EDA OF TRAVEL DATASET, LIMITED DESCRIPTION OF THE DATASET IS PRESENT ,HENCE SOME ASSUMPTION OF FEATURES HAVE BEEN TAKEN

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import statistics
```

Import the data set as pandas dataframe

In [3]:
```
data = pd.read_csv(r"F:\ineuron FSDS\Dataset EDA sunny\Dataset\data1\Travel.csv")
```

In [4]:
```
data_clean=data # make a copy of dataset to perform Feature engineering together
```

The dataset is imported. Lets check the data set

In [5]:
```
data.head() #to view top 5 record
```

Out[5]:

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	N
0	200000	1	41.0	Self Enquiry	3	6.0	Salaried	Female	
1	200001	0	49.0	Company Invited	1	14.0	Salaried	Male	
2	200002	1	37.0	Self Enquiry	1	8.0	Free Lancer	Male	
3	200003	0	33.0	Company Invited	1	9.0	Salaried	Female	
4	200004	0	NaN	Self Enquiry	1	8.0	Small Business	Male	

In [6]:
```
data.tail() # to view bottom 5 record
```

Out[6]:

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender
4883	204883	1	49.0	Self Enquiry	3	9.0	Small Business	Male
4884	204884	1	28.0	Company Invited	1	31.0	Salaried	Male
4885	204885	1	52.0	Self Enquiry	3	17.0	Salaried	Female
4886	204886	1	19.0	Self Enquiry	3	16.0	Small Business	Male
4887	204887	1	36.0	Self Enquiry	1	14.0	Salaried	Male

DESCRIPTION OF COLUMNS

```
CustomerID                  - Customer ID
ProdTaken                   - Product taken or not
Age                         - Age of customer
TypeofContact               - The type of source
CityTier                    - City tier of customer
DurationOfPitch             - Time of pitch given to customer
Occupation                  - Occupation of customer
Gender                      - Gender of customer
NumberOfPersonVisiting      - Total number of person visiting
NumberOfFollowups           - Number of follw-ups done(More clairity
required)
ProductPitched              - Type of product pitched
PreferredPropertyStar       - Property opted
MaritalStatus               - Marital status of customer
NumberOfTrips               - Number of times travelled
Passport                    - If possess a passport
PitchSatisfactionScore      - Pitch satisfaction score
OwnCar                      - Does own a car or not
NumberOfChildrenVisiting    - Number of childerns visiting
Designation                 - Designation of customer
MonthlyIncome               - Total Monthly income
```

In [7]:
```
# check the shape of data
data.shape
```

Out[7]:
```
(4888, 20)
```

In [302…
```
#observation = 4888 rows and 20 columns
```

In [9]:
```
#to check datatype
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 20 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   CustomerID               4888 non-null   int64
 1   ProdTaken                4888 non-null   int64
 2   Age                      4662 non-null   float64
 3   TypeofContact            4863 non-null   object
 4   CityTier                 4888 non-null   int64
 5   DurationOfPitch          4637 non-null   float64
 6   Occupation               4888 non-null   object
 7   Gender                   4888 non-null   object
 8   NumberOfPersonVisiting   4888 non-null   int64
 9   NumberOfFollowups        4843 non-null   float64
 10  ProductPitched           4888 non-null   object
 11  PreferredPropertyStar    4862 non-null   float64
 12  MaritalStatus            4888 non-null   object
 13  NumberOfTrips            4748 non-null   float64
 14  Passport                 4888 non-null   int64
 15  PitchSatisfactionScore   4888 non-null   int64
 16  OwnCar                   4888 non-null   int64
```

```
 17  NumberOfChildrenVisiting  4822 non-null   float64
 18  Designation               4888 non-null   object
 19  MonthlyIncome             4655 non-null   float64
dtypes: float64(7), int64(7), object(6)
memory usage: 763.9+ KB
```

To check null- values of data set

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
CustomerID                   0
ProdTaken                    0
Age                        226
TypeofContact               25
CityTier                     0
DurationOfPitch            251
Occupation                   0
Gender                       0
NumberOfPersonVisiting       0
NumberOfFollowups           45
ProductPitched               0
PreferredPropertyStar       26
MaritalStatus                0
NumberOfTrips              140
Passport                     0
PitchSatisfactionScore       0
OwnCar                       0
NumberOfChildrenVisiting    66
Designation                  0
MonthlyIncome              233
dtype: int64
```

In [11]:
```
data['CustomerID'].nunique()
```

Out[11]:
```
4888
```

In [12]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   CustomerID                4888 non-null   int64
 1   ProdTaken                 4888 non-null   int64
 2   Age                       4662 non-null   float64
 3   TypeofContact             4863 non-null   object
 4   CityTier                  4888 non-null   int64
 5   DurationOfPitch           4637 non-null   float64
 6   Occupation                4888 non-null   object
 7   Gender                    4888 non-null   object
 8   NumberOfPersonVisiting    4888 non-null   int64
 9   NumberOfFollowups         4843 non-null   float64
 10  ProductPitched            4888 non-null   object
 11  PreferredPropertyStar     4862 non-null   float64
 12  MaritalStatus             4888 non-null   object
 13  NumberOfTrips             4748 non-null   float64
 14  Passport                  4888 non-null   int64
 15  PitchSatisfactionScore    4888 non-null   int64
 16  OwnCar                    4888 non-null   int64
 17  NumberOfChildrenVisiting  4822 non-null   float64
```

```
 18  Designation               4888 non-null   object
 19  MonthlyIncome             4655 non-null   float64
dtypes: float64(7), int64(7), object(6)
memory usage: 763.9+ KB
```

Observation from above = few of the columns have null values and needs to be treated

Treating Age column for missing data, Rest Null-Values will be done later in Feature Engineering

In [305…
```
data['Age'].mean() #mean age is 37.61
```

Out[305…
```
37.54725859247136
```

In [306…
```
data['Age'].min()
```

Out[306…
```
18.0
```

In [307…
```
data['Age'].max()
```

Out[307…
```
61.0
```

In [308…
```
data['Age'].median() # to find the median of Age
```

Out[308…
```
36.0
```

In [309…
```
data['Age'].mode() # mode - 35
```

Out[309…
```
0    36.0
dtype: float64
```

Conclusion - There is no major difference between between mean , median and mode, Hence it is safe to replace null values with median

In [310…
```
data_clean['Age'].fillna(data['Age'].median(),inplace=True)
```

In [311…
```
data[data['Age'].isnull()==True] # Age column have no null value now
```

Out[311…

CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Nur

In [312…
```
data_clean['Age'].isnull().sum() # All null values have been replaced
```

Out[312…
```
0
```

In [314…
```
data.memory_usage() ## Memomry usage of the dataset
```

Out[314…
```
Index                      128
CustomerID               39104
ProdTaken                39104
Age                      39104
TypeofContact            39104
CityTier                 39104
DurationOfPitch          39104
Occupation               39104
Gender                   39104
NumberOfPersonVisiting   39104
NumberOfFollowups        39104
ProductPitched           39104
PreferredPropertyStar    39104
MaritalStatus            39104
NumberOfTrips            39104
Passport                 39104
PitchSatisfactionScore   39104
OwnCar                   39104
NumberOfChildrenVisiting 39104
Designation              39104
MonthlyIncome            39104
dtype: int64
```

In [315…
```
data[data['CustomerID'].duplicated()==True] # no duplicate customerID found
```

Out[315…

CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	Nur

Let's segregate data based on numerical / categorical columns

In [25]:
```
cat_col=[fea for fea in data.columns if data[fea].dtypes=='O']
```

In [26]:
```
num_col=[fea for fea in data.columns if data[fea].dtypes!='O']
```

In [316…
```
cat_col
```

Out[316…
```
['TypeofContact',
 'Occupation',
 'Gender',
 'ProductPitched',
 'MaritalStatus',
 'Designation']
```

6 categorical columns found

In [28]:
```
num_col
```

Out[28]:
```
['CustomerID',
 'ProdTaken',
 'Age',
 'CityTier',
 'DurationOfPitch',
 'NumberOfPersonVisiting',
 'NumberOfFollowups',
```

```
'PreferredPropertyStar',
'NumberOfTrips',
'Passport',
'PitchSatisfactionScore',
'OwnCar',
'NumberOfChildrenVisiting',
'MonthlyIncome']
```

Above are Numerical columns

In [29]:
```
data.describe().T # Description of Numerical Columns
```

Out[29]:

	count	mean	std	min	25%	50%	7
CustomerID	4888.0	202443.500000	1411.188388	200000.0	201221.75	202443.5	20366
ProdTaken	4888.0	0.188216	0.390925	0.0	0.00	0.0	(
Age	4888.0	37.547259	9.104795	18.0	31.00	36.0	4
CityTier	4888.0	1.654255	0.916583	1.0	1.00	1.0	
DurationOfPitch	4637.0	15.490835	8.519643	5.0	9.00	13.0	2(
...	...	...	...	...	...	...	
Passport	4888.0	0.290917	0.454232	0.0	0.00	0.0	
PitchSatisfactionScore	4888.0	3.078151	1.365792	1.0	2.00	3.0	
OwnCar	4888.0	0.620295	0.485363	0.0	0.00	1.0	
NumberOfChildrenVisiting	4822.0	1.187267	0.857861	0.0	1.00	1.0	
MonthlyIncome	4655.0	23619.853491	5380.698361	1000.0	20346.00	22347.0	2557

In [318...
```
data[num_col].head()
```

Out[318...

	CustomerID	ProdTaken	Age	CityTier	DurationOfPitch	NumberOfPersonVisiting	NumberOfFollc
0	200000	1	41.0	3	6.0	3	
1	200001	0	49.0	1	14.0	3	
2	200002	1	37.0	1	8.0	3	
3	200003	0	33.0	1	9.0	2	
4	200004	0	36.0	1	8.0	2	

Selecting discrete and continious features

In [32]:
```
#selecting based on data type 'int64' as discrete
num_col_discrete = [fea for fea in data.columns if data[fea].dtypes=='int64']
```

In [33]:
```
#selecting based on data type 'float64' as continiuos
num_col_continious = [fea for fea in data.columns if data[fea].dtypes=='float64']
```

In [34]:
```
data[num_col_continious].describe().T
```

Out[34]:

	count	mean	std	min	25%	50%	75%	
Age	4888.0	37.547259	9.104795	18.0	31.0	36.0	43.0	
DurationOfPitch	4637.0	15.490835	8.519643	5.0	9.0	13.0	20.0	1
NumberOfFollowups	4843.0	3.708445	1.002509	1.0	3.0	4.0	4.0	
PreferredPropertyStar	4862.0	3.581037	0.798009	3.0	3.0	3.0	4.0	
NumberOfTrips	4748.0	3.236521	1.849019	1.0	2.0	3.0	4.0	
NumberOfChildrenVisiting	4822.0	1.187267	0.857861	0.0	1.0	1.0	2.0	
MonthlyIncome	4655.0	23619.853491	5380.698361	1000.0	20346.0	22347.0	25571.0	986

Further selecting contnious features based on unique values , if feature have unique values greater than 5 it is considered

as continious

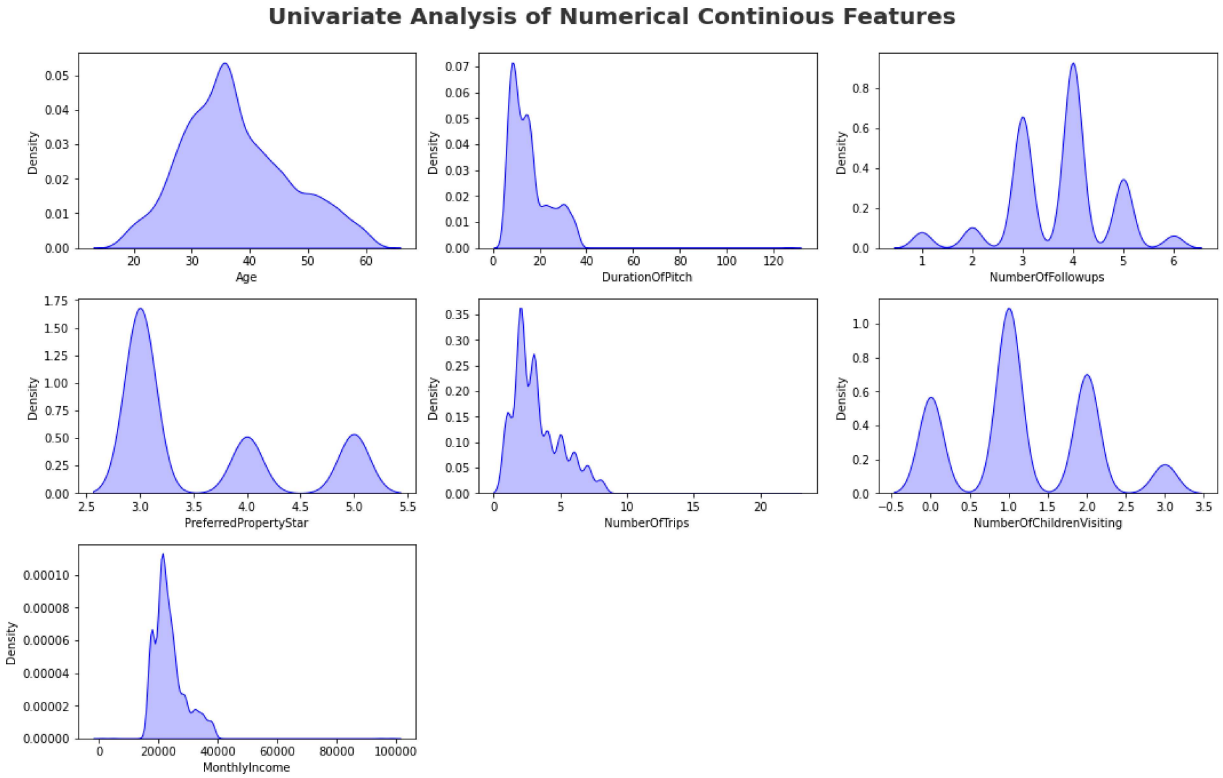
In [35]:
```
continious_features=[feature for feature in data[num_col_continious] if len(data[fea
print('Num of continious features :',continious_features)
```

```
Num of continious features : ['Age', 'DurationOfPitch', 'NumberOfFollowups', 'Number
OfTrips', 'NumberOfChildrenVisiting', 'MonthlyIncome']
```

Ploting continious features to understand distribuition of continious features

In [36]:
```
plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Continious Features', fontsize=20, fo

for i in range(0, len(num_col_continious)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=data[num_col_continious[i]],shade=True, color='b')
    plt.xlabel(num_col_continious[i])
    plt.tight_layout()
```



Observations

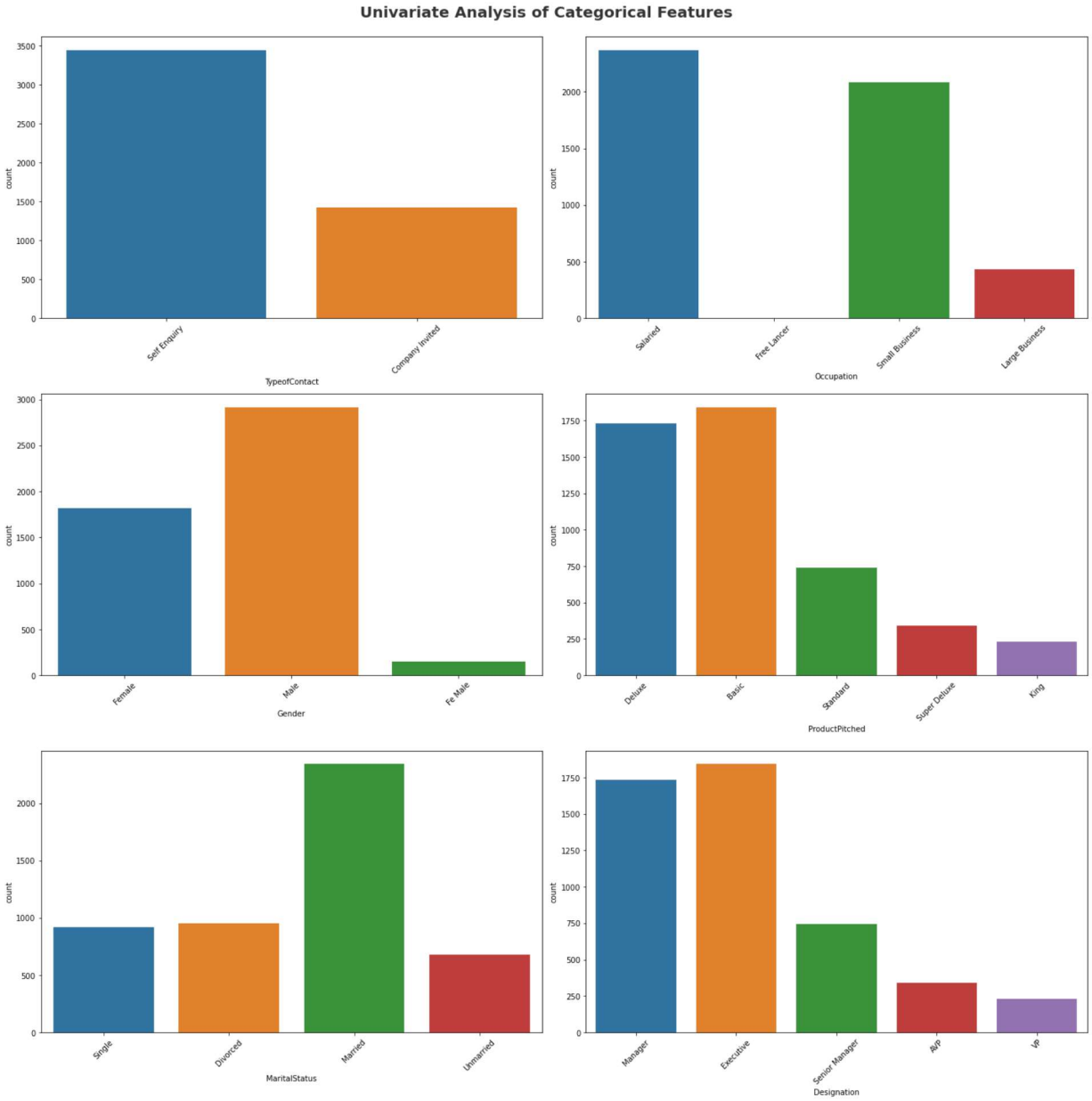
from above we can find out that duration of pitch , number of trips and mothly income have outliers and are right-skewed.

Analysis of Categorical data

In [321…
```
cat_col  ## Categorical columns
```

Out[321…
```
['TypeofContact',
 'Occupation',
 'Gender',
 'ProductPitched',
 'MaritalStatus',
 'Designation']
```

In [38]:
```
# categorical columns
plt.figure(figsize=(20, 20))
plt.suptitle('Univariate Analysis of Categorical Features', fontsize=20, fontweight=
for i in range(0, len(cat_col)):
    plt.subplot(3,2,i+1)
    sns.countplot(x=data[cat_col[i]])
    plt.xlabel(cat_col[i])
    plt.xticks(rotation=45)
    plt.tight_layout()
```



OBSERVATIONS from above

1. There are more self-enquiries than company invited contacts
2. Salaried people are the most preffered occupation among the dataset
3. Male customers have the most enquiries
4. Basic and Deluxe are the two most prefered prodcuts pitched.
5. Married persons does most enquires
6. Executive level designations travels most.

Let's now check for some correlations among continious features

In [323…

```
data.corr() # checking correlaitons
```

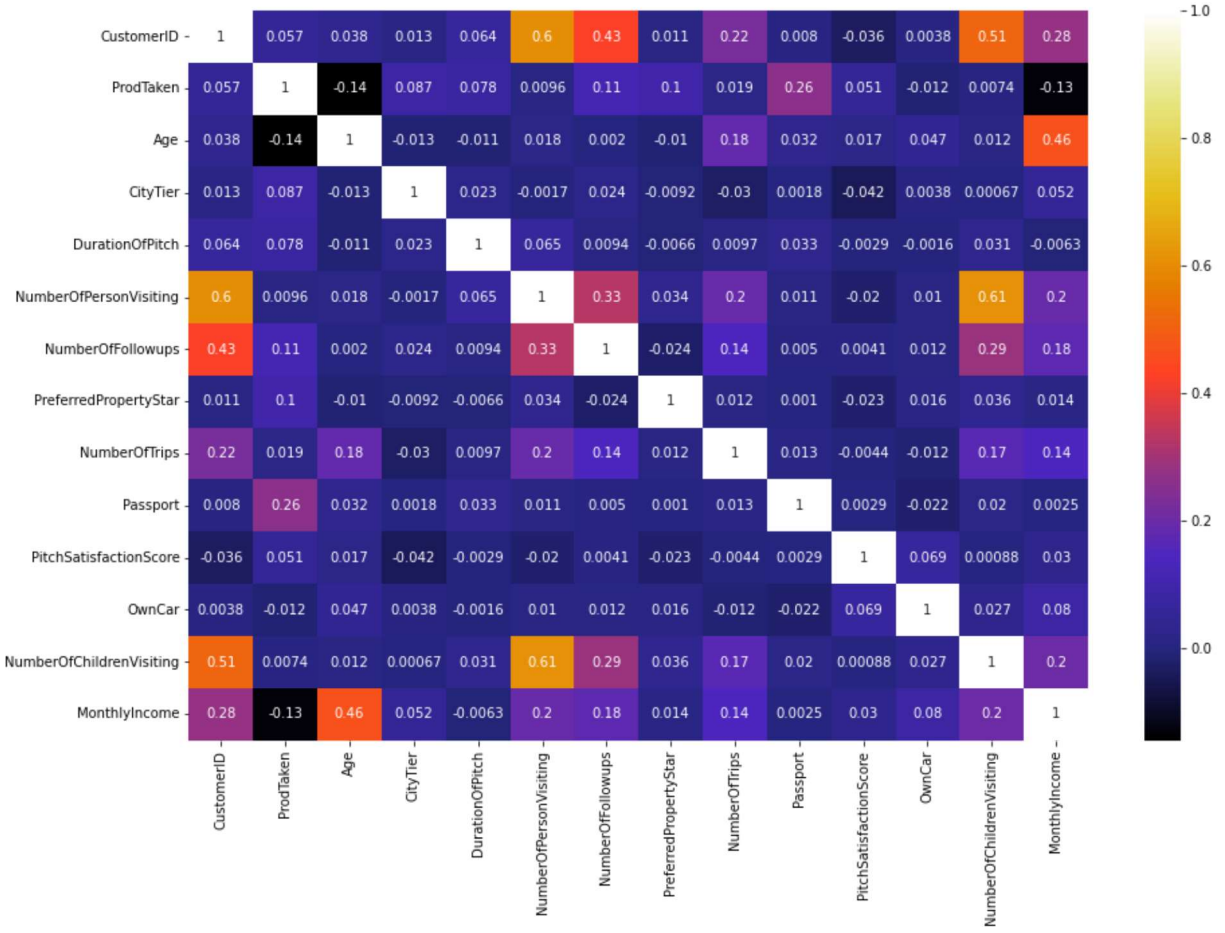
Out[323…

	CustomerID	ProdTaken	Age	CityTier	DurationOfPitch	NumberO
CustomerID	1.000000	0.056506	0.038442	0.012975	0.064298	
ProdTaken	0.056506	1.000000	-0.143753	0.086852	0.078257	
Age	0.038442	-0.143753	1.000000	-0.012754	-0.011443	
CityTier	0.012975	0.086852	-0.012754	1.000000	0.022703	

	CustomerID	ProdTaken	Age	CityTier	DurationOfPitch	NumberO
DurationOfPitch	0.064298	0.078257	-0.011443	0.022703	1.000000	
...	...	...	...	...	...	
Passport	0.007974	0.260844	0.032398	0.001793	0.033034	
PitchSatisfactionScore	-0.035847	0.051394	0.017392	-0.042160	-0.002880	
OwnCar	0.003805	-0.011508	0.047356	0.003817	-0.001626	
NumberOfChildrenVisiting	0.511763	0.007421	0.011999	0.000672	0.031408	
MonthlyIncome	0.276833	-0.130585	0.463867	0.051817	-0.006252	

Correlations through Heatmaps

In [40]:
```
plt.figure(figsize = (15,10))
sns.heatmap(data.corr(), cmap="CMRmap", annot=True)
plt.show()
```



Observations

1. ProdTaken has a weak negative correlation on Age and MontlyIncome.
2. The NumberOfFollowups and Passport columns also have a weak positive correlation with ProdTaken.
3. The NumberOfPersonVisiting and NumberOfChildrenVisiting columns have a strong enough positive correlation.
4. Age has a moderately positive correlation with Monthly Income.

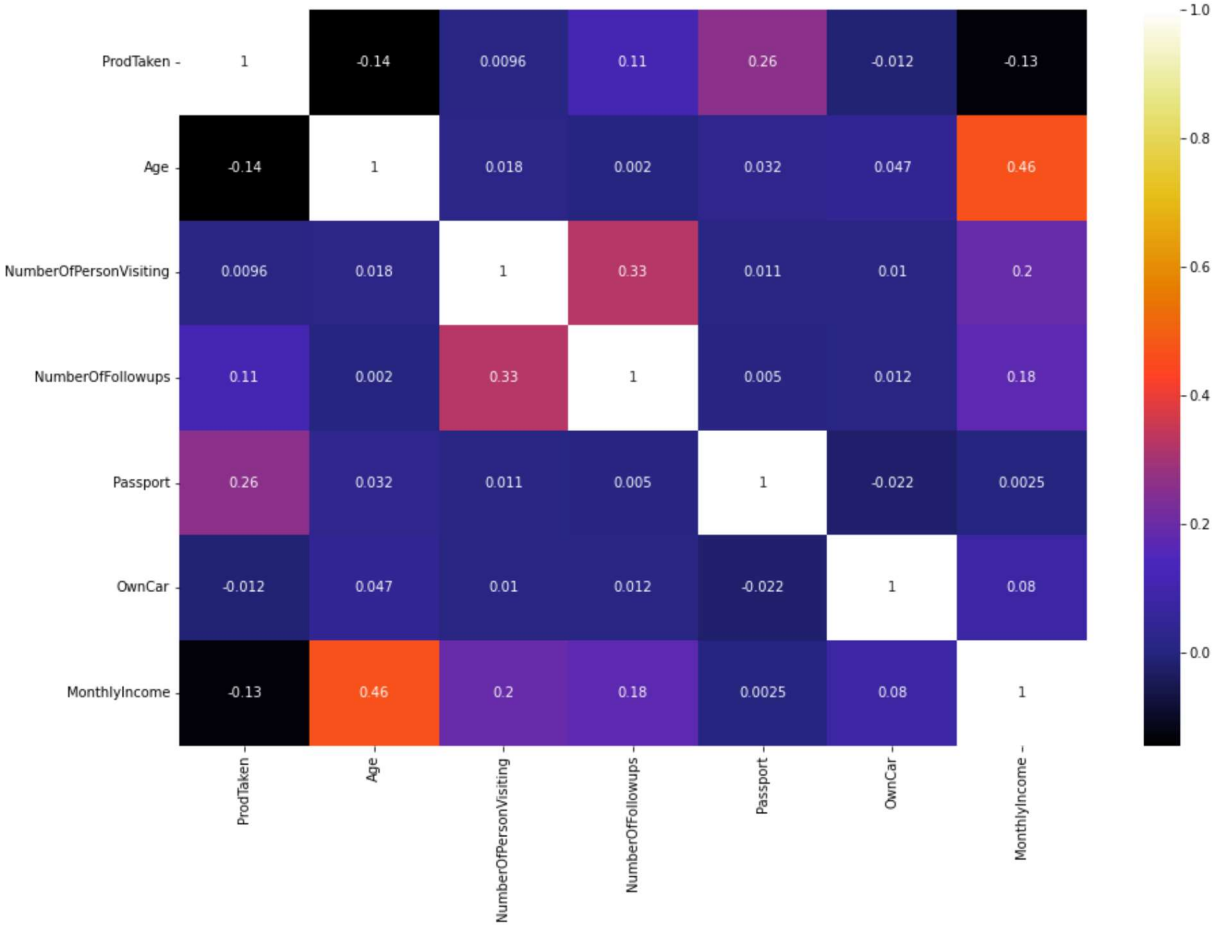
Let's reduce the number of columns and work on only selected columns to checl correlations

In [42]:
```
selected_num_col=['ProdTaken',
 'Age',
 'NumberOfPersonVisiting',
 'NumberOfFollowups',
 'Passport',
 'OwnCar',
 'MonthlyIncome']
```

In [43]:
```
selected_num_col# have removed some features and only want to see correlation of thi
```

Out[43]:
```
['ProdTaken',
 'Age',
 'NumberOfPersonVisiting',
 'NumberOfFollowups',
 'Passport',
 'OwnCar',
 'MonthlyIncome']
```

In [44]:
```
plt.figure(figsize = (15,10))
sns.heatmap(data[selected_num_col].corr(), cmap="CMRmap", annot=True)
plt.show()
```



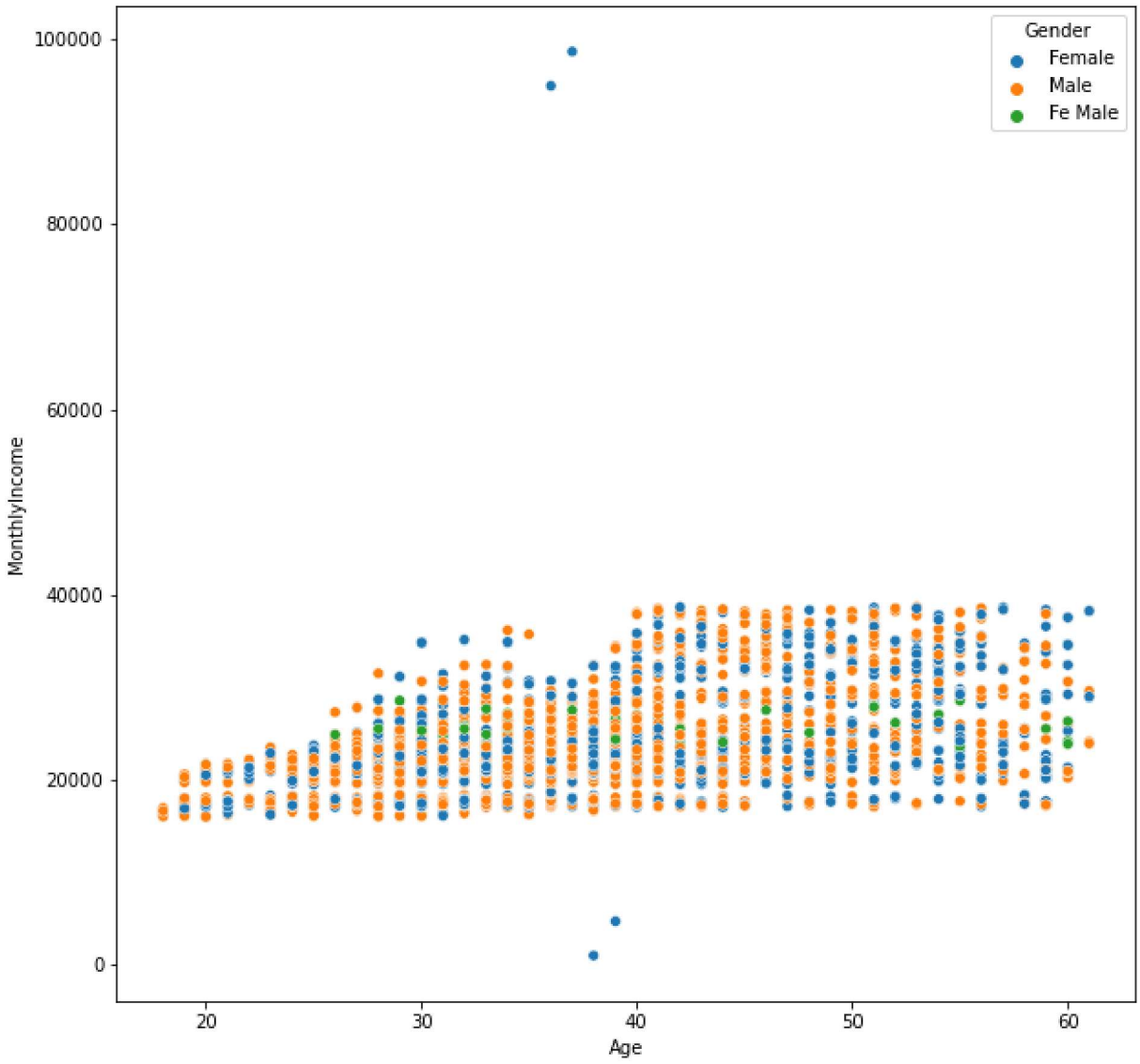
The observation remains same as previous heatmap

Let's now visualize our data to retrieve some insights

Distribution to show monthly income vs age

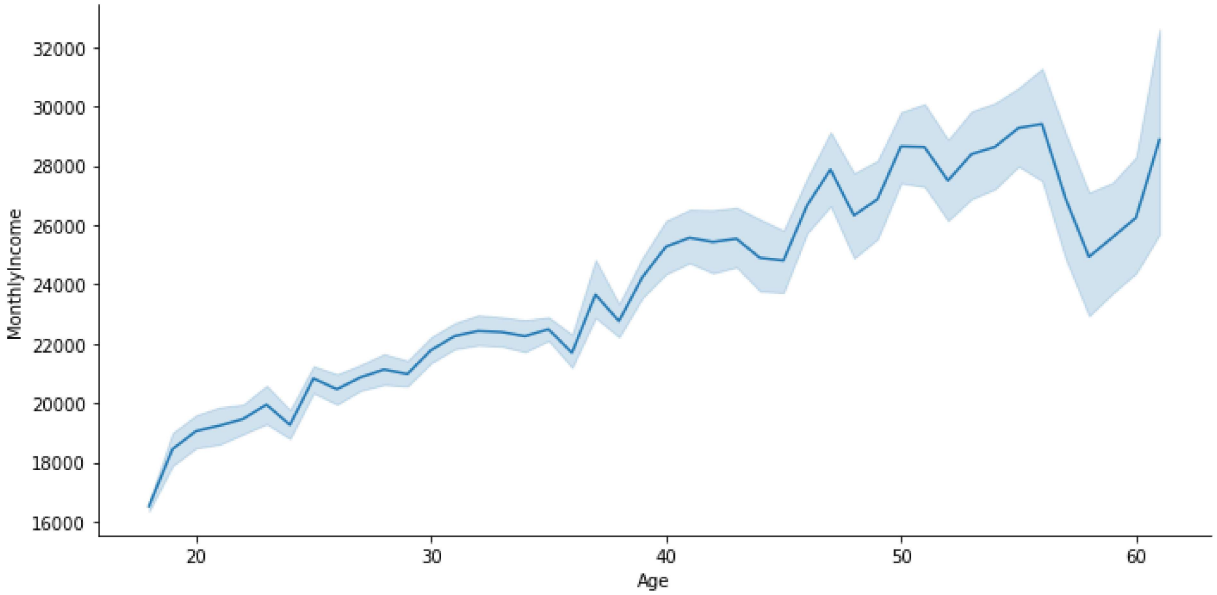
In [45]:
```
plt.figure(figsize=(10,10))
sns.scatterplot(x=data['Age'],y=data['MonthlyIncome'],hue=data['Gender'])
```

Out[45]:
```
<AxesSubplot:xlabel='Age', ylabel='MonthlyIncome'>
```



In [326…
```
sns.relplot(x=data['Age'],y=data['MonthlyIncome'],kind="line",height=5,aspect=2)
```

Out[326…
```
<seaborn.axisgrid.FacetGrid at 0x1e63e798b50>
```



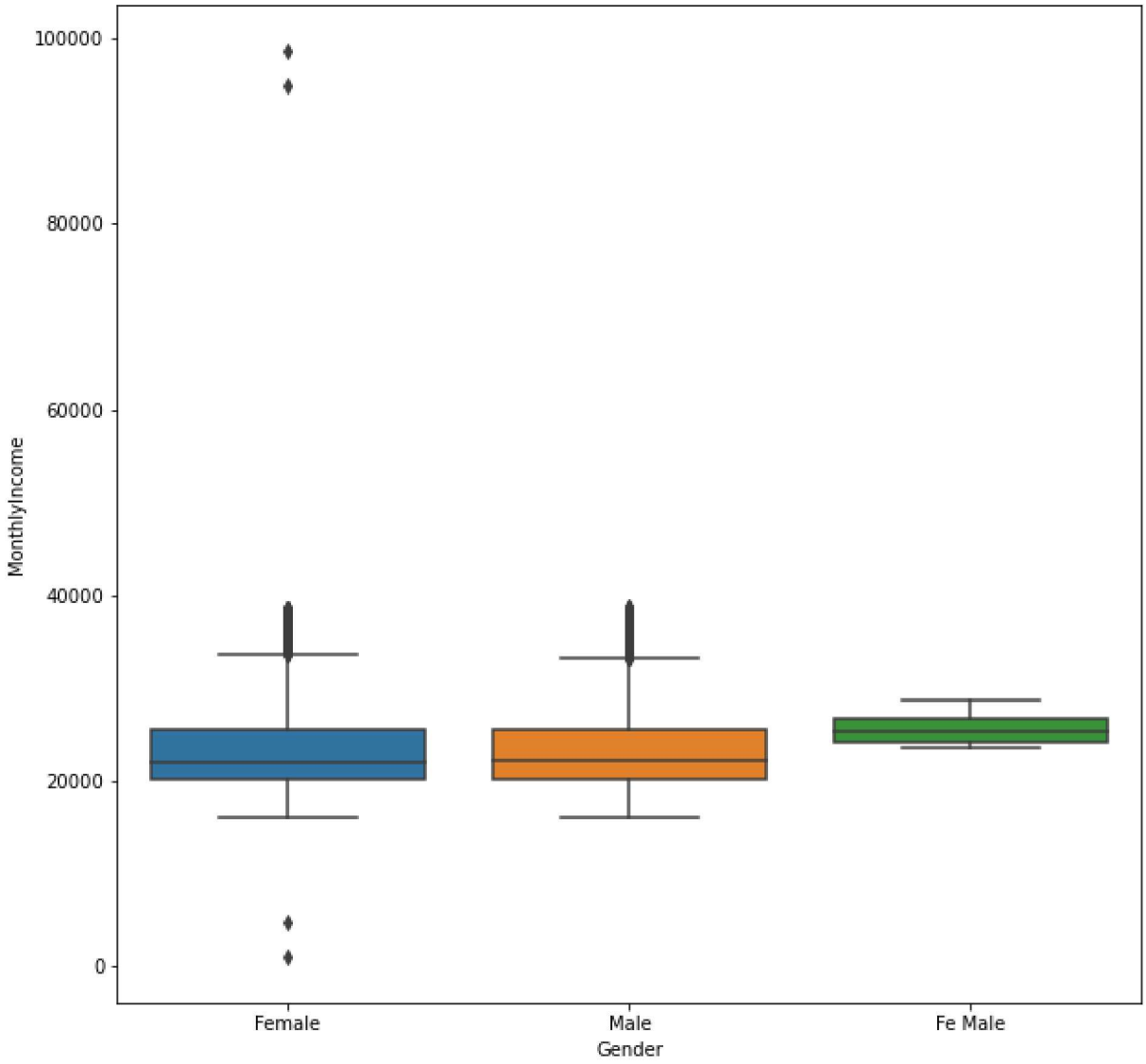
Observations - Two outliers can be seen in income and needs some attention

Trend of income increases with age however dips around age 55

In [328…
```
## Box plot to check Monthly income vs gender and also identify outliers
```

In [330…
```
plt.figure(figsize=(10,10))
sns.boxplot(data=data,x="Gender",y="MonthlyIncome")
```

Out[330…
```
<AxesSubplot:xlabel='Gender', ylabel='MonthlyIncome'>
```



Observations - To be specific the outliers are seen in Female gender type

Also no clarity is present on for 3rd Gender type which can be treated later

Income range for both Male/Female is moderately same

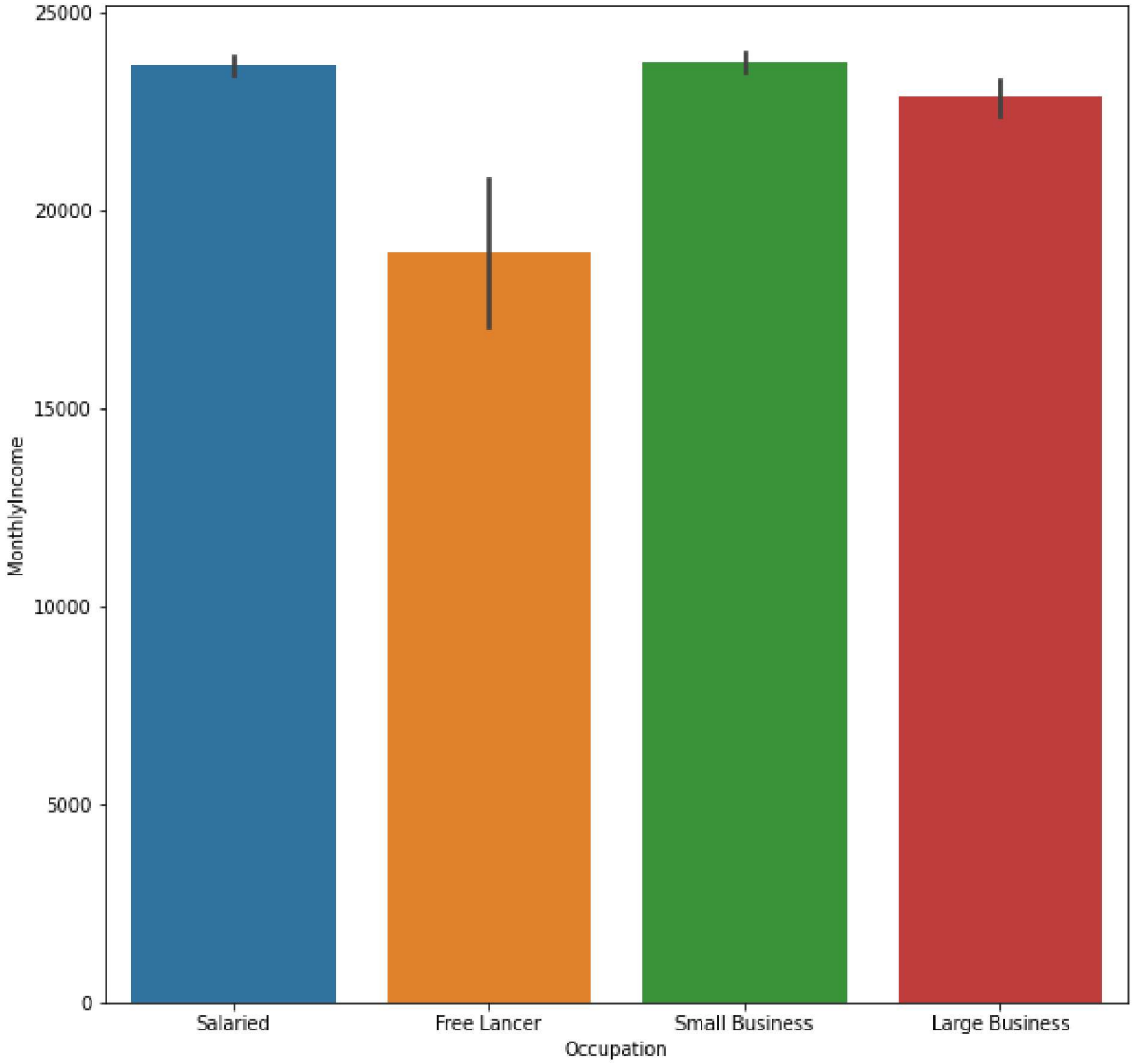
In [48]:
```
data.columns
```

Out[48]:
```
Index(['CustomerID', 'ProdTaken', 'Age', 'TypeofContact', 'CityTier',
       'DurationOfPitch', 'Occupation', 'Gender', 'NumberOfPersonVisiting',
       'NumberOfFollowups', 'ProductPitched', 'PreferredPropertyStar',
       'MaritalStatus', 'NumberOfTrips', 'Passport', 'PitchSatisfactionScore',
       'OwnCar', 'NumberOfChildrenVisiting', 'Designation', 'MonthlyIncome'],
      dtype='object')
```

Bar plot to check occupation vs mean monthly Income

In [49]:
```
plt.figure(figsize=(10,10))
sns.barplot(data=data,x='Occupation',y='MonthlyIncome')
```

Out[49]:
```
<AxesSubplot:xlabel='Occupation', ylabel='MonthlyIncome'>
```

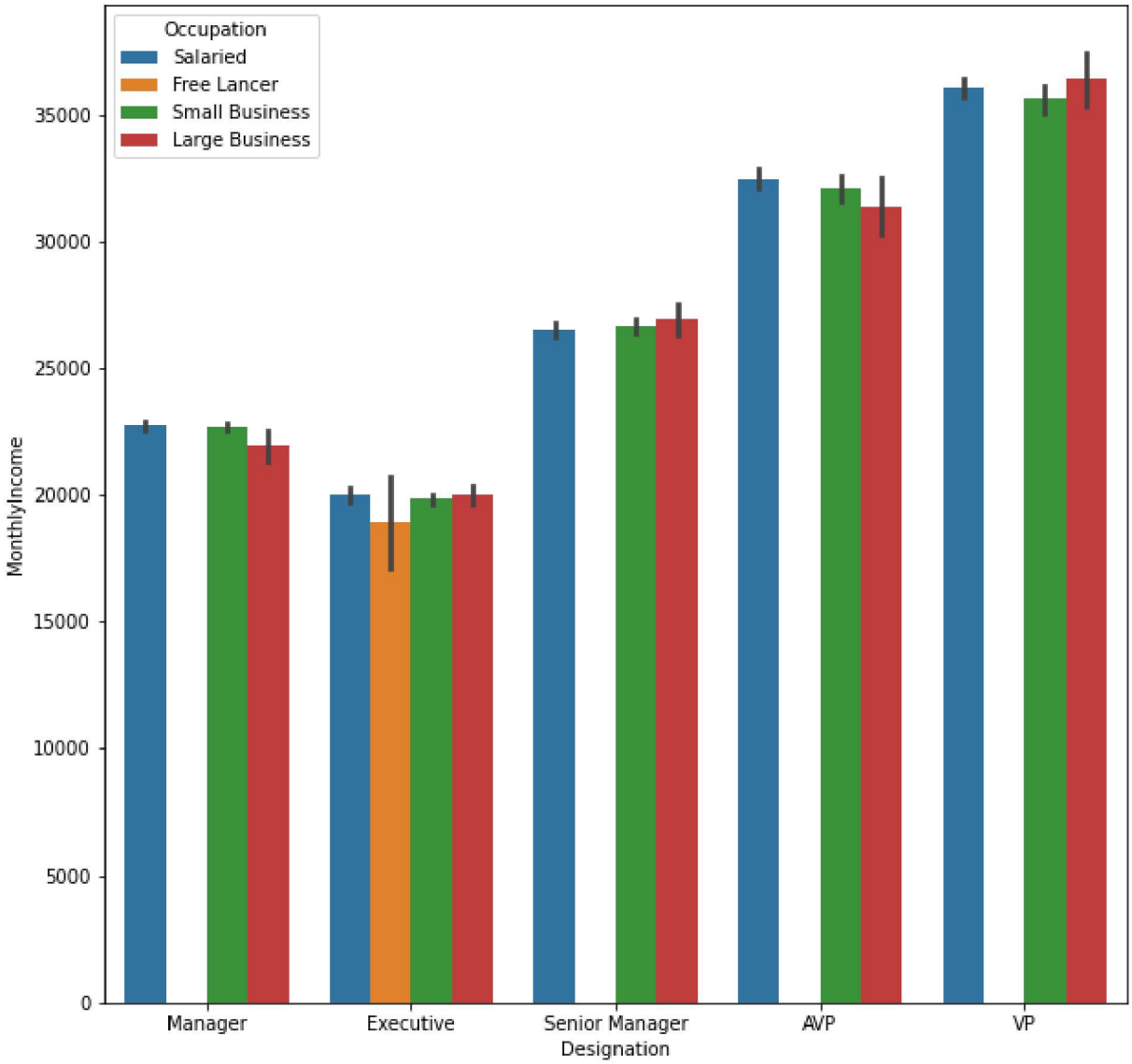


Observation - Salaried and small business owners have higher mean income

Bar plot based on Occupation to show monthly income for different designations

In [50]:
```
plt.figure(figsize=(10,10))
sns.barplot(data=data,x='Designation',y='MonthlyIncome',hue='Occupation')
```

Out[50]:
```
<AxesSubplot:xlabel='Designation', ylabel='MonthlyIncome'>
```



Lets find out which age has maximum number of trips

In [351…
```
# Groupby operation to filter count of trips by age
df_age_trips = data.groupby(['Age'])['NumberOfTrips'].count().sort_values(ascending=
```

In [352…
```
df_age_trips.head()
```

Out[352…

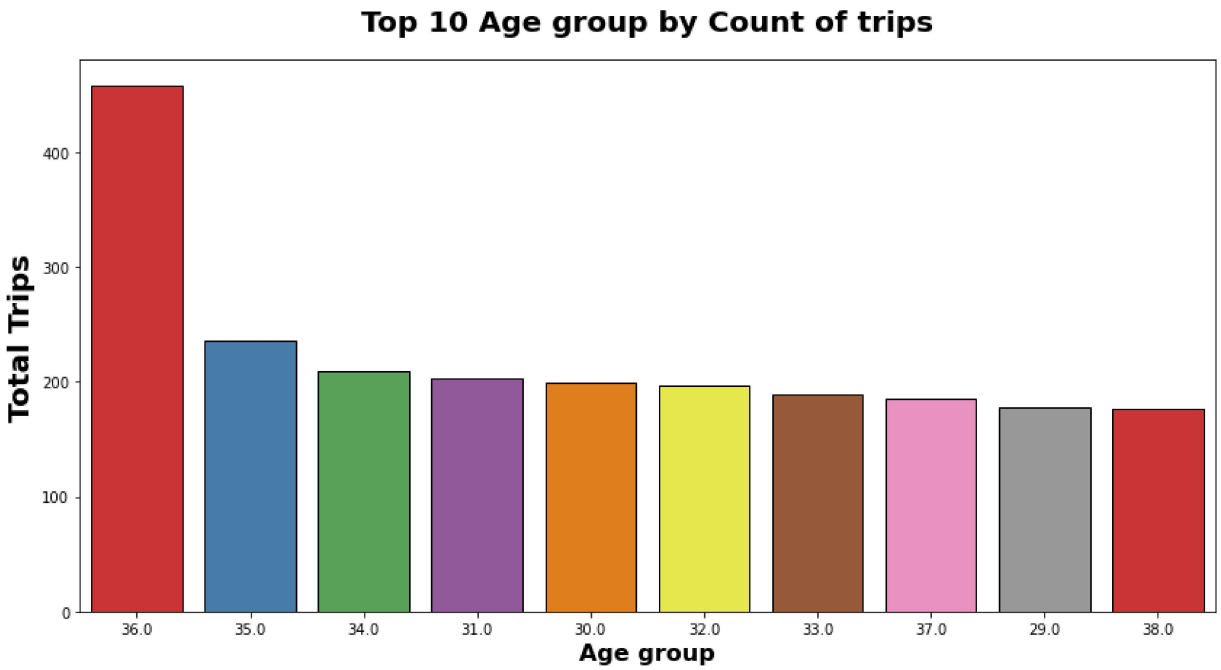
	Age	NumberOfTrips
0	36.0	457
1	35.0	236
2	34.0	209
3	31.0	203
4	30.0	199

In [347…
```
top_10_age=df_age_trips.Age.values[:10] #slicing operation for top 10 age from a sor
```

In [348…
```
top_10_trips = df_age_trips.NumberOfTrips.values[:10] #slicing operation for top 10
```

In [349…
```
#slicing operation for bottome 10 age and trips from a sorted dataframe
bottom_10_age,bottom_10_trips = df_age_trips.Age.values[(len(df_age_trips)-10):],df_
```

In [350…
```
# Plotting top 10 age group by total trips
plt.subplots(figsize=(14,7))
sns.barplot(data=df_age_trips,x=top_10_age,y=top_10_trips,order=top_10_age,palette="
plt.title("Top 10 Age group by Count of trips", weight="bold",fontsize=20, pad=20)
plt.ylabel("Total Trips", weight="bold", fontsize=20)
plt.xlabel("Age group", weight="bold", fontsize=16)
plt.show()
```



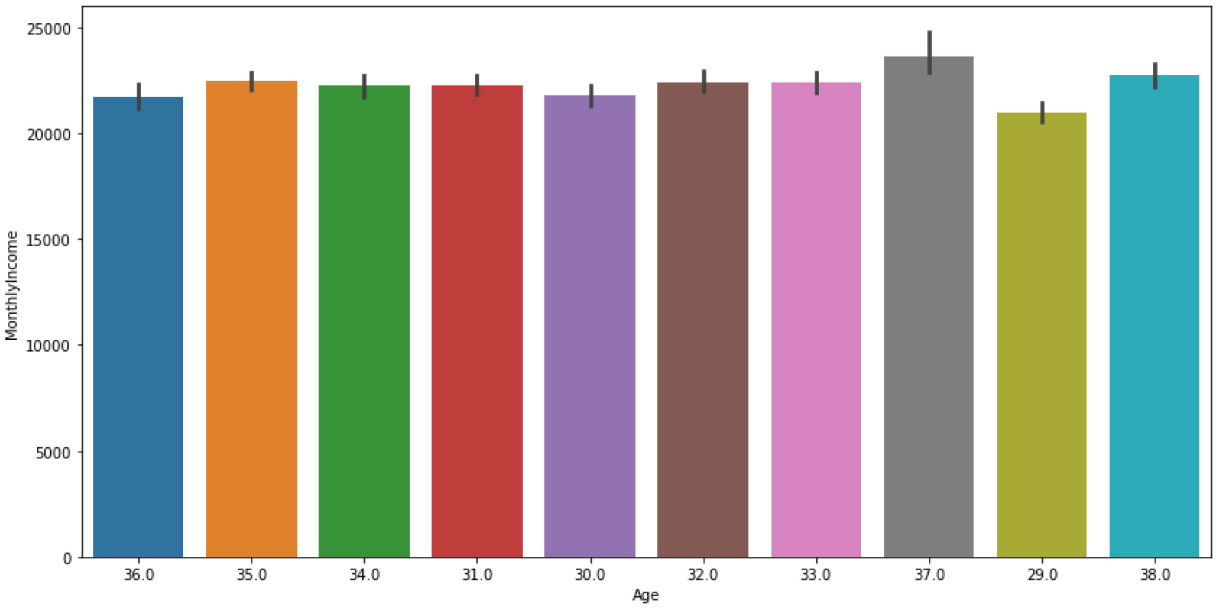
Obseravation - At the age of 36, people travelled most

Let's see if income effects the spirit to travel anyhow or not below

Age 29-38 forms the top 10 age to make trips

In [353…
```
plt.figure(figsize=(14,7))
sns.barplot(data=data,x=data['Age'],y='MonthlyIncome',order=top_10_age)
```

Out[353…
```
<AxesSubplot:xlabel='Age', ylabel='MonthlyIncome'>
```

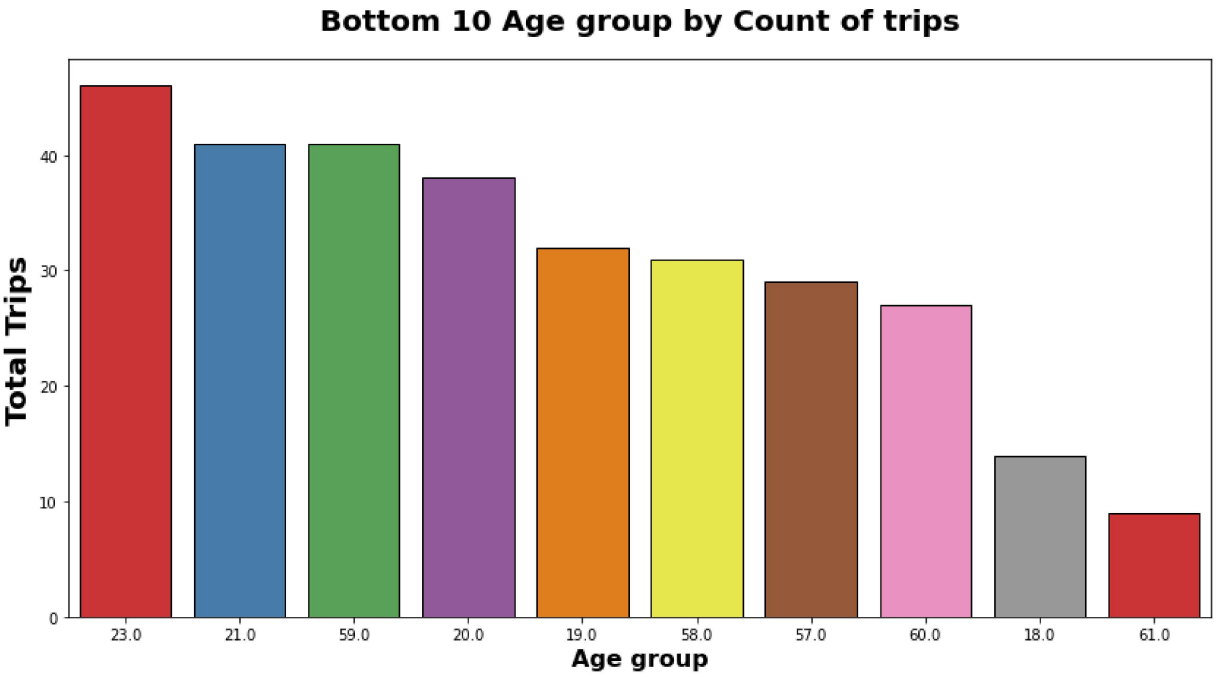


Observations - The income level is not highest at age 36, but travelling is most

Let's check for bottom 10 age group by trips next

In [354…

```
plt.subplots(figsize=(14,7))
sns.barplot(data=df_age_trips,x=bottom_10_age,y=bottom_10_trips,order=bottom_10_age,
plt.title("Bottom 10 Age group by Count of trips", weight="bold",fontsize=20, pad=20
plt.ylabel("Total Trips", weight="bold", fontsize=20)
plt.xlabel("Age group", weight="bold", fontsize=16)
plt.show()
```

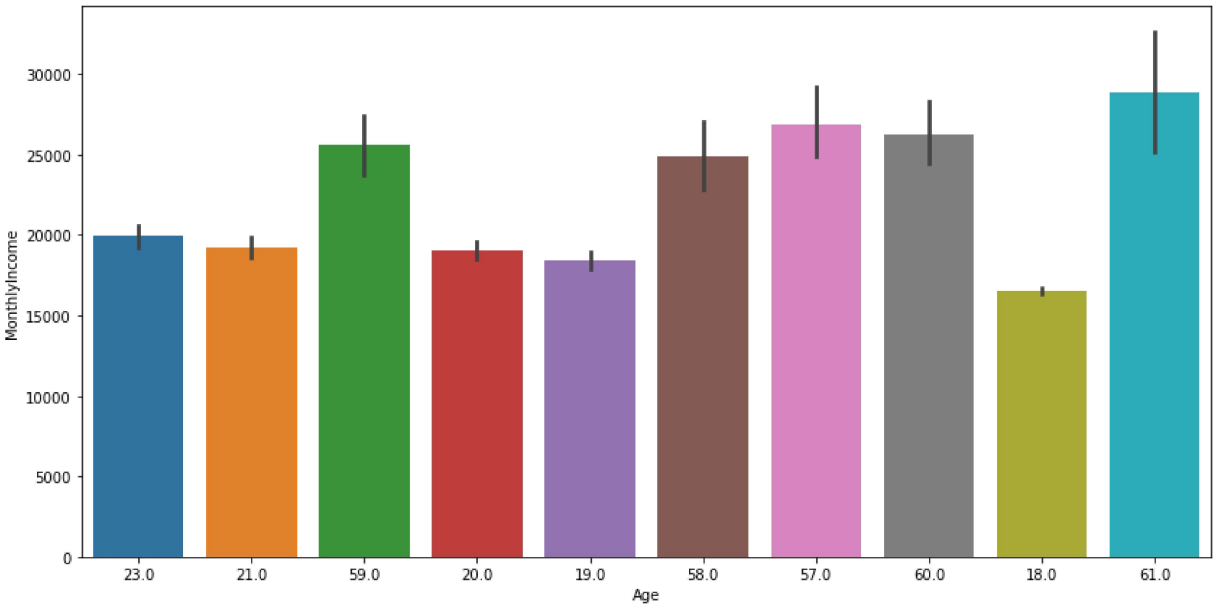


In [355…

```
plt.figure(figsize=(14,7))
sns.barplot(data=data,x=data['Age'],y='MonthlyIncome',order=bottom_10_age)
```

Out[355…

```
<AxesSubplot:xlabel='Age', ylabel='MonthlyIncome'>
```



Observation - Though we can earn high at 61, but we can't travel more,

At age 18, money is a factor which is why people at 18 are at second last in terms of trips

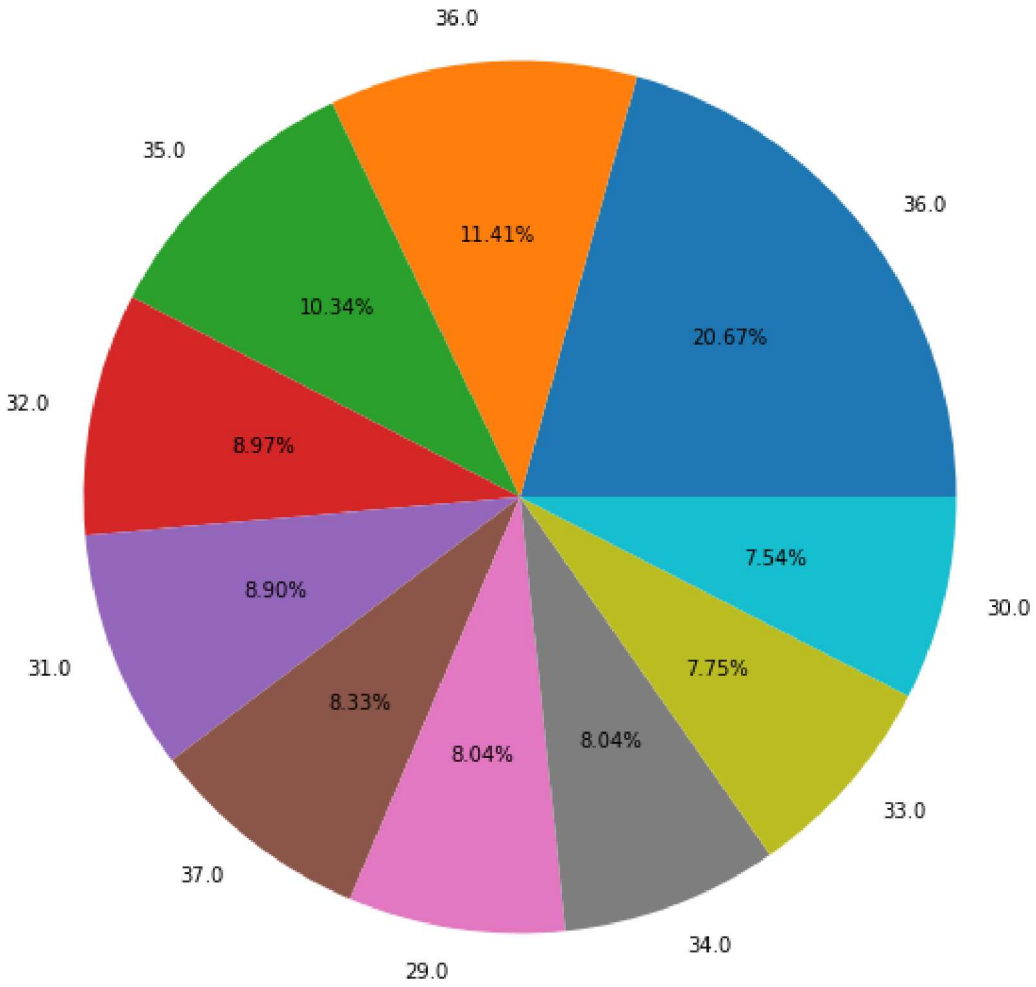
Age group 57-61 and 18-23 forms the lowest 10 age's to trip

In [61]:
```
plt.figure(figsize=(10,10))

plt.pie(data=df_age_trips,x=top_10_trips,labels=top_10_age,autopct='%1.2f%%')
```

Out[61]:
```
([<matplotlib.patches.Wedge at 0x1e634d3a4c0>,
  <matplotlib.patches.Wedge at 0x1e6370f0520>,
  <matplotlib.patches.Wedge at 0x1e6370f0dc0>,
  <matplotlib.patches.Wedge at 0x1e637116e20>,
  <matplotlib.patches.Wedge at 0x1e637116a00>,
  <matplotlib.patches.Wedge at 0x1e6370e5a60>,
  <matplotlib.patches.Wedge at 0x1e6370e55e0>,
  <matplotlib.patches.Wedge at 0x1e6370e5df0>,
  <matplotlib.patches.Wedge at 0x1e636ff9940>,
  <matplotlib.patches.Wedge at 0x1e636ff9790>],
 [Text(0.8760128984831363, 0.6652829485949376, '36.0'),
  Text(-0.09539076095295007, 1.0958561049356879, '36.0'),
  Text(-0.7658870154267502, 0.7895676535932213, '35.0'),
  Text(-1.0793749363854745, 0.21201355311124137, '32.0'),
  Text(-1.0265081650679744, -0.3953239014387829, '31.0'),
  Text(-0.6760975753671973, -0.8676935337909331, '37.0'),
  Text(-0.16190200345607478, -1.088020101504062, '29.0'),
  Text(0.38488595476677284, -1.0304672735333567, '34.0'),
  Text(0.8290311303287367, -0.7229850516752456, '33.0'),
  Text(1.0693021080282021, -0.2580561988529696, '30.0')],
 [Text(0.47782521735443795, 0.36288160832451144, '20.67%'),
  Text(-0.05203132415615458, 0.5977396936012842, '11.41%'),
  Text(-0.4177565538691364, 0.4306732655963025, '10.34%'),
  Text(-0.5887499653011679, 0.11564375624249527, '8.97%'),
  Text(-0.5599135445825314, -0.2156312189666088, '8.90%'),
  Text(-0.36878049565483484, -0.4732873820677816, '8.33%'),
  Text(-0.0883101837033135, -0.5934655099113064, '8.04%'),
  Text(0.2099377935091488, -0.5620730582909218, '8.04%'),
```

```
Text(0.452198798361129, -0.3943554827319521, '7.75%'),
Text(0.5832556952881102, -0.1407579266470743, '7.54%')])
```



OF the top 10 age, 20.67% people are of age 36

For more clarity below is illustration of top 20 age group with mean monthly income

In [62]:
```
income_age = data.groupby(['Age'])['MonthlyIncome'].mean().sort_values(ascending=Fal
```

In [356…
```
income_age.head()
```

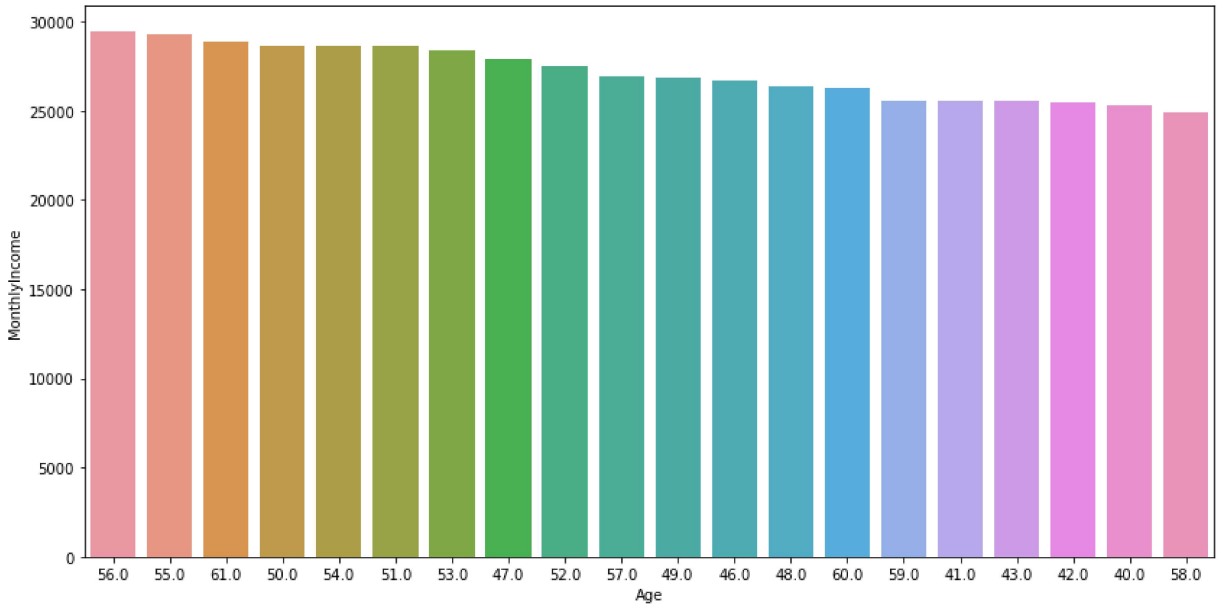
Out[356…

	Age	MonthlyIncome
0	56.0	29407.696429
1	55.0	29271.625000
2	61.0	28877.666667
3	50.0	28653.534884
4	54.0	28636.333333

In [64]:
```
plt.figure(figsize=(14,7))
```

```
sns.barplot(data=income_age,x=income_age['Age'][:20],y=income_age['MonthlyIncome'],o
```

Out[64]:
```
<AxesSubplot:xlabel='Age', ylabel='MonthlyIncome'>
```



Observation - Age 56 have the highest mean average monthly income

Let's now explore Who travels the most based on marital status

In [357…
```
# Group by operation to get marital staus vs count of trips
MaritalStatus_trips=data.groupby(['MaritalStatus'])['NumberOfTrips'].count().sort_va
```

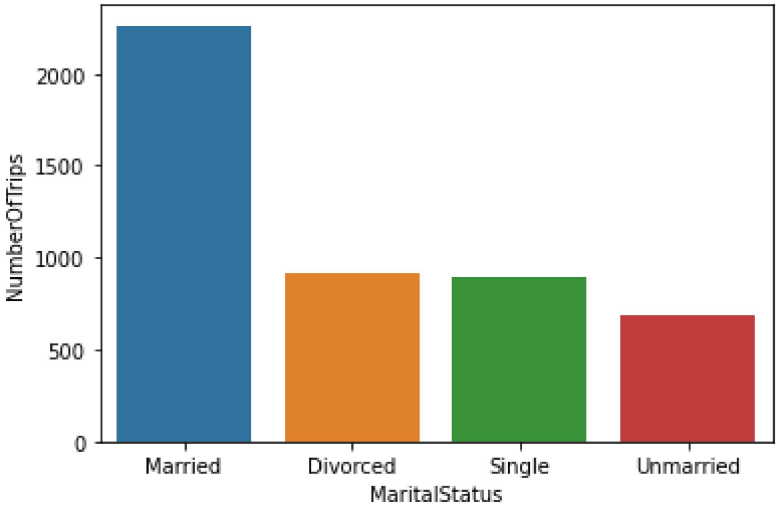
In [358…
```
MaritalStatus_trips.head()
```

Out[358…

	MaritalStatus	NumberOfTrips
0	Married	2262
1	Divorced	915
2	Single	889
3	Unmarried	682

In [68]:
```
sns.barplot(data=MaritalStatus_trips,y="NumberOfTrips",x='MaritalStatus',)
```

Out[68]:
```
<AxesSubplot:xlabel='MaritalStatus', ylabel='NumberOfTrips'>
```



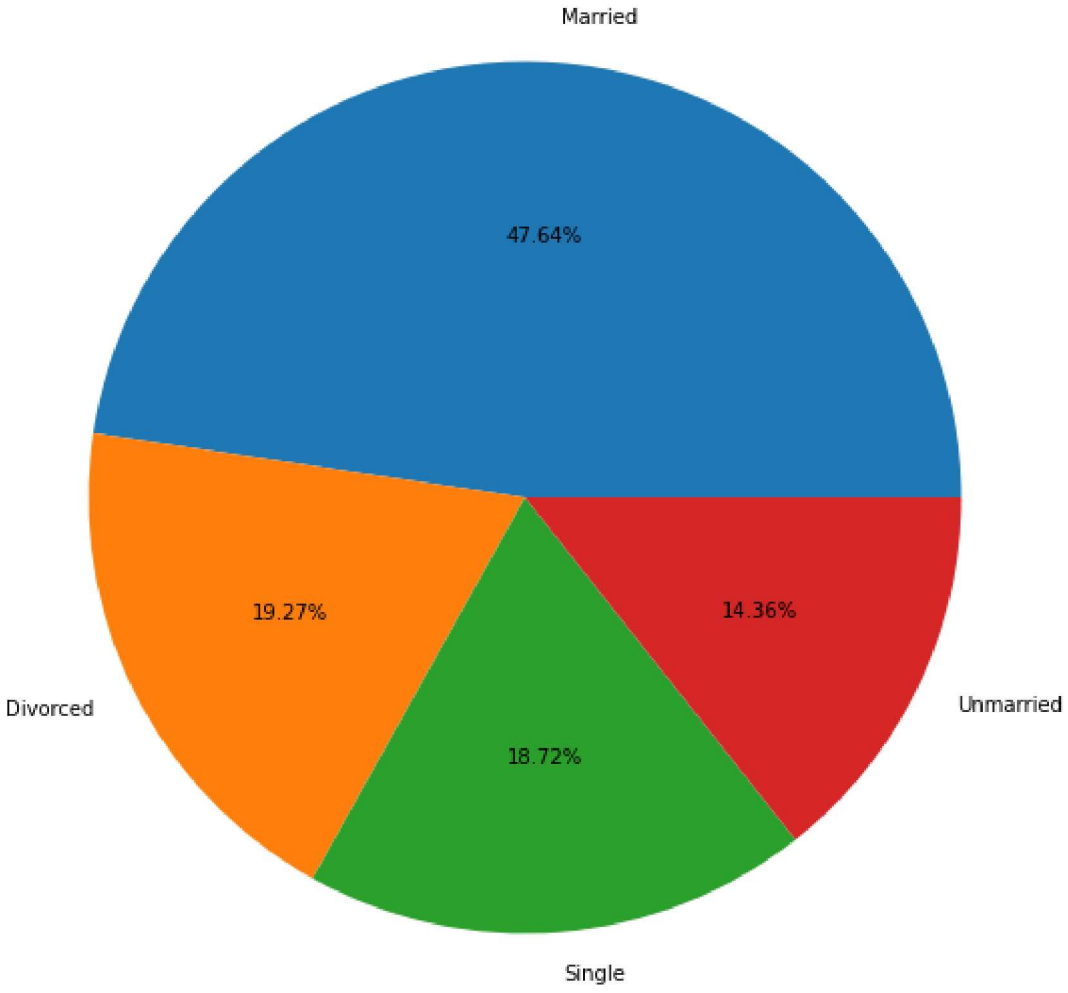
Observation - Married people travelled most

In [363…

```
plt.figure(figsize=(10,10))
plt.pie(data=MaritalStatus_trips,x='NumberOfTrips',labels='MaritalStatus',autopct="%
```

Out[363…

```
([<matplotlib.patches.Wedge at 0x1e64096f3a0>,
  <matplotlib.patches.Wedge at 0x1e64096fb50>,
  <matplotlib.patches.Wedge at 0x1e6409782b0>,
  <matplotlib.patches.Wedge at 0x1e6409789d0>],
 [Text(0.08144267680282738, 1.0969808979171836, 'Married'),
  Text(-0.9870155787798027, -0.4855926762688777, 'Divorced'),
  Text(0.08797385779808073, -1.0964764476923903, 'Single'),
  Text(0.9898898501048947, -0.479706248301301, 'Unmarried')],
 [Text(0.04442327825608766, 0.5983532170457364, '47.64%'),
  Text(-0.5383721338798924, -0.2648687325102969, '19.27%'),
  Text(0.04798574061713494, -0.5980780623776674, '18.72%'),
  Text(0.5399399182390334, -0.26165795361889144, '14.36%')])
```



Observation - 47.64% of people who travelled are married

Let's now find out how many people who are married, travelled with thier childrens

In [70]:
```
married = data[data['MaritalStatus']=='Married'][['MaritalStatus','NumberOfChildrenV
```

In [71]:
```
married.head()
```

Out[71]:

	MaritalStatus	NumberOfChildrenVisiting
0	Married	1.0
1	Married	1.0
2	Married	2.0
3	Married	1.0
4	Married	0.0

In [72]:
```
married_children=married.groupby('NumberOfChildrenVisiting').count().reset_index().r
```

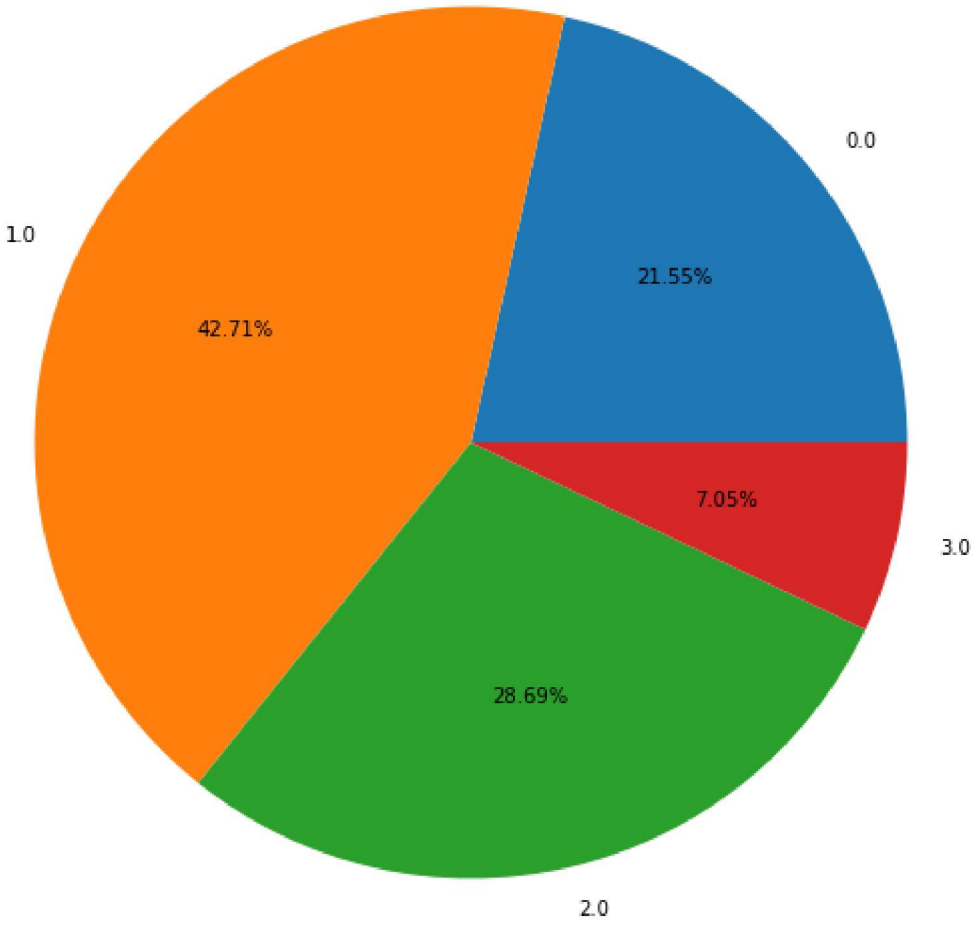
In [73]:
```
married_children
```

Out[73]:

	Children	Count
0	0.0	498
1	1.0	987
2	2.0	663
3	3.0	163

In [75]:
```
plt.figure(figsize=(10,10))
plt.pie(data=married_children,x='Count',labels='Children',autopct="%1.2f%%")
```

Out[75]:
```
([<matplotlib.patches.Wedge at 0x1e637166610>,
  <matplotlib.patches.Wedge at 0x1e637166ca0>,
  <matplotlib.patches.Wedge at 0x1e63717f3d0>,
  <matplotlib.patches.Wedge at 0x1e63717faf0>],
 [Text(0.8574112480933711, 0.6890906701174874, '0.0'),
  Text(-0.992451858614569, 0.47438308183628125, '1.0'),
  Text(0.2468551977883388, -1.0719433340083235, '2.0'),
  Text(1.073105697783523, -0.24175227275982025, '3.0')],
 [Text(0.4676788625963842, 0.37586763824590214, '21.55%'),
  Text(-0.5413373774261284, 0.2587544082743352, '42.71%'),
  Text(0.13464828970273024, -0.5846963640045401, '28.69%'),
  Text(0.5853303806091943, -0.13186487605081104, '7.05%')])
```



Observation

42.71% of married people travelled with 1 children

and 28.69% of married people travelled with 2 children

Conclusion - Married people presumably between age group 30-38 are the most preffered travellers

Duration of pitch vs satisfaction score

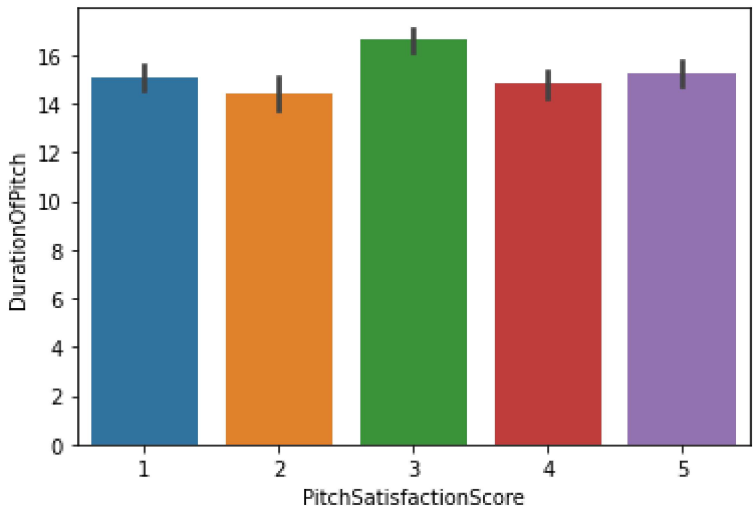
In [77]:
```
data.columns
```

Out[77]:
```
Index(['CustomerID', 'ProdTaken', 'Age', 'TypeofContact', 'CityTier',
       'DurationOfPitch', 'Occupation', 'Gender', 'NumberOfPersonVisiting',
       'NumberOfFollowups', 'ProductPitched', 'PreferredPropertyStar',
       'MaritalStatus', 'NumberOfTrips', 'Passport', 'PitchSatisfactionScore',
       'OwnCar', 'NumberOfChildrenVisiting', 'Designation', 'MonthlyIncome'],
      dtype='object')
```

In [367…
```
# Since duration of pitch have two outliers, we kept the pitch time below 100
data_pitch = data[data['DurationOfPitch']<100]
```

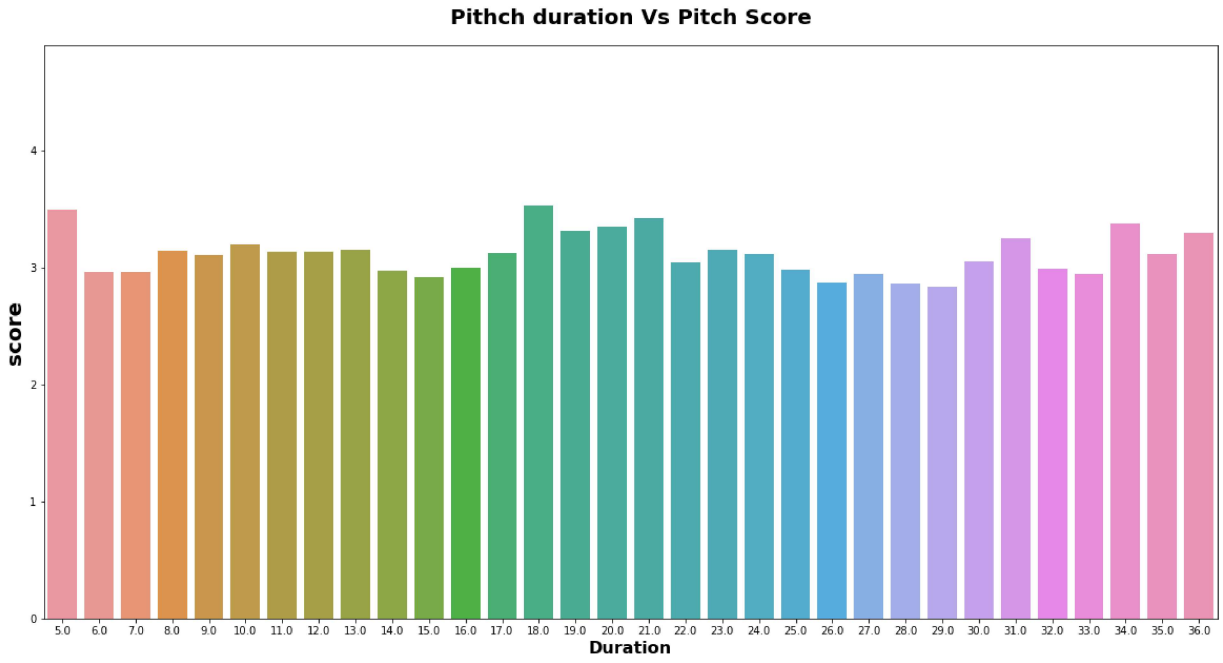
In [368…
```
sns.barplot(y=data_pitch['DurationOfPitch'],x=data_pitch['PitchSatisfactionScore'])
```

Out[368…
```
<AxesSubplot:xlabel='PitchSatisfactionScore', ylabel='DurationOfPitch'>
```



Highest rating 5 have around 15 minutes of pitch time, lets investigate further

In [369…
```
# bar plot to find out mean rating with pitch categories
plt.figure(figsize=(20,10))
sns.barplot(x=data_pitch['DurationOfPitch'],y=data_pitch['PitchSatisfactionScore'],e
plt.title("Pithch duration Vs Pitch Score", weight="bold",fontsize=20, pad=20)
plt.ylabel("score", weight="bold", fontsize=20)
plt.xlabel("Duration", weight="bold", fontsize=16)
plt.show()
```



Observations - 5 minutes,18 minutes and 34 minutes shows most average ratings

Let's make some more analysis

In [372…
```
#dataframe for DurationOfPitch and PitchSatisfactionScore
f = data_pitch[['DurationOfPitch','PitchSatisfactionScore']]
```

In [162…
```
# duplicated column PitchSatisfactionScore
f['Pitch'] = f['PitchSatisfactionScore']
```

In [370…
```
f.head()
```

Out[370…

	DurationOfPitch	PitchSatisfactionScore	Pitch
0	6.0	2.0	2.0
1	14.0	3.0	3.0
2	8.0	3.0	3.0
3	9.0	5.0	5.0
4	8.0	5.0	5.0

In [262…
```
# Used pivot table to categorize the data to show count of ratings per rating
s=f.pivot_table(index="DurationOfPitch",columns="PitchSatisfactionScore",values="Pit
```

In [375…
```
#renamed the index column
s.rename_axis("index").head()
```

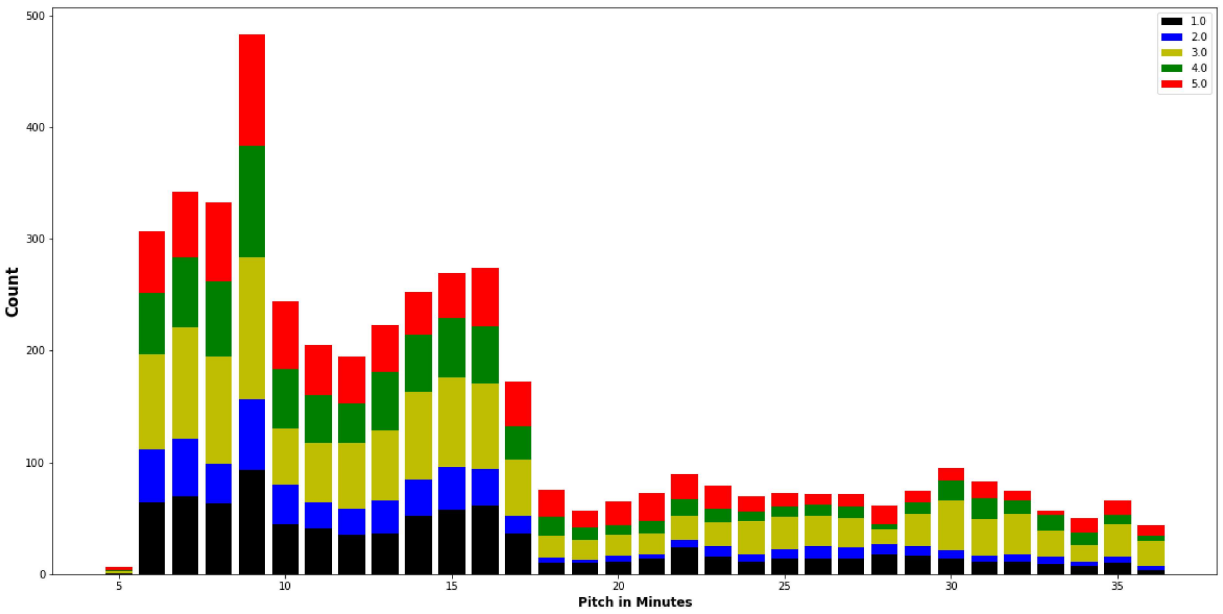
Out[375…

PitchSatisfactionScore	1.0	2.0	3.0	4.0	5.0
index					
5.0	1	0	2	1	2

PitchSatisfactionScore	1.0	2.0	3.0	4.0	5.0
index					
6.0	64	48	85	55	55
7.0	70	51	100	62	59
8.0	63	36	96	67	71
9.0	93	64	126	100	100

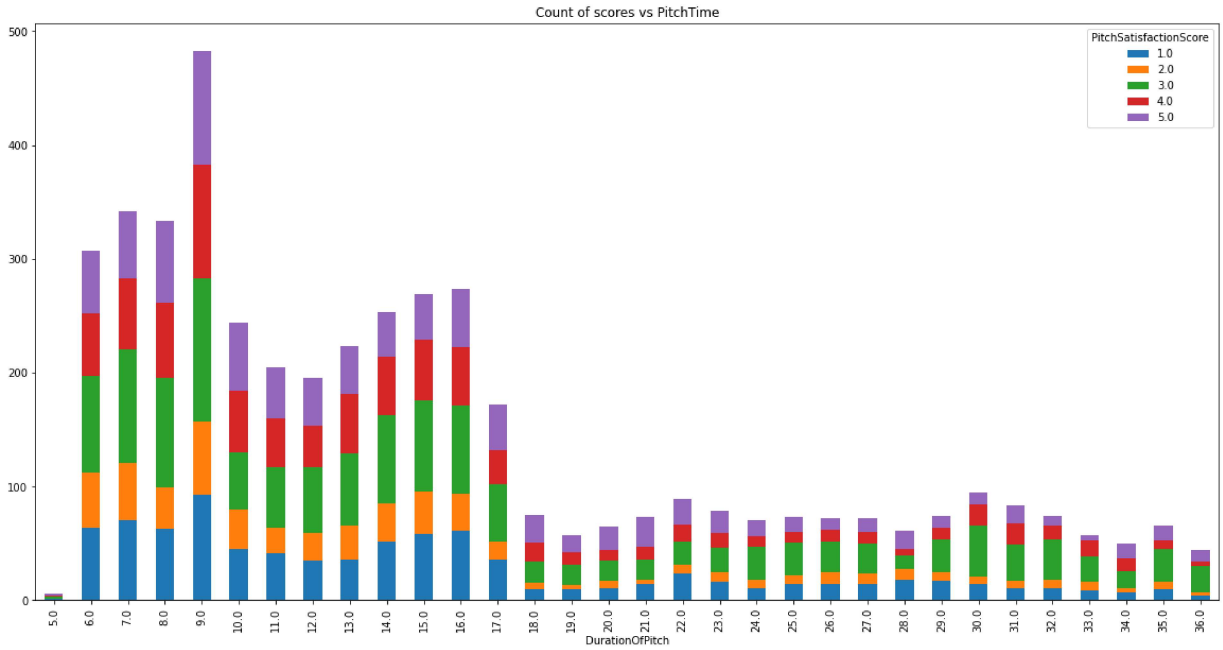
In [376…

```
# Stacked Bar Plot to show the count of ratings per pitch duration
plt.figure(figsize=(20,10))
plt.bar(s.index.values, s[1], color='black')
plt.bar(s.index.values, s[2], bottom=s[1], color='b')
plt.bar(s.index.values, s[3], bottom=s[1]+s[2], color='y')
plt.bar(s.index.values, s[4], bottom=s[1]+s[2]+s[3], color='g')
plt.bar(s.index.values, s[5], bottom=s[1]+s[2]+s[3]+s[4], color='r')
plt.legend(s.columns)
plt.ylabel("Count", weight="bold", fontsize=15)
plt.xlabel("Pitch in Minutes", weight="bold", fontsize=12)
plt.show()
```



In [377…

```
# same stacked bar plot using dataframe plots
s.plot(kind='bar', stacked=True,title='Count of scores vs PitchTime',figsize=(20,10)
plt.show()
```



Observations - Pitch of 9 minutes have got most ratings, with most top 5 ratings. Hence optimal pitch timings can be concluded to be 9 minutes

We can infer more insights on the dataset..This is entirely for learning purpose, feature enginnering to be continued in further works

In [ ]: