7.2-Data_Visualization_using_Seaborn

KeytoDataScience.com

Seaborn

- Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- Seaborn is used for more complex visualizations
- Built on matplotlib and works best with pandas dataframes
- Visualization is the central part of Seaborn which helps in exploration and understanding of data.

Table of Contents

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

In [2]:
```
# supress warnings
import warnings
warnings.filterwarnings("ignore")
```

1 Univariate Analysis

In [3]:
```
# Get the current directory
import os
os.getcwd()
```

Out[3]:
```
'F:\\Work\\Site\\KDS - Career Now Program\\DS\\Syllabus\\1. Programming\\3. Python\\Pyth
on\\Module 7 - Data Visualization\\Reference Materials'
```

In [4]:
```
# csv files are stored at Input/Seaborn folder
input_files=os.getcwd()+"/Input/Seaborn/"
```

In [5]:
```
grant_file=pd.read_csv(input_files+"schoolimprovement2010grants.csv")
grant_file.head(5)
```

Out[5]:

	Unnamed: 0	School Name	City	State	District Name	Model Selected	Award_Amount	Region
0	0	HOGARTH KINGEEKUK MEMORIAL SCHOOL	SAVOONGA	AK	BERING STRAIT SCHOOL DISTRICT	Transformation	471014	West
1	1	AKIACHAK SCHOOL	AKIACHAK	AK	YUPIIT SCHOOL DISTRICT	Transformation	520579	West
2	2	GAMBELL SCHOOL	GAMBELL	AK	BERING STRAIT SCHOOL DISTRICT	Transformation	449592	West
3	3	BURCHELL HIGH SCHOOL	WASILLA	AK	MATANUSKA-SUSITNA BOROUGH SCHOOL DISTRICT	Transformation	641184	West
4	4	AKIAK SCHOOL	AKIAK	AK	YUPIIT SCHOOL DISTRICT	Transformation	399686	West

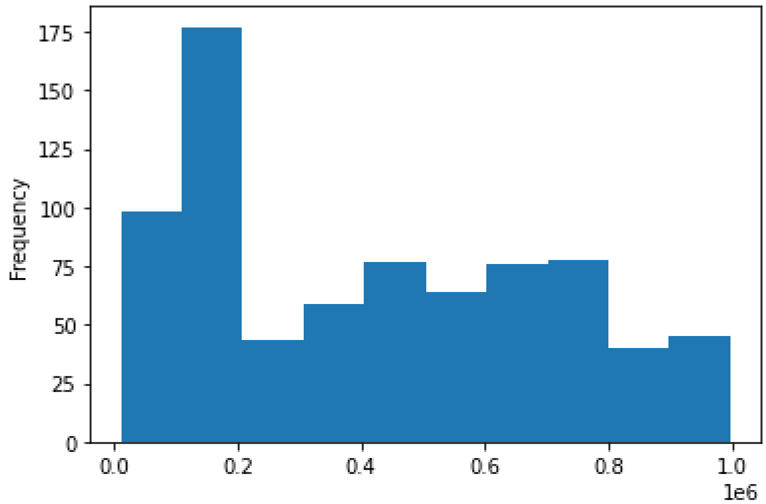
1.1 Pandas Histogram vs Seaborn Distplot

In [6]:

```
# Displays a pandas histogram
grant_file["Award_Amount"].plot.hist()
```

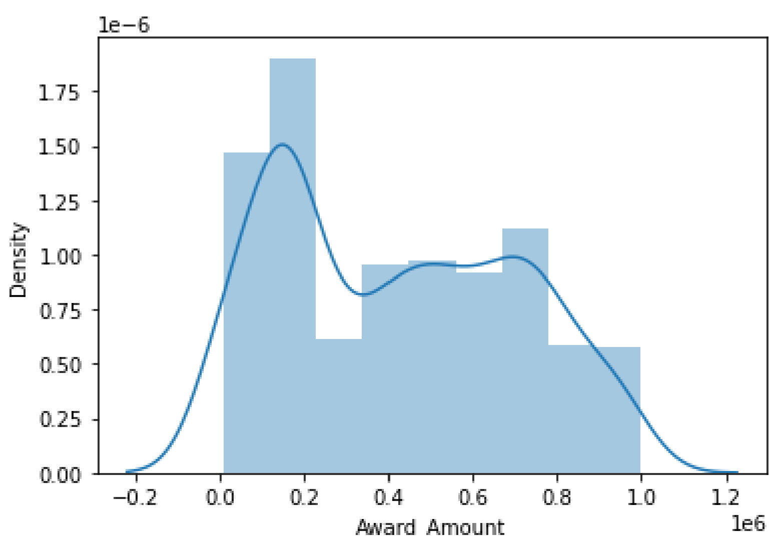
Out[6]:

```
<AxesSubplot:ylabel='Frequency'>
```



In [7]:

```
# Clear the histogram
plt.clf()

# Display a Seaborn distplot
sns.distplot(grant_file['Award_Amount'])
plt.show()

# Clear the pevious plot
plt.clf()
```




```
<Figure size 432x288 with 0 Axes>
```

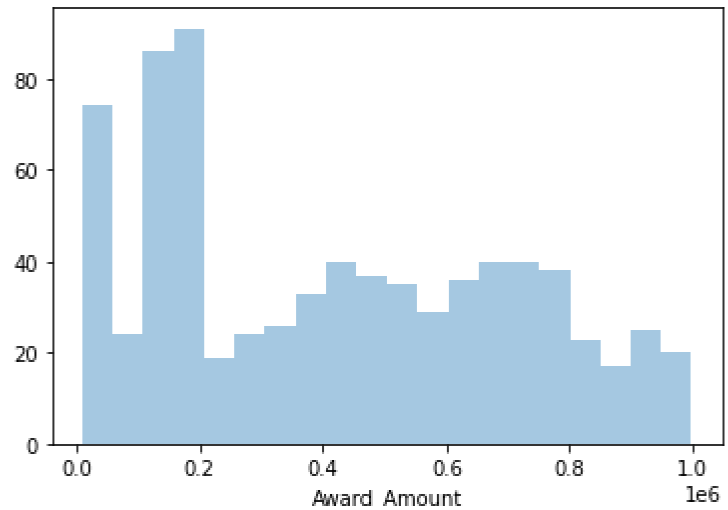
1.2 KDE Plot (Kernel Density Estimate Plot)

`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

In [8]:

```python
# Display a Seaborn distplot with options on KDE and bins
sns.distplot(grant_file['Award_Amount'],kde=False, bins=20)
plt.show()

# Clear the plot
plt.clf()
```




```
<Figure size 432x288 with 0 Axes>
```

1.3 Rug Plot

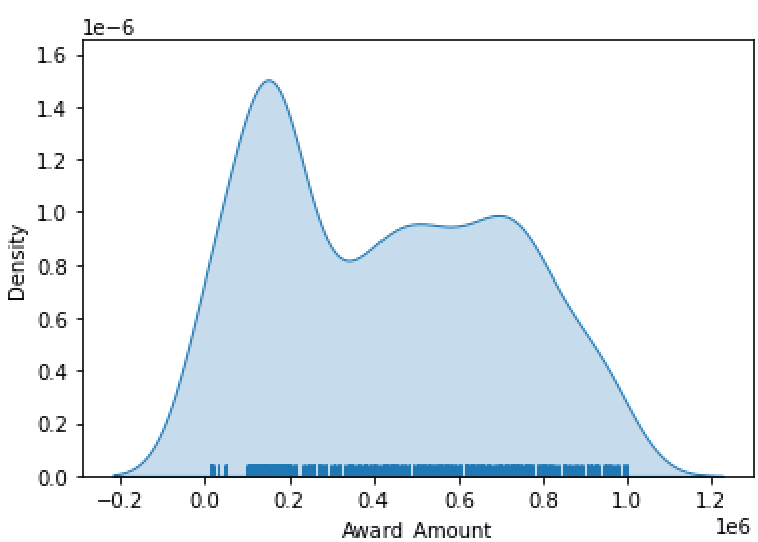
A `rug plot` is a plot of data for a single quantitative variable, displayed as marks along an axis.

It is used to visualise the distribution of the data.

As such it is analogous to a histogram with zero-width bins, or a one-dimensional scatter plot

In [9]:

```python
# Display a Seaborn distplot with options on hist and rug and
sns.distplot(grant_file['Award_Amount'],hist=False, rug=True,kde_kws={'shade':True})
plt.show()
```



↥ back to top

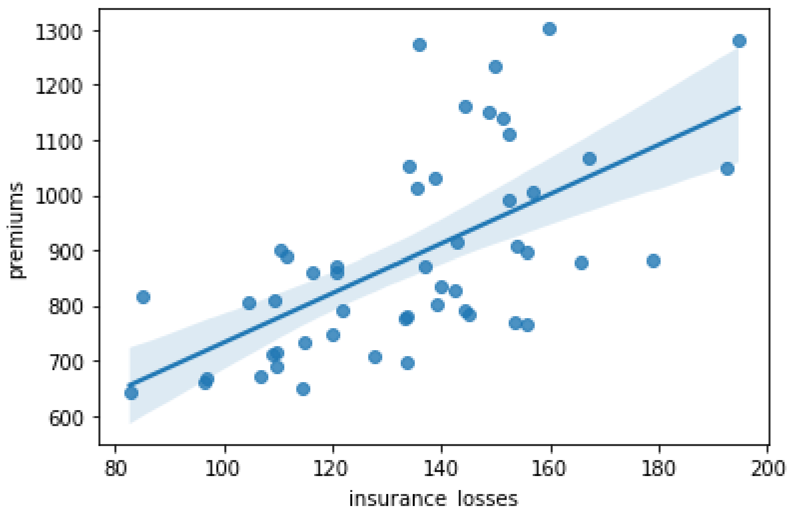
2 Bivariate Analysis

2.1 Regplot

Regression Plot to show relationship between two variables

In [10]:

```
insurance_premiums_df=pd.read_csv(input_files+"insurance_premiums.csv")

# Create a regression plot of premiums vs. insurance_losses
#sns.regplot(insurance_premiums["insurance_losses"],insurance_premiums["premiums"])
sns.regplot(data=insurance_premiums_df,x="insurance_losses",y="premiums")

# Display the plot
plt.show()
```

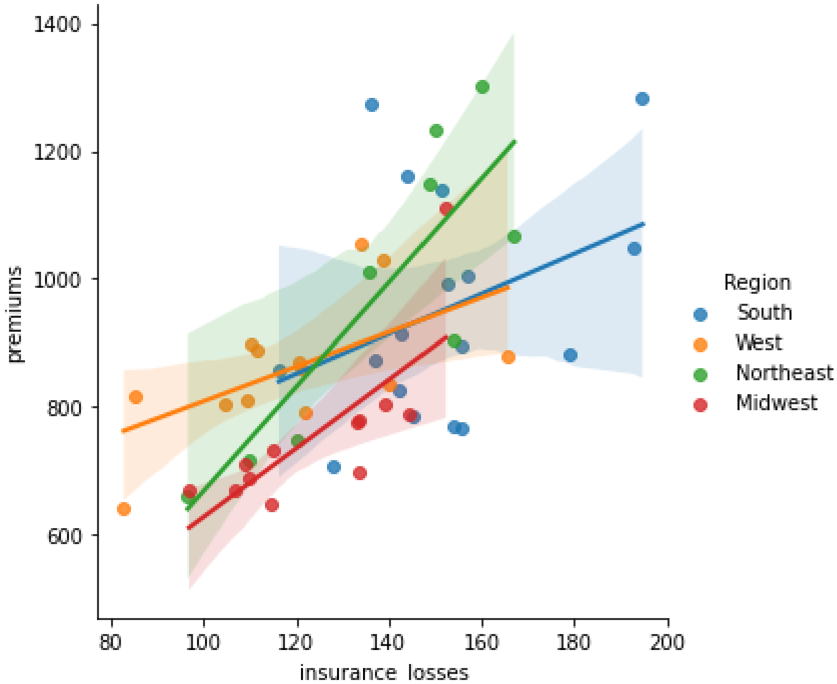


2.2 lmplot

Built on top of Regplot , lmplot is much more powerful and flexible.

In [11]:

```
sns.lmplot(data=insurance_premiums_df,x="insurance_losses",y="premiums",hue="Region")
plt.show()
```

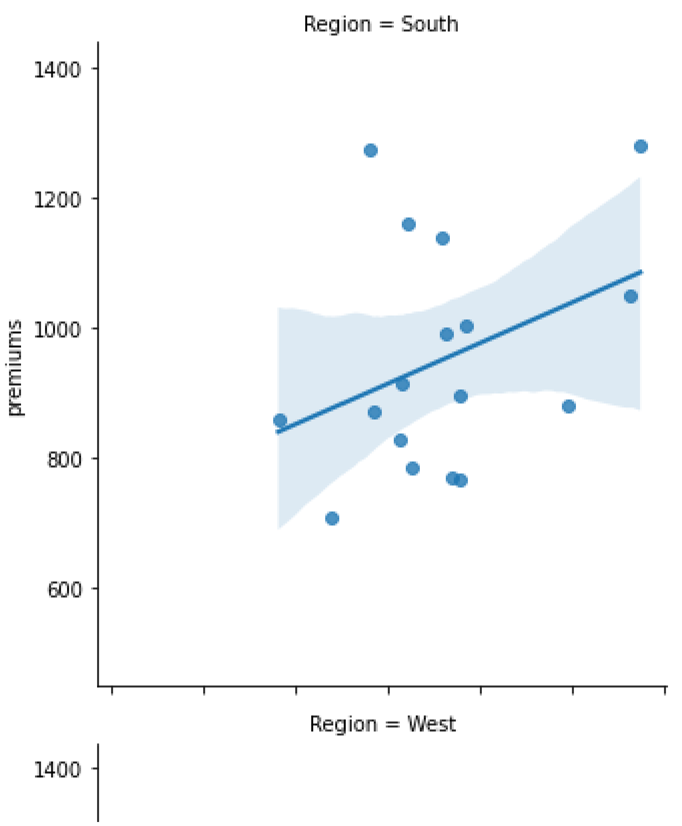


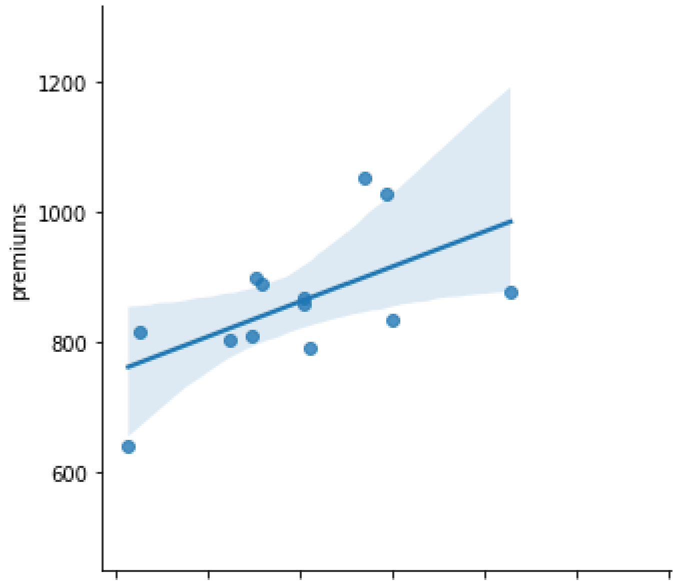
2.3 Faceting

Faceting is the act of breaking data variables up across multiple subplots, and combining those subplots into a single figure. So instead of one bar chart, we might have, say, four, arranged together in a grid.

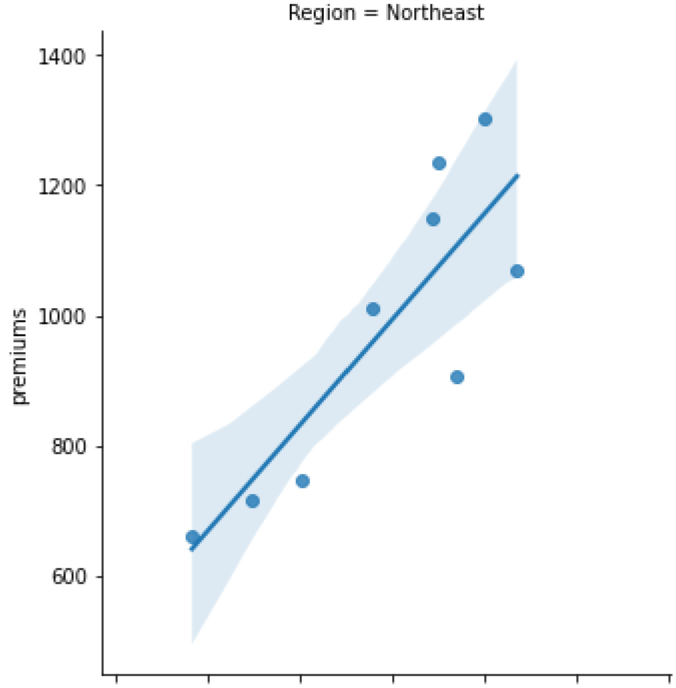
In [12]:

```
# FACETING TO SEE DATA MORE CLEARLY
sns.lmplot(data=insurance_premiums_df,x="insurance_losses",y="premiums",row="Region")
plt.show()
```

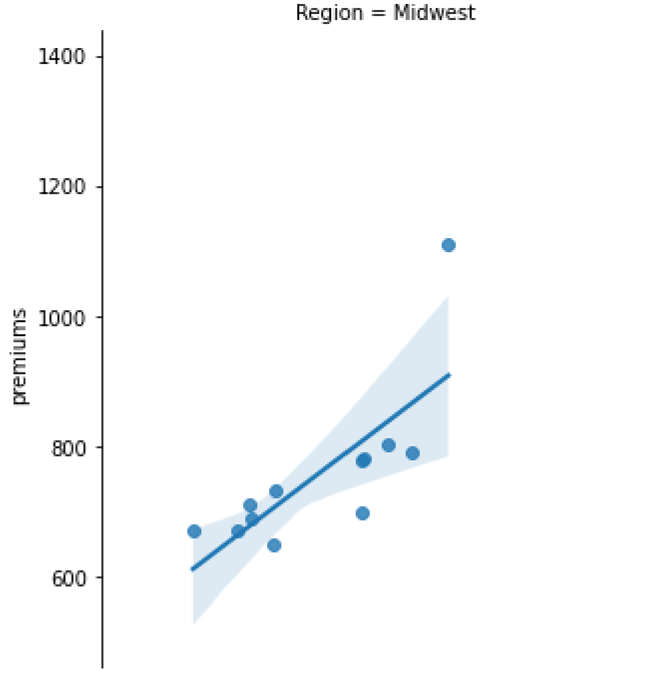


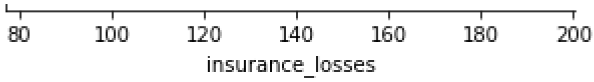


Region = Northeast



Region = Midwest



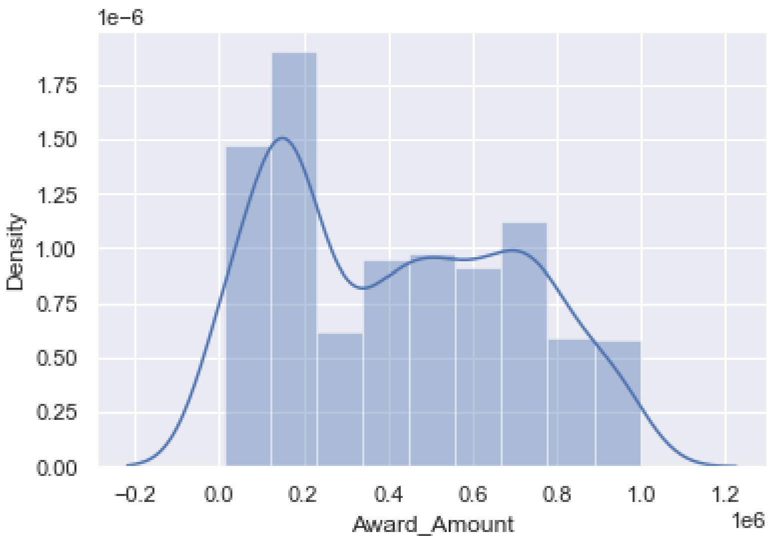


↥ back to top

3 Seaborn Styles

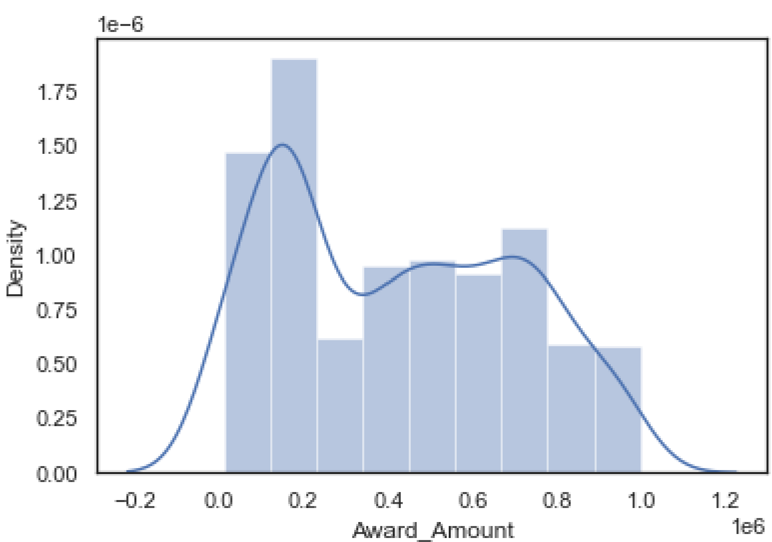
In [13]:

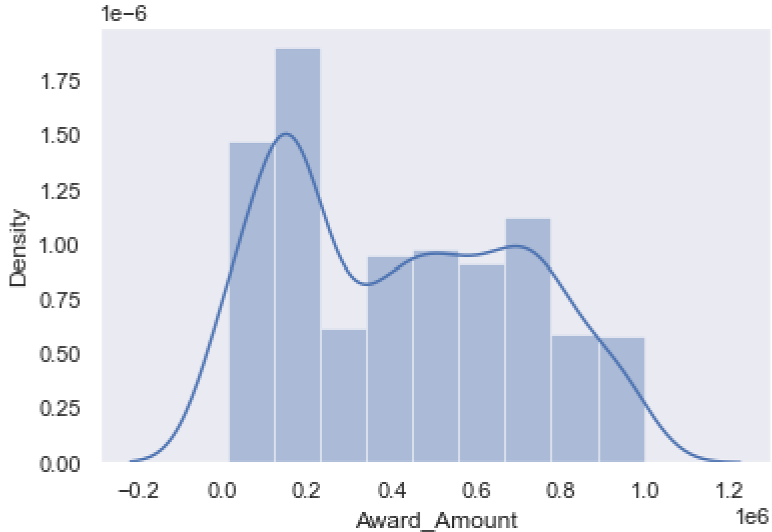
```
sns.set() #for default seaborn style
# Display a Seaborn distplot
sns.distplot(grant_file['Award_Amount'])
plt.show()
```

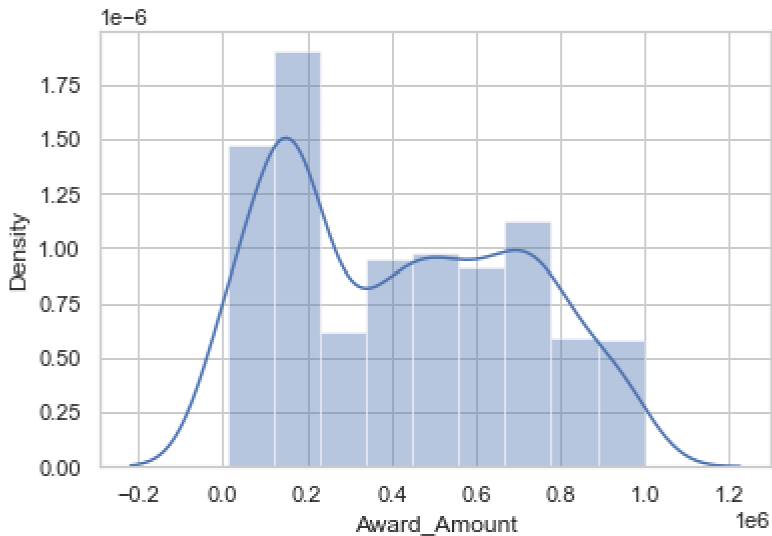


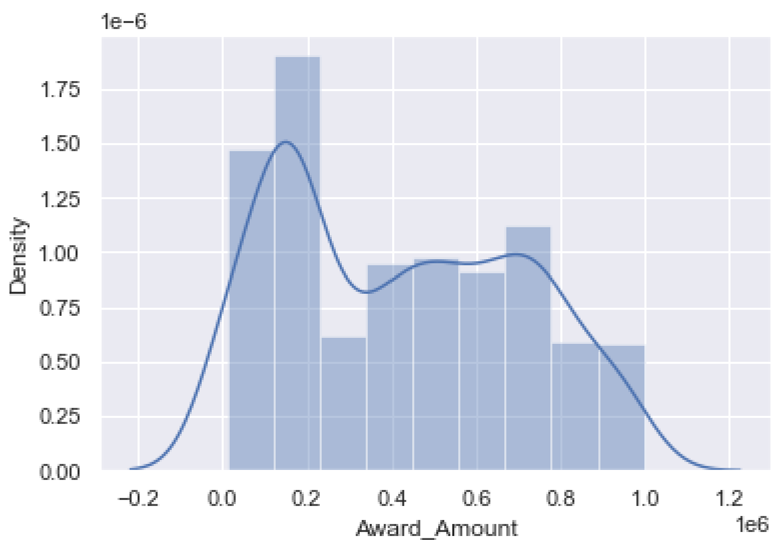
In [14]:

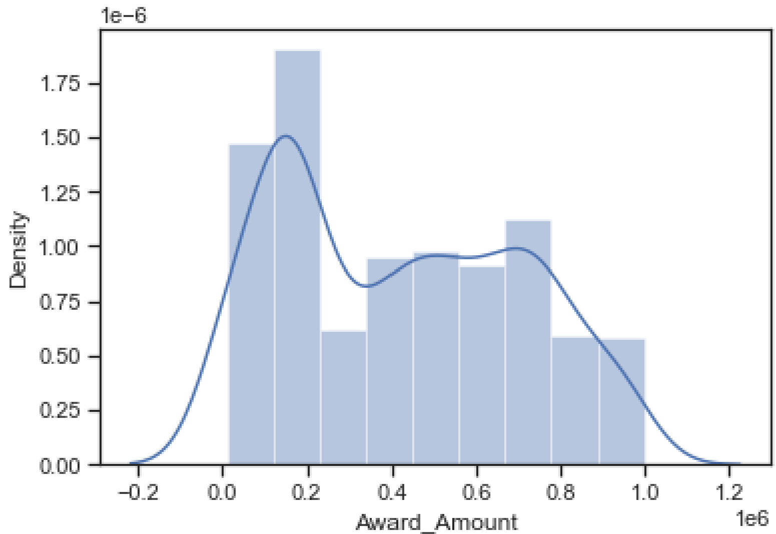
```
for style in ['white','dark','whitegrid','darkgrid','ticks']:
    sns.set_style(style)
    sns.distplot(grant_file['Award_Amount'])
    plt.show()
```




1e−6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Density
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Award_Amount
1e6


1e−6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Density
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Award_Amount
1e6


1e−6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Density
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Award_Amount
1e6



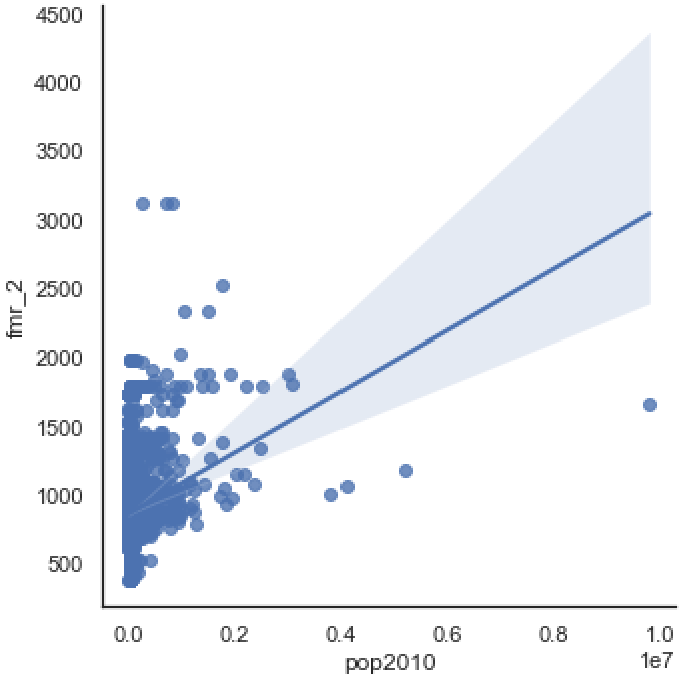
3.1 Despining Graph

Removing the top and right boundary

In [15]:

```
FMR_df = pd.read_csv(input_files+"FY18_4050_FMRs.csv")
print(FMR_df.filter(['pop2010','fmr_2']).head(10))
# Set the style to white
sns.set_style('white')

# Create a regression plot
sns.lmplot(data=FMR_df,
           x='pop2010',
           y='fmr_2')
# Remove the spines
sns.despine()

# Show the plot and clear the figure
plt.show()
plt.clf()
```

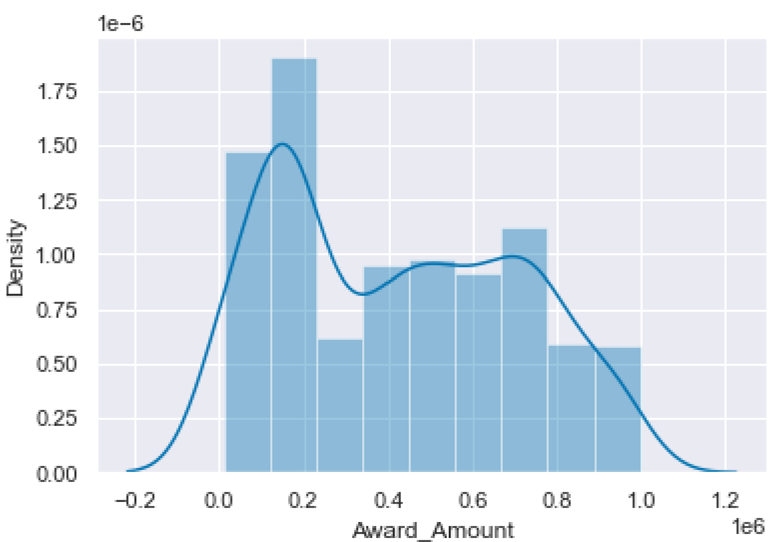
```
    pop2010  fmr_2
0   54571.0    829
1  182265.0    879
2   27457.0    657
3   22915.0    882
4   57322.0    882
5   10914.0    606
6   20947.0    606
7  118572.0    679
8   34215.0    676
9   25989.0    606
```

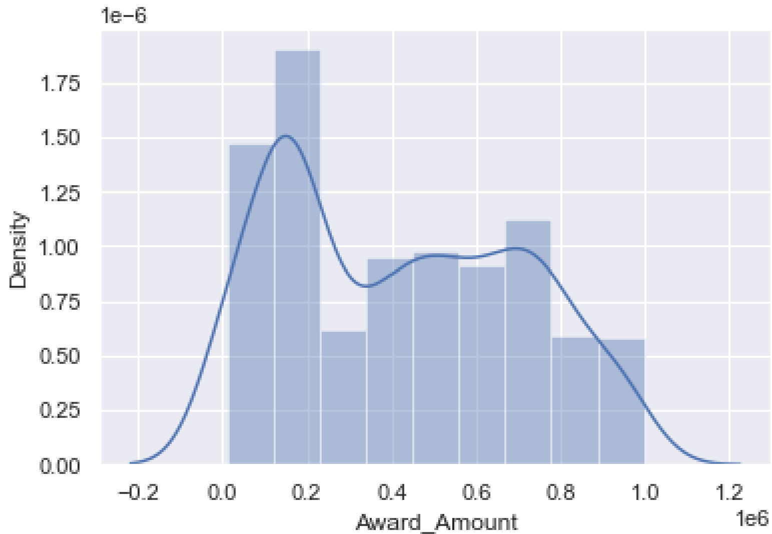



```
<Figure size 432x288 with 0 Axes>
```

In [16]:

```
sns.set(color_codes=True)  ## assigning colors from matplotlib color codes

for p in ['colorblind6','deep']:
    # for all styles use "sns.palettes.SEABORN_PALETTES" instead of list
    sns.set_palette(p)
    sns.distplot(grant_file['Award_Amount'])
    plt.show()
    plt.clf()
```





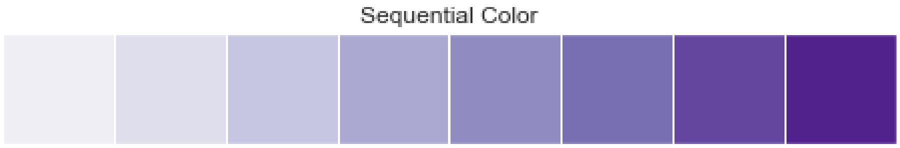

```
<Figure size 432x288 with 0 Axes>
```

3.2 Different types of Color Palette

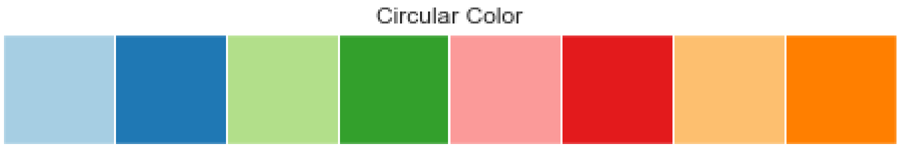
More info: https://seaborn.pydata.org/tutorial/color_palettes.html

In [17]:

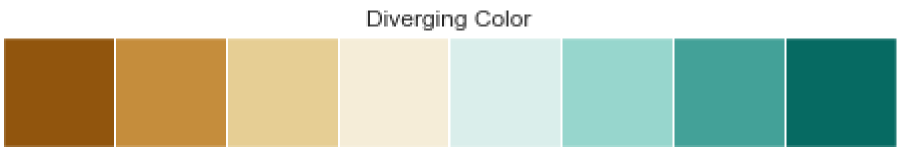
```
## Sequential Color  -  When data has consistent range from high to low
sns.palplot(sns.color_palette('Purples',8))
plt.title("Sequential Color")
plt.show()
plt.clf()

## Circular Color  - When data is not orderd

sns.palplot(sns.color_palette('Paired',8))
plt.title("Circular Color")
plt.show()
plt.clf()

## Diverging Color  - When both low and high values are interesting
sns.palplot(sns.color_palette('BrBG',8))
plt.title("Diverging Color")
plt.show()
plt.clf()


## HUsl Color
sns.palplot(sns.color_palette('husl',10))
plt.title("Husl Color")
plt.show()
plt.clf()


## CoolWarm Color
sns.palplot(sns.color_palette('coolwarm',6))
plt.title("Coolwarm Color")
plt.show()
plt.clf()
```



<Figure size 432x288 with 0 Axes>

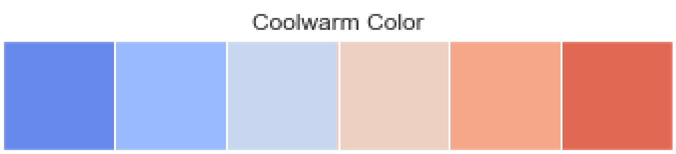


<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

Uncomment below code to check all available palettes

In [18]:

```
## sns.palplot(sns.color_palette())  # current color palette

# import itertools
# SEABORN_PALETTES = dict(itertools.islice(sns.palettes.SEABORN_PALETTES.items(), 5))

# for p in SEABORN_PALETTES:
#     sns.set_palette(p)
#     sns.palplot(sns.color_palette())
#     plt.show()
#     plt.clf()
#     print("Palette Name: ",p)
```

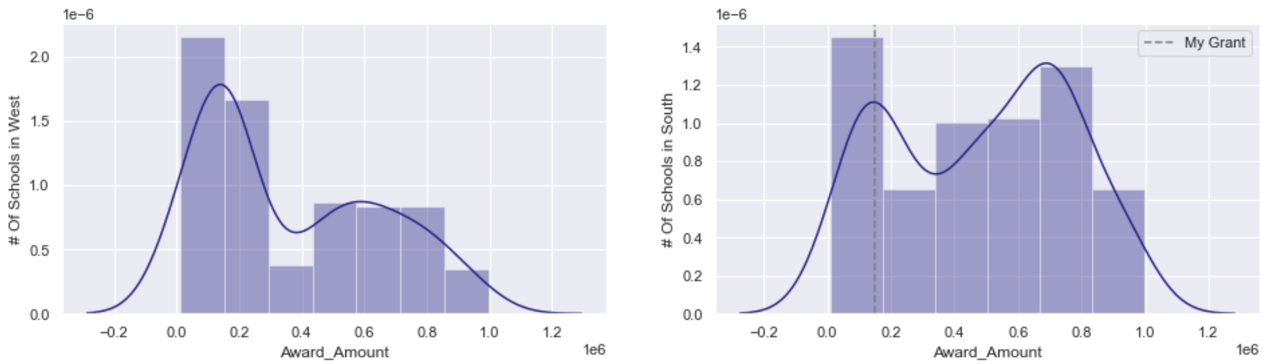
3.3 Customizing with Matplotlib functions

In [19]:

```
sns.set_palette(sns.color_palette("CMRmap"))

fig,(ax0,ax1) = plt.subplots(1,2,figsize=(16,4))

sns.distplot(grant_file.query('Region=="West"')["Award_Amount"],ax=ax0)
sns.distplot(grant_file.query('Region=="South"')["Award_Amount"],ax=ax1)

ax0.set(ylabel="# Of Schools in West")
ax1.set(ylabel="# Of Schools in South")
```

```
ax1.axvline(x=150000,label='My Grant',linestyle="--",color='grey')
ax1.legend()
```

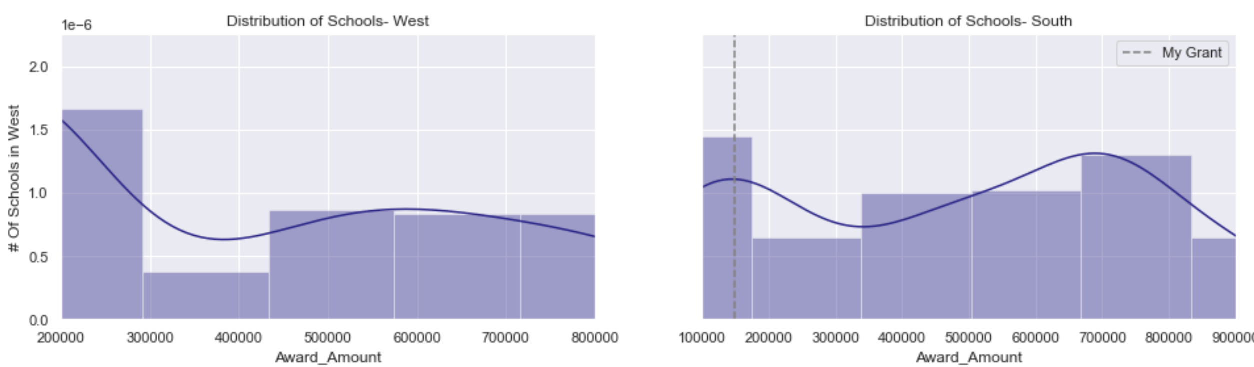
Out[19]: <matplotlib.legend.Legend at 0x244c0544ac0>



In [20]:

```
fig,(ax0,ax1) = plt.subplots(1,2,sharey=True,figsize=(16,4))

sns.distplot(grant_file.query('Region=="West"')["Award_Amount"],ax=ax0)
sns.distplot(grant_file.query('Region=="South"')["Award_Amount"],ax=ax1,)

ax0.set(ylabel="# Of Schools in West",xlabel="Award_Amount",xlim=(200000,800000),title=
ax1.set(ylabel="# Of Schools in South",xlabel="Award_Amount",xlim=(100000,900000),title
ax1.axvline(x=150000,label='My Grant',linestyle="--",color='grey')
ax1.legend()
```

Out[20]: <matplotlib.legend.Legend at 0x244c0d45b80>



↥ back to top

4 Categorical Plot types

- **4.1, 4.2 Each Observation - Strip plot, swarm plot**

  `Strip plot` shows each data point, but could sometime become difficult to understand with large datasets. Better understood if some Jitter is created.

  `Swarm plot` shows the same as Stripplot but tries to avoid overlaps. Because of this, it is not the most accurate representation and doesnt scale well with large data.

- **4.3, 4.4, 4.5 Abstract Representations - Box plot, violin plot, lv plot**

When we need to understand the distribution of data with call outs on outlier points we can use Boxplot, violin or Lvplot.

- **4.6 Statistical Estimates - Bar plot, point plot, count plot**

In [21]:

```
df = pd.read_csv(input_files+"college_datav3.csv")
df.head(10)
```

Out[21]:

	INSTNM	OPEID	REGION	SAT_AVG_ALL	PCTPELL	PCTFLOAN	ADM_RATE_ALL	UG	AVGFA
0	Alabama A & M University	100200	5	850.0	0.7249	0.8159	0.653841	4380.0	1
1	University of Alabama at Birmingham	105200	5	1147.0	0.3505	0.5218	0.604275	10331.0	10
2	Amridge University	2503400	5	NaN	0.7455	0.8781	NaN	98.0	3
3	University of Alabama in Huntsville	105500	5	1221.0	0.3179	0.4589	0.811971	5220.0	9
4	Alabama State University	100500	5	844.0	0.7567	0.7692	0.463858	4348.0	1
5	The University of Alabama	105100	5	1181.0	0.2009	0.4059	0.535867	15318.0	9
6	Central Alabama Community College	100700	5	NaN	0.5554	0.3574	NaN	1577.0	5
7	Athens State University	100800	5	NaN	0.4233	0.6512	NaN	2662.0	1
8	Auburn University at Montgomery	831000	5	990.0	0.4373	0.5584	0.787089	4098.0	1
9	Auburn University	100900	5	1218.0	0.1631	0.3470	0.776605	18326.0	9

10 rows × 24 columns

In [22]:

```
df = df.filter(["Tuition","Regions","REGION","Ownership"])
print(df["REGION"].unique())
print(df["Regions"].unique())
print(df["Ownership"].unique())
```

```
[5 8 6 4 7 1 2 3 0 9]
['South East' 'Far West' 'South West' 'Plains' 'Rocky Mtns' 'New England'
```

```
 'Mid East' 'Great Lakes' 'US Service Schools' 'Outlying']
['Public' 'Private non-profit' 'Private for-profit']
```

4.1 Strip Plot

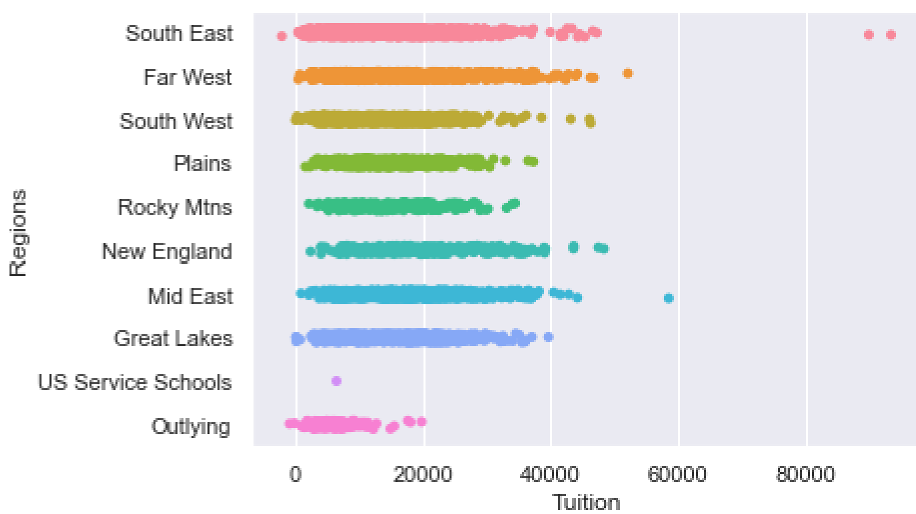
`Strip plot` shows each data point, but could sometime become difficult to understand with large datasets. Better understood if some Jitter is created.

In [23]:

```
sns.stripplot(data=df,y="Regions",x="Tuition",jitter=True)
```

Out[23]:

```
<AxesSubplot:xlabel='Tuition', ylabel='Regions'>
```

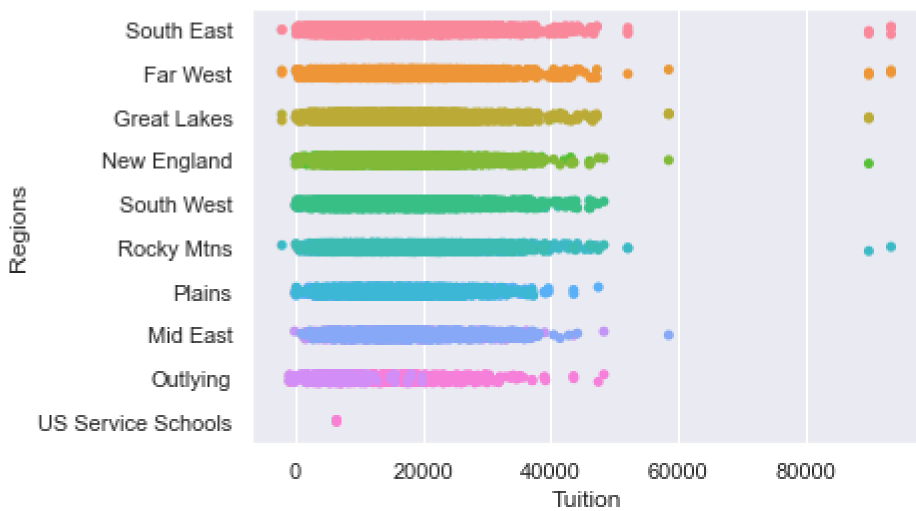


In [24]:

```
import time

for i in np.arange(0.1,1.1,0.1):    # 0 to 100% incr of 5 %
    start_time = time.process_time()
    df2 = df.sample(frac=i)
    sns.stripplot(data=df2,y="Regions",x="Tuition",jitter=True)
    print(time.process_time() - start_time, "seconds")
    print("Samprate= "+ str(int(i*100)) +"%, Rows= " + str(df2.shape[0])+"\n")
```

```
0.078125 seconds
Samprate= 10%, Rows= 670

0.078125 seconds
Samprate= 20%, Rows= 1340

0.0625 seconds
Samprate= 30%, Rows= 2011

0.046875 seconds
Samprate= 40%, Rows= 2681

0.078125 seconds
Samprate= 50%, Rows= 3351

0.0625 seconds
Samprate= 60%, Rows= 4021

0.046875 seconds
```

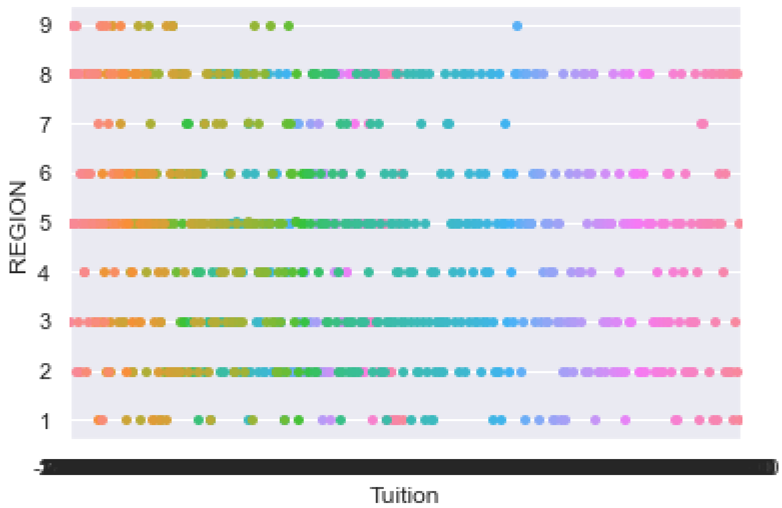
```
Samprate= 70%, Rows= 4691

0.078125 seconds
Samprate= 80%, Rows= 5362

0.109375 seconds
Samprate= 90%, Rows= 6032

0.046875 seconds
Samprate= 100%, Rows= 6702
```



In [25]:

```
import time

for i in np.arange(0.05,0.15,0.05):    # 0 to 10% incr of 5 %
    start_time = time.process_time()
    df2 = df.sample(frac=i)
    sns.stripplot(data=df2,y="REGION",x="Tuition")
    print(time.process_time() - start_time, "seconds")
    print("Samprate= "+ str(int(i*100)) +"%, Rows= " + str(df2.shape[0])+"\n")
```

```
12.046875 seconds
Samprate= 5%, Rows= 335

47.25 seconds
Samprate= 10%, Rows= 670
```



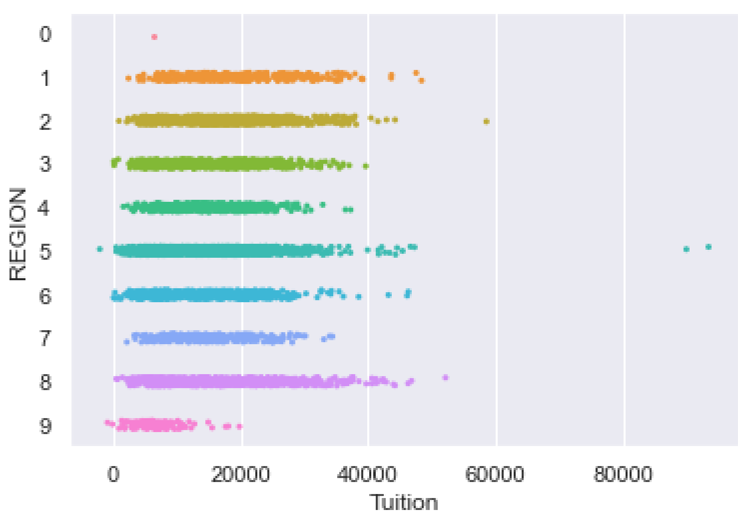
Tip: Always convert any variables into categorical type if they are int type and are supposed to be categorical. Else it will take a long time

In [26]:

```
df["REGION"]=df["REGION"].astype('category')
sns.stripplot(data=df,y="REGION",x="Tuition",size=3)
```

Out[26]:

```
<AxesSubplot:xlabel='Tuition', ylabel='REGION'>
```



4.2 Swarm Plot

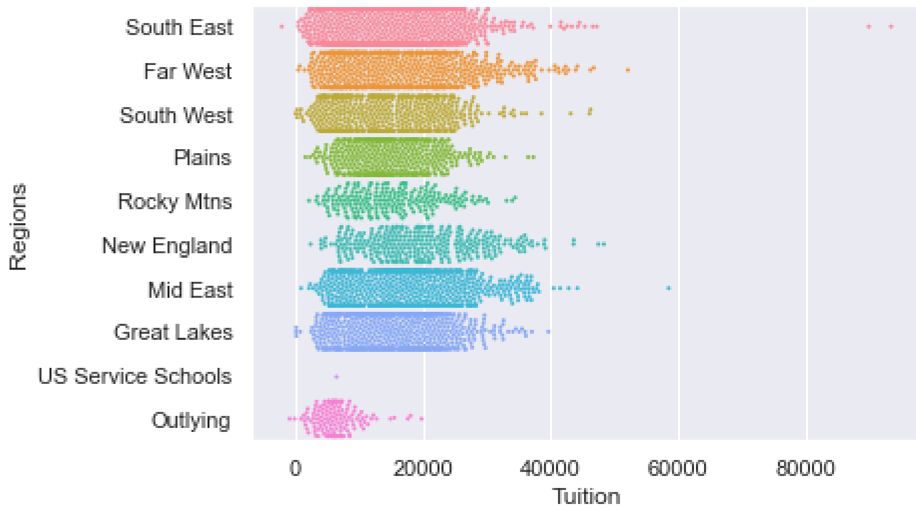
`Swarmplot` shows the same as Stripplot but tries to avoid overlaps. Because of this, it is not the most accurate representation and doesnt scale well with large data.

In [27]:

```
# plt.subplots(figsize=(12,10))
sns.swarmplot(data=df,y="Regions",x="Tuition",size=2)
```

Out[27]:

```
<AxesSubplot:xlabel='Tuition', ylabel='Regions'>
```



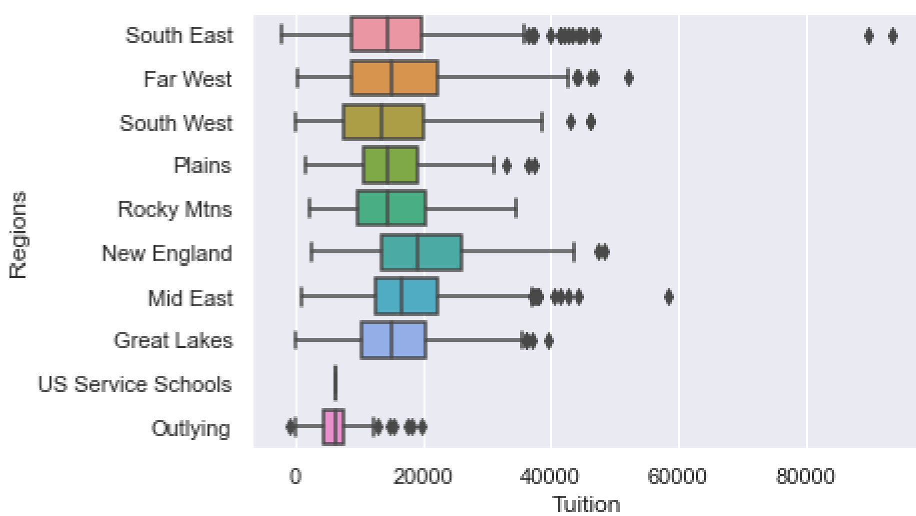
4.3 Box Plot

In [28]:

```
sns.boxplot(data=df,y="Regions",x="Tuition")
```

Out[28]:

```
<AxesSubplot:xlabel='Tuition', ylabel='Regions'>
```



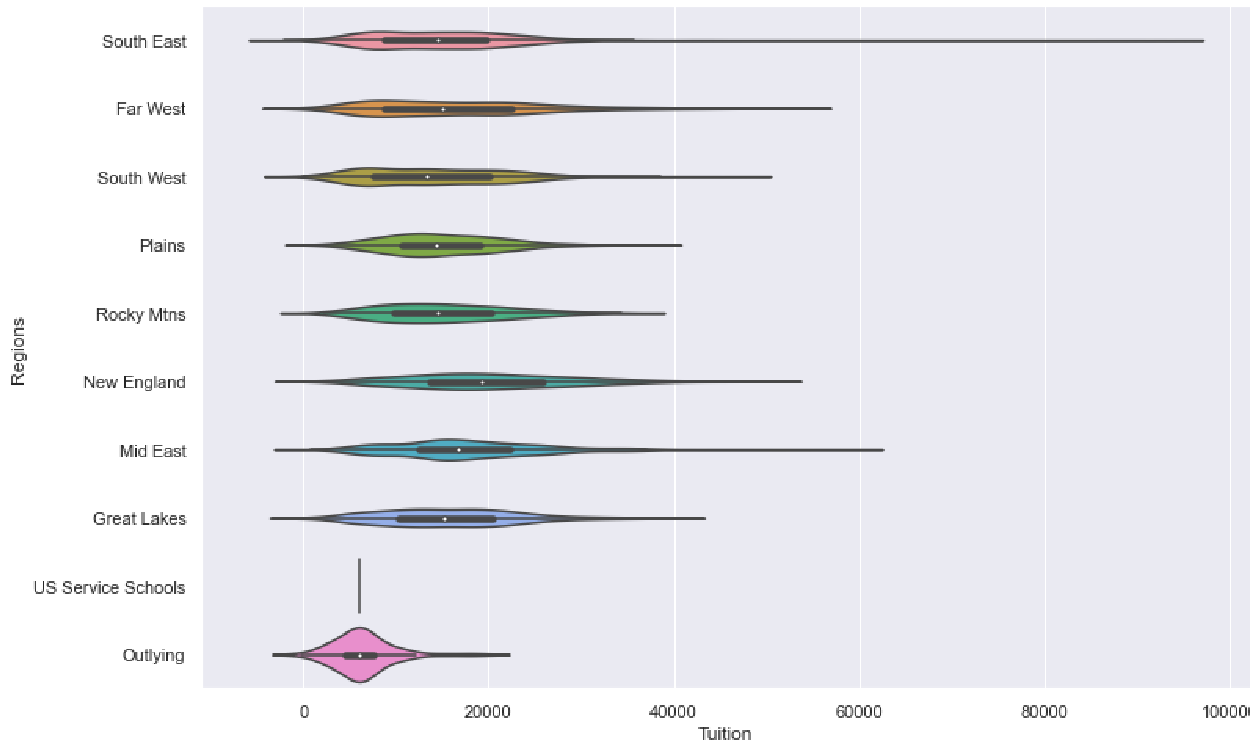
4.4 Violin Plot

In [29]:

```
fig,ax = plt.subplots(figsize=(12,8))
sns.violinplot(data=df,y="Regions",x="Tuition")    # Does Kernel density cal and hence
                                                    # Can be compoutationally intensive
```

Out[29]:

```
<AxesSubplot:xlabel='Tuition', ylabel='Regions'>
```



4.5 LP Plot / Boxen Plot

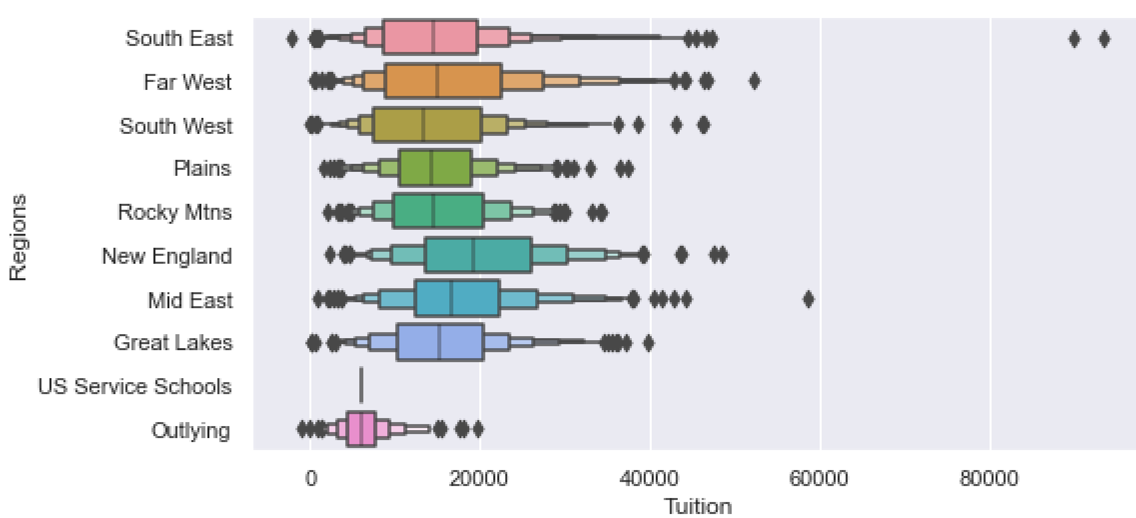
Renamed as boxenplot. Can scale more easily for large dataset

In [30]:

```
fig,ax = plt.subplots(figsize=(8,4))
sns.boxenplot(data=df,y="Regions",x="Tuition")
```

Out[30]:

```
<AxesSubplot:xlabel='Tuition', ylabel='Regions'>
```



4.6 Bar Plot (Different Types)

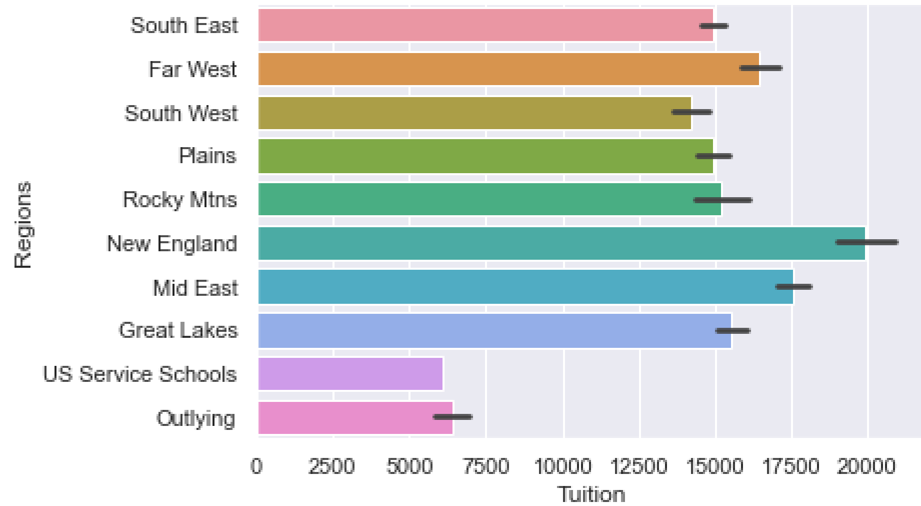
Bar Plot

In [31]:

```
sns.barplot(data=df,y="Regions",x="Tuition")
```

```
<AxesSubplot:xlabel='Tuition', ylabel='Regions'>
```

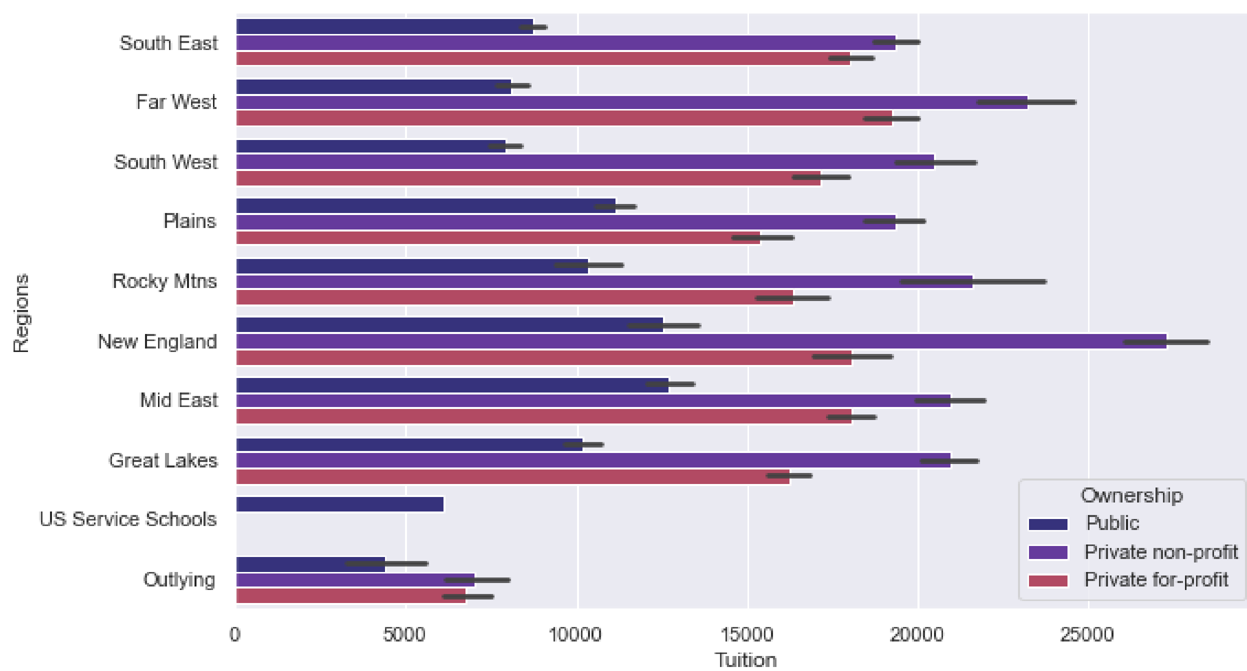
Out[31]:



In [34]:

```
fig,ax = plt.subplots(figsize=(10,6))
sns.barplot(data=df,y="Regions",x="Tuition",hue="Ownership")
```

Out[34]: <AxesSubplot:xlabel='Tuition', ylabel='Regions'>

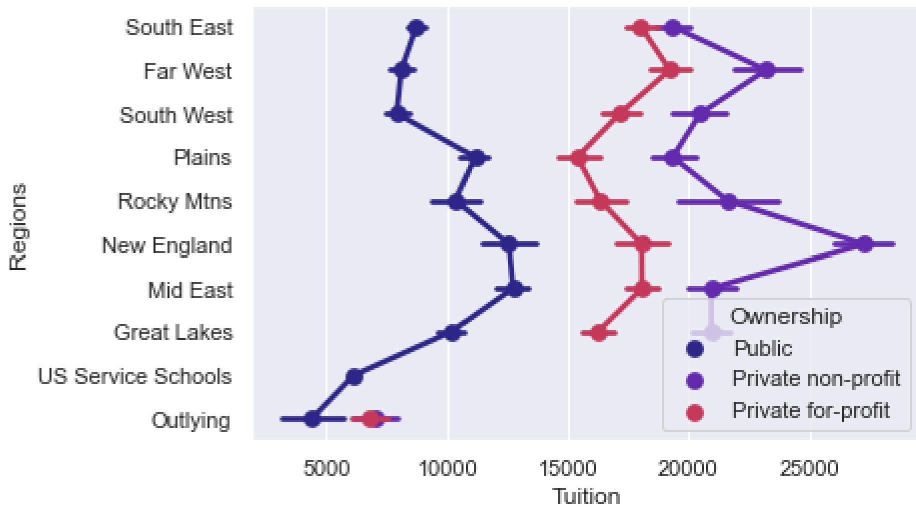


Point Plot

In [32]:

```
# fig,ax = plt.subplots(figsize=(12,10))
sns.pointplot(data=df,y="Regions",x="Tuition",hue="Ownership")
```

Out[32]: <AxesSubplot:xlabel='Tuition', ylabel='Regions'>



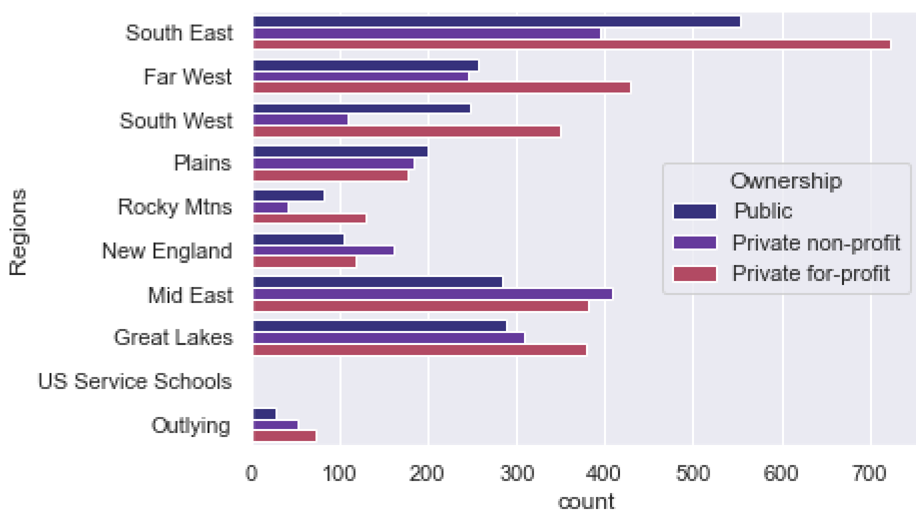
Count Plot

In [33]:

```
# fig,ax = plt.subplots(figsize=(12,10))
sns.countplot(data=df,y="Regions",hue="Ownership")
```

Out[33]:

```
<AxesSubplot:xlabel='count', ylabel='Regions'>
```



↥ back to top

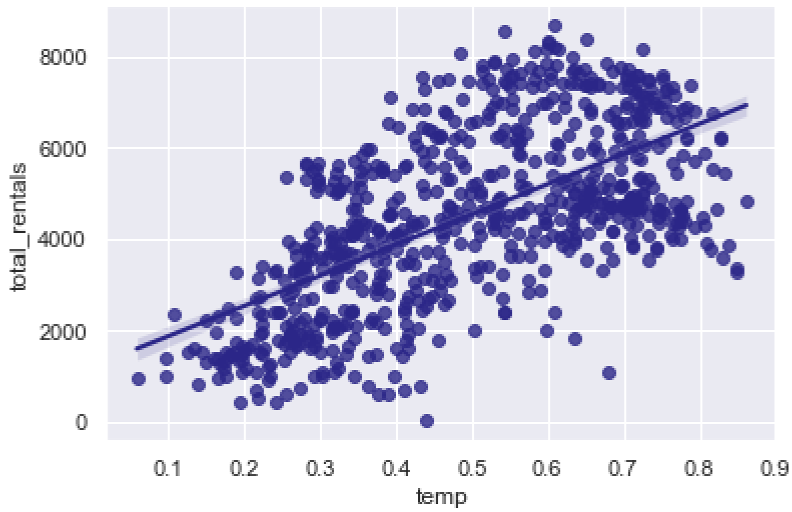
5 Regression Plots

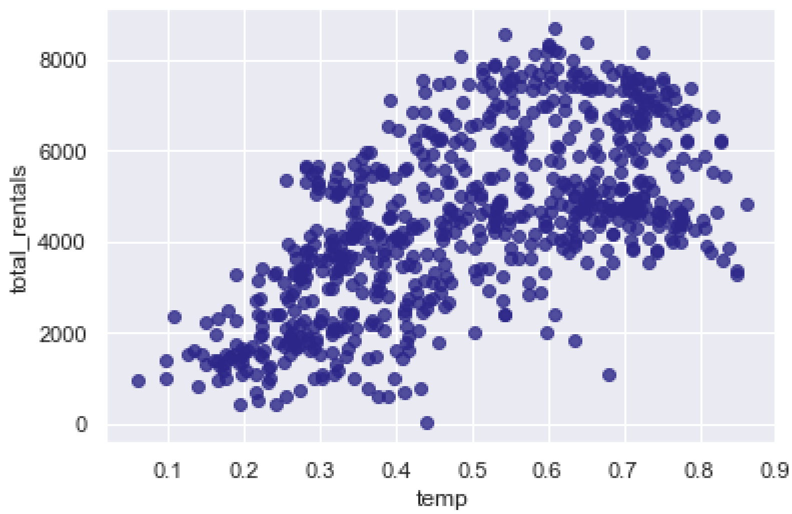
In [35]:

```
df = pd.read_csv(input_files+"bike_share.csv")
```

In [36]:

```
sns.regplot(data=df,x='temp',y='total_rentals')  # Defaults to a linear regression
plt.show()
sns.regplot(data=df,x='temp',y='total_rentals',fit_reg=False)  # Defaults to a linear r
plt.show()
```



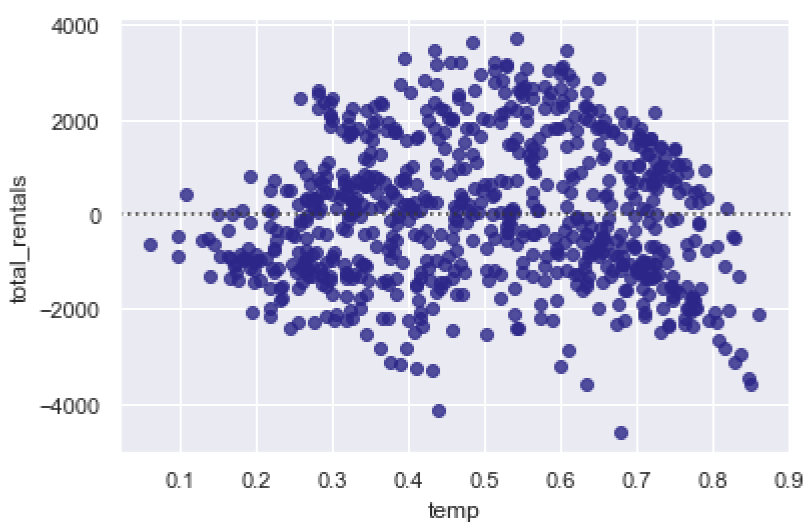


In [37]:

```
# RESIDUAL PLOT to understand residuals from the models and evaluate the fit
sns.residplot(data=df,x='temp',y='total_rentals')
```

Out[37]:

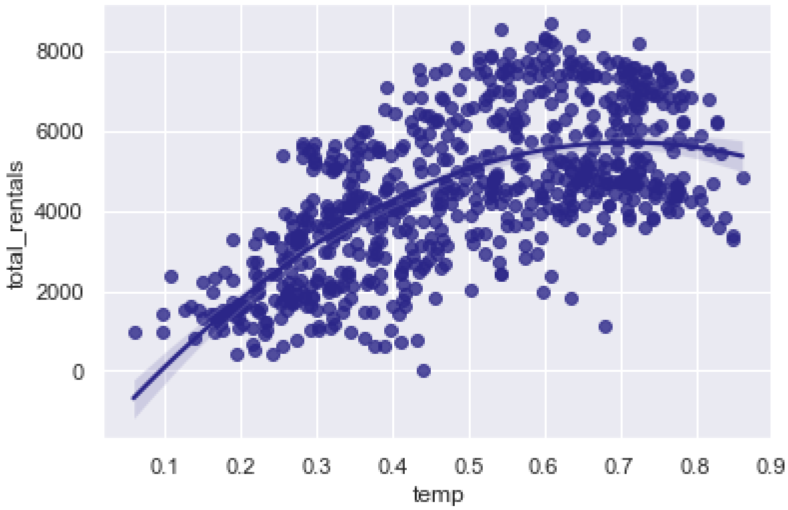
```
<AxesSubplot:xlabel='temp', ylabel='total_rentals'>
```



In [38]:

```
sns.regplot(data=df,x='temp',y='total_rentals',order=2)  # Polynomial function with ord
```

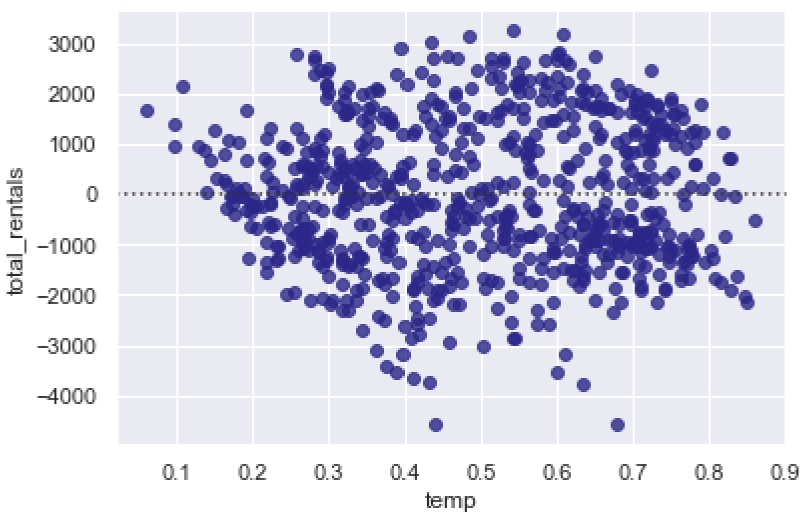
Out[38]: <AxesSubplot:xlabel='temp', ylabel='total_rentals'>



In [39]:

```
# RESIDUAL PLOT to understand residuals from the models and evaluate the fit
sns.residplot(data=df,x='temp',y='total_rentals',order=2)
```

Out[39]: <AxesSubplot:xlabel='temp', ylabel='total_rentals'>



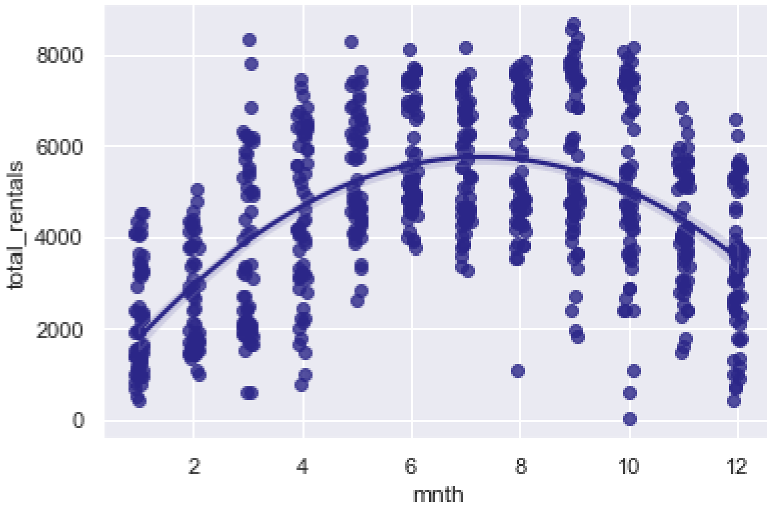
More random residuals with a polynomial fit (order =2). Hence better fit.

5.1 With Categorical Variables

In [40]:

```
sns.regplot(data=df,x='mnth',y='total_rentals',x_jitter=0.1,order=2)
```

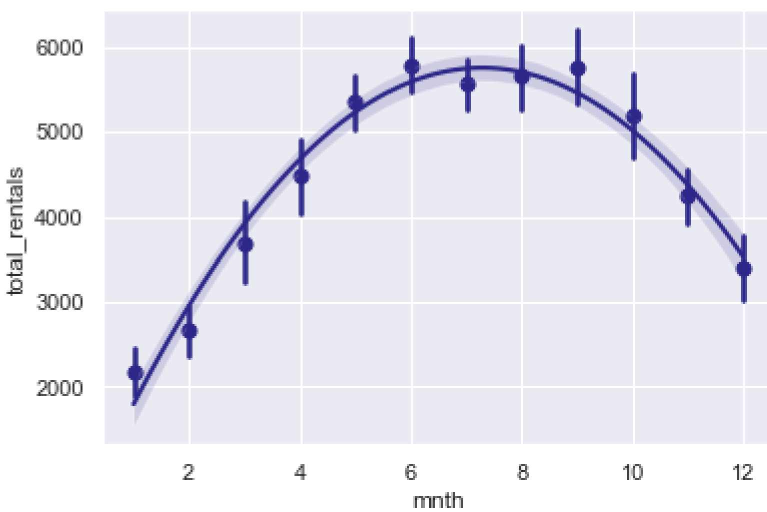
Out[40]: <AxesSubplot:xlabel='mnth', ylabel='total_rentals'>



In [41]:

```
sns.regplot(data=df,x='mnth',y='total_rentals',x_estimator=np.mean,order=2)
```

Out[41]: <AxesSubplot:xlabel='mnth', ylabel='total_rentals'>

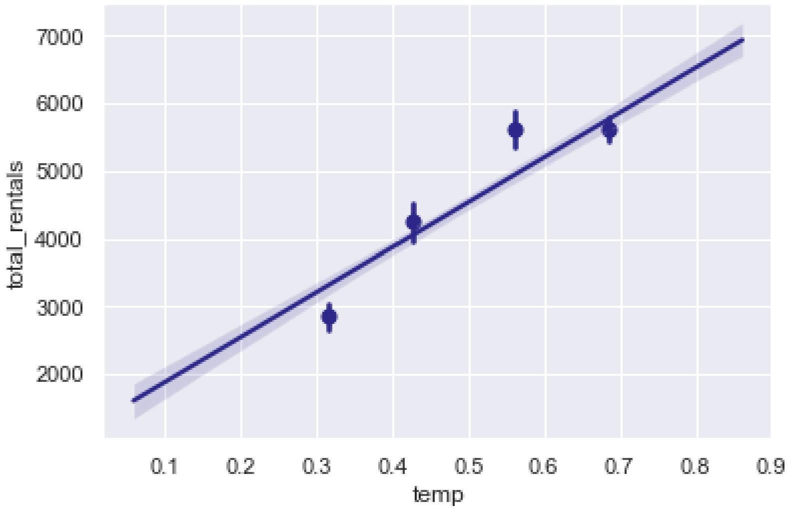


5.2 With Continuous Variables with Automated Bins

In [42]:

```
sns.regplot(data=df,x='temp',y='total_rentals',x_bins=4)
```

Out[42]: <AxesSubplot:xlabel='temp', ylabel='total_rentals'>



6 Matrix Plot

Heatmap function expects data to be in a `matrix`. We can use **`crosstab()`** in `pandas` to do this.

In [43]:

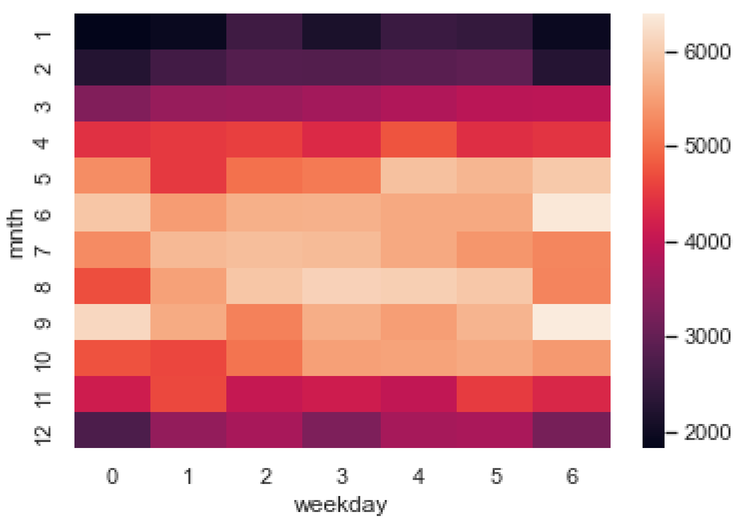
```
mat = pd.crosstab(df["mnth"],df["weekday"],values=df["total_rentals"],aggfunc='mean').a
```

In [44]:

```
sns.heatmap(mat)
```

Out[44]:

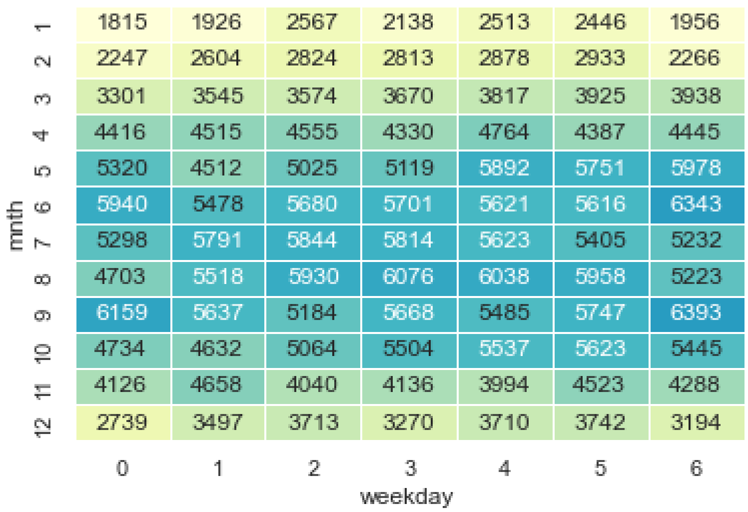
```
<AxesSubplot:xlabel='weekday', ylabel='mnth'>
```



In [45]:

```
## Customizing a heat map
sns.heatmap(mat,
            annot=True,
            fmt="d",
            cmap="YlGnBu",
            cbar=False,
            linewidths=0.5,
            center = mat.loc[6,3]
           )
```

Out[45]: <AxesSubplot:xlabel='weekday', ylabel='mnth'>

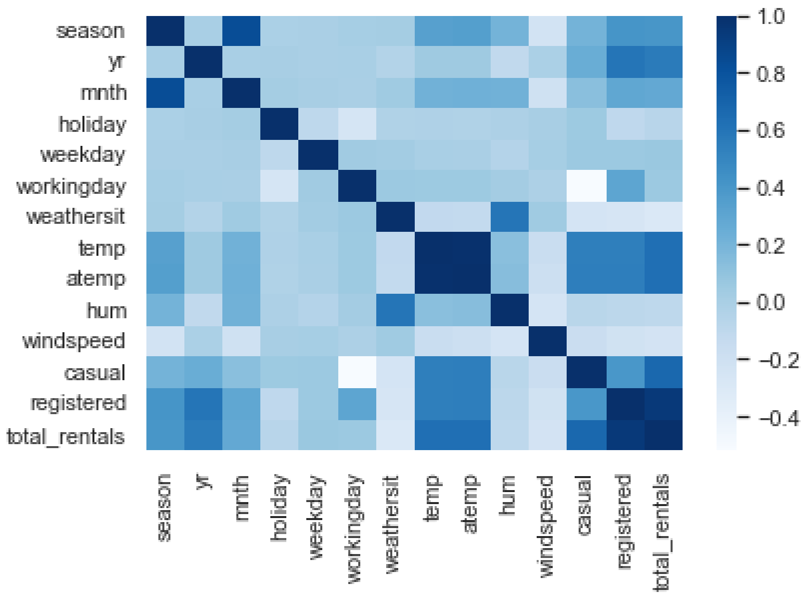


↥ back to top

7 Correlation Map Plot

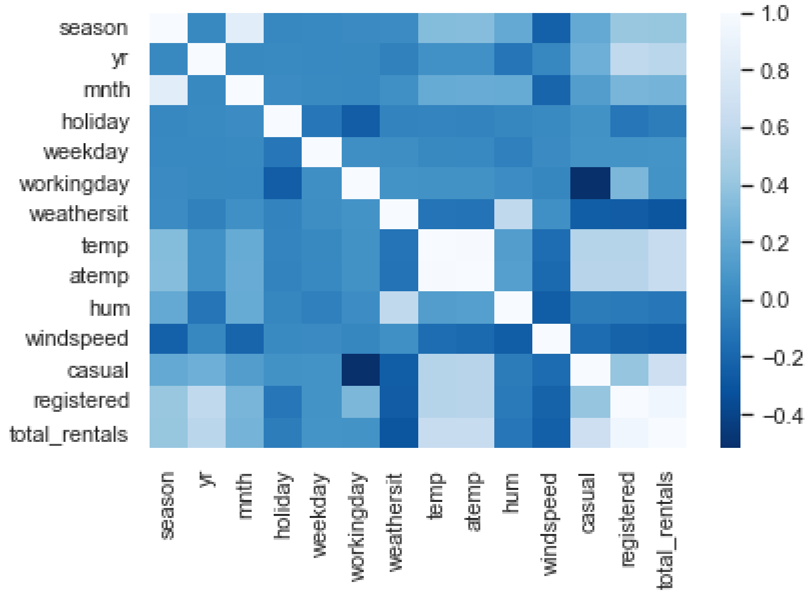
In [46]:
```
sns.heatmap(df.corr(),cmap="Blues")
```

Out[46]: <AxesSubplot:>



In [47]:
```
sns.heatmap(df.corr(),cmap="Blues_r")  ## Reverse the color scheme
```

Out[47]: <AxesSubplot:>

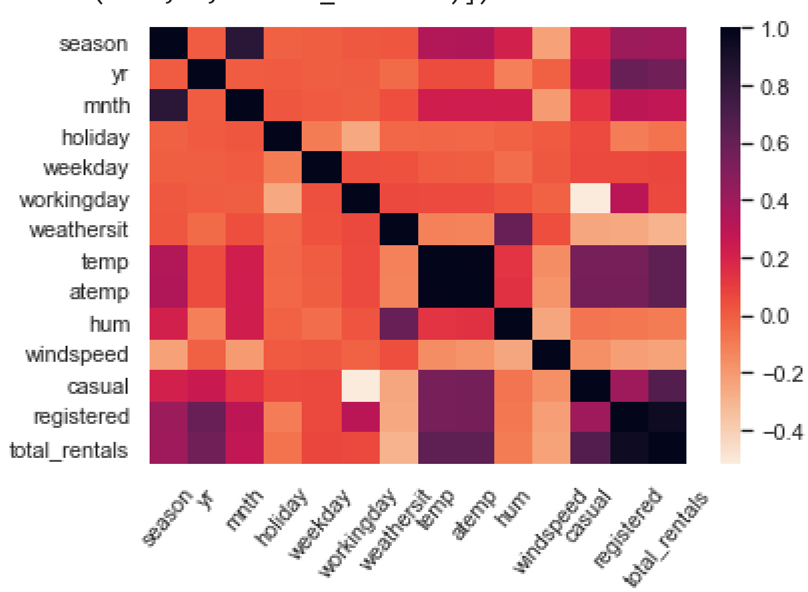


In [48]:

```
sns.heatmap(df.corr(),cmap="rocket_r",)  ## Reverse the default color scheme rocket
# Rotate tick marks for visibility
plt.yticks(rotation=0)
plt.xticks(rotation=50)
```

Out[48]:

```
(array([ 0.5,  1.5,  2.5,  3.5,  4.5,  5.5,  6.5,  7.5,  8.5,  9.5, 10.5,
        11.5, 12.5, 13.5]),
 [Text(0.5, 0, 'season'),
  Text(1.5, 0, 'yr'),
  Text(2.5, 0, 'mnth'),
  Text(3.5, 0, 'holiday'),
  Text(4.5, 0, 'weekday'),
  Text(5.5, 0, 'workingday'),
  Text(6.5, 0, 'weathersit'),
  Text(7.5, 0, 'temp'),
  Text(8.5, 0, 'atemp'),
  Text(9.5, 0, 'hum'),
  Text(10.5, 0, 'windspeed'),
  Text(11.5, 0, 'casual'),
  Text(12.5, 0, 'registered'),
  Text(13.5, 0, 'total_rentals')])
```



8 Seaborn Cheat sheet

Seaborn Cheatsheet

↥ back to top

Great Job!
