Missing Values and Plotting Pretty Graphs

by Alex Dance

Purpose

Share how cleaned data and how present data

Agenda

- Cleaning
- Presenting
- Comparing

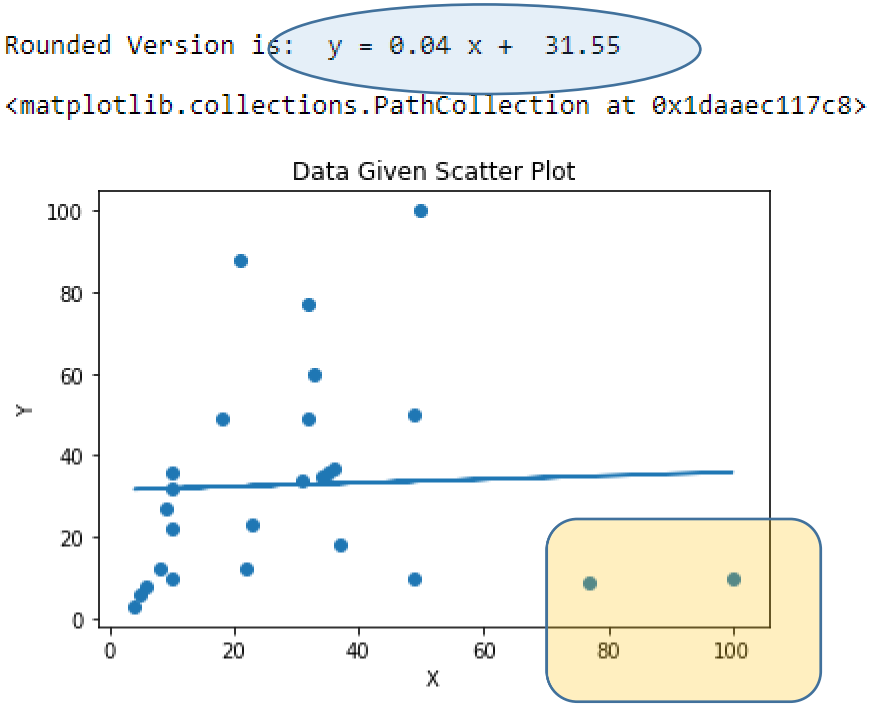
- Not Work
- Not Work Well
- Not Work Well
- Work Just OK
- Work Well

Not Showed as we have all done in exercises

- Profile report
- Standard Correlations

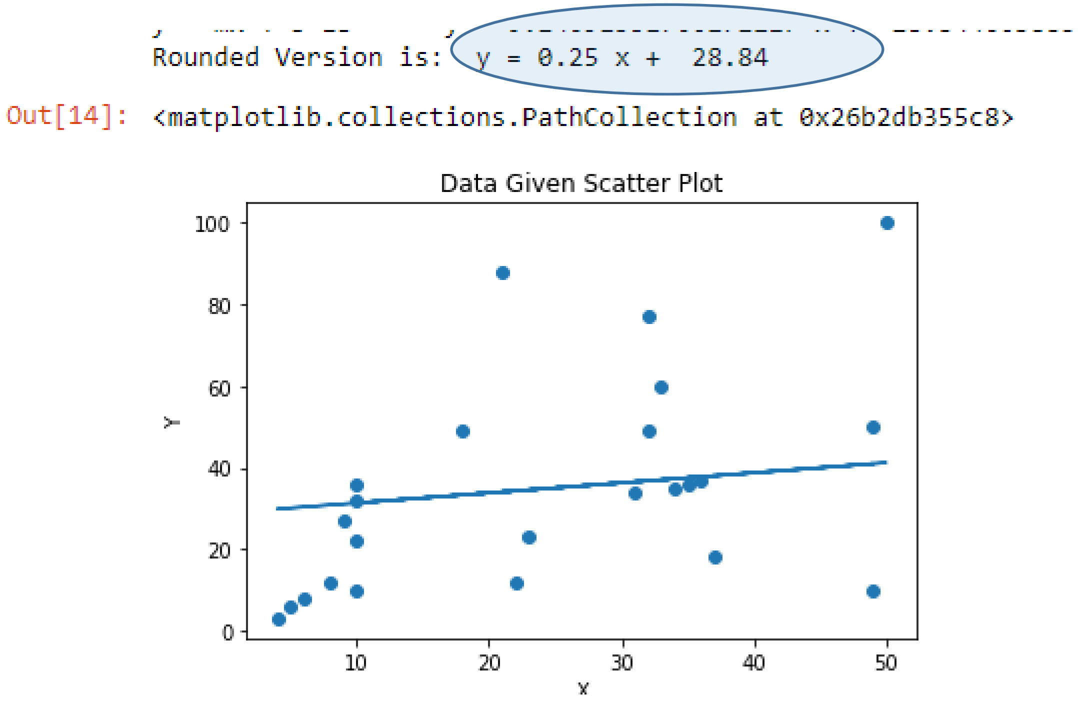
Stats Lab

1.1.4 Lab 1.1.4 Statistics - part 2 l



```
Rounded Version is:  y = 0.04 x +  31.55
<matplotlib.collections.PathCollection at 0x1daaec117c8>
```

Data Given Scatter Plot



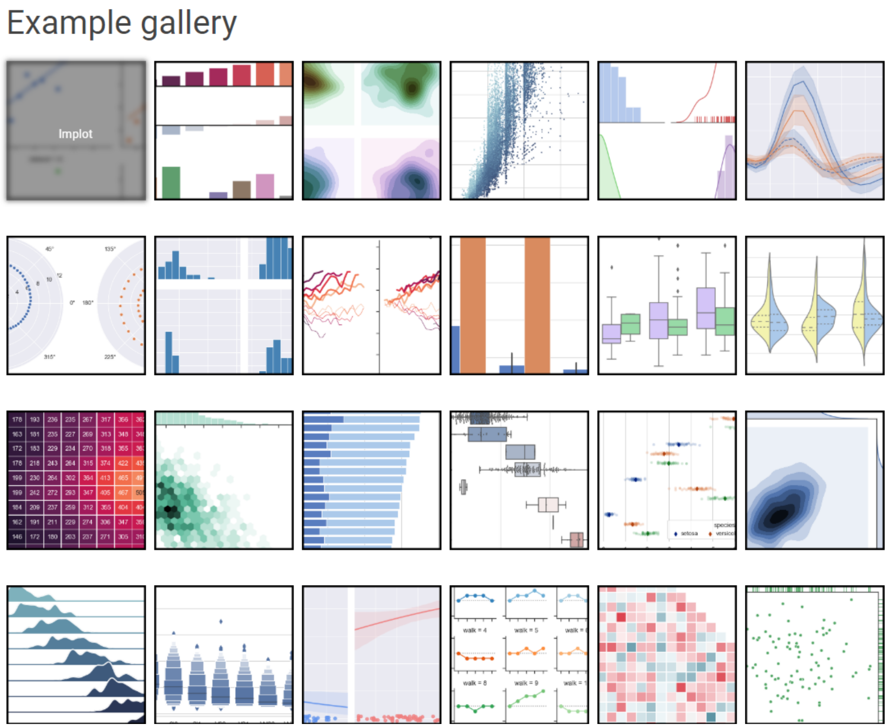
```
Rounded Version is:  y = 0.25 x +  28.84
Out[14]: <matplotlib.collections.PathCollection at 0x26b2db355c8>
```

Data Given Scatter Plot

Presenting

Wanted to print some impressive graphs

https://seaborn.pydata.org/generated/seaborn.kdeplot.html

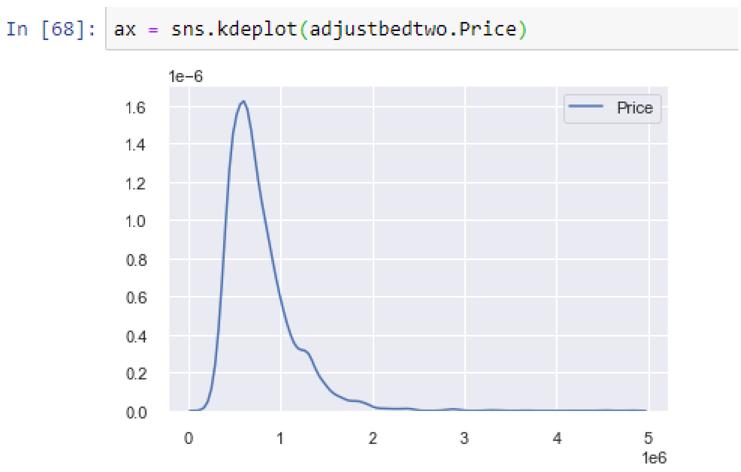


Example gallery

Dataset of Melbourne Data and Model Comparison from Kaggle

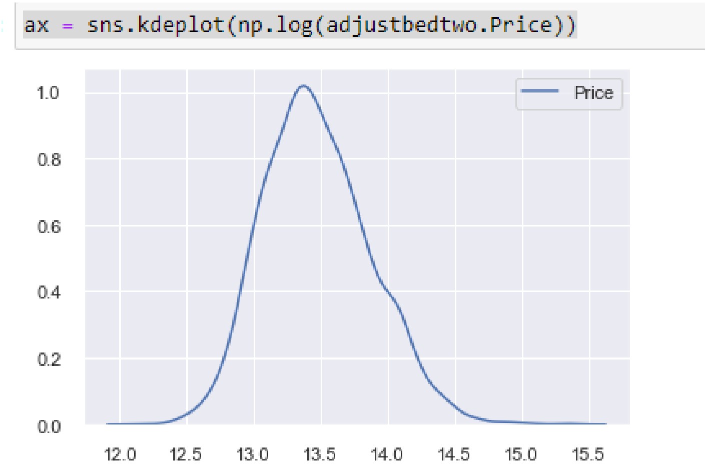
Started with some basic graphs

Price of 2 beds



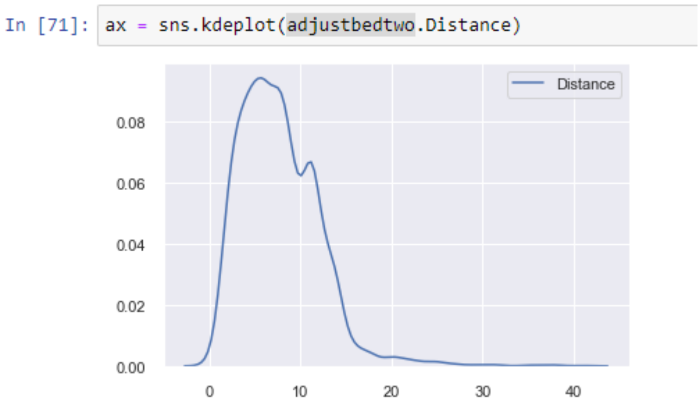
```
In [68]: ax = sns.kdeplot(adjustbedtwo.Price)
```

Log of the same thing

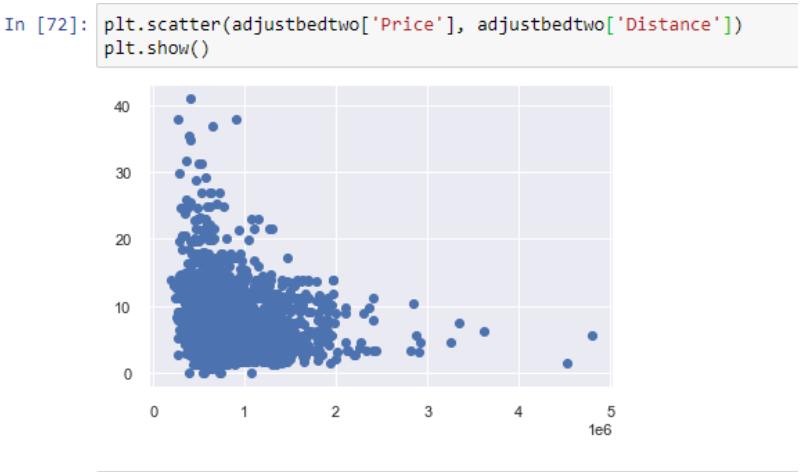


```
ax = sns.kdeplot(np.log(adjustbedtwo.Price))
```

Distance of 2 beds

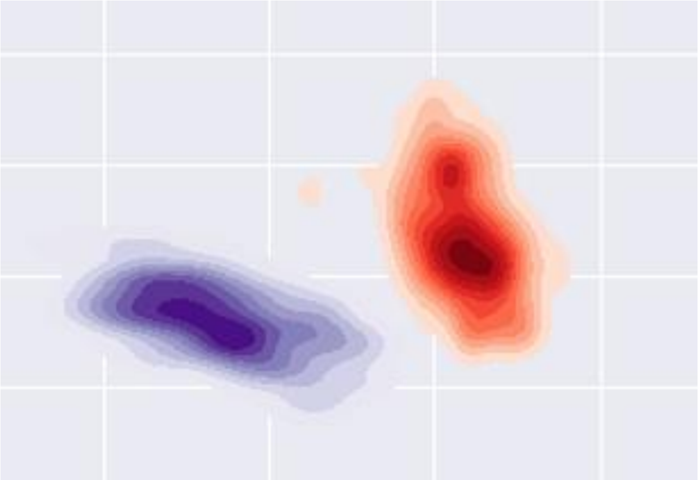


```
In [71]: ax = sns.kdeplot(adjustbedtwo.Distance)
```

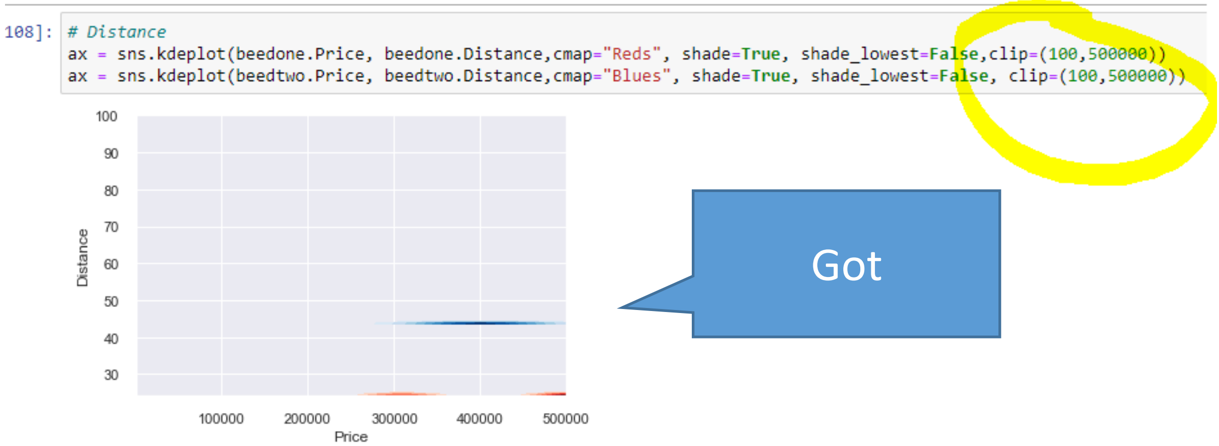


```
In [72]: plt.scatter(adjustbedtwo['Price'], adjustbedtwo['Distance'])
         plt.show()
```

Wanted a good KDE Plot



Wanted



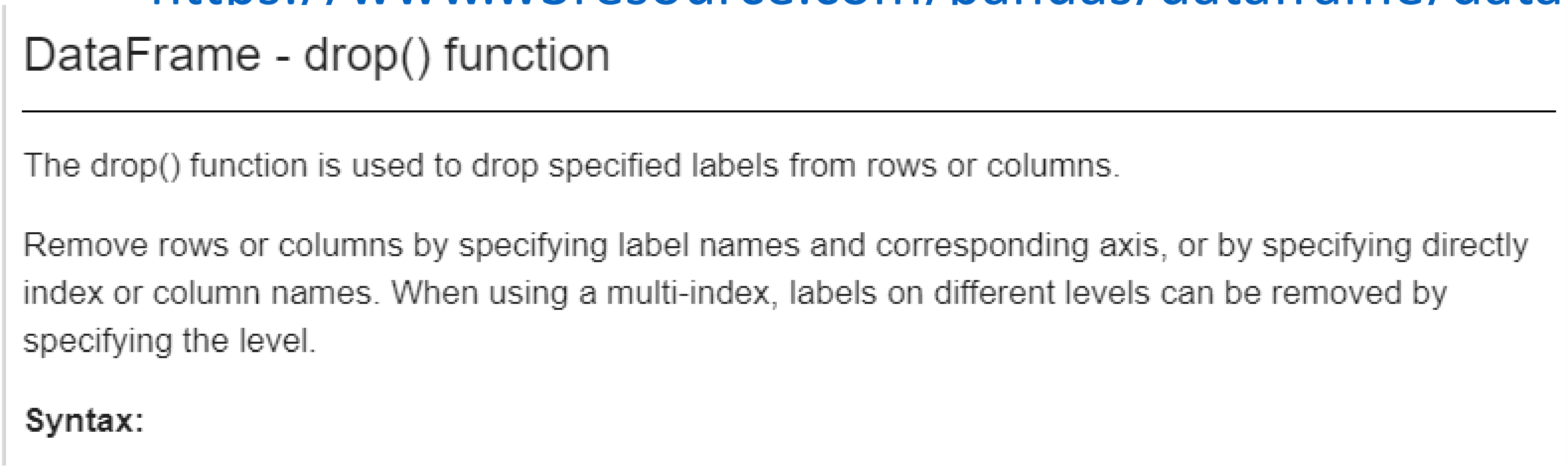
```
108]: # Distance
ax = sns.kdeplot(beedone.Price, beedone.Distance,cmap="Reds", shade=True, shade_lowest=False,clip=(100,500000))
ax = sns.kdeplot(beedtwo.Price, beedtwo.Distance,cmap="Blues", shade=True, shade_lowest=False, clip=(100,500000))
```

Got

Started Fixing the Data

Dropping

- reduced_X_train = X_train.drop(cols_with_missing, axis=1)
- https://www.w3resource.com/pandas/dataframe/dataframe-drop.php

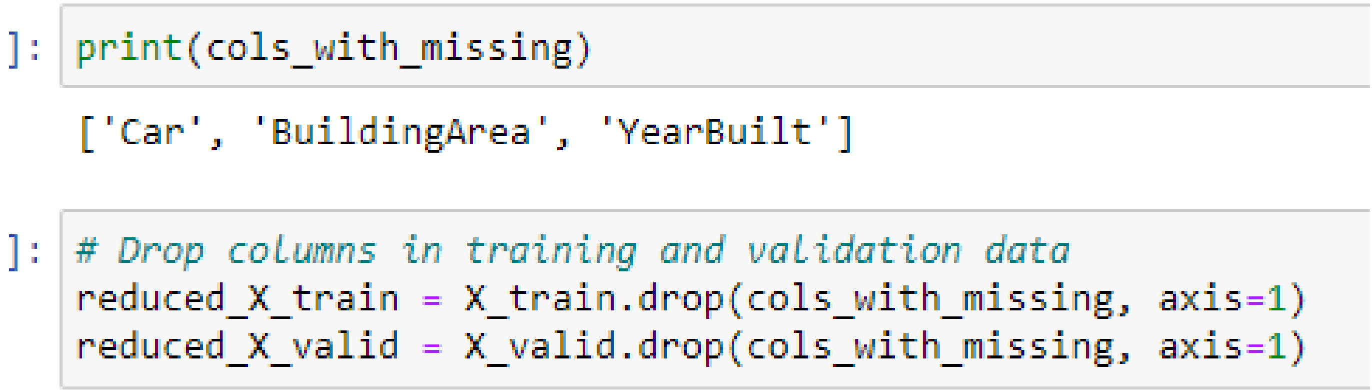


DataFrame - drop() function

The drop() function is used to drop specified labels from rows or columns.

Remove rows or columns by specifying label names and corresponding axis, or by specifying directly index or column names. When using a multi-index, labels on different levels can be removed by specifying the level.

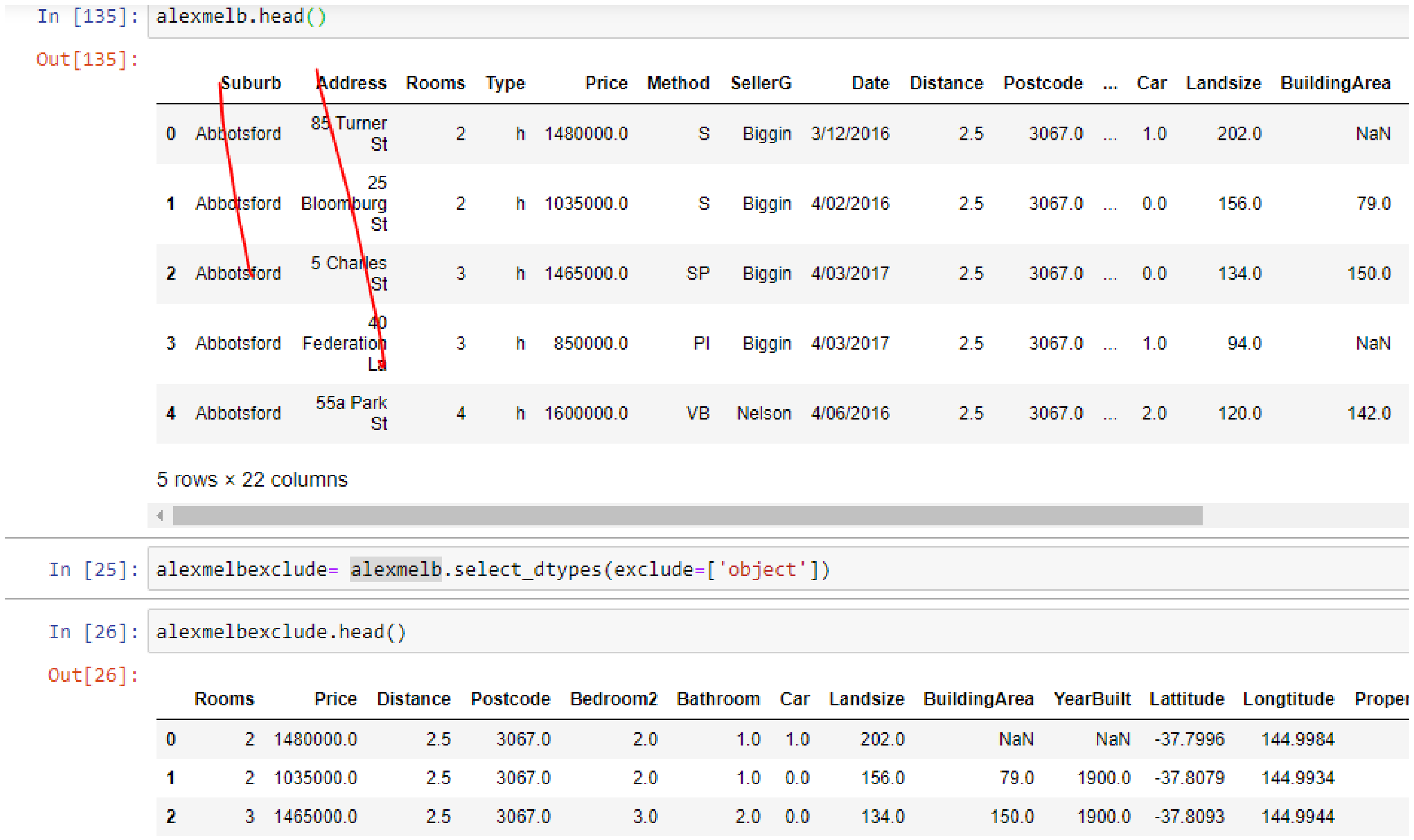
Syntax:



```
]: print(cols_with_missing)

   ['Car', 'BuildingArea', 'YearBuilt']

]: # Drop columns in training and validation data
   reduced_X_train = X_train.drop(cols_with_missing, axis=1)
   reduced_X_valid = X_valid.drop(cols_with_missing, axis=1)
```

Excluding

alexmelbexclude= alexmelb.select_dtypes(exclude=['object'])



```
In [135]: alexmelb.head()
```

Out[135]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...	Car	Landsize	BuildingArea
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5	3067.0	...	1.0	202.0	NaN
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5	3067.0	...	0.0	156.0	79.0
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5	3067.0	...	0.0	134.0	150.0
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	2.5	3067.0	...	1.0	94.0	NaN
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	2.5	3067.0	...	2.0	120.0	142.0

5 rows × 22 columns

```
In [25]: alexmelbexclude= alexmelb.select_dtypes(exclude=['object'])
```

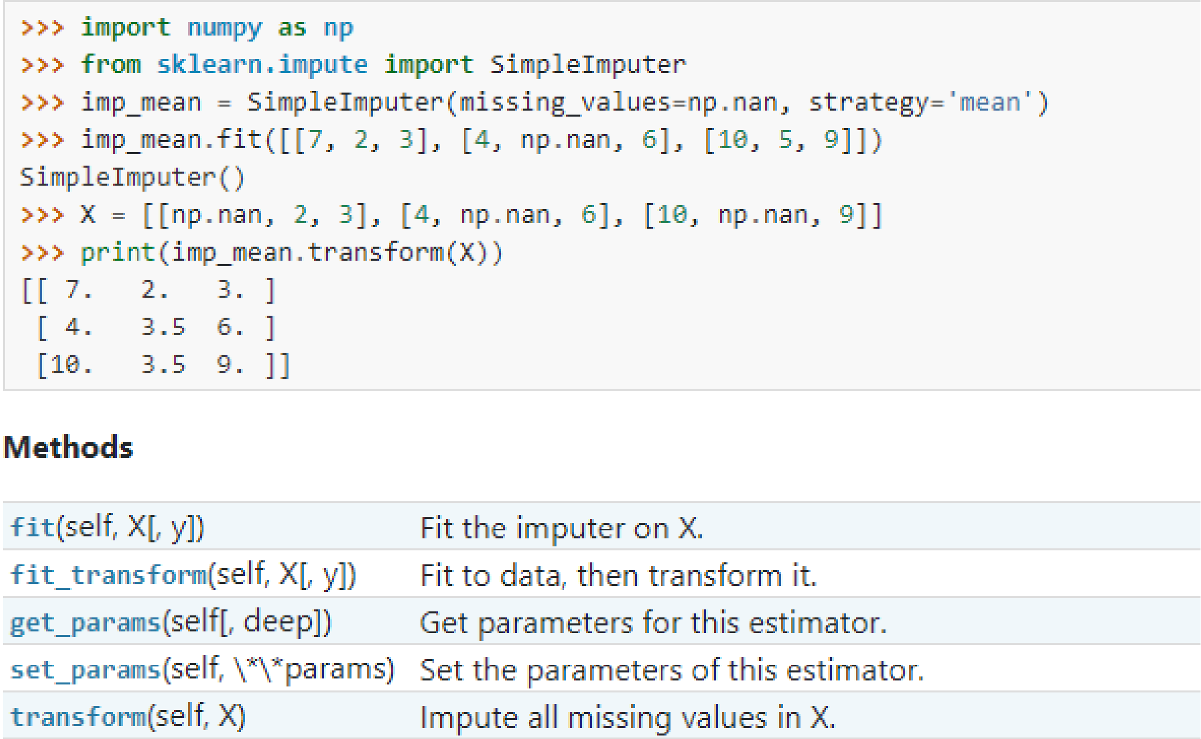
```
In [26]: alexmelbexclude.head()
```

Out[26]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude	Proper
0	2	1480000.0	2.5	3067.0	2.0	1.0	1.0	202.0	NaN	NaN	-37.7996	144.9984	
1	2	1035000.0	2.5	3067.0	2.0	1.0	0.0	156.0	79.0	1900.0	-37.8079	144.9934	
2	3	1465000.0	2.5	3067.0	3.0	2.0	0.0	134.0	150.0	1900.0	-37.8093	144.9944	

The Simple Imputer Library is an easy option

https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

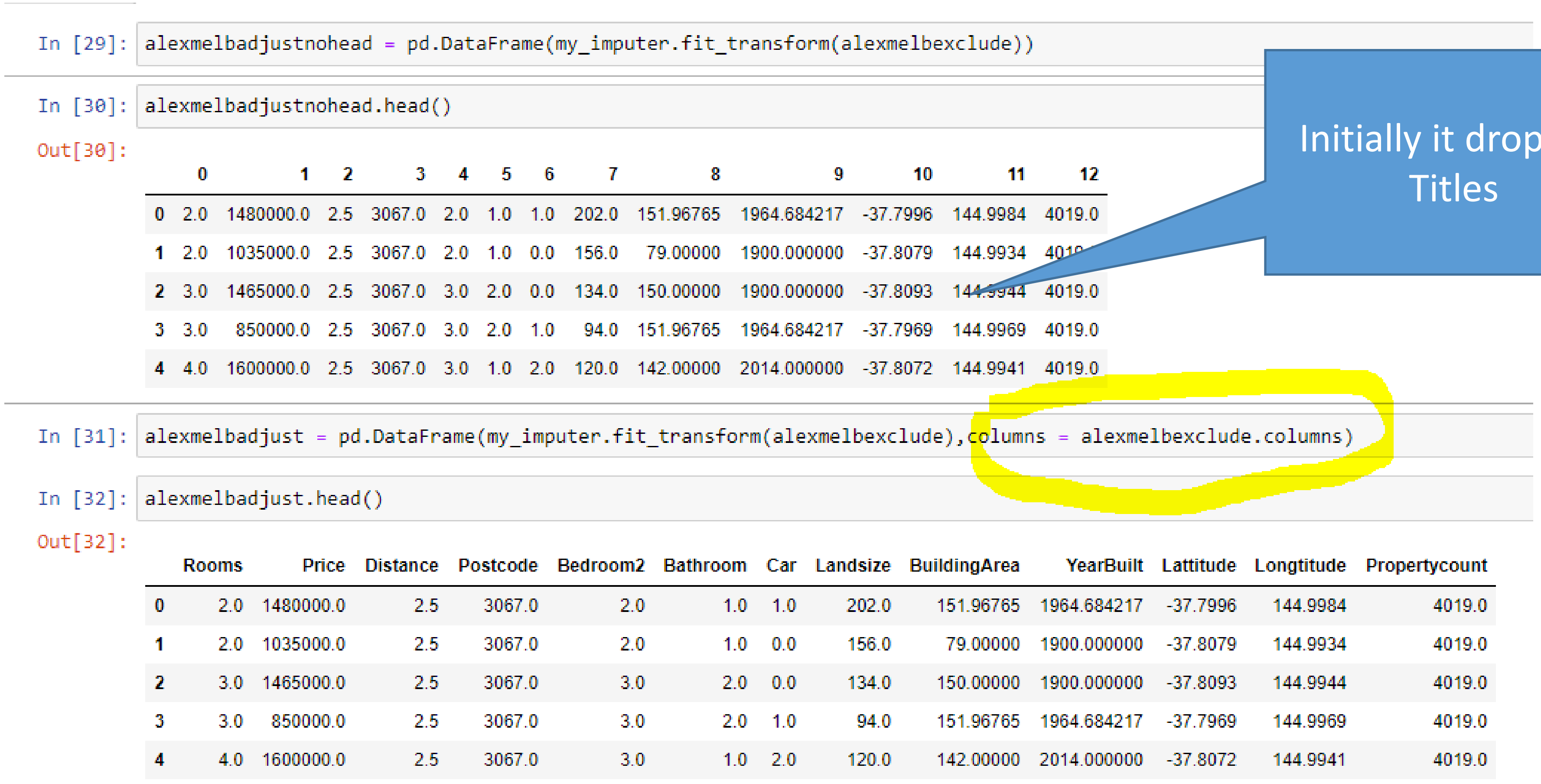


```
>>> import numpy as np
>>> from sklearn.impute import SimpleImputer
>>> imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
>>> imp_mean.fit([[7, 2, 3], [4, np.nan, 6], [10, 5, 9]])
SimpleImputer()
>>> X = [[np.nan, 2, 3], [4, np.nan, 6], [10, np.nan, 9]]
>>> print(imp_mean.transform(X))
[[ 7.   2.   3. ]
 [ 4.   3.5  6. ]
 [10.   3.5  9. ]]
```

Methods

fit(self, X[, y])	Fit the imputer on X.
fit_transform(self, X[, y])	Fit to data, then transform it.
get_params(self[, deep])	Get parameters for this estimator.
set_params(self, **params)	Set the parameters of this estimator.
transform(self, X)	Impute all missing values in X.

I used it myself

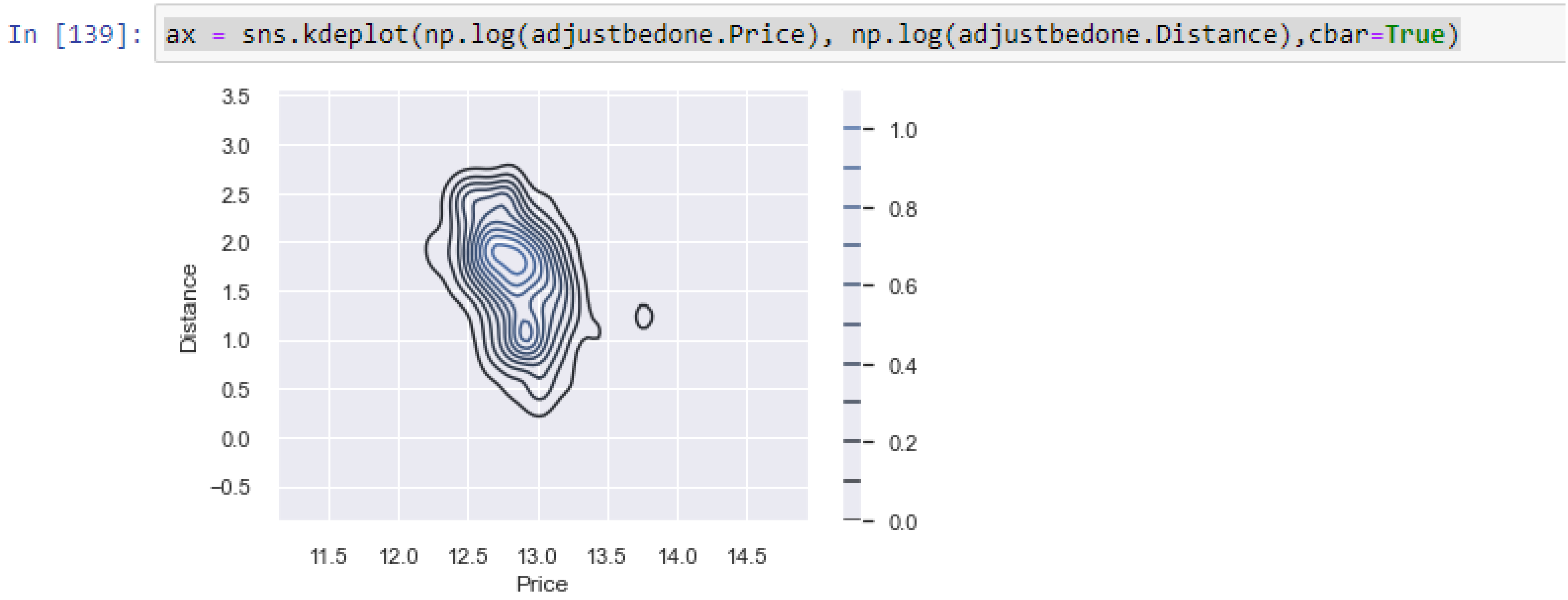


```
In [29]: alexmelbadjustnohead = pd.DataFrame(my_imputer.fit_transform(alexmelbexclude))

In [30]: alexmelbadjustnohead.head()

Out[30]:
     0          1    2       3    4    5    6      7          8            9        10        11       12
0  2.0  1480000.0  2.5  3067.0  2.0  1.0  1.0  202.0  151.96765  1964.684217  -37.7996  144.9984  4019.0
1  2.0  1035000.0  2.5  3067.0  2.0  1.0  0.0  156.0   79.00000  1900.000000  -37.8079  144.9934  401
2  3.0  1465000.0  2.5  3067.0  3.0  2.0  0.0  134.0  150.00000  1900.000000  -37.8093  144.9944  4019.0
3  3.0   850000.0  2.5  3067.0  3.0  2.0  1.0   94.0  151.96765  1964.684217  -37.7969  144.9969  4019.0
4  4.0  1600000.0  2.5  3067.0  3.0  1.0  2.0  120.0  142.00000  2014.000000  -37.8072  144.9941  4019.0

In [31]: alexmelbadjust = pd.DataFrame(my_imputer.fit_transform(alexmelbexclude),columns = alexmelbexclude.columns)

In [32]: alexmelbadjust.head()

Out[32]:
   Rooms      Price  Distance  Postcode  Bedroom2  Bathroom  Car  Landsize  BuildingArea    YearBuilt  Lattitude  Longtitude  Propertycount
0    2.0  1480000.0       2.5    3067.0       2.0       1.0  1.0     202.0     151.96765  1964.684217   -37.7996    144.9984         4019.0
1    2.0  1035000.0       2.5    3067.0       2.0       1.0  0.0     156.0      79.00000  1900.000000   -37.8079    144.9934         4019.0
2    3.0  1465000.0       2.5    3067.0       3.0       2.0  0.0     134.0     150.00000  1900.000000   -37.8093    144.9944         4019.0
3    3.0   850000.0       2.5    3067.0       3.0       2.0  1.0      94.0     151.96765  1964.684217   -37.7969    144.9969         4019.0
4    4.0  1600000.0       2.5    3067.0       3.0       1.0  2.0     120.0     142.00000  2014.000000   -37.8072    144.9941         4019.0
```

Initially it dropped Titles

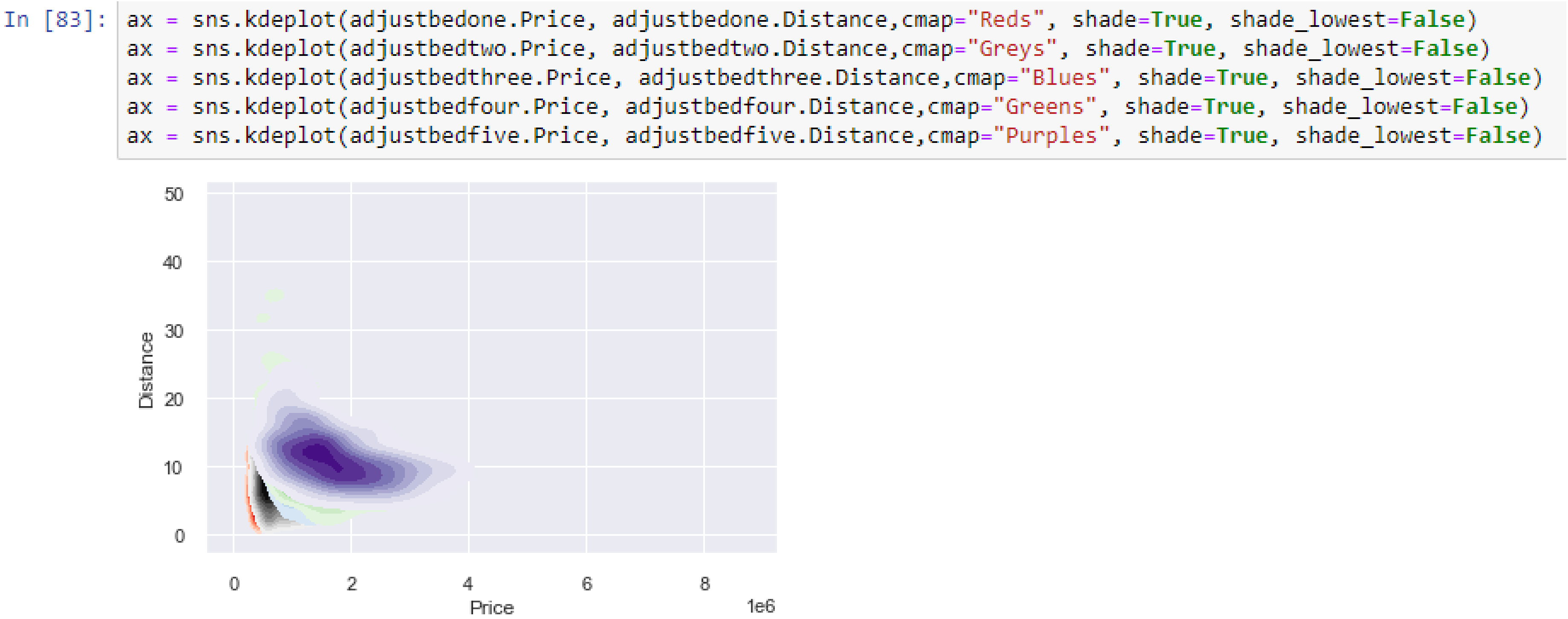
Then Presenting Got Better

Started Getting Somewhere



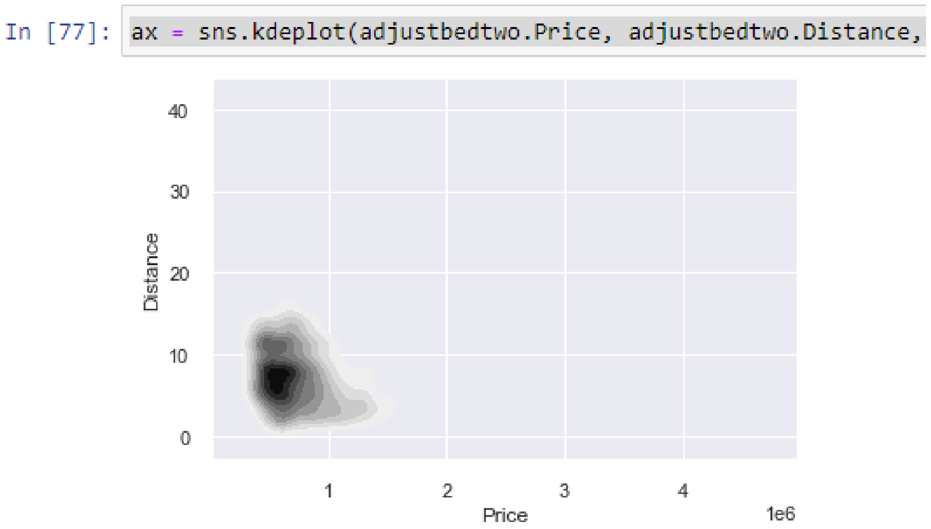
```
In [139]: ax = sns.kdeplot(np.log(adjustbedone.Price), np.log(adjustbedone.Distance),cbar=True)
```

Not Work

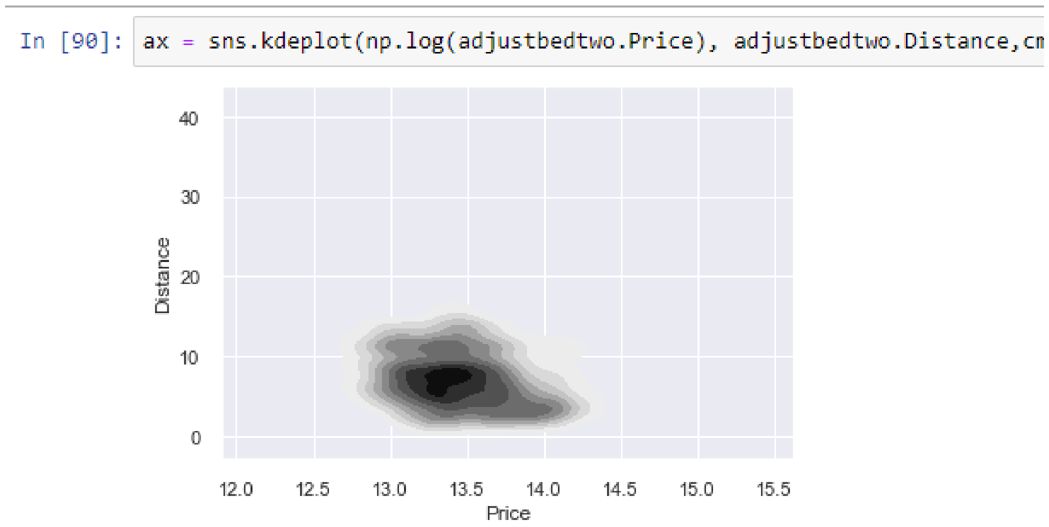


```
In [83]: ax = sns.kdeplot(adjustbedone.Price, adjustbedone.Distance,cmap="Reds", shade=True, shade_lowest=False)
         ax = sns.kdeplot(adjustbedtwo.Price, adjustbedtwo.Distance,cmap="Greys", shade=True, shade_lowest=False)
         ax = sns.kdeplot(adjustbedthree.Price, adjustbedthree.Distance,cmap="Blues", shade=True, shade_lowest=False)
         ax = sns.kdeplot(adjustbedfour.Price, adjustbedfour.Distance,cmap="Greens", shade=True, shade_lowest=False)
         ax = sns.kdeplot(adjustbedfive.Price, adjustbedfive.Distance,cmap="Purples", shade=True, shade_lowest=False)
```

Graphs started Looking OK

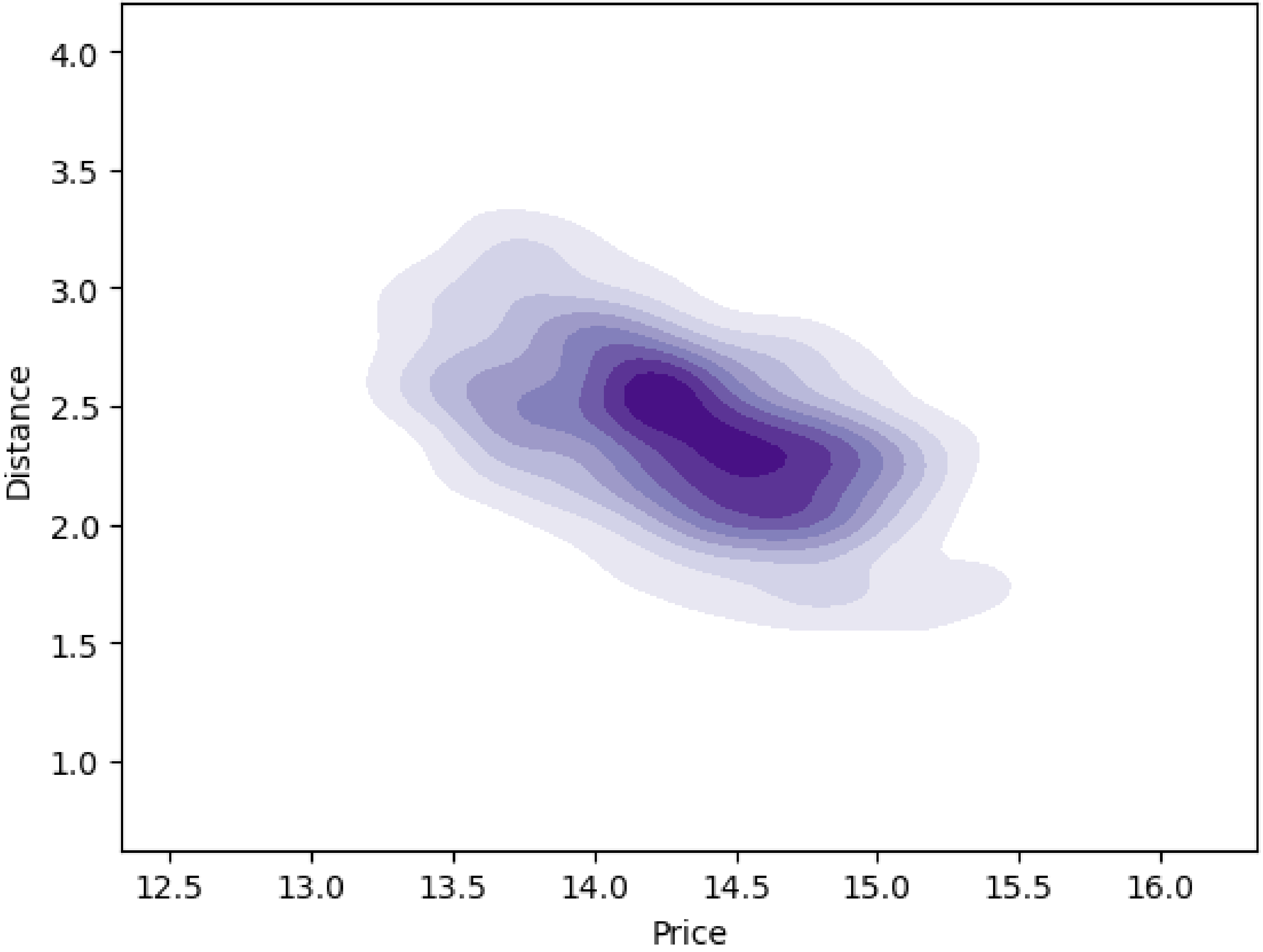


```
In [77]: ax = sns.kdeplot(adjustbedtwo.Price, adjustbedtwo.Distance,
```



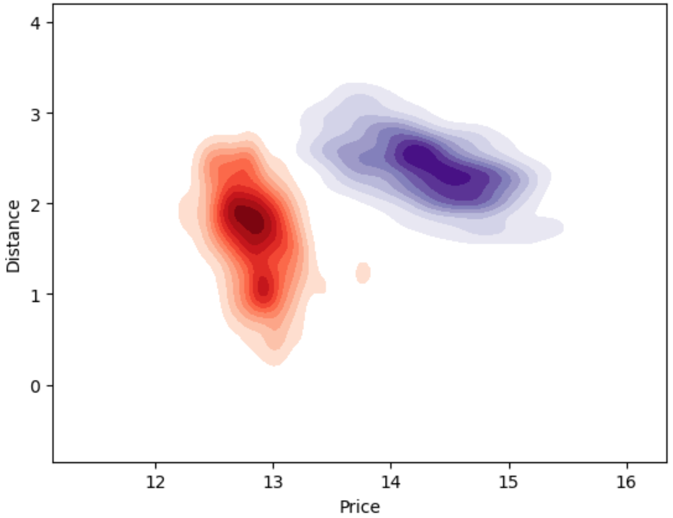
```
In [90]: ax = sns.kdeplot(np.log(adjustbedtwo.Price), adjustbedtwo.Distance,cm
```

I like looking at them

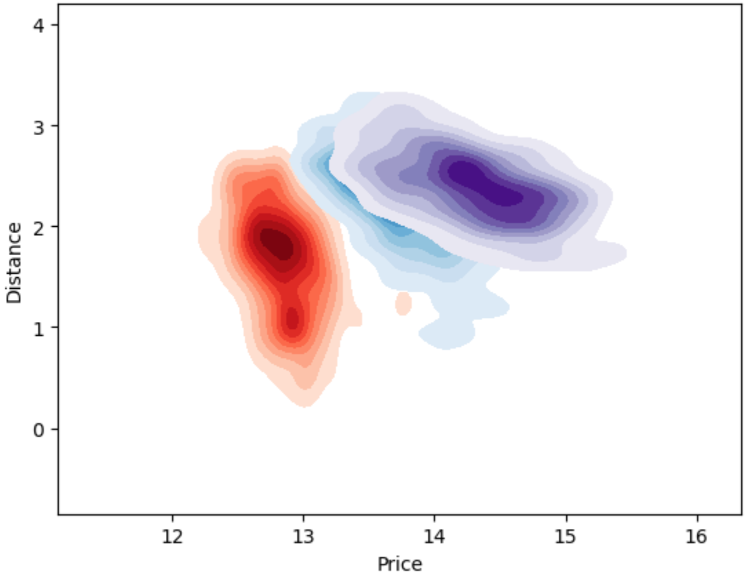


Works best with limited options – 1 bed Vs 5 Bed

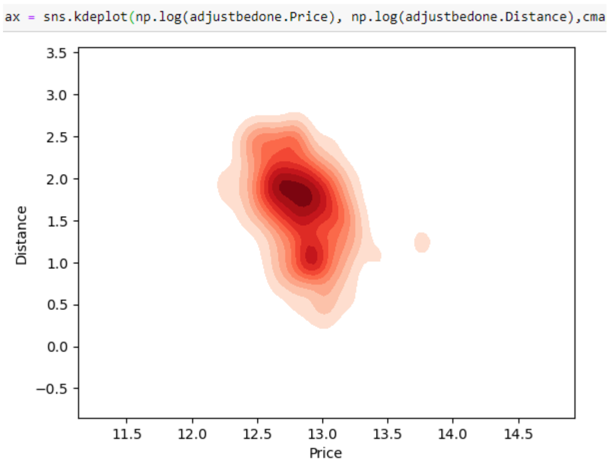
1 Bed Vs 5 Bed



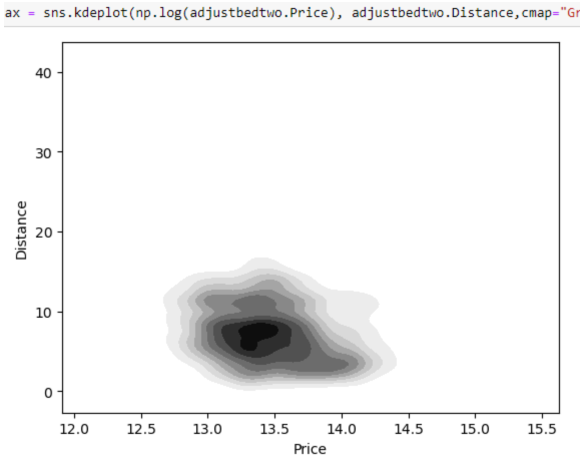
1 Bed Vs 3 bed Vs 5 Bed



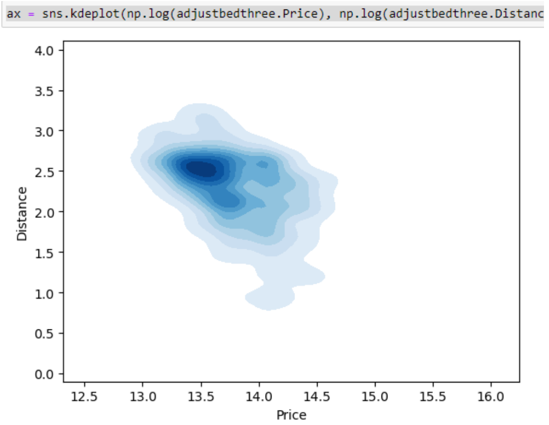
Graphs Comparison



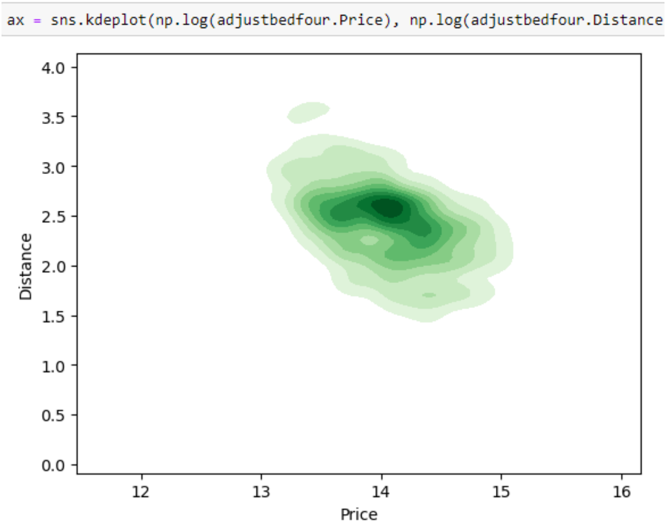
```
ax = sns.kdeplot(np.log(adjustbedone.Price), np.log(adjustbedone.Distance),cma
```



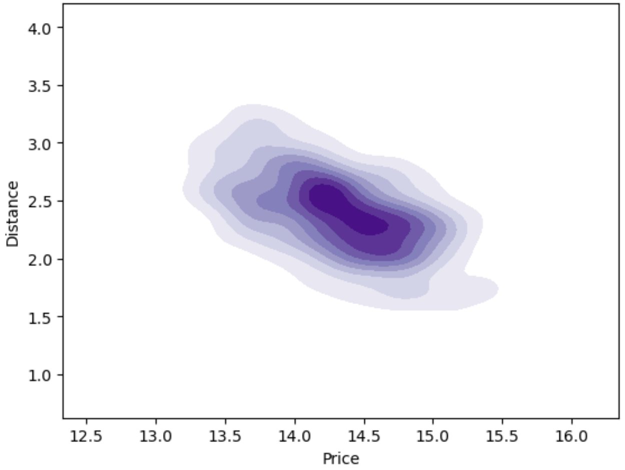
```
ax = sns.kdeplot(np.log(adjustbedtwo.Price), adjustbedtwo.Distance,cmap="Gr
```



```
ax = sns.kdeplot(np.log(adjustbedthree.Price), np.log(adjustbedthree.Distanc
```

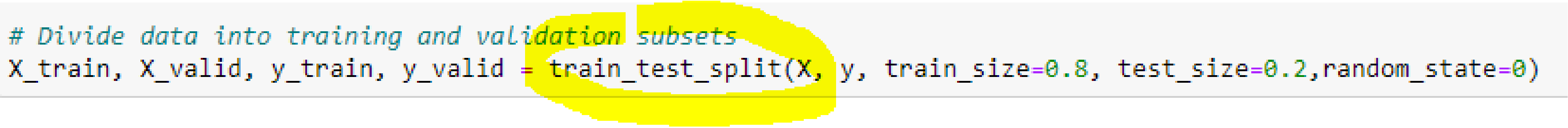


```
ax = sns.kdeplot(np.log(adjustbedfour.Price), np.log(adjustbedfour.Distance
```



Useful

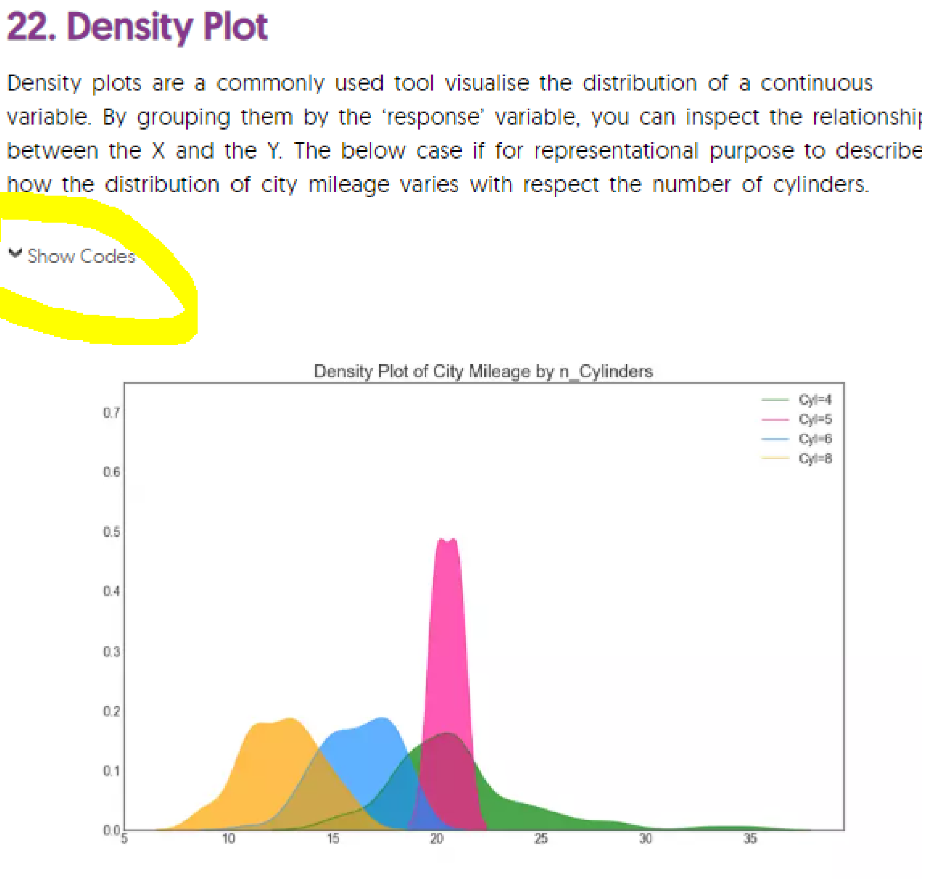
Found how to split data



```
# Divide data into training and validation subsets
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size=0.2,random_state=0)
```

Useful resource

https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/

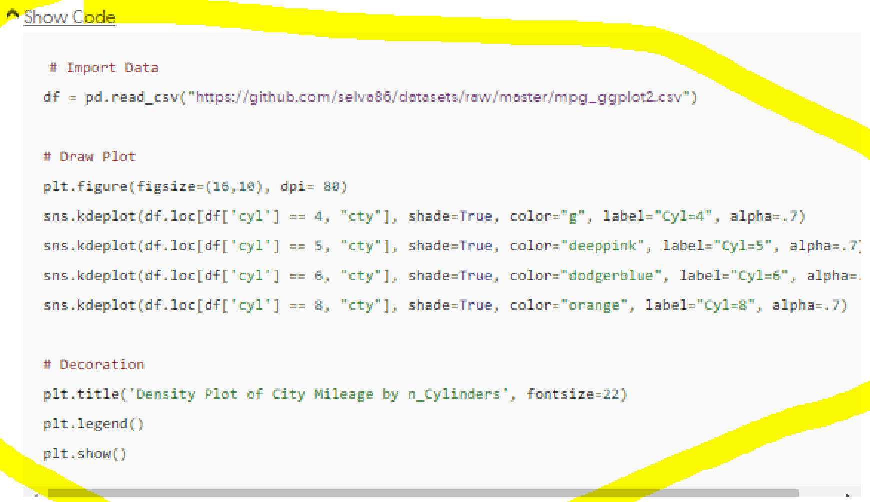


22. Density Plot

Density plots are a commonly used tool visualise the distribution of a continuous variable. By grouping them by the 'response' variable, you can inspect the relationshi between the X and the Y. The below case if for representational purpose to describe how the distribution of city mileage varies with respect the number of cylinders.

⌄ Show Codes

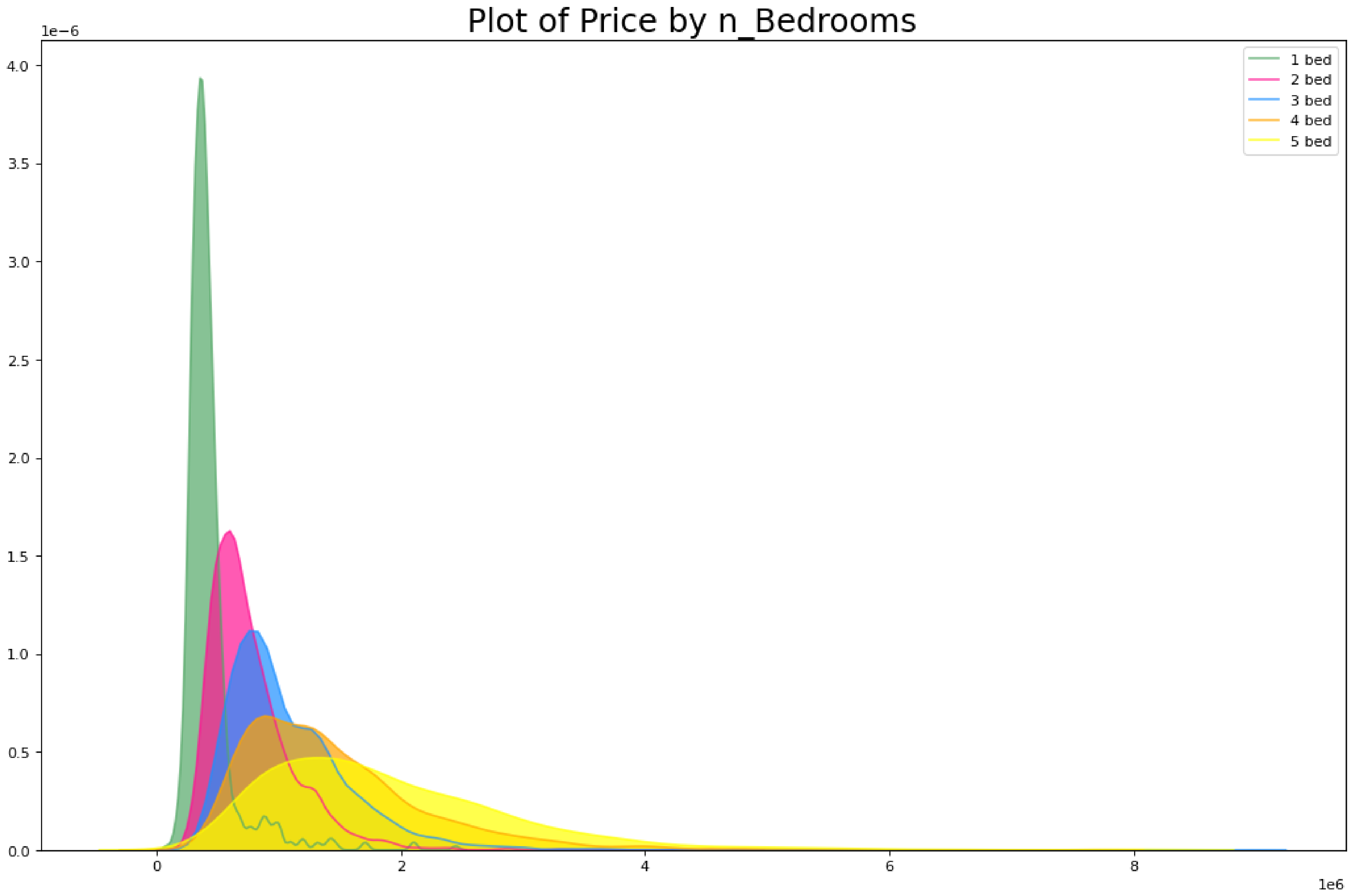
Density Plot of City Mileage by n_Cylinders



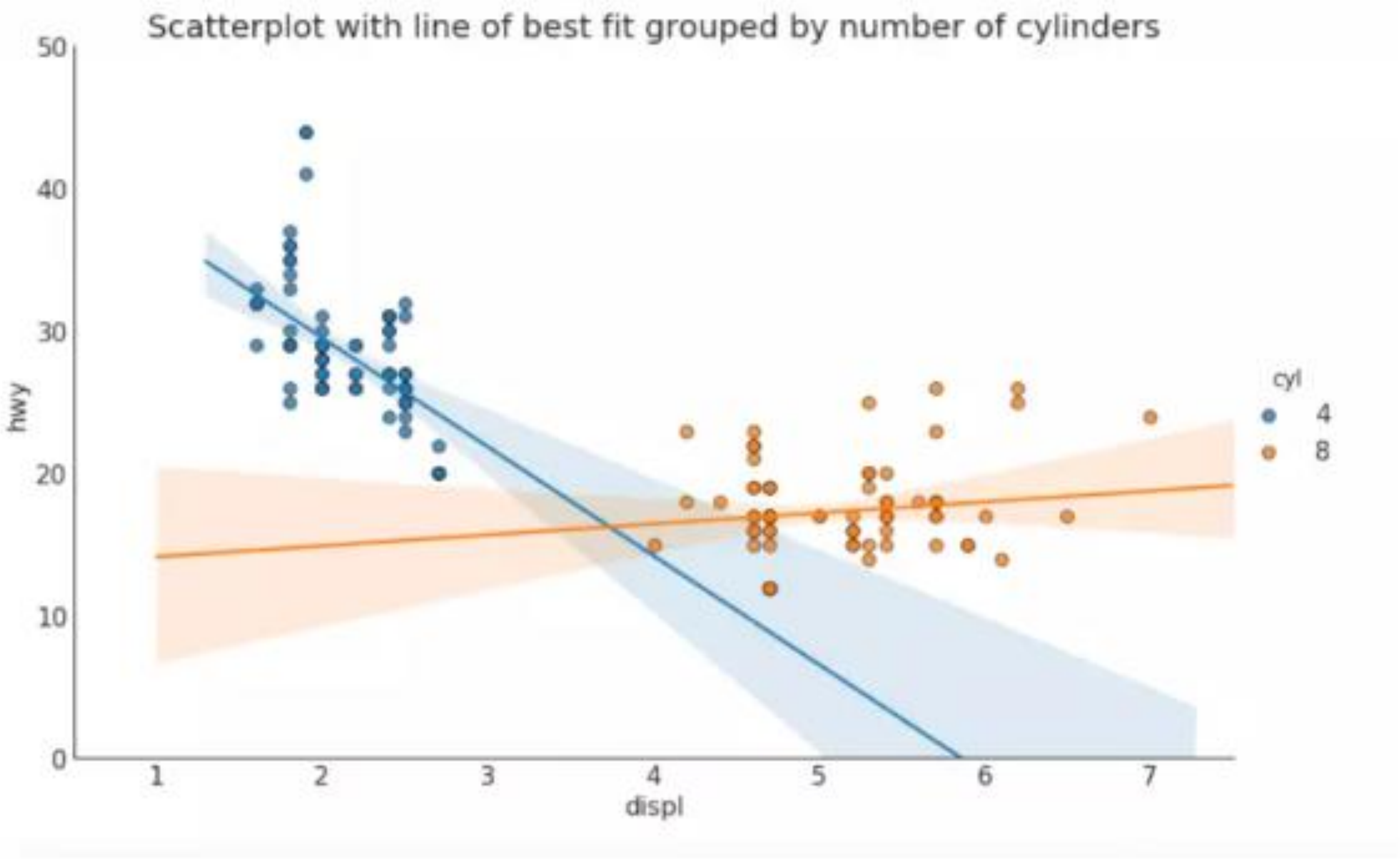
^ Show Code

```
 # Import Data
df = pd.read_csv("https://github.com/selva86/datasets/raw/master/mpg_ggplot2.csv")

# Draw Plot
plt.figure(figsize=(16,10), dpi= 80)
sns.kdeplot(df.loc[df['cyl'] == 4, "cty"], shade=True, color="g", label="Cyl=4", alpha=.7)
sns.kdeplot(df.loc[df['cyl'] == 5, "cty"], shade=True, color="deeppink", label="Cyl=5", alpha=.7
sns.kdeplot(df.loc[df['cyl'] == 6, "cty"], shade=True, color="dodgerblue", label="Cyl=6", alpha=.
sns.kdeplot(df.loc[df['cyl'] == 8, "cty"], shade=True, color="orange", label="Cyl=8", alpha=.7)

# Decoration
plt.title('Density Plot of City Mileage by n_Cylinders', fontsize=22)
plt.legend()
plt.show()
```

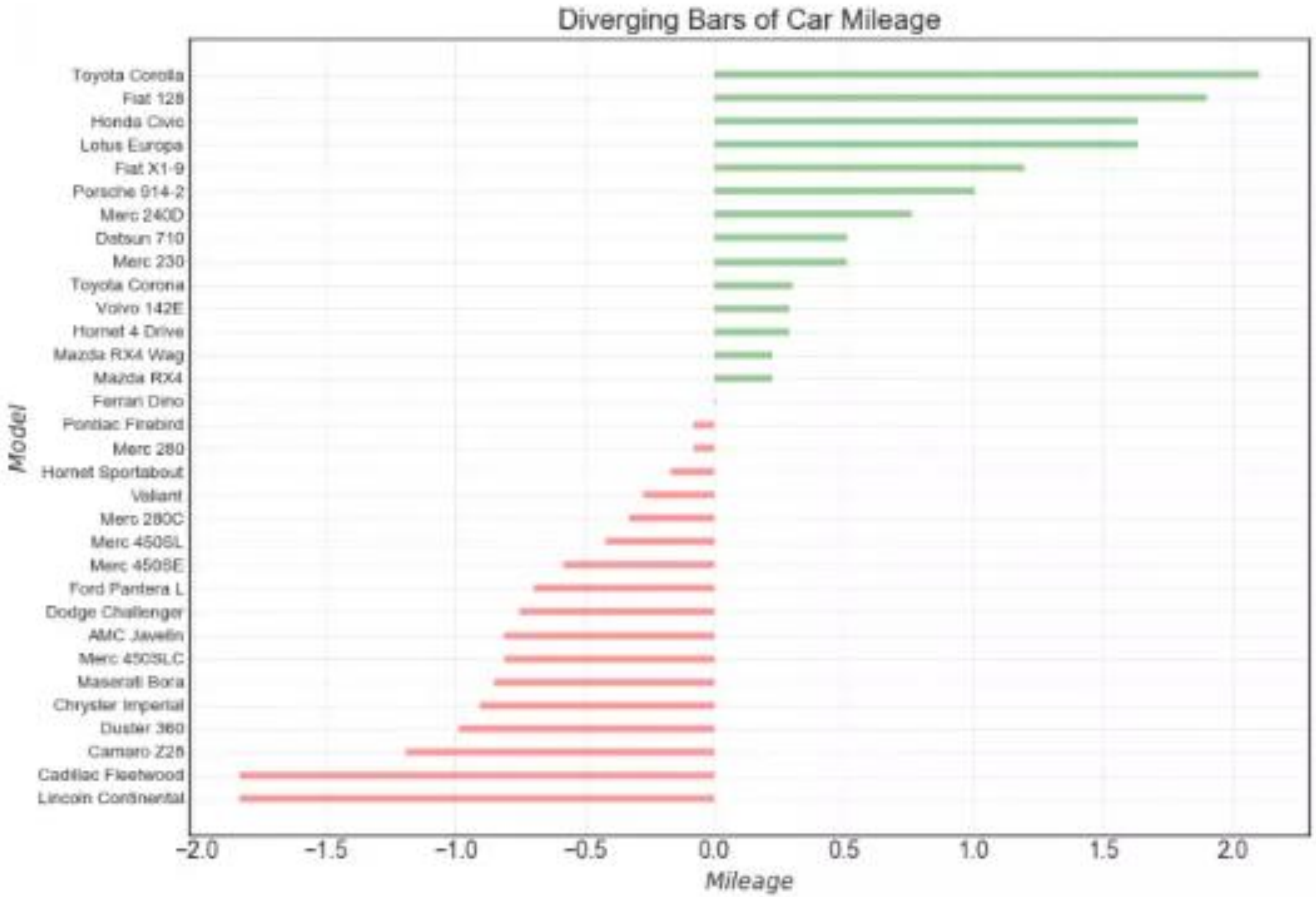
Worked well for Melbourne House Price Data



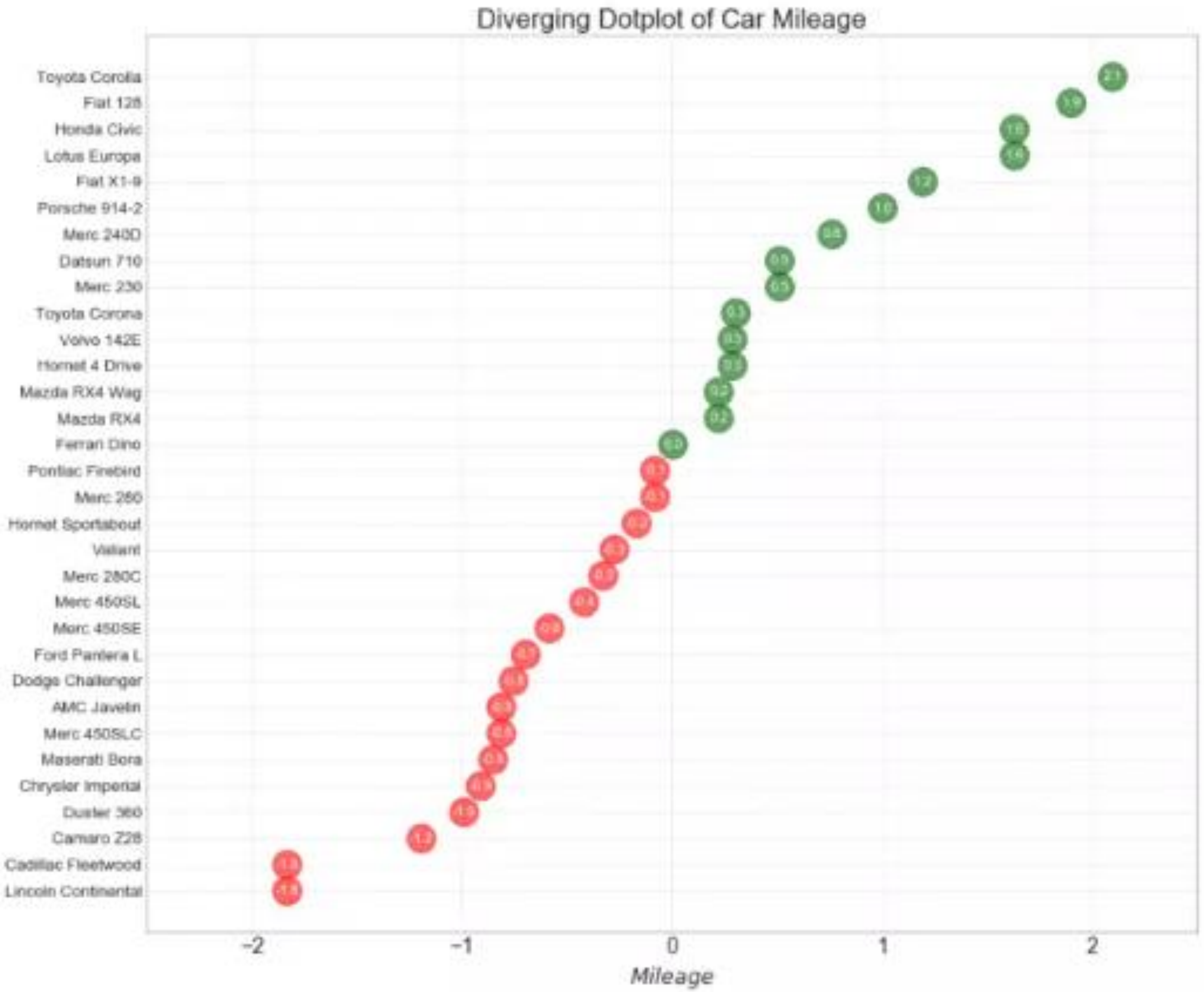
Useful resource



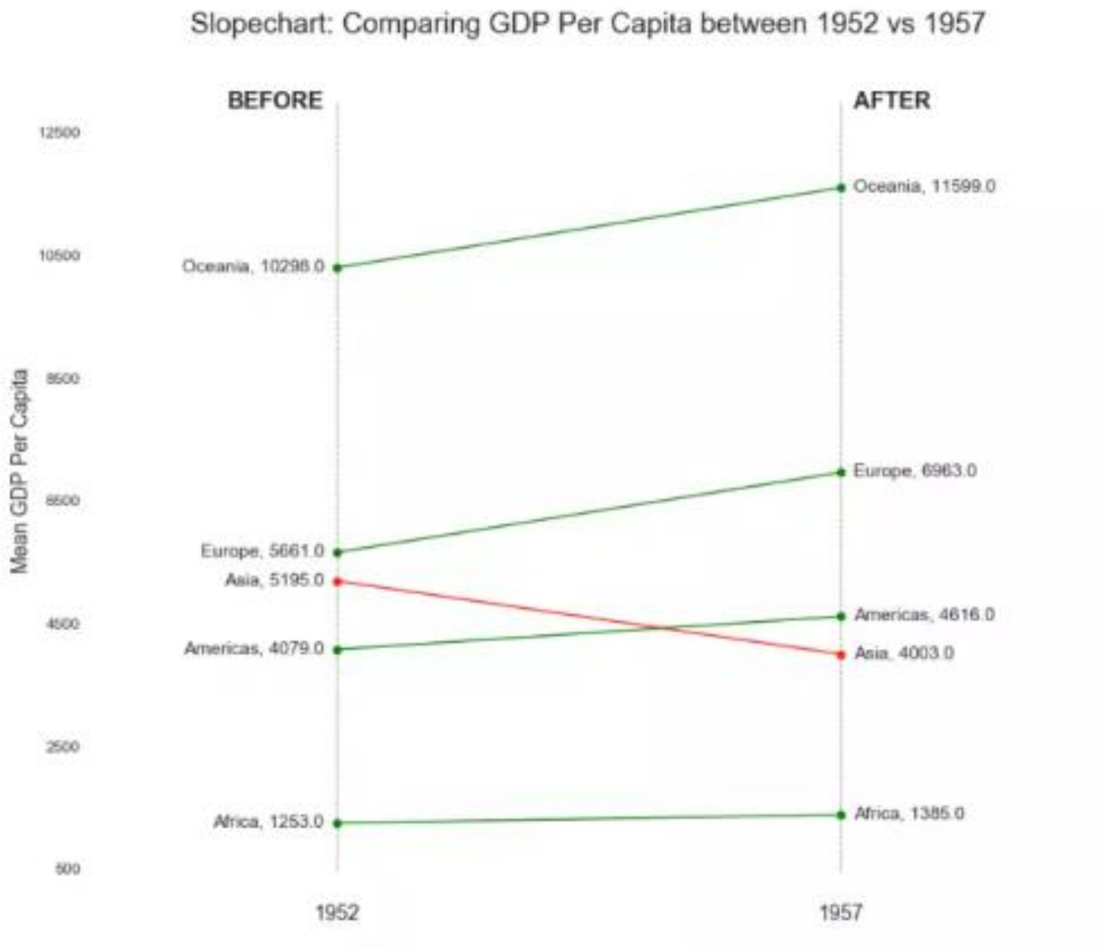
Useful resource



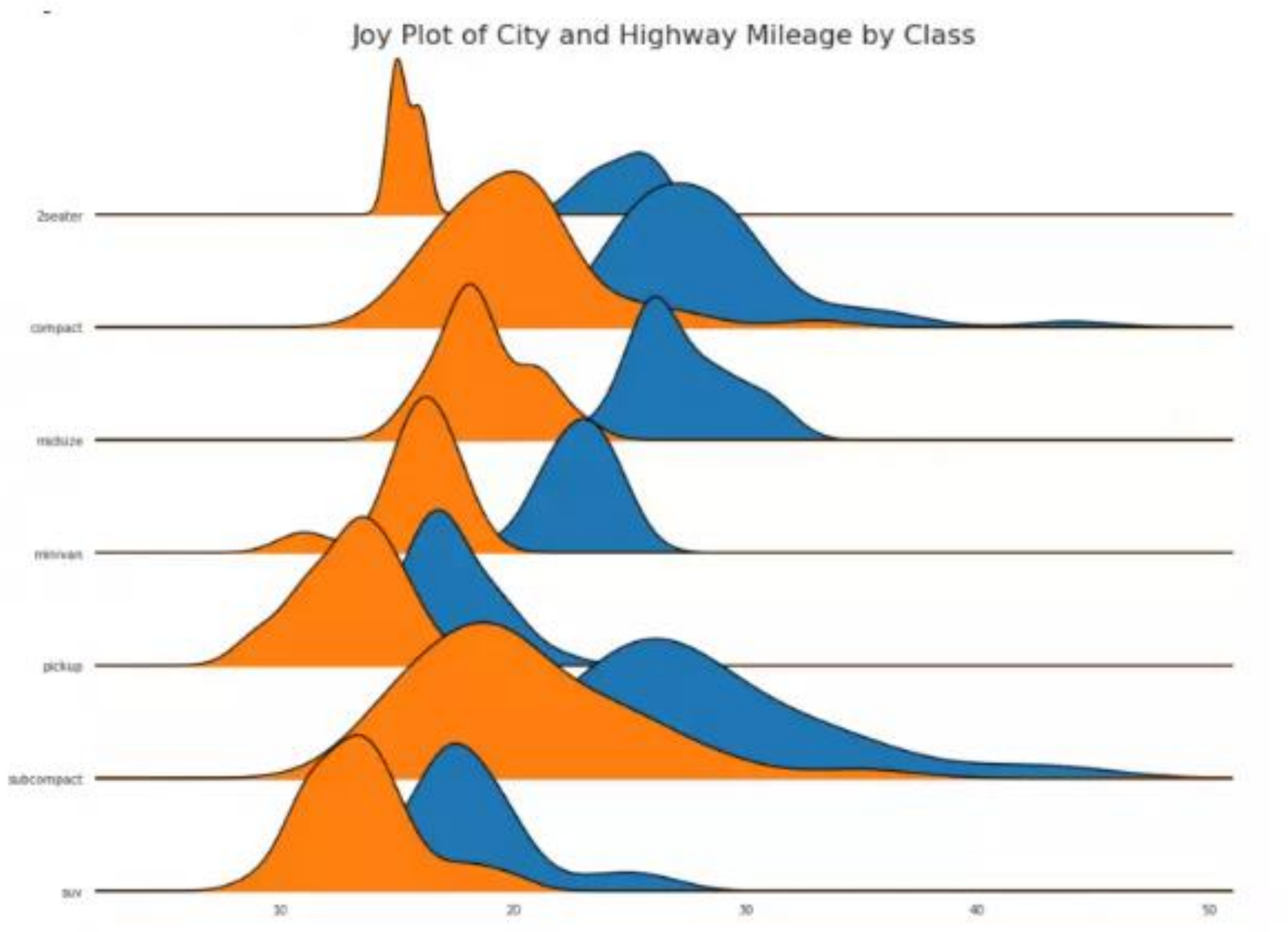
Useful resource



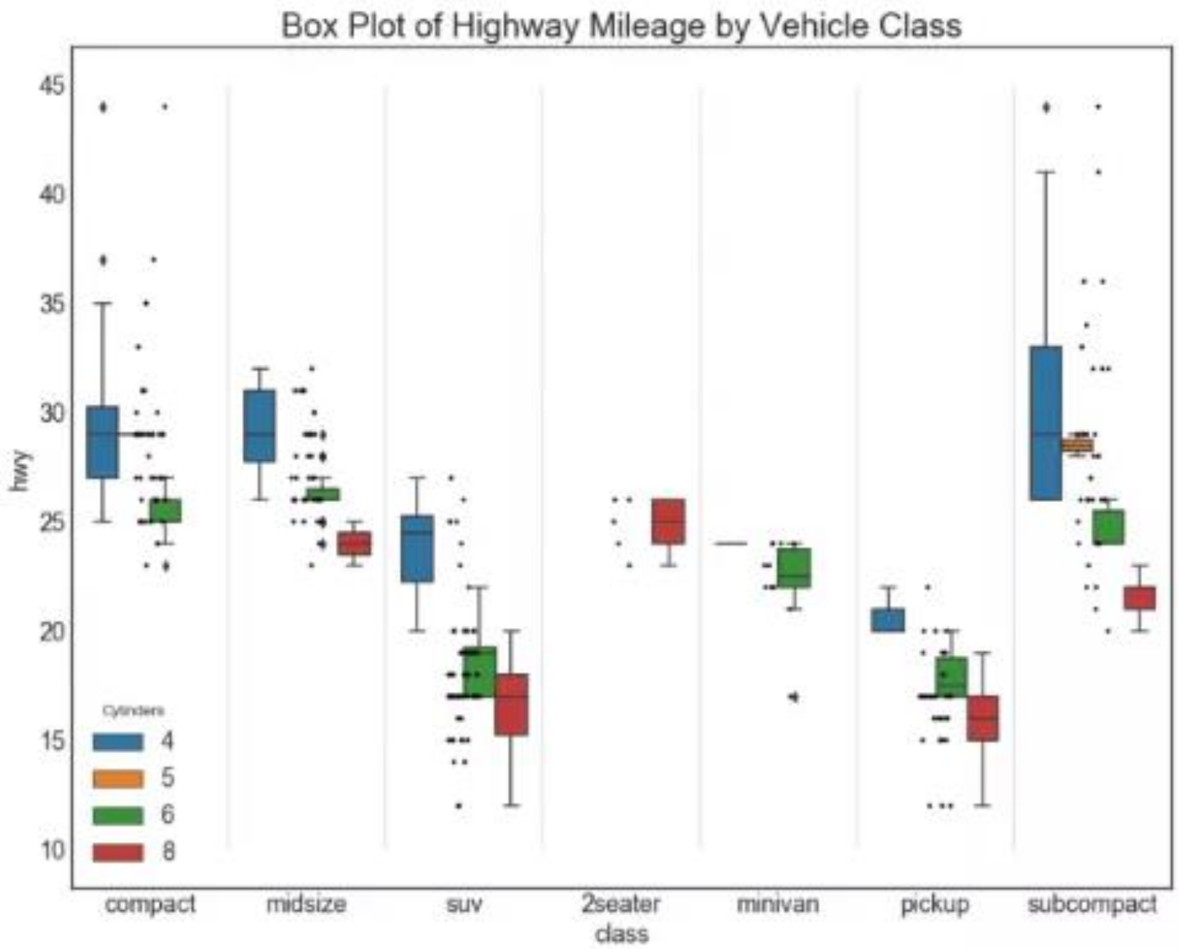
Useful resource



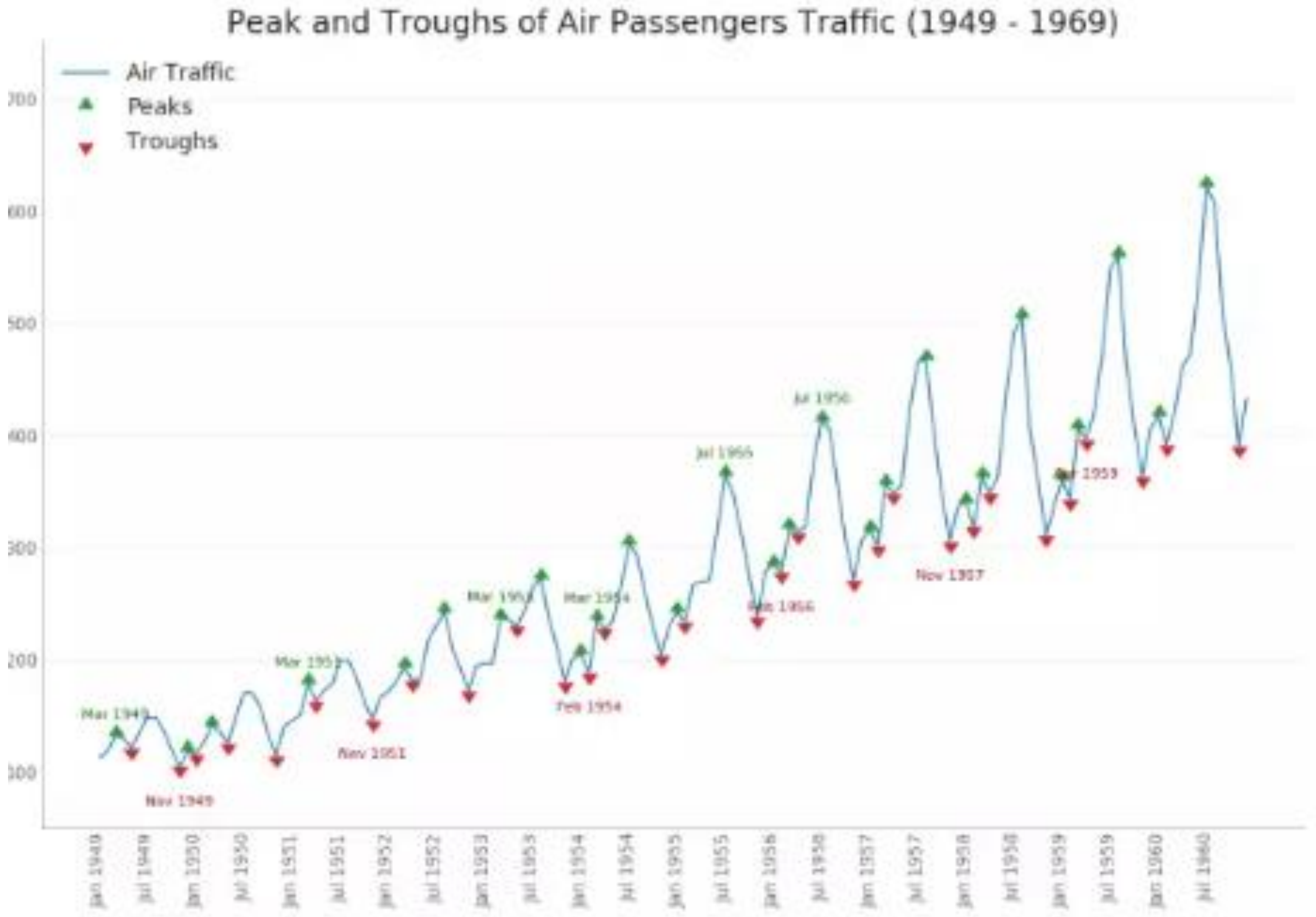
Useful resource



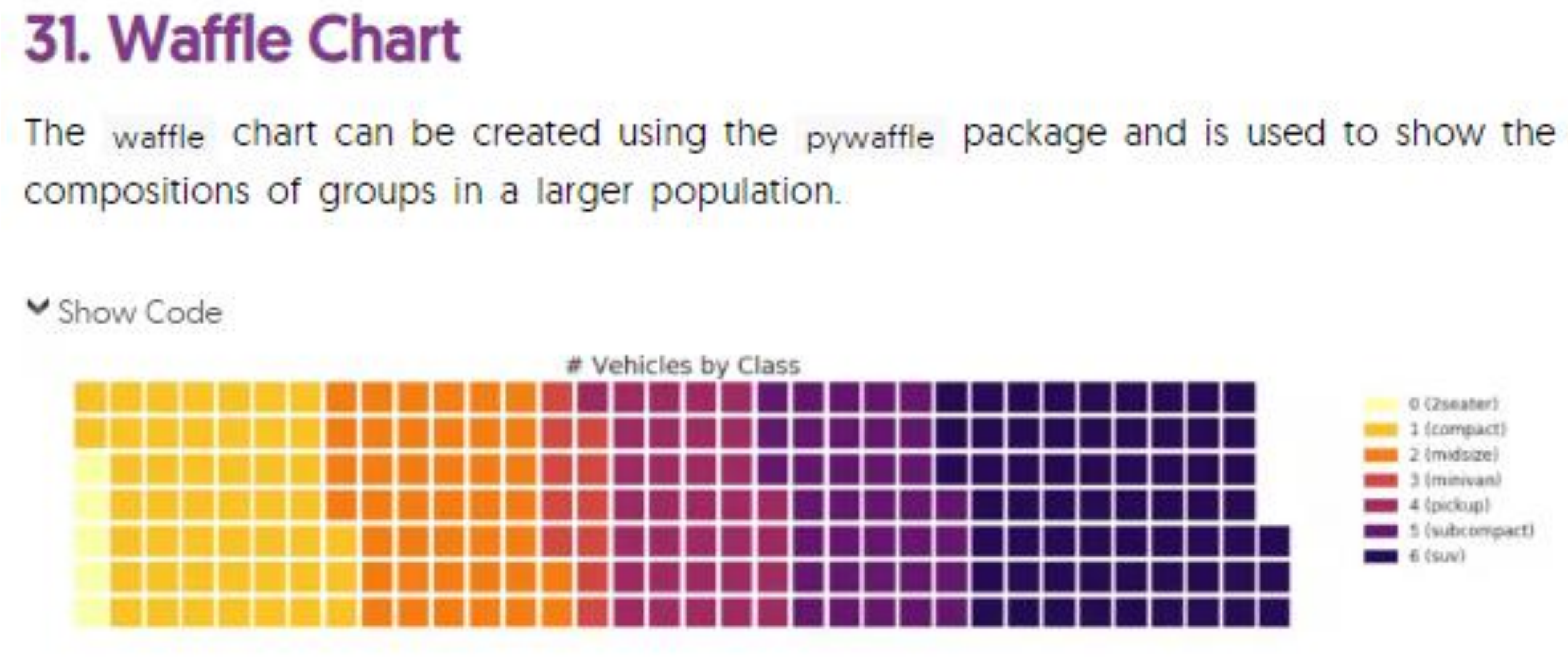
Useful resource



Useful resource



Useful resource

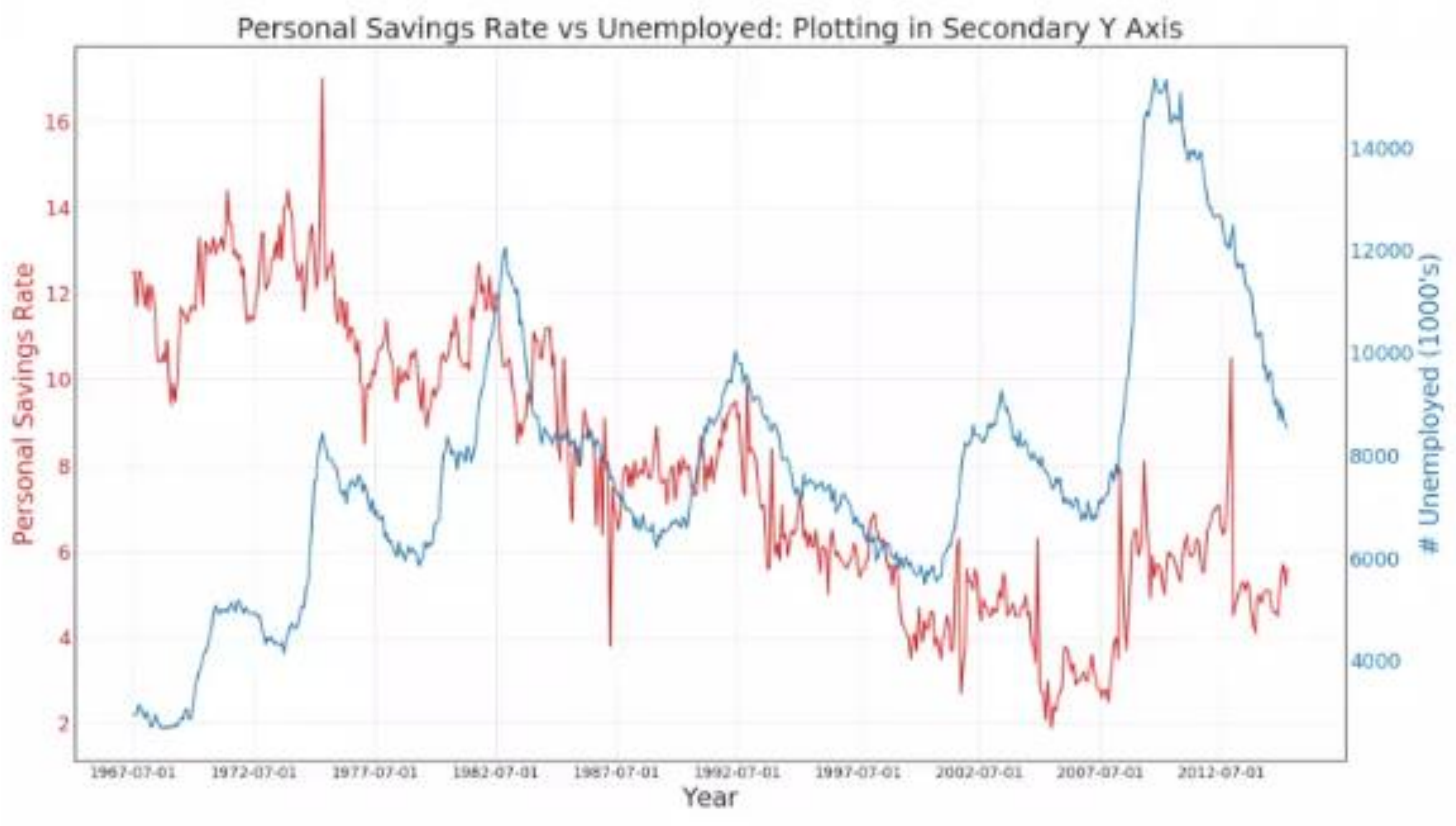


31. Waffle Chart

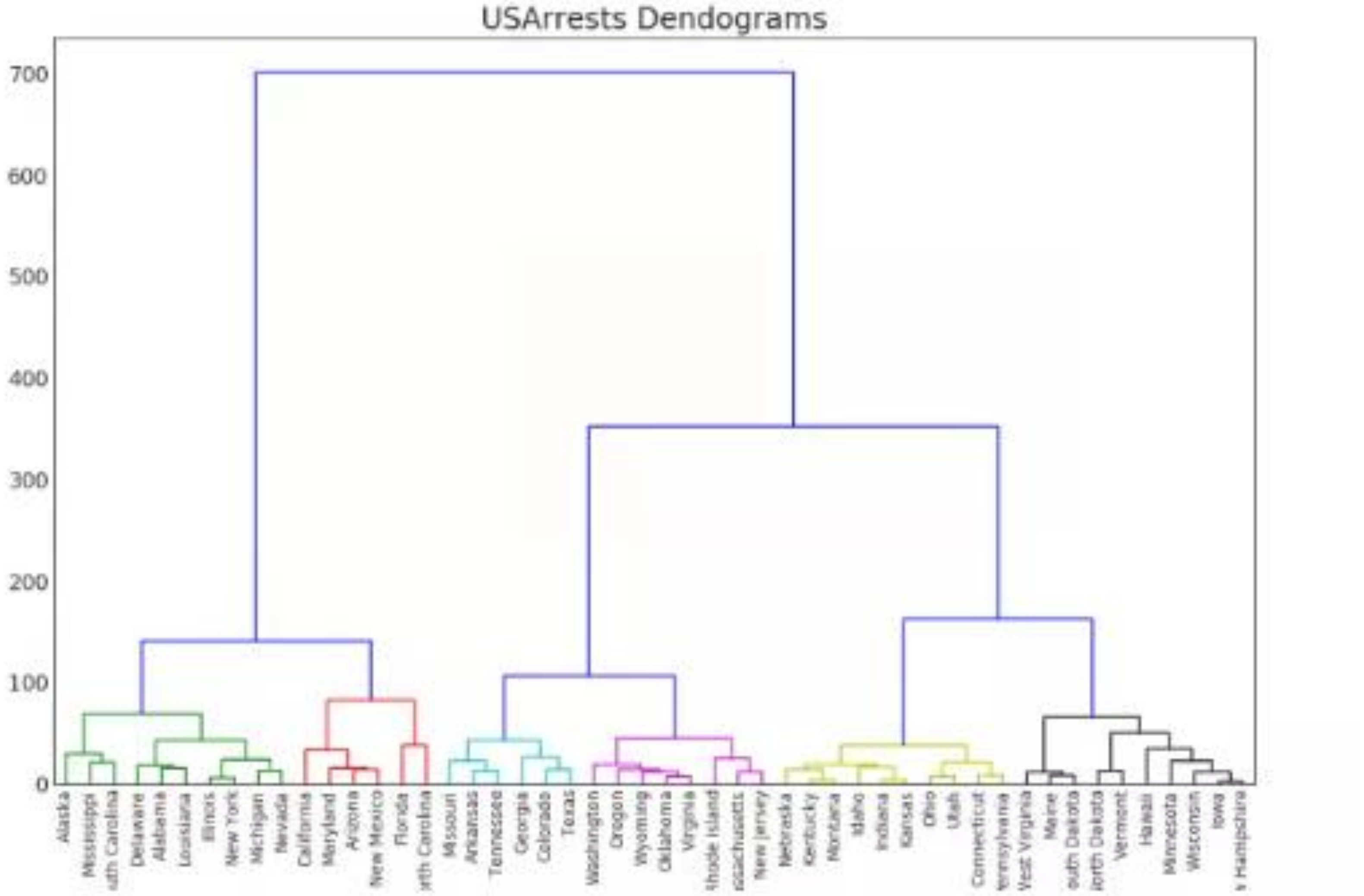
The `waffle` chart can be created using the `pywaffle` package and is used to show the compositions of groups in a larger population.

Show Code

Useful resource



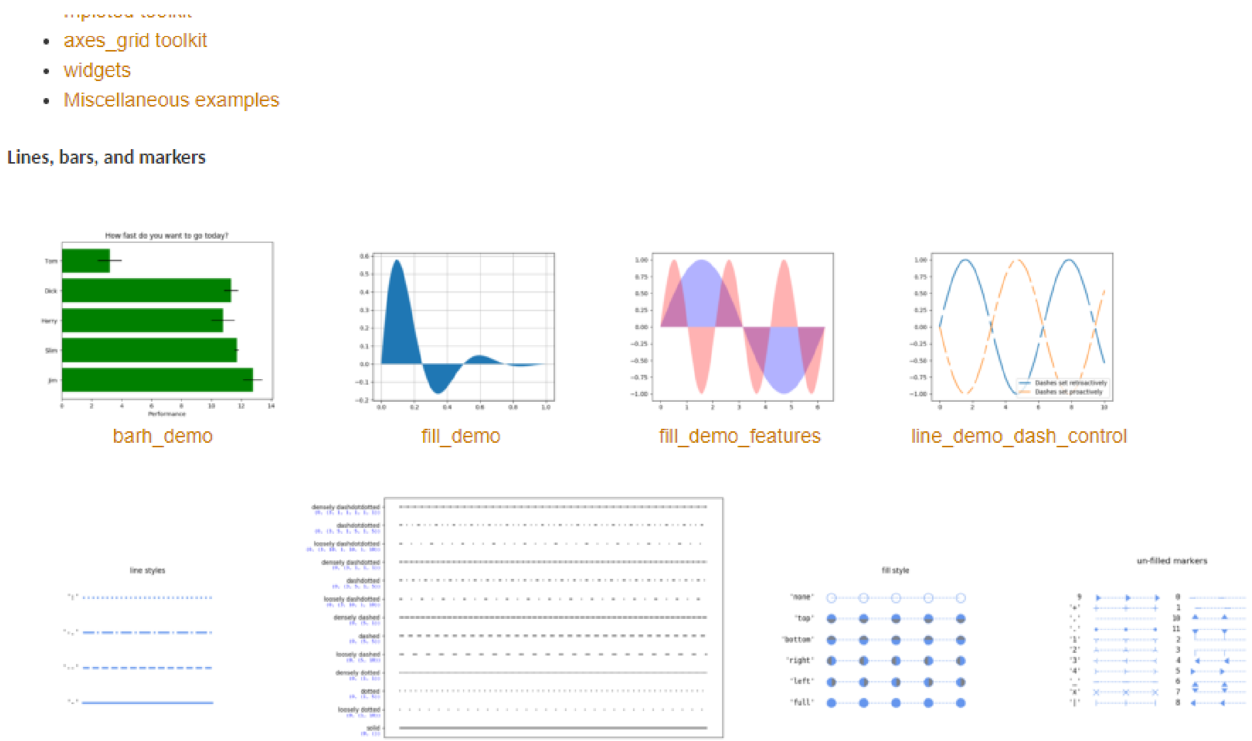
Useful resource



Useful

- https://matplotlib.org/gallery.html

Originally from Genson



- axes_grid toolkit
- widgets
- Miscellaneous examples

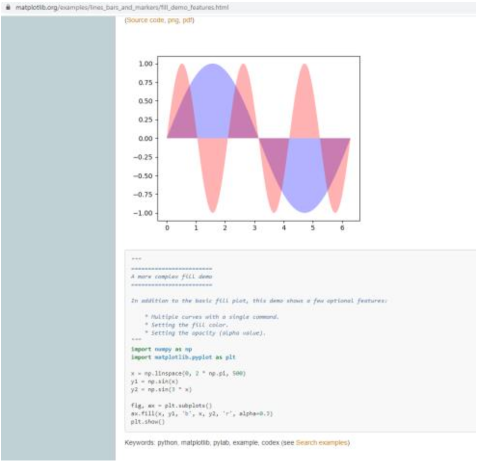
Lines, bars, and markers

barh_demo

fill_demo

fill_demo_features

line_demo_dash_control



matplotlib.org/examples/lines_bars_and_markers/fill_demo_features.html

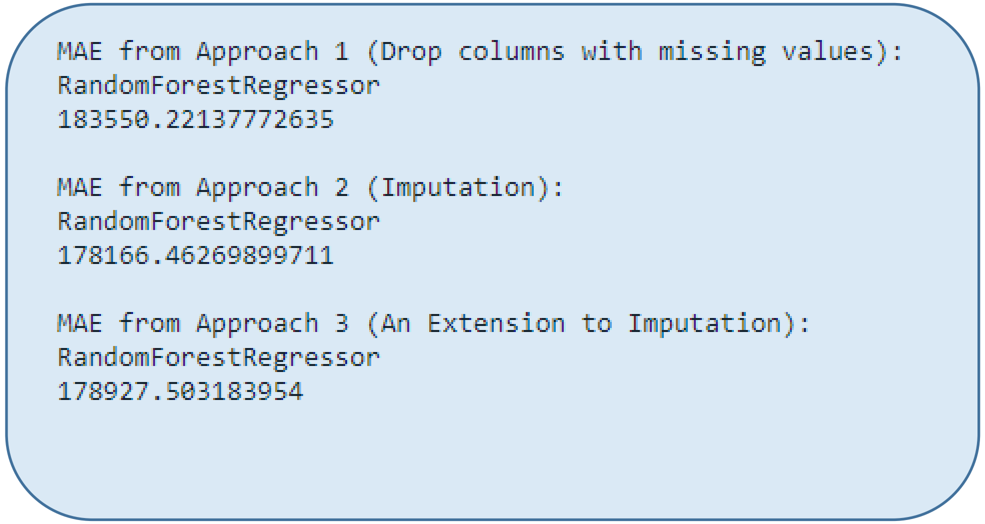
(Source code, png, pdf)

```
"""
========================
A more complex fill demo
========================

In addition to the basic fill plot, this demo shows a few optional features:

    * Multiple curves with a single command.
    * Setting the fill color.
    * Setting the opacity (alpha value).
"""
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi, 500)
y1 = np.sin(x)
y2 = np.sin(3 * x)

fig, ax = plt.subplots()
ax.fill(x, y1, 'b', x, y2, 'r', alpha=0.3)
plt.show()
```

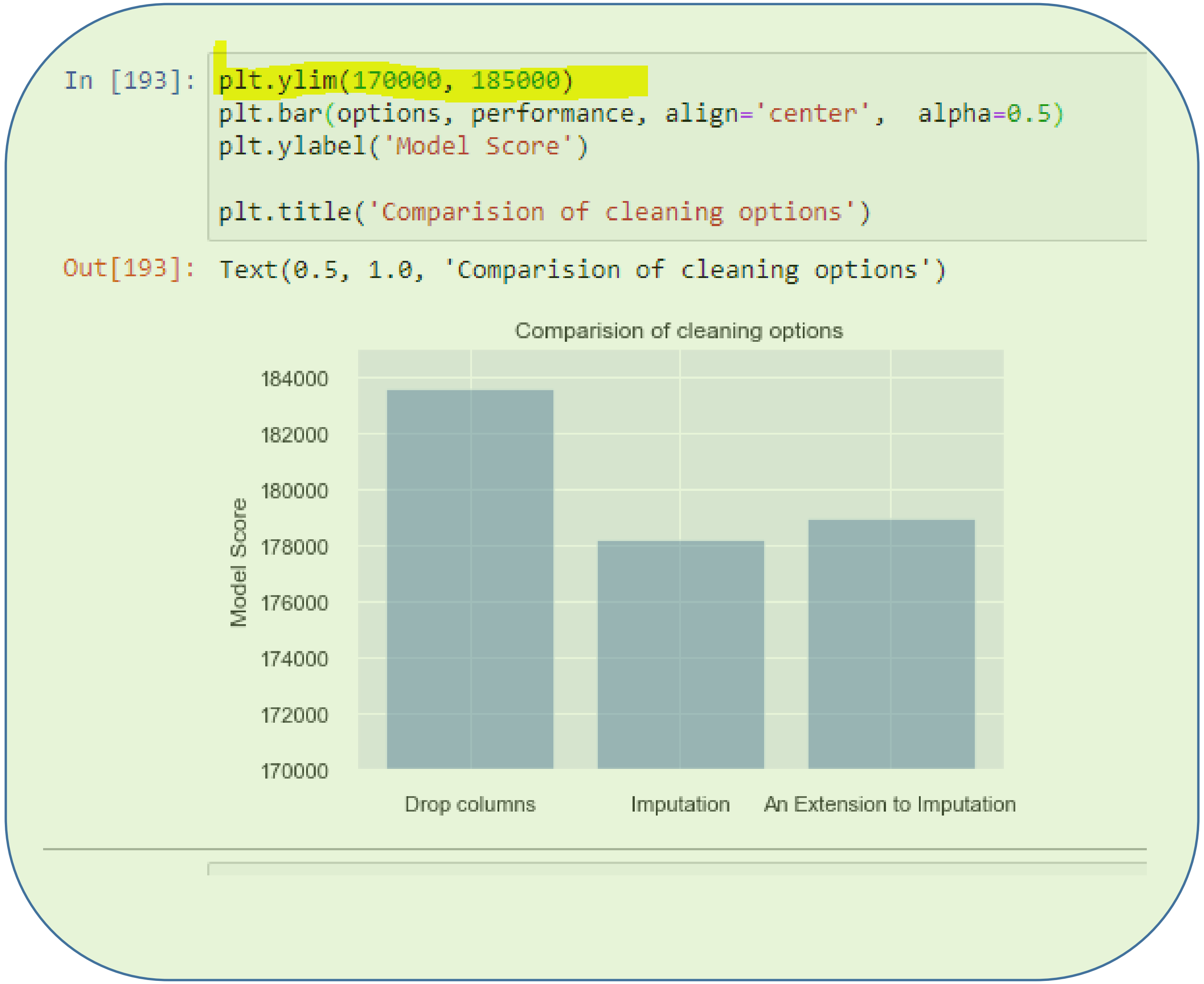
Keywords: python, matplotlib, pylab, example, codex (see Search examples)

Comparisons

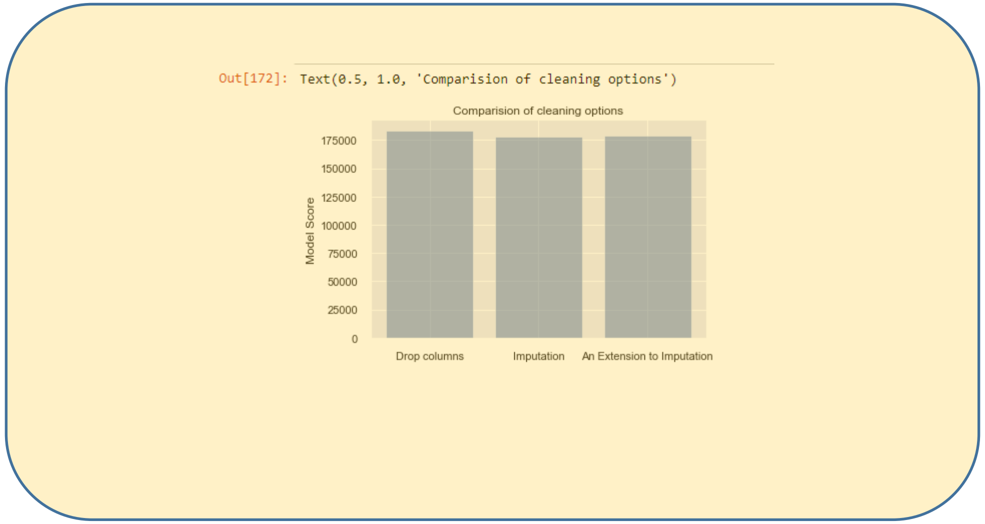
And the winner is simple imputation:



```
MAE from Approach 1 (Drop columns with missing values):
RandomForestRegressor
183550.22137772635

MAE from Approach 2 (Imputation):
RandomForestRegressor
178166.46269899711

MAE from Approach 3 (An Extension to Imputation):
RandomForestRegressor
178927.503183954
```



```
In [193]: plt.ylim(170000, 185000)
          plt.bar(options, performance, align='center',  alpha=0.5)
          plt.ylabel('Model Score')

          plt.title('Comparision of cleaning options')

Out[193]: Text(0.5, 1.0, 'Comparision of cleaning options')
```



```
Out[172]: Text(0.5, 1.0, 'Comparision of cleaning options')
```

Thanks

Alex Dance

Background

- Maths / statistics degree
- Background in big data, strategy, analytics
- Worked at Optus, Salmat, Reuters, Pathfinder Solutions

Copy of This Presentation and code

https://github.com/alexdance2468/

Plus other data science projects completed

Contact Details

www.linkedin.com/in/alex-dance/

Thanks

Sources:
https://www.kaggle.com/alexisbcook/missing-values - main code for comparisons
https://www.kaggle.com/dansbecker/melbourne-housing-snapshot/home data
https://matplotlib.org/3.1.0/tutorials/text/text_intro.html printing above and below graphs
https://matplotlib.org/gallery.html
https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/