Traffic, Population & Employment as Predictors of US GDP

By Kamron Afshar & Shishir Kumar

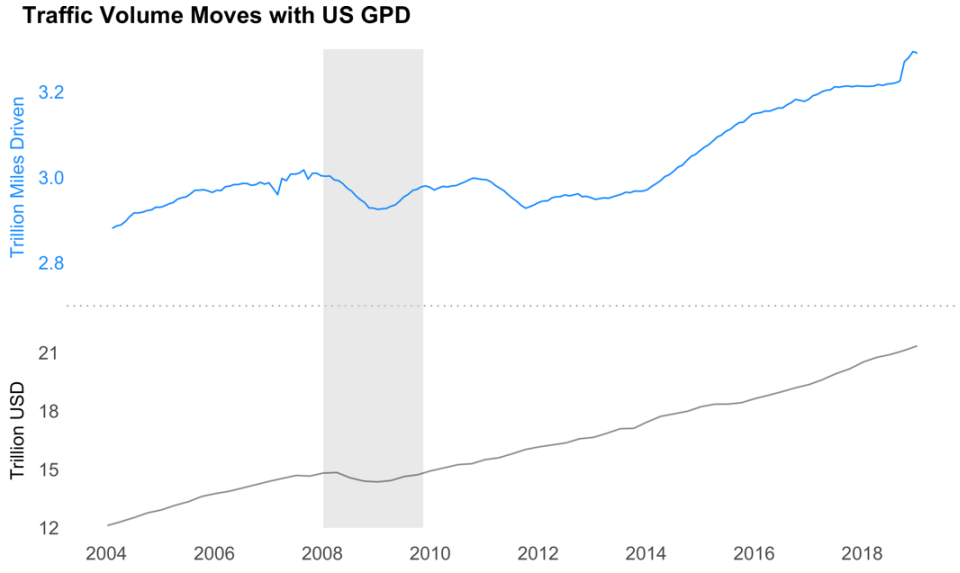
1. **Dataset:** 60 values corresponding with 3-month calendar quarters
    a. Response Variable
        i. GDP: US Gross Domestic Product in Billion US Dollars
    b. Predictors
        i. 12 month rolling miles driven across us, averaged at the quarter level
        ii. % total mileage contribution in the North East US
        iii. % total mileage contribution in the North Central US
        iv. % total mileage contribution in the South Atlantic US
        v. % total mileage contribution in the West US
        vi. % total mileage contribution from Urban Arterial Roads
        vii. US Population
        viii. % US Employment

2. **Research Problem:** Living in the Bay Area during the "Great Recession of 2008" Kamron noticed a decrease in the traffic levels, which later increased greatly as the economy recovered. Furthermore, there is some logical credence to the notion that traffic, population, and employment can generate or indicate a strong economy.

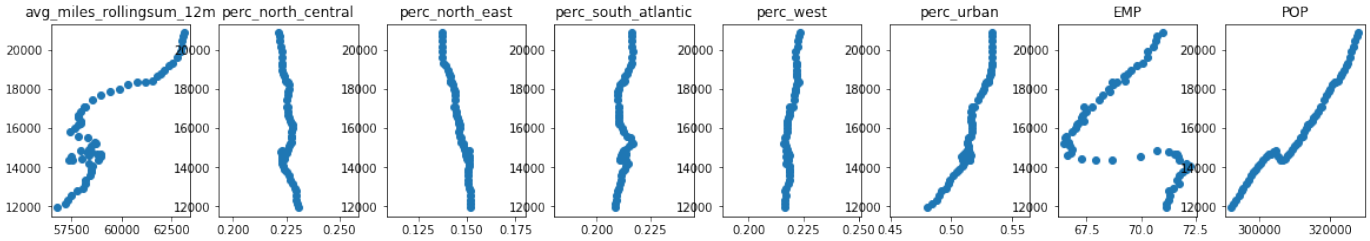
    We determined based on the graphs below there is some relationship between traffic and the US GDP which led us to further analyze this relationship.

    We chose to attempt a Multiple Linear Regression model to determine which combination of the variables can best predict the US GDP. Given the nature of the data is clear there is some autocorrelation/ time related effect, but we will attempt to use MLR; a Time Series Analysis would probably be the most appropriate.

3. **Exploratory Analysis:** The graph below is what encouraged us to examine this trend in the first place. It shows 12 month rolling sum of the overall vehicular miles driven in the US every year, contrasted with yearly GDP. We can see that the movement in GDP is almost mirrored by miles data. It is important to note that there is a dip during the recession of 2008 which will have an affect across multiple variables in our model.



Based on the following scatter plots, there does appear to be strong relationships between some of these variables although there is an irregularity within the plots for Employment Rate, Rolling Sum of miles, and Population. We suspect this is due the effect of the recession on these variables. It also might be worthwhile to consider rescaling the data in some way based on the shear slope of the relationships indicated in the scatter plots.



Based on these scatter plots we felt this analysis can be fruitful. That being said, we must consider that the direction of relationship we are assuming here might be reversed. Another analysis might treat our predictor variables as responses to GDP instead of GDP as the response.

Also, our variables for percentage contribution of miles driven are interrelated which will probably lead to correlation between these variables. The correlation heat map below confirms many of our predictors have intercorrelation.

4&5: Regression Analysis & Model Selection: We started with an MLS model with all of our variables included which produced the regression summary below:

OLS Regression Results

Dep. Variable:	GDP	**R-squared:**	0.997
Model:	OLS	**Adj. R-squared:**	0.997
Method:	Least Squares	**F-statistic:**	2505.
Date:	Sun, 13 Oct 2019	**Prob (F-statistic):**	2.26e-63
Time:	03:30:51	**Log-Likelihood:**	-372.29
No. Observations:	60	**AIC:**	762.6
Df Residuals:	51	**BIC:**	781.4
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.011e+05	1.66e+04	-6.075	0.000	-1.34e+05	-6.77e+04
avg_miles_rollingsum_12m	-0.1009	0.048	-2.120	0.039	-0.196	-0.005
perc_north_central	1.061e+04	1.85e+04	0.572	0.570	-2.66e+04	4.78e+04
perc_north_east	5112.3242	3.05e+04	0.168	0.867	-5.6e+04	6.63e+04
perc_south_atlantic	5.525e+04	1.96e+04	2.822	0.007	1.59e+04	9.46e+04
perc_west	6.735e+04	3.37e+04	1.999	0.051	-303.523	1.35e+05
perc_urban	-1.833e+04	9329.298	-1.964	0.055	-3.71e+04	403.630
EMP	297.2237	27.069	10.980	0.000	242.880	351.568
POP	0.2645	0.021	12.881	0.000	0.223	0.306

Omnibus:	0.424	**Durbin-Watson:**	0.811
Prob(Omnibus):	0.809	**Jarque-Bera (JB):**	0.575
Skew:	-0.159	**Prob(JB):**	0.750
Kurtosis:	2.642	**Cond. No.**	8.01e+08

This regression summary output helped us determine at a surface-level which variable were not helpful in prediction. R Squared value is very high but that is probably due to the correlation between the variables. Just based on the P values we chose to remove % North Central and % North East. Although the p value for % Urban and % West variables are over 5%, they are very close so we kept them in the model.

After removing these variables, we noted a decrease in AIC & BIC while maintaining the same R Squared. The p values for % contribution West and Urban decreased to below the threshold which led to us keeping these variables.

OLS Regression Results

Dep. Variable:	GDP	**R-squared:**	0.997
Model:	OLS	**Adj. R-squared:**	0.997
Method:	Least Squares	**F-statistic:**	3448.
Date:	Sun, 13 Oct 2019	**Prob (F-statistic):**	7.72e-67
Time:	03:30:54	**Log-Likelihood:**	-372.49
No. Observations:	60	**AIC:**	759.0
Df Residuals:	53	**BIC:**	773.6
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-9.522e+04	7259.016	-13.118	0.000	-1.1e+05	-8.07e+04
avg_miles_rollingsum_12m	-0.0959	0.045	-2.136	0.037	-0.186	-0.006
perc_south_atlantic	5.21e+04	1.79e+04	2.914	0.005	1.62e+04	8.8e+04
perc_west	6.689e+04	3.24e+04	2.068	0.044	1999.845	1.32e+05
perc_urban	-2.034e+04	5129.333	-3.964	0.000	-3.06e+04	-1e+04
EMP	288.0799	19.587	14.708	0.000	248.794	327.366
POP	0.2626	0.008	33.602	0.000	0.247	0.278

Omnibus:	1.073	**Durbin-Watson:**	0.803
Prob(Omnibus):	0.585	**Jarque-Bera (JB):**	1.100
Skew:	-0.216	**Prob(JB):**	0.577
Kurtosis:	2.497	**Cond. No.**	6.99e+08

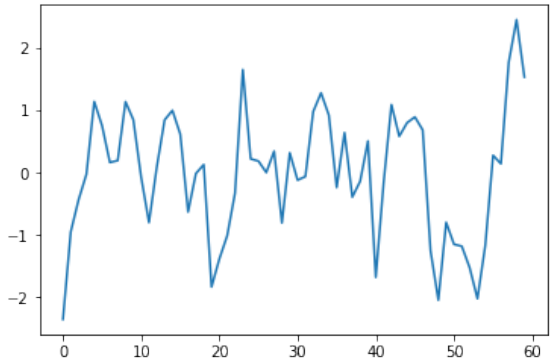
An analysis of VIF indicated there is strong multicollinearity in our data unfortunately. This is to be expected based on our understanding of these variables. The relationship between any two of these variables would provide ample opportunity for analysis on their own. Given this however we chose to proceed with the given variables in our second model: Miles, % South Atlantic, % West, % Urban, Employment Rate and Population.

6. Model Diagnosis:

a. First we analyzed the heteroskedasticity. LM and F probability values were very close to 5% but slightly over. There is likely some heteroskedasticity but given these results arguably the assumption of constant variance is safe.

b. Next we tested Auto correlation. P values for both LM and F indicated there is definitely autocorrelation. This was our assumption from the beginning, but we chose to proceed to learn more about the data. This confirms that we should be using a time series type analysis here in place of MLR.

c. Based on the plot of externally studentized residuals below, the residuals appear to be randomized around 0 so that assumption is satisfied.



d. based on the DFFITS the first value, the 24th value and the second to last value appear to be influential. We can't seem to explain why the first and last would be influential so those weren't removed. The 24th however is related to the recession, which is somewhat expected. We believe we should keep this data point because it is an important element in analyzing the relationships between these predictors and GDP. Perhaps another variable to indicated where the economy is in recession or not would be valuable to account for a change in the relationships during economic downturns.

7. Model of Choice: We performed a general linear F test between the reduced and full mode. The p-value comes out to be much greater than 0.05, hence we cannot reject the null hypothesis and accept the reduced model with parameters as mileage, % South Atlantic, % West, % Urban, Employment Rate and Population. The coefficients are a bit hard to understand given some are negative despite a seeming positive correlation in the individual scatter plots. Perhaps the negative coefficient is accounting for the multicollinearity in a way. We expected each variable to have a positive coefficient.

8. Summary: While we were able to generate a model with a high R Squared value, there is obviously a lot of underlying interactions that render this model less than ideal for accurate predictions. Most importantly, this would be a more apt example of a time series analysis problem instead of MLR. That being said we were able to prove there is

some relationship between these variables as expected, and in the case of some of the variables a visibly strong relationship. The expertise of an economist would be helpful in determining other potential variables we should include as well as potential transformation to these variables/response to account for the interrelationship and economic cycles. Some of our variables are “adjusted values” which rely on some analysis that was not clear to us to smooth out or improve accuracy. To take this model further we would need to understand how the adjustments work and apply them across all our variables to improve accuracy.

Regression Analysis

In [25]:

```
import os
#os.environ['KMP_DUPLICATE_LIB_OK']='True'
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.stats.outliers_influence import variance_inflation_factor as vi
f
import scipy.stats as stats
import matplotlib.cm as cm
from IPython.display import display
from mpl_toolkits.mplot3d import Axes3D
from sklearn.feature_selection import f_regression
from statsmodels.stats.anova import anova_lm
import matplotlib.pyplot as plt
```

In [26]:

```
mileage = pd.read_csv('data_reshaped.csv')
gdp=pd.read_csv('GDP.csv')
US_EMP = pd.read_csv('US_EMP_RATE.csv')
```

In [27]:

```
US_EMP = US_EMP.iloc[:-1,:]
US_EMP.columns = ['date1','EMP']
US_POP = pd.read_csv('US_POP.csv')
US_POP = US_POP.iloc[:-1,:]
US_POP.columns = ['date2','POP']
```

In [28]:

```
len(mileage)==len(gdp)==len(US_EMP)==len(US_POP) == 60
```

Out[28]:

```
True
```

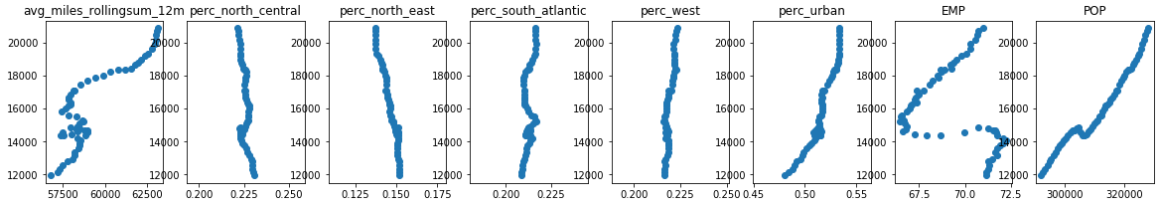
In [29]:

```
data = pd.concat([mileage,gdp,US_EMP,US_POP],axis=1)
data = data.drop(columns = ['DATE','date2','date1','quarter'])
features = data.drop(columns = ['GDP'])
```

See relationship between the response and predictors

In [30]:

```
plt.figure(figsize=(20, 3))
for i,col in enumerate(features.columns):
    plt.subplot(1,len(features.columns),i+1)
    plt.scatter(features[col],data.GDP)
    plt.title(col)
```



Correlation between the response and predictors

In [31]:

```
import seaborn as sns
correlation_matrix = data.corr().round(2)
# annot = True to print the values inside the square
plt.figure(figsize = (16,5))
sns.heatmap(data=correlation_matrix, annot=True)
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c2383ab00>
```

Full Model

In [33]:

```
reg_M1 = smf.ols('GDP~avg_miles_rollingsum_12m + perc_north_central + \
                  perc_north_east + perc_south_atlantic + perc_west + \
                  perc_urban + EMP + POP',data=data).fit()
reg_M1.summary()
```

Out[33]:

OLS Regression Results

Dep. Variable:	GDP	**R-squared:**	0.997
Model:	OLS	**Adj. R-squared:**	0.997
Method:	Least Squares	**F-statistic:**	2505.
Date:	Sun, 13 Oct 2019	**Prob (F-statistic):**	2.26e-63
Time:	20:17:20	**Log-Likelihood:**	-372.29
No. Observations:	60	**AIC:**	762.6
Df Residuals:	51	**BIC:**	781.4
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.011e+05	1.66e+04	-6.075	0.000	-1.34e+05	-6.77e+04
avg_miles_rollingsum_12m	-0.1009	0.048	-2.120	0.039	-0.196	-0.005
perc_north_central	1.061e+04	1.85e+04	0.572	0.570	-2.66e+04	4.78e+04
perc_north_east	5112.3242	3.05e+04	0.168	0.867	-5.6e+04	6.63e+04
perc_south_atlantic	5.525e+04	1.96e+04	2.822	0.007	1.59e+04	9.46e+04
perc_west	6.735e+04	3.37e+04	1.999	0.051	-303.523	1.35e+05
perc_urban	-1.833e+04	9329.298	-1.964	0.055	-3.71e+04	403.630
EMP	297.2237	27.069	10.980	0.000	242.880	351.568
POP	0.2645	0.021	12.881	0.000	0.223	0.306

Omnibus:	0.424	**Durbin-Watson:**	0.811
Prob(Omnibus):	0.809	**Jarque-Bera (JB):**	0.575
Skew:	-0.159	**Prob(JB):**	0.750
Kurtosis:	2.642	**Cond. No.**	8.01e+08

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.01e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

Reduced Model

In [34]:

```
features2 = features.drop(columns=['perc_north_east','perc_north_east'])
```

In [35]:

```
reg_M2 = smf.ols('GDP~avg_miles_rollingsum_12m + perc_south_atlantic + perc_west
+ \
                 perc_urban + EMP + POP',data=data).fit()
```

In [36]:

```
reg_M2.summary()
```

Out[36]:

OLS Regression Results

Dep. Variable:	GDP	**R-squared:**	0.997
Model:	OLS	**Adj. R-squared:**	0.997
Method:	Least Squares	**F-statistic:**	3448.
Date:	Sun, 13 Oct 2019	**Prob (F-statistic):**	7.72e-67
Time:	20:17:27	**Log-Likelihood:**	-372.49
No. Observations:	60	**AIC:**	759.0
Df Residuals:	53	**BIC:**	773.6
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-9.522e+04	7259.016	-13.118	0.000	-1.1e+05	-8.07e+04
avg_miles_rollingsum_12m	-0.0959	0.045	-2.136	0.037	-0.186	-0.006
perc_south_atlantic	5.21e+04	1.79e+04	2.914	0.005	1.62e+04	8.8e+04
perc_west	6.689e+04	3.24e+04	2.068	0.044	1999.845	1.32e+05
perc_urban	-2.034e+04	5129.333	-3.964	0.000	-3.06e+04	-1e+04
EMP	288.0799	19.587	14.708	0.000	248.794	327.366
POP	0.2626	0.008	33.602	0.000	0.247	0.278

Omnibus:	1.073	**Durbin-Watson:**	0.803
Prob(Omnibus):	0.585	**Jarque-Bera (JB):**	1.100
Skew:	-0.216	**Prob(JB):**	0.577
Kurtosis:	2.497	**Cond. No.**	6.99e+08

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.99e+08. This might indicate that there are
strong multicollinearity or other numerical problems.

Multicollinearity

In [37]:

```
[vif(np.array(features2),i) for i in range(1,len(features2.columns))]
```

Out[37]:

```
[29003.525591546262,
 18901.71434688995,
 101389.48686298347,
 49777.60035428292,
 7869.024549142253,
 23646.53179038305]
```

In [38]:

```
from statsmodels.stats.diagnostic import het_breuschpagan
bp_test = het_breuschpagan(reg_M2.resid, reg_M2.model.exog)
labels = ['LM Statistic', 'LM-Test p-value', 'F-Statistic', 'F-Test p-value']
print(dict(zip(labels, bp_test)))
```

```
{'LM Statistic': 12.04268347554496, 'LM-Test p-value': 0.06102333452
9280264, 'F-Statistic': 2.218160753699983, 'F-Test p-value': 0.05543
935852191886}
```

Serial Correlation

In [39]:

```
bg_test=sm.stats.diagnostic.acorr_breusch_godfrey(reg_M2)
labels = ['LM Statistic', 'LM-Test p-value', 'F-Statistic', 'F-Test p-value']
print(dict(zip(labels, bg_test)))
```

```
{'LM Statistic': 28.239651583435375, 'LM-Test p-value': 0.0016524026
139669256, 'F-Statistic': 3.8233365773028156, 'F-Test p-value': 0.00
09555603321952383}
```

Normality Check

In [40]:

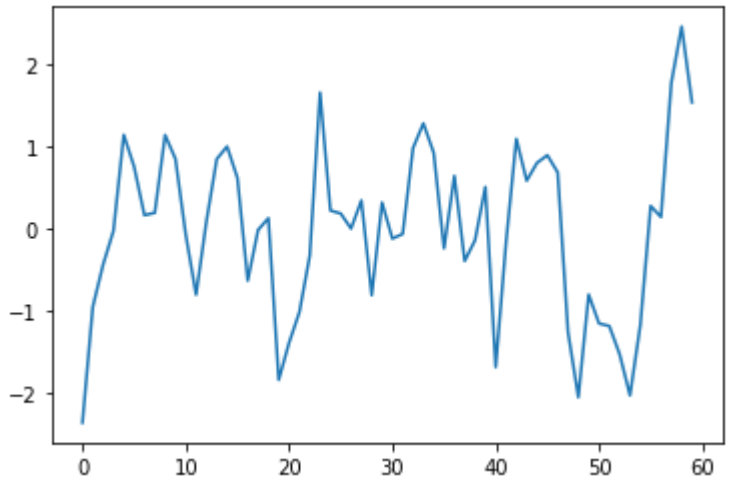
```
infl = reg_M2.get_influence()
```

In [41]:

```
plt.plot(data.index,infl.resid_studentized_external)
```

Out[41]:

```
[<matplotlib.lines.Line2D at 0x1c21554470>]
```



Influential Points

In [42]:

```
n = len(data)
p = len(features2.columns) + 1
```

In [43]:

```
inflsum = infl.summary_frame()
reg_dffits=inflsum.dffits
dffits_thresh = 2*np.sqrt((p+1)/(n-p-1))
atyp_dffits = np.abs(reg_dffits) > dffits_thresh
a_dffits_ind = set(data.index[atyp_dffits])
a_dffits_ind
```

Out[43]:

```
{0, 23, 58}
```

F-Test

In [58]:

```
## F-test : reduced vs full model
## reg_M2 : reduced model (i.e. null hypothesis), reg_M1 : full model (ie altern
ate hypothesis)
F_stat = ((reg_M2.ssr - reg_M1.ssr) / (reg_M2.df_resid - reg_M1.df_resid) / (reg
_M1.ssr / reg_M1.df_resid))
F_stat
```

Out[58]:

```
0.16600064173028078
```

In [61]:

```
## p - value is 0.15 >> 0.05, hence we donot have enough information to reject t
he null hypothesis
## hence we accept the reduced model
import scipy.stats as stats
alpha = 0.05
p_value = stats.f.cdf(F_stat, reg_M2.df_resid - reg_M1.df_resid, reg_M1.df_resid
)
p_value
```

Out[61]:

```
0.1524984894387115
```

```
#### Collating required datapoints, reshaping them and calculating important features ###

import xlrd
import pandas as pd
import numpy as np
import os
from time import strptime
import datetime

def get_arterial(file_path,category):
    """
    variable path is the path of the xls workbook and category is "rural" / "urban" / "all",
    returns dataframe containing the values for given category for each state
    """
    book = xlrd.open_workbook(file_path)
    file_name = os.path.basename(file_path)
    year = str(20) + "".join([str(s) for s in file_name if s.isdigit()]) ## gets the year from
filename
    Month = strptime(file_name[2:5],'%b').tm_mon ## gets month no
    mydate = datetime.date(int(year),Month, 1) ## first day of the month and year
    mydate_1 = mydate - datetime.timedelta(days=1) ## interested in last month of this year
as data corresponds to last month and same year
    mydate_2 = mydate - datetime.timedelta(days=368) ## interested in last month of last
year as data corresponds to last month and last year
    #monthid1 = str(mydate_1.strftime("%Y")) + str(mydate_1.strftime("%m")) ## 200706 for
July 2007 file
    monthid2 = str(mydate_2.strftime("%Y")) + str(mydate_2.strftime("%m")) ## 200606 for
July 2007 file
    try:
        if category.lower() == "rural":
            index = 3
        elif category.lower() == "urban":
            index = 4
        else:
            index = 5
        sheet = book.sheet_by_index(index)
        list_states = sheet.col_values(0)
        xstart = list_states.index('Connecticut')
        xend = list_states.index('TOTALS')
        #list1 = sheet.col_slice(colx= 8,start_rowx=xstart,end_rowx= xend - 1)
        #list1 = [w.value for w in list1]
        list2 = sheet.col_slice(colx= 9,start_rowx=xstart,end_rowx= xend - 1)
        list2 = [w.value for w in list2]
        list3 = sheet.col_slice(colx= 0,start_rowx=xstart,end_rowx= xend - 1)
        list3 = [w.value.lower() for w in list3] ## take lowercase for direct match later
        df = pd.concat([pd.DataFrame(list3),pd.DataFrame(list2)], axis = 1) #
,pd.DataFrame(list1)
```

```
        #col_name_1 = category + '_Arterial_' + monthid1
        col_name_2 = category + '_Arterial_' + monthid2
        df.columns = ['State', col_name_2 ] # col_name_1,
        df[col_name_2].replace('', np.nan, inplace=True) ## removes rows with blank records (
zonal categories)
        df['State'].replace('', np.nan, inplace=True)
        curr_monthid = str(mydate.strftime("%Y")) + str(mydate.strftime("%m")) ## 200707 for
July 2007 file
        df['data_monthid'] = curr_monthid
        df.dropna(subset=[col_name_2], inplace=True)
        df.dropna(subset=['State'], inplace=True)
        df = df[~df.State.str.contains("subtotal")] ### causes problems on joins, there in most
files
        df = df[df.State != "total"] ## causes problems on joins, is there only in specific files
        df['State'] = df.State.str.strip() ## removes leading and lagging white spaces if any
        df2 = pd.melt(df,id_vars=['State','data_monthid'],var_name=['category'],
value_name='Million_Vehicle_Miles')
        return df2
    except:
        print("error in file ",os.path.basename(file_path))

## get all the files
def filelist(root):
    """Return a fully-qualified list of filenames under root directory"""
    allfiles = []
    for path, subdirs, files in os.walk(root):
        for name in files:
            if name.find("xls") >= 0:
                allfiles.append(os.path.join(path, name))
    return allfiles

file_list = filelist('/Users/MrMndFkr/Desktop/linear-regression-project/Datasets')


### check function get_arterial and append dataframes for Dataset 1
for file in file_list:
    try:
        df1 = get_arterial(file,"Rural")
        df2 = get_arterial(file,"Urban")
        df3 = get_arterial(file,"All")
        df_final = pd.concat([df1,df2,df3], axis = 0)
        #df_temp = pd.merge(df1,df2, how = 'inner', on = 'State')
        #df_final = pd.merge(df_temp,df3, how = 'inner', on = 'State')
        #assert df_final.shape[0] == df3.shape[0]
        #assert df_final.shape[0] == df_temp.shape[0]
    except:
        print('error encountered at ' + os.path.basename(file))
```

```
## get view of large no of columns and rows
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

## appending these dataframes
df1 = get_arterial(file_list[0],"Rural")
df2 = get_arterial(file_list[0],"Urban")
df3 = get_arterial(file_list[0],"All")
df_final = pd.concat([df1,df2,df3], axis = 0)
#df_temp = pd.merge(df1,df2, how = 'inner', on = 'State')
#df_final = pd.merge(df_temp,df3, how = 'inner', on = 'State')
for file in file_list[1:]:
   try:
      df1 = get_arterial(file,"Rural")
      df2 = get_arterial(file,"Urban")
      df3 = get_arterial(file,"All")
      df_final = pd.concat([df_final,df1,df2,df3], axis = 0)
      #df_temp = pd.merge(df1,df2, how = 'inner', on = 'State')
      #df_temp2 = pd.merge(df_temp,df3, how = 'inner', on = 'State')
      #df_final = pd.merge(df_final,df_temp2, how = 'inner', on = 'State')
      #assert df_final.shape[0] == df_temp.shape[0]
      #assert df_final.shape[0] == df_temp2.shape[0]
   except:
      print('error encountered at ' + os.path.basename(file))


## only change is that the required datapoints are in cols 7 and 8 for Dataset III ( in Dataset
I, they are in cols 9 and 10)
def get_arterial(file_path,category):
   """
   variable path is the path of the xls workbook and category is "rural" / "urban" / "all",
   returns dataframe containing the values for given category for each state
   """
   book = xlrd.open_workbook(file_path)
   file_name = os.path.basename(file_path)
   year = str(20) + "".join([str(s) for s in file_name if s.isdigit()]) ## gets the year from
filename
   Month = strptime(file_name[2:5],'%b').tm_mon ## gets month no
   mydate = datetime.date(int(year),Month, 1) ## first day of the month and year
   #mydate_1 = mydate - datetime.timedelta(days=1) ## interested in last month of this
year as data corresponds to last month and same year
   mydate_2 = mydate - datetime.timedelta(days=368) ## interested in last month of last
year as data corresponds to last month and last year
   #monthid1 = str(mydate_1.strftime("%Y")) + str(mydate_1.strftime("%m")) ## 200706 for
July 2007 file
```

```
   monthid2 = str(mydate_2.strftime("%Y")) + str(mydate_2.strftime("%m")) ## 200606 for
July 2007 file
   try:
      if category.lower() == "rural":
         index = 3
      elif category.lower() == "urban":
         index = 4
      else:
         index = 5
      sheet = book.sheet_by_index(index)
      list_states = sheet.col_values(0)
      xstart = list_states.index('Connecticut')
      xend = list_states.index('TOTALS')
      #list1 = sheet.col_slice(colx= 6,start_rowx=xstart,end_rowx= xend - 1)
      #list1 = [w.value for w in list1]
      list2 = sheet.col_slice(colx= 7,start_rowx=xstart,end_rowx= xend - 1)
      list2 = [w.value for w in list2]
      list3 = sheet.col_slice(colx= 0,start_rowx=xstart,end_rowx= xend - 1)
      list3 = [w.value.lower() for w in list3] ## take lowercase for direct match later
      df = pd.concat([pd.DataFrame(list3),pd.DataFrame(list2)], axis = 1) #
pd.DataFrame(list1),
      #col_name_1 = category + '_Arterial_' + monthid1
      col_name_2 = category + '_Arterial_' + monthid2
      df.columns = ['State',  col_name_2 ] ## col_name_1,
      df[col_name_2].replace('', np.nan, inplace=True) ## removes rows with blank records (
zonal categories)
      df['State'].replace('', np.nan, inplace=True)
      curr_monthid = str(mydate.strftime("%Y")) + str(mydate.strftime("%m")) ## 200707 for
July 2007 file
      df['data_monthid'] = curr_monthid
      df.dropna(subset=[col_name_2], inplace=True)
      df.dropna(subset=['State'], inplace=True)
      df = df[~df.State.str.contains("subtotal")] ### causes problems on joins, there in most
files
      df = df[df.State != "total"] ## causes problems on joins, is there only in specific files
      df['State'] = df.State.str.strip() ## removes leading and lagging white spaces if any
      df2 = pd.melt(df,id_vars=['State','data_monthid'],var_name=['category'],
value_name='Million_Vehicle_Miles')
      return df2
   except:
      print("error in file ",os.path.basename(file_path))

# get new filelist from dataset III
file_list = filelist('/Users/MrMndFkr/Desktop/linear-regression-project/Datasets_III')

## get collated dataset
#df1 = get_arterial(file_list[0],"Rural")
```

```
#df2 = get_arterial(file_list[0],"Urban")
#df3 = get_arterial(file,"All")
#df_temp = pd.merge(df1,df2, how = 'inner', on = 'State')
#df_final = pd.merge(df_temp,df3, how = 'inner', on = 'State')
for file in file_list[1:]:
    try:
        df1 = get_arterial(file,"Rural")
        df2 = get_arterial(file,"Urban")
        df3 = get_arterial(file,"All")
        df_final = pd.concat([df_final,df1,df2,df3], axis = 0)
        #df_temp = pd.merge(df1,df2, how = 'inner', on = 'State')
        #df_temp2 = pd.merge(df_temp,df3, how = 'inner', on = 'State')
        #df_final = pd.merge(df_final,df_temp2, how = 'inner', on = 'State')
        #assert df_final.shape[0] == df_temp.shape[0]
        #assert df_final.shape[0] == df_temp2.shape[0]
    except:
        print('error encountered at ' + os.path.basename(file))


## Now fetching the 2018 data for months after Jun 2018
def get_arterial(file_path,category):
    """
    variable path is the path of the xls workbook and category is "rural" / "urban" / "all",
    returns dataframe containing the values for given category for each state
    """
    book = xlrd.open_workbook(file_path)
    file_name = os.path.basename(file_path)
    year = str(20) + "".join([str(s) for s in file_name if s.isdigit()]) ## gets the year from
filename
    Month = strptime(file_name[2:5],'%b').tm_mon ## gets month no
    mydate = datetime.date(int(year),Month, 1) ## first day of the month and year
    mydate_1 = mydate - datetime.timedelta(days=1) ## interested in last month of this year
as data corresponds to last month and same year
    #mydate_2 = mydate - datetime.timedelta(days=368) ## interested in last month of last
year as data corresponds to last month and last year
    monthid1 = str(mydate_1.strftime("%Y")) + str(mydate_1.strftime("%m")) ## 200706 for
July 2007 file
    #monthid2 = str(mydate_2.strftime("%Y")) + str(mydate_2.strftime("%m")) ## 200606 for
July 2007 file
    try:
        if category.lower() == "rural":
            index = 3
        elif category.lower() == "urban":
            index = 4
        else:
            index = 5
        sheet = book.sheet_by_index(index)
```

```
        list_states = sheet.col_values(0)
        xstart = list_states.index('Connecticut')
        xend = list_states.index('TOTALS')
        #list1 = sheet.col_slice(colx= 6,start_rowx=xstart,end_rowx= xend - 1)
        #list1 = [w.value for w in list1]
        list2 = sheet.col_slice(colx= 8,start_rowx=xstart,end_rowx= xend - 1)
        list2 = [w.value for w in list2]
        list3 = sheet.col_slice(colx= 0,start_rowx=xstart,end_rowx= xend - 1)
        list3 = [w.value.lower() for w in list3] ## take lowercase for direct match later
        df = pd.concat([pd.DataFrame(list3),pd.DataFrame(list2)], axis = 1) #
pd.DataFrame(list1),
        #col_name_1 = category + '_Arterial_' + monthid1
        col_name_2 = category + '_Arterial_' + monthid1
        df.columns = ['State',  col_name_2 ] ## col_name_1,
        df[col_name_2].replace('', np.nan, inplace=True) ## removes rows with blank records (
zonal categories)
        df['State'].replace('', np.nan, inplace=True)
        curr_monthid = str(mydate.strftime("%Y")) + str(mydate.strftime("%m")) ## 200707 for
July 2007 file
        df['data_monthid'] = curr_monthid
        df.dropna(subset=[col_name_2], inplace=True)
        df.dropna(subset=['State'], inplace=True)
        df = df[~df.State.str.contains("subtotal")] ### causes problems on joins, there in most
files
        df = df[df.State != "total"] ## causes problems on joins, is there only in specific files
        df['State'] = df.State.str.strip() ## removes leading and lagging white spaces if any
        df2 = pd.melt(df,id_vars=['State','data_monthid'],var_name=['category'],
value_name='Million_Vehicle_Miles')
        return df2
    except:
        print("error in file ",os.path.basename(file_path))


file_list = ['/Users/MrMndFkr/Desktop/linear-regression-project/Datasets/18augtvt.xls', \
             '/Users/MrMndFkr/Desktop/linear-regression-project/Datasets/18septvt.xls', \
             '/Users/MrMndFkr/Desktop/linear-regression-project/Datasets/18octtvt.xls', \
             '/Users/MrMndFkr/Desktop/linear-regression-project/Datasets/18novtvt.xls', \
             '/Users/MrMndFkr/Desktop/linear-regression-project/Datasets/18dectvt.xls', \
             '/Users/MrMndFkr/Desktop/linear-regression-project/Datasets/19jantvt.xls']

## get collated dataset
for file in file_list:
    try:
        df1 = get_arterial(file,"Rural")
        df2 = get_arterial(file,"Urban")
        df3 = get_arterial(file,"All")
        df_final = pd.concat([df_final,df1,df2,df3], axis = 0, sort = True)
```

```
    except:
        print('error encountered at ' + os.path.basename(file))

## basic QC
df_final['monthid'] = df_final.category.str[-6:]
df_final['year'] = df_final.category.str[-6:-2]
df_final.monthid.value_counts(sort = True)
qc = df_final.monthid.value_counts().sort_index()
max(qc), min(qc) ## equal value confirms all monthids have the same no of states
df_final.year.value_counts().sort_index()

# remove data for 2002
df_final['year'] = pd.to_numeric(df_final.year).astype(dtype = 'int64')
df_final.year.value_counts().sort_index()
df_final = df_final[df_final.year >= 2003]
df_final.year.value_counts().sort_index()

## get different column for category
df_final.loc[df_final.category.str.find("Rural") >= 0,'area'] = 'rural'
df_final.loc[df_final.category.str.find("Urban") >= 0,'area'] = 'urban'
df_final.loc[df_final.category.str.find("All") >= 0,'area'] = 'all'
#df_final.iloc[0:10,]

## changing variables into appropriate datatypes
df = df_final.copy()
df['Million_Vehicle_Miles'] = pd.to_numeric(df.Million_Vehicle_Miles, errors='coerce')
df['monthid'] = pd.to_numeric(df.monthid, errors='coerce')

## 12 month rolling sum - overall vehicle miles
df = df.loc[df.area == 'all']
df.sort_values(['State','monthid'], ascending = True, inplace = True)
df['rolling_miles_12m'] =
df.groupby(['State'])['Million_Vehicle_Miles'].rolling(12).sum().values

## sub regions
Northeast = list(map(lambda x:x.lower(), ['Connecticut','Maine','Massachusetts','New
Hampshire','New Jersey','New York','Pennsylvania','Rhode Island','Vermont']))
Southatlantic = list(map(lambda x:x.lower(), ['Delaware','District of
Columbia','Florida','Georgia','Maryland','North Carolina','South Carolina','Virginia','West
Virginia']))
Northcentral = list(map(lambda x:x.lower(),
['Illinois','Indiana','Iowa','Kansas','Michigan','Minnesota','Missouri','Nebraska','North
Dakota','Ohio','South Dakota','Wisconsin']))
Southgulf = list(map(lambda x:x.lower(),
['Alabama','Arkansas','Kentucky','Louisiana','Mississippi','Oklahoma','Tennessee','Texas']))
```

```
West = list(map(lambda x:x.lower(),
['Alaska','Arizona','California','Colorado','Hawaii','Idaho','Montana','Nevada','New
Mexico','Oregon','Utah','Washington','Wyoming']))
df.loc[df.State.isin(Northeast),'region'] = 'North_East'
df.loc[df.State.isin(Southatlantic),'region'] = 'South_Atlantic'
df.loc[df.State.isin(Northcentral),'region'] = 'North_Central'
df.loc[df.State.isin(Southgulf),'region'] = 'South_Gulf'
df.loc[df.State.isin(West),'region'] = 'West'

## make quarter
df['date'] = pd.to_datetime(df.monthid, format='%Y%m')
df['quarter'] = df.date.dt.quarter
df['quarter'] = df.year*10 +  df.quarter
#df.drop(columns = 'quarterid', inplace = True)

## create dataset at a quarterly level
data = df.groupby('quarter').agg({'rolling_miles_12m' : ['mean']}).reset_index()
data.columns = ['quarter','avg_miles_rollingsum_12m']
data = data[data.quarter >= 20041]

## calculate area wise percentages
a = df.groupby(['quarter','region']).agg({'rolling_miles_12m':['sum']}).reset_index()
a.columns = ['quarter','region','total']
a = a.pivot(columns = 'region',index = 'quarter', values = 'total')

b = df.groupby(['quarter']).agg({'rolling_miles_12m':['sum']}).reset_index()
b.columns = ['quarter','val']

c = pd.merge(a, b, how = 'inner', on = 'quarter')
c['perc_north_central'] = c.North_Central / c.val
c['perc_north_east'] = c.North_East / c.val
c['perc_south_atlantic'] = c.South_Atlantic / c.val
c['perc_west'] = c.West / c.val

## get them into dataset
data =
pd.merge(data,c[['quarter','perc_north_central','perc_north_east','perc_south_atlantic','per
c_west']], how = 'inner', on = 'quarter')

# urban vehicular miles
df = df_final.copy()
df['Million_Vehicle_Miles'] = pd.to_numeric(df.Million_Vehicle_Miles, errors='coerce')
df['monthid'] = pd.to_numeric(df.monthid, errors='coerce')
df = df.loc[df.area == 'urban']
df.sort_values(['State','monthid'], ascending = True, inplace = True)
df['rolling_miles_12m'] =
df.groupby(['State'])['Million_Vehicle_Miles'].rolling(12).sum().values
```

```
df['date'] = pd.to_datetime(df.monthid, format='%Y%m')
df['quarter'] = df.date.dt.quarter
df['quarter'] = df.year*10 +  df.quarter
df.sort_values(['State','monthid'], ascending = True, inplace = True)
df['rolling_miles_12m'] =
df.groupby(['State'])['Million_Vehicle_Miles'].rolling(12).sum().values
d = df.groupby('quarter').agg({'rolling_miles_12m' : ['mean']}).reset_index()
d.columns = ['quarter','urban_avg_rolling_miles_12m']
d.head()

## merge urban and get percentage
data = pd.merge(data,d, how = 'inner', on = 'quarter')
data['perc_urban'] = data.urban_avg_rolling_miles_12m / data.avg_miles_rollingsum_12m
data =
data[['quarter','avg_miles_rollingsum_12m','perc_north_central','perc_north_east','perc_so
uth_atlantic','perc_west','perc_urban']]

## save it to disk
data.to_csv("/Users/MrMndFkr/Desktop/linear-regression-project/data_reshaped.csv")
```