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

from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classif
ier
# import the regressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split # Import train_test_split
function
from sklearn import metrics #Import scikit-learn metrics module for accuracy c
alculation
from sklearn import tree
from scipy.stats import norm, skew
from scipy.special import boxcox1p
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
```

In [2]:
```
os.chdir('E:')
#file = pd.read_excel('E:\\eval.xlsx')
file = pd.read_excel('E:\\file.xlsx')
```

In [3]:
```
file.shape
```

Out[3]:
```
(191, 17)
```

In [4]:
```
file.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 191 entries, 0 to 190
Data columns (total 17 columns):
State                     191 non-null object
District                  191 non-null object
Population                191 non-null int64
calcu mmr                 187 non-null float64
calcu IMR 12              191 non-null float64
pnc 48 per                191 non-null float64
pnc per live birth        191 non-null float64
per capita health exp     191 non-null int64
health per of gdp         191 non-null float64
phc per 100000            173 non-null float64
female pop per            191 non-null float64
literacy                  191 non-null float64
anc                       191 non-null float64
anc check                 191 non-null float64
per home skill            182 non-null float64
percent insti             191 non-null float64
haemo                     191 non-null float64
dtypes: float64(13), int64(2), object(2)
memory usage: 25.5+ KB
```

In [5]:
```
file.describe()
```

Out[5]:

	Population	calcu mmr	calcu IMR 12	pnc 48 per	pnc per live birth	per capita health exp	health per of gdp
count	1.910000e+02	187.000000	191.000000	191.000000	191.000000	191.000000	191.000000
mean	1.976512e+06	116.236033	16.929357	11.717047	0.721601	1011.460733	0.867958
std	1.476784e+06	96.986826	10.985973	15.895449	0.445789	339.496527	0.223881
min	2.552300e+05	5.020080	1.055528	0.000000	0.004433	716.000000	0.600000
25%	1.127692e+06	55.742187	8.479796	0.486583	0.333550	716.000000	0.630000
50%	1.563715e+06	99.037917	15.195694	3.324870	0.756028	1011.000000	0.820000
75%	2.374760e+06	148.612299	22.905693	20.460703	1.025323	1119.000000	1.040000
max	1.106015e+07	639.836546	50.547538	81.630170	1.819892	3643.000000	1.340000

In [6]:
```
file.head(10)
```

Out[6]:

	State	District	Population	calcu mmr	calcu IMR 12	pnc 48 per	pnc per live birth	per capita health exp	healt per c gd
0	Andhra Pradesh	Anantapur	4081148	87.204323	14.513291	6.905952	0.719763	1013	0.8
1	Andhra Pradesh	Chittoor	4174064	106.414208	11.522090	38.204631	0.916868	1013	0.8
2	Andhra Pradesh	East Godavari	5154296	115.647356	16.755419	2.223678	1.355212	1013	0.8
3	Andhra Pradesh	Guntur	4887813	133.824021	17.245050	6.171856	1.011953	1013	0.8
4	Andhra Pradesh	Krishna	4517398	84.362760	10.756252	1.952910	0.191308	1013	0.8
5	Andhra Pradesh	Kurnool	4053463	148.166830	19.498124	11.125496	0.974039	1013	0.8
6	Andhra Pradesh	Nellore	2963557	43.552953	3.871374	0.844179	0.963053	1013	0.8
7	Andhra Pradesh	Prakasam	3397448	38.599057	6.094588	0.290818	1.180562	1013	0.8
8	Andhra Pradesh	Srikakulam	2703114	62.293484	10.752397	6.345035	1.205866	1013	0.8
9	Andhra Pradesh	Vishakapatnam	4290589	244.482730	22.430060	6.982224	0.640610	1013	0.8

In [7]:
```
#handling missing data
all_data_na = (file.isnull().sum() / len(file)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_value
s(ascending=False)[:30]
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head()
```

Out[7]:

	Missing Ratio
phc per 100000	9.424084
per home skill	4.712042
calcu mmr	2.094241

In [8]:
```
file["calcu mmr"] = file.groupby("State")["calcu mmr"].transform(
    lambda x: x.fillna(x.mean()))
file["phc per 100000"] = file.groupby("State")["phc per 100000"].transform(
    lambda x: x.fillna(x.mean()))
file["per home skill"] = file.groupby("State")["per home skill"].transform(
    lambda x: x.fillna(x.mean()))
```

In [9]:
```
#handling missing data
all_data_na = (file.isnull().sum() / len(file)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).sort_value
s(ascending=False)[:30]
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head()
```

Out[9]:

Missing Ratio

In [10]:
```
cols = ['calcu mmr', 'calcu IMR 12', 'literacy', 'phc per 100000', 'percent in
sti', 'per home skill', 'per capita health exp', 'anc', 'anc check', 'pnc per
 live birth', 'haemo']
data = file.loc[:, cols]
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 191 entries, 0 to 190
Data columns (total 11 columns):
calcu mmr                191 non-null float64
calcu IMR 12             191 non-null float64
literacy                 191 non-null float64
phc per 100000           191 non-null float64
percent insti            191 non-null float64
per home skill           191 non-null float64
per capita health exp    191 non-null int64
anc                      191 non-null float64
anc check                191 non-null float64
pnc per live birth       191 non-null float64
haemo                    191 non-null float64
dtypes: float64(10), int64(1)
memory usage: 16.5 KB
```

In [11]:
```
data.head()
```

Out[11]:

	calcu mmr	calcu IMR 12	literacy	phc per 100000	percent insti	per home skill	per capita health exp	anc	anc check	l
0	87.204323	14.513291	63.57	1.960233	99.482275	69.418960	1013	3.993238	3.905973	(
1	106.414208	11.522090	71.53	2.252002	99.947186	53.571429	1013	3.567691	4.161339	(
2	115.647356	16.755419	70.99	2.308754	99.552218	78.048780	1013	3.299216	3.521318	1
3	133.824021	17.245050	67.40	1.677642	99.929149	64.583333	1013	3.199218	3.997480	1
4	84.362760	10.756252	73.74	1.793068	99.986234	77.777778	1013	3.333573	3.315837	(

In [12]:
```
numeric_feats = data.dtypes[data.dtypes != "object"].index

# Check the skew of all numerical features
skewed_feats = data[numeric_feats].apply(lambda x: skew(x.dropna())).sort_valu
es(ascending=False)
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

```
Skew in numerical features:
```

Out[12]:

	Skew
anc check	11.332658
anc	10.485773
per capita health exp	4.922015
calcu mmr	2.797309
haemo	1.794508
phc per 100000	1.336422
calcu IMR 12	0.947322
per home skill	0.682056
pnc per live birth	0.131436
literacy	-0.454455

In [13]:
```
#box-cox transformation
skewness = skewness[abs(skewness) > 0.75]
print("There are {} skewed numerical features to Box Cox transform".format(ske
wness.shape[0]))

from scipy.special import boxcox1p
skewed_features = skewness.index
lam = 0
for feat in skewed_features:
    #data[feat] += 1
    data[feat] = boxcox1p(data[feat], lam)

#data[skewed_features] = np.log1p(data[skewed_features])
```

```
There are 11 skewed numerical features to Box Cox transform
```

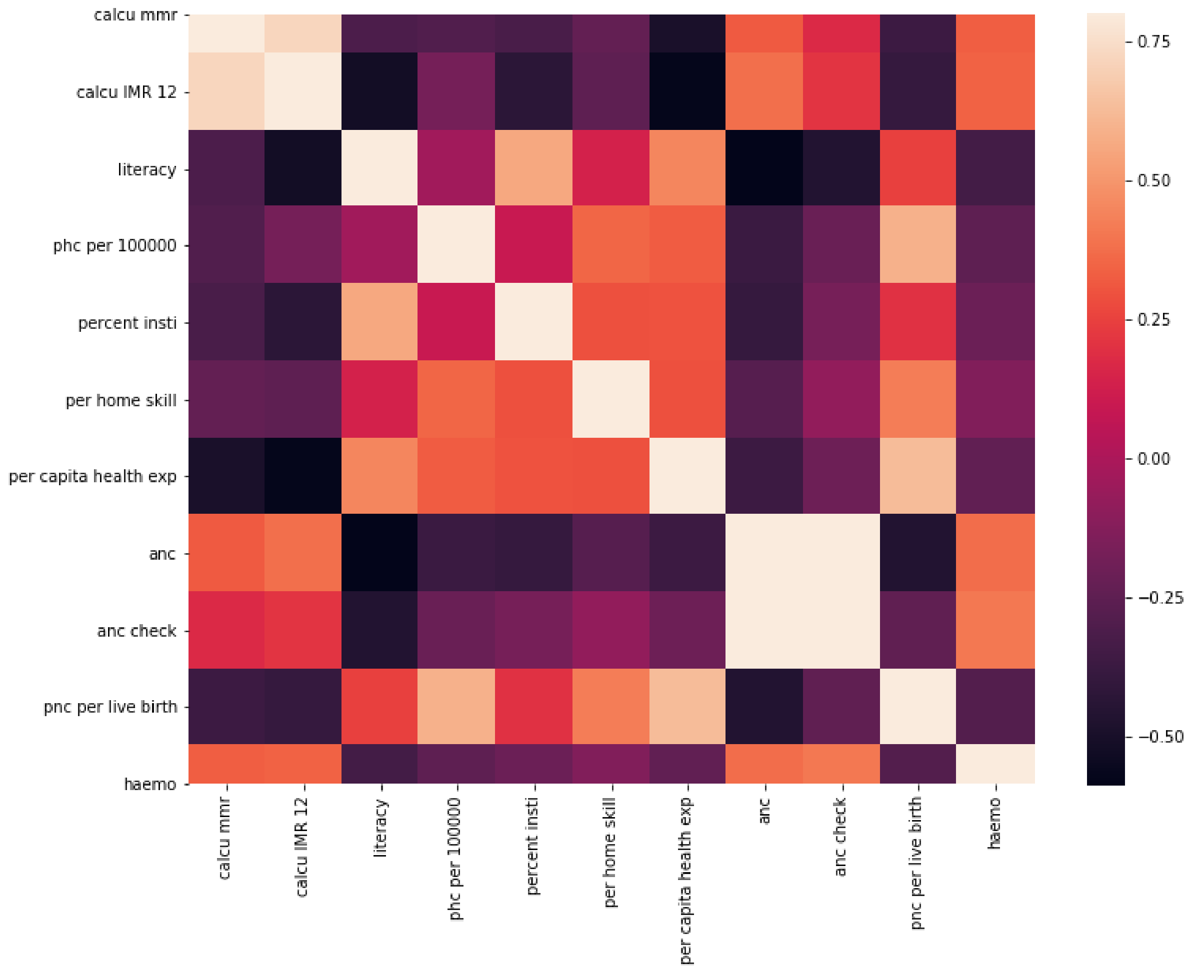
In [14]:
```
data.describe()
```

Out[14]:

	calcu mmr	calcu IMR 12	literacy	phc per 100000	percent insti	per home skill	per capita health exp	
count	191.000000	191.000000	191.000000	191.000000	191.000000	191.000000	191.000000	191.
mean	4.476621	2.679011	4.303420	1.217615	4.586634	3.146472	6.884695	1.
std	0.802962	0.689933	0.161916	0.329786	0.054521	1.099575	0.247782	0.
min	1.795101	0.720533	3.613617	0.321777	4.246262	0.160429	6.575076	1.
25%	4.038518	2.249120	4.232656	1.004658	4.586280	2.684448	6.575076	1.
50%	4.610115	2.784745	4.322277	1.179271	4.609294	3.433987	6.919684	1.
75%	5.006536	3.174116	4.410064	1.435571	4.613923	3.984476	7.021084	1.
max	6.462774	3.942504	4.587108	2.187049	4.615121	4.615121	8.200837	3.

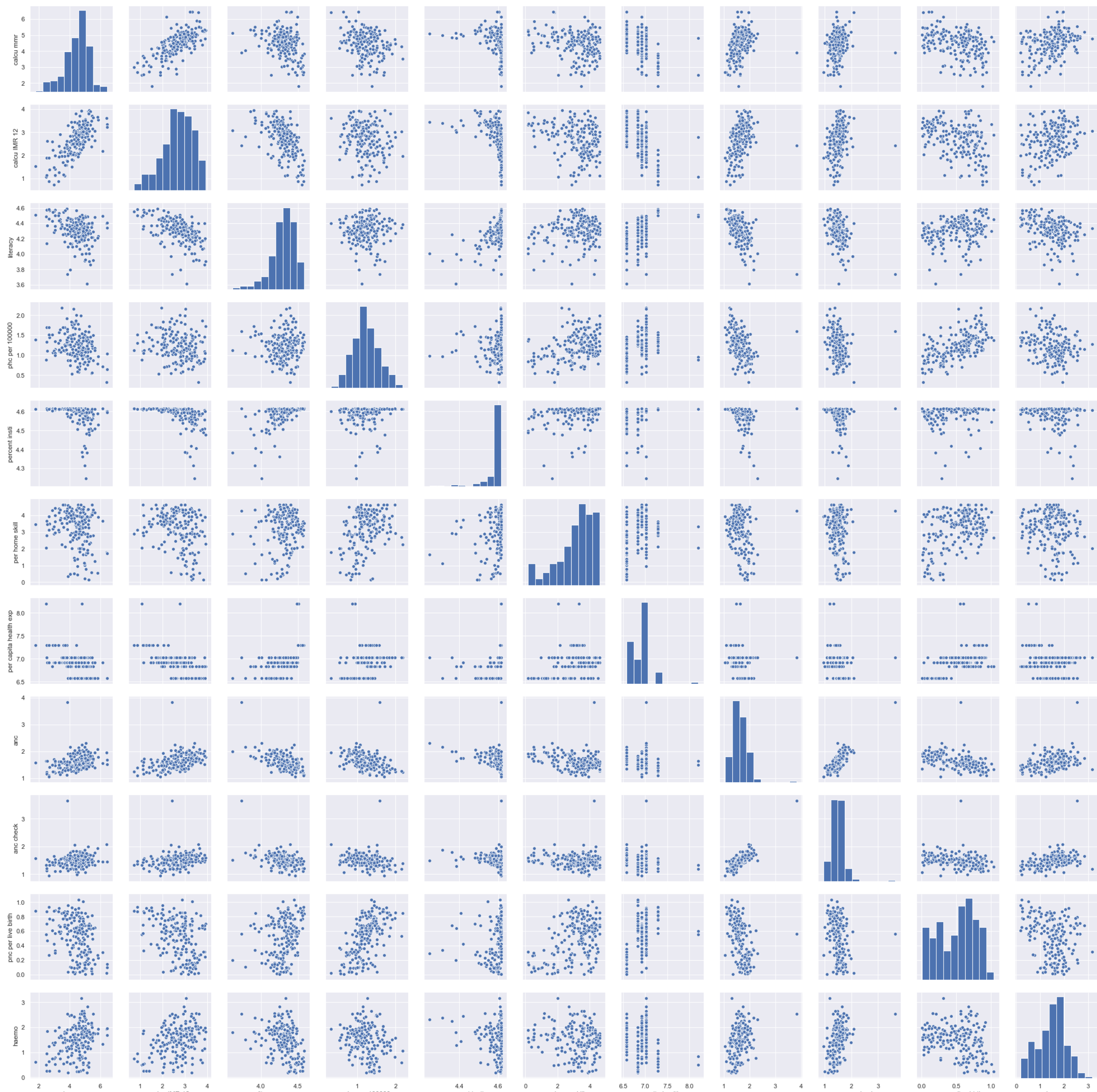
In [15]:
```
corrmat = data.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True)
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1a47608e248>

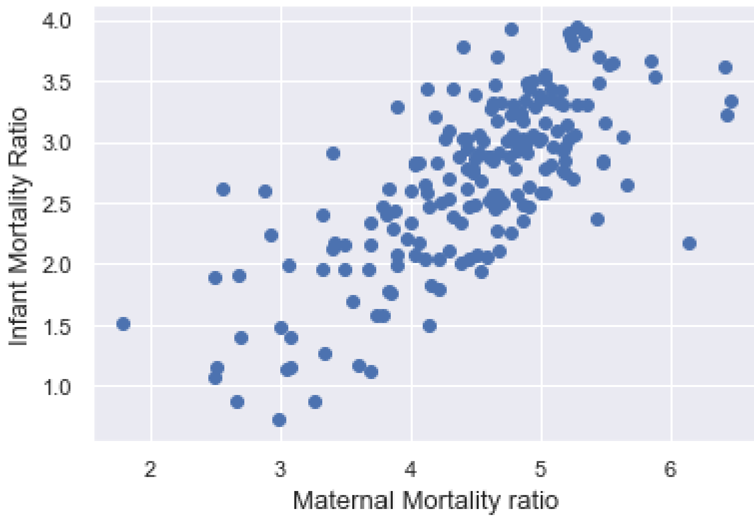


In [16]:
```
sns.set()
sns.pairplot(data[cols], size=2.5)
plt.show()
```

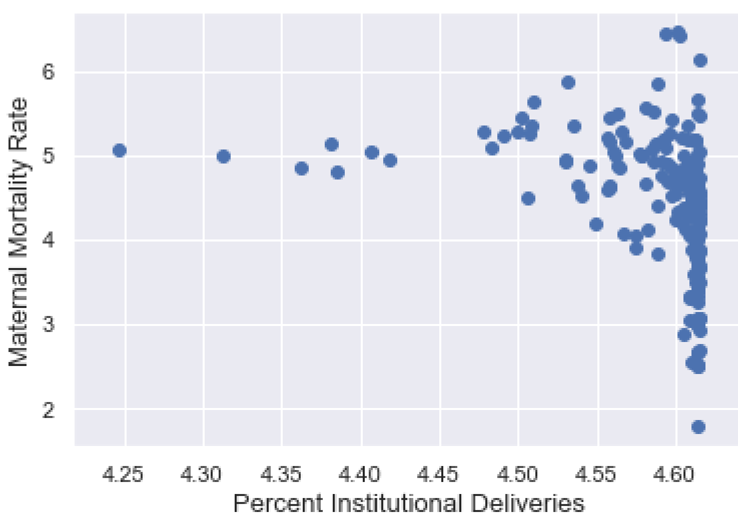
```
E:\Anaconda3\lib\site-packages\seaborn\axisgrid.py:2065: UserWarning: The `si
ze` parameter has been renamed to `height`; pleaes update your code.
  warnings.warn(msg, UserWarning)
```



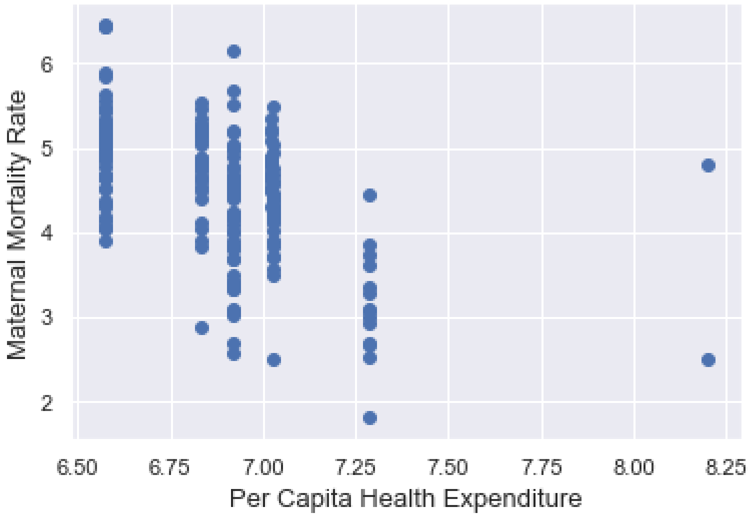
In [17]:
```
fig, ax = plt.subplots()
ax.scatter(data['calcu mmr'], data['calcu IMR 12'])
plt.ylabel('Infant Mortality Ratio', fontsize=13)
plt.xlabel('Maternal Mortality ratio', fontsize=13)
plt.show()
```



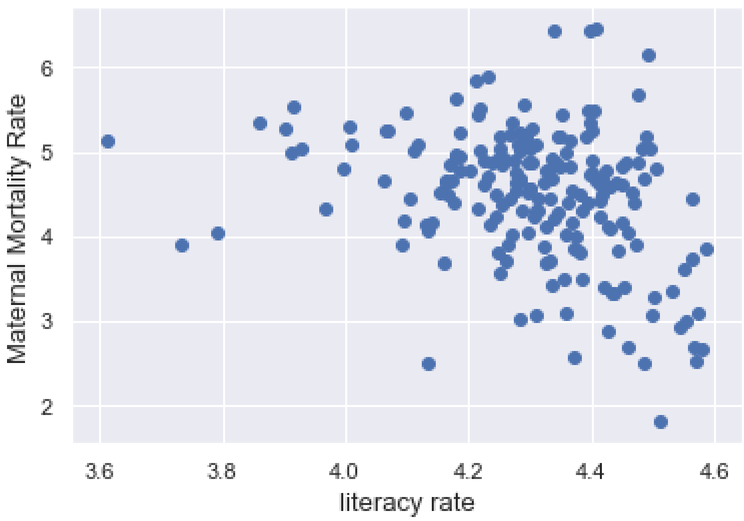
In [34]:
```
fig, ax = plt.subplots()
ax.scatter(data['percent insti'], data['calcu mmr'])
plt.ylabel('Maternal Mortality Rate', fontsize=13)
plt.xlabel('Percent Institutional Deliveries', fontsize=13)
plt.show()
```



In [35]:
```
fig, ax = plt.subplots()
ax.scatter(data['per capita health exp'], data['calcu mmr'])
plt.ylabel('Maternal Mortality Rate', fontsize=13)
plt.xlabel('Per Capita Health Expenditure', fontsize=13)
plt.show()
```

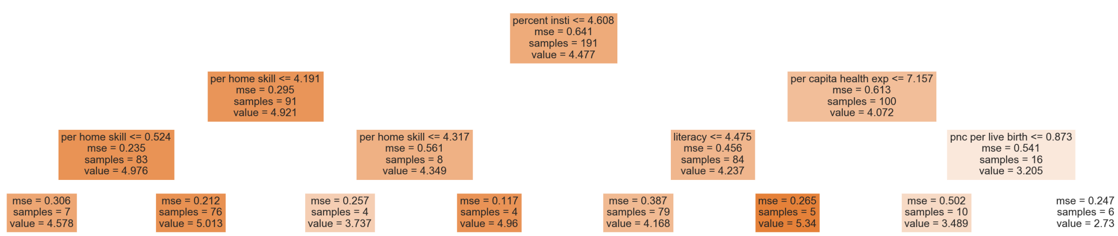


In [18]:
```
fig, ax = plt.subplots()
ax.scatter(data['literacy'], data['calcu mmr'])
plt.ylabel('Maternal Mortality Rate', fontsize=13)
plt.xlabel('literacy rate', fontsize=13)
plt.show()
```



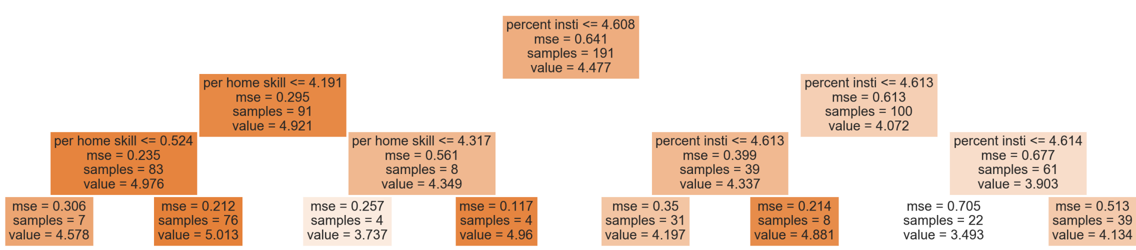
In [19]:
```
feature_cols = ['literacy', 'percent insti', 'per home skill', 'pnc per live b
irth', 'per capita health exp']
x = data[feature_cols]
y = data['calcu mmr']

# create a regressor object
reg = DecisionTreeRegressor(random_state = 0, max_depth = 3)

# fit the regressor with X and Y data
model = reg.fit(x, y)

fig = plt.figure(figsize=(50,10))
tree.plot_tree(reg, feature_names = feature_cols,  class_names = 'calcu mmr',
filled=True)
```

Out[19]:
```
[Text(1395.0, 475.65000000000003, 'percent insti <= 4.608\nmse = 0.641\nsampl
es = 191\nvalue = 4.477'),
 Text(697.5, 339.75, 'per home skill <= 4.191\nmse = 0.295\nsamples = 91\nval
ue = 4.921'),
 Text(348.75, 203.85000000000002, 'per home skill <= 0.524\nmse = 0.235\nsamp
les = 83\nvalue = 4.976'),
 Text(174.375, 67.94999999999999, 'mse = 0.306\nsamples = 7\nvalue = 4.578'),
 Text(523.125, 67.94999999999999, 'mse = 0.212\nsamples = 76\nvalue = 5.01
3'),
 Text(1046.25, 203.85000000000002, 'per home skill <= 4.317\nmse = 0.561\nsam
ples = 8\nvalue = 4.349'),
 Text(871.875, 67.94999999999999, 'mse = 0.257\nsamples = 4\nvalue = 3.737'),
 Text(1220.625, 67.94999999999999, 'mse = 0.117\nsamples = 4\nvalue = 4.96'),
 Text(2092.5, 339.75, 'per capita health exp <= 7.157\nmse = 0.613\nsamples =
100\nvalue = 4.072'),
 Text(1743.75, 203.85000000000002, 'literacy <= 4.475\nmse = 0.456\nsamples =
84\nvalue = 4.237'),
 Text(1569.375, 67.94999999999999, 'mse = 0.387\nsamples = 79\nvalue = 4.16
8'),
 Text(1918.125, 67.94999999999999, 'mse = 0.265\nsamples = 5\nvalue = 5.34'),
 Text(2441.25, 203.85000000000002, 'pnc per live birth <= 0.873\nmse = 0.541
\nsamples = 16\nvalue = 3.205'),
 Text(2266.875, 67.94999999999999, 'mse = 0.502\nsamples = 10\nvalue = 3.48
9'),
 Text(2615.625, 67.94999999999999, 'mse = 0.247\nsamples = 6\nvalue = 2.73')]
```



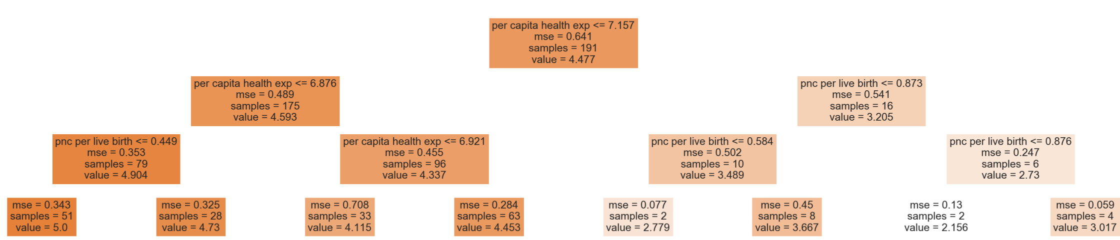
In [20]:
```
feature_cols = ['percent insti', 'per home skill']
x = data[feature_cols]
y = data['calcu mmr']

# create a regressor object
reg = DecisionTreeRegressor(random_state = 0, max_depth = 3)

# fit the regressor with X and Y data
model = reg.fit(x, y)

fig = plt.figure(figsize=(50,10))
tree.plot_tree(reg, feature_names = feature_cols,  class_names = 'calcu mmr',
filled=True)
```

Out[20]:
```
[Text(1395.0, 475.65000000000003, 'percent insti <= 4.608\nmse = 0.641\nsampl
es = 191\nvalue = 4.477'),
 Text(697.5, 339.75, 'per home skill <= 4.191\nmse = 0.295\nsamples = 91\nval
ue = 4.921'),
 Text(348.75, 203.85000000000002, 'per home skill <= 0.524\nmse = 0.235\nsamp
les = 83\nvalue = 4.976'),
 Text(174.375, 67.94999999999999, 'mse = 0.306\nsamples = 7\nvalue = 4.578'),
 Text(523.125, 67.94999999999999, 'mse = 0.212\nsamples = 76\nvalue = 5.01
3'),
 Text(1046.25, 203.85000000000002, 'per home skill <= 4.317\nmse = 0.561\nsam
ples = 8\nvalue = 4.349'),
 Text(871.875, 67.94999999999999, 'mse = 0.257\nsamples = 4\nvalue = 3.737'),
 Text(1220.625, 67.94999999999999, 'mse = 0.117\nsamples = 4\nvalue = 4.96'),
 Text(2092.5, 339.75, 'percent insti <= 4.613\nmse = 0.613\nsamples = 100\nva
lue = 4.072'),
 Text(1743.75, 203.85000000000002, 'percent insti <= 4.613\nmse = 0.399\nsamp
les = 39\nvalue = 4.337'),
 Text(1569.375, 67.94999999999999, 'mse = 0.35\nsamples = 31\nvalue = 4.19
7'),
 Text(1918.125, 67.94999999999999, 'mse = 0.214\nsamples = 8\nvalue = 4.88
1'),
 Text(2441.25, 203.85000000000002, 'percent insti <= 4.614\nmse = 0.677\nsamp
les = 61\nvalue = 3.903'),
 Text(2266.875, 67.94999999999999, 'mse = 0.705\nsamples = 22\nvalue = 3.49
3'),
 Text(2615.625, 67.94999999999999, 'mse = 0.513\nsamples = 39\nvalue = 4.13
4')]
```

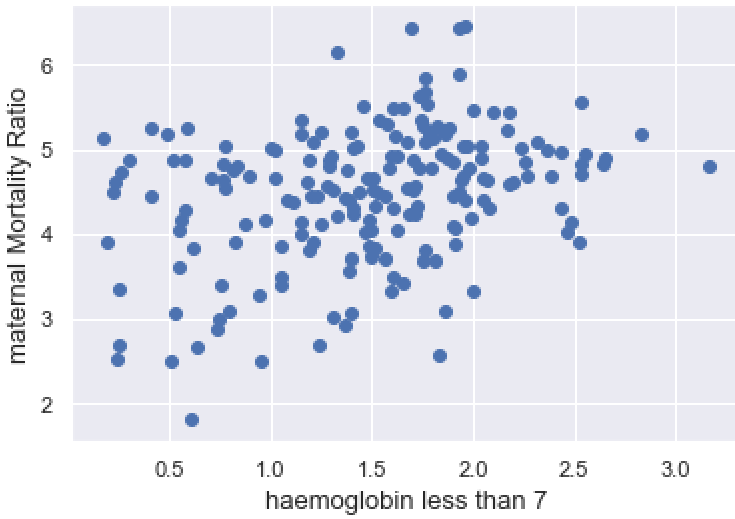


In [21]:
```python
feature_cols = ['pnc per live birth', 'per capita health exp']
x = data[feature_cols]
y = data['calcu mmr']

# create a regressor object
reg = DecisionTreeRegressor(random_state = 0, max_depth = 3)

# fit the regressor with X and Y data
model = reg.fit(x, y)

fig = plt.figure(figsize=(50,10))
tree.plot_tree(reg, feature_names = feature_cols,  class_names = 'calcu mmr',
filled=True)
```

Out[21]:
```
[Text(1395.0, 475.65000000000003, 'per capita health exp <= 7.157\nmse = 0.64
1\nsamples = 191\nvalue = 4.477'),
 Text(697.5, 339.75, 'per capita health exp <= 6.876\nmse = 0.489\nsamples =
175\nvalue = 4.593'),
 Text(348.75, 203.85000000000002, 'pnc per live birth <= 0.449\nmse = 0.353\n
samples = 79\nvalue = 4.904'),
 Text(174.375, 67.94999999999999, 'mse = 0.343\nsamples = 51\nvalue = 5.0'),
 Text(523.125, 67.94999999999999, 'mse = 0.325\nsamples = 28\nvalue = 4.73'),
 Text(1046.25, 203.85000000000002, 'per capita health exp <= 6.921\nmse = 0.4
55\nsamples = 96\nvalue = 4.337'),
 Text(871.875, 67.94999999999999, 'mse = 0.708\nsamples = 33\nvalue = 4.11
5'),
 Text(1220.625, 67.94999999999999, 'mse = 0.284\nsamples = 63\nvalue = 4.45
3'),
 Text(2092.5, 339.75, 'pnc per live birth <= 0.873\nmse = 0.541\nsamples = 16
\nvalue = 3.205'),
 Text(1743.75, 203.85000000000002, 'pnc per live birth <= 0.584\nmse = 0.502
\nsamples = 10\nvalue = 3.489'),
 Text(1569.375, 67.94999999999999, 'mse = 0.077\nsamples = 2\nvalue = 2.77
9'),
 Text(1918.125, 67.94999999999999, 'mse = 0.45\nsamples = 8\nvalue = 3.667'),
 Text(2441.25, 203.85000000000002, 'pnc per live birth <= 0.876\nmse = 0.247
\nsamples = 6\nvalue = 2.73'),
 Text(2266.875, 67.94999999999999, 'mse = 0.13\nsamples = 2\nvalue = 2.156'),
 Text(2615.625, 67.94999999999999, 'mse = 0.059\nsamples = 4\nvalue = 3.01
7')]
```



In [24]:
```
fig, ax = plt.subplots()
ax.scatter(data['haemo'], data['calcu mmr'])
plt.ylabel('Maternal Mortality Ratio', fontsize=13)
plt.xlabel('Percent women with haemoglobin less than 7', fontsize=13)
plt.show()
```



In [25]:
```
haemo =  data['haemo'].values.reshape(-1, 1)
mmr = data['calcu mmr'].values.reshape(-1, 1)
regressor = LinearRegression()
regressor.fit(haemo, mmr)
```

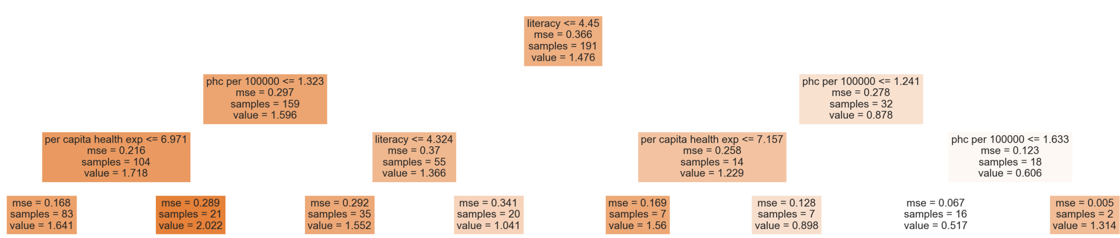
Out[25]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

In [26]:
```
#To retrieve the intercept:
print(regressor.intercept_)
#For retrieving the slope:
print(regressor.coef_)
```

```
[3.83009803]
[[0.43799844]]
```

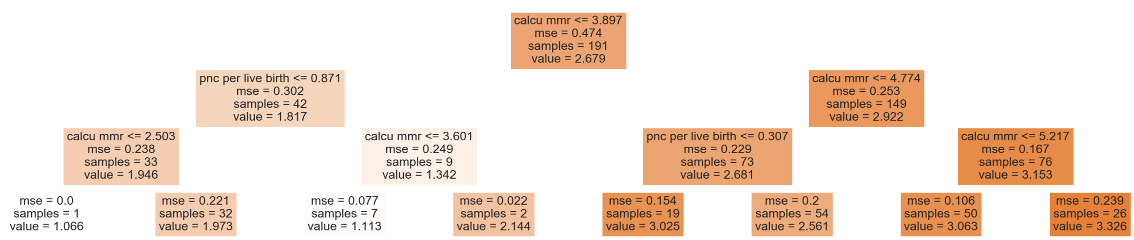
In [27]:
```
feature_cols = ['literacy', 'per capita health exp', 'phc per 100000']
x = data[feature_cols]
y = data['haemo']

# create a regressor object
reg = DecisionTreeRegressor(random_state = 0, max_depth = 3)

# fit the regressor with X and Y data
model = reg.fit(x, y)

fig = plt.figure(figsize=(50,10))
tree.plot_tree(reg, feature_names = feature_cols,  class_names = 'haemo', fill
ed=True)
```

Out[27]:
```
[Text(1395.0, 475.65000000000003, 'literacy <= 4.45\nmse = 0.366\nsamples = 1
91\nvalue = 1.476'),
 Text(697.5, 339.75, 'phc per 100000 <= 1.323\nmse = 0.297\nsamples = 159\nva
lue = 1.596'),
 Text(348.75, 203.85000000000002, 'per capita health exp <= 6.971\nmse = 0.21
6\nsamples = 104\nvalue = 1.718'),
 Text(174.375, 67.94999999999999, 'mse = 0.168\nsamples = 83\nvalue = 1.64
1'),
 Text(523.125, 67.94999999999999, 'mse = 0.289\nsamples = 21\nvalue = 2.02
2'),
 Text(1046.25, 203.85000000000002, 'literacy <= 4.324\nmse = 0.37\nsamples =
55\nvalue = 1.366'),
 Text(871.875, 67.94999999999999, 'mse = 0.292\nsamples = 35\nvalue = 1.55
2'),
 Text(1220.625, 67.94999999999999, 'mse = 0.341\nsamples = 20\nvalue = 1.04
1'),
 Text(2092.5, 339.75, 'phc per 100000 <= 1.241\nmse = 0.278\nsamples = 32\nva
lue = 0.878'),
 Text(1743.75, 203.85000000000002, 'per capita health exp <= 7.157\nmse = 0.2
58\nsamples = 14\nvalue = 1.229'),
 Text(1569.375, 67.94999999999999, 'mse = 0.169\nsamples = 7\nvalue = 1.56'),
 Text(1918.125, 67.94999999999999, 'mse = 0.128\nsamples = 7\nvalue = 0.89
8'),
 Text(2441.25, 203.85000000000002, 'phc per 100000 <= 1.633\nmse = 0.123\nsam
ples = 18\nvalue = 0.606'),
 Text(2266.875, 67.94999999999999, 'mse = 0.067\nsamples = 16\nvalue = 0.51
7'),
 Text(2615.625, 67.94999999999999, 'mse = 0.005\nsamples = 2\nvalue = 1.31
4')]
```



In [28]:
```
feature_cols = ['calcu mmr', 'pnc per live birth']
x = data[feature_cols]
y = data['calcu IMR 12']

# create a regressor object
reg = DecisionTreeRegressor(random_state = 0, max_depth = 3)

# fit the regressor with X and Y data
model = reg.fit(x, y)

fig = plt.figure(figsize=(50,10))
tree.plot_tree(reg, feature_names = feature_cols,  class_names = 'calcu IMR 1
2', filled=True)
```

Out[28]:
```
[Text(1395.0, 475.65000000000003, 'calcu mmr <= 3.897\nmse = 0.474\nsamples =
191\nvalue = 2.679'),
 Text(697.5, 339.75, 'pnc per live birth <= 0.871\nmse = 0.302\nsamples = 42
\nvalue = 1.817'),
 Text(348.75, 203.85000000000002, 'calcu mmr <= 2.503\nmse = 0.238\nsamples =
33\nvalue = 1.946'),
 Text(174.375, 67.94999999999999, 'mse = 0.0\nsamples = 1\nvalue = 1.066'),
 Text(523.125, 67.94999999999999, 'mse = 0.221\nsamples = 32\nvalue = 1.97
3'),
 Text(1046.25, 203.85000000000002, 'calcu mmr <= 3.601\nmse = 0.249\nsamples
= 9\nvalue = 1.342'),
 Text(871.875, 67.94999999999999, 'mse = 0.077\nsamples = 7\nvalue = 1.113'),
 Text(1220.625, 67.94999999999999, 'mse = 0.022\nsamples = 2\nvalue = 2.14
4'),
 Text(2092.5, 339.75, 'calcu mmr <= 4.774\nmse = 0.253\nsamples = 149\nvalue
= 2.922'),
 Text(1743.75, 203.85000000000002, 'pnc per live birth <= 0.307\nmse = 0.229
\nsamples = 73\nvalue = 2.681'),
 Text(1569.375, 67.94999999999999, 'mse = 0.154\nsamples = 19\nvalue = 3.02
5'),
 Text(1918.125, 67.94999999999999, 'mse = 0.2\nsamples = 54\nvalue = 2.561'),
 Text(2441.25, 203.85000000000002, 'calcu mmr <= 5.217\nmse = 0.167\nsamples
= 76\nvalue = 3.153'),
 Text(2266.875, 67.94999999999999, 'mse = 0.106\nsamples = 50\nvalue = 3.06
3'),
 Text(2615.625, 67.94999999999999, 'mse = 0.239\nsamples = 26\nvalue = 3.32
6')]
```



In [29]:
```
imr =  data['calcu IMR 12'].values.reshape(-1, 1)
mmr = data['calcu mmr'].values.reshape(-1, 1)
regressor = LinearRegression()
regressor.fit(mmr, imr)
```

Out[29]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

In [30]:
```
#To retrieve the intercept:
print(regressor.intercept_)
#For retrieving the slope:
print(regressor.coef_)
```

```
[-0.0906748]
[[0.61870019]]
```

In [32]:
```
#x = daata["RM"] ## X usually means our input variables (or independent variab
les)
#y = target["MEDV"] ## Y usually means our output/dependent variable
imr =  data['calcu IMR 12'].values.reshape(-1, 1)
mmr = data['calcu mmr'].values.reshape(-1, 1)
mmr = sm.add_constant(mmr) ## let's add an intercept (beta_0) to our model

# Note the difference in argument order
model = sm.OLS(imr, mmr).fit() ## sm.OLS(output, input)

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

Out[32]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.518
Model:	OLS	**Adj. R-squared:**	0.516
Method:	Least Squares	**F-statistic:**	203.5
Date:	Tue, 18 Aug 2020	**Prob (F-statistic):**	8.13e-32
Time:	20:43:21	**Log-Likelihood:**	-129.83
No. Observations:	191	**AIC:**	263.7
Df Residuals:	189	**BIC:**	270.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.0907	0.197	-0.460	0.646	-0.480	0.298
x1	0.6187	0.043	14.266	0.000	0.533	0.704

Omnibus:	0.610	**Durbin-Watson:**	1.620
Prob(Omnibus):	0.737	**Jarque-Bera (JB):**	0.601
Skew:	-0.134	**Prob(JB):**	0.740
Kurtosis:	2.937	**Cond. No.**	27.0

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

In [36]:
```
#x = daata["RM"] ## X usually means our input variables (or independent variab
les)
#y = target["MEDV"] ## Y usually means our output/dependent variable
yy =  data['calcu mmr'].values.reshape(-1, 1)
xx = data['haemo'].values.reshape(-1, 1)
xx = sm.add_constant(xx) ## let's add an intercept (beta_0) to our model

# Note the difference in argument order
model = sm.OLS(yy, xx).fit() ## sm.OLS(output, input)

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

Out[36]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.109
Model:	OLS	**Adj. R-squared:**	0.105
Method:	Least Squares	**F-statistic:**	23.23
Date:	Wed, 19 Aug 2020	**Prob (F-statistic):**	2.95e-06
Time:	00:15:04	**Log-Likelihood:**	-217.53
No. Observations:	191	**AIC:**	439.1
Df Residuals:	189	**BIC:**	445.6
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	3.8301	0.145	26.418	0.000	3.544	4.116
x1	0.4380	0.091	4.819	0.000	0.259	0.617

Omnibus:	3.886	**Durbin-Watson:**	1.337
Prob(Omnibus):	0.143	**Jarque-Bera (JB):**	3.573
Skew:	-0.329	**Prob(JB):**	0.168
Kurtosis:	3.122	**Cond. No.**	5.68

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