, kde=True)
plt.show()
```



In [55]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy10 = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índic
```

In [56]:
```
# criando uma função para conversão de booleano em binário
def temp(x):
    if x==True:
        return 1
    else:
        return 0

df_copy10.indebt=df_copy10.indebt.apply(temp)
df_copy10.isHomeOwner=df_copy10.isHomeOwner.apply(temp)
df_copy10.Owns_Car=df_copy10.Owns_Car.apply(temp)
```

In [57]:
```
# correlação entre as variáveis
df_copy10[['age', 'time_spent', 'income', 'indebt', 'isHomeOwner', 'Owns_Car']].corr()
```

Out [57]:

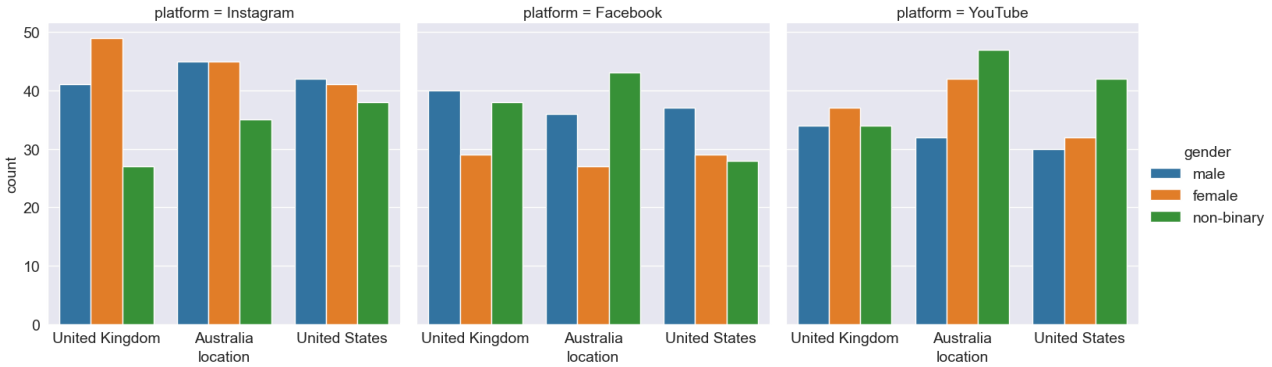
	age	time_spent	income	indebt	isHomeOwner	Owns_Car
age	1.000000	-0.033827	-0.087391	-0.017055	-0.005321	0.006921
time_spent	-0.033827	1.000000	0.004757	0.013079	0.029388	-0.020271
income	-0.087391	0.004757	1.000000	0.037860	0.006072	0.019789
indebt	-0.017055	0.013079	0.037860	1.000000	0.038102	-0.035641
isHomeOwner	-0.005321	0.029388	0.006072	0.038102	1.000000	-0.051411
Owns_Car	0.006921	-0.020271	0.019789	-0.035641	-0.051411	1.000000

In [58]:
```
# mapa de calor da correlação entre as variáveis
sns.heatmap(df_copy10[['age', 'time_spent', 'income', 'indebt', 'isHomeOwner', 'Owns_Car']].corr(), cmap='Blue
plt.show()
```

In [59]:
```
# Existe correlação positiva entre idade e renda.
#        Isto sugere que à medida que as pessoas envelhecem, o seu rendimento tende a aumentar.
#        Isso pode ser devido a fatores como progressão na carreira e acúmulo de experiência.
# Existe correlação positiva entre o tempo gasto e o endividamento.
#        Isto pode indicar que quanto mais tempo as pessoas passam nas redes sociais, maior é a probabilidade
#        Isso pode estar relacionado a hábitos de consumo ou escolhas de estilo de vida.
# Existe uma correlação notável entre renda e ser proprietário de uma casa.
#        Isto sugere que níveis de rendimento mais elevados podem levar a taxas mais elevadas de aquisição de
# Existe uma correlação entre estar endividado e possuir um carro.
#        Isso pode ser devido aos custos associados à propriedade do carro, como empréstimos, manutenção e seg
```

In [60]:
```
# gráficos de barras relacionando a utilização das plataformas nos países por gênero
sns.catplot(data=df_copy10, col='platform', hue='gender', x='location', kind='count')
plt.show()
```

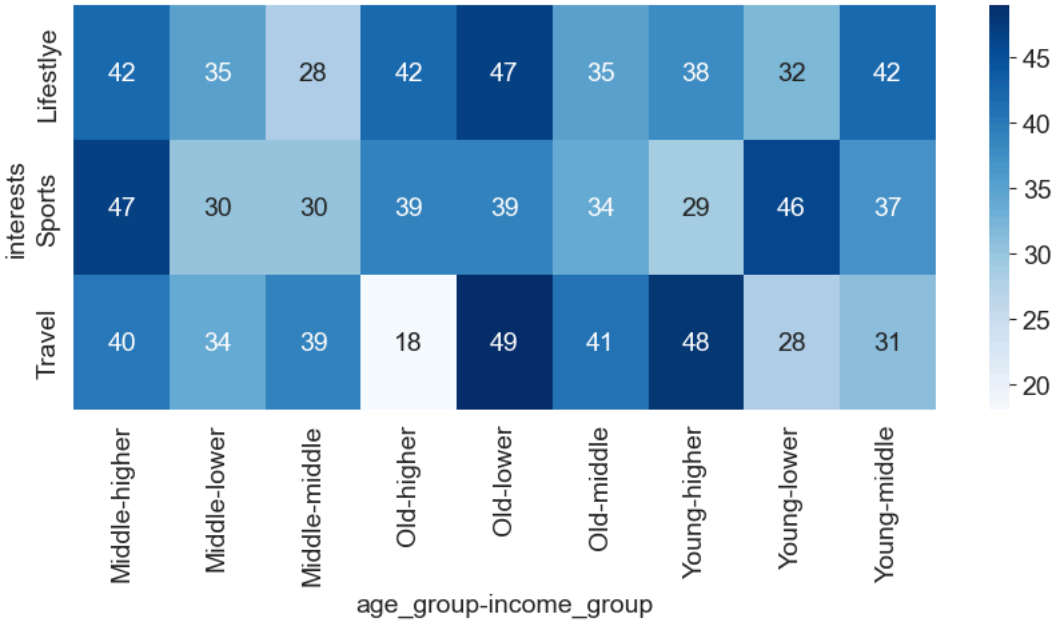
```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:

The figure layout has changed to tight
```



In [61]:
```
# O Instagram parece ser popular entre um público mais amplo em comparação com o Facebook e o YouTube.
# Cada plataforma (Instagram, Facebook, YouTube) exibe dados demográficos de gênero distintos nos locais estud
#      indicando preferências ou padrões de uso variados entre diferentes grupos de usuários. Por exemplo, a b
#      utilizadores do Instagram no Reino Unido é dominada por mulheres, enquanto a base de utilizadores do Yo
#      nos Estados Unidos é predominantemente não binária.
```

In [62]:
```
# criando classificações para idade e renda
np.linspace(18, 64, 4)
def age_group(x):
    if x<34:
        return 'Young'
    elif x>48:
        return 'Old'
    else:
        return 'Middle'

df_copy10['age_group']=df_copy10.age.apply(age_group)

np.linspace(10012, 19980, 4)
def income_group(x):
    if x<13335:
        return 'lower'
    elif x>16657:
        return 'higher'
    else:
        return 'middle'

df_copy10['income_group']=df_copy10.income.apply(income_group)
```

In [63]:
```
# mapa de calor dos interesses por idade e renda
sns.heatmap(pd.crosstab(df_copy10.interests, [df_copy10.age_group, df_copy10.income_group]), annot=True, cmap=
plt.show()
```



In [64]:
```
# normalizando os dados de interesse, profissão e plataforma
pd.crosstab(df_copy10.interests, [df_copy10.profession, df_copy10.platform], margins=True, normalize=True)
```

Out [64]:

profession	Marketer Manager			Software Engineer			Student			All
platform	Facebook	Instagram	YouTube	Facebook	Instagram	YouTube	Facebook	Instagram	YouTube	
interests										
Lifestlye	0.029	0.046	0.047	0.032	0.044	0.029	0.031	0.040	0.043	0.341
Sports	0.047	0.047	0.027	0.032	0.047	0.038	0.040	0.031	0.022	0.331
Travel	0.034	0.035	0.043	0.030	0.037	0.047	0.032	0.036	0.034	0.328
All	0.110	0.128	0.117	0.094	0.128	0.114	0.103	0.107	0.099	1.000

In [65]:
```
# normalizando os dados de interesse, profissão e plataforma
pd.crosstab(df_copy10.interests, [df_copy10.profession, df_copy10.platform], margins=True, normalize='index')
```

Out [65]:

profession	Marketer Manager			Software Engineer			Student		
platform	Facebook	Instagram	YouTube	Facebook	Instagram	YouTube	Facebook	Instagram	YouTube
interests									
Lifestlye	0.085044	0.134897	0.137830	0.093842	0.129032	0.085044	0.090909	0.117302	0.126100
Sports	0.141994	0.141994	0.081571	0.096677	0.141994	0.114804	0.120846	0.093656	0.066465
Travel	0.103659	0.106707	0.131098	0.091463	0.112805	0.143293	0.097561	0.109756	0.103659
All	0.110000	0.128000	0.117000	0.094000	0.128000	0.114000	0.103000	0.107000	0.099000

In [66]:
```
# normalizando os dados de interesse, profissão e plataforma
pd.crosstab(df_copy10.interests, [df_copy10.profession, df_copy10.platform], margins=True, normalize='columns'
```

Out [66]:

profession	Marketer Manager			Software Engineer			Student			All
platform	Facebook	Instagram	YouTube	Facebook	Instagram	YouTube	Facebook	Instagram	YouTube	
interests										
Lifestlye	0.263636	0.359375	0.401709	0.340426	0.343750	0.254386	0.300971	0.373832	0.434343	0.341
Sports	0.427273	0.367188	0.230769	0.340426	0.367188	0.333333	0.388350	0.289720	0.222222	0.331
Travel	0.309091	0.273438	0.367521	0.319149	0.289062	0.412281	0.310680	0.336449	0.343434	0.328

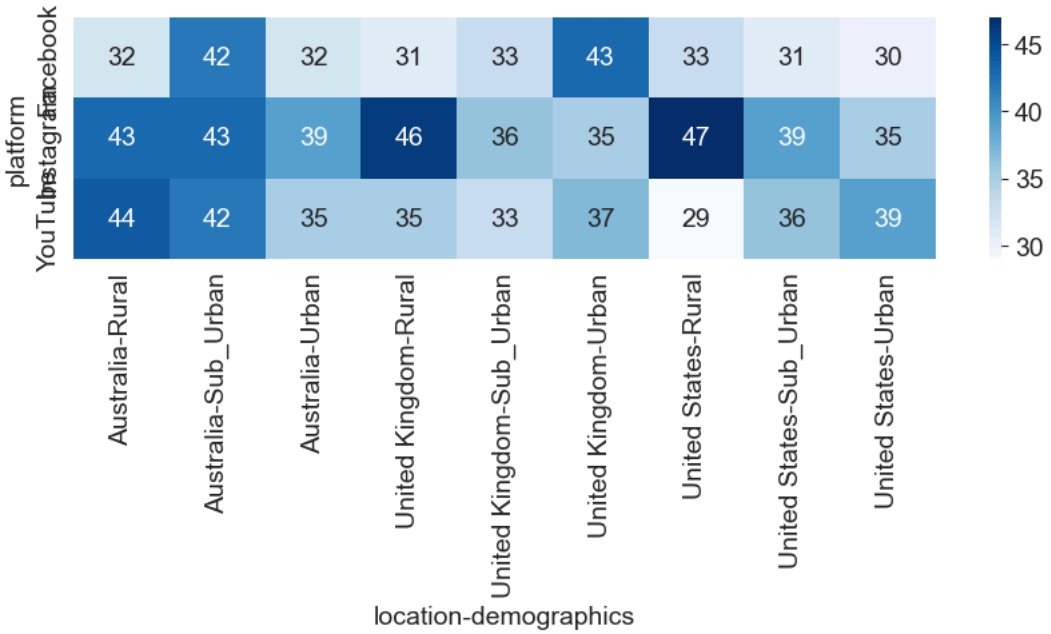
In [67]:
```
# Estilo de vida (34,1%), Desporto (33,1%) e Viagens (32,8%), o que indica uma distribuição relativamente equi
#        de interesses em geral, com uma ligeira inclinação para Estilo de vida.
# Gerentes de marketing com maior preferência no Instagram (4,6%) e YouTube (4,7%). Também têm interesse signi
#        em Esportes, principalmente no Facebook (4,7%) e Viagens, principalmente no YouTube (4,3%).
# Engenheiros de Software têm um forte interesse em Estilo de Vida, sendo o Instagram (4,4%) a plataforma pref
#        O interesse por esportes é notavelmente alto em todas as plataformas, sendo o Instagram (4,7%) o mais
#        seguido pelo Facebook (4,7%). Eles também demonstram um interesse considerável em Viagens no YouTube
# Os alunos demonstram um interesse relativamente equilibrado entre plataformas, com o estilo de vida sendo o
#        no YouTube (4,3%) e os esportes sendo o mais alto no Facebook (4,7%). O interesse por viagens também
#        distribuído pelas plataformas, sendo Instagram (4,0%) e YouTube (4,0%) as preferidas. Estilo de vida
#        Esportes (33,1%) e Viagens (32,8%) o que indica uma distribuição relativamente equilibrada de interes
#        com uma ligeira inclinação para o estilo de vida.
```

In [68]:
```
# tabela cruzada relacionando plataforma, localização e classificação demográfica
pd.crosstab(df_copy10.platform, [df_copy10.location, df_copy10.demographics])
```

Out [68]:

location	Australia			United Kingdom			United States		
demographics	Rural	Sub_Urban	Urban	Rural	Sub_Urban	Urban	Rural	Sub_Urban	Urban
platform									
Facebook	32	42	32	31	33	43	33	31	30
Instagram	43	43	39	46	36	35	47	39	35
YouTube	44	42	35	35	33	37	29	36	39

In [69]:
```
# mapa de calor com dados da tabela cruzada
sns.heatmap(pd.crosstab(df_copy10.platform, [df_copy10.location, df_copy10.demographics]), annot=True, cmap='B
plt.show()
```



In [70]:
```
# tabela cruzada normalizada
pd.crosstab(df.platform, [df_copy10.location, df_copy10.demographics], margins=True, normalize=True)
```

Out [70]:

location	Australia			United Kingdom			United States			All
demographics	Rural	Sub_Urban	Urban	Rural	Sub_Urban	Urban	Rural	Sub_Urban	Urban	
platform										
Facebook	0.032	0.042	0.032	0.031	0.033	0.043	0.033	0.031	0.030	0.307
Instagram	0.043	0.043	0.039	0.046	0.036	0.035	0.047	0.039	0.035	0.363
YouTube	0.044	0.042	0.035	0.035	0.033	0.037	0.029	0.036	0.039	0.330
All	0.119	0.127	0.106	0.112	0.102	0.115	0.109	0.106	0.104	1.000

In [71]:
```
# tabela cruzada normalizada
pd.crosstab(df_copy10.platform, [df_copy10.location, df_copy10.demographics], margins=True, normalize='index')
```

Out [71]:

location	Australia			United Kingdom			United States		
demographics	Rural	Sub_Urban	Urban	Rural	Sub_Urban	Urban	Rural	Sub_Urban	Urban
platform									
Facebook	0.104235	0.136808	0.104235	0.100977	0.107492	0.140065	0.107492	0.100977	0.097720
Instagram	0.118457	0.118457	0.107438	0.126722	0.099174	0.096419	0.129477	0.107438	0.096419
YouTube	0.133333	0.127273	0.106061	0.106061	0.100000	0.112121	0.087879	0.109091	0.118182
All	0.119000	0.127000	0.106000	0.112000	0.102000	0.115000	0.109000	0.106000	0.104000

In [72]:
```
# tabela cruzada normalizada
pd.crosstab(df_copy10.platform, [df_copy10.location, df_copy10.demographics], margins=True, normalize='columns
```

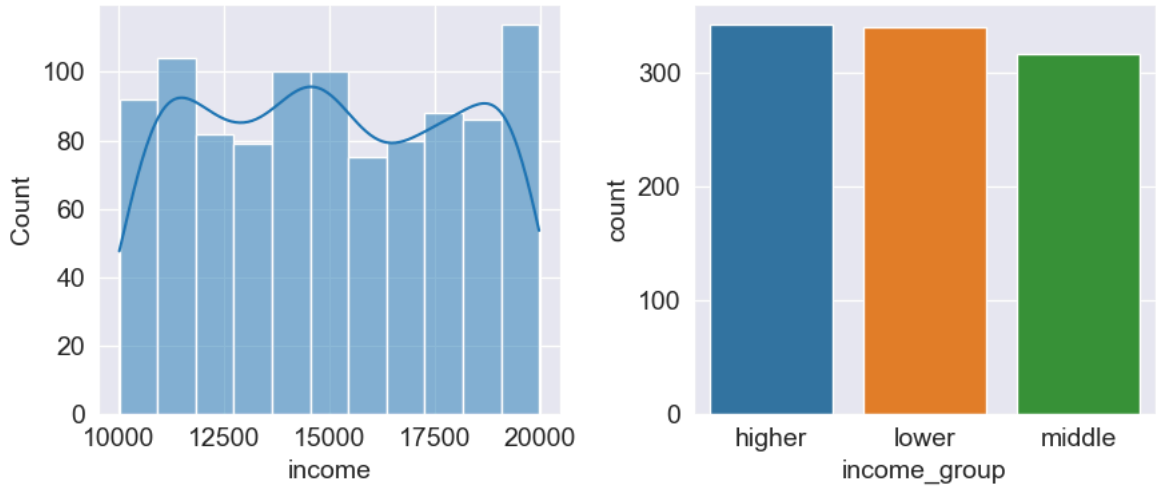
Out [72]:

location	Australia			United Kingdom			United States			All
demographics	Rural	Sub_Urban	Urban	Rural	Sub_Urban	Urban	Rural	Sub_Urban	Urban	
platform										
Facebook	0.268908	0.330709	0.301887	0.276786	0.323529	0.373913	0.302752	0.292453	0.288462	0.307
Instagram	0.361345	0.338583	0.367925	0.410714	0.352941	0.304348	0.431193	0.367925	0.336538	0.363
YouTube	0.369748	0.330709	0.330189	0.312500	0.323529	0.321739	0.266055	0.339623	0.375000	0.330

In [73]:
```
# Instagram: O Instagram é a plataforma de mídia social mais popular em geral, com uma taxa de uso de 36,3%.
#         Além disso, as taxas de utilização mais elevadas entre utilizadores rurais nos Estados Unidos (12,95
#         Unido (2,67%) e utilizadores suburbanos na Austrália indicam preferências demográficas específicas ou
```

```
#         comportamentos em relação ao envolvimento nas redes sociais em 11,84%.
# YouTube: o YouTube tem uma base de usuários substancial, com uma taxa de uso de aproximadamente 33,0%, acompa
#         de perto o domínio do Instagram. Os dados também destacam preferências demográficas específicas, com
#         utilizadores rurais na Austrália a apresentarem a maior utilização, com 13,33%, seguidos pelos utili
#         suburbanos na Austrália, utilizadores urbanos nos Estados Unidos e utilizadores urbanos no Reino Uni
#         com taxas de utilização de 12,72%. 11,81% e 11,21%, respectivamente.
# Facebook: o percentual de uso do Facebook é menor em comparação ao Instagram e YouTube, ele ainda detém uma
#         parcela significativa do uso geral, respondendo por aproximadamente 30,7%. Especificamente, entre os
#         urbanos no Reino Unido e os utilizadores suburbanos na Austrália, a utilização do Facebook é relativa
#         elevada, situando-se em 14% e 13,68%, respetivamente.
```

In [74]:
```
# análise financeira
plt.figure(figsize=(9,4))
plt.subplot(1,2,1)
sns.histplot(data=df_copy10, x='income', kde=True)
plt.subplot(1,2,2)
sns.countplot(data=df_copy10, x='income_group')
plt.show()
```



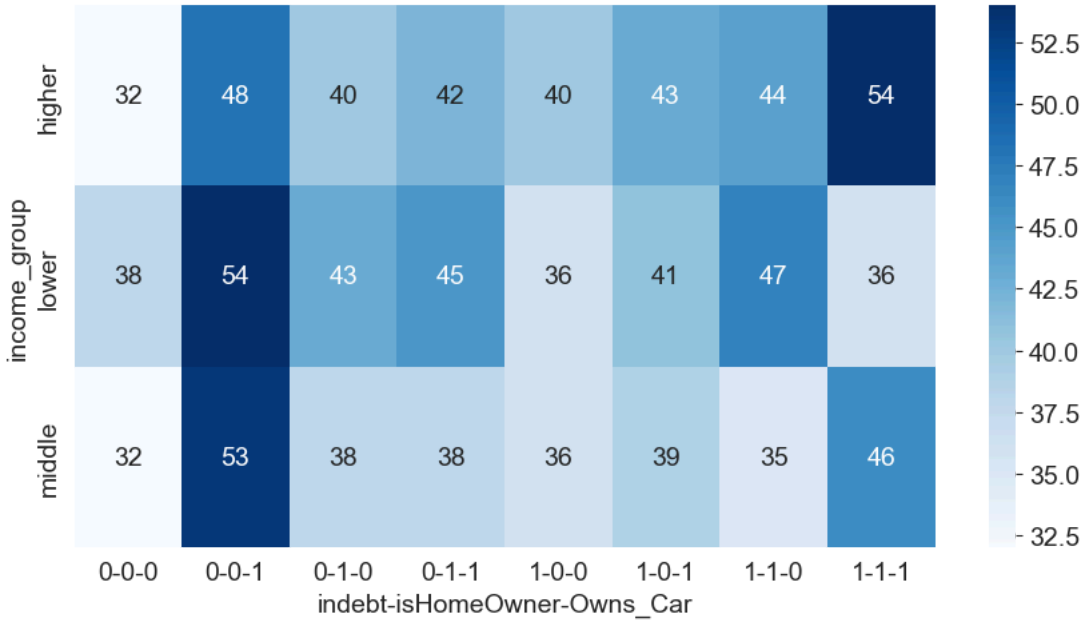
In [75]:
```
# No gráfico da esquerda, existem flutuações significativas na contagem entre diferentes níveis de rendimento,
#    indicando variabilidade na distribuição de rendimento.
# O gráfico da direita mostra que a maioria dos indivíduos pertence ao grupo de rendimentos mais elevados.
# As pessoas com rendimentos mais elevados têm menos probabilidades de se endividarem e mais probabilidades de
#    própria. Isto pode dever-se à sua estabilidade financeira, que lhes permite gerir dívidas de forma eficie
#    a casa própria.
# O grupo de rendimentos mais baixos tem um nível notável de endividamento que pode ser atribuído à instabilid
#    financeira ou à falta de acesso a recursos para gestão financeira.
# A posse de um automóvel não varia significativamente com os níveis de rendimento, indicando que pode ser con
#    uma necessidade, independentemente da situação económica de alguém.
```

In [76]:
```
# tabela cruzada renda, endividamento, posses de casas e de carros
pd.crosstab(df_copy10.income_group, [df_copy10.indebt, df_copy10.isHomeOwner, df_copy10.Owns_Car], margins=Tru
```

Out [76]:

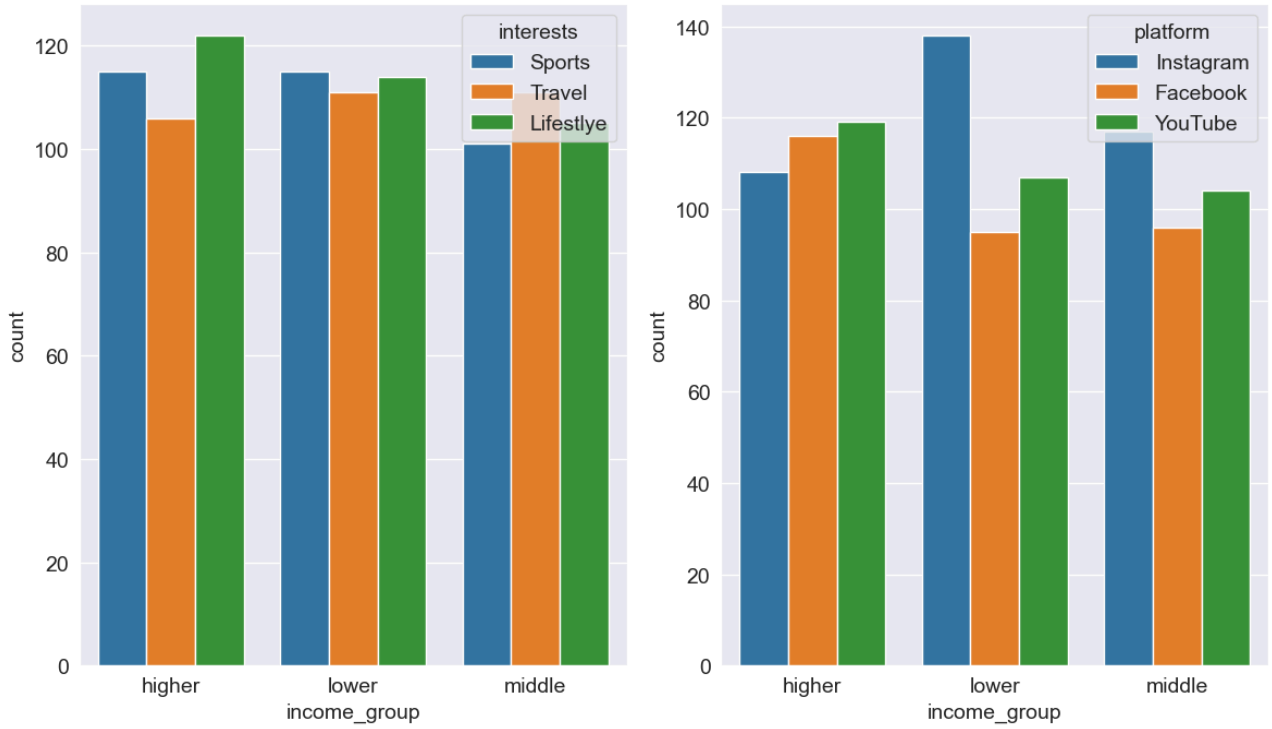
indebt	0				1				All
isHomeOwner	0		1		0		1		
Owns_Car	0	1	0	1	0	1	0	1	
income_group									
higher	32	48	40	42	40	43	44	54	343
lower	38	54	43	45	36	41	47	36	340
middle	32	53	38	38	36	39	35	46	317
All	102	155	121	125	112	123	126	136	1000

In [77]:
```
# mapa de calor d atabela cruzada
sns.heatmap(pd.crosstab(df_copy10.income_group, [df_copy10.indebt, df_copy10.isHomeOwner, df_copy10.Owns_Car])
plt.show()
```



In [78]:
```
# Indivíduos com renda mais alta, que não têm dívidas, são proprietários de casa própria e possuem carro, têm a
#   pontuação mais alta de 54.
# O grupo de renda mais baixa tende a ter pontuações mais altas quando não está endividado, independentemente d
#   proprietário de uma casa ou de um carro.
# O grupo de renda média tem o maior número de usuários quando não estão endividados, mas possuem uma casa e um
```

In [79]:
```
# gráficos de barras relacionando interesses e plataformas com o nível de renda
plt.figure(figsize=(12,7))
plt.subplot(1,2,1)
sns.countplot(data=df_copy10, x='income_group', hue='interests')
plt.subplot(1,2,2)
sns.countplot(data=df_copy10, x='income_group', hue='platform')
plt.show()
```



In [80]:
```
# Indivíduos de renda média demonstram um interesse equilibrado em esportes, viagens e estilo de vida em compar
#    com outros grupos.
# Indivíduos de renda mais alta estão mais inclinados ao Instagram, indicando uma preferência por conteúdo visu
#    ou talvez uma plataforma com status ou estética percebida mais elevada.
# Indivíduos de baixa renda têm uma preferência notável pelo Facebook, o que pode ser devido à sua acessibilida
#    à diversidade de ofertas de conteúdo.
```

In [81]:
```
# tabela cruzada renda, plataforma e interesses normalizada
pd.crosstab(df_copy10.income_group, [df_copy10.platform, df_copy10.interests], margins=True, normalize='all')
```

Out [81]:

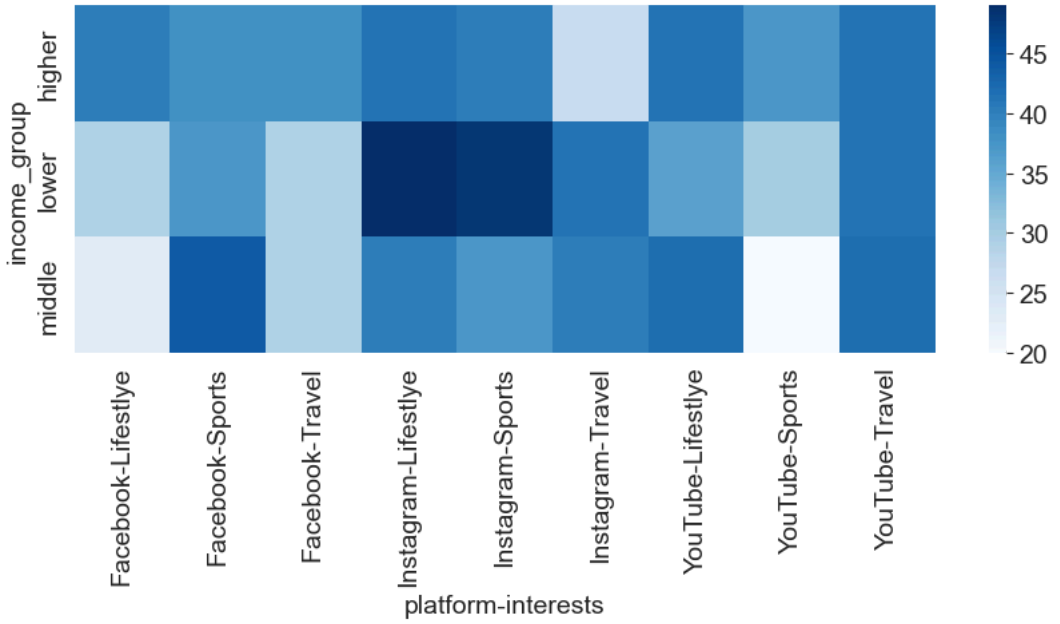
platform	Facebook			Instagram			YouTube			All
interests	Lifestlye	Sports	Travel	Lifestlye	Sports	Travel	Lifestlye	Sports	Travel	
income_group										
higher	0.040	0.038	0.038	0.041	0.040	0.027	0.041	0.037	0.041	0.343
lower	0.029	0.037	0.029	0.049	0.048	0.041	0.036	0.030	0.041	0.340
middle	0.023	0.044	0.029	0.040	0.037	0.040	0.042	0.020	0.042	0.317
All	0.092	0.119	0.096	0.130	0.125	0.108	0.119	0.087	0.124	1.000

In [82]:
```
# tabela cruzada renda, plataforma e interesses normalizada
pd.crosstab(df_copy10.income_group, [df_copy10.platform, df_copy10.interests], margins=True, normalize=0)
```

Out [82]:

platform	Facebook			Instagram			YouTube		
interests	Lifestlye	Sports	Travel	Lifestlye	Sports	Travel	Lifestlye	Sports	Travel
income_group									
higher	0.116618	0.110787	0.110787	0.119534	0.116618	0.078717	0.119534	0.107872	0.119534
lower	0.085294	0.108824	0.085294	0.144118	0.141176	0.120588	0.105882	0.088235	0.120588
middle	0.072555	0.138801	0.091483	0.126183	0.116719	0.126183	0.132492	0.063091	0.132492
All	0.092000	0.119000	0.096000	0.130000	0.125000	0.108000	0.119000	0.087000	0.124000

In [83]:
```
# mapa de calor da tabela cruzada
sns.heatmap(pd.crosstab(df_copy10.income_group, [df_copy10.platform, df_copy10.interests]), cmap='Blues')
plt.show()
```



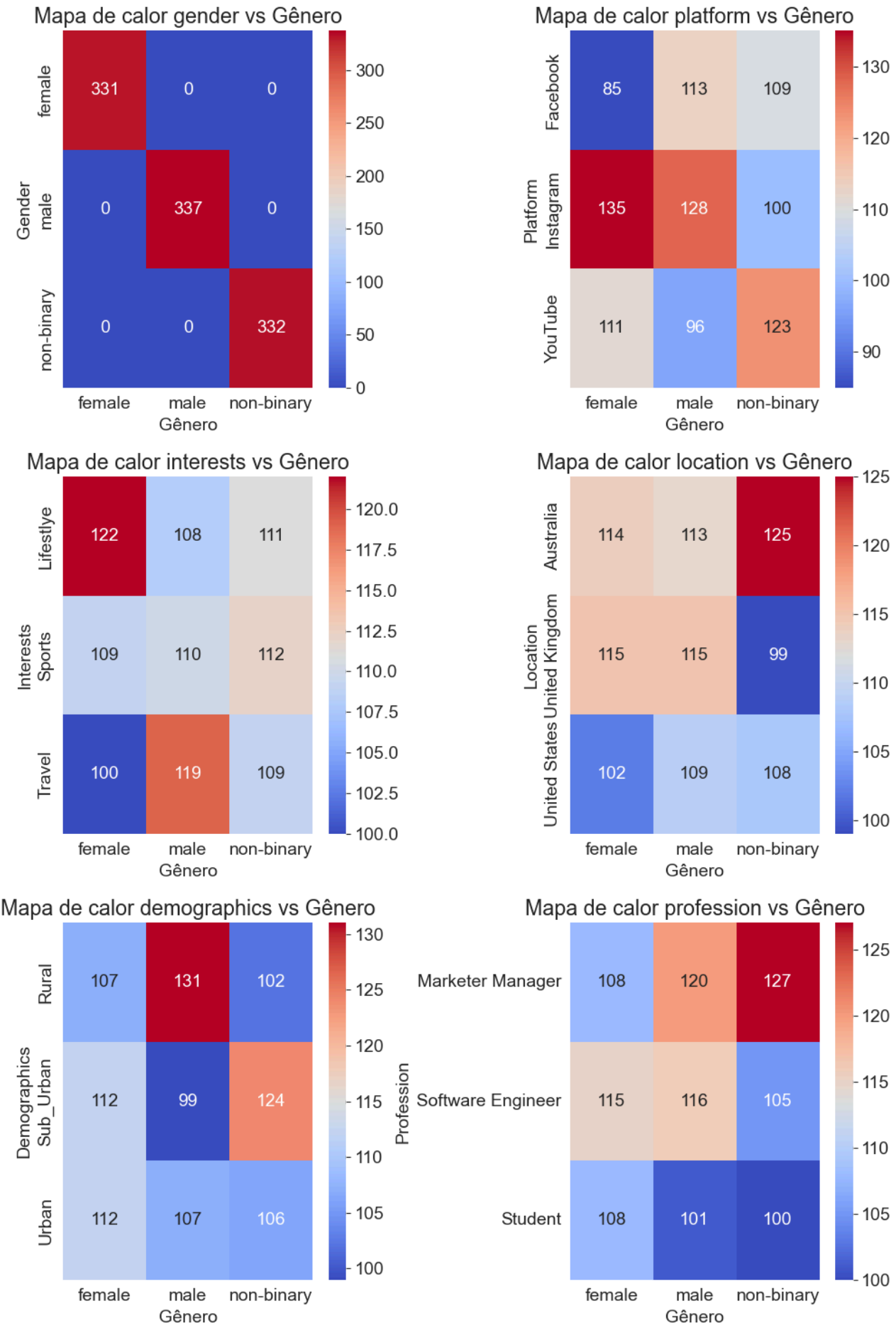
In [84]:
```
# O grupo de renda mais alta mostra interesse significativo em viagens em todas as plataformas, mas principalme
#      no YouTube, que é de 11,95% de 34,30%, que é do grupo de renda mais alta do total.
# O grupo de renda média tem um interesse equilibrado em estilo de vida e viagens em todas as plataformas, com
#      inclinação um pouco maior para esportes no Instagram, que é de 13,88% de 31,70%, que é do grupo de renda
#      do total.
# O grupo de baixa renda está mais interessado em esportes no Facebook e estilo de vida no YouTube.
# Os dados mostram que o uso das mídias sociais é direcionado para indivíduos de renda mais alta. Isto sugere
#      acesso à tecnologia e aos serviços de Internet pode ser mais prevalente entre aqueles com rendimentos ma
# Indivíduos com rendimentos mais elevados têm menos probabilidades de estar endividados e mais propensos a po
#      e carros. Isto indica uma maior estabilidade financeira deste grupo, permitindo-lhes participar em ativ
#      aquisição de casa própria e viagens, que se refletem nos seus interesses nas redes sociais.
# Há uma distinção clara nas preferências de plataforma entre grupos de renda. Indivíduos de renda mais alta p
#      o Instagram, que está associado a conteúdo visual e status social potencialmente mais elevado. Os indiv
#      baixa renda, por outro lado, preferem o Facebook, provavelmente devido à sua ampla acessibilidade e à d
#      de ofertas de conteúdo.
# As preferências de conteúdo variam entre grupos de renda. Indivíduos de renda mais alta demonstram um forte
#      em conteúdo de viagens, enquanto indivíduos de renda média demonstram um interesse equilibrado em vário
#      como esportes, viagens e estilo de vida. Indivíduos de baixa renda tendem a se envolver mais com conteú
#      relacionado a esportes no Facebook e com conteúdo de estilo de vida no YouTube.
# Análise e previsão geral
# Preferências demográficas por plataforma: Dada a distribuição demográfica por plataforma e localização, poder
#      que o Instagram continuará a atrair um público diversificado, com tendência para utilizadores do sexo f
#      no Reino Unido e utilizadores não binários nos Estados Unidos. A base de usuários do Facebook pode cont
#      dominada por homens no Reino Unido, enquanto o YouTube pode ter uma proporção maior de usuários não bin
#      Estados Unidos.
# Interesse em estilo de vida, esportes e viagens: o conteúdo de estilo de vida provavelmente permanecerá popu
#      todas as plataformas, com uma ligeira preferência por estilo de vida no Instagram. O conteúdo esportivo
```

```
#       ser altamente engajado, especialmente no Facebook e no Instagram. O conteúdo de viagens poderá registar
#       sustentado, especialmente entre os grupos etários mais velhos com níveis de rendimento mais baixos.
# Preferências baseadas na profissão: os gerentes de marketing provavelmente continuarão mostrando preferência
#       Instagram e YouTube, indicando um forte interesse em conteúdo visual e marketing de vídeo. Os engenheir
#       software podem continuar interessados em conteúdo de estilo de vida e esportes, principalmente no Insta
#       Facebook. Espera-se que os alunos mantenham um interesse equilibrado em todas as plataformas, sendo o Y
#       plataforma preferida para conteúdo de estilo de vida.
# Domínio urbano no uso das mídias sociais: As áreas urbanas continuarão a dominar o uso das mídias sociais em
#       plataformas, exceto o YouTube nos Estados Unidos, onde as áreas suburbanas apresentam maior uso. Isto s
#       factores como a acessibilidade à Internet e as preferências de estilo de vida desempenham um papel sign
#       no envolvimento nas redes sociais.
# Preferências de plataforma baseadas em renda: Indivíduos de renda mais alta provavelmente continuarão favore
#       Instagram, enquanto indivíduos de renda mais baixa podem preferir o Facebook por sua acessibilidade e d
#       ofertas de conteúdo. As preferências de conteúdo irão variar em conformidade, com os indivíduos com ren
#       mais elevados a demonstrarem um maior interesse em conteúdos de viagens, enquanto os indivíduos com ren
#       médios se envolvem com uma combinação equilibrada de conteúdos sobre desporto, viagens e estilo de vida
```

In [85]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy11 = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índic
```

In [86]:
```
# colunas categóricas
categorical_columns=["gender","platform","interests","location","demographics","profession"]
```

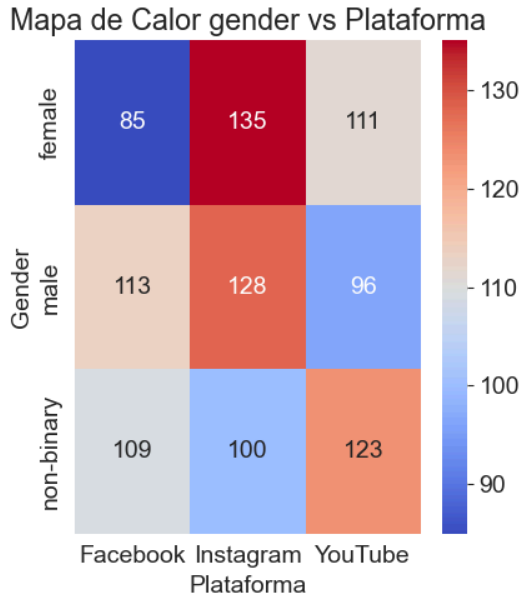
In [87]:
```
# mapas de calor com relação ao gênero
ig, axs = plt.subplots(3, 2, figsize=(10,15))

for i, col in enumerate(categorical_columns):
    cross_tab = pd.crosstab(df_copy11[col], df_copy11["gender"])
    sns.heatmap(cross_tab, ax=axs[i // 2, i % 2], cmap='coolwarm', annot=True, fmt='d')

    axs[i // 2, i % 2].set_title(f'Mapa de calor {col} vs Gênero')
    axs[i // 2, i % 2].set_xlabel('Gênero')
    axs[i // 2, i % 2].set_ylabel(col.capitalize())

plt.tight_layout()
plt.show()
```

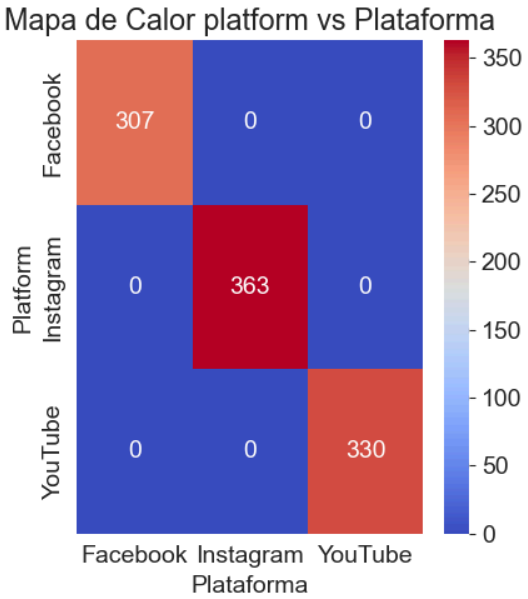


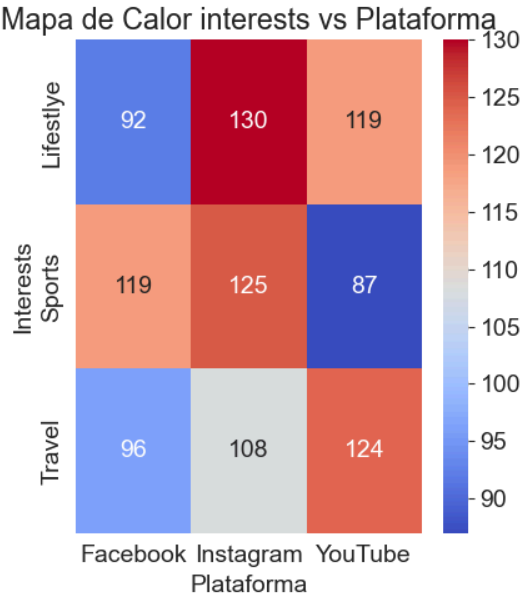
In [88]:

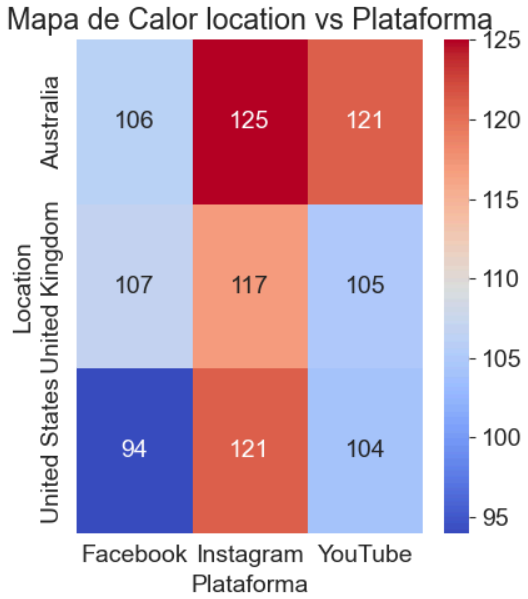
```
# mapas de calor com relação à plataforma
ig, axs = plt.subplots(3, 2, figsize=(10, 15))

for i, col in enumerate(categorical_columns):
    cross_tab = pd.crosstab(df_copy11[col], df_copy11["platform"])
    sns.heatmap(cross_tab, ax=axs[i // 2, i % 2], cmap='coolwarm', annot=True, fmt='d')

    axs[i // 2, i % 2].set_title(f'Mapa de Calor {col} vs Plataforma')
    axs[i // 2, i % 2].set_xlabel('Plataforma')
    axs[i // 2, i % 2].set_ylabel(col.capitalize())
```

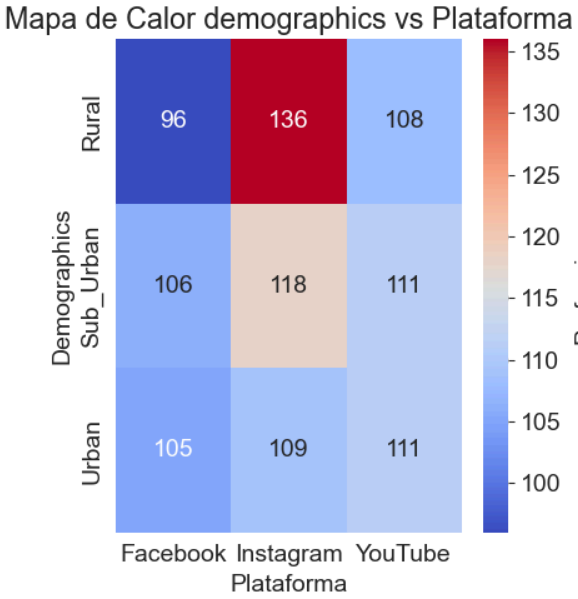
```
plt.tight_layout()
plt.show()
```

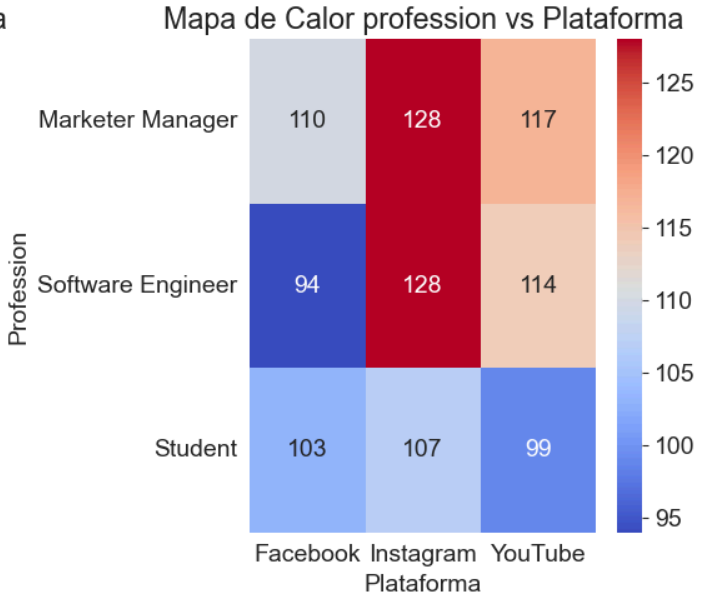








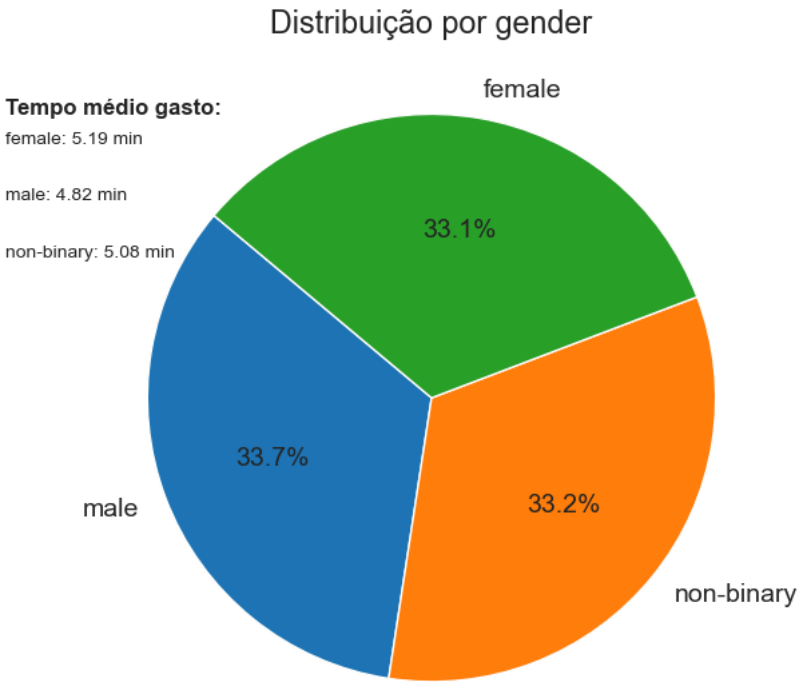


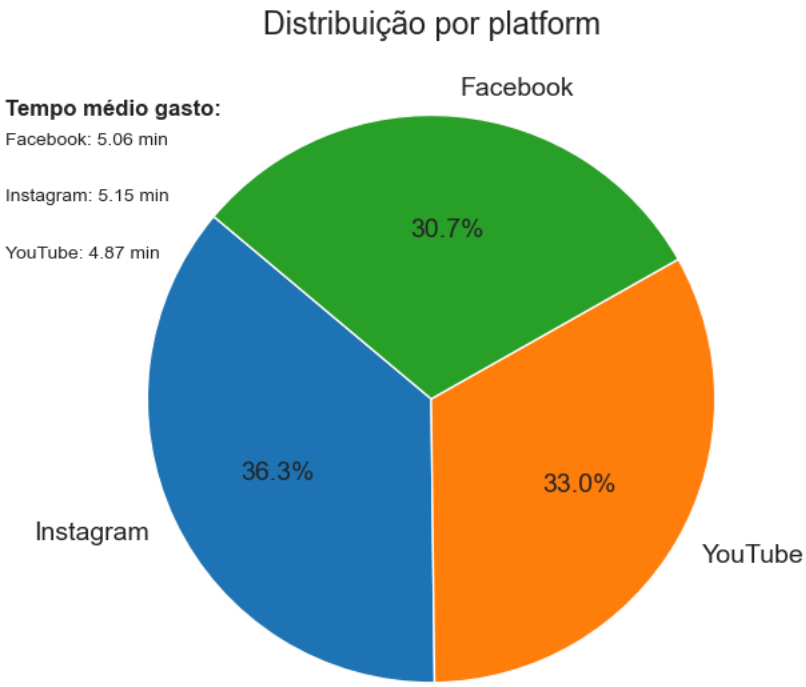


In [89]:

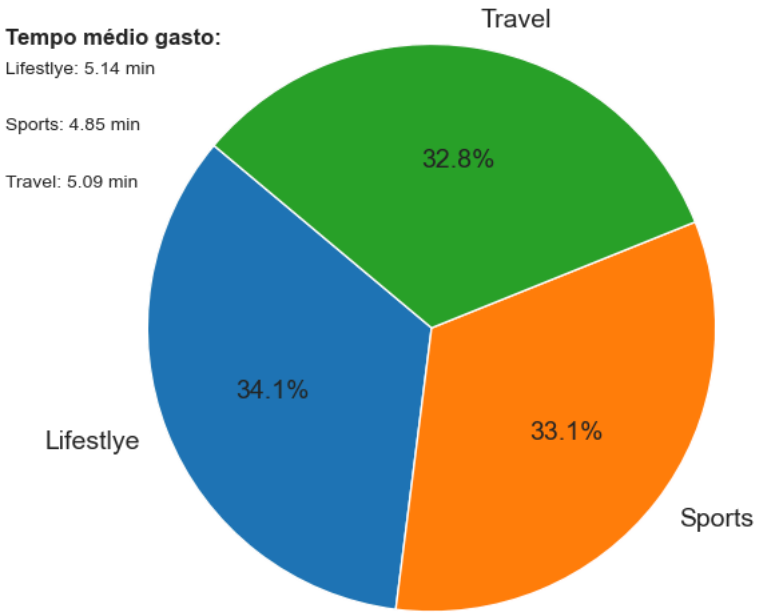
```
# distribuição de tempomédio gasto por categorias
categorical_features = df_copy11.select_dtypes(include=['object', 'bool'])

for column in categorical_features:
    plt.figure(figsize=(8, 6))
    avg_time_spent = df_copy11.groupby(column)['time_spent'].mean()
    df_copy11[column].value_counts().plot(kind='pie', autopct='%1.1f%%', startangle=140)
    plt.text(-1.5, 1, 'Tempo médio gasto:', fontsize=12, fontweight='bold')
    for i, (index, value) in enumerate(avg_time_spent.items()):
        plt.text(-1.5, 0.9-i*0.2, f"{index}: {value:.2f} min", fontsize=10)
```

```
    plt.title('Distribuição por {}'.format(column))
    plt.ylabel('')
    plt.show()
```

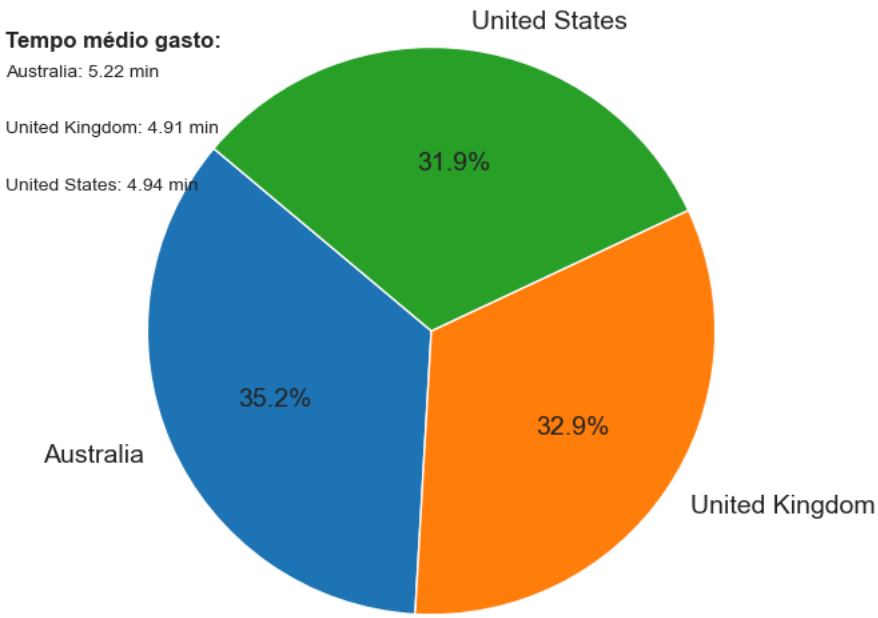




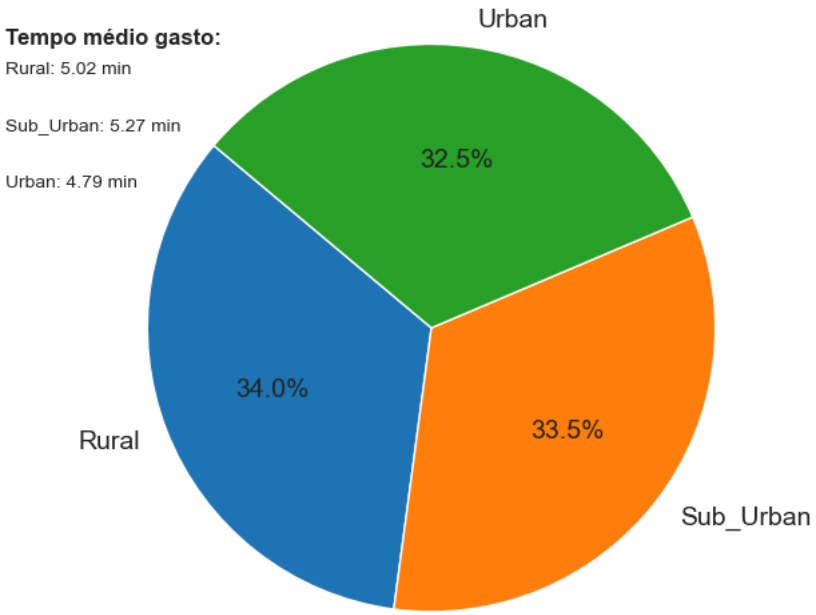
Distribuição por interests



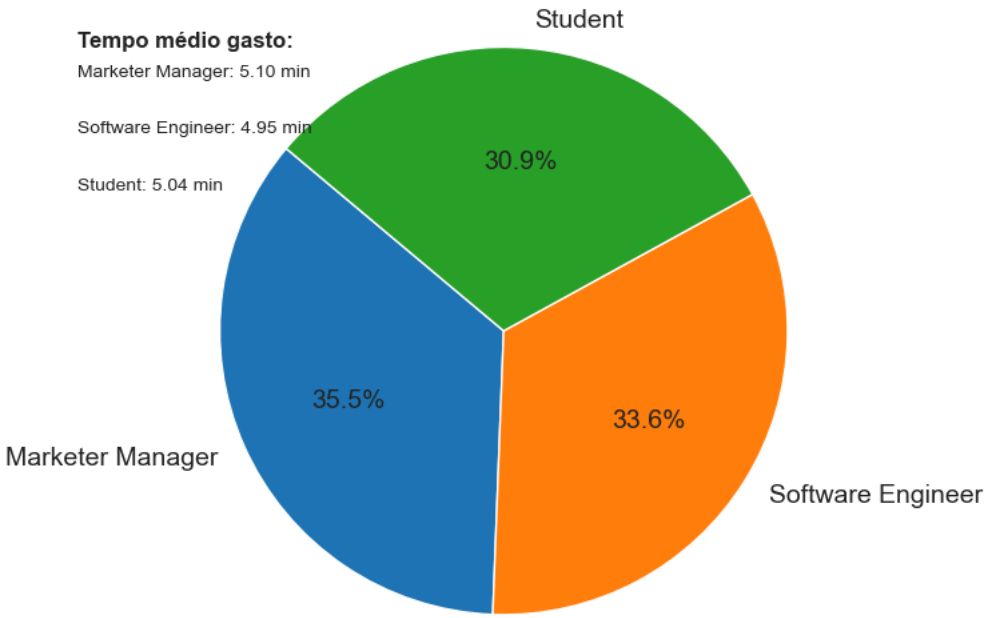
Distribuição por location



Distribuição por demographics



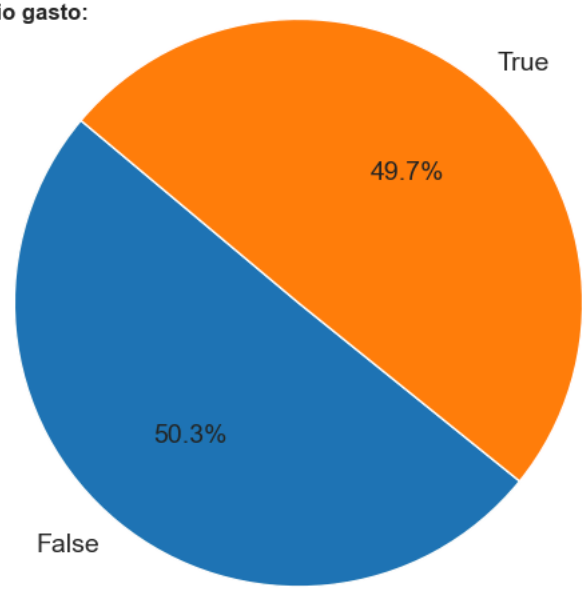
Distribuição por profession



Distribuição por indebt

Tempo médio gasto:
False: 5.00 min

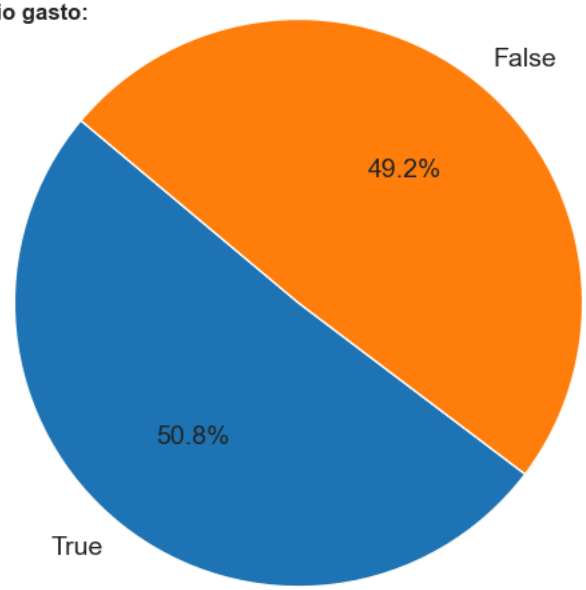
True: 5.06 min

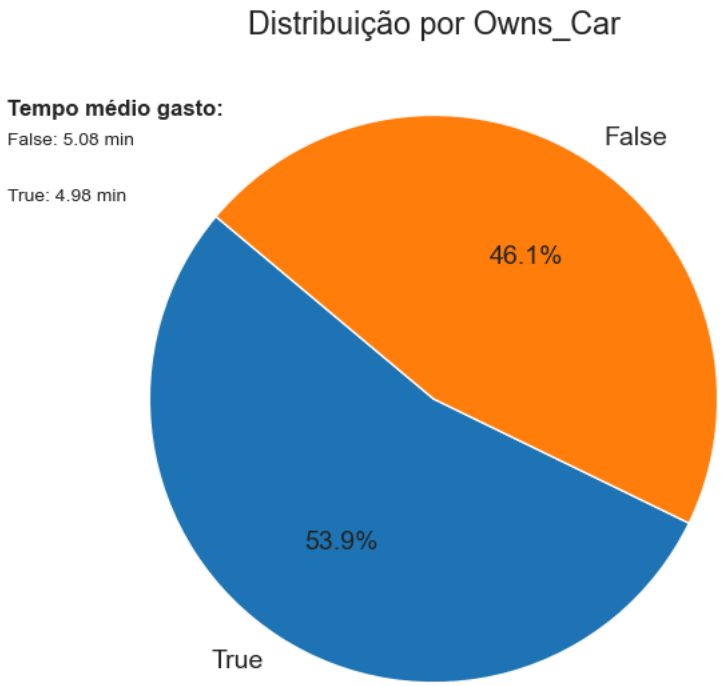


Distribuição por isHomeOwner

Tempo médio gasto:
False: 4.95 min

True: 5.10 min





In [90]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy12 = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índic
```

In [91]:
```
# modelo de classificação por RandonForest
y = df_copy12["age"]

features = ['isHomeOwner', 'Owns_Car',"indebt",'income']
X = pd.get_dummies(df_copy12[features])

# divisão de treinamento/validação
from sklearn.model_selection import train_test_split

train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# treinamento
test_model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=1)
test_model.fit(train_X, train_y)
predictions = test_model.predict(train_X)

# geração de previsão inicial
print("First in-sample predictions:", test_model.predict(X.head()))
print("Actual target values for those homes:", y.head().tolist())

# validação
val_predictions = test_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE: {:,.0f}".format(val_mae))

# função get_mae
def get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y):
    model = RandomForestClassifier(n_estimators=100, max_leaf_nodes=max_leaf_nodes, random_state=1)
    model.fit(train_X, train_y)
    preds_val = model.predict(val_X)
    mae = mean_absolute_error(val_y, preds_val)
    return(mae)

# árvore de candidatos
candidate_max_leaf_nodes = range(2,100,2)

for max_leaf_nodes in candidate_max_leaf_nodes :
    my_mae = get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y)
    print("Max leaf nodes: %d  \t\t Mean Absolute Error:  %d" %(max_leaf_nodes, my_mae))

# Armazena o melhor valor (5, 25, 50, 100, 250 ou 500)
scores = {leaf_size: get_mae(leaf_size, train_X, val_X, train_y, val_y) for leaf_size in candidate_max_leaf_no
best_tree_size = min(scores, key=scores.get)

# Cálculo do Modelo Final
final_model = RandomForestClassifier(max_leaf_nodes = best_tree_size, random_state=1)
```

```
final_model.fit(X, y)
print("First in-sample predictions:", final_model.predict(X.head()))
print("Actual target values for those homes:", y.head().tolist())
print(final_model.predict(X))
print(y)
```

```
First in-sample predictions: [56 46 23 29 33]
Actual target values for those homes: [56, 46, 32, 60, 25]
Validation MAE: 15
Max leaf nodes: 2               Mean Absolute Error:  14
Max leaf nodes: 4               Mean Absolute Error:  15
Max leaf nodes: 6               Mean Absolute Error:  16
Max leaf nodes: 8               Mean Absolute Error:  16
Max leaf nodes: 10              Mean Absolute Error:  15
Max leaf nodes: 12              Mean Absolute Error:  15
Max leaf nodes: 14              Mean Absolute Error:  15
Max leaf nodes: 16              Mean Absolute Error:  15
Max leaf nodes: 18              Mean Absolute Error:  15
Max leaf nodes: 20              Mean Absolute Error:  15
Max leaf nodes: 22              Mean Absolute Error:  15
Max leaf nodes: 24              Mean Absolute Error:  15
Max leaf nodes: 26              Mean Absolute Error:  15
Max leaf nodes: 28              Mean Absolute Error:  15
Max leaf nodes: 30              Mean Absolute Error:  15
Max leaf nodes: 32              Mean Absolute Error:  15
Max leaf nodes: 34              Mean Absolute Error:  15
Max leaf nodes: 36              Mean Absolute Error:  15
Max leaf nodes: 38              Mean Absolute Error:  15
Max leaf nodes: 40              Mean Absolute Error:  15
Max leaf nodes: 42              Mean Absolute Error:  15
Max leaf nodes: 44              Mean Absolute Error:  15
Max leaf nodes: 46              Mean Absolute Error:  15
Max leaf nodes: 48              Mean Absolute Error:  15
Max leaf nodes: 50              Mean Absolute Error:  15
Max leaf nodes: 52              Mean Absolute Error:  15
Max leaf nodes: 54              Mean Absolute Error:  15
Max leaf nodes: 56              Mean Absolute Error:  15
Max leaf nodes: 58              Mean Absolute Error:  15
Max leaf nodes: 60              Mean Absolute Error:  15
Max leaf nodes: 62              Mean Absolute Error:  15
Max leaf nodes: 64              Mean Absolute Error:  15
Max leaf nodes: 66              Mean Absolute Error:  15
Max leaf nodes: 68              Mean Absolute Error:  15
Max leaf nodes: 70              Mean Absolute Error:  15
Max leaf nodes: 72              Mean Absolute Error:  15
Max leaf nodes: 74              Mean Absolute Error:  15
Max leaf nodes: 76              Mean Absolute Error:  15
Max leaf nodes: 78              Mean Absolute Error:  15
Max leaf nodes: 80              Mean Absolute Error:  15
Max leaf nodes: 82              Mean Absolute Error:  15
Max leaf nodes: 84              Mean Absolute Error:  15
Max leaf nodes: 86              Mean Absolute Error:  15
Max leaf nodes: 88              Mean Absolute Error:  15
Max leaf nodes: 90              Mean Absolute Error:  15
Max leaf nodes: 92              Mean Absolute Error:  15
Max leaf nodes: 94              Mean Absolute Error:  15
Max leaf nodes: 96              Mean Absolute Error:  15
Max leaf nodes: 98              Mean Absolute Error:  15
First in-sample predictions: [43 50 43 54 54]
Actual target values for those homes: [56, 46, 32, 60, 25]
[43 50 43 54 54 43 50 43 54 43 50 43 45 54 43 43 50 54 43 43 54 43 50 50
 43 43 43 43 43 54 54 43 54 43 50 43 43 50 43 43 43 43 50 54 54 43 43 50
 43 43 50 43 43 43 54 43 54 43 43 54 43 43 54 54 41 54 43 54 43 43 43 43
 50 50 22 43 43 43 43 43 43 50 54 50 54 43 43 43 43 50 18 43 54 41 43 22
 54 41 43 50 50 43 43 43 54 43 54 50 43 43 43 54 43 43 54 43 54 50 43 54
 43 54 43 54 50 54 43 54 43 45 43 43 40 43 43 43 54 43 43 54 43 54 54 50
 41 43 43 50 54 43 43 54 45 43 43 43 54 43 43 43 54 43 50 54 54 43 54 54
 43 43 18 54 54 54 54 43 43 43 43 43 50 54 54 43 43 43 50 50 43 54 43 54
 50 54 43 54 54 43 43 54 43 54 43 54 50 54 43 50 54 43 50 54 50 50 43 43
 43 43 54 43 43 43 43 54 43 50 43 43 54 54 54 50 54 43 43 43 50 43 45 43
 50 43 54 54 54 54 43 45 43 43 54 43 43 43 50 43 50 54 43 43 43 43 43 43
 22 45 43 50 43 50 54 50 50 43 50 50 43 41 54 43 50 41 43 50 54 43 22 43
 50 43 43 54 50 50 43 43 50 45 43 43 50 50 50 54 43 50 50 43 50 50 43 43
 54 43 43 50 43 54 43 43 50 43 50 43 50 54 43 43 43 54 43 43 43 54 50 54
 43 43 43 50 54 43 50 43 43 43 43 43 43 50 54 43 54 54 22 50 50 43 50 43
 54 43 43 43 50 43 43 50 43 41 43 45 50 43 50 43 50 43 50 43 50 43 43 43
 43 50 43 43 43 50 43 43 43 43 50 54 50 43 54 40 22 43 54 54 54 41 43 43
 54 50 43 50 54 50 43 43 50 43 54 43 50 43 43 41 43 54 41 54 50 54 54 43
 50 50 43 43 43 43 41 43 54 43 43 43 50 43 50 50 43 22 43 50 50 43 54 54
 43 43 50 43 50 43 43 54 50 54 43 54 43 43 50 50 54 43 43 43 43 54 22 43
 43 43 54 54 22 43 54 43 54 43 41 43 43 43 50 45 50 43 50 43 50 54 54 43
 43 43 50 50 50 43 43 43 54 43 54 50 43 54 43 54 50 54 43 43 43 43 54 50
 43 43 43 43 43 50 54 43 43 43 43 54 18 43 54 54 50 43 50 18 50 18 54 43
 43 40 43 50 50 43 43 43 54 43 50 43 50 54 43 43 54 50 50 54 43 43 50 54
 54 43 43 54 43 54 45 43 43 43 43 22 54 54 50 50 43 54 54 43 43 43 50 54
 50 50 43 54 54 43 22 43 43 43 43 54 22 43 54 54 54 43 50 54 43 50 43 43
 43 43 43 54 43 54 43 43 43 22 43 43 43 50 50 50 43 43 43 40 43 50 54 41
 43 50 43 43 54 43 43 54 54 43 54 43 54 54 45 54 43 50 50 43 43 43 54 43
 50 54 45 43 50 43 43 43 43 54 54 43 50 43 43 50 43 54 54 54 50 43 43 50
 54 43 43 43 43 43 50 43 41 50 43 54 50 43 54 50 22 50 50 43 50 50 43 43
 43 43 43 43 54 43 54 43 43 43 40 43 50 43 54 43 43 43 54 43 43 43 43 43
 50 50 43 22 43 43 43 43 43 43 43 43 54 43 43 43 50 54 43 43 43 50 54
 50 43 50 43 43 43 43 41 43 43 43 43 43 43 54 43 43 43 43 43 50 43 43 50 43
 50 50 50 43 45 54 43 50 54 43 54 43 54 54 54 43 43 54 54 54 41 43 54
 43 43 54 43 54 43 43 54 54 43 50 43 50 54 54 54 43 43 50 43 43 50 43 43
 54 43 54 43 50 18 43 43 54 54 43 43 43 50 54 43 54 50 50 43 18 43 43 43
 43 43 43 54 50 43 43 43 54 43 22 43 54 54 43 50 50 43 54 43 43 50 43 54
 43 43 50 43 43 43 54 54 50 43 43 54 54 43 54 50 43 43 43 54 43 50 43 43
 22 43 43 54 43 18 54 43 54 54 54 43 43 54 54 43 54 54 43 43 43 43 43 50
 50 43 54 43 54 54 50 54 43 50 43 43 43 50 43 54 54 43 54 54 43 54 43 22 43
 54 50 54 43 50 54 43 43 50 50 43 54 54 43 43 54 50 43 43 43 18 43 54 43
 43 43 54 54 43 43 50 43 43 43 54 43 43 43 50 43]
0      56
1      46
2      32
3      60
4      25
       ..
995    22
996    40
997    27
998    61
999    19
Name: age, Length: 1000, dtype: int64
```

In [92]:
```
# criando uma cópia do dataframe para manter o backup do original
df_copy13 = df.copy(deep=True)  # deep=True (padrão) o novo objeto será criado com uma cópia dos dados e índic
```

In [93]:
```
# regressão linear

count = df_copy13['age'].value_counts()

# selecionando colunas categóricas
categorical_cols = ['gender', 'platform', 'interests', 'location', 'demographics', 'profession']

df_encoded = pd.get_dummies(df_copy13, columns=categorical_cols)

# Exibe as primeiras linhas do DataFrame codificado
df_encoded.head()

# separando as variáveis
X = df_encoded.drop(columns=['time_spent'])
y = df_encoded['time_spent']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# imprimindo os conjuntos de treinamento e teste
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)

# Inicializndo o modelo de regressão linear
model = LinearRegression()

# Ajustando o modelo aos dados de treinamento
model.fit(X_train, y_train)

# Prevendo os dados de teste
y_pred = model.predict(X_test)

# avaliando o modelo
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R-squared Score:", r2)

model1 = RandomForestRegressor()
model.fit(X_train, y_train)

# Fazendo previsões e avaliando o modelo
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
X_train shape: (750, 23)
y_train shape: (750,)
X_test shape: (250, 23)
y_test shape: (250,)
Mean Squared Error: 6.693470020701071
R-squared Score: -0.027188114719796763
Mean Squared Error: 6.693470020701071
```

In [94]:
```
# Gradient Boosting Regression
r2 =r2_score(y_test,y_pred)

# Inicializando o modelo Gradient Boosting Regression
gb_model = GradientBoostingRegressor(random_state=42)

# Ajustando o modelo aos dados de treinamento
gb_model.fit(X_train, y_train)

# Prevendo os dados de teste
gb_y_pred = gb_model.predict(X_test)

# avaliando o modelo
gb_mse = mean_squared_error(y_test, gb_y_pred)
gb_r2 = r2_score(y_test, gb_y_pred)

print("Gradient Boosting Regression:")
```

```
print("Mean Squared Error:", gb_mse)
print("R-squared Score:", gb_r2)
```

```
Gradient Boosting Regression:
Mean Squared Error: 6.934903547850391
R-squared Score: -0.06423879976293168
```

In [95]:
```
# SVR
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Inicializando o modelo de regressão do vetor de suporte
svr_model = SVR()

# Ajustando o modelo aos dados de treinamento
svr_model.fit(X_train_scaled, y_train)

# Prevendo os dados de teste
svr_y_pred = svr_model.predict(X_test_scaled)

# avaliando o modelo
svr_mse = mean_squared_error(y_test, svr_y_pred)
svr_r2 = r2_score(y_test, svr_y_pred)

print("Support Vector Regression:")
print("Mean Squared Error:", svr_mse)
print("R-squared Score:", svr_r2)
```

```
Support Vector Regression:
Mean Squared Error: 7.049447302973701
R-squared Score: -0.08181682484023156
```

In [96]:
```
# Crie recursos de interação
df_encoded['age_income_interaction'] = df_encoded['age'] * df_encoded['income']
df_encoded['age_time_spent_interaction'] = df_encoded['age'] * df_encoded['time_spent']
df_encoded['income_time_spent_interaction'] = df_encoded['income'] * df_encoded['time_spent']

df_encoded.head()
```

Out [96]:

	age	time_spent	income	indebt	isHomeOwner	Owns_Car	gender_female	gender_male	gender_non-binary	platform_Facebook	...	location_United States
0	56	3	19774	True	False	False	False	True	False	False	...	False
1	46	2	10564	True	True	True	True	False	False	True	...	False
2	32	8	13258	False	False	False	False	True	False	False	...	False
3	60	5	12500	False	True	False	False	False	True	False	...	False
4	25	1	14566	False	True	True	False	True	False	False	...	False

5 rows × 27 columns

In [99]:
```
# variável polinomial
poly_features = PolynomialFeatures(degree=2, include_bias=False)
poly_data = poly_features.fit_transform(X)
poly_feature_names = poly_features.get_feature_names_out(input_features=X.columns)

# criando dataframe com recursos polinomiais
df_poly = pd.DataFrame(poly_data, columns=poly_feature_names)
df_poly = pd.concat([df_encoded, df_poly], axis=1)
```

In [100]:
```
# Split the dataset into training and testing sets (assuming df_poly contains the polynomial features)
X_poly_train, X_poly_test, y_train, y_test = train_test_split(df_poly, y, test_size=0.2, random_state=42)

# Initialize and train the models
linear_model = LinearRegression()
linear_model.fit(X_poly_train, y_train)

rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_poly_train, y_train)

gb_model = GradientBoostingRegressor(random_state=42)
gb_model.fit(X_poly_train, y_train)

svr_model = SVR()
svr_model.fit(X_poly_train, y_train)

# Evaluate the models
models = {
    "Linear Regression": linear_model,
```

```
    "Random Forest Regression": rf_model,
    "Gradient Boosting Regression": gb_model,
    "Support Vector Regression": svr_model
}

for name, model in models.items():
    y_pred = model.predict(X_poly_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"{name}:")
    print(f"Mean Squared Error: {mse}")
    print(f"R-squared Score: {r2}")
    print()
```

```
Linear Regression:
Mean Squared Error: 3.0512002799783678e-15
R-squared Score: 0.9999999999999996

Random Forest Regression:
Mean Squared Error: 2.700000000000005e-05
R-squared Score: 0.999995725666974

Gradient Boosting Regression:
Mean Squared Error: 5.345220802341236e-09
R-squared Score: 0.9999999991538054

Support Vector Regression:
Mean Squared Error: 6.3378968889271246
R-squared Score: -0.00334377731154345
```