visualization

February 27, 2020

##
Building Visualization With Python

This notebook showa all visualizations that can be made using Python. This can be a good resource to learn and play with python and its available and libraries.

The dataset used in this notebook is IBM's HR dataset which is free to download from Kaggle.com

Download the IBM HR dataset from here

After downloading the dataset to your personal computer, upload it into your Jupyter project folder.

###
Initial Preparation

####
First import the libraries necessary to build visualization.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
print('Done')
```

```
/home/nbuser/anaconda3_501/lib/python3.6/site-
packages/matplotlib/font_manager.py:229: UserWarning: Matplotlib is building the
font cache using fc-list. This may take a moment.
  'Matplotlib is building the font cache using fc-list. '

Done
```

####
Load the HR dataset as pandas data frame.

[2]:
```
df = pd.read_csv ("IBM_HR.csv")
```

####
Have a look at your columns heading and types of data within the dataset.

[3]:
```
df.head(5)
```

[3]:
```
   Age Attrition     BusinessTravel  DailyRate              Department  \
0   41       Yes      Travel_Rarely       1102                   Sales
1   49        No  Travel_Frequently        279  Research & Development
2   37       Yes      Travel_Rarely       1373  Research & Development
3   33        No  Travel_Frequently       1392  Research & Development
4   27        No      Travel_Rarely        591  Research & Development

   DistanceFromHome  Education EducationField  EmployeeCount  EmployeeNumber  \
0                 1          2  Life Sciences              1               1
1                 8          1  Life Sciences              1               2
2                 2          2          Other              1               4
3                 3          4  Life Sciences              1               5
4                 2          1        Medical              1               7

            ...           RelationshipSatisfaction StandardHours  \
0           ...                                  1            80
1           ...                                  4            80
2           ...                                  2            80
3           ...                                  3            80
4           ...                                  4            80

   StockOptionLevel  TotalWorkingYears  TrainingTimesLastYear WorkLifeBalance  \
0                 0                  8                      0               1
1                 1                 10                      3               3
2                 0                  7                      3               3
3                 0                  8                      3               3
4                 1                  6                      3               3

   YearsAtCompany YearsInCurrentRole  YearsSinceLastPromotion  \
0               6                  4                        0
1              10                  7                        1
2               0                  0                        0
3               8                  7                        3
4               2                  2                        2

   YearsWithCurrManager
0                     5
1                     7
2                     0
3                     0
4                     2

[5 rows x 35 columns]
```

###
Pie Chart
####
To see the number of unique values (different categories) in the department column first must

perform some preprocessing

[4]:
```
df['Department'].unique()
```

[4]:
```
array(['Sales', 'Research & Development', 'Human Resources'], dtype=object)
```

####

Convert string values to numeric values

[5]:
```
def label_to_numeric(x):
    if x=='Sales':
        return 1
    if x=='Research & Development':
        return 2
    if x=='Human Resources':
        return 3

df['Department'] = df['Department'].apply(label_to_numeric)
```

####

Check the result

[6]:
```
df['Department'].unique()
```

[6]:
```
array([1, 2, 3])
```

####

Check the datatype of the column

[7]:
```
df['Department'].dtypes
```

[7]:
```
dtype('int64')
```

####

Convert the datatype from int to category

[52]:
```
df['Department'] = df['Department'].astype('category', copy=False)
```

####

Now your variable is ready and prepared for visualization

Pie chart is used to illustrate numerical proportion. The point of a pie chart is to show the relationship of parts out of a whole. For example comparison between a percentiles from 100%.

####

First create an object (dpartmentgrouped) from department and use groupby function to calculate the sum of employees in each department. Then use this object to build the chart

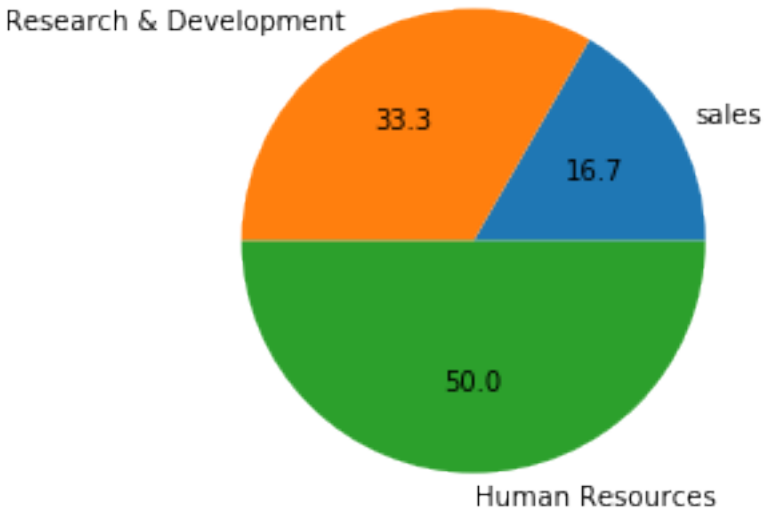
[53]:
```
dpartmentgrouped = df.groupby('Department', as_index=False).sum()
```

####

To make the chart more comprehensible, define lables for each categories (thus instead of 1,2,3 the chart will show exact names of each department) and run the pie plot code

[54]:
```
LABELS = 'sales', 'Research & Development', 'Human Resources'
plt.pie(dpartmentgrouped['Department'],
labels=LABELS,
autopct='%.1f')
```

```
plt.show()
```



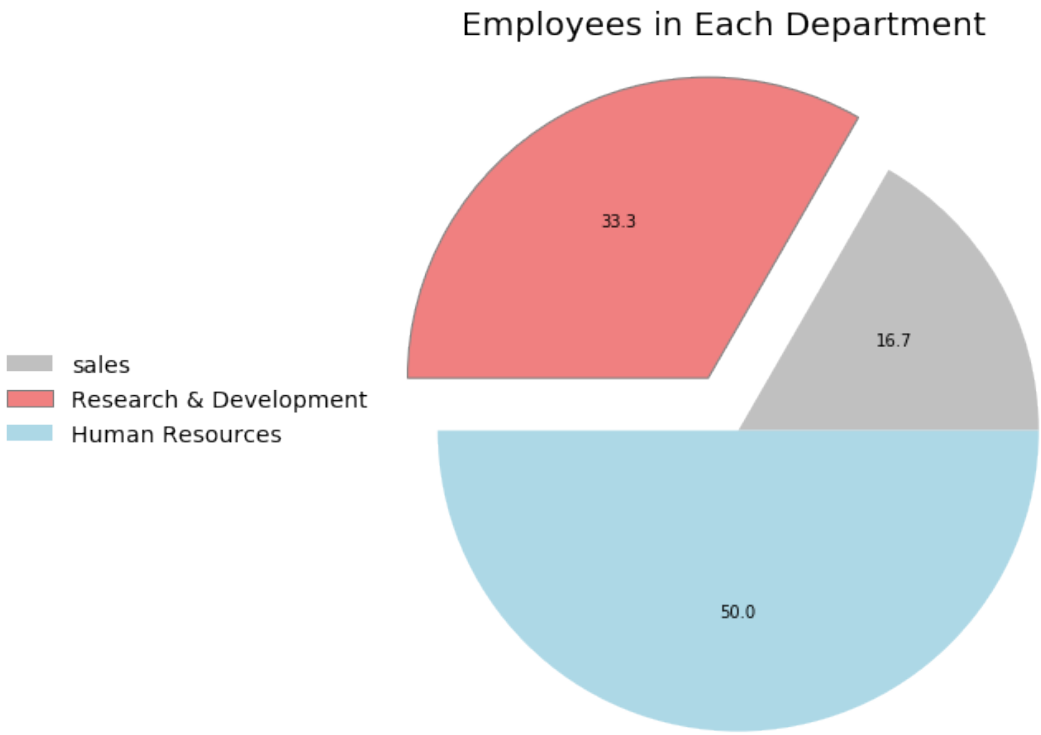
####

The same pie chart can be modified by adding legend, defining it's sizes, defining a part to explode, and adding colors to labels.

[149]:
```
COLORS = ['silver', 'lightcoral', 'lightblue']
LABELS = 'sales', 'Research & Development', 'Human Resources'
EXPLODE = (0, 0.2, 0)

fig, ax = plt.subplots(figsize=(8,8))
wedges, texts, autotexts = plt.pie(dpartmentgrouped['Department'],
autopct='%.1f', colors=COLORS, explode=EXPLODE)

wedges[1].set(edgecolor="gray", linewidth=1)
plt.title('Employees in Each Department', loc='center', fontsize=20)
plt.legend(LABELS, loc='upper left', fontsize=14, frameon=False, bbox_to_anchor=␣
 ↪(-0.5, 0.6))
plt.axis('equal')
plt.show()
```

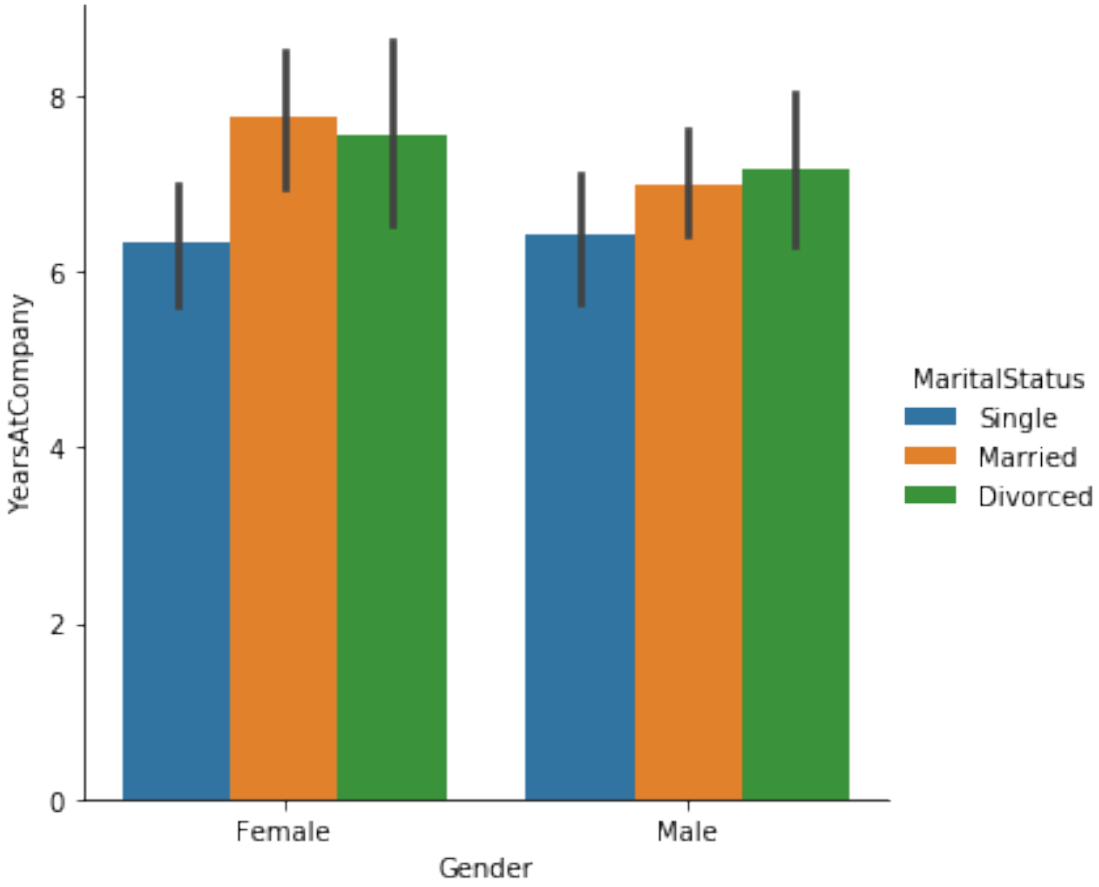


###
Bar Chart (Catplot)

Catplot is best used for categorical data to show frequency information. Bar plots help to easily see the differences between the categories based on the size of the bar. categories are also easily divided and colour coded too.

[192]:
```
sns.catplot(x= 'Gender', y= 'YearsAtCompany', data=df,
              hue='MaritalStatus',
            kind='bar')
```

[192]:
```
<seaborn.axisgrid.FacetGrid at 0x7f4cf40f00b8>
```

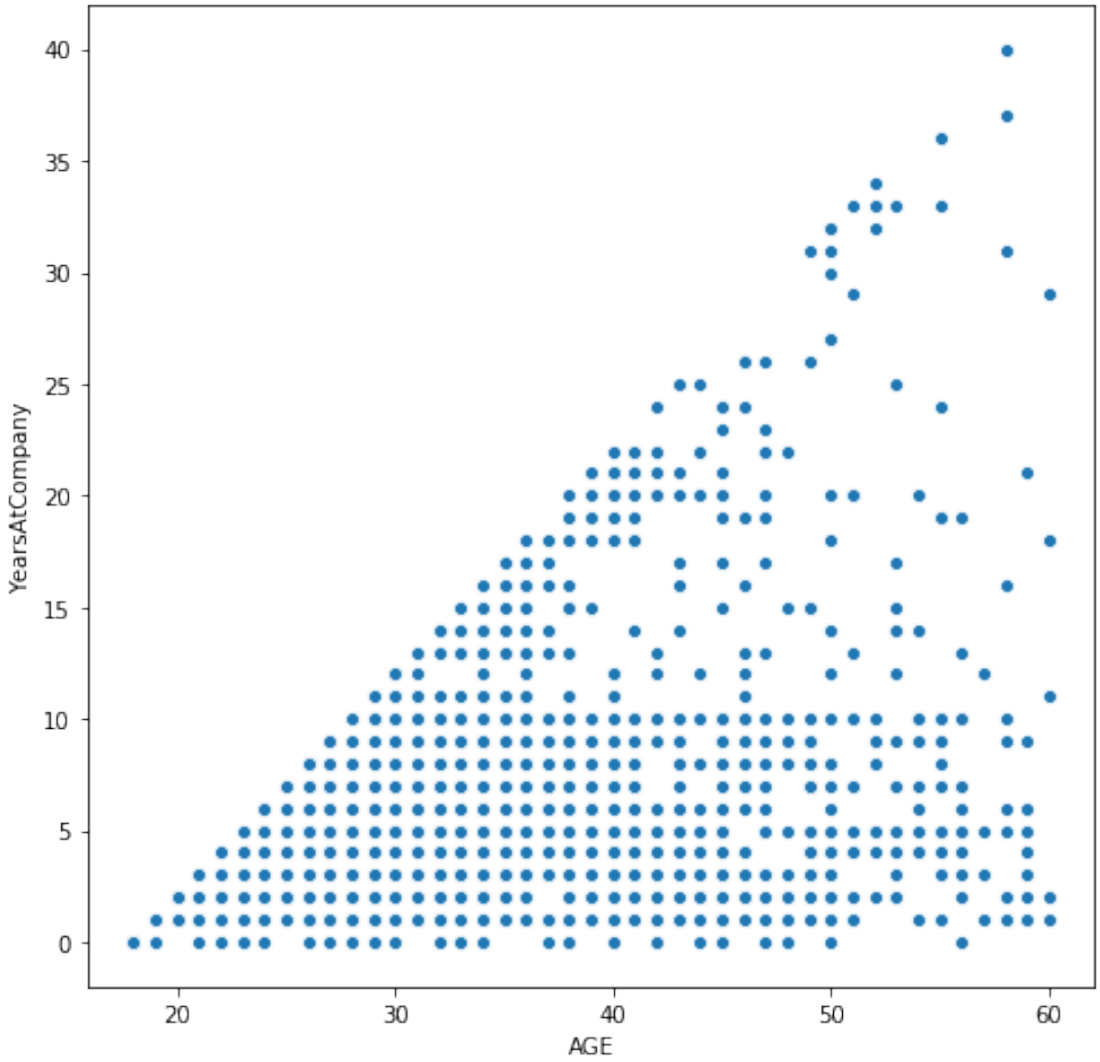


###
Scatterplot

plot data points on a horizontal and a vertical axis in the attempt to show how much one variable is affected by another.

[184]:
```
plt.figure(figsize=(8, 8))
sns.scatterplot(x='Age', y='YearsAtCompany', data=df)
plt.xlabel('AGE')
plt.ylabel('YearsAtCompany')
plt.show
```

[184]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



#####

The same scatter plot can be modified and improved by defining hue, color and size of bubble.

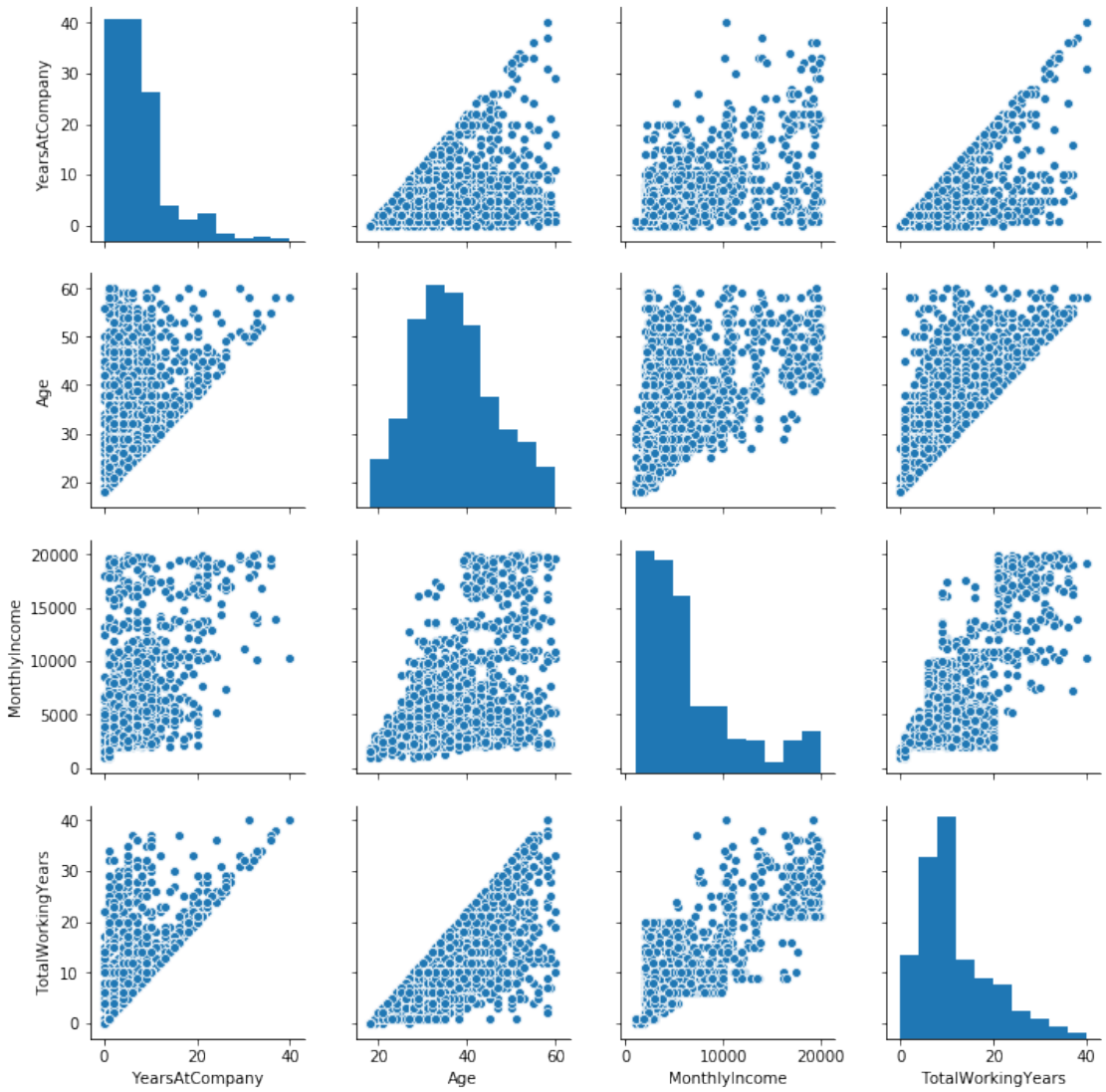
[297]:
```
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Age', y='YearsAtCompany',
                hue='Gender',
                size='Gender',
                sizes=(30, 300), hue_norm=(0, 7), legend="full",
                data=df, palette="Set2")
plt.xlabel('Age')
plt.ylabel('Years At Company')
plt.show
```

[297]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```

###
Pair Plot
#####
The relationship between all variables are analyzed and illustrated in pair.

[207]:
```
sns.pairplot(df1)
```

[207]:
```
<seaborn.axisgrid.PairGrid at 0x7f4cfe659828>
```

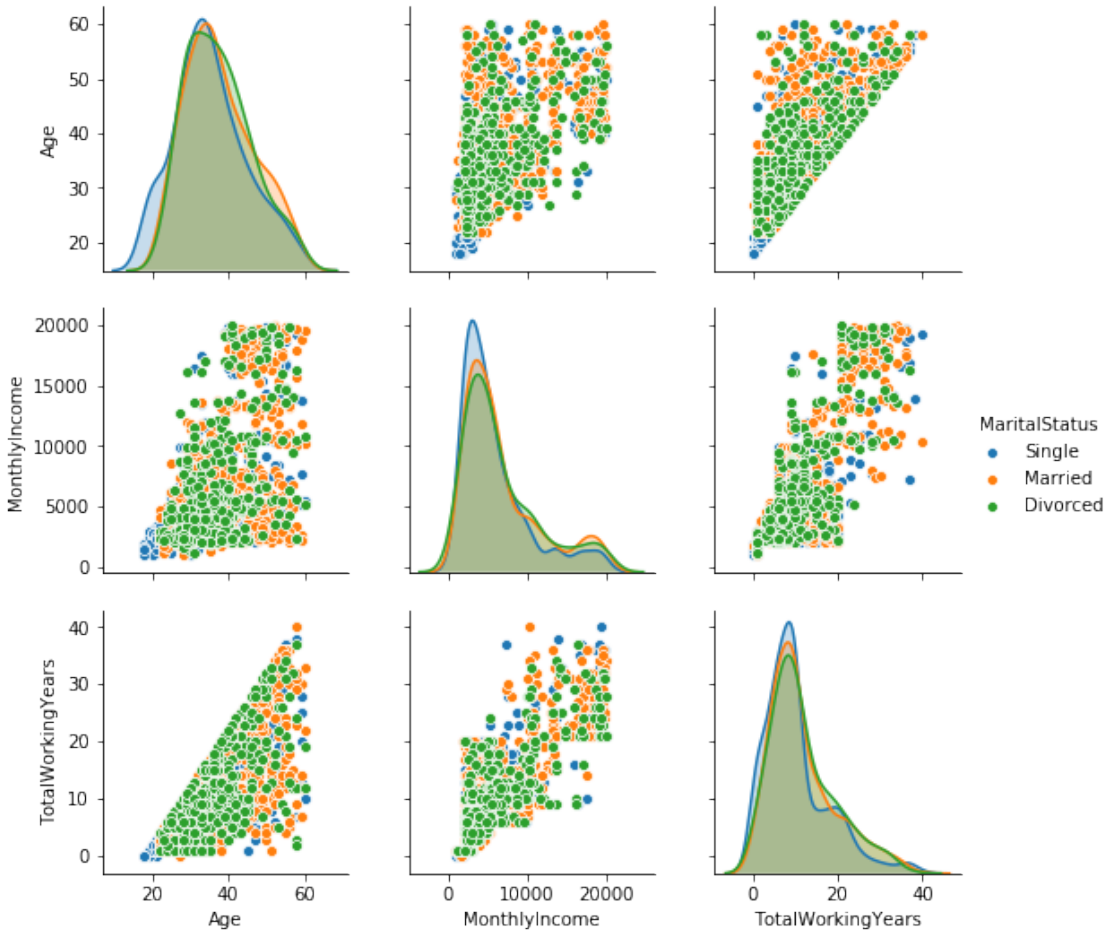


####

The color of data points can be defined based on additional variable.

[234]:
```
sns.pairplot(df, vars=['Age', 'MonthlyIncome', 'TotalWorkingYears'],␣
↪hue='MaritalStatus')
```

[234]:
```
<seaborn.axisgrid.PairGrid at 0x7f4cda099e10>
```



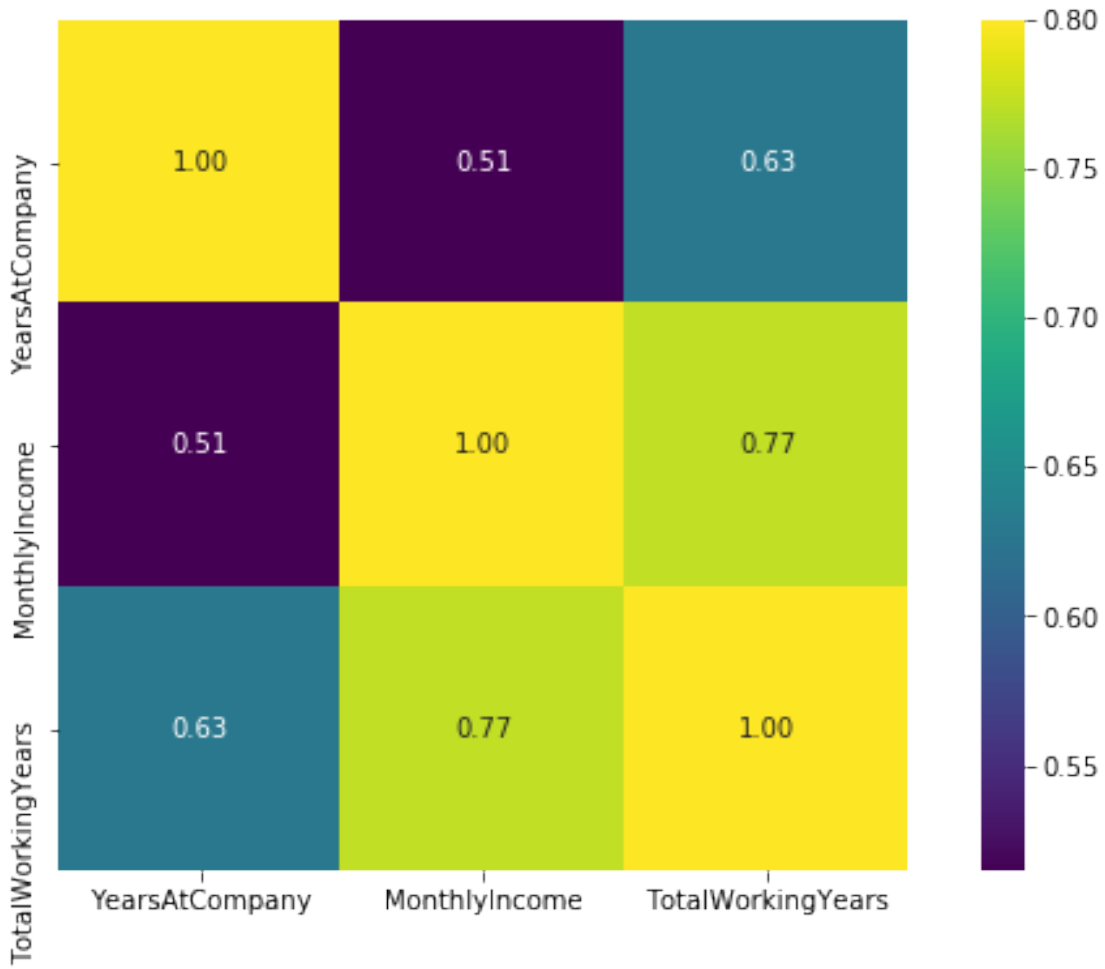
###
Heat map

All values are illustrated in a matrix and represented as colors. ####

First create an object (sub dataframe) with all the variables that need to be included in correlation analysis and then create the plot.

[210]:
```
df2 = df[['YearsAtCompany', 'MonthlyIncome', 'TotalWorkingYears']]
```

[215]:
```
corrmat = df2.corr()
f, ax = plt.subplots(figsize=(10, 6))
sns.heatmap(corrmat, vmax=.8, square=True, annot=True, fmt='.2f', cmap =␣
↪'viridis')
plt.show()
```



###
Stripplot

Shows all the observations within a variable while the locations of points are adjusted automatically to avoid overlap.

[213]:
```
plt.figure(figsize=(8, 6))
sns.stripplot(x= 'Gender', y= 'MonthlyIncome', data=df, jitter=True)
```

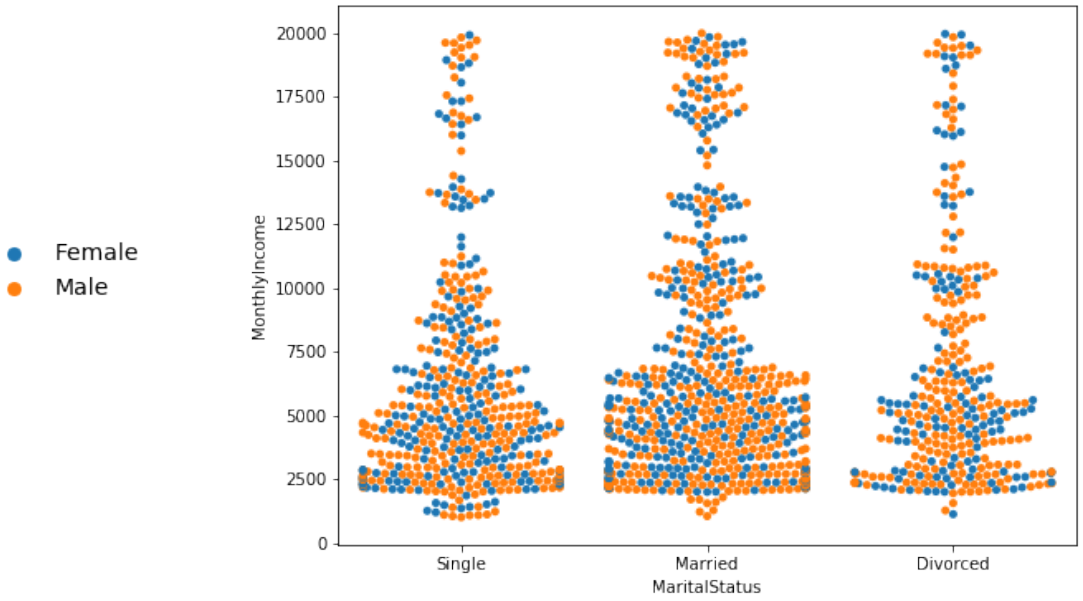
[213]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4cdb562048>
```

###
Swarmplots

Shows all the observations within a variable while the locations of points are adjusted automatically to avoid overlap.

[230]:
```
plt.figure(figsize=(8, 6))
sns.swarmplot(x= 'MaritalStatus', y= 'MonthlyIncome', hue='Gender',  data=df)
plt.legend(loc='upper left', fontsize=14, frameon=False, bbox_to_anchor= (-0.5,␣
↪0.6))
```

[230]:
```
<matplotlib.legend.Legend at 0x7f4cda182cc0>
```

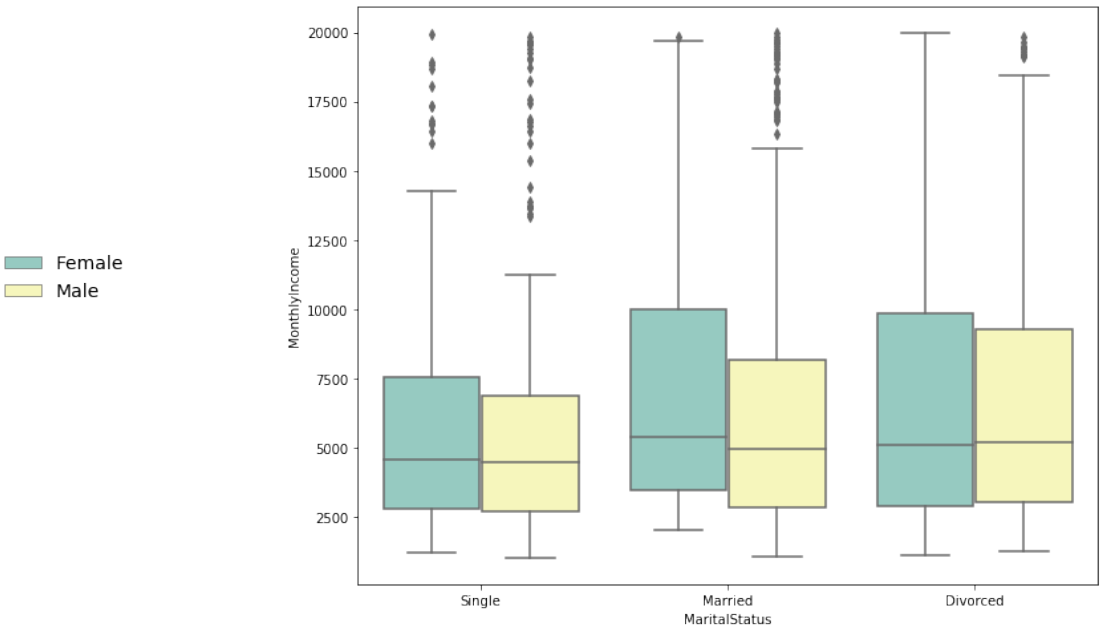


###

Boxplot

Boxplots visualize the distribution of variables and provide a clearer view of the standard deviation, median, mean, range of data, and outliers. The bottom and top of the solid-lined box are the first and third quartiles, and the band inside the box is the median. The whiskers extend from the box to show the range of the data.

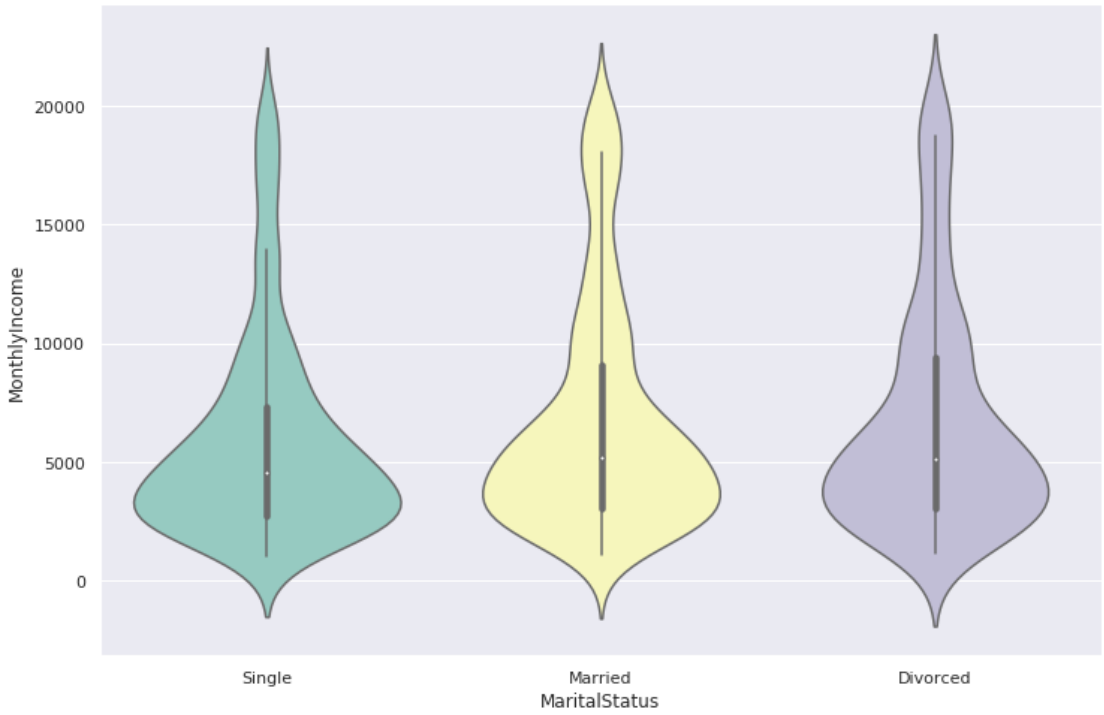
[232]:
```
f, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(x= 'MaritalStatus', y= 'MonthlyIncome', hue='Gender', data=df,␣
 ↪palette="Set3")
plt.legend(loc='upper left', fontsize=14, frameon=False, bbox_to_anchor= (-0.5,␣
 ↪0.6))
plt.show()
```



###
Violin Plot

Violin plot shows the distribution and range of the data

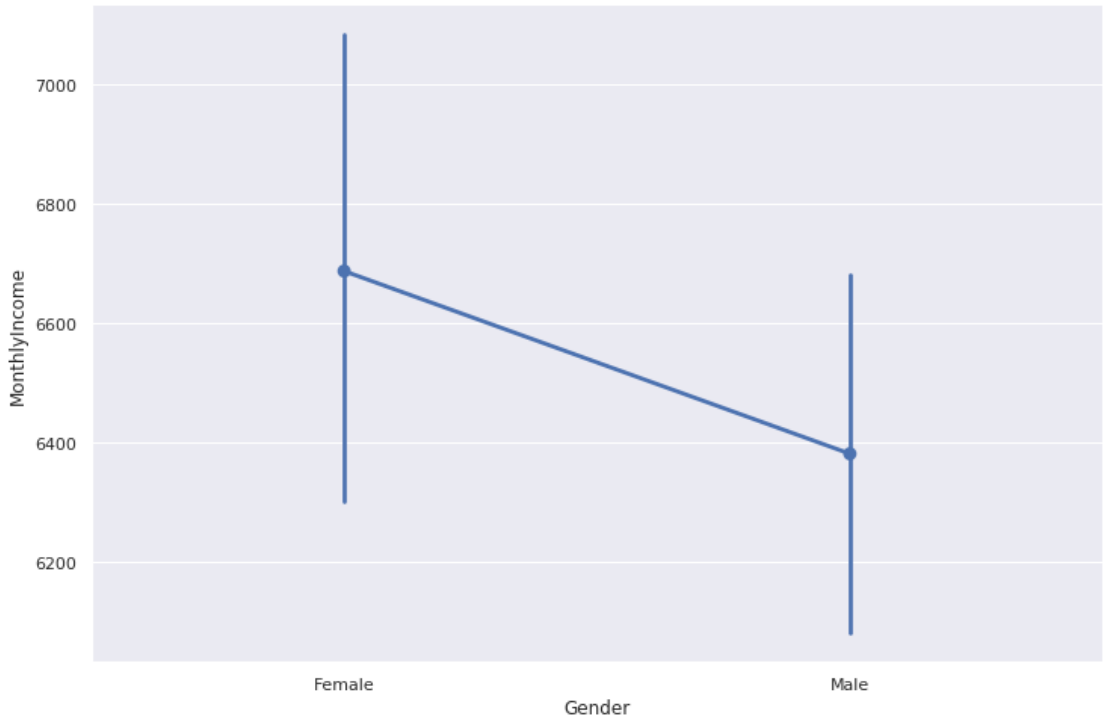
[248]:
```
f, ax = plt.subplots(figsize=(12, 8))
sns.violinplot(x= 'MaritalStatus', y= 'MonthlyIncome', data=df, palette="Set3")
plt.show()
```



###

Point Plot

[250]:
```
f, ax = plt.subplots(figsize=(12, 8))
sns.pointplot(x= 'Gender', y= 'MonthlyIncome', data=df)
plt.show()
```

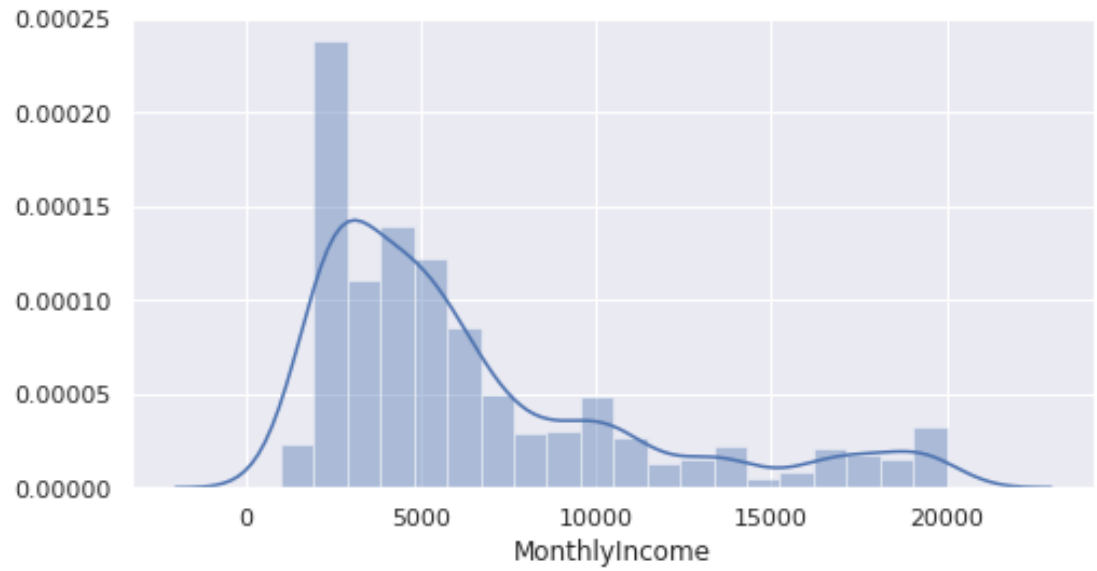


###

Distribution Plot with KDE line

[252]:
```
plt.figure(figsize=(8, 4))
sns.distplot(df['MonthlyIncome'])
plt.show
```

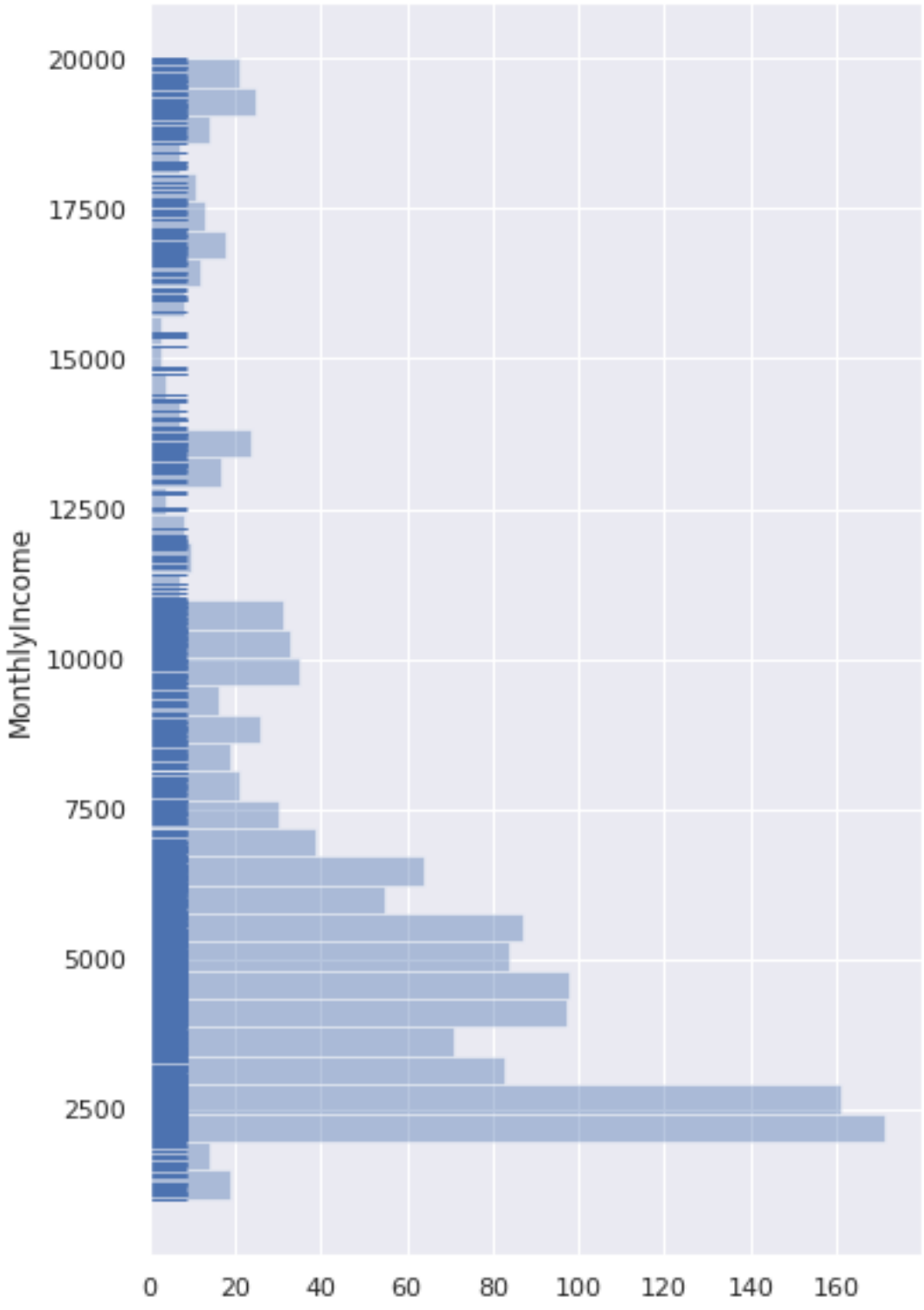
[252]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



The same distribution plot without Kernel Distribution Estimator line, vertical axis, and all individual datapoints. The data aso are devided to 40 bins

[254]:
```
plt.figure(figsize=(6, 10))
sns.distplot(df['MonthlyIncome'], kde=False, vertical=True, rug=True, bins=40)
plt.show
```

[254]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```


MonthlyIncome
20000
17500
15000
12500
10000
7500
5000
2500
0
20
40
60
80
100
120
140
160

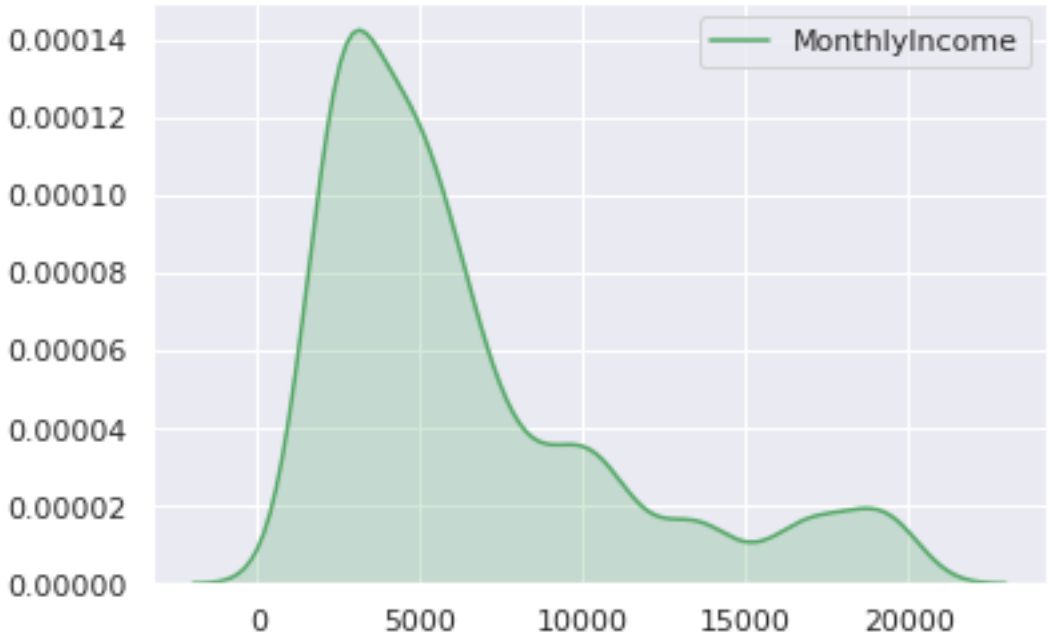
###
KDE Plot

KDE Plot described as Kernel Density Estimate is used for visualizing the Probability Density of a continuous variable. It depicts the probability density at different values in a continuous variable.

[241]:
```
plt.figure(figsize=(6, 4))

sns.set(color_codes=True)
sns.kdeplot(df['MonthlyIncome'], shade=True, color='g')
plt.show
```

[241]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```

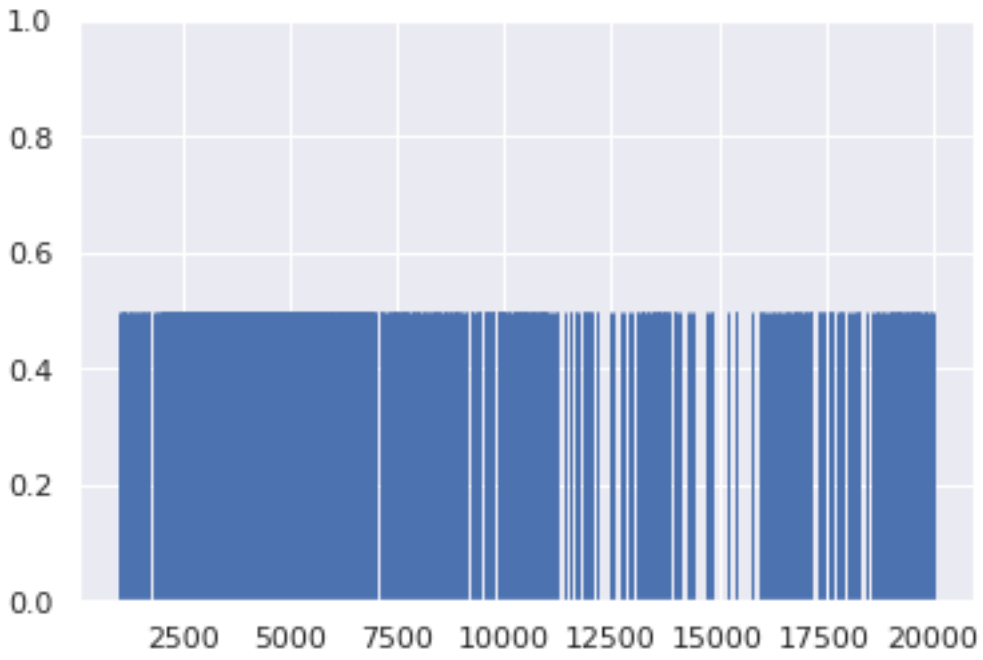


###
Rug Plot

Only sticks representing actual datapoints with height of 0.5 and shows the distribution of data

[244]:
```
plt.figure(figsize=(6, 4))
sns.rugplot(df['MonthlyIncome'], height=0.5)
plt.show
```

[244]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



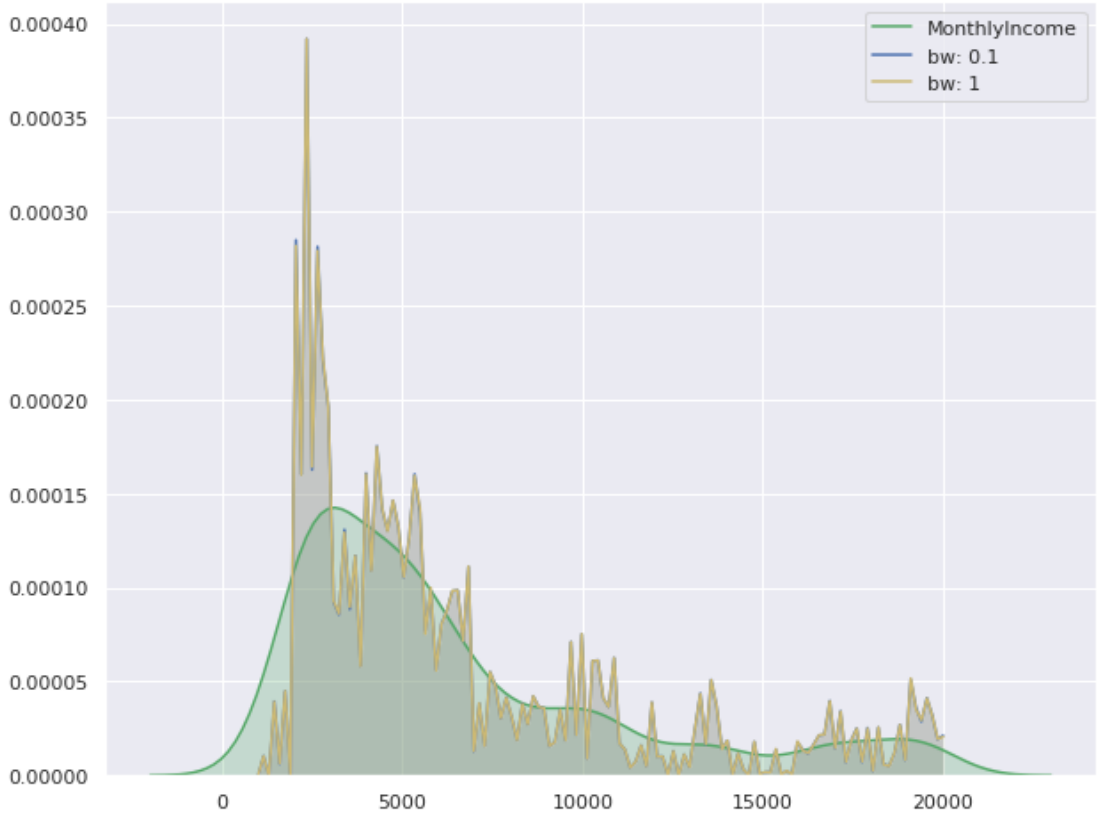
###
Multiple KDE plot

To configure the bandwith(std)

[245]:
```
plt.figure(figsize=(10, 8))

sns.set(color_codes=True)
sns.kdeplot(df['MonthlyIncome'], shade=True, color='g')
sns.kdeplot(df['MonthlyIncome'], bw=0.1, label='bw: 0.1', shade=True, color='b')
sns.kdeplot(df['MonthlyIncome'], bw=1, label='bw: 1', shade=True, color='y')
plt.show
```

[245]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```

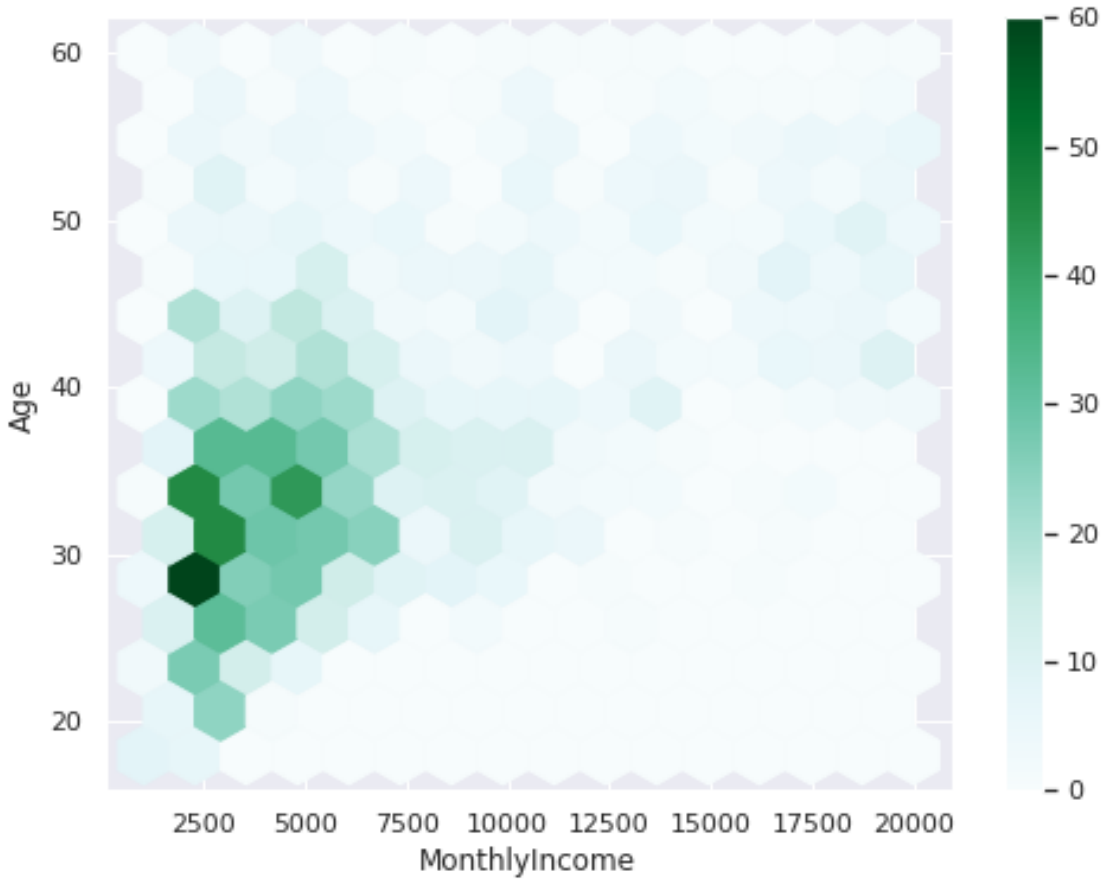


###
Hexbin Plot

The bivariate analougues are histograms

[265]:
```
df.plot.hexbin(x= 'MonthlyIncome', y= 'Age',
               gridsize=15, figsize=(8, 6), sharex=False)
plt.xlabel('MonthlyIncome')
plt.ylabel('Age')
plt.show
```

[265]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```

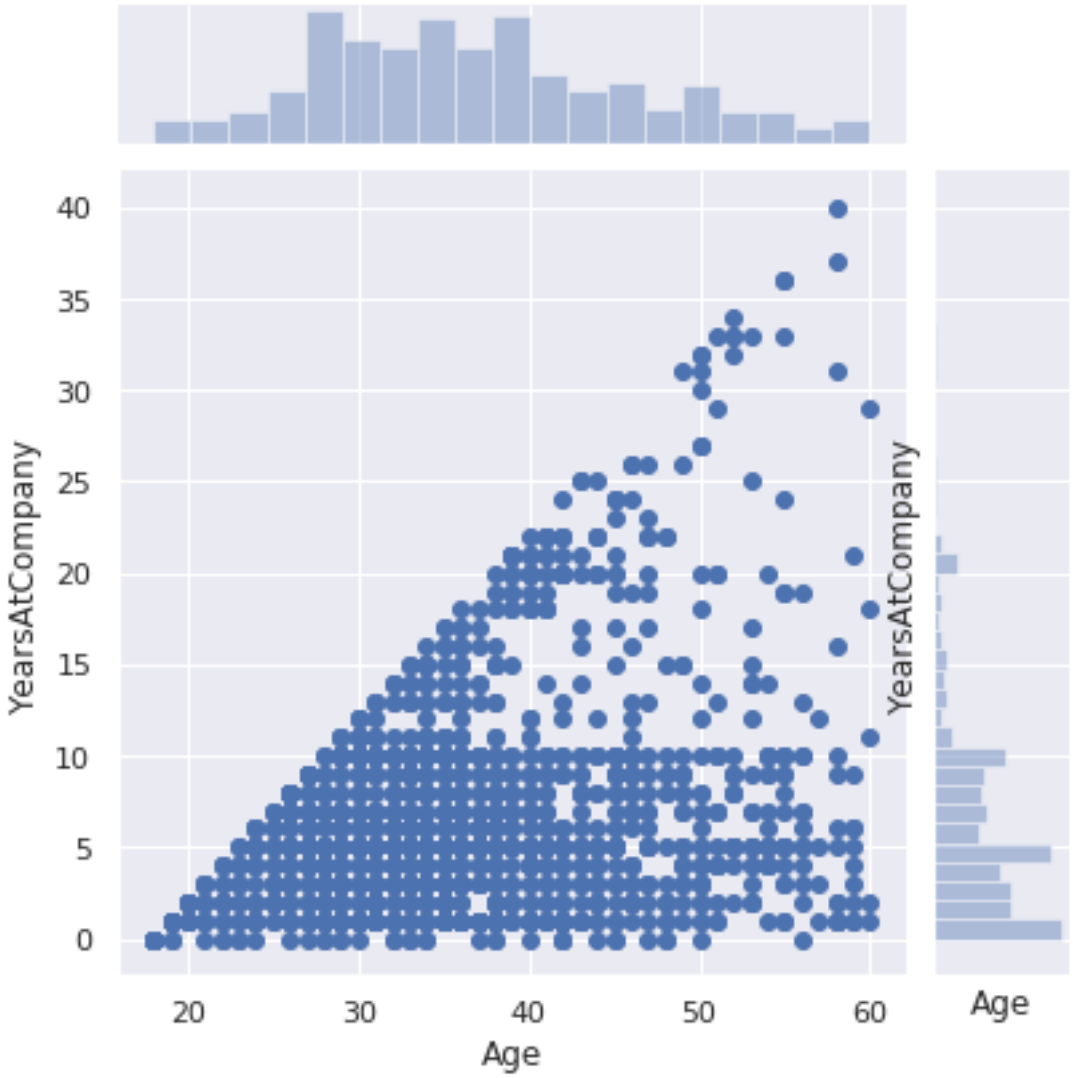


###
Joint Plot

Can join any plot of choice

[271]:
```
sns.jointplot(x='Age', y='YearsAtCompany', data=df)
plt.xlabel('Age')
plt.ylabel('YearsAtCompany')
plt.show
```

[271]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



###
Lmplot

Regression model with confidence interval line

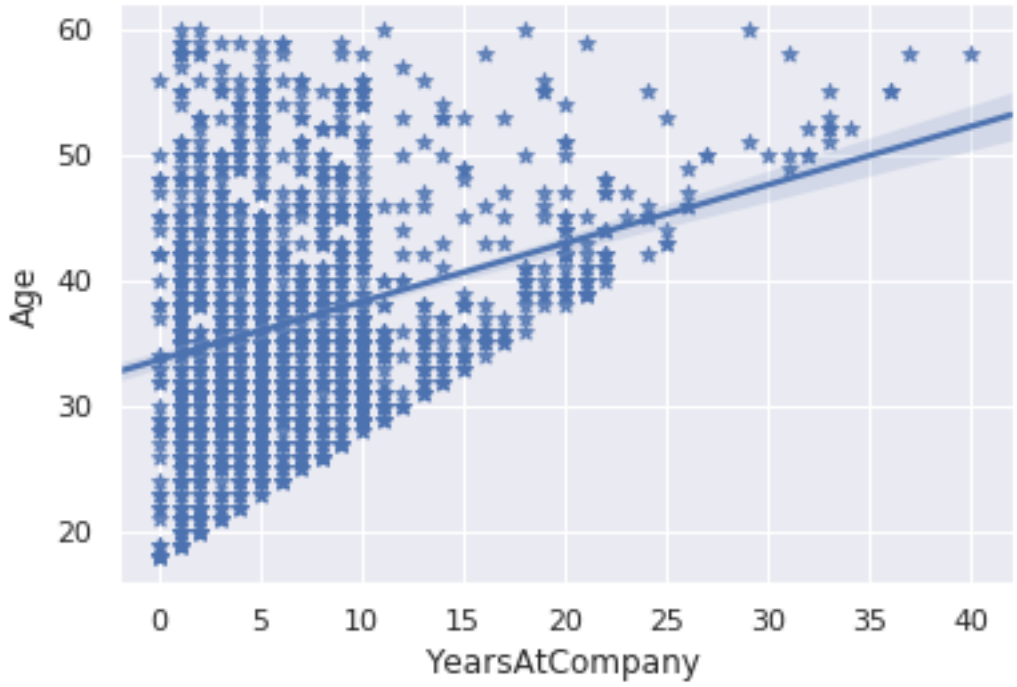
[294]:
```
sns.lmplot(x= 'YearsAtCompany', y= 'Age', hue='StockOptionLevel',␣
↪palette='Set3', aspect=1.5, data=df)
```

[294]:
```
<seaborn.axisgrid.FacetGrid at 0x7f4cdac83978>
```

###
Regplot

The regplot and lmplot functions are closely related, but regplot is an axes-level function while the lmplot is a figure-level function.

[295]:
```
sns.regplot(df['YearsAtCompany'], df['Age'], color='b', marker='*')
plt.show()
```



###
MAP

[102]:
```
import plotly.graph_objs as go
print('Done!')
```

```
Done!
```

[127]:
```
import plotly.graph_objs as go
import plotly.offline as offline
offline.init_notebook_mode(connected=True)
print('Done!')
```

```
Done!
```

[162]:
```
trace = dict (type = 'scattergeo',
              lon = [-122.1, -77.2, 51.3],
              lat = [37.4, 38.9, 35.6],
              marker = dict(size = 10),
              mode = 'markers',
             )
```

[169]:
```
data = [trace]
```

```
layout = dict(showlegend = False, geo = dict(showland = True, landcolor =␣
↪'silver'))
```

[170]:
```
fig = dict(data = data, layout = layout)
offline.iplot(fig)
```