Part 1

In [1]:
```
'''
1. Identify each column as nominal, ordinal, interval, or ratio in the A
uto_mpg_raw.csv data set.
a.      Miles per gallon: ratio
b.      Cylinders: ratio
c.      Displacement: nominal
d.      Horsepower: ratio
e.      Weight: ratio
f.      Acceleration: nominal
g.      Model Year: interval
h.      Origin: nominal
i.      Car Name: nominal

'''
```

Out[1]:
```
'\n1. Identify each column as nominal, ordinal, interval, or ratio in t
he Auto_mpg_raw.csv data set.\na.\tMiles per gallon: ratio\nb.\tCylinde
rs: ratio\nc.\tDisplacement: nominal\nd.\tHorsepower: ratio\ne.\tWeigh
t: ratio\nf.\tAcceleration: nominal\ng.\tModel Year: interval\nh.\tOrig
in: nominal\ni.\tCar Name: nominal\n\n'
```

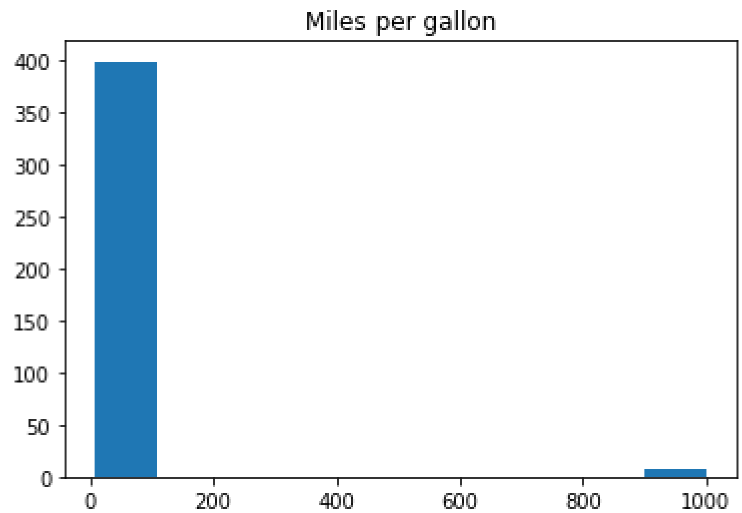
Part 2

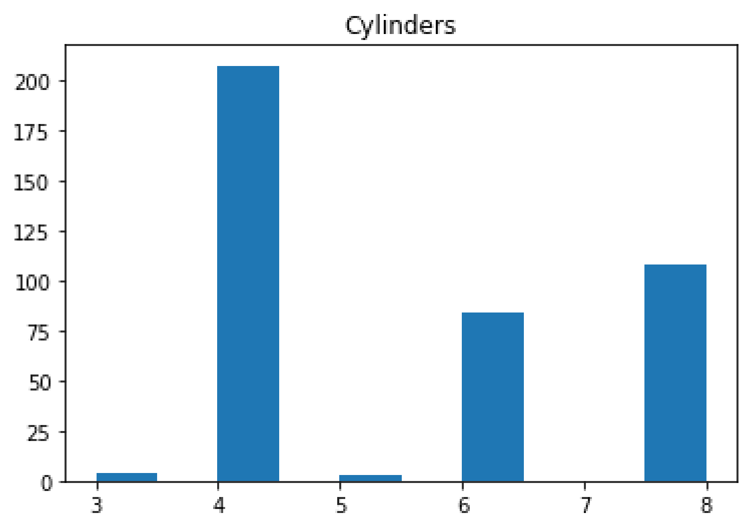
In [2]:
```
# Importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
pd.set_option('display.max_rows', 500)

# importing the Auto_mpg_raw.csv file using pandas
auto_Mpg = pd.read_csv("Auto_mpg_raw.csv")


# Plotting all the histograms
# histogram for miles per gallon
plt.figure()
plt.hist(auto_Mpg["Miles per gallon"])
plt.title("Miles per gallon")

#histogram for Cylinders
plt.figure()
plt.hist(auto_Mpg["Cylinders"])
plt.title("Cylinders")

#histogram for Displacement
plt.figure()
plt.hist(auto_Mpg["Displacement"])
plt.title("Displacement")

# histogram for Horsepower
plt.figure()
plt.hist(auto_Mpg["Horsepower"])
plt.title("Horsepower")

#histogram for Weight
plt.figure()
plt.hist(auto_Mpg["Weight"])
plt.title("Weight")

#histogram for Acceleration
plt.figure()
plt.hist(auto_Mpg["Acceleration"])
plt.title("Acceleration")

#histogram for Model Year
plt.figure()
plt.hist(auto_Mpg["Model year"])
plt.title("Model year")

#histogram for origin
plt.figure()
plt.hist(auto_Mpg["Origin"])
plt.title("Origin")
```

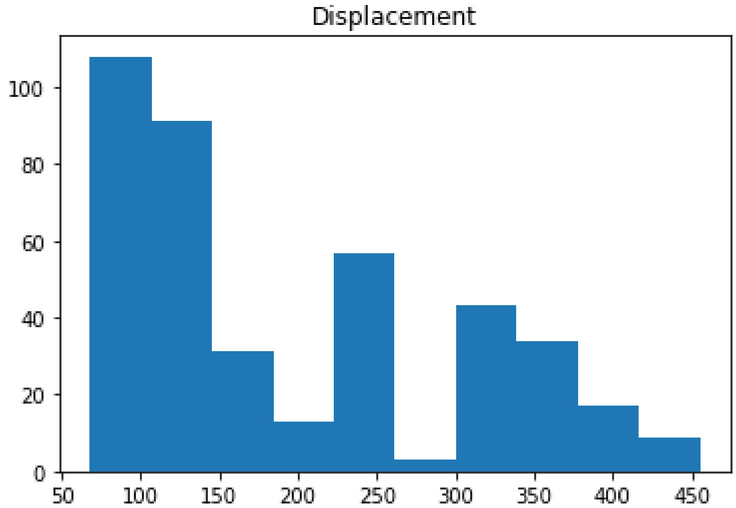
```
""" Answer to question number 2
 Here from seeing the distribution in the histograms, some misleading da
tapoints are observed.
    The misleading data points are discussed below:
    1. The first attribute Miles_per_gallon has an extreme value 1000 th
at is an outlier in the distribution
        and definately is a misleading data that needs some adjustment.
 Summary of Miles per gallon:

count     406.000000
mean       42.755665
std       136.102120
min         9.000000
25%        17.500000
50%        23.000000
75%        29.800000
max      1000.000000
Name: Miles per gallon, dtype: float64




    2. The second attribute that requires attention is the "Horsepower".
Since horsepower is a ratio,
        0 horsepower means absence of horsepower that does not make sens
e in the real world. Therefore,
        0's need some other relevant values.

count    405.000000
mean     103.785185
std       40.241668
min        0.000000
25%       75.000000
50%       94.000000
75%      129.000000
max      230.000000
Name: Horsepower, dtype: float64


    Other than those two attributes, the rest of the attributes does not
have misleading datapoints.
        """
```

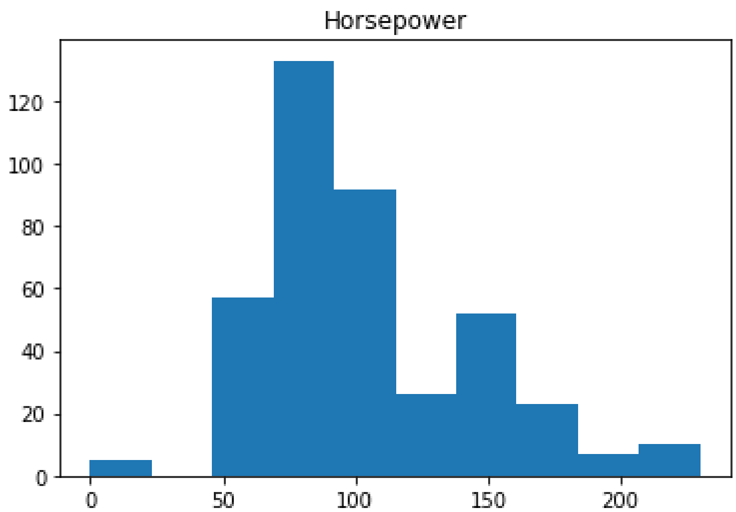
```
/Library/Python/3.7/site-packages/numpy/lib/histograms.py:839: RuntimeW
arning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
/Library/Python/3.7/site-packages/numpy/lib/histograms.py:840: RuntimeW
arning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```

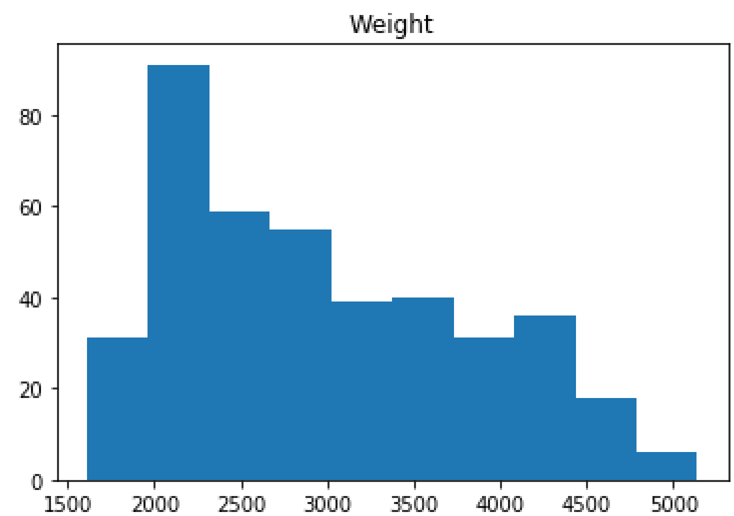
Out[2]:
```
' Answer to question number 2\n Here from seeing the distribution in th
e histograms, some misleading datapoints are observed.\n    The mislead
ing data points are discussed below:\n    1. The first attribute Miles_
per_gallon has an extreme value 1000 that is an outlier in the distribu
tion\n        and definately is a misleading data that needs some adjus
tment. Summary of Miles per gallon:\n        \ncount     406.000000\nme
an       42.755665\nstd       136.102120\nmin         9.000000\n25%    
17.500000\n50%        23.000000\n75%        29.800000\nmax      1000.00
0000\nName: Miles per gallon, dtype: float64\n\n\n\n\n    2. The second
attribute that requires attention is the "Horsepower". Since horsepower
is a ratio,\n        0 horsepower means absence of horsepower that does
not make sense in the real world. Therefore, \n        0\'s need some o
ther relevant values.\n        \ncount    405.000000\nmean     103.7851
85\nstd       40.241668\nmin        0.000000\n25%       75.000000\n50%  
94.000000\n75%      129.000000\nmax      230.000000\nName: Horsepower,
dtype: float64\n\n    \n    Other than those two attributes, the rest o
f the attributes does not have misleading datapoints. \n        '
```

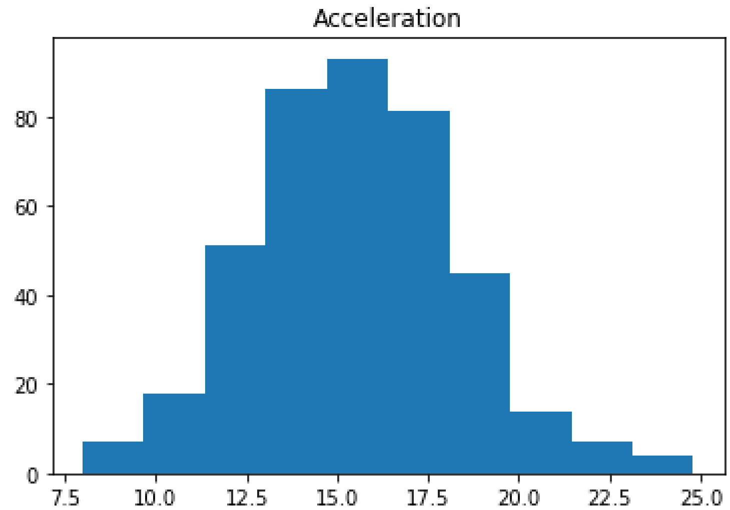


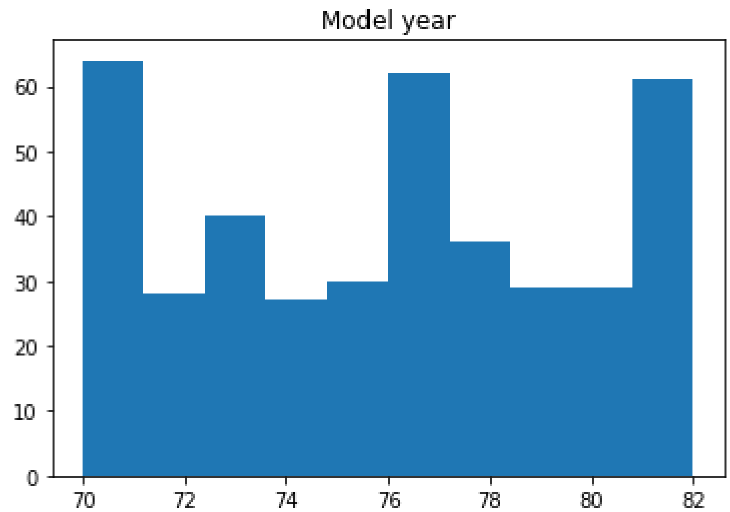


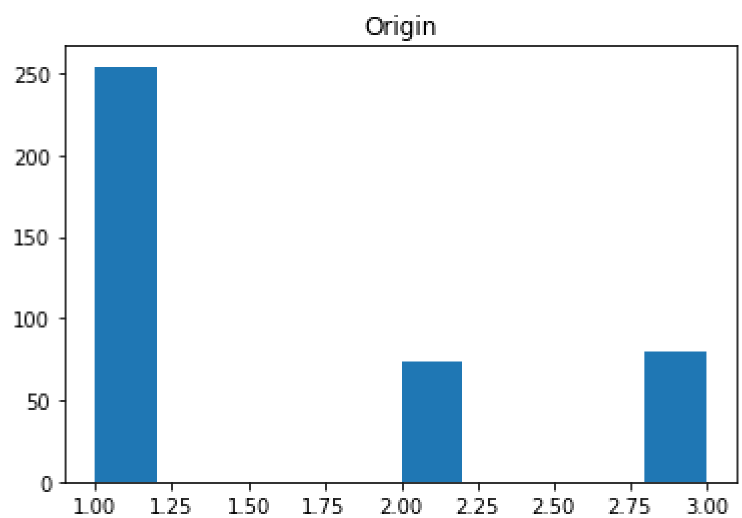

Displacement
0
20
40
60
80
100
50
100
150
200
250
300
350
400
450


Horsepower
0
20
40
60
80
100
120
0
50
100
150
200


Weight
0
20
40
60
80
1500
2000
2500
3000
3500
4000
4500
5000


Acceleration
0
20
40
60
80
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0


Model year
0
10
20
30
40
50
60
70
72
74
76
78
80
82


Origin
0
50
100
150
200
250
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00

In [3]:
```
# Lets change all the values equal to 1000 for miles_Per_gallon to NaN
auto_Mpg["Miles per gallon"] = auto_Mpg["Miles per gallon"].replace({100
0:np.NaN})
print(f" The new distribution of Miles Per Gallon is: ")
auto_Mpg["Miles per gallon"].describe()
```

```
 The new distribution of Miles Per Gallon is:
```

Out[3]:
```
count    398.000000
mean      23.514573
std        7.815984
min        9.000000
25%       17.500000
50%       23.000000
75%       29.000000
max       46.600000
Name: Miles per gallon, dtype: float64
```

In [4]:
```
# Again, lets change all the values equal to 0 for horsepower to NaN
auto_Mpg["Horsepower"] = auto_Mpg["Horsepower"].replace({0:np.NaN})
print(f" The new distribution of Horsepower is: ")
auto_Mpg["Horsepower"].describe()
```

```
 The new distribution of Horsepower is:
```

Out[4]:
```
count    400.000000
mean     105.082500
std       38.768779
min       46.000000
25%       75.750000
50%       95.000000
75%      130.000000
max      230.000000
Name: Horsepower, dtype: float64
```

In [5]:
```
# Now lets create a table with only three attributes (miles_per_gallon,
 Cylinders, Horsepower)
new_Table = auto_Mpg[["Miles per gallon","Cylinders","Horsepower"]]
new_Table.describe()
```

Out[5]:

	Miles per gallon	Cylinders	Horsepower
count	398.000000	406.000000	400.000000
mean	23.514573	5.475369	105.082500
std	7.815984	1.712160	38.768779
min	9.000000	3.000000	46.000000
25%	17.500000	4.000000	75.750000
50%	23.000000	4.000000	95.000000
75%	29.000000	8.000000	130.000000
max	46.600000	8.000000	230.000000

In [ ]:

In [6]:

```
# Now lets find an appropriate value to replace the NaN's in the miles_p
er_gallon column
""" One approach is to find cylinders for each cars is to find a mean to
replace those NaNs.
    We are going to take this approach to fill in the blanks or NaNs."""



# All rows with NaN mileage

new_Table[new_Table['Miles per gallon'].isnull()]
```

Out[6]:

	Miles per gallon	Cylinders	Horsepower
10	NaN	4.0	115.0
11	NaN	8.0	165.0
12	NaN	8.0	153.0
13	NaN	8.0	175.0
14	NaN	8.0	175.0
17	NaN	8.0	140.0
39	NaN	4.0	48.0
367	NaN	4.0	110.0

In [7]:

```
# All rows with NaN horsepowers
new_Table[new_Table['Horsepower'].isnull()]
```

Out[7]:

	Miles per gallon	Cylinders	Horsepower
38	25.0	4.0	NaN
133	21.0	6.0	NaN
337	40.9	4.0	NaN
343	23.6	4.0	NaN
361	34.5	4.0	NaN
382	23.0	4.0	NaN

In [8]:
```
# Calculating the mean mileage for different cars
def meanOfMileage(numOfCylinders):
    num_Cylinder_Cars = new_Table[new_Table.Cylinders == numOfCylinders]
    meanOfMileage = (num_Cylinder_Cars['Miles per gallon']).mean()
    return meanOfMileage


# Function to calculate the mean of horsepowers for different cars
def meanOfHorsepower(numOfCylinders):
    num_Cylinder_Cars = new_Table[new_Table.Cylinders == numOfCylinders]
    meanOfHorsepower = (num_Cylinder_Cars['Horsepower']).mean()
    return meanOfHorsepower


# Function to update the horsepowers
def horsepowerUpdater(numOfCylinders):
    num_Cylinder_Cars = new_Table[new_Table.Cylinders == numOfCylinders]
    NaNHorsepower = num_Cylinder_Cars[num_Cylinder_Cars['Horsepower'].is
null()]
    NaNHorsepower['Horsepower']= meanOfHorsepower(numOfCylinders)
    return NaNHorsepower


#function to update the mileage
def mileageUpdater(numOfCylinders):
    num_Cylinder_Cars = new_Table[new_Table.Cylinders == numOfCylinders]
    NaNMileage = num_Cylinder_Cars[num_Cylinder_Cars['Miles per gallon']
.isnull()]
    NaNMileage['Miles per gallon']= meanOfMileage(numOfCylinders)
    return NaNMileage

updatedHorsepower4 = horsepowerUpdater(4)
updatedHorsepower6 = horsepowerUpdater(6)
updatedHorsepower8 = horsepowerUpdater(8)
updatedHorsepower = pd.concat([updatedHorsepower4,updatedHorsepower6,upd
atedHorsepower8])
updatedHorsepower
```

```
/Library/Python/3.7/site-packages/ipykernel_launcher.py:19: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[8]:

	Miles per gallon	Cylinders	Horsepower
38	25.0	4.0	78.470297
337	40.9	4.0	78.470297
343	23.6	4.0	78.470297
361	34.5	4.0	78.470297
382	23.0	4.0	78.470297
133	21.0	6.0	101.506024

In [9]:

```
updatedMileage4 = mileageUpdater(4)
updatedMileage6 = mileageUpdater(6)
updatedMileage8 = mileageUpdater(8)
updatedMileage = pd.concat([updatedMileage4,updatedMileage6,updatedMilea
ge8])
updatedMileage
```

```
/Library/Python/3.7/site-packages/ipykernel_launcher.py:27: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[9]:

	Miles per gallon	Cylinders	Horsepower
10	29.286765	4.0	115.0
39	29.286765	4.0	48.0
367	29.286765	4.0	110.0
11	14.963107	8.0	165.0
12	14.963107	8.0	153.0
13	14.963107	8.0	175.0
14	14.963107	8.0	175.0
17	14.963107	8.0	140.0

In [10]:
```
# Add two dataframes together
updatedVals = pd.concat([updatedMileage,updatedHorsepower])
print("The dataframe with updated values are: ")
updatedVals
```

```
The dataframe with updated values are:
```

Out[10]:

	Miles per gallon	Cylinders	Horsepower
10	29.286765	4.0	115.000000
39	29.286765	4.0	48.000000
367	29.286765	4.0	110.000000
11	14.963107	8.0	165.000000
12	14.963107	8.0	153.000000
13	14.963107	8.0	175.000000
14	14.963107	8.0	175.000000
17	14.963107	8.0	140.000000
38	25.000000	4.0	78.470297
337	40.900000	4.0	78.470297
343	23.600000	4.0	78.470297
361	34.500000	4.0	78.470297
382	23.000000	4.0	78.470297
133	21.000000	6.0	101.506024

In [11]:
```
# Updating the original dataset with the interpolated values
auto_Mpg.update(updatedVals, join='left', overwrite=True, filter_func=None, errors='ignore')
print("The new dataset is: ")
auto_Mpg.head()
```

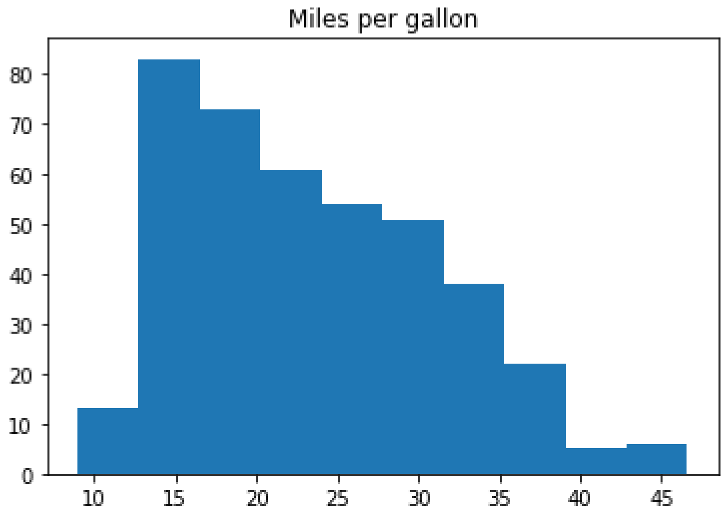
The new dataset is:

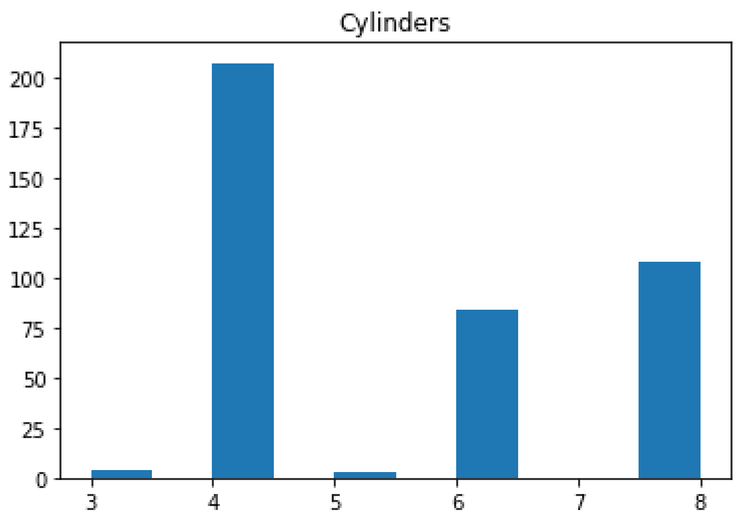
Out[11]:

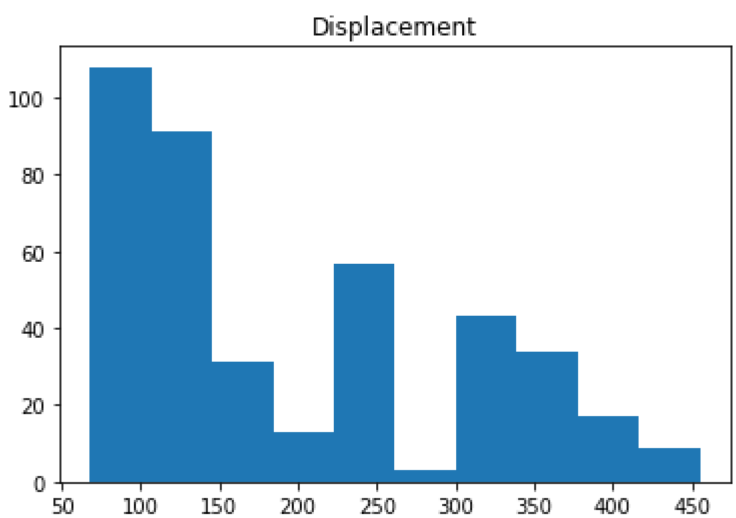
	Miles per gallon	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model year	Origin	Car name
0	18.0	8.0	307.0	130.0	3504.0	12.0	70.0	1.0	chevrolet chevelle malibu
1	15.0	8.0	350.0	165.0	3693.0	11.5	70.0	1.0	buick skylark 320
2	18.0	8.0	318.0	150.0	3436.0	11.0	70.0	1.0	plymouth satellite
3	16.0	8.0	304.0	150.0	3433.0	12.0	70.0	1.0	amc rebel sst
4	17.0	8.0	302.0	140.0	3449.0	10.5	70.0	1.0	ford torino

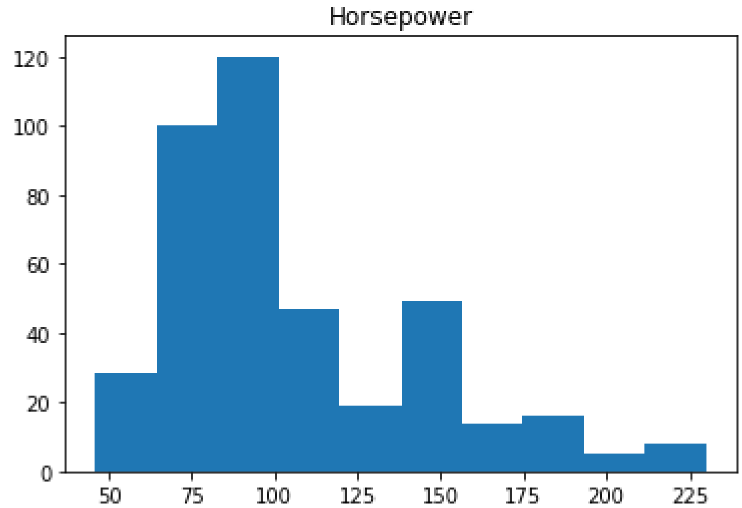
In [12]:
```
# Plotting histograms with cleaned dataset
# Plotting all the histograms
# histogram for miles per gallon
plt.figure()
plt.hist(auto_Mpg["Miles per gallon"])
plt.title("Miles per gallon")

#histogram for Cylinders
plt.figure()
plt.hist(auto_Mpg["Cylinders"])
plt.title("Cylinders")

#histogram for Displacement
plt.figure()
plt.hist(auto_Mpg["Displacement"])
plt.title("Displacement")

# histogram for Horsepower
plt.figure()
plt.hist(auto_Mpg["Horsepower"])
plt.title("Horsepower")

#histogram for Weight
plt.figure()
plt.hist(auto_Mpg["Weight"])
plt.title("Weight")

#histogram for Acceleration
plt.figure()
plt.hist(auto_Mpg["Acceleration"])
plt.title("Acceleration")

#histogram for Model Year
plt.figure()
plt.hist(auto_Mpg["Model year"])
plt.title("Model year")

#histogram for origin
plt.figure()
plt.hist(auto_Mpg["Origin"])
plt.title("Origin")
```

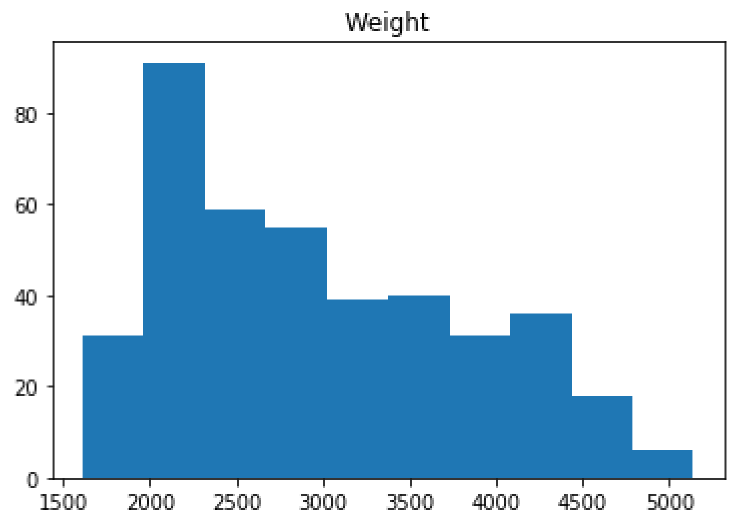
```
Out[12]: Text(0.5, 1.0, 'Origin')
```

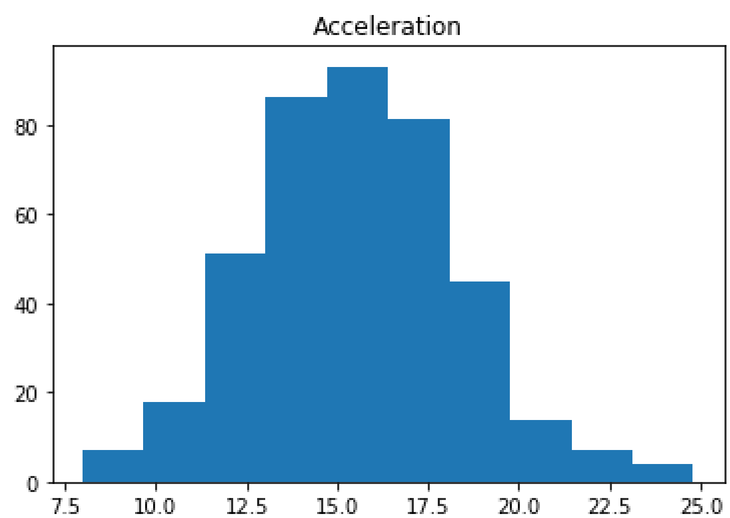


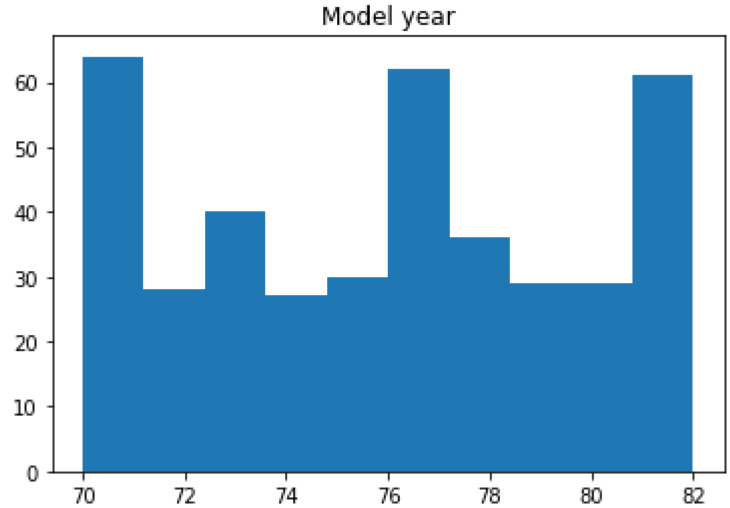


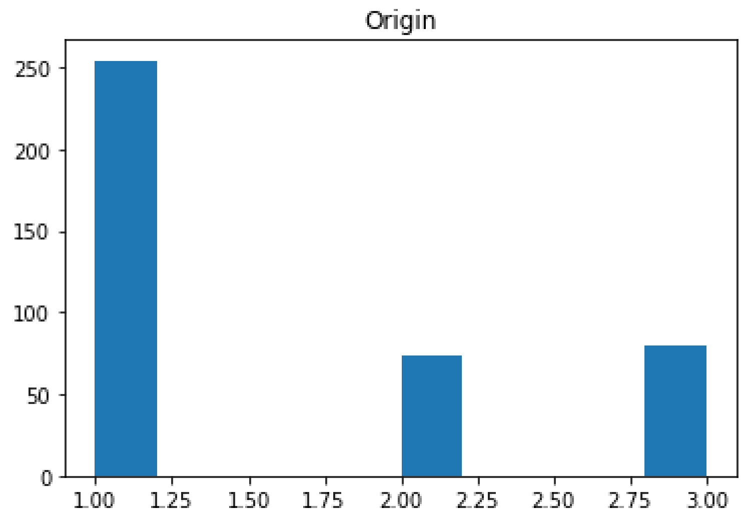



Horsepower
0
20
40
60
80
100
120
50
75
100
125
150
175
200
225


Weight
0
20
40
60
80
1500
2000
2500
3000
3500
4000
4500
5000


Acceleration
0
20
40
60
80
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0





In [13]:
```
# Exporting the file as a .csv to local folder
auto_Mpg.to_csv("Auto_mpg_adjust.csv",index = False)
```

Part 3

In [14]:
```
# Problem 3
""" Using z values to check for outliers in the dataset """
from scipy import stats
auto_Mpg['milesPerGallon_Z'] = stats.zscore(auto_Mpg["Miles per gallon"
])
# print(auto_Mpg['Miles per gallon_Z'])

# Looking for outliers in the dataset using z values
MPGOutliers = auto_Mpg.query('milesPerGallon_Z >3 | milesPerGallon_Z <-
3')
numberOfMPGOutliers = len(MPGOutliers)
print(f'There are %d outliers present in MPG data.' % numberOfMPGOutlier
s)
```

```
There are 0 outliers present in MPG data.
```

In [15]:
```python
# Looking into Cylinder data
auto_Mpg['cylinders_Z'] = stats.zscore(auto_Mpg["Cylinders"])
# print(auto_Mpg['cylinders_Z'])
cylindersOutliers = auto_Mpg.query('cylinders_Z >3 | cylinders_Z <-3')
numberOfCylindersOutliers = len(cylindersOutliers)
print(f'There are %d outliers present in Cylinders data.' % numberOfCyli
ndersOutliers)
```

```
There are 0 outliers present in Cylinders data.
```

In [16]:
```python
# Looking into Displacement data
auto_Mpg['displacement_Z'] = stats.zscore(auto_Mpg["Displacement"])
# print(auto_Mpg['displacement_Z'])
displacementOutliers = auto_Mpg.query('displacement_Z >3 | displacement_
Z <-3')
numberOfDisplacementOutliers = len(displacementOutliers)
print(f'There are %d outliers present in Displacements data.' % numberOf
DisplacementOutliers)
```

```
There are 0 outliers present in Displacements data.
```

In [17]:
```python
# Looking into Horsepower data
auto_Mpg['horsepower_Z'] = stats.zscore(auto_Mpg["Horsepower"])
# print(auto_Mpg['horsepower_Z'])
horsepowerOutliers = auto_Mpg.query('horsepower_Z >3 | horsepower_Z <-3'
)
numberOfHorsepowerOutliers = len(horsepowerOutliers)
print(f'There are %d outliers present in Horsepower data.' % numberOfHor
sepowerOutliers)
horsepowerSort = auto_Mpg.sort_values('horsepower_Z',ascending = False)
horsepowerSort.head(n = 4)
```

```
There are 4 outliers present in Horsepower data.
```

Out[17]:

	Miles per gallon	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model year	Origin	Car name
123	16.0	8.0	400.0	230.0	4278.0	9.5	73.0	1.0	pontiac grand prix
102	12.0	8.0	455.0	225.0	4951.0	11.0	73.0	1.0	buick electra 225 custom
19	14.0	8.0	455.0	225.0	3086.0	10.0	70.0	1.0	buick estate wagon (sw)
8	14.0	8.0	455.0	225.0	4425.0	10.0	70.0	1.0	pontiac catalina

In [18]:
```
# Looking into Weight data
auto_Mpg['weight_Z'] = stats.zscore(auto_Mpg["Weight"])
# print(auto_Mpg['weight_Z'])
weightOutliers = auto_Mpg.query('weight_Z >3 | weight_Z <-3')
numberOfWeightOutliers = len(weightOutliers)
print(f'There are %d outliers present in Weight data.' % numberOfWeightO
utliers)
```

```
There are 0 outliers present in Weight data.
```

In [19]:
```
# Looking into Acceleration data
auto_Mpg['acceleration_Z'] = stats.zscore(auto_Mpg["Acceleration"])
# print(auto_Mpg['acceleration_Z'])
accelerationOutliers = auto_Mpg.query('acceleration_Z >3 | acceleration_
Z <-3')
numberOfAccelerationOutliers = len(accelerationOutliers)
print(f'There are %d outliers present in Acceleration data.' % numberOfA
ccelerationOutliers)
accelerationSort = auto_Mpg.sort_values('acceleration_Z',ascending = Fal
se)
accelerationSort.head(n = 2)
```

```
There are 2 outliers present in Acceleration data.
```

Out[19]:

	Miles per gallon	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model year	Origin	Car name
306	27.2	4.0	141.0	71.0	3190.0	24.8	79.0	2.0	peugeot 504
402	44.0	4.0	97.0	52.0	2130.0	24.6	82.0	2.0	vw pickup

In [20]:
```
# Looking into modelYear data
auto_Mpg['modelYear_Z'] = stats.zscore(auto_Mpg["Model year"])
# print(auto_Mpg['modelYear_Z'])
modelYearOutliers = auto_Mpg.query('modelYear_Z >3 | modelYear_Z <-3')
numberOfModelYearOutliers = len(modelYearOutliers)
print(f'There are %d outliers present in Model year data.' % numberOfMod
elYearOutliers)
```

```
There are 0 outliers present in Model year data.
```

In [21]:
```
# Looking into Origin data
auto_Mpg['origin_Z'] = stats.zscore(auto_Mpg["Origin"])
# print(auto_Mpg['origin_Z'])
originOutliers = auto_Mpg.query('origin_Z >3 | origin_Z <-3')
numberOfOriginOutliers = len(originOutliers)
print(f'There are %d outliers present in Origin data.' % numberOfOriginO
utliers)
```

```
There are 0 outliers present in Origin data.
```

Part 4

In [22]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
pd.set_option('display.max_rows', 500)


cereals = pd.read_csv("cereals.csv")
cereals.head()
```

Out[22]:

	Name	Manuf	Type	Calories	Protein	Fat	Sodium	Fiber	Carbo	Sugars	...
0	100%_Bran	N	C	70	4	1	130	10.0	5.0	6.0	...
1	100%_Natural_Bran	Q	C	120	3	5	15	2.0	8.0	8.0	...
2	All-Bran	K	C	70	4	1	260	9.0	7.0	5.0	...
3	All-Bran_with_Extra_Fiber	K	C	50	4	0	140	14.0	8.0	0.0	...
4	Almond_Delight	R	C	110	2	2	200	1.0	14.0	8.0	...

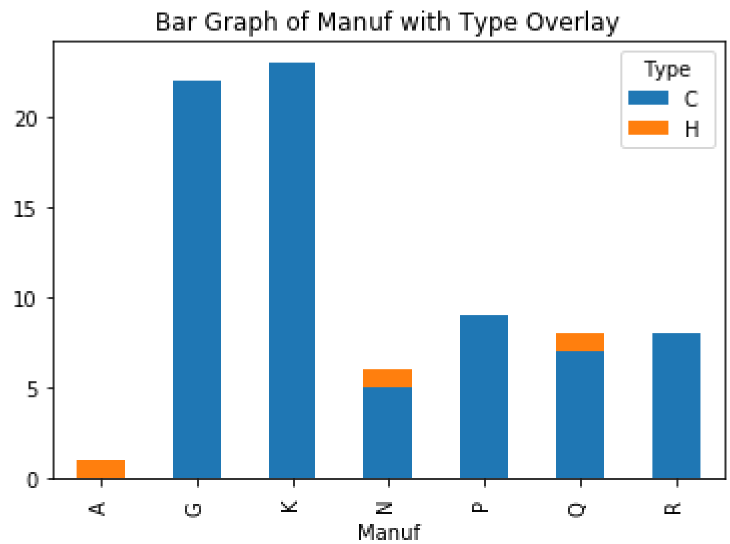
5 rows × 23 columns

Part a

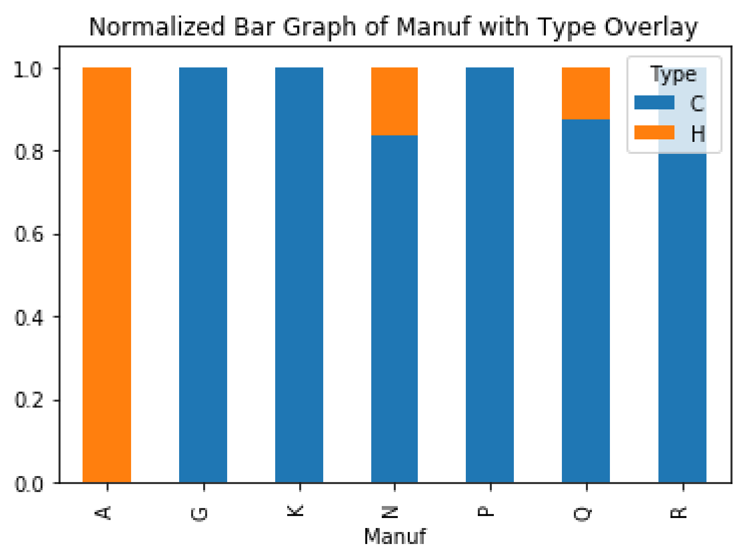
In [23]:
```
# create a bar graph and normalized bar graph of the “Manuf” variable wi
th “Type” overlay
crosstab_01 = pd.crosstab(cereals["Manuf"],cereals['Type'])
plt.figure()
crosstab_01.plot(kind='bar', stacked=True)
plt.title('Bar Graph of Manuf with Type Overlay')

# Normalized bar graph
plt.figure()
crosstab_norm = crosstab_01.div(crosstab_01.sum(1),axis=0)
crosstab_norm.plot(kind='bar', stacked=True)
plt.title('Normalized Bar Graph of Manuf with Type Overlay')
```

Out[23]:
```
Text(0.5, 1.0, 'Normalized Bar Graph of Manuf with Type Overlay')
```

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




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



Part b

In [24]:
```
# create a contingency table of "Manuf" and "Type"

crosstab_02 = pd.crosstab(cereals["Type"],cereals['Manuf'])
crosstab_norm_02 = round(crosstab_02.div(crosstab_02.sum(0),axis=1)* 100
,1)
crosstab_02
```

Out[24]:

Manuf	A	G	K	N	P	Q	R
Type							
C	0	22	23	5	9	7	8
H	1	0	0	1	0	1	0

In [25]:
```
crosstab_norm_02
```

Out[25]:

Manuf	A	G	K	N	P	Q	R
Type							
C	0.0	100.0	100.0	83.3	100.0	87.5	100.0
H	100.0	0.0	0.0	16.7	0.0	12.5	0.0

Part c

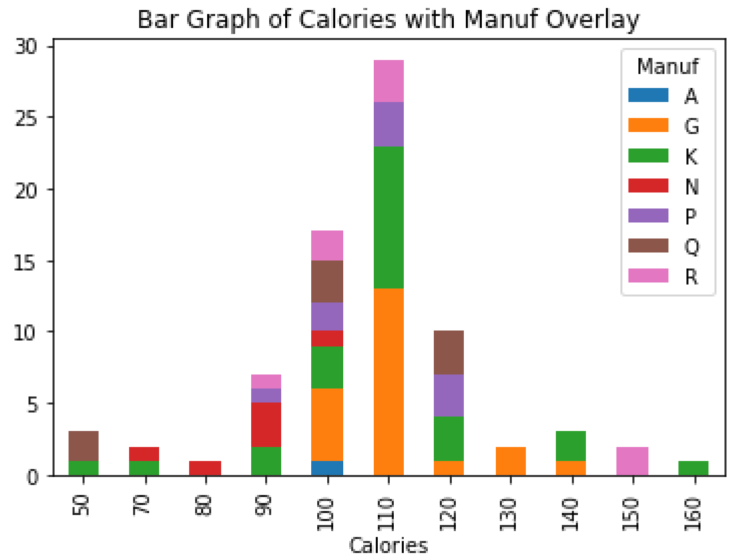
In [26]:
```
# create a histogram and normalized histogram of “Calories” with “Manuf”
overlay
crosstab_03 = pd.crosstab(cereals["Calories"],cereals['Manuf'])
plt.figure()
crosstab_03.plot(kind='bar', stacked=True)
plt.title('Bar Graph of Calories with Manuf Overlay')


# Normalized bar graph
plt.figure()
crosstab_03_norm = crosstab_03.div(crosstab_03.sum(1),axis=0)
crosstab_03_norm.plot(kind='bar', stacked=True)
plt.title('Normalized Bar Graph of Calories with Manuf Overlay')

# Stacked histogram
plt.figure()
cerealCaloriesManuf_A = cereals[cereals.Manuf == 'A '][ 'Calories']
cerealCaloriesManuf_G = cereals[cereals.Manuf == "G "][ 'Calories']
cerealCaloriesManuf_K = cereals[cereals.Manuf == "K "][ 'Calories']
cerealCaloriesManuf_N = cereals[cereals.Manuf == "N "][ 'Calories']
cerealCaloriesManuf_P = cereals[cereals.Manuf == "P "][ 'Calories']
cerealCaloriesManuf_Q = cereals[cereals.Manuf == "Q "][ 'Calories']
cerealCaloriesManuf_R = cereals[cereals.Manuf == "R "][ 'Calories']
plt.hist([cerealCaloriesManuf_A,cerealCaloriesManuf_G,cerealCaloriesManu
f_K,
          cerealCaloriesManuf_N,cerealCaloriesManuf_P,cerealCaloriesManu
f_Q,
          cerealCaloriesManuf_R],bins = 10, stacked = True)
plt.legend(['Manuf A', 'Manuf G', 'Manuf K', 'Manuf N', 'Manuf P', 'Manu
f Q', 'Manuf R'])
plt.title('Histogram of Calories with Manuf Overlay')
plt.ylabel('Frequency')
plt.xlabel('Calories')



# Normalized histogram of Calories with Manuf Overlay
(n,bins,patches)= plt.hist([cerealCaloriesManuf_A,cerealCaloriesManuf_G,
cerealCaloriesManuf_K,
          cerealCaloriesManuf_N,cerealCaloriesManuf_P,cerealCaloriesManu
f_Q,
          cerealCaloriesManuf_R],bins = 10, stacked = True)

# Creating a new plot
plt.figure()
n_table = np.column_stack((n[0],n[1]))
n_norm = n_table/n_table.sum(axis = 1)[:,None]

# creating an array of bin cuts
our_bins = np.column_stack((bins[0:10],bins[1:11]))
p1 = plt.bar(x = our_bins[:,0],height = n_norm[:,0],width = our_bins[:,1
]-our_bins[:,0])
p2 = plt.bar(x = our_bins[:,0],height = n_norm[:,1],width = our_bins[:,1
]-our_bins[:,0],bottom = n_norm[:,0])
```

```
''' I could not figure this out. '''
```

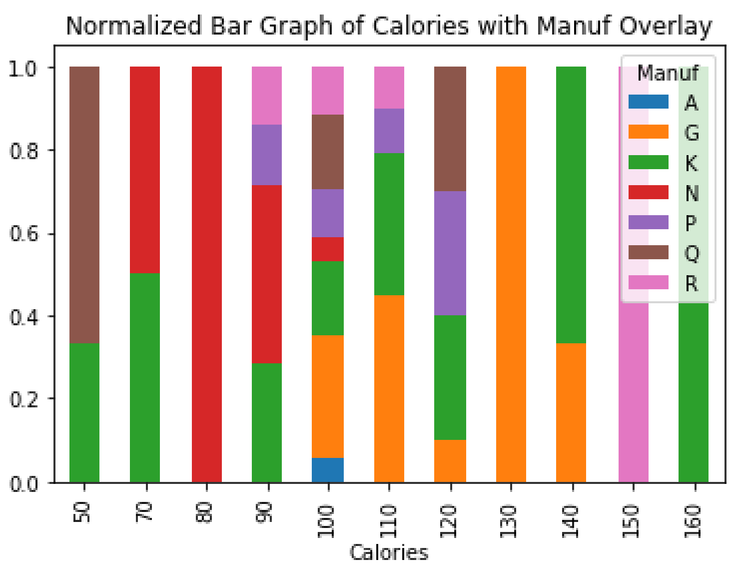
```
/Library/Python/3.7/site-packages/ipykernel_launcher.py:41: RuntimeWarn
ing: invalid value encountered in true_divide
```

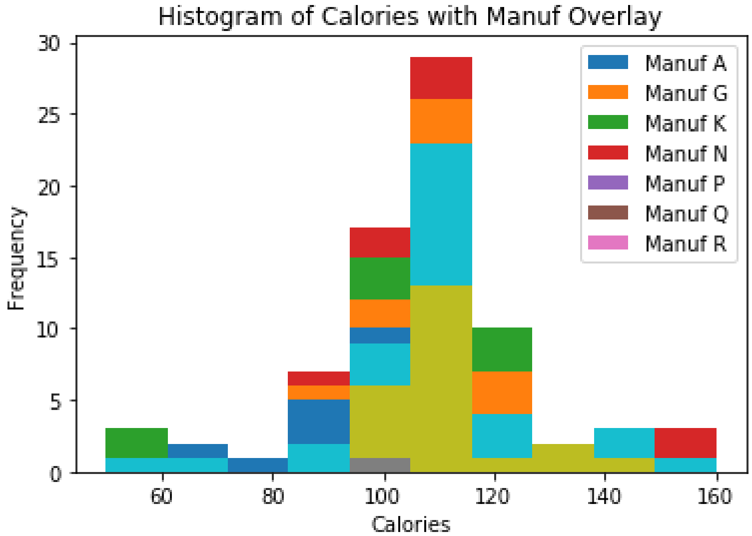
Out[26]: ' I could not figure this out. '

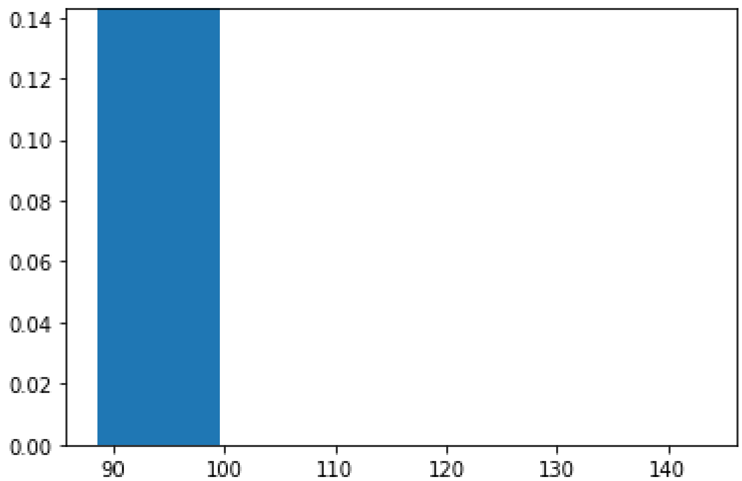
```
<Figure size 432x288 with 0 Axes>
```




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




Histogram of Calories with Manuf Overlay
Frequency
Calories
Manuf A
Manuf G
Manuf K
Manuf N
Manuf P
Manuf Q
Manuf R
0
5
10
15
20
25
30
60
80
100
120
140
160


0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
90
100
110
120
130
140

Part d

In [27]:
```
# Bin the "Calories" variable using bins for 0 to 90, 91-110, and over 1
10 calories and ...
# create a bar chart of the binned calories variable with "Manuf" overla
y
# Binning the calories
cereals["caloriesBinned"] = pd.cut(x = cereals['Calories'],bins = [0,91,
111,165],
                                   labels=['0-90','90-110','Over 110'],
right = False)
crosstab_04 = pd.crosstab(cereals['caloriesBinned'],cereals['Manuf'])
crosstab_04.plot(kind='bar',stacked = True,title = 'Bar chart of Binned
 Caloris with Manuf Overlay')
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1229259e8>