1.

$$\theta^T X^T = \left[\theta_0 + \theta_1 x_1^{(1)} + \cdots + \theta_n x_n^{(1)}, \cdots \cdot \theta_0 + \theta_1 x_1^{(m)} + \cdots + \theta_n x_n^{(m)}\right]$$

$$X \cdot \theta = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} = \begin{bmatrix} \theta_0 + \theta_1 x_1^{(1)} + \cdots + \theta_n x_n^{(1)} \\ \vdots \\ \theta_0 + \theta_1 x_1^{(m)} + \cdots + \theta_n x_n^{(m)} \end{bmatrix}$$

$$\Rightarrow \theta^T X^T X \theta = \left[\theta_0 + \theta_1 x_1^{(1)} + \cdots + \theta_n x_n^{(1)}\right]^2 + \cdots + \left[\theta_0 + \theta_1 x_1^{(m)} + \cdots + \theta_n x_n^{(m)}\right]^2$$

$$\frac{\partial \theta^T X^T X \theta}{\partial \theta} = 2 \cdot \begin{bmatrix} (\theta_0 + \theta_1 x_1^{(1)} + \cdots + \theta_n \cdot x_n^{(1)}) + \cdots + (\theta_0 + x_1^{(m)} \cdot \theta_1 + \cdots + \theta_n \cdot x_n^{(m)}) \\ x_1^{(1)} \cdot (\theta_0 + \theta_1 x_1^{(1)} + \cdots + \theta_n x_n^{(1)}) + \cdots + x_1^{(m)} (\theta_0 + \theta_1 x_1^{(m)} + \cdots + \theta_n x_n^{(m)}) \\ \vdots \\ x_n^{(1)} \cdot (\theta_0 + \theta_1 x_1^{(1)} + \cdots + \theta_n x_n^{(1)}) + \cdots + x_n^{(m)} \cdot (\theta_0 + \cdots + \theta_n x_n^{(m)}) \end{bmatrix}$$

Mean while,

$$X^T \cdot X\theta = \begin{bmatrix} 1 & \cdots & 1 \\ x_1^{(1)} & \cdots & x_1^{(m)} \\ \vdots & & \vdots \\ x_n^{(1)} & \cdots & x_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} \theta_0 + x_1^{(1)}\theta_1 + \cdots + x_n^{(1)}\theta_n \\ \vdots \\ \theta_0 + x_1^{(m)}\theta_1 + \cdots + x_n^{(m)}\theta_n \end{bmatrix}$$

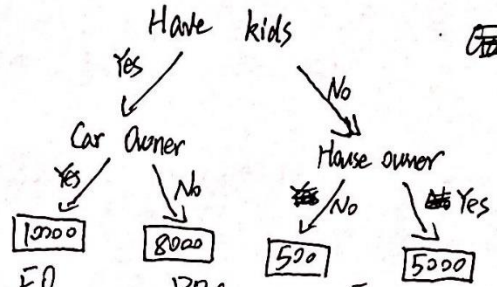
$$= \begin{bmatrix} (\theta_0 + x_1^{(1)}\theta_1 + \cdots + x_n^{(1)}\theta_n) + \cdots + (\theta_0 + x_1^{(m)} \cdot \theta_1 + \cdots + x_n^{(m)} \cdot \theta_n) \\ \vdots \\ x_n^{(1)} \cdot (\theta_0 + x_1^{(1)}\theta_1 + \cdots + x_n^{(1)}\theta_n) + \cdots + x_n^{(m)} \cdot (\theta_0 + x_1^{(m)}\theta_1 + \cdots + x_n^{(m)}\theta_n) \end{bmatrix}$$

Therefore, $\frac{\partial \theta^T X^T X \theta}{\partial \theta} = 2X^T X\theta$

2.

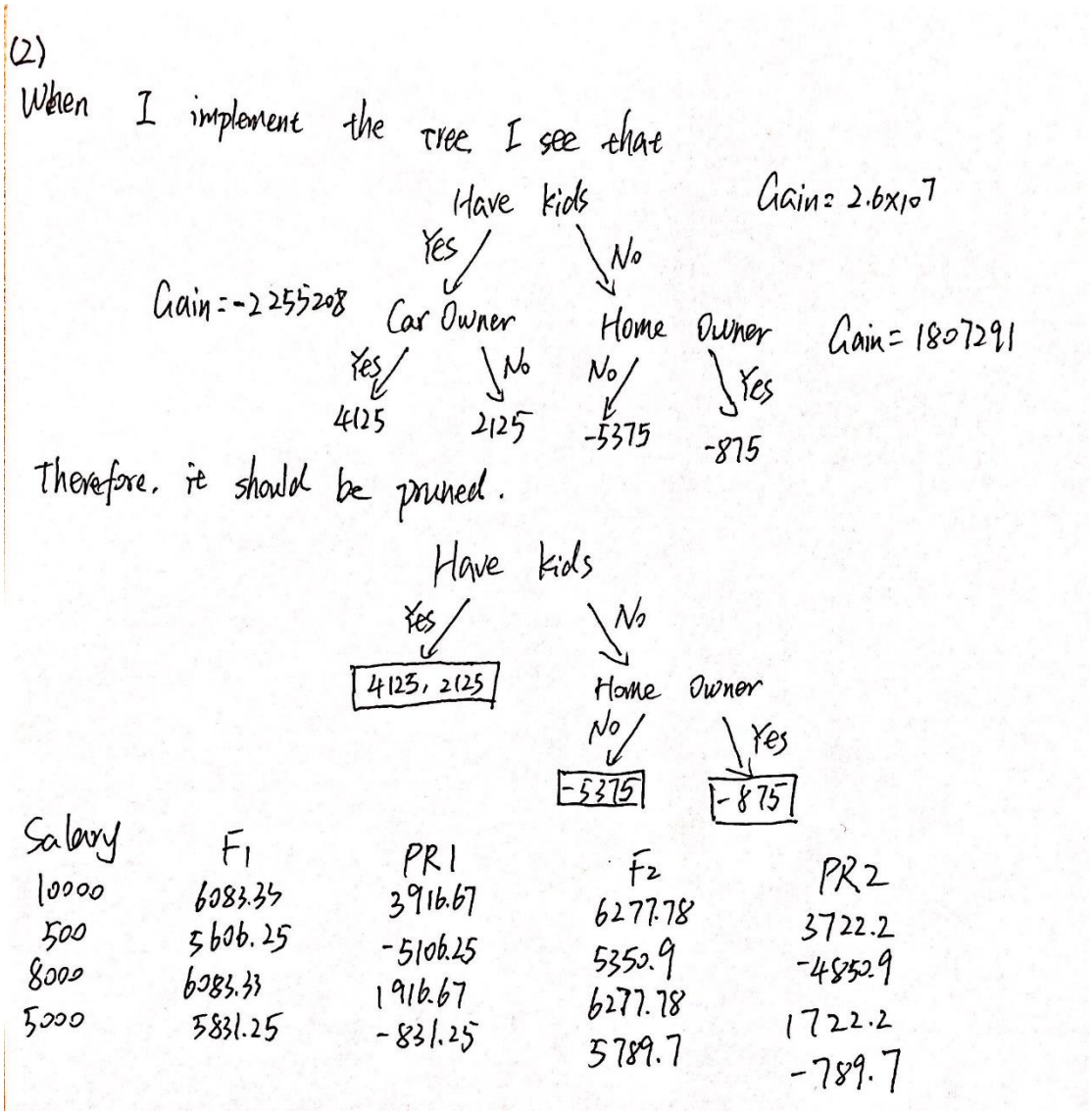
(1)

The tree should be



Deviance = 1.21×10^7

Salary	F0	PR0	F1	PR1	F2	PR2
10000	5875	4125	6287.5	3712.5	6658.75	3341.25
500	5875	-5375	5337.5	-4837.5	4858.75	-4353.75
8000	5875	2125	6087.5	1912.5	6278.75	1721.25
5000	5875	-875	5787.5	-787.5	5708.75	-708.75



3.

(1)

For Linear Regression, the result is:

```
R2 sq:  0.6083953032289722
Mean squared error: 0.53
Test Variance score: 0.60
```

And the coefficients of features are (the input has been normalized):

HouseAge	Latitude	AveOccup	Longitude	Population	MedInc	AveRooms	AveBedrms

```
model.coef_
```

```
array([[  0.47803918,   -3.99331079,   -4.33923437,   -4.40246164,
         -0.19455488,    6.33030675,  -15.47310652,   21.11153934]])
```

We can see that House age and Population weight least and AveBedrms weights most. This is probably because that the houses in CA are built during the same time period, and the

public traffic there is very convenient.

(2)
For Gradient Boosting Machine, the result is:

```
R2 sq:  0.8391271074833364
Mean squared error: 0.25
Test Variance score: 0.81
```

And the importance of features is:

AveOccup Latitude Longitude AveBedrms Population HouseAge MedInc AveRooms

```
model.feature_importances_
```

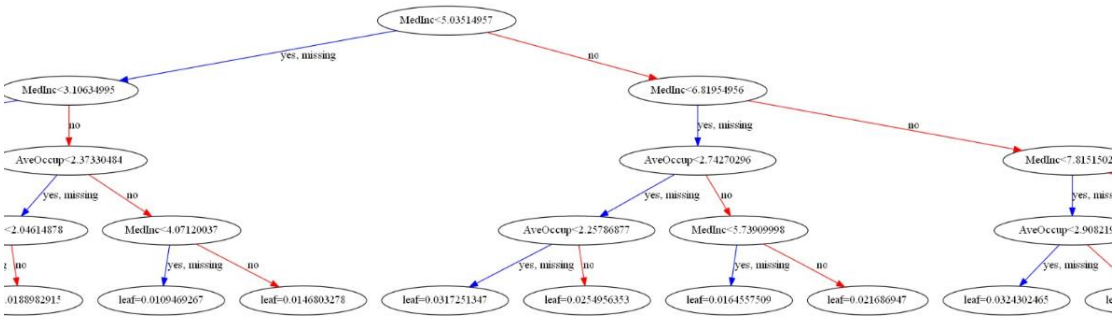
```
array([0.13055554, 0.09604826, 0.10855568, 0.0074221 , 0.00636641,
       0.04285832, 0.58718297, 0.02101071])
```

It shows that Median income weights most, which is much more than others. This is reasonable because the median income decides the consumption level, which decides the house price.

(3)
For XGBoosting, the result is:

```
Mean squared error: 0.30
Test Variance score: 0.77
```

The decision tree is:



The importance is:

HouseAge AveBedrms MedInc Latitude AveOccup AveRooms Longitude Population

```
xg_reg.feature_importances_
```

```
array([0.06191953, 0.04073134, 0.40880224, 0.09781619, 0.15168677,
       0.13000618, 0.09757712, 0.01146065], dtype=float32)
```

From the importance, we see that it mainly depends on MedInc, AveOccup and AveRooms. They are the only three factors which appear in the decision tree. This model is a little much simple than the GBM, so that it is more stable (lower in variance) but also a little inaccurate (higher in MSE).

APPENDIX

Python Cod:

Linear Regression:

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn import linear_model
from sklearn.model_selection import train_test_split


# In[2]:


dataset = pd.read_csv('C:/Users/69009/Desktop/last/441/problem/CALI.csv
')
print(dataset)


# In[3]:


feature_names={"MedInc","HouseAge","AveRooms","AveBedrms","Population",
"AveOccup","Latitude","Longitude"}
df_x = pd.DataFrame(dataset, columns = feature_names)
df_y = pd.DataFrame(dataset, columns = {"HOUSING PRICE"})
df_x


# In[4]:


for column in df_x.columns:
    df_x[column] = (df_x[column] - df_x[column].min()) / (df_x[column].
max() - df_x[column].min())
df_x


# In[5]:
```

```
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_si
ze = 0.2, random_state = 4)


# In[6]:


model = linear_model.LinearRegression()
model.fit(x_train,y_train)
results = model.predict(x_test)
print(results)


# In[7]:


model.coef_


# In[8]:


from sklearn.metrics import mean_squared_error, r2_score
model_score = model.score(x_train,y_train)
print('R2 sq: ', model_score)
# The mean squared error
print("Mean squared error: %.2f"% mean_squared_error(y_test, results))
# Explained variance score: 1 is perfect prediction
print('Test Variance score: %.2f' % r2_score(y_test, results))


# In[9]:


print(np.c_[y_test.values, results][0:5,:])


# In[ ]:
```

GBM:

```
#!/usr/bin/env python
# coding: utf-8
```

```
# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn import ensemble
from sklearn import tree


# In[2]:


dataset = pd.read_csv('C:/Users/69009/Desktop/last/441/problem/CALI.csv
')
feature_names={"MedInc","HouseAge","AveRooms","AveBedrms","Population",
"AveOccup","Latitude","Longitude"}
df_x = pd.DataFrame(dataset, columns = feature_names)
df_y = pd.DataFrame(dataset, columns = {"HOUSING PRICE"})
for column in df_x.columns:
    df_x[column] = (df_x[column] - df_x[column].min()) / (df_x[column].
max() - df_x[column].min())
x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_si
ze = 0.2, random_state = 4)
df_x


# In[3]:


params = {'n_estimators': 1000, 'max_depth': 4, 'min_samples_split': 2,
 'learning_rate': 0.01, 'loss': 'ls'}
model = ensemble.GradientBoostingRegressor(**params)

model.fit(x_train, y_train)


# In[4]:


model.feature_importances_


# In[5]:
```

```
from sklearn.metrics import mean_squared_error, r2_score
model_score = model.score(x_train,y_train)
print('R2 sq: ',model_score)

y_predicted = model.predict(x_test)
# The mean squared error
print("Mean squared error: %.2f"% mean_squared_error(y_test, y_predicte
d))
# Explained variance score: 1 is perfect prediction
print('Test Variance score: %.2f' % r2_score(y_test, y_predicted))


# In[6]:


from sklearn.model_selection import cross_val_predict

fig, ax = plt.subplots()
ax.scatter(y_test, y_predicted, edgecolors=(0, 0, 0))
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k-
-', lw=4)
ax.set_xlabel('Actual')
ax.set_ylabel('Predicted')
ax.set_title("Ground Truth vs Predicted")
plt.show()


# In[ ]:
```

XBG:

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score

from sklearn import ensemble
import xgboost as xgb


# In[2]:


dataset = pd.read_csv('C:/Users/69009/Desktop/last/441/problem/CALI.csv
')
feature_names={"MedInc","HouseAge","AveRooms","AveBedrms","Population",
"AveOccup","Latitude","Longitude"}
df_x = pd.DataFrame(dataset, columns = feature_names)
df_y = pd.DataFrame(dataset, columns = {"HOUSING PRICE"})
for column in df_x.columns:
    df_x[column] = (df_x[column] - df_x[column].min()) / (df_x[column].
max() - df_x[column].min())
df_x


# In[3]:


x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_si
ze = 0.2, random_state = 4)


# In[4]:


params = {'n_estimators': 500, "objective":"reg:linear",'colsample_bytr
ee': 0.5,'learning_rate': 0.01,
                'max_depth': 4, 'alpha': 1}
xg_reg = xgb.XGBRegressor(**params)

xg_reg.fit(x_train,y_train)
y_predicted = xg_reg.predict(x_test)


# In[5]:


xg_reg.feature_importances_
```

```
# In[6]:


print("Mean squared error: %.2f"% mean_squared_error(y_test, y_predicte
d))
# Explained variance score: 1 is perfect prediction
print('Test Variance score: %.2f' % r2_score(y_test, y_predicted))


# In[8]:


params = {"objective":"reg:linear",'colsample_bytree': 0.5,'learning_ra
te': 0.01, 'max_depth': 4, 'alpha': 1}
data_dmatrix = xgb.DMatrix(data=df_x,label=df_y)
xg_reg = xgb.train(params=params, dtrain=data_dmatrix, num_boost_round=
10)
xgb.plot_tree(xg_reg,num_trees=9)
plt.rcParams['figure.figsize'] = (70.0, 70.0)
plt.savefig('C:/Users/69009/Desktop/last/441/problem/XGB/fig1.jpg')
plt.show()


# In[ ]:
```