Given data about various cars lets predict the miles per gallon of the given vehicle

will be using linear regression, decission tree, random forest model to make our prediction.

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

import re
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

In [2]:
```python
df_mpg = pd.read_csv("auto-mpg.csv")
df_mpg.head()
```

Out[2]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

In [3]:
```python
df_mpg.info()
#Before analysing the data, we need to learn the column types of the dataset
#As we can see in the Datatypes we should change the object to the float or int
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    int64
 8   car name      398 non-null    object
dtypes: float64(3), int64(4), object(2)
memory usage: 24.9+ KB
```

In [4]:
```python
#Firstly we can check is there any null values in the dataset
df_mpg.isnull().sum()
```

Out[4]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

In [5]:
```python
#we check the horse power here because it had values as object
df_mpg[df_mpg['horsepower'].str.isnumeric()==False]
```

Out[5]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
32	25.0	4	98.0	?	2046	19.0	71	1	ford pinto
126	21.0	6	200.0	?	2875	17.0	74	1	ford maverick
330	40.9	4	85.0	?	1835	17.3	80	2	renault lecar deluxe
336	23.6	4	140.0	?	2905	14.3	80	1	ford mustang cobra
354	34.5	4	100.0	?	2320	15.8	81	2	renault 18i
374	23.0	4	151.0	?	3035	20.5	82	1	amc concord dl

In [6]:
```python
df_mpg['horsepower'] = df_mpg['horsepower'].replace('?',np.NaN)
df_mpg['horsepower'] = df_mpg['horsepower'].astype(np.float)
#If we havent change the values of horsepower to float some of the functions that we are gonna use doesn't work properly
#So we changed it to Nan then float
```

In [7]:
```python
#and now we can see the horsepower is numeric value as well
print(df_mpg.dtypes)
```

```
mpg             float64
cylinders         int64
displacement    float64
horsepower      float64
weight            int64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

In [8]:
```python
#Filling the NaN value with the mean of horsepower
df_mpg['horsepower'].fillna(df_mpg['horsepower'].mean(),inplace = True)
```

In [9]:
```python
#We are replacing car name with Brand and model.
df_mpg[['brand','model']] = df_mpg['car name'].str.split(" ", n=1, expand=True)
df_mpg.drop('car name', axis=1, inplace=True)
df_mpg.head(1)
```

Out[9]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	brand	model
0	18.0	8	307.0	130.0	3504	12.0	70	1	chevrolet	chevelle malibu

In [10]:
```python
#In brand we can see some of these are named wrong
print(sorted(df_mpg.brand.unique()))
```

```
['amc', 'audi', 'bmw', 'buick', 'cadillac', 'capri', 'chevroelt', 'chevrolet', 'chevy', 'chrysler', 'datsun', 'dodge', 'fiat', 'ford', 'hi', 'hond
a', 'maxda', 'mazda', 'mercedes', 'mercedes-benz', 'mercury', 'nissan', 'oldsmobile', 'opel', 'peugeot', 'plymouth', 'pontiac', 'renault', 'saab',
'subaru', 'toyota', 'toyouta', 'triumph', 'vokswagen', 'volkswagen', 'volvo', 'vw']
```

In [11]:
```python
#So we need to fix the brand name
wrong_brand = {
    'vokswagen' : 'volkswagen',
    'vw' : 'volkswagen',
    'toyouta' : 'toyota',
    'mercedes_benz' : 'mercedes',
    'chevroelt' : 'chevrolet',
    'maxda' : 'mazda'
}

df_mpg.brand = df_mpg.brand.map(wrong_brand).fillna(df_mpg.brand)
```

In [12]:
```python
print(sorted(df_mpg.brand.unique()))
```

```
['amc', 'audi', 'bmw', 'buick', 'cadillac', 'capri', 'chevrolet', 'chevy', 'chrysler', 'datsun', 'dodge', 'fiat', 'ford', 'hi', 'honda', 'mazda',
'mercedes', 'mercedes-benz', 'mercury', 'nissan', 'oldsmobile', 'opel', 'peugeot', 'plymouth', 'pontiac', 'renault', 'saab', 'subaru', 'toyota', 't
riumph', 'volkswagen', 'volvo']
```

In [13]:
```python
#In this section we can see the top 10 most efficient cars
#We can make a few assumption with these 4 cylinders means more efficient
#The cars that came out 1980 has better fuel consumption
#Volkswagen was better at creating cars with less fuel consumption
df_mpg.nlargest(10,'mpg')
```

Out[13]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	brand	model
322	46.6	4	86.0	65.000000	2110	17.9	80	3	mazda	glc
329	44.6	4	91.0	67.000000	1850	13.8	80	3	honda	civic 1500 gl
325	44.3	4	90.0	48.000000	2085	21.7	80	2	volkswagen	rabbit c (diesel)
394	44.0	4	97.0	52.000000	2130	24.6	82	2	volkswagen	pickup
326	43.4	4	90.0	48.000000	2335	23.7	80	2	volkswagen	dasher (diesel)
244	43.1	4	90.0	48.000000	1985	21.5	78	2	volkswagen	rabbit custom diesel
309	41.5	4	98.0	76.000000	2144	14.7	80	2	volkswagen	rabbit
330	40.9	4	85.0	104.469388	1835	17.3	80	2	renault	lecar deluxe
324	40.8	4	85.0	65.000000	2110	19.2	80	3	datsun	210
247	39.4	4	85.0	70.000000	2070	18.6	78	3	datsun	b210 gx

In [14]:
```python
#Describe the data to get an idea about the dataset

df_mpg.describe()
```

Out[14]:

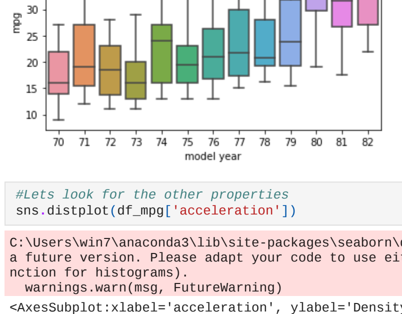
	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050	1.572864
std	7.815984	1.701004	104.269838	38.199187	846.841774	2.757689	3.697627	0.802055
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	104.250000	76.000000	2223.750000	13.825000	73.000000	1.000000
50%	23.000000	4.000000	148.500000	95.000000	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	125.000000	3608.000000	17.175000	79.000000	2.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	3.000000

In [15]:
```python
#As we can see from the histogram of mpg, the data is moderately skewed to the right
#this implies that the there are more numbers of cars which have low mpg than those with high mpg.

sns.distplot(df_mpg['mpg'])
```

```
C:\Users\win7\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level fu
nction for histograms).
  warnings.warn(msg, FutureWarning)
```

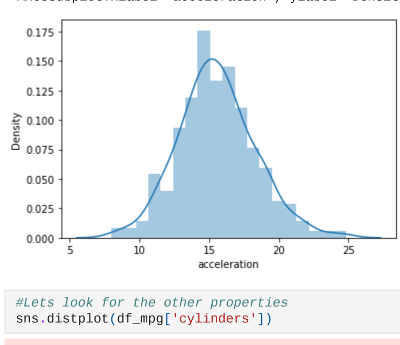
Out[15]: <AxesSubplot:xlabel='mpg', ylabel='Density'>



In [16]:
```python
#lets see the mpg effeciency over the years

sns.boxplot(x = 'model year', y = 'mpg', data = df_mpg)
```

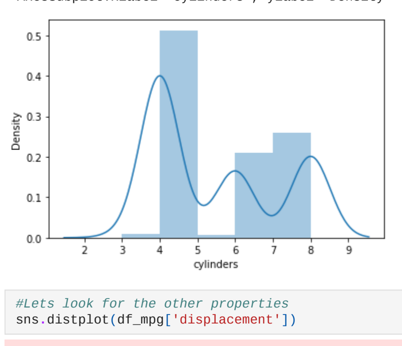
Out[16]: <AxesSubplot:xlabel='model year', ylabel='mpg'>



In [17]:
```python
#Lets look for the other properties
sns.distplot(df_mpg['acceleration'])
```

```
C:\Users\win7\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level fu
nction for histograms).
  warnings.warn(msg, FutureWarning)
```

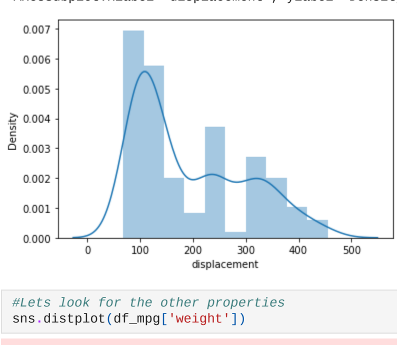
Out[17]: <AxesSubplot:xlabel='acceleration', ylabel='Density'>



In [18]:
```python
#Lets look for the other properties
sns.distplot(df_mpg['cylinders'])
```

```
C:\Users\win7\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level fu
nction for histograms).
  warnings.warn(msg, FutureWarning)
```

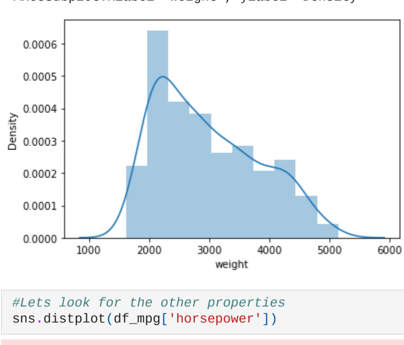
Out[18]: <AxesSubplot:xlabel='cylinders', ylabel='Density'>



In [19]:
```python
#Lets look for the other properties
sns.distplot(df_mpg['displacement'])
```

```
C:\Users\win7\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level fu
nction for histograms).
  warnings.warn(msg, FutureWarning)
```

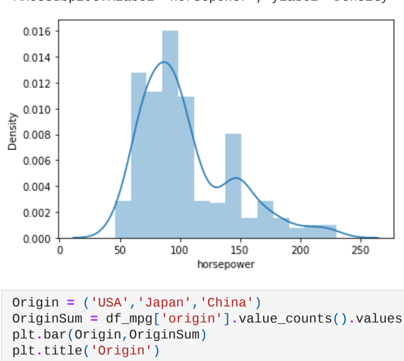
Out[19]: <AxesSubplot:xlabel='displacement', ylabel='Density'>



In [20]:
```python
#Lets look for the other properties
sns.distplot(df_mpg['weight'])
```

```
C:\Users\win7\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level fu
nction for histograms).
  warnings.warn(msg, FutureWarning)
```

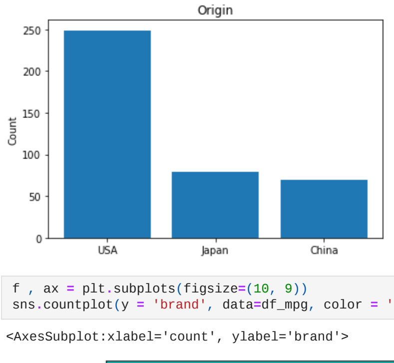
Out[20]: <AxesSubplot:xlabel='weight', ylabel='Density'>



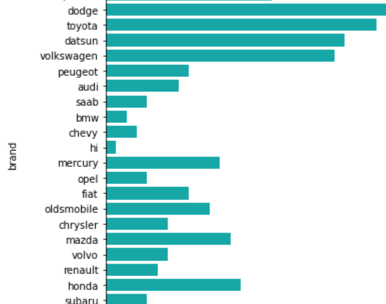
In [21]:
```python
#Lets look for the other properties
sns.distplot(df_mpg['horsepower'])
```

```
C:\Users\win7\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level fu
nction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[21]: <AxesSubplot:xlabel='horsepower', ylabel='Density'>

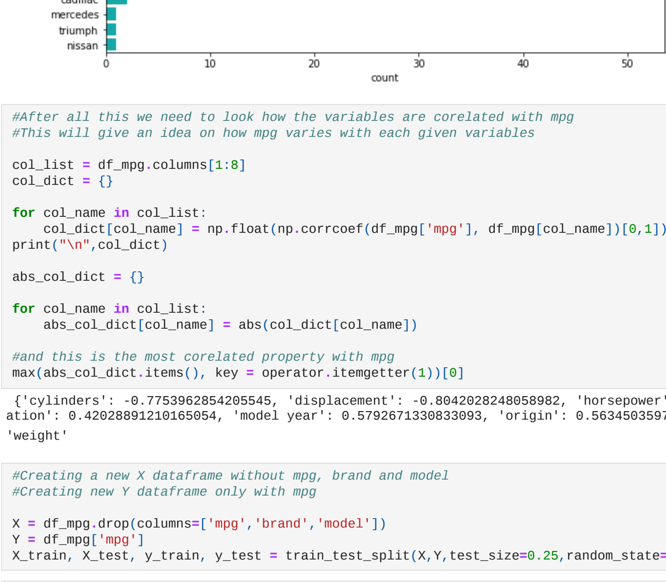


In [22]:
```python
Origin = ('USA','Japan','China')
OriginSum = df_mpg['origin'].value_counts().values
plt.bar(Origin,OriginSum)
plt.title('Origin')
plt.ylabel('Count')
plt.show()
```



In [23]:
```python
f , ax = plt.subplots(figsize=(10, 9))
sns.countplot(y = 'brand', data=df_mpg, color = 'c')
```

Out[23]: <AxesSubplot:xlabel='count', ylabel='brand'>



In [24]:
```python
#After all this we need to look how the variables are corelated with mpg
#This will give an idea on how mpg varies with each given variables

col_list = df_mpg.columns[1:8]
col_dict = {}

for col_name in col_list:
    col_dict[col_name] = np.float(np.corrcoef(df_mpg['mpg'], df_mpg[col_name])[0,1])
print("\n",col_dict)

abs_col_dict = {}

for col_name in col_list:
    abs_col_dict[col_name] = abs(col_dict[col_name])

#and this is the most corelated property with mpg
max(abs_col_dict.items(), key = operator.itemgetter(1))[0]
```

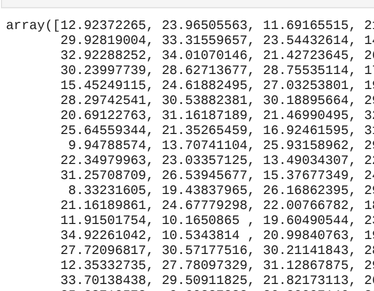
```
 {'cylinders': -0.7753962854205545, 'displacement': -0.8042028248058982, 'horsepower': -0.7714371350025525, 'weight': -0.8317409332443345, 'acceler
ation': 0.42028891210165054, 'model year': 0.5792671330833093, 'origin': 0.5634503597738432}
```

Out[24]: 'weight'

In [25]:
```python
#Creating a new X dataframe without mpg, brand and model
#Creating new Y dataframe only with mpg

X = df_mpg.drop(columns=['mpg','brand','model'])
Y = df_mpg['mpg']
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.25,random_state=0)
```

In [26]:
```python
model=LinearRegression()
model.fit(X_train,y_train)
LinearReg=model.predict(X_train)
LinearReg=LinearReg.flatten()
fig,ax=plt.subplots()
ax.scatter(LinearReg,y_train)
plt.plot(np.arange(0,45),np.arange(0,45),color='green')
plt.title('Linear Regression')
plt.show()
```



In [27]:
```python
#Using Linear regression model

linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

linear_r2 = linear_model.score(X_test, y_test)
print('Linear Regression accuracy:{:.5f}'.format(linear_r2))
```

```
Linear Regression accuracy:0.81346
```

In [28]:
```python
predictions=model.predict(X_test)
predictions
```

Out[28]:
```
array([12.92372265, 23.96505563, 11.69165515, 21.0938141 , 17.37956039,
       29.92819004, 33.31559657, 23.54432614, 14.22778927, 26.42032754,
       32.92288252, 34.01070146, 21.42723645, 26.81613499, 16.16898361,
       30.23997739, 28.62713677, 28.75535114, 17.43655388, 30.60585406,
       15.45249115, 24.61882495, 27.03253801, 19.89133655, 29.16656011,
       28.29742541, 30.53882381, 30.18895664, 29.05770776, 18.20363647,
       20.69122763, 31.16187189, 21.46990495, 32.22486329, 23.79224245,
       25.64559344, 21.35265459, 16.92461595, 31.71565227,  8.71275881,
        9.94788574, 13.70741104, 25.93158962, 29.86619781, 31.36247232,
       22.34979963, 23.03357125, 13.49034307, 22.1046017 , 27.93806199,
       31.25708709, 26.53945677, 15.37677349, 24.85345291, 14.84249433,
        8.33231605, 19.43837965, 26.16862395, 29.91615796, 14.60535057,
       21.16189861, 24.67779298, 22.00766782, 18.98821152, 10.57385387,
       11.91501754, 10.1650865 , 19.60490544, 23.90821299,  9.93054337,
       34.92261042, 10.5343814 , 20.99840763, 19.01557377, 23.96860455,
       27.72096817, 30.57177516, 30.21141843, 28.35750451, 15.70604926,
       12.35332735, 27.78097329, 31.12867875, 29.20020761, 31.65217545,
       33.70138438, 29.50911825, 21.82173113, 26.74385777, 31.65723715,
       25.33719578,  9.60337283, 26.20097146, 32.05360537, 27.41492429,
       20.28716766, 19.74866665, 25.12546132, 23.86339537, 11.63020133])
```

In [29]:
```python
df = pd.DataFrame({'Actual':y_test,'Predicted':predictions})
df
```

Out[29]:

	Actual	Predicted
65	14.0	12.923723
132	25.0	23.965056
74	13.0	11.691655
78	21.0	21.093814
37	18.0	17.379560
...	...	...
286	17.6	20.287168
263	17.7	19.748667
146	28.0	25.125461
259	20.8	23.863395
63	14.0	11.630201

100 rows × 2 columns

In [30]:
```python
#Using Decision Tree model

tree_model = DecisionTreeRegressor()
tree_model.fit(X_train, y_train)

tree_r2 = tree_model.score(X_test, y_test)
print('Decision Tree accuracy:{:.5f}'.format(tree_r2))
```

```
Decision Tree accuracy:0.84077
```

In [31]:
```python
#Using Random forest regression model

rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)

rf_r2 = rf_model.score(X_test, y_test)
print('Random Forest accuracy:{:.5f}'.format(rf_r2))
```

```
Random Forest accuracy:0.90052
```

After we have analyzed the data, we can see that there is a pattern with this data. Usally most of the cars have 4 cylinders and they are top tier with mpg. Weight of a car has the most impact on mpg. People tend to buy there cars from brand Ford. The most efficient year was 1980 and after that with the data we have, we can see that it decreases next 2 years.