In [14]:
```
import pandas as pd

divorce = pd.read_csv("divorce.csv", parse_dates= ["marriage_date"])
divorce.dtypes
```

Out[14]:
```
divorce_date                 object
dob_man                      object
education_man                object
income_man                  float64
dob_woman                    object
education_woman              object
income_woman                float64
marriage_date        datetime64[ns]
marriage_duration           float64
num_kids                    float64
dtype: object
```

In [15]:
```
Divorce = pd.read_csv("divorce.csv")
Divorce["marriage_date"] = pd.to_datetime(divorce["marriage_date"])
Divorce.dtypes
```

Out[15]:
```
divorce_date                 object
dob_man                      object
education_man                object
income_man                  float64
dob_woman                    object
education_woman              object
income_woman                float64
marriage_date        datetime64[ns]
marriage_duration           float64
num_kids                    float64
dtype: object
```

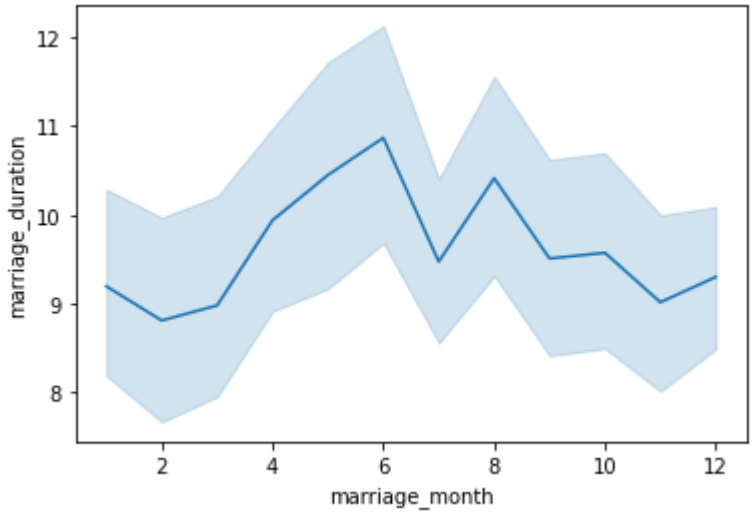
In [16]:
```
Divorce["marriage_month"] = Divorce["marriage_date"].dt.month
Divorce.head()
```

Out[16]:

	divorce_date	dob_man	education_man	income_man	dob_woman	education_woman	i
0	2006-09-06	1975-12-18	Secondary	2000.0	1983-08-01	Secondary	
1	2008-01-02	1976-11-17	Professional	6000.0	1977-03-13	Professional	
2	2011-01-02	1969-04-06	Preparatory	5000.0	1970-02-16	Professional	
3	2011-01-02	1979-11-13	Secondary	12000.0	1981-05-13	Secondary	
4	2011-01-02	1982-09-20	Professional	6000.0	1988-01-30	Professional	

In [ ]:

In [26]:
```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.lineplot(data= Divorce, x = "marriage_month", y = "marriage_duration")
plt.show()
```

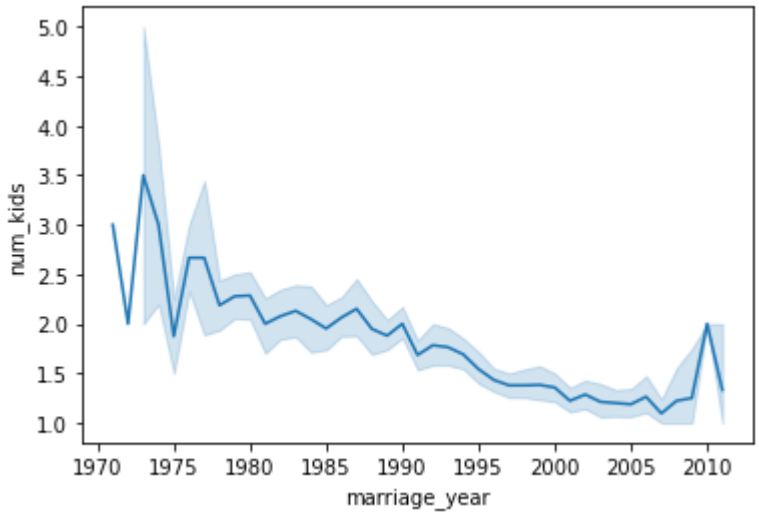


In [30]:
```
#Ex 1
divorce = pd.read_csv('divorce.csv', parse_dates=['divorce_date', 'dob_man
divorce.dtypes

#Marriage_date should be updated to the Date Time data type.
#It is not in the dataframe.
```

Out[30]:
```
divorce_date         datetime64[ns]
dob_man              datetime64[ns]
education_man                 object
income_man                   float64
dob_woman            datetime64[ns]
education_woman               object
income_woman                 float64
marriage_date                 object
marriage_duration            float64
num_kids                     float64
dtype: object
```

In [35]:

```
#Ex2
#Changing marriage_date column to DateTime value.
divorce["marriage_date"] = pd.to_datetime(divorce["marriage_date"])
divorce.dtypes

#define the marriage_year column
divorce["marriage_year"] = divorce["marriage_date"].dt.year

sns.lineplot(data= divorce, x= "marriage_year", y = "num_kids")
plt.show()
```



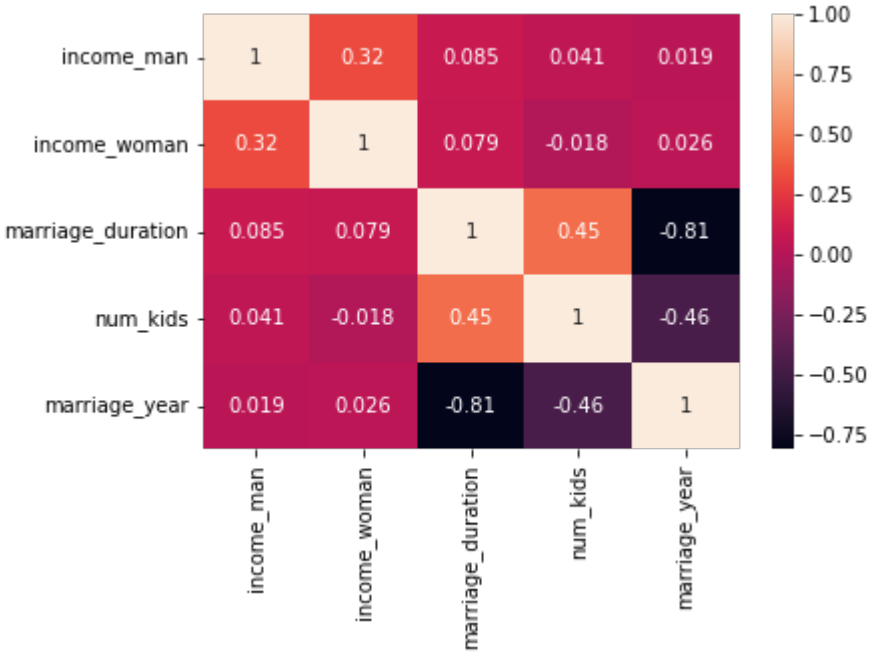
In [36]:

```
#Correlation
divorce.corr()
```

Out[36]:

	income_man	income_woman	marriage_duration	num_kids	marriage_yea
income_man	1.000000	0.318047	0.085321	0.040848	0.01917
income_woman	0.318047	1.000000	0.078677	-0.018015	0.02643
marriage_duration	0.085321	0.078677	1.000000	0.447358	-0.81246
num_kids	0.040848	-0.018015	0.447358	1.000000	-0.46149
marriage_year	0.019170	0.026433	-0.812469	-0.461495	1.00000

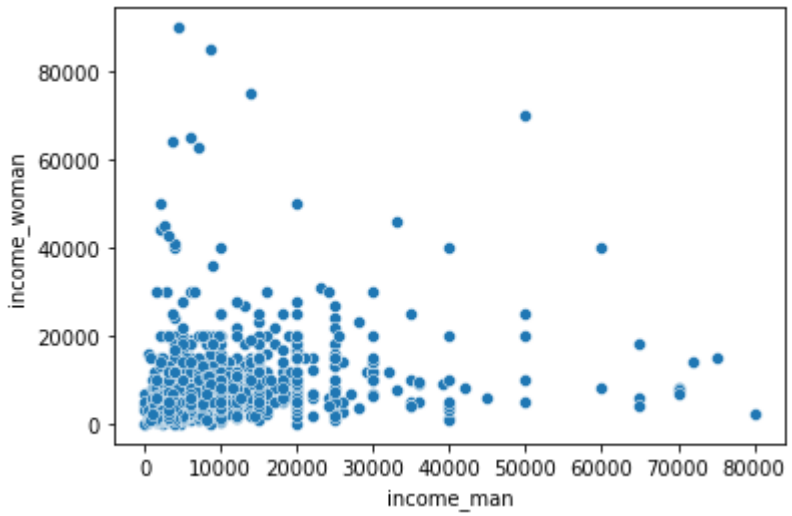
In [38]:
```
#Correlation heatmaps
sns.heatmap(divorce.corr(), annot =True)
plt.show()
```



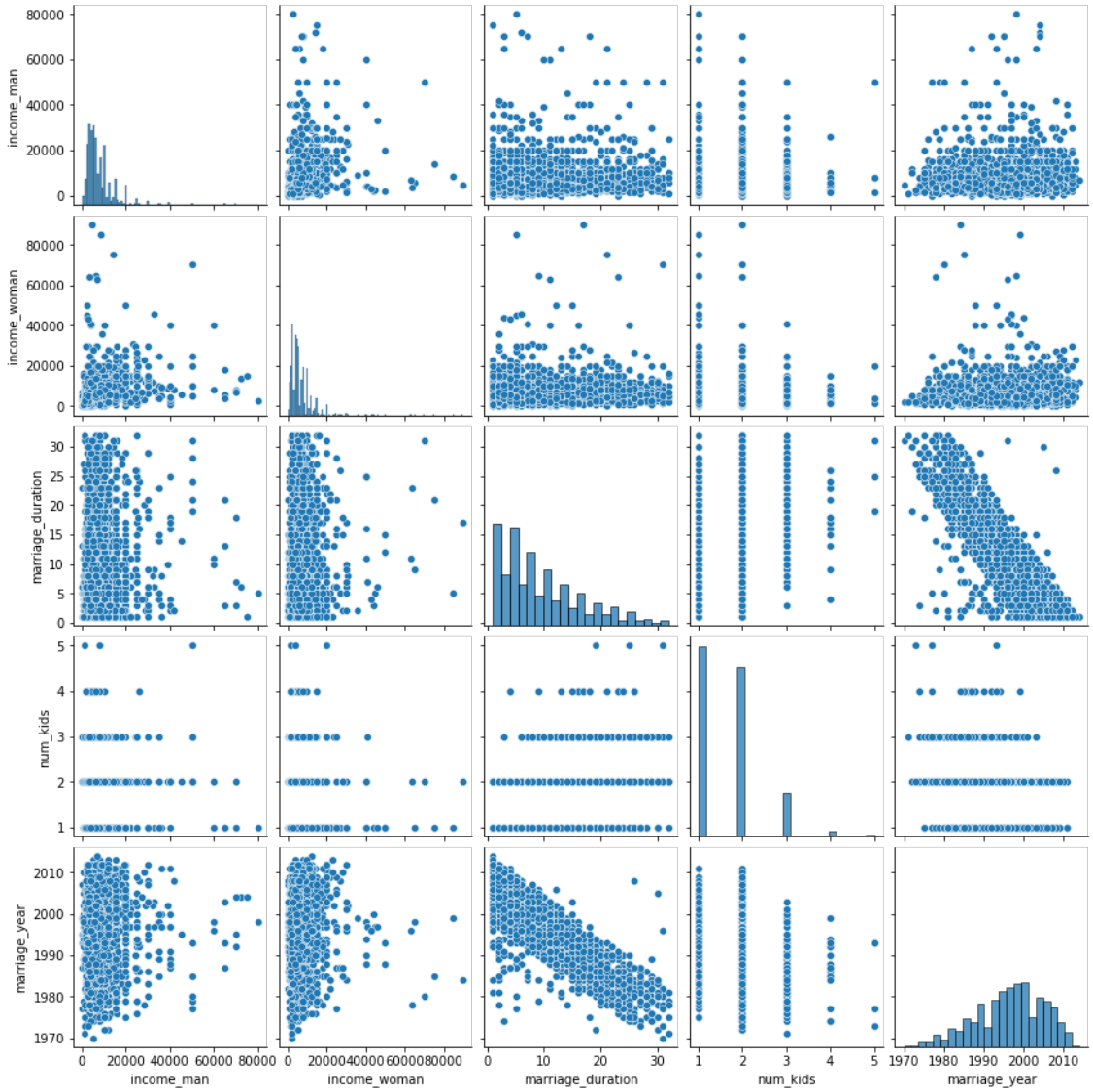
In [39]:
```
#Correlation in context
divorce["divorce_date"].min()

divorce["divorce_date"].max()
```

Out[39]: Timestamp('2015-11-03 00:00:00')

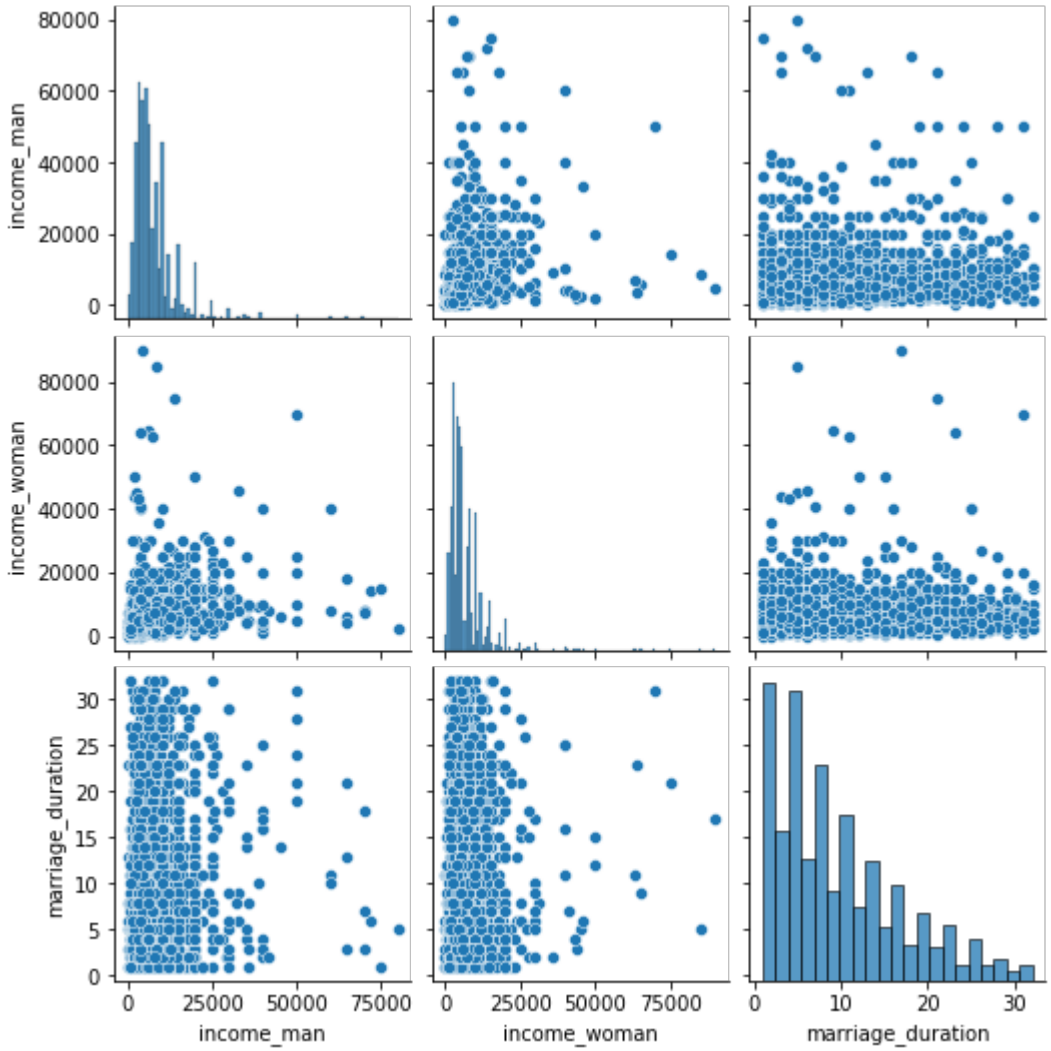
In [40]:
```
sns.scatterplot(data = divorce, x= "income_man", y = "income_woman")
plt.show()
```



In [41]:
```
sns.pairplot(data = divorce)
plt.show()
```



In [42]:
```
sns.pairplot(data= divorce, vars=["income_man","income_woman", "marriage_du
plt.show()
```



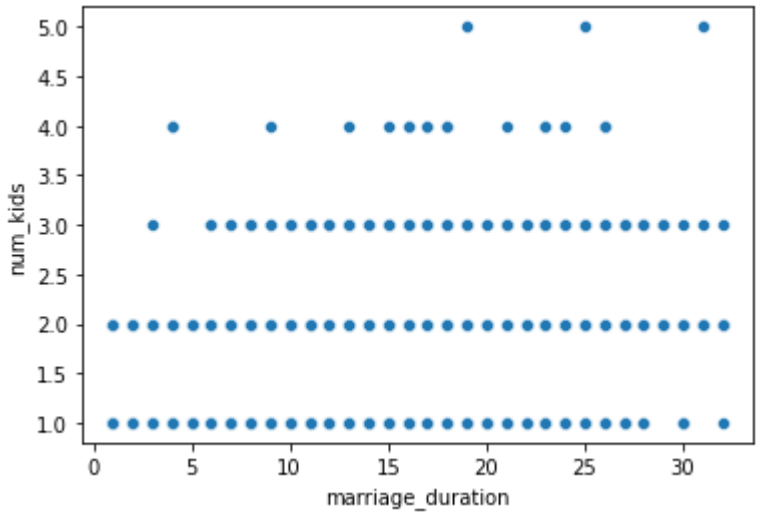
In [43]:
```
divorce.corr()
```

Out[43]:

	income_man	income_woman	marriage_duration	num_kids	marriage_yea
income_man	1.000000	0.318047	0.085321	0.040848	0.01917
income_woman	0.318047	1.000000	0.078677	-0.018015	0.02643
marriage_duration	0.085321	0.078677	1.000000	0.447358	-0.81246
num_kids	0.040848	-0.018015	0.447358	1.000000	-0.46149
marriage_year	0.019170	0.026433	-0.812469	-0.461495	1.00000

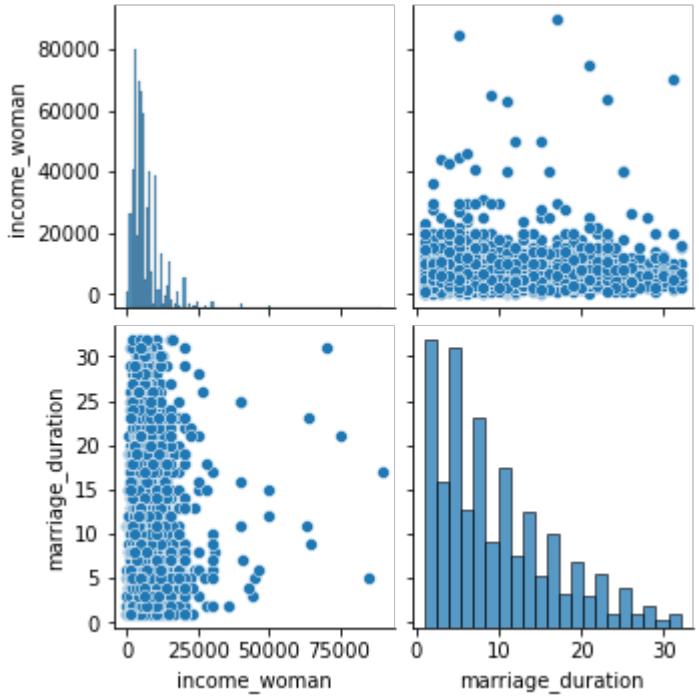
In [44]:

```
#Ex3

sns.scatterplot(data = divorce, x = "marriage_duration", y = "num_kids")
plt.show()
```



In [45]:

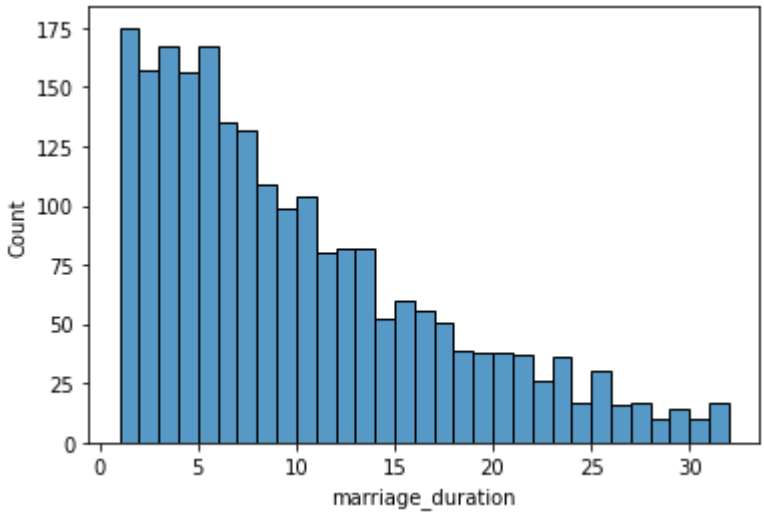
```
#Ex4

sns.pairplot(data = divorce, vars=["income_woman", "marriage_duration"])
plt.show()
```



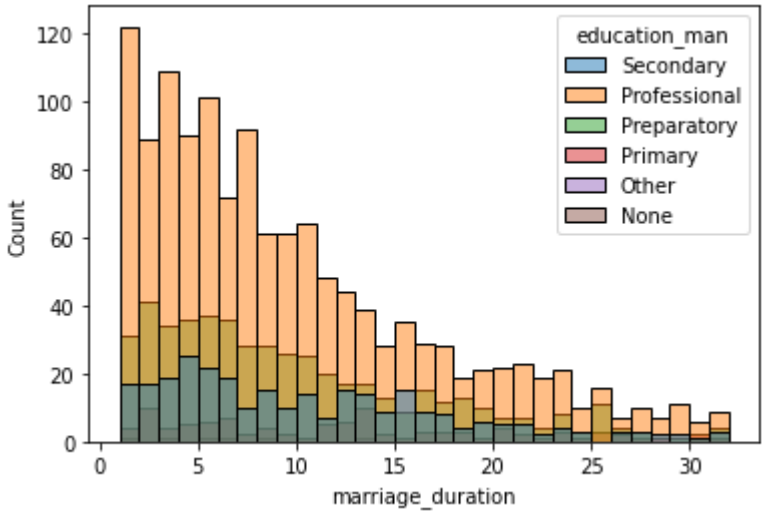
In [46]:
```
#Categorail Relationships

divorce["education_man"].value_counts()
```

Out[46]:
```
Professional    1313
Preparatory      501
Secondary        288
Primary          100
None               4
Other              3
Name: education_man, dtype: int64
```

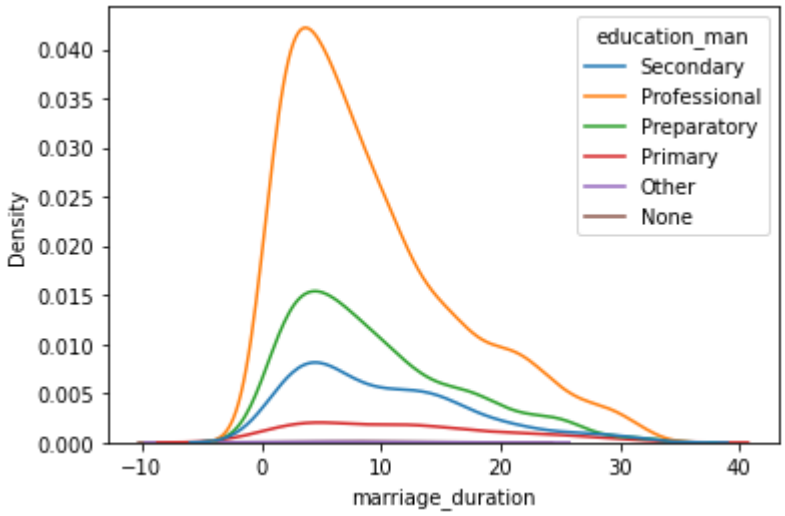
In [47]:
```
sns.histplot(data= divorce, x="marriage_duration", binwidth = 1)
plt.show()
```



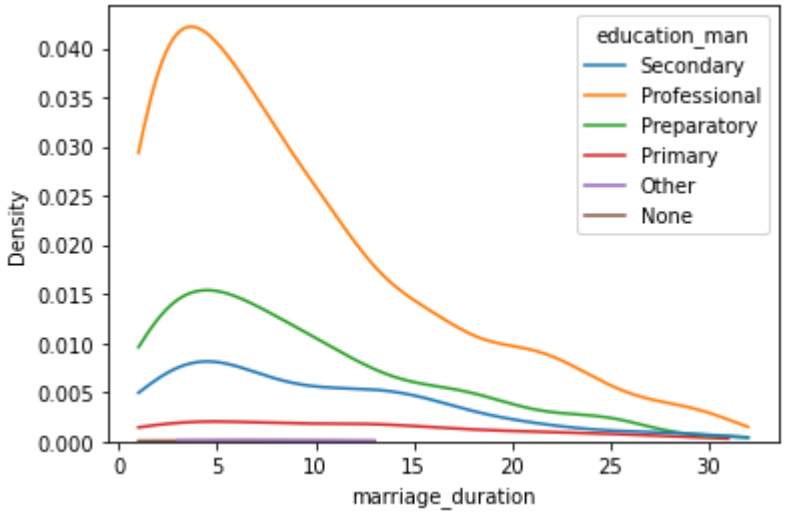
In [48]:
```
sns.histplot(data=divorce, x="marriage_duration", hue="education_man", binw
plt.show()
```



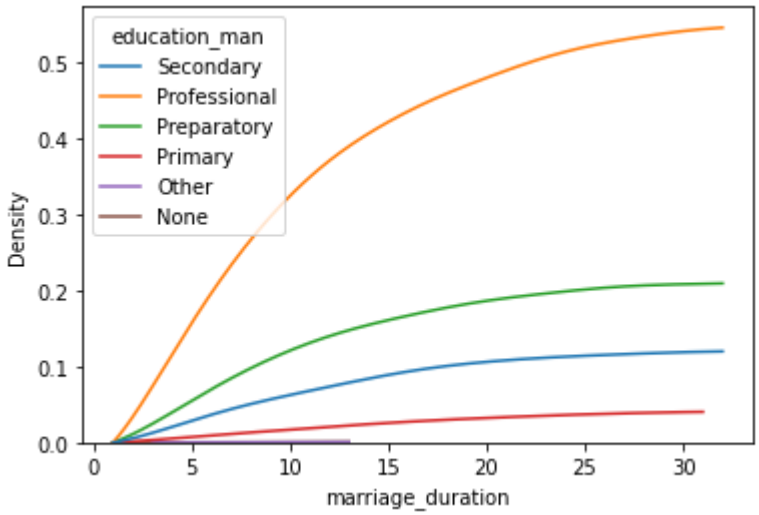
In [49]:
```
#Kernel Density Estimate (KDE) plots
sns.kdeplot(data = divorce, x="marriage_duration", hue="education_man")
plt.show()
```



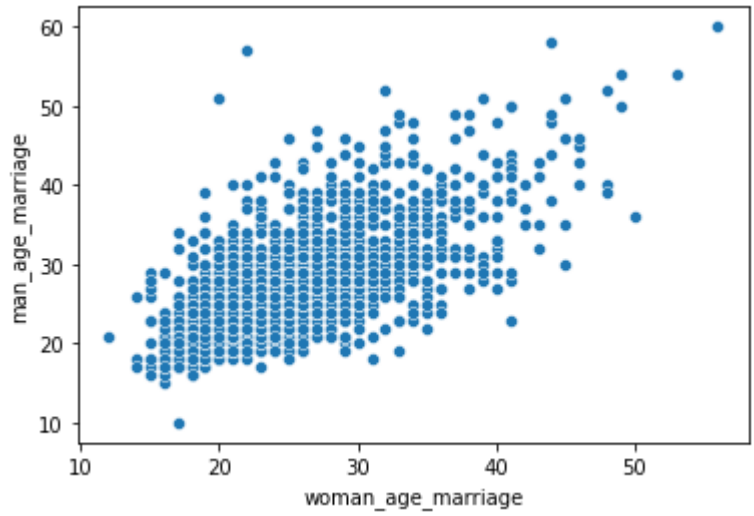
In [50]:
```
sns.kdeplot(data= divorce, x="marriage_duration", hue="education_man", cut
plt.show()
```



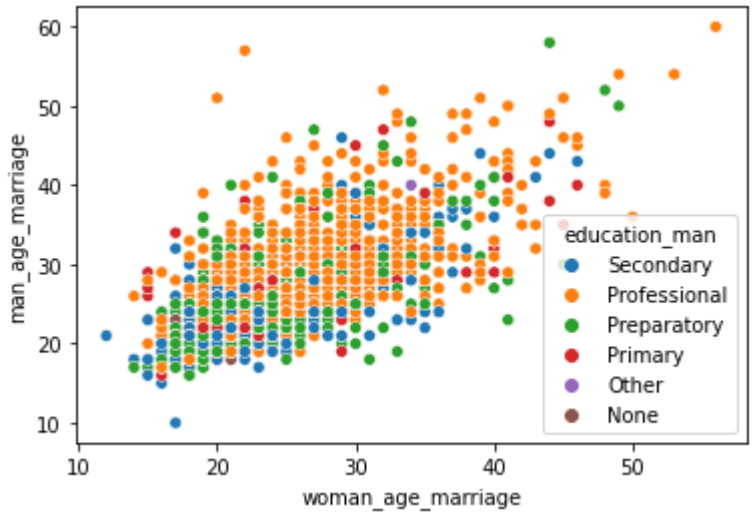
In [51]:
```
sns.kdeplot(data=divorce, x="marriage_duration", hue="education_man", cut=
plt.show()
```



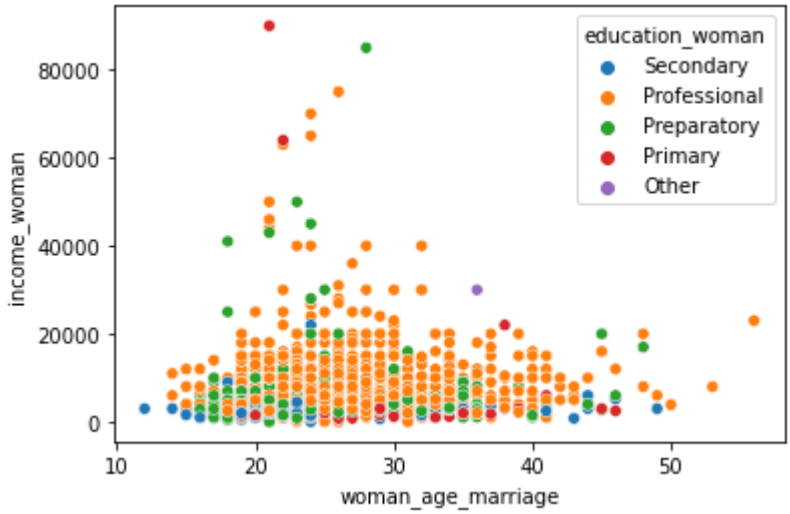
In [53]:
```
#Relationship between marriage age and education

divorce["man_age_marriage"] = divorce["marriage_year"] - divorce["dob_man"
divorce["woman_age_marriage"] = divorce["marriage_year"] - divorce["dob_wo

sns.scatterplot(data=divorce, x="woman_age_marriage", y="man_age_marriage"
plt.show()
```



In [54]:
```
sns.scatterplot(data=divorce, x="woman_age_marriage",y= "man_age_marriage"
plt.show()
```

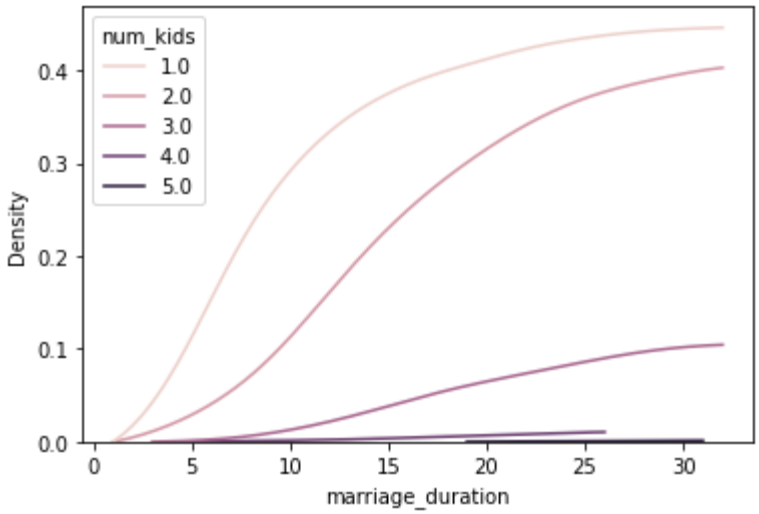


In [56]:
```
#Ex5
sns.scatterplot(data=divorce, x="woman_age_marriage" , y="income_woman", h
plt.show()
```



In [58]:
```
#Ex6
sns.kdeplot(data = divorce, x="marriage_duration", hue="num_kids", cut=0,
```

Out[58]:
```
<AxesSubplot:xlabel='marriage_duration', ylabel='Density'>
```



In [63]:
```
#Class Frequency
planes = pd.read_csv("Airlines_unclean.csv")
print(planes["Destination"].value_counts())
```

```
Cochin       4391
Banglore     2773
Delhi        1219
New Delhi     888
Hyderabad     673
Kolkata       369
Name: Destination, dtype: int64
```

In [64]:
```
planes["Destination"].value_counts(normalize=True)
```

Out[64]:
```
Cochin       0.425773
Banglore     0.268884
Delhi        0.118200
New Delhi    0.086105
Hyderabad    0.065257
Kolkata      0.035780
Name: Destination, dtype: float64
```

In [65]:
```
pd.crosstab(planes["Source"], planes["Destination"])
```

Out[65]:

Destination	Banglore	Cochin	Delhi	Hyderabad	Kolkata	New Delhi
Source						
Banglore	0	0	1199	0	0	868
Chennai	0	0	0	0	364	0
Delhi	0	4318	0	0	0	0
Kolkata	2720	0	0	0	0	0
Mumbai	0	0	0	662	0	0

In [66]:
```
pd.crosstab(planes["Source"], planes["Destination"],
            values=planes["Price"], aggfunc="median")
```

Out[66]:

Destination	Banglore	Cochin	Delhi	Hyderabad	Kolkata	New Delhi
Source						
Banglore	NaN	NaN	4823.0	NaN	NaN	10976.5
Chennai	NaN	NaN	NaN	NaN	3850.0	NaN
Delhi	NaN	10262.0	NaN	NaN	NaN	NaN
Kolkata	9345.0	NaN	NaN	NaN	NaN	NaN
Mumbai	NaN	NaN	NaN	3342.0	NaN	NaN

In [78]:
```
#Ex7
salaries = pd.read_csv("Salary_Rupee_USD.csv", index_col=0)
salaries["Job_Category"].value_counts(normalize=True)
```

Out[78]:
```
Data Science        0.277641
Data Engineering    0.272727
Data Analytics      0.226044
Machine Learning    0.120393
Other               0.068796
Managerial          0.034398
Name: Job_Category, dtype: float64
```

In [108]:
```
#Ex8
pd.crosstab(salaries["Company_Size"], salaries["Experience"])
```

Out[108]:

Experience	EN	EX	MI	SE
Company_Size				
L	24	7	49	44
M	25	9	58	136
S	18	1	21	15

In [109]:
```
#Ex8

pd.crosstab(salaries["Job_Category"], salaries["Company_Size"])
```

Out[109]:

Company_Size	L	M	S
Job_Category			
Data Analytics	23	61	8
Data Engineering	28	72	11
Data Science	38	59	16
Machine Learning	17	19	13
Managerial	5	8	1
Other	13	9	6

In [110]:
```
#Ex8

pd.crosstab(salaries["Job_Category"], salaries["Company_Size"],values=sala
```

Out[110]:

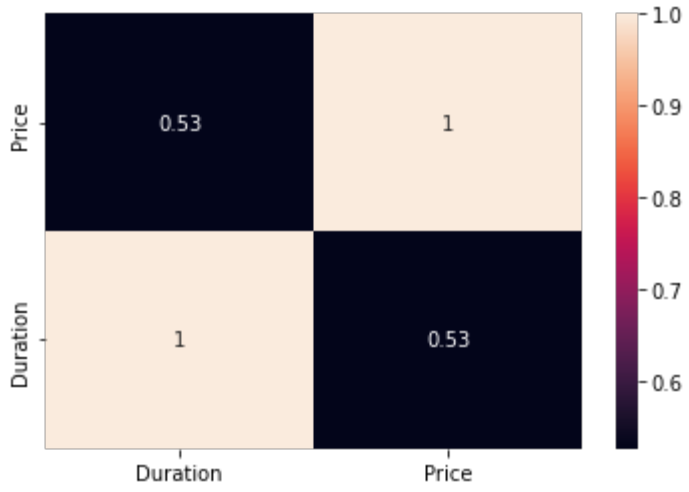
Company_Size	L	M	S
Job_Category			
Data Analytics	112851.749217	95912.685246	53741.877000
Data Engineering	118939.035000	121287.060500	86927.136000
Data Science	96489.520105	116044.455864	62241.749250
Machine Learning	140779.491529	100794.236842	78812.586462
Managerial	190551.448800	150713.628000	31484.700000
Other	92873.911385	89750.578667	69871.248000

In [88]:
```
sns.heatmap(planes.corr(), annot=True)
plt.show()
```

In [89]:

```
planes = pd.read_csv('Airlines_unclean.csv', index_col = 0,
parse_dates=['Date_of_Journey','Dep_Time','Arrival_Time'])
# Remove the string character
planes["Duration"] = planes["Duration"].str.replace("h", ".")
planes["Duration"] = planes["Duration"].str.replace("m", "")
planes["Duration"] = planes["Duration"].str.replace(" ", "")
# Convert to float data type
planes["Duration"] = planes["Duration"].astype(float)
print(planes.info())
ax = sns.heatmap(planes.corr(), annot=True)
ax.set_ylim([0,2])
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10660 entries, 0 to 10659
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Airline          10233 non-null  object
 1   Date_of_Journey  10338 non-null  datetime64[ns]
 2   Source           10473 non-null  object
 3   Destination      10313 non-null  object
 4   Route            10404 non-null  object
 5   Dep_Time         10400 non-null  datetime64[ns]
 6   Arrival_Time     10466 non-null  datetime64[ns]
 7   Duration         10446 non-null  float64
 8   Total_Stops      10448 non-null  object
 9   Additional_Info  10071 non-null  object
 10  Price            10044 non-null  float64
dtypes: datetime64[ns](3), float64(2), object(6)
memory usage: 999.4+ KB
None
```



In [90]:
```
print(planes["Total_Stops"].value_counts())
```

```
1 stop      5503
non-stop    3411
2 stops     1488
3 stops       45
4 stops        1
Name: Total_Stops, dtype: int64
```

In [97]:

```
planes["Total_Stops"] = planes["Total_Stops"].str.replace(" stops", "")
planes["Total_Stops"] = planes["Total_Stops"].str.replace(" stop", "")
planes["Total_Stops"] = planes["Total_Stops"].str.replace("non-stops", "0"
planes["Total_Stops"] = planes["Total_Stops"].astype(int)
sns.heatmap(planes.corr(), annot =True)
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [97], in <cell line: 4>()
      2 planes["Total_Stops"] = planes["Total_Stops"].str.replace(" stop", "")
      3 planes["Total_Stops"] = planes["Total_Stops"].str.replace("non-stops", "0")
----> 4 planes["Total_Stops"] = planes["Total_Stops"].astype(int)
      5 sns.heatmap(planes.corr(), annot =True)
      6 plt.show()

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:5912, in NDFrame.astype(self, dtype, copy, errors)
   5905     results = [
   5906         self.iloc[:, i].astype(dtype, copy=copy)
   5907         for i in range(len(self.columns))
   5908     ]
   5910 else:
   5911     # else, only a single dtype is given
-> 5912     new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   5913     return self._constructor(new_data).__finalize__(self, method="astype")
   5915 # GH 33113: handle empty frame or series

File ~\anaconda3\lib\site-packages\pandas\core\internals\managers.py:419, in BaseBlockManager.astype(self, dtype, copy, errors)
    418 def astype(self: T, dtype, copy: bool = False, errors: str = "raise") -> T:
--> 419     return self.apply("astype", dtype=dtype, copy=copy, errors=errors)

File ~\anaconda3\lib\site-packages\pandas\core\internals\managers.py:304, in BaseBlockManager.apply(self, f, align_keys, ignore_failures, **kwargs)
    302         applied = b.apply(f, **kwargs)
    303     else:
--> 304         applied = getattr(b, f)(**kwargs)
    305 except (TypeError, NotImplementedError):
    306     if not ignore_failures:

File ~\anaconda3\lib\site-packages\pandas\core\internals\blocks.py:580, in Block.astype(self, dtype, copy, errors)
    562 """
    563 Coerce to the new dtype.
    564
   (...)
    576 Block
```

```
    577 """
    578 values = self.values
--> 580 new_values = astype_array_safe(values, dtype, copy=copy, errors=e
rrors)
    582 new_values = maybe_coerce_values(new_values)
    583 newb = self.make_block(new_values)

File ~\anaconda3\lib\site-packages\pandas\core\dtypes\cast.py:1292, in as
type_array_safe(values, dtype, copy, errors)
   1289     dtype = dtype.numpy_dtype
   1291 try:
-> 1292     new_values = astype_array(values, dtype, copy=copy)
   1293 except (ValueError, TypeError):
   1294     # e.g. astype_nansafe can fail on object-dtype of strings
   1295     #  trying to convert to float
   1296     if errors == "ignore":

File ~\anaconda3\lib\site-packages\pandas\core\dtypes\cast.py:1237, in as
type_array(values, dtype, copy)
   1234     values = values.astype(dtype, copy=copy)
   1236 else:
-> 1237     values = astype_nansafe(values, dtype, copy=copy)
   1239 # in pandas we don't store numpy str dtypes, so convert to object
   1240 if isinstance(dtype, np.dtype) and issubclass(values.dtype.type,
str):

File ~\anaconda3\lib\site-packages\pandas\core\dtypes\cast.py:1154, in as
type_nansafe(arr, dtype, copy, skipna)
   1150 elif is_object_dtype(arr.dtype):
   1151
   1152     # work around NumPy brokenness, #1987
   1153     if np.issubdtype(dtype.type, np.integer):
-> 1154         return lib.astype_intsafe(arr, dtype)
   1156     # if we have a datetime/timedelta array of objects
   1157     # then coerce to a proper dtype and recall astype_nansafe
   1159     elif is_datetime64_dtype(dtype):

File ~\anaconda3\lib\site-packages\pandas\_libs\lib.pyx:668, in pandas._l
ibs.lib.astype_intsafe()

ValueError: invalid literal for int() with base 10: 'non-stop'
```

In [98]:
```
planes.dtypes
```

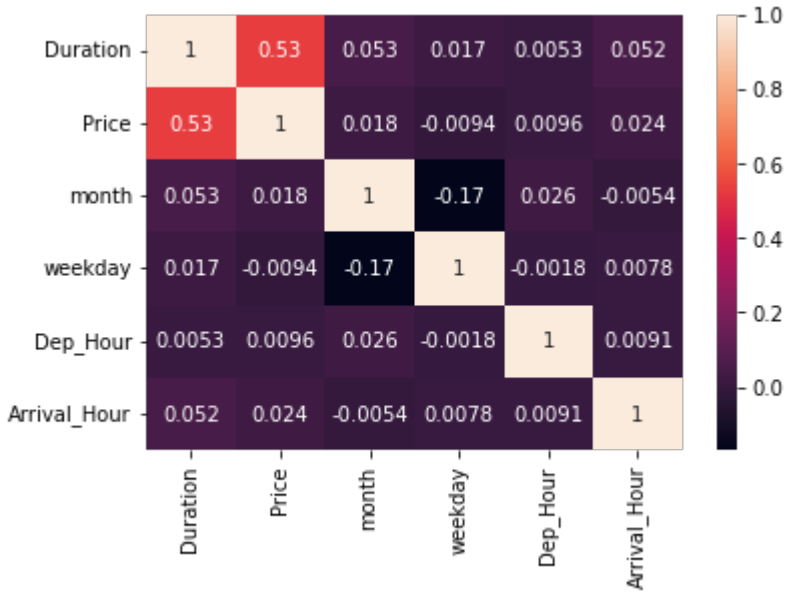
Out[98]:
```
Airline                     object
Date_of_Journey    datetime64[ns]
Source                      object
Destination                 object
Route                       object
Dep_Time           datetime64[ns]
Arrival_Time       datetime64[ns]
Duration                  float64
Total_Stops                 object
Additional_Info             object
Price                     float64
dtype: object
```

In [99]:
```
planes["month"] = planes["Date_of_Journey"].dt.month
planes["weekday"] = planes["Date_of_Journey"].dt.weekday
print(planes[["month", "weekday", "Date_of_Journey"]].head())
```

```
   month  weekday Date_of_Journey
0    9.0      4.0      2019-09-06
1   12.0      3.0      2019-12-05
2    1.0      3.0      2019-01-03
3    6.0      0.0      2019-06-24
4   12.0      1.0      2019-12-03
```

In [106]:
```
planes["Dep_Hour"] = planes["Dep_Time"].dt.hour
planes["Arrival_Hour"] = planes["Arrival_Time"].dt.hour
sns.heatmap(planes.corr(), annot =True)
```

Out[106]:
```
<AxesSubplot:>
```



In [107]:
```
print(planes["Price"].describe())
```

```
count    10044.000000
mean      9044.411191
std       4472.304869
min       1759.000000
25%       5276.750000
50%       8366.000000
75%      12373.000000
max      54826.000000
Name: Price, dtype: float64
```

In [111]:
```
twenty_fifth = planes["Price"].quantile(0.25)
median = planes["Price"].median()
seventy_fifth = planes["Price"].quantile(0.75)
maximum = planes["Price"].max()
```

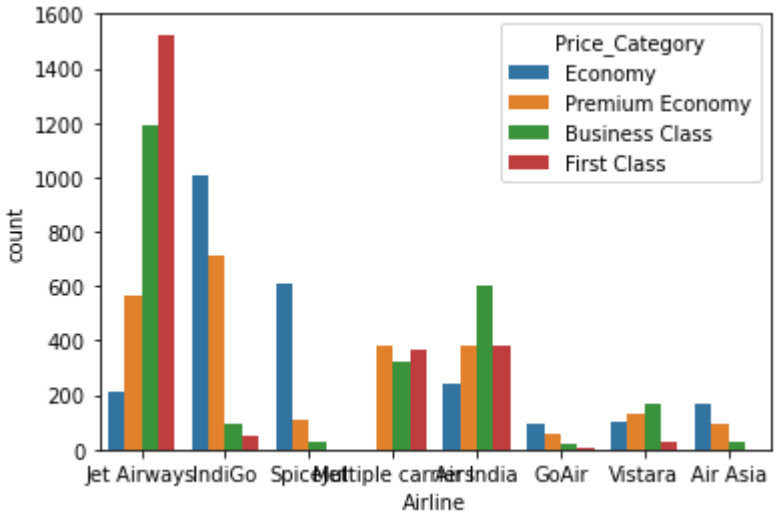
In [112]:
```
labels = ["Economy", "Premium Economy", "Business Class", "First Class"]
bins = [0, twenty_fifth, median, seventy_fifth, maximum]
```

In [113]:
```
planes["Price_Category"] = pd.cut(planes["Price"], labels=labels, bins=bin
```

In [114]:
```
print(planes[["Price", "Price_Category"]].head())
```

```
     Price   Price_Category
0  13882.0      First Class
1   6218.0  Premium Economy
2  13302.0      First Class
3   3873.0          Economy
4  11087.0   Business Class
```

In [115]:
```
sns.countplot(data = planes, x= "Airline", hue = "Price_Category")
plt.show()
```



In [122]:

```
#Ex9
salaries = pd.read_csv("Salaries_with_date_of_response.csv", index_col=0, 



salaries["month"] = salaries["date_of_response"].dt.month



salaries["weekday"] = salaries["date_of_response"].dt.weekday



correlation_matrix = salaries.corr()



plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", li
plt.title("Correlation Coefficients Heatmap")
plt.show()
```

```
C:\Users\Lut Lat Aung\anaconda3\lib\site-packages\pandas\io\parsers\base_
parser.py:1070: UserWarning: Parsing '21/11/2020' in DD/MM/YYYY format. P
rovide format or specify infer_datetime_format=True for consistent parsin
g.
  return tools.to_datetime(
C:\Users\Lut Lat Aung\anaconda3\lib\site-packages\pandas\io\parsers\base_
parser.py:1070: UserWarning: Parsing '29/11/2020' in DD/MM/YYYY format. P
rovide format or specify infer_datetime_format=True for consistent parsin
g.
  return tools.to_datetime(
C:\Users\Lut Lat Aung\anaconda3\lib\site-packages\pandas\io\parsers\base_
parser.py:1070: UserWarning: Parsing '13/10/2020' in DD/MM/YYYY format. P
rovide format or specify infer_datetime_format=True for consistent parsin
g.
  return tools.to_datetime(
C:\Users\Lut Lat Aung\anaconda3\lib\site-packages\pandas\io\parsers\base_
parser.py:1070: UserWarning: Parsing '15/10/2020' in DD/MM/YYYY format. P
rovide format or specify infer_datetime_format=True for consistent parsin
g.
```

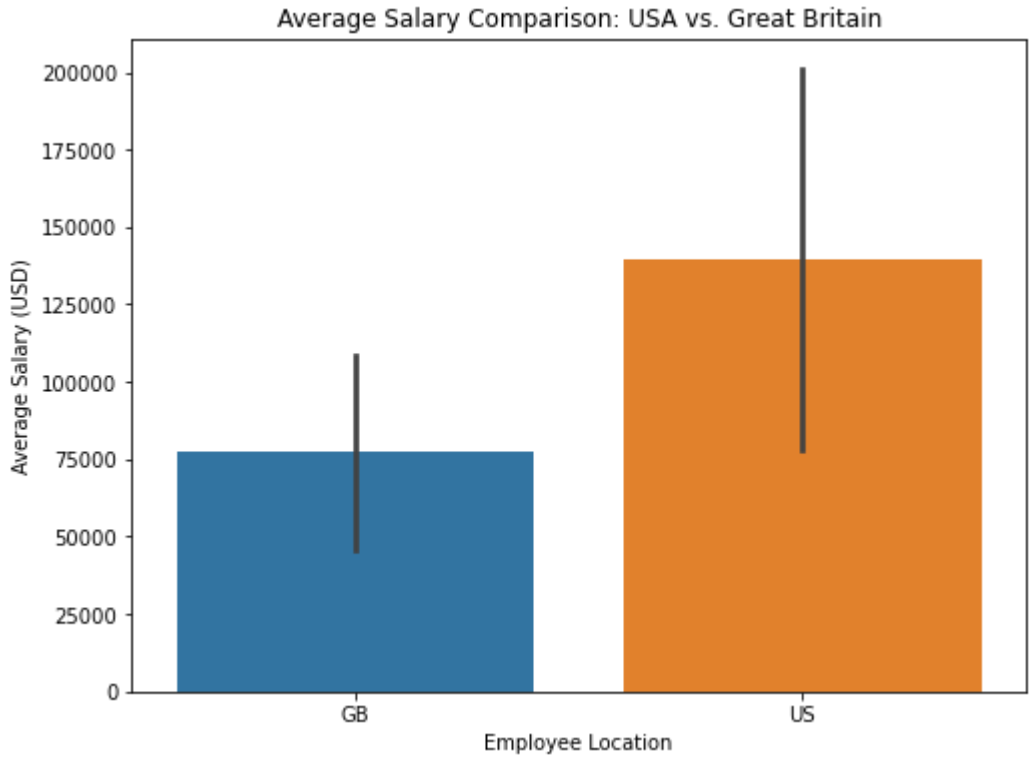
In [124]:

```
#Ex10

twenty_fifth = salaries["Salary_USD"].quantile(0.25)

salaries_median = salaries["Salary_USD"].median()

seventy_fifth = seventy_fifth = salaries["Salary_USD"].quantile(0.75)

print(twenty_fifth, salaries_median, seventy_fifth)
```

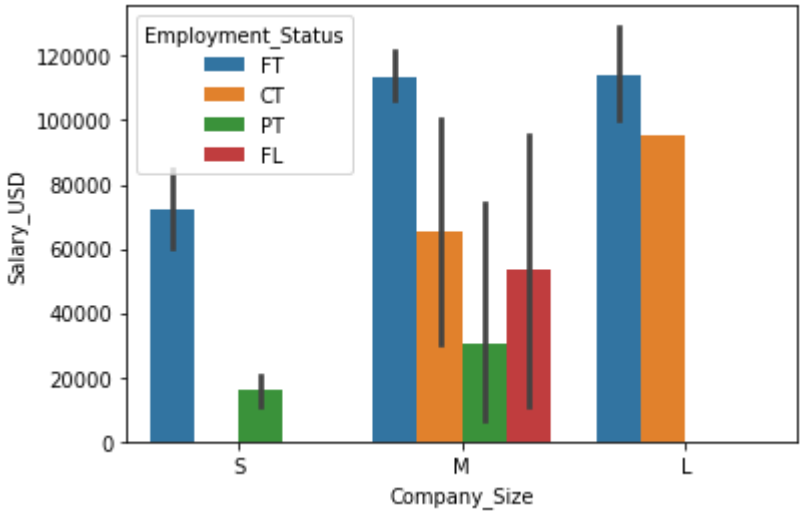
```
60880.691999999995 97488.552 143225.1
```

In [125]:

```
#Ex11 (A)

usa_and_gb = salaries[salaries["Employee_Location"].isin(["US", "GB"])]


plt.figure(figsize=(8, 6))

sns.barplot(data = usa_and_gb, x = "Employee_Location", y = "Salary_USD",

plt.title("Average Salary Comparison: USA vs. Great Britain")

plt.xlabel("Employee Location")

plt.ylabel("Average Salary (USD)")

plt.show()
```



In [126]:
```
#Ex11 (B)

sns.barplot(data = salaries, y = "Salary_USD", x = "Company_Size", hue = "

plt.show()
```



In [ ]:
```
#Chossing a hypothesis

I choose B - b. Freelancers earn more at medium-sized companies compared t
```