In [42]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression as LR
```

In [93]:
```
def DoKFold(model,X,y,k,random_state=146,scaler=None):
    '''Function will perform K-fold validation and return a list of K training and testing scores, incl
duing R^2 as well as MSE.

        Inputs:
            model: An sklearn model with defined 'fit' and 'score' methods
            X: An N by p array containing the features of the model.  The N rows are observations, and
 the p columns are features.
            y: An array of length N containing the target of the model
            k: The number of folds to split the data into for K-fold validation
            random_state: used when splitting the data into the K folds (default=146)
            scaler: An sklearn feature scaler.  If none is passed, no feature scaling will be performed
        Outputs:
            train_scores: A list of length K containing the training scores
            test_scores: A list of length K containing the testing scores
            train_mse: A list of length K containing the MSE on training data
            test_mse: A list of length K containing the MSE on testing data
    '''

    from sklearn.model_selection import KFold
    kf = KFold(n_splits=k,shuffle=True,random_state=random_state)

    train_scores=[]
    test_scores=[]
    train_mse=[]
    test_mse=[]

    for idxTrain, idxTest in kf.split(X):
        Xtrain = X[idxTrain,:]
        Xtest = X[idxTest,:]
        ytrain = y[idxTrain]
        ytest = y[idxTest]

        if scaler != None:
            Xtrain = scaler.fit_transform(Xtrain)
            Xtest = scaler.transform(Xtest)

        model.fit(Xtrain,ytrain)

        train_scores.append(model.score(Xtrain,ytrain))
        test_scores.append(model.score(Xtest,ytest))

        # Compute the mean squared errors
        ytrain_pred = model.predict(Xtrain)
        ytest_pred = model.predict(Xtest)
        train_mse.append(np.mean((ytrain-ytrain_pred)**2))
        test_mse.append(np.mean((ytest-ytest_pred)**2))

    return train_scores,test_scores,train_mse,test_mse

def CompareClasses(actual, predicted, names=None):
    '''Function returns a confusion matrix, and overall accuracy given:
            Input:  actual - a list of actual classifications
                    predicted - a list of predicted classifications
                    names (optional) - a list of class names
    '''
    import pandas as pd
    accuracy = sum(actual==predicted)/actual.shape[0]
    classes = pd.DataFrame(columns=['Actual','Predicted'])
    classes['Actual'] = actual
    classes['Predicted'] = predicted
    conf_mat = pd.crosstab(classes['Predicted'],classes['Actual'])
    # Relabel the rows/columns if names was provided
    if type(names) != type(None):
        conf_mat.index=y_names
        conf_mat.index.name='Predicted'
        conf_mat.columns=y_names
        conf_mat.columns.name = 'Actual'
    print('Accuracy = ' + format(accuracy, '.2f'))
    return conf_mat, accuracy

def GetColors(N, map_name='rainbow'):
    '''Function returns a list of N colors from a matplotlib colormap
            Input: N = number of colors, and map_name = name of a matplotlib colormap

            For a list of available colormaps:
                https://matplotlib.org/3.1.1/gallery/color/colormap_reference.html
    '''
    import matplotlib
    cmap = matplotlib.cm.get_cmap(name=map_name)
    n = np.linspace(0,N,N)/N
    colors = cmap(n)
    return colors

def PlotGroups(points, groups, colors, ec='black', ax='None'):
    '''Function makes a scatter plot, given:
            Input:  points (array)
                    groups (an integer label for each point)
                    colors (one rgb tuple for each group)
                    ec (edgecolor for markers, default is black)
                    ax (optional handle to an existing axes object to add the new plot on top of)
            Output: handles to the figure (fig) and axes (ax) objects
    '''
    import matplotlib.pyplot as plt
    import numpy as np

    # Create a new plot, unless something was passed for 'ax'
    if ax == 'None':
        fig,ax = plt.subplots()
    else:
        fig = plt.gcf()

    for i in np.unique(groups):
        idx = (groups==i)
        ax.scatter(points[idx,0], points[idx,1],color=colors[i],
                   ec=ec,alpha=0.5,label = 'Group ' + str(i))
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')
    ax.legend(bbox_to_anchor=[1, 0.5], loc='center left')
    return fig, ax
```

In [44]:
```
df = pd.read_csv('./data/lbr_persons.csv')
```

In [45]:
```
df.head()
```

Out[45]:

	Unnamed: 0	location	size	wealth	gender	age	education
0	1	6	11	1	1	54	2
1	2	6	3	4	1	37	3
2	3	6	10	4	2	50	0
3	4	6	1	3	1	32	2
4	5	6	7	3	1	36	0

In [46]:
```
df = df.set_index('Unnamed: 0')
```

In [47]:
```
df.head()
print(df.shape)
```

```
(48219, 6)
```

In [48]:
```
#seperate X and y variables
X = df.copy().drop('education', axis = 1)
y = df.copy()['education']
```

In [172]:
```
y.mean()
```

Out[172]:
```
0.6653393890375163
```

In [173]:
```
y.median()
```

Out[173]:
```
0.0
```

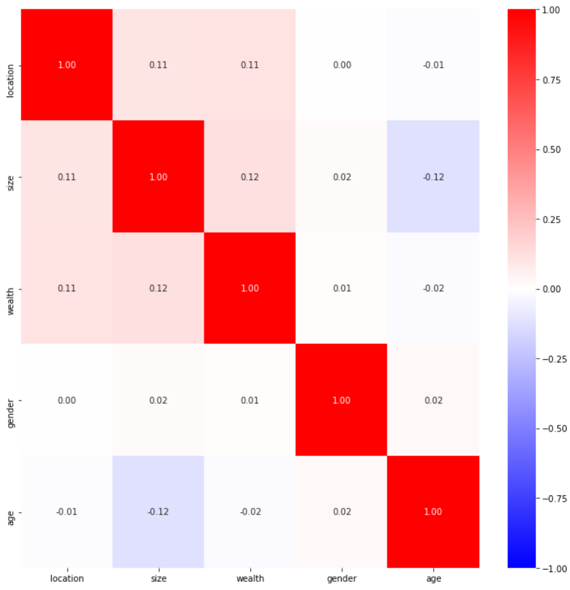
In [179]:
```
print(y.unique())
```

```
[2 3 0 1 9 8]
```

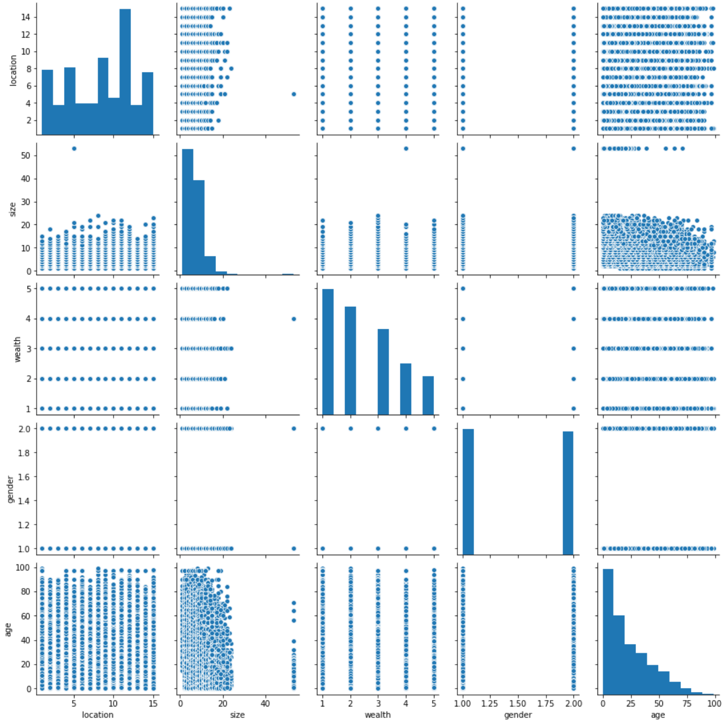
Look at the Data

In [191]:
```
#Look at a heat map of the X data
import seaborn as sns
plt.figure(figsize=(12,12))

sns.heatmap(X.corr(), xticklabels=X.columns,yticklabels=X.columns,vmin=-1,vmax=1,cmap='bwr',annot=True
,fmt='.2f')
plt.savefig('corr_heat.png')
plt.show()
```



In [189]:
```
import seaborn as sns
sns.pairplot(X[['location', 'size', 'wealth', 'gender', 'age']])
plt.savefig('pair_plot.png')
plt.show()
```



In [50]:
```
#split data into training and testing date
from sklearn.model_selection import train_test_split as tts
```

In [51]:
```
Xtrain, Xtest, ytrain, ytest = tts(X,y, test_size=.4, random_state=146)
```

In [52]:
```
#fit and train logisitic regression on raw data
lr = LR(random_state=146, max_iter=5000)
lr.fit(Xtrain,ytrain)
y_pred = lr.predict(Xtest)
```

In [53]:
```
#create a dictionaries to put all the accuracies in a dictionary
accuracies = dict()
```

In [54]:
```
from sklearn.metrics import accuracy_score
#compute accurarcy of LR on raw data
acc = accuracy_score(ytest,y_pred)
print('Accuracy of LR on Raw tts data: ' + str(acc))
accuracies['Raw'] = acc
```

```
Accuracy of LR on Raw tts data: 0.5702509332227291
```

In [55]:
```
#Now standardize the dat using minmax ss
from sklearn.preprocessing import MinMaxScaler as MMS
```

In [56]:
```
mms = MMS()
Xtrain_mms = mms.fit_transform(Xtrain)
Xtest_mms = mms.transform(Xtest)
```

In [57]:
```
#compute accuracy for mms data
lr = LR(random_state=146, max_iter=5000)
lr.fit(Xtrain_mms,ytrain)
y_pred = lr.predict(Xtest_mms)
```

In [58]:
```
acc = accuracy_score(ytest,y_pred)
accuracies['MMS']= acc
print('Accuracy of LR on mms: ' + str(acc))
```

```
Accuracy of LR on mms: 0.5701990875155537
```

In [59]:
```
# use Standard Scaler instead
from sklearn.preprocessing import StandardScaler as SS
ss = SS()
Xtrain_ss = ss.fit_transform(Xtrain)
Xtest_ss = ss.transform(Xtest)
```

In [60]:
```
lr = LR(random_state=146, max_iter=5000)
lr.fit(Xtrain_ss,ytrain)
y_pred = lr.predict(Xtest_ss)
acc = accuracy_score(ytest,y_pred)
accuracies['SS']= acc
print('Accuracy of LR on ss: ' + str(acc))
```

```
Accuracy of LR on ss: 0.5703027789299046
```

In [61]:
```
#Use RobustScaler
from sklearn.preprocessing import RobustScaler as RS
rs = RS()
Xtrain_rs = rs.fit_transform(Xtrain)
Xtest_rs = rs.transform(Xtest)
```

In [62]:
```
lr = LR(random_state=146, max_iter=5000)
lr.fit(Xtrain_rs,ytrain)
y_pred = lr.predict(Xtest_rs)
acc = accuracy_score(ytest,y_pred)
accuracies['RS']= acc
print('Accuracy of LR on rs: ' + str(acc))
```

```
Accuracy of LR on rs: 0.5701990875155537
```

In [63]:
```
#Use Normalizer
from sklearn.preprocessing import Normalizer as NZ
nz = NZ()
Xtrain_nz = rs.fit_transform(Xtrain)
Xtest_nz = rs.transform(Xtest)
```

In [64]:
```
lr = LR(random_state=146, max_iter=5000)
lr.fit(Xtrain_nz,ytrain)
y_pred = lr.predict(Xtest_nz)
acc = accuracy_score(ytest,y_pred)
accuracies['NZ']= acc
print('Accuracy of LR on rs: ' + str(acc))
```

```
Accuracy of LR on rs: 0.5701990875155537
```

In [65]:
```
accuracies
```

Out[65]:
```
{'Raw': 0.5702509332227291,
 'MMS': 0.5701990875155537,
 'SS': 0.5703027789299046,
 'RS': 0.5701990875155537,
 'NZ': 0.5701990875155537}
```

In [66]:
```
max_lracc = max(accuracies.values())
max_lracc
```

Out[66]:
```
0.5703027789299046
```

In [70]:
```
#perform a kfolds validation with LR and Standard Scalar to make sure it is in the norm
lr = LR(random_state=146,max_iter=5000)
ss=SS()
train_scores_ss,test_scores_ss,mse_train_scores_ss,mse_test_ss = DoKFold(lr,X.values,y.values,20, scale
r=ss)
```

In [75]:
```
import numpy as np
print('Avg train score on lr_ss: ' + str(np.mean(train_scores_ss)))
print('Avg test score on lr_ss: ' + str(np.mean(test_scores_ss)))
```

```
Avg train score on lr_ss: 0.5723579156895722
Avg test score on lr_ss: 0.5722226964586584
```

In [ ]:
```
#perform a lr on tsne data for rr
```

In [76]:
```
#perform k-fold on raw data
lr = LR(random_state=146,max_iter=5000)
train_scores_raw,test_scores_raw,mse_train_scores_raw,mse_test_raw = DoKFold(lr,X.values,y.values,20)
print('Avg train score on lr_raw: ' + str(np.mean(train_scores_raw)))
print('Avg test score on lr_raw: ' + str(np.mean(test_scores_raw)))
```

```
Avg train score on lr_raw: 0.5723557326647731
Avg test score on lr_raw: 0.5722226878535619
```

The maximum accuracy for a logisitic regression was reached with the use of Standard scaler. In fact, doing K-fold validation, we see that it has an even heigher training and testing score than any of the accuracies reported for the other cases, including running on raw data

Perform K-NN model prediction

In [95]:
```
values, counts = np.unique(ytrain, return_counts = True)
```

In [96]:
```
values
```

Out[96]:
```
array([0, 1, 2, 3, 8, 9])
```

In [97]:
```
counts
```

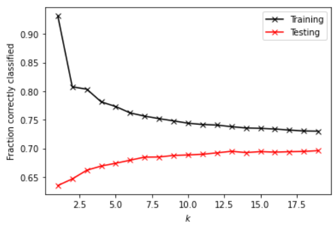
Out[97]:
```
array([16170,  7212,  4975,   522,     1,    51])
```

In [101]:
```
cols = ['location', 'size', 'wealth', 'gender', 'age']
```

In [125]:
```
#perfom a K-NN model on raw data, find best k from a range of 0-15
from sklearn.neighbors import KNeighborsClassifier as KNN
k_range = range(1,20)
ktest_raw_scores = dict()
ktrain_raw_scores = dict()
for k in k_range:
    knn = KNN(n_neighbors = k) #, weights='distance'
    knn.fit(Xtrain,ytrain)
    acc_train = knn.score(Xtrain,ytrain)
    ktrain_raw_scores[k]=acc_train
    y_pred = knn.predict(Xtest)
    acc = accuracy_score(ytest,y_pred)
    ktest_raw_scores[k] = acc
##Raw scores to better without inverse weighting
```

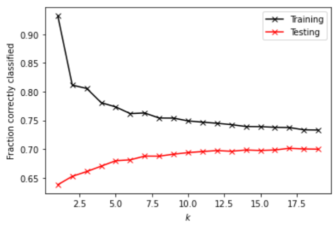
In [132]:
```
lis_train = []
for val in ktrain_raw_scores.values():
    lis_train.append(val)
lis_test=[]
for val in ktest_raw_scores.values():
    lis_test.append(val)
```

```
In [133]: #graph comparison
          k_range = np.arange(1, 20, 1)
          plt.plot(k_range, lis_train,'-xk', label='Training')
          plt.plot(k_range,lis_test,'-xr', label='Testing')
          plt.xlabel('$k$')
          plt.ylabel('Fraction correctly classified')
          plt.legend()
          plt.show()
          #I would use 6/7 neighbors
```



```
In [139]: #find best number of neibors for ss
          from sklearn.neighbors import KNeighborsClassifier as KNN
          k_range = np.arange(1,20,1)
          Xtrain_ss = ss.fit_transform(Xtrain)
          Xtest_ss = ss.transform(Xtest)
          ktrain_ss_scores =[]
          ktest_ss_scores = []
          for k in k_range:
              knn = KNN(n_neighbors = k)
              knn.fit(Xtrain_ss,ytrain)
              acc_train = knn.score(Xtrain_ss,ytrain)
              ktrain_ss_scores.append(acc_train)
              y_pred = knn.predict(Xtest_ss)
              acc = accuracy_score(ytest,y_pred)
              ktest_ss_scores.append(acc)

          #plot scores
          plt.plot(k_range, ktrain_ss_scores,'-xk', label='Training')
          plt.plot(k_range,ktest_ss_scores,'-xr', label='Testing')
          plt.xlabel('$k$')
          plt.ylabel('Fraction correctly classified')
          plt.legend()
          plt.show()
          #again use 6
```

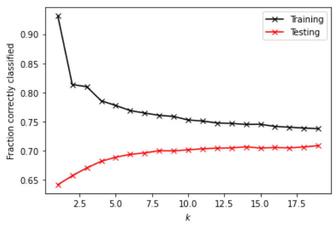


```
In [143]: print(ktest_ss_scores[6]) #so ss at 6 is better
          print(lis_test[6])
          print(ktest_mms_scores[6])

          0.6876814599751141
          0.6847781003732891
          0.6961323102447118
```

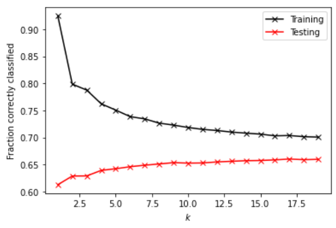
```
In [190]: #do same for minmax
          k_range = np.arange(1,20,1)
          mms = MMS()
          Xtrain_mms = mms.fit_transform(Xtrain)
          Xtest_mms = mms.transform(Xtest)
          ktrain_mms_scores =[]
          ktest_mms_scores = []
          for k in k_range:
              knn = KNN(n_neighbors = k)
              knn.fit(Xtrain_mms,ytrain)
              acc_train = knn.score(Xtrain_mms,ytrain)
              ktrain_mms_scores.append(acc_train)
              y_pred = knn.predict(Xtest_mms)
              acc = accuracy_score(ytest,y_pred)
              ktest_mms_scores.append(acc)

          #plot scores
          plt.plot(k_range, ktrain_mms_scores,'-xk', label='Training')
          plt.plot(k_range,ktest_mms_scores,'-xr', label='Testing')
          plt.xlabel('$k$')
          plt.ylabel('Fraction correctly classified')
          plt.legend()
          plt.savefig('KNN_N_Neighbors.png')
          plt.show()
          #again use 6
```



```
In [144]: #KNN for Nz
          #do same for minmax
          k_range = np.arange(1,20,1)
          nz = NZ()
          Xtrain_nz = nz.fit_transform(Xtrain)
          Xtest_nz = nz.transform(Xtest)
          ktrain_nz_scores =[]
          ktest_nz_scores = []
          for k in k_range:
              knn = KNN(n_neighbors = k)
              knn.fit(Xtrain_nz,ytrain)
              acc_train = knn.score(Xtrain_nz,ytrain)
              ktrain_nz_scores.append(acc_train)
              y_pred = knn.predict(Xtest_nz)
              acc = accuracy_score(ytest,y_pred)
              ktest_nz_scores.append(acc)

          #plot scores
          plt.plot(k_range, ktrain_nz_scores,'-xk', label='Training')
          plt.plot(k_range,ktest_nz_scores,'-xr', label='Testing')
          plt.xlabel('$k$')
          plt.ylabel('Fraction correctly classified')
          plt.legend()
          plt.show()
          #again use 6
```

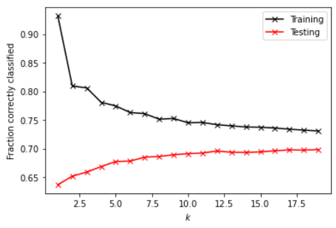


```
In [145]: ktest_nz_scores[6]

Out[145]: 0.6488490253007051
```

```
In [146]: #KNN for RS
          k_range = np.arange(1,20,1)
          rs = RS()
          Xtrain_rs = rs.fit_transform(Xtrain)
          Xtest_rs = rs.transform(Xtest)
          ktrain_rs_scores =[]
          ktest_rs_scores = []
          for k in k_range:
              knn = KNN(n_neighbors = k)
              knn.fit(Xtrain_rs,ytrain)
              acc_train = knn.score(Xtrain_rs,ytrain)
              ktrain_rs_scores.append(acc_train)
              y_pred = knn.predict(Xtest_rs)
              acc = accuracy_score(ytest,y_pred)
              ktest_rs_scores.append(acc)

          #plot scores
          plt.plot(k_range, ktrain_rs_scores,'-xk', label='Training')
          plt.plot(k_range,ktest_rs_scores,'-xr', label='Testing')
          plt.xlabel('$k$')
          plt.ylabel('Fraction correctly classified')
          plt.legend()
          plt.show()
          ktest_nz_scores[6]
```



```
Out[146]: 0.6488490253007051
```

```
In [194]: #perform kfold for n=6 mms to ensure validaty
          knn = KNN(n_neighbors=6)
          k_range = np.arange(2,20,1)
          mms = MMS()
          train_scores_mms_knn,test_scores_mms_knn,mse_train_scores_mms_knn,mse_test_mms_knn = DoKFold(knn,X.val
          ues,y.values,20,scaler=mms)
          print('Avg train score on lr_raw: ' + str(np.mean(train_scores_mms_knn)))
          print('Avg test score on lr_raw: ' + str(np.mean(test_scores_mms_knn)))
          #the average k-fold testing checks out

          Avg train score on lr_raw: 0.7697468012474872
          Avg test score on lr_raw: 0.6959496068331351
```

```
In [ ]:
```

Decision Trees

In [160]:
```python
from sklearn.tree import DecisionTreeClassifier as DTC
from sklearn import tree

dt_raw_scores = []
d_range = np.arange(1,20, 1)
m_range = np.arange(2,20,1)
for d in d_range:
    for i in m_range:
        params = [d,i]
        dt = DTC(random_state=146, max_depth=d,min_samples_split=i)
        dt.fit(Xtrain,ytrain)
        score = dt.score(Xtest,ytest)
        dt_raw_scores.append([*params,score])
dt_raw_scores
```

Out[160]:
```
[[1, 2, 0.5526752384902531],
 [1, 3, 0.5526752384902531],
 [1, 4, 0.5526752384902531],
 [1, 5, 0.5526752384902531],
 [1, 6, 0.5526752384902531],
 [1, 7, 0.5526752384902531],
 [1, 8, 0.5526752384902531],
 [1, 9, 0.5526752384902531],
 [1, 10, 0.5526752384902531],
 [1, 11, 0.5526752384902531],
 [1, 12, 0.5526752384902531],
 [1, 13, 0.5526752384902531],
 [1, 14, 0.5526752384902531],
 [1, 15, 0.5526752384902531],
 [1, 16, 0.5526752384902531],
 [1, 17, 0.5526752384902531],
 [1, 18, 0.5526752384902531],
 [1, 19, 0.5526752384902531],
 [2, 2, 0.62811074243052681],
 [2, 3, 0.62811074243052681],
 [2, 4, 0.62811074243052681],
 [2, 5, 0.62811074243052681],
 [2, 6, 0.62811074243052681],
 [2, 7, 0.62811074243052681],
 [2, 8, 0.62811074243052681],
 [2, 9, 0.62811074243052681],
 [2, 10, 0.62811074243052681],
 [2, 11, 0.62811074243052681],
 [2, 12, 0.62811074243052681],
 [2, 13, 0.62811074243052681],
 [2, 14, 0.62811074243052681],
 [2, 15, 0.62811074243052681],
 [2, 16, 0.62811074243052681],
 [2, 17, 0.62811074243052681],
 [2, 18, 0.62811074243052681],
 [2, 19, 0.62811074243052681],
 [3, 2, 0.6779344670261302],
 [3, 3, 0.6779344670261302],
 [3, 4, 0.6779344670261302],
 [3, 5, 0.6779344670261302],
 [3, 6, 0.6779344670261302],
 [3, 7, 0.6779344670261302],
 [3, 8, 0.6779344670261302],
 [3, 9, 0.6779344670261302],
 [3, 10, 0.6779344670261302],
 [3, 11, 0.6779344670261302],
 [3, 12, 0.6779344670261302],
 [3, 13, 0.6779344670261302],
 [3, 14, 0.6779344670261302],
 [3, 15, 0.6779344670261302],
 [3, 16, 0.6779344670261302],
 [3, 17, 0.6779344670261302],
 [3, 18, 0.6779344670261302],
 [3, 19, 0.6779344670261302],
 [4, 2, 0.683481957693903],
 [4, 3, 0.683481957693903],
 [4, 4, 0.683481957693903],
 [4, 5, 0.683481957693903],
 [4, 6, 0.683481957693903],
 [4, 7, 0.683481957693903],
 [4, 8, 0.683481957693903],
 [4, 9, 0.683481957693903],
 [4, 10, 0.683481957693903],
 [4, 11, 0.683481957693903],
 [4, 12, 0.683481957693903],
 [4, 13, 0.683481957693903],
 [4, 14, 0.683481957693903],
 [4, 15, 0.683481957693903],
 [4, 16, 0.683481957693903],
 [4, 17, 0.683481957693903],
 [4, 18, 0.683481957693903],
 [4, 19, 0.683481957693903],
 [5, 2, 0.705360846121941],
 [5, 3, 0.705360846121941],
 [5, 4, 0.705360846121941],
 [5, 5, 0.705360846121941],
 [5, 6, 0.705360846121941],
 [5, 7, 0.705360846121941],
 [5, 8, 0.705360846121941],
 [5, 9, 0.705360846121941],
 [5, 10, 0.705360846121941],
 [5, 11, 0.705360846121941],
 [5, 12, 0.705360846121941],
 [5, 13, 0.705360846121941],
 [5, 14, 0.705360846121941],
 [5, 15, 0.705360846121941],
 [5, 16, 0.705360846121941],
 [5, 17, 0.705360846121941],
 [5, 18, 0.705360846121941],
 [5, 19, 0.705360846121941],
 [6, 2, 0.7128784736623808],
 [6, 3, 0.7128784736623808],
 [6, 4, 0.7128784736623808],
 [6, 5, 0.7128784736623808],
 [6, 6, 0.7128784736623808],
 [6, 7, 0.7128784736623808],
 [6, 8, 0.7128784736623808],
 [6, 9, 0.7128784736623808],
 [6, 10, 0.7128784736623808],
 [6, 11, 0.7128784736623808],
 [6, 12, 0.7128784736623808],
 [6, 13, 0.713137702198258],
 [6, 14, 0.713137702198258],
 [6, 15, 0.713137702198258],
 [6, 16, 0.713137702198258],
 [6, 17, 0.713137702198258],
 [6, 18, 0.713137702198258],
 [6, 19, 0.7130858564910826],
 [7, 2, 0.7178556615512236],
 [7, 3, 0.7178556615512236],
 [7, 4, 0.7178556615512236],
 [7, 5, 0.7178556615512236],
 [7, 6, 0.717907507258399],
 [7, 7, 0.7178556615512236],
 [7, 8, 0.7178556615512236],
 [7, 9, 0.7178556615512236],
 [7, 10, 0.7178556615512236],
 [7, 11, 0.7178556615512236],
 [7, 12, 0.7178556615512236],
 [7, 13, 0.7181148900871008],
 [7, 14, 0.7181148900871008],
 [7, 15, 0.7181148900871008],
 [7, 16, 0.7181148900871008],
 [7, 17, 0.7181148900871008],
 [7, 18, 0.7181148900871008],
 [7, 19, 0.7180630443799253],
 [8, 2, 0.7143301534632932],
 [8, 3, 0.714537536291995],
 [8, 4, 0.7144856905848196],
 [8, 5, 0.714537536291995],
 [8, 6, 0.7145893819991704],
 [8, 7, 0.7145893819991704],
 [8, 8, 0.7145893819991704],
 [8, 9, 0.714537536291995],
 [8, 10, 0.714537536291995],
 [8, 11, 0.714537536291995],
 [8, 12, 0.7145893819991704],
 [8, 13, 0.7149004562422231],
 [8, 14, 0.7149004562422231],
 [8, 15, 0.7149004562422231],
 [8, 16, 0.7149004562422231],
 [8, 17, 0.7153152218996267],
 [8, 18, 0.7153152218996267],
 [8, 19, 0.7152633761924513],
 [9, 2, 0.7146930734135214],
 [9, 3, 0.7145893819991704],
 [9, 4, 0.7144338448776442],
 [9, 5, 0.7149004562422231],
 [9, 6, 0.7146412277063459],
 [9, 7, 0.7146412277063459],
 [9, 8, 0.7146412277063459],
 [9, 9, 0.7145893819991704],
 [9, 10, 0.7144338448776442],
 [9, 11, 0.7146930734135214],
 [9, 12, 0.7147967648278722],
 [9, 13, 0.7150559933637495],
 [9, 14, 0.7153670676068021],
 [9, 15, 0.7153670676068021],
 [9, 16, 0.7153152218996267],
 [9, 17, 0.7157299875570303],
 [9, 18, 0.7158336789713812],
 [9, 19, 0.7157818332642057],
 [10, 2, 0.7103380340107839],
 [10, 3, 0.7099751140605558],
 [10, 4, 0.7103380340107839],
 [10, 5, 0.7107527996681875],
 [10, 6, 0.7103380340107839],
 [10, 7, 0.7101824968892576],
 [10, 8, 0.710234342596433],
 [10, 9, 0.710234342596433],
 [10, 10, 0.7100269597677312],
 [10, 11, 0.710234342596433],
 [10, 12, 0.7102861883036085],
 [10, 13, 0.7107527996681875],
 [10, 14, 0.7113231024471174],
 [10, 15, 0.7113749481542928],
 [10, 16, 0.7113231024471174],
 [10, 17, 0.7118415595188718],
 [10, 18, 0.7119970966403982],
 [10, 19, 0.7120489423475737],
 [11, 2, 0.7028722521775197],
 [11, 3, 0.7030277892990461],
 [11, 4, 0.7030796350062215],
 [11, 5, 0.7029759435918705],
 [11, 6, 0.7030796350062215],
 [11, 7, 0.7027167150559933],
 [11, 8, 0.7029240978846951],
 [11, 9, 0.7031314807133969],
 [11, 10, 0.7031833264205724],
 [11, 11, 0.703961012028204],
 [11, 12, 0.7041683948569059],
 [11, 13, 0.7042720862712567],
 [11, 14, 0.7045831605143095],
 [11, 15, 0.7044794690999585],
 [11, 16, 0.7044276233927831],
 [11, 17, 0.7052571547075902],
 [11, 18, 0.7052571547075902],
 [11, 19, 0.7053090004147656],
 [12, 2, 0.7027167150559933],
 [12, 3, 0.7032870178349233],
 [12, 4, 0.7024574865201161],
 [12, 5, 0.7027685607631688],
 [12, 6, 0.7031314807133969],
 [12, 7, 0.7029759435918705],
 [12, 8, 0.7029240978846951],
 [12, 9, 0.7032351721277478],
 [12, 10, 0.7031314807133969],
 [12, 11, 0.704064703442555],
 [12, 12, 0.7033907092492742],
 [12, 13, 0.7034425549564496],
 [12, 14, 0.7043239319784322],
 [12, 15, 0.7042202405640813],
 [12, 16, 0.7042202405640813],
 [12, 17, 0.705360846121941],
 [12, 18, 0.705931148900871],
 [12, 19, 0.705931148900871],
 [13, 2, 0.691051430941518],
 [13, 3, 0.6912588137702198],
 [13, 4, 0.6909995852343426],
 [13, 5, 0.691155122355869],
 [13, 6, 0.691155122355869],
 [13, 7, 0.6927104935711323],
 [13, 8, 0.6923475736209042],
 [13, 9, 0.6937992534218167],
 [13, 10, 0.6943177104935712],
 [13, 11, 0.6960286188303608],
 [13, 12, 0.6956138531729573],
 [13, 13, 0.6958730817088346],
 [13, 14, 0.6971692243882207],
 [13, 15, 0.6974284529240978],
 [13, 16, 0.6973766072169224],
 [13, 17, 0.6989838241393612],
 [13, 18, 0.6993467440895894],
 [13, 19, 0.6995541269182912],
 [14, 2, 0.683378266279552],
 [14, 3, 0.6837411862297802],
 [14, 4, 0.6835856491082538],
 [14, 5, 0.6840004147656574],
 [14, 6, 0.6843114890087101],
 [14, 7, 0.6849854832019909],
 [14, 8, 0.6860223973454997],
 [14, 9, 0.6865927001224297],
 [14, 10, 0.6870593114890087],
 [14, 11, 0.6887702198257984],
 [14, 12, 0.689755288262132],
 [14, 13, 0.6901700539195355],
 [14, 14, 0.691621733720448],
 [14, 15, 0.6921920364993779],
 [14, 16, 0.6928660306926586],
 [14, 17, 0.693954790543343],
 [14, 18, 0.6946287847366238],
 [14, 19, 0.6957175445873082],
 [15, 2, 0.6782455412691829],
 [15, 3, 0.6786603069265865],
 [15, 4, 0.6789195354624638],
 [15, 5, 0.6792824554126918],
 [15, 6, 0.6799046038987971],
 [15, 7, 0.6814081294068851],
 [15, 8, 0.6823931978432186],
 [15, 9, 0.6830671920364993],
 [15, 10, 0.6854002488593944],
 [15, 11, 0.6869037743674824],
 [15, 12, 0.6885109912899212],
 [15, 13, 0.6888220655329739],
 [15, 14, 0.6907403566984653],
 [15, 15, 0.6914661965989216],
 [15, 16, 0.6914143508917462],
 [15, 17, 0.6930734135213604],
 [15, 18, 0.6937992534218167],
 [15, 19, 0.6947324761509747],
 [16, 2, 0.6678245541269183],
 [16, 3, 0.6689133139776027],
 [16, 4, 0.669535462463708],
 [16, 5, 0.6688096225632517],
 [16, 6, 0.6705723766072169],
 [16, 7, 0.672438822065533],
 [16, 8, 0.6742534218166736],
 [16, 9, 0.6753940273745334],
 [16, 10, 0.6772604728328494],
 [16, 11, 0.680578598092078],
 [16, 12, 0.6824968892575695],
 [16, 13, 0.6827042720862713],
 [16, 14, 0.6843633347158855],
 [16, 15, 0.685244711737868],
 [16, 16, 0.6855039402737453],
 [16, 17, 0.6870074657818332],
 [16, 18, 0.688148071339693],
 [16, 19, 0.6894442140190792],
 [17, 2, 0.665232268768146],
 [17, 3, 0.6670468685192866],
 [17, 4, 0.66860223973455],
 [17, 5, 0.66803193695562],
 [17, 6, 0.6705205309000415],
 [17, 7, 0.6715574450435504],
 [17, 8, 0.6725425134798839],
 [17, 9, 0.6748237245956035],
 [17, 10, 0.6769493985897967],
 [17, 11, 0.6794898382413936],
 [17, 12, 0.6814599751140605],
 [17, 13, 0.682082123600166],
 [17, 14, 0.6843114890087101],
 [17, 15, 0.6851928660306926],
 [17, 16, 0.6851928660306926],
 [17, 17, 0.6875259228535877],
 [17, 18, 0.6881999170468686],
 [17, 19, 0.6892886768975529],
 [18, 2, 0.6588034010783907],
 [18, 3, 0.6586478639568644],
 [18, 4, 0.6607216922438822],
 [18, 5, 0.6594773952716715],
 [18, 6, 0.6622252177519702],
 [18, 7, 0.6653878058896724],
 [18, 8, 0.6669950228121112],
 [18, 9, 0.6689651596847781],
 [18, 10, 0.6723351306511821],
 [18, 11, 0.6753940273745334],
 [18, 12, 0.677675238490253],
 [18, 13, 0.6788158440481128],
 [18, 14, 0.6814081294068851],
 [18, 15, 0.6823413521360432],
 [18, 16, 0.6831708834508503],
 [18, 17, 0.6851410203235172],
 [18, 18, 0.6862297801742016],
 [18, 19, 0.6874222314392369],
 [19, 2, 0.6500414765657404],
 [19, 3, 0.6525300705101618],
 [19, 4, 0.6535151389464953],
 [19, 5, 0.654292824554127],
 [19, 6, 0.6577146412277064],
 [19, 7, 0.6617586063873911],
 [19, 8, 0.6630547490667773],
 [19, 9, 0.6655433430111987],
 [19, 10, 0.669068851099129],
 [19, 11, 0.6710908336789714],
 [19, 12, 0.6744608046453754],
 [19, 13, 0.6757569473247615],
 [19, 14, 0.679178763998341],
 [19, 15, 0.6800601410203235],
 [19, 16, 0.6809933637494815],
 [19, 17, 0.6828598092077975],
 [19, 18, 0.6841559518871837],
 [19, 19, 0.6847781003732891]]
```

In [159]:
```python
dt = DTC(random_state=146)
dt.fit(Xtrain,ytrain)
dt.score(Xtest,ytest)
```

Out[159]: 0.6419535462463708

In [ ]:

In [181]:
```python
dt = DTC(random_state=146, max_depth=9,min_samples_split=18)
dt_train,dt_test,_,_ = DoKFold(dt,X.values,y.values,20)
print(np.mean(dt_train))
print(np.mean(dt_test))
```

```
0.7326343295614747
0.7204213743716127
```

In [171]:
```python
dt = DTC(random_state=146, max_depth=1)
dt.fit(Xtrain,ytrain)
tree.plot_tree(dt,feature_names=list(X.columns));
```