Advanced Analytics Capstone

CSDA 1050

Sprint 3 Report

Housing Rent Prices for 2-bedroom apartments in the major Canadian Cities

Susiette Adams

303594

Introduction:

We will be evaluating the cost of living for renters in the major Canadian Cities. This information will provide a better understanding of what drives rent costs. The focus will be to find fact-based insights on meaningful patterns in the housing rental market. Owning a home is an ideal need for young adults in Canada and the social pressures along with increasing opportunities for profit, were driving the growth of the market, causing first time home buyers to struggle in finding a place to live at a reasonable price. Which forces most young adults to rent rather than owning their own home.

About the data set:

The datasets were gathered from third-party external source from the Canadian Mortgage and Housing Corporation website. The datasets include Average Income After Tax Renters, Average Rent 2 Bedrooms and Housing market Indicators. They can be found using these links below:

https://www.cmhc-schl.gc.ca/en/data-and-research/data-tables/real-average-after-tax-

https://www.cmhc-schl.gc.ca/en/data-and-research/data-tables/average-rent-2-bedroom-

https://www.cmhc-schl.gc.ca/en/data-and-research/data-tables/housing-market-indicators

Objective:

Our objective is to determine the most cost-effective price for renter's base on their household income. Even though factors, such as rise in unemployment rates increase in some of Canada's major cities, housing prices are still rising. Renting has started to consume over 50% of the average household's monthly income. We conclude that with increases in household debt, stagnant wages and expected rises in interest rates, a decline is inevitable. These factors are also forcing home buyers sell their homes and go back to renting.

Overview

My analysis has been carefully explained to its simplest form to accommodate stakeholders that may not understand the various analytical terms. Each step was carefully explained and visualized.

Loading some of the Libraries

```
In [1]: # Loading packages
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import os
        from pandas import read_table
        from scipy.stats import norm
        %matplotlib inline
```

Preparing the Average price per 2 bedrooms data

```
In [63]: # Importing the Renter Household data
         rent= pd.read_csv(r"C:\\Users\\susie\\Documents\\Data Science\\Capstone data\\Average price 2 bedrooms.csv"
                           ,encoding="latin")
         rent.head(5)
```

Out[63]:

	City	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
0	St. JohnÕs	566.0	554.0	559.0	565.0	570.0	567.0	513.0	517.0	552.0	...	614.0	630.0	677.0	725.0	771.0	798.0	864.0	888.0	923.0	958.0
1	Halifax	595.0	604.0	610.0	615.0	617.0	616.0	631.0	637.0	648.0	...	815.0	833.0	877.0	891.0	925.0	954.0	976.0	1005.0	1048.0	1063.0
2	Moncton	469.0	479.0	495.0	504.0	513.0	523.0	531.0	538.0	560.0	...	643.0	656.0	675.0	691.0	715.0	731.0	742.0	762.0	760.0	798.0
3	Saint John	429.0	436.0	439.0	437.0	441.0	449.0	452.0	457.0	460.0	...	570.0	618.0	644.0	645.0	670.0	691.0	691.0	714.0	718.0	720.0
4	Saguenay	420.0	419.0	416.0	417.0	423.0	425.0	428.0	428.0	438.0	...	490.0	518.0	518.0	535.0	557.0	549.0	571.0	595.0	598.0	587.0

5 rows × 26 columns

5 rows × 26 columns

Looking at the data above, it seems, we only have mostly numeric values only one

Column has objects.

```
In [72]: rent.columns

Out[72]: Index(['City', '1992', '1993', '1994', '1995', '1996', '1997', '1998', '1999',
               '2000', '2001', '2002', '2003', '2004', '2005', '2006', '2007', '2008',
               '2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016'],
              dtype='object')
```

Checking for missing data

we don't need to do any data formatting

Top 10 most expensive cities

Finding the to 10 hightest and lowest rent prices for 2 bedroom apartments.

In [46]:
```
# Using the n.largest I will search the data to find the highest prices for 2 bedroom apartments.
lrg = rent.nlargest(10, ['1992', '1993', '1994', '1995', '1996', '1997', '1998', '1999',
       '2000', '2001', '2002', '2003', '2004', '2005', '2006', '2007', '2008',
       '2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016'])
lrg
```

Out[46]:

	City	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2007	2008	2009	2010	2011	2012	2013	2014	2015	20
31	Vancouver	771.0	790.0	812.0	826.0	845.0	852.0	870.0	864.0	890.0	...	1084.0	1124.0	1169.0	1195.0	1237.0	1261.0	1281.0	1311.0	1368.0	145
13	Toronto	754.0	773.0	784.0	805.0	819.0	821.0	881.0	916.0	979.0	...	1061.0	1095.0	1096.0	1123.0	1149.0	1183.0	1213.0	1251.0	1288.0	1327
9	Ottawa	700.0	727.0	738.0	738.0	739.0	729.0	754.0	783.0	877.0	...	961.0	995.0	1028.0	1048.0	1086.0	1115.0	1132.0	1132.0	1174.0	1201
32	Victoria	684.0	703.0	713.0	715.0	717.0	724.0	722.0	728.0	731.0	...	907.0	965.0	1001.0	1024.0	1045.0	1059.0	1068.0	1095.0	1128.0	118
21	Barrie	653.0	670.0	687.0	712.0	713.0	737.0	774.0	788.0	830.0	...	934.0	954.0	961.0	968.0	1001.0	1037.0	1048.0	1118.0	1167.0	115
12	Oshawa	651.0	659.0	659.0	689.0	700.0	691.0	726.0	745.0	778.0	...	877.0	889.0	900.0	903.0	941.0	939.0	985.0	1010.0	1035.0	110
23	Thunder Bay	620.0	632.0	655.0	659.0	672.0	666.0	647.0	647.0	654.0	...	709.0	719.0	742.0	763.0	772.0	818.0	858.0	888.0	917.0	94
20	Windsor	620.0	631.0	643.0	667.0	682.0	680.0	680.0	696.0	736.0	...	773.0	772.0	747.0	752.0	753.0	778.0	788.0	798.0	824.0	85
18	Guelph	618.0	626.0	642.0	642.0	658.0	678.0	686.0	702.0	736.0	...	848.0	869.0	874.0	887.0	903.0	941.0	957.0	988.0	1027.0	107
30	Abbotsford-Mission	615.0	635.0	640.0	651.0	645.0	628.0	633.0	630.0	632.0	...	752.0	765.0	781.0	785.0	800.0	818.0	820.0	835.0	864.0	91

From the output above the most expensive cities are Vancouver, Toronto, Ottawa, Victoria, Barrie, Oshawa, Thunder Bay, Windsor, Guelph and Abbotsford Mission.

The top 10 least expensive cities

Here we are filtering the rent dataset to find the top 10 least expensive cities

In [47]:
```
# The .nsmallest was used t get the cities with the lowest prices for 2 bedroom apartments
sml = rent.nsmallest(10, ['1992', '1993', '1994', '1995', '1996', '1997', '1998', '1999',
       '2000', '2001', '2002', '2003', '2004', '2005', '2006', '2007', '2008',
       '2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016'])
sml
```

Out[47]:

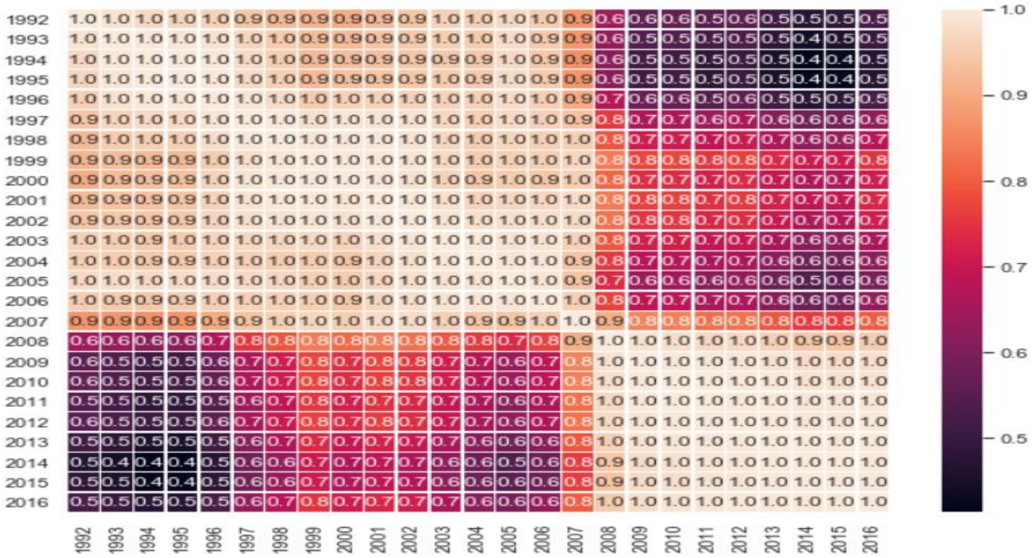
	City	1992	1993	1994	1995	1996	1997	1998	1999	2000	...	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
6	Trois-Rivi□res	395.0	400.0	402.0	406.0	405.0	406.0	411.0	403.0	413.0	...	487.0	505.0	520.0	533.0	547.0	550.0	555.0	568.0	581.0	587.0
5	Sherbrooke	408.0	418.0	420.0	422.0	426.0	426.0	433.0	434.0	437.0	...	529.0	543.0	553.0	566.0	577.0	578.0	591.0	604.0	608.0	622.0
4	Saguenay	420.0	419.0	416.0	417.0	423.0	425.0	428.0	428.0	438.0	...	490.0	518.0	518.0	535.0	557.0	549.0	571.0	595.0	598.0	587.0
3	Saint John	429.0	436.0	439.0	437.0	441.0	449.0	452.0	457.0	460.0	...	570.0	618.0	644.0	645.0	670.0	691.0	691.0	714.0	718.0	720.0
26	Saskatoon	444.0	449.0	452.0	460.0	479.0	500.0	516.0	529.0	541.0	...	693.0	841.0	905.0	934.0	966.0	1002.0	1041.0	1091.0	1087.0	1100.0
2	Moncton	469.0	479.0	495.0	504.0	513.0	523.0	531.0	538.0	560.0	...	643.0	656.0	675.0	691.0	715.0	731.0	742.0	762.0	760.0	798.0
25	Regina	484.0	487.0	485.0	487.0	494.0	512.0	525.0	547.0	549.0	...	661.0	756.0	832.0	881.0	932.0	979.0	1018.0	1079.0	1097.0	1109.0
7	Montr□al	488.0	484.0	484.0	494.0	491.0	491.0	499.0	506.0	509.0	...	647.0	659.0	669.0	700.0	719.0	711.0	730.0	739.0	760.0	791.0
8	Gatineau	513.0	519.0	528.0	536.0	537.0	530.0	529.0	534.0	544.0	...	662.0	677.0	690.0	711.0	731.0	743.0	744.0	750.0	751.0	762.0
17	Brantford	541.0	572.0	593.0	606.0	610.0	612.0	617.0	614.0	639.0	...	749.0	752.0	754.0	778.0	792.0	838.0	835.0	855.0	870.0	908.0

10 rows × 26 columns

The Top 10 least expensive cities are Tros-Rivers, Sherbrooke, Saguenay, Saint John, Saskatoon, Moncton, Regina, Montreal, Ga and Bradford. We will be focusing on the least expensive rent prices since our research question is to figure out the most livable cities.

Correlation plot for the sml data values

```
# Plotting the correlation data to get a better visual on the graph
f,ax = plt.subplots(figsize=(10,8))
sns.heatmap(sml.corr(), annot = True,linewidths=.4, fmt='.1f', ax=ax)
plt.show()
```



In the correlation heatmap it seems as though 2001, 2002,1999 and 2000 have the strongest correlation.

Renaming the variables that will be used for modelling

The dataset variables are years over a 25-year period with

In [6]:
```
#Renaming 2001 column name
sml.rename(columns={'2001': 'price1'}, inplace=True)
```

In [8]:
```
# Renameing 2007 column name
sml.rename(columns={'2002': 'price2'}, inplace=True)
```

In [9]:
```
# Renameing 2007 column name
sml.rename(columns={'1999': 'price99'}, inplace=True)
```

These variables are much easier to work with if they are renamed, because the column names were numbers, we might run into issues later if they aren't renamed before hand.

Checking for outliers

There are certain things, if not done in the EDA phase, can affect further statistical Machine Learning modelling. One of them is finding outliers. We will be using Box plots to help us detect outliers. Box plot is a method for graphically depicting groups of numerical data through their quartiles. They may also have lines extending vertically or horizontally from the indicating variability outside the upper and lower quartiles.

```
In [96]: sml.plot(kind='box', figsize=(15,8))
```

Out[96]: <matplotlib.axes._subplots.AxesSubplot at 0x2d6d3eba160>

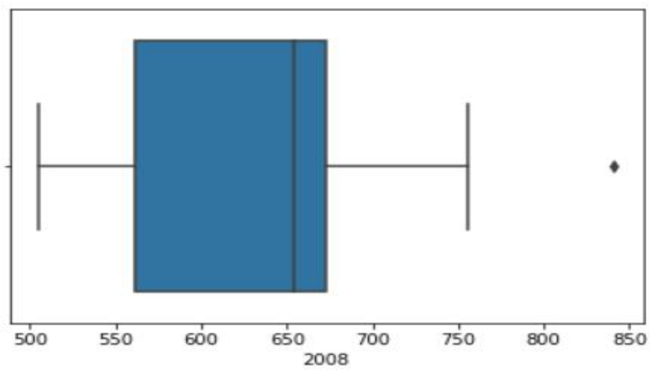
Looking at the graph above there seem to several outliers in the sml dataset from 2008 to 2016 we will take a closer look below.

Identifying the outliers

Before we try to understand whether to ignore the outliers or not, we need to know ways to identify them. Mostly we will try to see visualization methods rather mathematical. Outliers may be plotted as individual points.

```
In [232]: # identify the outlier in 2018
          sns.boxplot(sml['2008'])
```

Out[232]: <matplotlib.axes._subplots.AxesSubplot at 0x1f73cb43198>



Above plot shows 2008 has one point between 800 to 850, the outlier is not included in the box of other observation nowhere near the quartiles. This we will remove, and the same process will be repeated to clean any outlier in the dataset.

Removing Outlier

[234]:

```python
# Identifyng and removig the outliers from the 2008 values
from numpy import percentile

# calculate interquartile range
q25, q75 = percentile(data, 25), percentile(data, 75)
iqr = q75 - q25
print('Percentiles: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))
# remove outliers
removed = [x for x in data if x >= lower and x <= upper]
print('Non-outlier observations: %d' % len(removed))
```

```
Percentiles: 25th=561.750, 75th=672.500, IQR=110.750
Identified outliers: 1
Non-outlier observations: 9
```

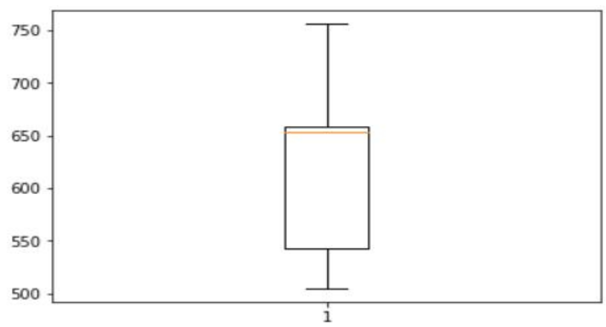
In the 2008 feature there are 9 non-outlier and one outlier that was identified.

Clean data

Here is the result of the cleaned data after the outlier was removed if there is an outlier it will plotted as point in boxplot, but other population will be grouped together and display as box. Let's try and see it ourselves.

In [236]:

```python
# plot showing the 2008 clean data values
plt.boxplot(removed)
plt.show()
```

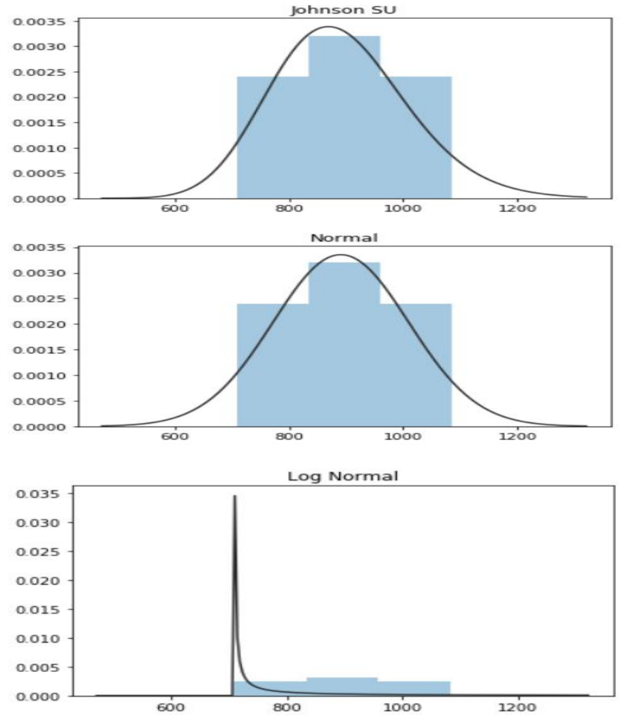


Even though there was an outlier it seems as though it didn't affected the mean price it remained the same before and after the outlier was removed at around 650.

Johnson SU distribution plot

The mu and sigma are the mean and the standard deviation of the distribution. The Johnson plot is a transformation of the normal distribution the Johnson SU was developed to in order to apply the established methods and theory of the normal distribution to non-normal datasets.

<matplotlib.axes._subplots.AxesSubplot at 0x1f73f4f8390>

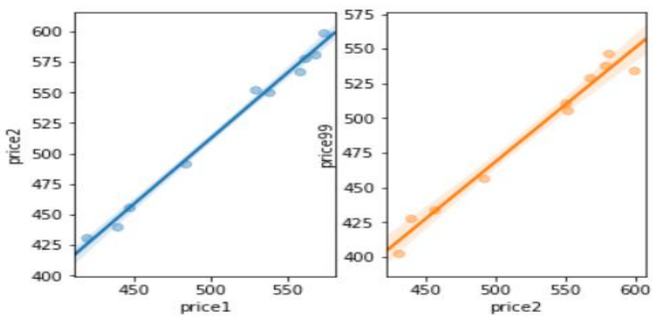


Regression Plot

The line of best fit is calculated by minimizing the ordinary least squares error function, that Seaborn module does automatically using the regplot function. The shaded area around the line represents 95% confidence intervals. When running a regression, statwing automatically calculates and plots residuals to help understand and improve the regression model.

In [18]:
```
# Regression plot
fig, ax = plt.subplots(1,2)
sns.regplot('price1','price2',sml, ax=ax[0], scatter_kws={'alpha':0.4})
sns.regplot('price2','price99',sml,ax=ax[1], scatter_kws={'alpha':0.4})
```

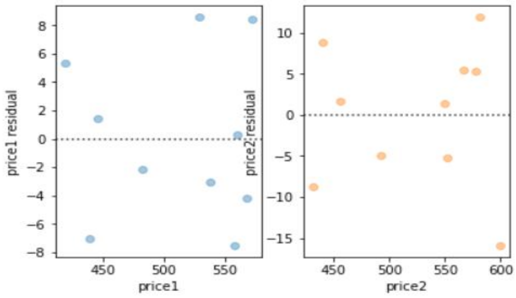
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x218db4cc9b0>



As you can see from the above plots our data is very clean the points on the model are almost perfect.

Residual Plot

```
In [20]: # visualizing the residuals by creating residual plots .
fig, ax = plt.subplots(1,2)
ax[0]= sns.residplot('price1','price2', sml,ax=ax[0], scatter_kws={'alpha':0.4})
ax[0].set_ylabel('price1 residual')
ax[1]=sns.residplot('price2','price99',sml,ax=ax[1], scatter_kws={'alpha':0.4})
ax[1].set_ylabel('price2 residual')

Out[20]: Text(0, 0.5, 'price2 residual')
```



The points in the residual plot represent the difference between the sample (y) and the predicted value (y'). Residuals that are greater than zero are points that are underestimated by the model and residuals less than zero are points that are overestimated by the model.

Model Creation

Creating a fitted linear model

```
In [259]: # create a fitted model
import statsmodels.formula.api as smf
model = smf.ols(formula='price1 ~ price2', data=sml).fit()

# print the coefficients
lm1.params

Out[259]: Intercept    12.191914
price2        0.952636
dtype: float64

In [ ]: # The price for rent seem to have increased by $95.26 from 2001 to 2002
```

The price for rent seem to have increased by $95.26 from 2001 to 2002

P Values

```
In [260]: # print the p-values for the model coefficients
#Represents the probability that the coefficient is actually zero

lm1.pvalues

Out[260]: Intercept    4.506910e-01
price2       6.838871e-10
dtype: float64
```

P values represents the probability that the coefficient is actually zero.

```
In [232]: # print the R-squared value for the model
          lm1.rsquared

Out[232]: 0.9907773400504103
```

```
In [238]: # create a fitted model with all three features
          lm2 = smf.ols(formula='price1 ~ price2 + price98 + price99', data=sml).fit()

          # print the coefficients
          lm2.params

Out[238]: Intercept   -18.816983
          price2        0.440564
          price98       0.169263
          price99       0.444685
          dtype: float64
```

Price for rent seem to have decreased by $18.82 when evaluated againts price2, 98, and 99.

Model creation

Adding 3 more models to do our evaluation which includes price1(2001) and price2(2002). Model2 includes price98(1998) and price99(1999) as well as model 3 which includes price1, price2, price98, price99, and price00(2000) .

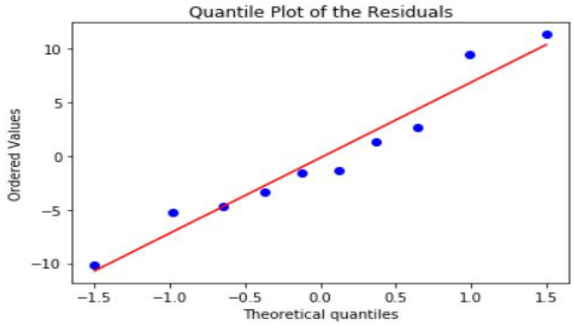
```
In [113]: # Next we'll want to fit a linear regression model. We need to choose variables that we think we'll be good predictors for the d
          # This can be done by checking the correlation(s) between variables, on what variables are good predictors of y.

          import statsmodels.api as sm
          X = sml["price1"]
          y = sml["price2"]

          # Note the difference in argument order
          model = sm.OLS(y, X).fit()
          predictions = model.predict(X) # make the predictions by the model

In [116]: # # Next we'll want to fit a linear regression model by adding a constant.
          X = sml["price98"] ## X usually means our input variables (or independent variables)
          y = sml["price99"] ## Y usually means our output/dependent variable
          X = sm.add_constant(X) ## let's add an intercept (beta_0) to our model

          # Note the difference in argument order
          model2 = sm.OLS(y, X).fit() ## sm.OLS(output, input)
          predictions = model2.predict(X)

In [125]: # set input and output variables to use in regression model
          x = sml[['price2', 'price98', 'price99','price00']]
          y = sml['price1']

          # add intercept to input variable
          x = sm.add_constant(x)

          # fit regression model, using statsmodels GLM uses a different method but gives the same results
          #model = sm.GLM(y, x, family=sm.families.Gaussian()).fit()
          model3 = sm.OLS(y, x).fit()
```

Quantile residual plots for all 3 models

Quantile regression models is the relation between a set of predictor variables and specific percentiles or quantiles of the response variable. It specifies changes in the quantiles of the response. Quantile regression makes no assumptions about the distribution of the residuals.

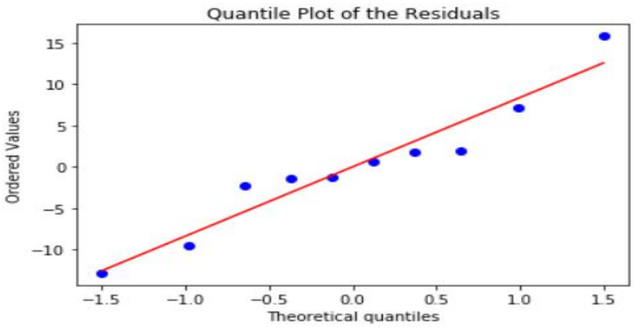
Plotting Model1

```
In [136]: import pylab
          # quantile plot of residuals
          stats.probplot(model.resid, plot=pylab)
          _ = plt.title('Quantile Plot of the Residuals');
```



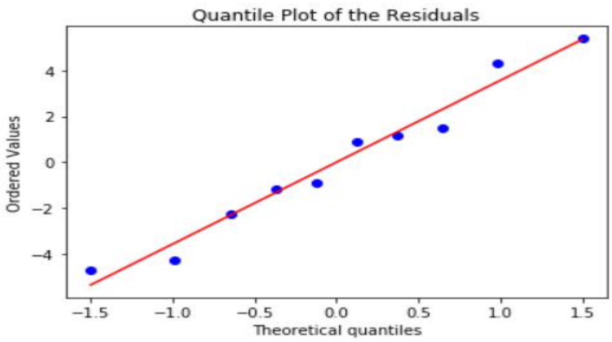
Plotting Model2

```
In [137]: # quantile plot of residuals
          stats.probplot(model2.resid, plot=pylab)
          _ = plt.title('Quantile Plot of the Residuals');
```



Plotting Model3

```
In [138]: import pylab
          # quantile plot of residuals
          stats.probplot(model3.resid, plot=pylab)
          _ = plt.title('Quantile Plot of the Residuals');
```

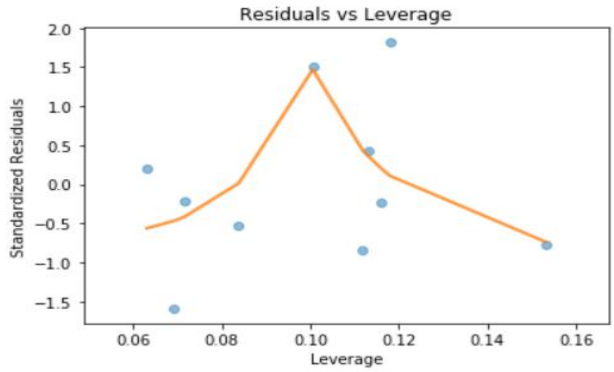


From the quantile plots above all 3 plots are close to perfect but model3 seem to be the most fitted plot.

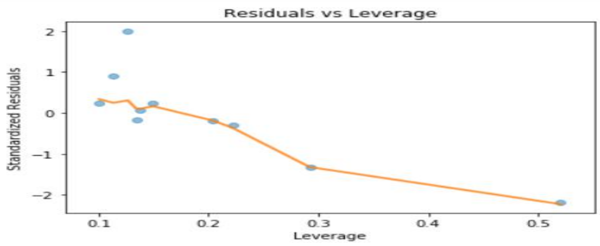
Normalizing the residuals for the models

The standardized residual is a measure of the strength of the difference between observed and expected values. It’s a measure of how significant your cells are to the chi-square value. When you compare the cells, the standardized residual makes it easy to see which cells are contributing the most to the value, and which are contributing the least.

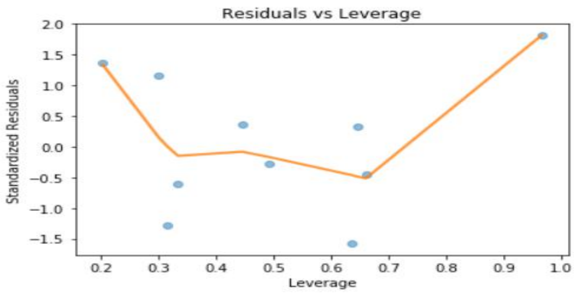
```
In [142]: # normalized residuals
model_norm_residuals = model.get_influence().resid_studentized_internal
# leverage, from statsmodels internals
model_leverage = model.get_influence().hat_matrix_diag
plt.scatter(model_leverage, model_norm_residuals, alpha=0.5)
sns.regplot(model_leverage, model_norm_residuals,
            scatter=False,
            ci=False,
            lowess=True,
            line_kws={'color': 'C1', 'alpha': 0.8})
_ = plt.title('Residuals vs Leverage')
_ = plt.xlabel('Leverage')
_ = plt.ylabel('Standardized Residuals');
```




```
In [143]: # normalized residuals
model_norm_residuals = model2.get_influence().resid_studentized_internal

# leverage, from statsmodels internals
model_leverage = model2.get_influence().hat_matrix_diag

plt.scatter(model_leverage, model_norm_residuals, alpha=0.5)
sns.regplot(model_leverage, model_norm_residuals,
            scatter=False,
            ci=False,
            lowess=True,
            line_kws={'color': 'C1', 'alpha': 0.8})

_ = plt.title('Residuals vs Leverage')
_ = plt.xlabel('Leverage')
_ = plt.ylabel('Standardized Residuals');
```



```
In [139]: # normalized residuals
model_norm_residuals = model3.get_influence().resid_studentized_internal
# leverage, from statsmodels internals
model_leverage = model3.get_influence().hat_matrix_diag
plt.scatter(model_leverage, model_norm_residuals, alpha=0.5)
sns.regplot(model_leverage, model_norm_residuals,
            scatter=False,
            ci=False,
            lowess=True,
            line_kws={'color': 'C1', 'alpha': 0.8})
_ = plt.title('Residuals vs Leverage')
_ = plt.xlabel('Leverage')
_ = plt.ylabel('Standardized Residuals');
```

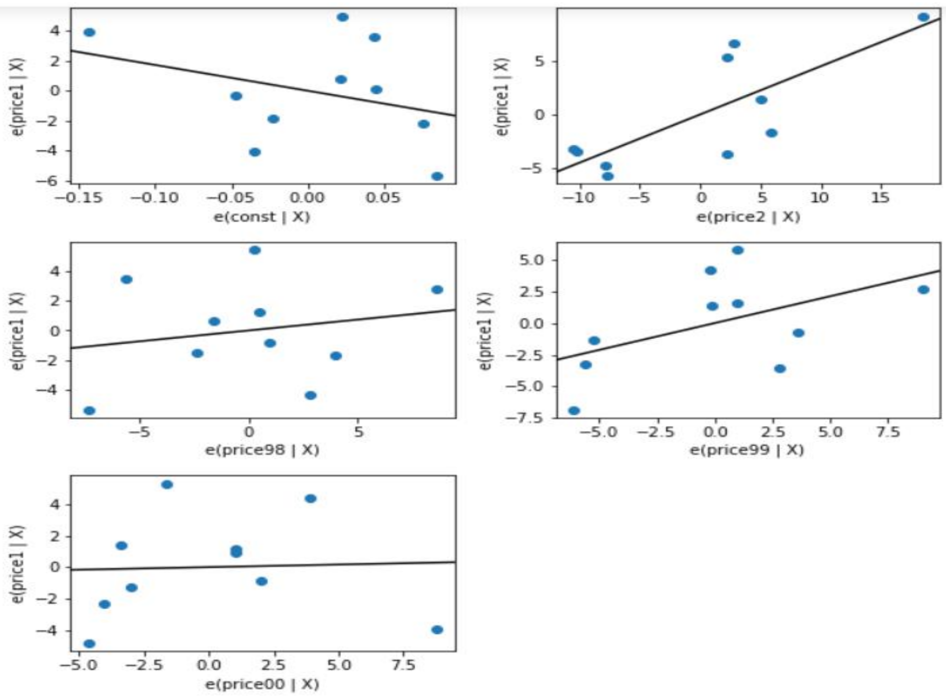


If the residual is less than -2, the cell's observed frequency is less than the expected frequency. Greater than 2 and the observed frequency is greater than the expected frequency.

Model3 Partial Regression Variable plots

Partial regression plots attempt to show the effect of adding an additional variable to the model given that one or more indpendent variables are already in the model. Partial regression plots are formed by:

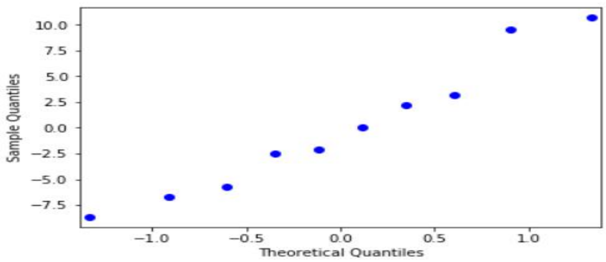
```
In [126]: # model3 variable plots
fig = plt.figure(figsize=(8,8))
fig = sm.graphics.plot_partregress_grid(model3, fig=fig)
```



QQ Plots

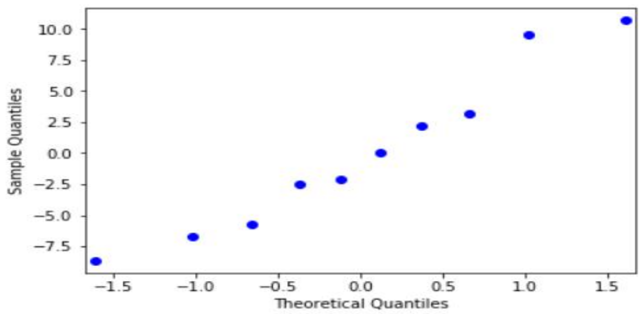
The Q-Q plot, or quantile-quantile plot, is a graphical tool to help us assess if a set of data plausibly came from some theoretical distribution such as a Normal or exponential. For example, if we run a statistical analysis that assumes our dependent variable is Normally distributed, we can use a Normal Q-Q plot to check that assumption. It's just a visual check, not an air-tight proof, so it is somewhat subjective. But it allows us to see at-a-glance if our assumption is plausible, and if not, how the assumption is violated and what data points contribute to the violation.

```
In [75]: # QQ Plots
         import statsmodels.api as sm
         from matplotlib import pyplot as plt
         res = model.resid # residuals
         fig = sm.qqplot(res)
         plt.show()
```



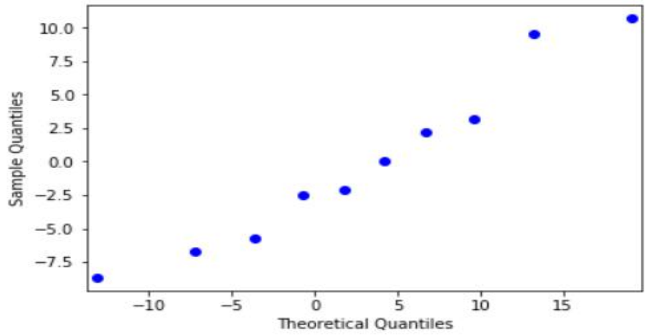
Qq plot of the residuals against quantiles of t-distribution with 4 degrees of freedom:

```
In [76]: import scipy.stats as stats
         fig = sm.qqplot(res, stats.t, distargs=(4,))
         plt.show()
```



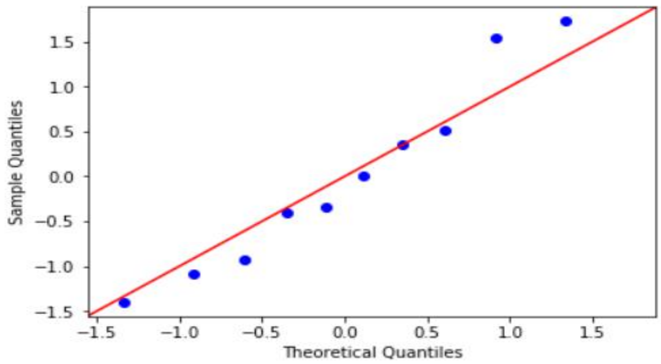
qqplot against same as above, but with mean 3 and std 10:

```
In [77]: fig = sm.qqplot(res, stats.t, distargs=(4,), loc=3, scale=10)
         plt.show()
```



Automatically determine parameters for t distribution including the loc and scale:

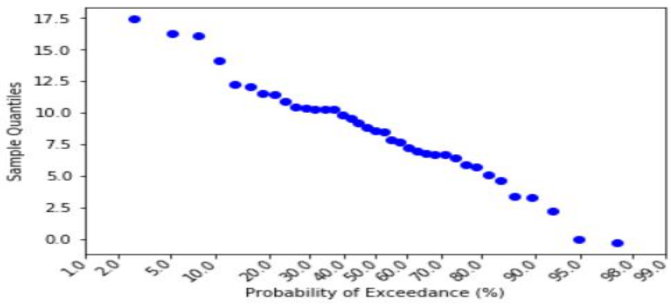
```
In [78]: fig = sm.qqplot(res, stats.t, fit=True, line='45')
         plt.show()
```

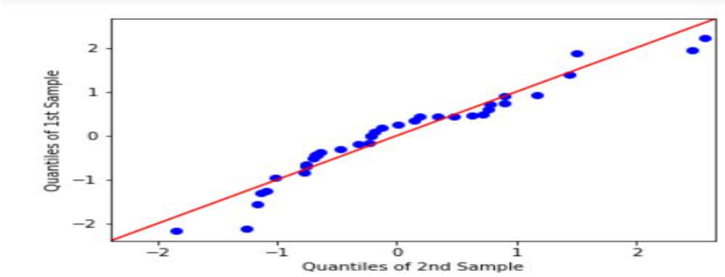


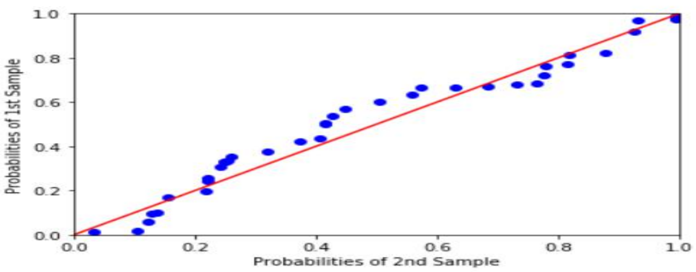
Probability Class

Randomly generate data from a standard Normal distribution and then find the quantiles.

```
In [79]: # ProbPlot class
         x = np.random.normal(loc=8.25, scale=3.5, size=37)
         y = np.random.normal(loc=8.00, scale=3.25, size=37)
         pp_x = sm.ProbPlot(x, fit=True)
         pp_y = sm.ProbPlot(y, fit=True)

         # probability of exceedance
         fig2 = pp_x.probplot(exceed=True)

         # compare x quantiles to y quantiles
         fig3 = pp_x.qqplot(other=pp_y, line='45')

         # same as above with probabilities/percentiles
         fig4 = pp_x.ppplot(other=pp_y, line='45')
```





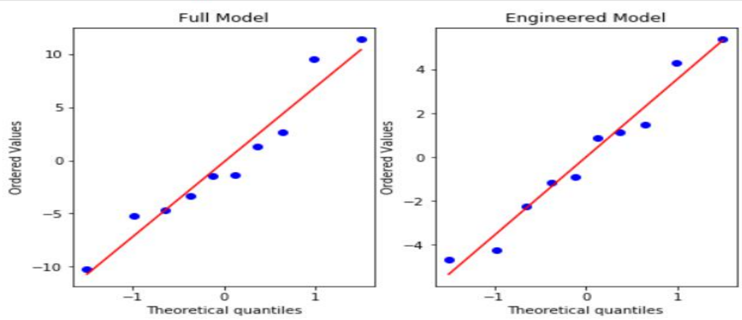


The results above is from the qq plot line 45

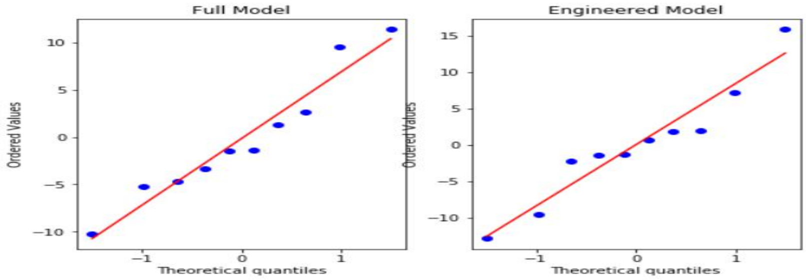
Probability Plots using all 3 models

Generates a probability plot of sample data against the quantiles of a specified theoretical distribution the normal distribution by default. probplot optionally calculates a best-fit line for the data and plots the results.

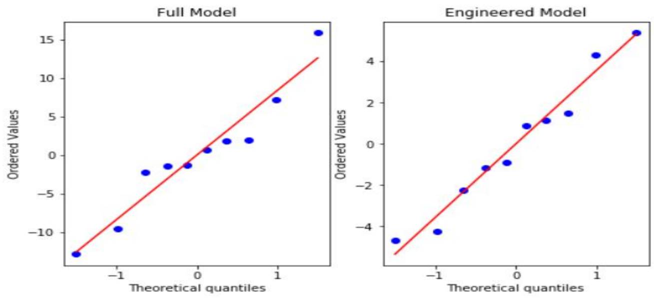
```
In [150]: plt.figure(figsize=(8, 5))

          plt.subplot(1, 2, 1)
          stats.probplot(model.resid, plot=pylab)
          _ = plt.title('Full Model');

          plt.subplot(1, 2, 2)
          stats.probplot(model3.resid, plot=pylab)
          _ = plt.title('Engineered Model');
```



```
In [151]: plt.figure(figsize=(8, 5))

          plt.subplot(1, 2, 1)
          stats.probplot(model.resid, plot=pylab)
          _ = plt.title('Full Model');

          plt.subplot(1, 2, 2)
          stats.probplot(model2.resid, plot=pylab)
          _ = plt.title('Engineered Model');
```




```
In [152]: plt.figure(figsize=(8, 5))

          plt.subplot(1, 2, 1)
          stats.probplot(model2.resid, plot=pylab)
          _ = plt.title('Full Model');

          plt.subplot(1, 2, 2)
          stats.probplot(model3.resid, plot=pylab)
          _ = plt.title('Engineered Model');
```



Handling Categorical Features

My goal to handel the categorical data was to be able to evaluate the cities, but when indexing there are only 2 categories which is 0 and 1. The error was spotted after creating the code because the dataset we are evaluating has 10 different cities it's challenging to evaluate the cities using indexing.

```
In [278]: # set a seed for reproducibility
          np.random.seed(12345)

          # create a Series of booleans in which roughly half are True
          nums = np.random.rand(len(sml))
          mask_large = nums > 0.5

          # initially set Size to small, then change roughly half to be large
          sml['City'] = 'Brantford'

          # set a seed for reproducibility

          # Series.loc is a purely label-location based indexer for selection by label
          sml.loc[mask_large, 'City'] = 'Sherbrooke'
          sml.loc[mask_large, 'City'] = 'Trois-Rivi•res'
          sml.loc[mask_large, 'City'] = 'Saint John'
          sml.loc[mask_large, 'City'] = 'Saguenay'
          sml.loc[mask_large, 'City'] = 'Saskatoon'
          sml.loc[mask_large, 'City'] = 'Moncton'
          sml.loc[mask_large, 'City'] = 'Regina'
          sml.loc[mask_large, 'City'] = 'Montral'
          sml.loc[mask_large, 'City'] = 'Gatineau'

          sml.head()
```

Out[278]:

	City	1992	1993	1994	1995	1996	1997	price98	price99	price00	...	2007	2008	2009	2010	2011	2012	2013	2014	2015	price16
6	Gatineau	395.0	400.0	402.0	406.0	405.0	406.0	411.0	403.0	413.0	...	487.0	505.0	520.0	533.0	547.0	550.0	555.0	568.0	581.0	587.0
5	Brantford	408.0	418.0	420.0	422.0	426.0	426.0	433.0	434.0	437.0	...	529.0	543.0	553.0	566.0	577.0	578.0	591.0	604.0	608.0	622.0
4	Brantford	420.0	419.0	416.0	417.0	423.0	425.0	428.0	428.0	438.0	...	490.0	518.0	518.0	535.0	557.0	549.0	571.0	595.0	598.0	587.0

```
In [279]: # create three dummy variables using get_dummies
          pd.get_dummies(sml.City, prefix='City').head()
```

Out[279]:

	City_Brantford	City_Gatineau
6	0	1
5	1	0
4	1	0
3	1	0
26	0	1

Model Summary

OLS stands for Ordinary Least Squares and the method “Least Squares” means that we’re trying to fit a regression line that would minimize the square of distance from the regression line.

Model1

```
In [281]: print(model.summary())

                            OLS Regression Results
==============================================================================
Dep. Variable:                 price1   R-squared:                       0.993
Model:                            OLS   Adj. R-squared:                  0.992
Method:                 Least Squares   F-statistic:                     1124.
Date:                Fri, 30 Aug 2019   Prob (F-statistic):           6.84e-10
Time:                        00:19:32   Log-Likelihood:                -31.966
No. Observations:                  10   AIC:                             67.93
Df Residuals:                       8   BIC:                             68.54
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     12.1919     15.376      0.793      0.451     -23.265      47.648
price2         0.9526      0.028     33.527      0.000       0.887       1.018
==============================================================================
Omnibus:                        0.836   Durbin-Watson:                   1.765
Prob(Omnibus):                  0.658   Jarque-Bera (JB):                0.677
Skew:                          -0.341   Prob(JB):                        0.713
Kurtosis:                       1.923   Cond. No.                     3.98e+03
==============================================================================
```

OLS stands for Ordinary Least Squares and the method “Least Squares” means that we’re trying to fit a regression line that would minimize the square of distance from the regression line.

Model2

```
In [194]: print(model2.summary())

                            OLS Regression Results
==============================================================================
Dep. Variable:                price99   R-squared:                       0.985
Model:                            OLS   Adj. R-squared:                  0.984
Method:                 Least Squares   F-statistic:                     540.3
Date:                Thu, 29 Aug 2019   Prob (F-statistic):           1.25e-08
Time:                        13:43:37   Log-Likelihood:                -34.456
No. Observations:                  10   AIC:                             72.91
Df Residuals:                       8   BIC:                             73.52
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -15.1279     22.280     -0.679      0.516     -66.506      36.251
price98        1.0405      0.045     23.245      0.000       0.937       1.144
==============================================================================
Omnibus:                        1.181   Durbin-Watson:                   1.262
Prob(Omnibus):                  0.554   Jarque-Bera (JB):                0.160
Skew:                           0.309   Prob(JB):                        0.923
Kurtosis:                       3.064   Cond. No.                     4.13e+03
==============================================================================
```

Model3

```
In [195]: print(model3.summary())

                            OLS Regression Results
==============================================================================
Dep. Variable:                 price1   R-squared:                       0.998
Model:                            OLS   Adj. R-squared:                  0.996
Method:                 Least Squares   F-statistic:                     623.7
Date:                Thu, 29 Aug 2019   Prob (F-statistic):           6.25e-07
Time:                        13:48:34   Log-Likelihood:                -25.656
No. Observations:                  10   AIC:                             61.31
Df Residuals:                       5   BIC:                             62.82
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -17.2774     21.621     -0.799      0.460     -72.857      38.302
price2         0.4473      0.162      2.759      0.040       0.030       0.864
price98        0.1458      0.320      0.455      0.668      -0.677       0.969
price99        0.4239      0.315      1.344      0.237      -0.387       1.235
price00        0.0330      0.353      0.094      0.929      -0.873       0.940
==============================================================================
Omnibus:                        0.248   Durbin-Watson:                   1.603
Prob(Omnibus):                  0.883   Jarque-Bera (JB):                0.403
Skew:                           0.157   Prob(JB):                        0.818
Kurtosis:                       2.068   Cond. No.                     1.58e+04
==============================================================================
```

The statistics in the last table are testing the normality of our data. If the Prob(Omnibus) is very small, and I took this to mean <.05 as this is standard statistical practice, then our data is probably not normal. This is a more precise way than graphing our data to determine if our data is normal.

Statsmodels also helps us determine which of our variables are statistically significant through the p-values. If our p-value is <.05, then that variable is statistically significant. This is a useful tool to tune your model. In the case of the iris data set we can put in all of our variables to determine which would be the best predictor.

Model1 Accuracy

```
In [38]: from sklearn.dummy import DummyRegressor
         from sklearn.metrics import mean_absolute_error
         from sklearn.metrics import r2_score
         from sklearn.model_selection import train_test_split
         y = price1
         X = price2
         X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=100)
         dummy_median = DummyRegressor(strategy='mean')
         dummy_regressor = dummy_median.fit(X_train,y_train)
         dummy_predicts = dummy_regressor.predict(X_test)
         print("Model Accuracy:", dummy_regressor.score(X_test,y_test)*100)

         Model Accuracy: -565.585963182117
```

Model Selection

Loading in the income dataset

In [20]:
```python
# Loading the income after tax dataset
income= pd.read_csv(r"C:\\Users\\susie\\Documents\\Data Science\\Capstone data\\Average after tax income.csv"
                    ,encoding="latin1")
```

Taking a look at the head of the dataset

In [21]:
```python
income.head(5)
```

Out[21]:

	Cities	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
0	Canada	42500	43800	45500	45400	45000	44800	45800	46700	48000	48200	49100	50300
1	Newfoundland and Labrador	37300	37400	38800	37400	39000	43100	42000	44400	45600	44000	45700	40200
2	Prince Edward Island	36500	38900	42400	42900	41200	38200	38400	40000	37200	42300	44900	43100
3	Nova Scotia	37400	40300	38200	38800	39600	42400	40800	42700	41200	41400	43000	43000
4	New Brunswick	36300	37100	38600	38100	41500	37000	36000	38900	38100	37900	38400	38200

In [22]:
```python
income.columns
```

Finding the cities with the lowest and highest income bracket

Finding the hightest and lowest income ranges for top 10 cities.

For the income dataset we will be forcusing on the the cities with the highest income bracket

In [ ]:
```python
# Filtering the data to find the cities with the lowest income
```

In [24]:
```python
# Using the n.smallest I will search the data to find cities with the highest income.
low = income.nsmallest(10, ['2006', '2007', '2008', '2009', '2010', '2011', '2012',
        '2013', '2014', '2015', '2016', '2017'])
low
```

Out[24]:

	Cities	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
17	Trois-Rivières	28700	36800	35800	36500	34800	34300	31300	34100	33600	31800	38500	36200
14	Saguenay	31600	33100	33800	34100	35400	38400	37000	38200	33000	35400	37800	35100
16	Sherbrooke	32900	34800	36100	35400	36600	34100	32600	35300	36200	41500	37700	35800
11	St John's	34400	33000	38000	38300	40000	42300	42900	48400	49500	48300	45700	43400
33	Regina	36000	44000	43700	44100	53400	51600	53300	53900	57600	53700	54600	53400
4	New Brunswick	36300	37100	38600	38100	41500	37000	36000	38900	38100	37900	38400	38200
2	Prince Edward Island	36500	38900	42400	42900	41200	38200	38400	40000	37200	42300	44900	43100
7	Manitoba	36800	40000	38600	38000	39200	43600	41900	43700	45200	44100	45500	44400
31	Thunder Bay	37100	38900	35000	39400	42800	36800	40500	39000	38800	45500	42800	44800
32	Winnipeg	37200	40400	38100	37600	39300	42900	42100	42800	43700	42400	44600	43700

In [ ]:
```python
# Filtering the data to find the cities with the lowest income
```

Finding cities witht the highest income brackets

In [23]:
```python
# Using the n.largest I will search the data to find cities with the highest income.
high = income.nlargest(10, ['2006', '2007', '2008', '2009', '2010', '2011', '2012',
        '2013', '2014', '2015', '2016', '2017'])
high
```

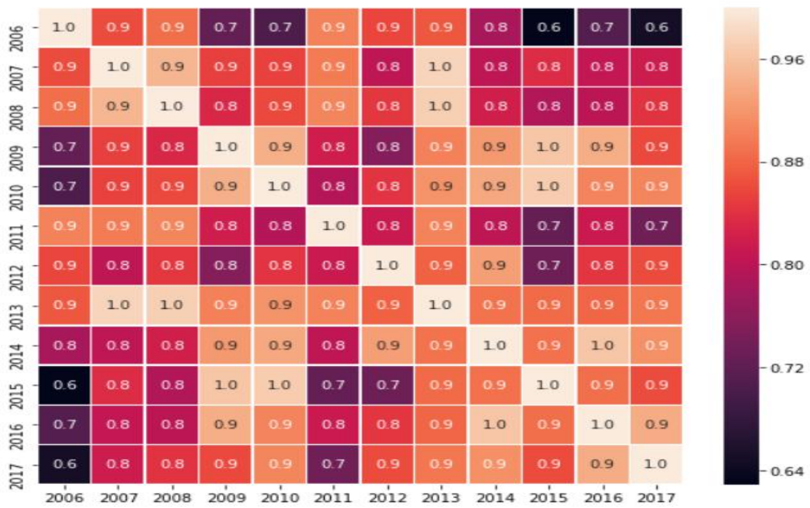
Out[23]:

	Cities	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017
35	Calgary	58000	65800	71000	61800	61700	63200	67300	70000	66500	63900	63400	66700
9	Alberta	54400	59800	62200	63300	62200	60000	64700	66000	67500	68600	64200	65400
36	Edmonton	50100	58400	56900	66800	65800	58600	57900	61800	67800	72200	66800	66200
37	Abbotsford	49200	44000	42100	47700	42500	47000	53100	44900	55300	46800	55400	53900
39	Victoria	47400	47300	47200	45300	38900	52900	45300	47000	45400	42700	54500	54900
38	Vancouver	46600	50600	47800	45800	49000	47300	54000	51100	51300	53000	54200	59300
10	British Columbia	46600	48400	48500	46000	46700	47900	50900	49600	49700	49500	53000	57700
21	Ottawa - Gatineau (Québec)	46400	44800	42100	42200	38600	46600	37100	41800	37600	44400	44600	44200
27	Kitchener	45800	50800	46100	50800	42900	45100	40200	49800	46300	54000	54000	55800
24	Toronto	45600	45700	50200	51900	51100	46200	47900	49200	53300	55500	55900	59800

For this dataset will be focusing on the cities with the highest income brackets. Usually cities with the highest income bracket would also have higher housing costs, but there are also other factors that affects the cost of living.

Plotting a heat map to find the most correlated years

```
f,ax = plt.subplots(figsize=(9,8))
sns.heatmap(high.corr(), annot = True,linewidths=.4, fmt='.1f', ax=ax)
plt.show()
```



Renaming the columns

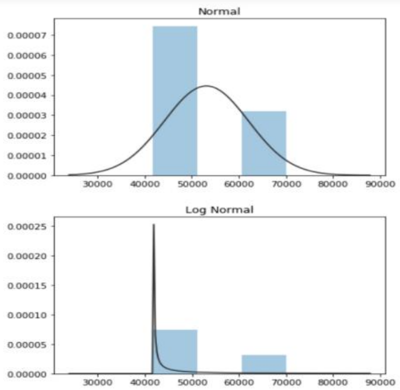
```
In [ ]: # Using Seaborn heatmap to plotting the correlation data to get a better visual on the graph
        f,ax = plt.subplots(figsize=(9,8))
        sns.heatmap(high.corr(), annot = True,linewidths=.4, fmt='.1f', ax=ax)
        plt.show()

In [31]: #Renaming 2013 column name
         high.rename(columns={'2013': 'income13'}, inplace=True)

In [32]: #Renaming 2015 column name
         high.rename(columns={'2015': 'incom15'}, inplace=True)

In [33]: #Renaming 2009 column name
         high.rename(columns={'2009': 'income9'}, inplace=True)

In [34]: high.columns

Out[34]: Index(['Cities', '2006', '2007', '2008', 'income9', '2010', '2011', '2012',
               'income13', '2014', 'incom15', '2016', '2017'],
              dtype='object')
```

In [41]:
```python
## let's do some analysis on this variable first.
from scipy import stats
from scipy.stats import norm, skew #for some statistics

sns.distplot(high['income13'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(high['income13'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('RentPrice distribution')

fig = plt.figure()
res = stats.probplot(high['income13'], plot=plt)
plt.show()
```

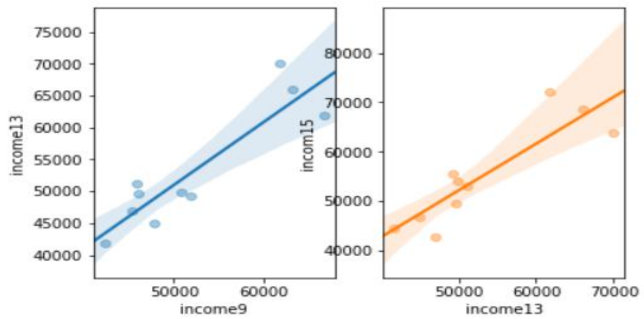
```
mu = 53120.00 and sigma = 8956.54
```

In [45]:
```python
#  Johnson Su distribution plot for 2005

import scipy.stats as st
y = high[['income13']]
plt.figure(1); plt.title('Johnson SU')
sns.distplot(y, kde=False, fit=st.johnsonsu)
plt.figure(2); plt.title('Normal')
sns.distplot(y, kde=False, fit=st.norm)
plt.figure(3); plt.title('Log Normal')
sns.distplot(y, kde=False, fit=st.lognorm)
```

Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x225508099b0>
```



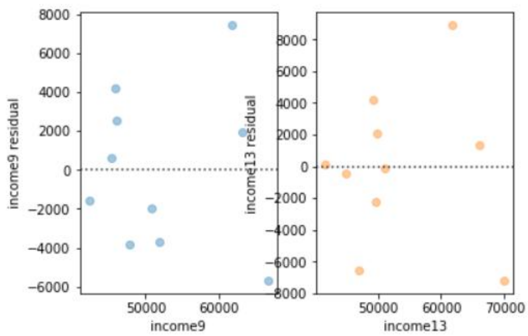
In [47]:
```python
# Regression plot
fig, ax = plt.subplots(1,2)
sns.regplot('income9','income13',high, ax=ax[0], scatter_kws={'alpha':0.4})
sns.regplot('income13','incom15',high,ax=ax[1], scatter_kws={'alpha':0.4})
```
Out[47]: `<matplotlib.axes._subplots.AxesSubplot at 0x22551a04550>`



the model for the chart on the left is very accurate, there's a strong correlation between the model's predictions and its actual results.

When you run a regression, Statwing automatically calculates and plots **residuals** to help you understand and improve your regression model.

In [48]:
```python
# visualizing the residuals by creating residual plots .
fig, ax = plt.subplots(1,2)
ax[0]= sns.residplot('income9','income13', high,ax=ax[0], scatter_kws={'alpha':0.4})
ax[0].set_ylabel('income9 residual')
ax[1]=sns.residplot('income13','incom15',high,ax=ax[1], scatter_kws={'alpha':0.4})
ax[1].set_ylabel('income13 residual')
```
Out[48]: `Text(0, 0.5, 'income13 residual')`



Model summary

In [49]:
```python
# Next we'll want to fit a linear regression model. We need to choose variables that we think we'll be good predictors for the d
# This can be done by checking the correlation(s) between variables, on what variables are good predictors of y.

import statsmodels.api as sm

X = high["income13"]
y = high["incom15"]

# Note the difference in argument order
model = sm.OLS(y, X).fit()
predictions = model.predict(X) # make the predictions by the model

# Print out the statistics
model.summary()
```

Out[49]:

OLS Regression Results

Dep. Variable:	incom15	R-squared:	0.993
Model:	OLS	Adj. R-squared:	0.993
Method:	Least Squares	F-statistic:	1342.
Date:	Mon, 26 Aug 2019	Prob (F-statistic):	4.17e-11
Time:	21:55:24	Log-Likelihood:	-98.445
No. Observations:	10	AIC:	198.9
Df Residuals:	9	BIC:	199.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
income13	1.0341	0.028	36.633	0.000	0.970	1.098

Omnibus:	0.457	Durbin-Watson:	1.436
Prob(Omnibus):	0.796	Jarque-Bera (JB):	0.122
Skew:	-0.229	Prob(JB):	0.941
Kurtosis:	2.712	Cond. No.	1.00

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [50]:
```
# # Next we'll want to fit a linear regression model by adding a constant.
X = high["income13"] ## X usually means our input variables (or independent variables)
y = high["incom15"] ## Y usually means our output/dependent variable
X = sm.add_constant(X) ## let's add an intercept (beta_0) to our model

# Note the difference in argument order
model = sm.OLS(y, X).fit() ## sm.OLS(output, input)
predictions = model.predict(X)

# Print out the statistics
model.summary()
```

Out[50]:

OLS Regression Results

Dep. Variable:	incom15	R-squared:	0.782
Model:	OLS	Adj. R-squared:	0.754
Method:	Least Squares	F-statistic:	28.63
Date:	Mon, 26 Aug 2019	Prob (F-statistic):	0.000685
Time:	21:57:30	Log-Likelihood:	-98.303
No. Observations:	10	AIC:	200.6
Df Residuals:	8	BIC:	201.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4593.2321	9564.469	0.480	0.644	-1.75e+04	2.66e+04
income13	0.9501	0.178	5.351	0.001	0.541	1.359

Omnibus:	0.356	Durbin-Watson:	1.549
Prob(Omnibus):	0.837	Jarque-Bera (JB):	0.064
Skew:	0.135	Prob(JB):	0.969
Kurtosis:	2.717	Cond. No.	3.24e+05

In [51]:
```
# Loading the monthly debt payments
market= pd.read_csv(r"C:\\Users\\susie\\Documents\\Data Science\\Capstone data\\Housing market indicators.csv"
                    ,encoding="latin1")
```

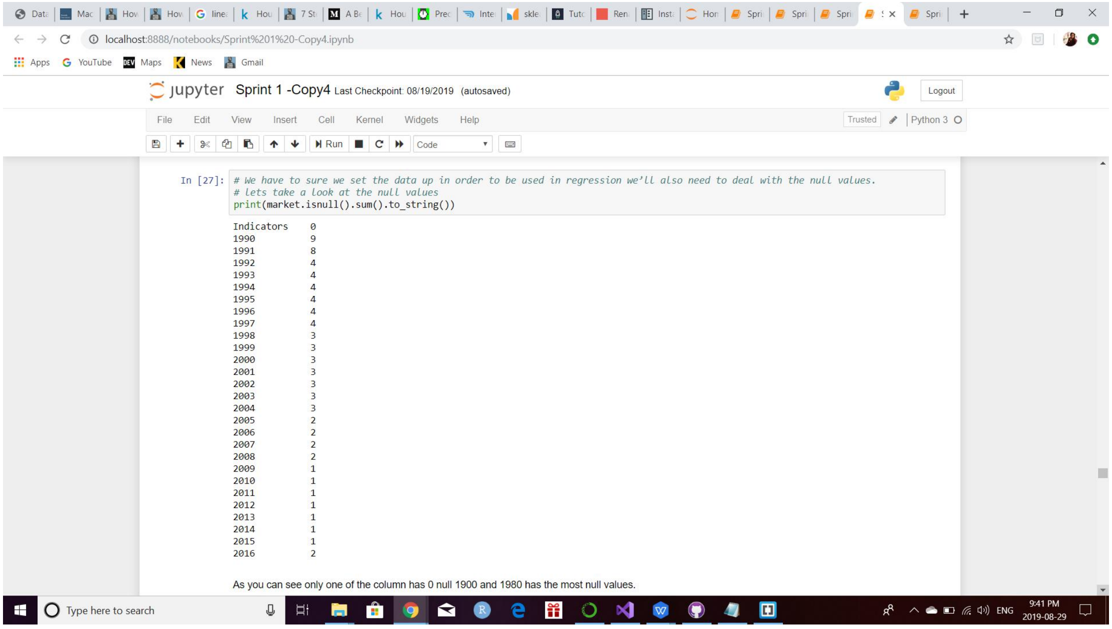
In [52]:
```
market.head(5)
```

Out[52]:

	Indicators	1990	1991	1992	1993	1994	1995	1996	1997	1998	...	2007	2008	2009	2010	2011	2012	20
0	Single-detached	32425.0	26290.0	27868.0	26240.0	30036.0	20124.0	27019.0	35401.0	32737.0	...	37910.0	31108.0	22634.0	28089.0	26884.0	25567.0	23270
1	Multiple	30224.0	26504.0	27904.0	18900.0	16609.0	15694.0	16043.0	18671.0	21093.0	...	30213.0	43968.0	27736.0	32344.0	40937.0	51175.0	37815
2	Semi-detached	2338.0	1730.0	2611.0	2537.0	3421.0	2306.0	3348.0	4299.0	4575.0	...	4284.0	3415.0	3007.0	3006.0	3142.0	3397.0	3116
3	Row	8462.0	9472.0	9246.0	7448.0	7226.0	6175.0	8124.0	9964.0	10073.0	...	11255.0	11212.0	7121.0	10255.0	9288.0	10577.0	9427
4	Apartment	19424.0	15302.0	16047.0	8915.0	5962.0	7213.0	4571.0	4408.0	6445.0	...	14674.0	29341.0	17608.0	19083.0	28507.0	37201.0	25272

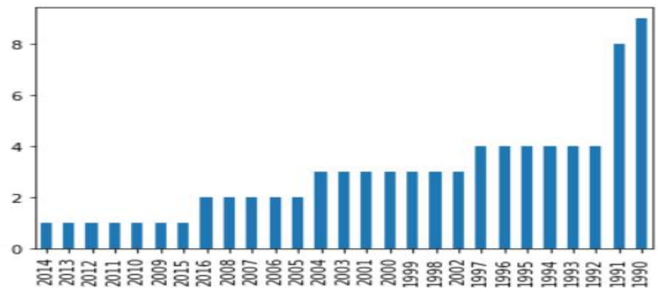
5 rows × 28 columns

Missing values

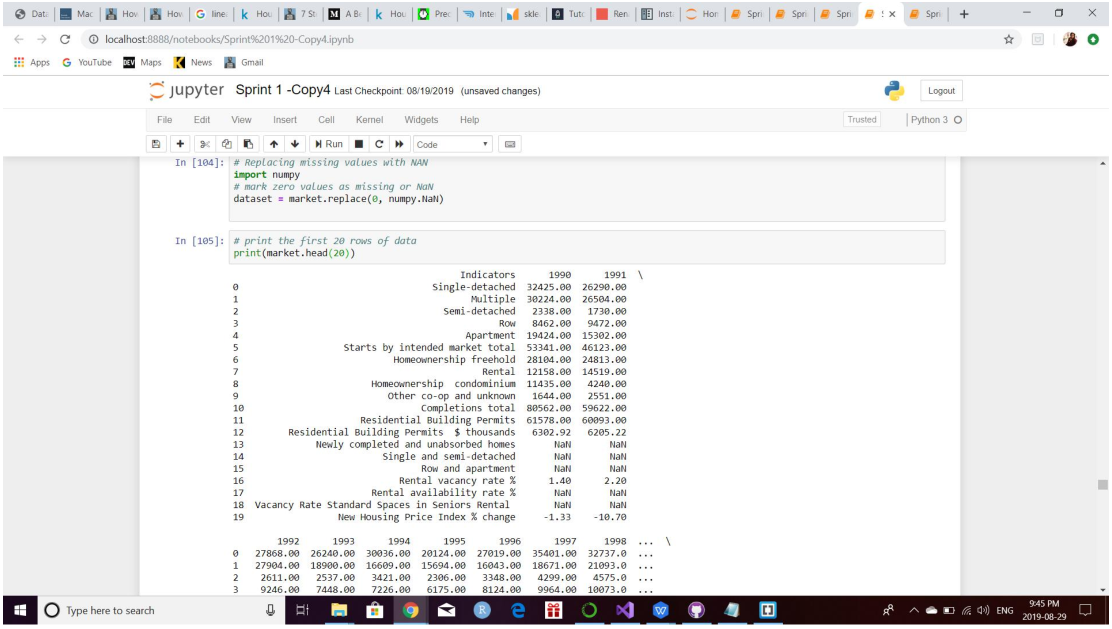


Plotting the missing values

```
In [54]: # Here we have a visual of the missing data
missing = market.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing.plot.bar()

Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x22551b2f7b8>
```



Replacing missing values with NAN



I had a very difficult time working with this dataset not only was the dataset small but because the dataset didn't have a lot of variables for me to work with. I was evaluating price against price which is not a realistic approach. The price against approach end up giving all perfect plots for our modeling section. With the project being analysing rent prices it's evident that the price will go up every year. Plotting the graphs