18120205004-SAFİYE SENA MERDİN

19120205029-RAVZANUR CANTÜRK

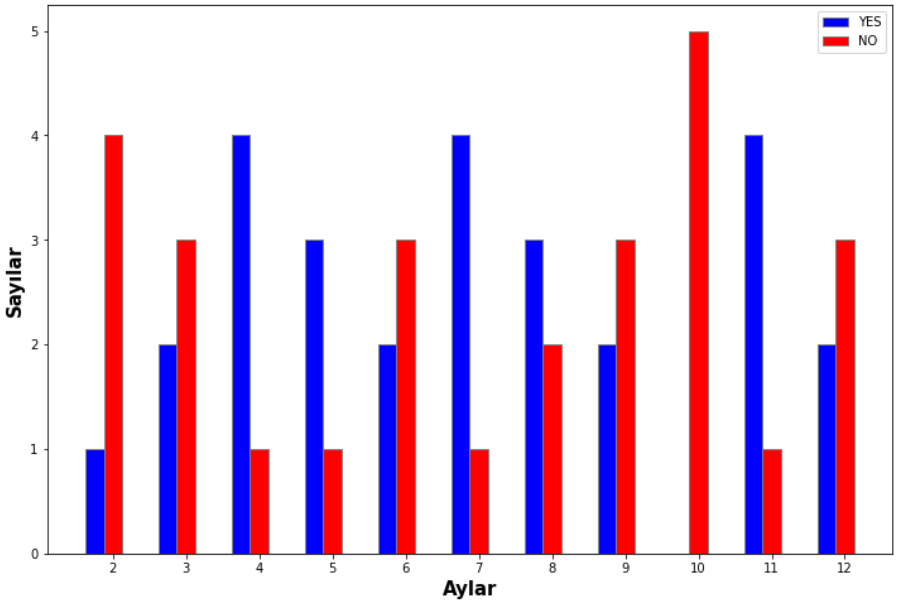
19120205015-SELCEN FETHİYE MERSİNLİ

SORU: 2020 datasının aylara göre gruplanmış verilerinin "retail_and_recreation_percent_change_from_baseline", "parks_percent_change_from_baseline", "residential_percent_change_from_baseline" sütunlarının her aylarını negatif ve pozitif olarak classifice edin. Pozitif olma durumu insan yoğunluğunun belirtilen alanlarda fazla negatif olma durumu ise az olduğunu gösterir. Buna göre en az 5 kişilik anket yapınız ve çoğunluğun cevabına göre "go_out" classını oluşturunuz. Oluşan datayı train ve test data olarak ikiye ayırınız. Elde edilen bu dataları ve desicion tree methodunu kullanarak karar ağacı oluşturup dışarı çıkılıp çıkılmayacağına karar veriniz. En son accuracy hesaplatınız.

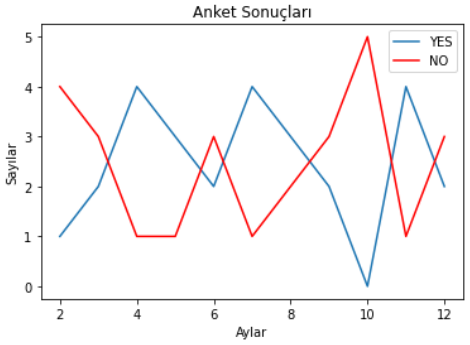
AYLARA GÖRE DIŞARI ÇIKMA/ÇIKMAMA TERCİH ANKETİ SONUÇLARI

```
Şubat   Ayı : No  | No  | No  | Yes | No
Mart    Ayı : No  | No  | Yes | No  | Yes
Nisan   Ayı : No  | Yes | Yes | Yes | Yes
Mayıs   Ayı : Yes | No  | Yes | No  | Yes
Haziran Ayı : Yes | No  | Yes | No  | No
Temmuz  Ayı : No  | Yes | Yes | Yes | Yes
Ağustos Ayı : No  | Yes | Yes | Yes | No
Eylül   Ayı : Yes | No  | No  | Yes | No
Ekim    Ayı : No  | No  | No  | No  | No
Kasım   Ayı : Yes | Yes | Yes | No  | Yes
Aralık  Ayı : No  | No  | Yes | Yes | No
```

In [362]:
```
#histogram graph of survey results
import numpy as np
import matplotlib.pyplot as plt

# set width of bar
barWidth = 0.25
fig = plt.subplots(figsize =(12, 8))

# set height of bar
YES = [1, 2, 4, 3, 2, 4, 3, 2, 0, 4, 2]
NO = [4, 3, 1, 1, 3, 1, 2, 3, 5, 1, 3]

# Set position of bar on X axis
br1 = np.arange(len(YES))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, YES, color ='b', width = barWidth,
        edgecolor ='grey', label ='YES')
plt.bar(br2, NO, color ='r', width = barWidth,
        edgecolor ='grey', label ='NO')

# Adding Xticks
plt.xlabel('Aylar', fontweight ='bold', fontsize = 15)
plt.ylabel('Sayılar', fontweight ='bold', fontsize = 15)
plt.xticks([r + barWidth for r in range(len(YES))],
        [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

plt.legend()
plt.show()
```



In [363]:
```
#the line graph of survey results
from matplotlib import pyplot as plt
plt.plot([2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], [1, 2, 4, 3, 2, 4, 3, 2, 0, 4, 2], label="YES")
plt.plot([2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], [4, 3, 1, 1, 3, 1, 2, 3, 5, 1, 3], label="NO", color='r')
plt.legend()
plt.xlabel('Aylar')
plt.ylabel('Sayılar')
plt.title('Anket Sonuçları')
plt.show()
```



In [364]:
```
import pandas as pd
import numpy as np
import math
```

In [365]:
```
url = 'https://drive.google.com/file/d/18gyHbx6rfogq3yQ-GR9COjcGgyYlCnBZ/view?usp=sharing'
url2020 = 'https://drive.google.com/uc?id=' + url.split('/')[-2]
```

In [366]:
```
df = pd.read_csv(url2020)
```

In [367]:
```
#dropping columns that be not used
to_drop = ['metro_area', 'census_fips_code', 'grocery_and_pharmacy_percent_change_from_baseline',
           'transit_stations_percent_change_from_baseline', 'workplaces_percent_change_from_baseline']

df.drop(to_drop, inplace=True, axis=1)
```

In [368]:
```
#grouping the months by indexing
df.index = pd.to_datetime(df[df.columns[6]])
monthly = df.groupby(pd.Grouper(freq='M')).mean()
```

In [369]:
```
monthly
```

Out[369]:

date	retail_and_recreation_percent_change_from_baseline	parks_percent_change_from_baseline	residential_percent_change_from_baseline
2020-02-29	2.028130	4.509374	-0.698443
2020-03-31	-21.700010	-6.153226	6.332819
2020-04-30	-63.148836	-42.117354	20.742720
2020-05-31	-55.673896	-28.837900	17.455508
2020-06-30	-18.156210	32.502002	4.339668
2020-07-31	-6.876458	55.750799	-0.082189
2020-08-31	-9.014545	60.977582	-0.669343
2020-09-30	-13.071653	38.977984	2.389219
2020-10-31	-13.590351	30.077698	3.886670
2020-11-30	-26.801938	-1.853678	8.382888
2020-12-31	-47.541929	-26.385514	15.067144

In [370]:
```
#classification of monthly 2020 data values and create label column by using survey results
columns = ['retail_and_recreation_percent_change_from_baseline', 'parks_percent_change_from_baseline',
           'residential_percent_change_from_baseline']
data = pd.DataFrame()
name = ['RR', 'PP', 'R']
for i in range(0,3):
    liste = []
    for j in range(0,11):
        if monthly.iloc[j][i] < 0:
            liste.append(name[i] + "-Negative")
        else:
            liste.append(name[i] + "-Positive")
    data[columns[i]] = liste
liste_go_out = ['NO', 'NO', 'YES', 'YES', 'NO', 'YES', 'YES', 'NO', 'NO', 'YES', 'NO']
data['go_out'] = liste_go_out
```

In [371]:
```
#show number of yes and no for each attribute
pd.crosstab([data['retail_and_recreation_percent_change_from_baseline'],data['parks_percent_change_from_baseline'],
             data['residential_percent_change_from_baseline']],data['go_out'])
```

Out[371]:

			go_out	NO	YES
retail_and_recreation_percent_change_from_baseline	parks_percent_change_from_baseline	residential_percent_change_from_baseline			
RR-Negative	PP-Negative	R-Positive		2	3
	PP-Positive	R-Negative		0	2
		R-Positive		3	0
RR-Positive	PP-Positive	R-Negative		1	0

In [372]:
```
#generating train and test data
train_data_m = data.sample(6) #importing train dataset into dataframe
test_data_m = data.drop(train_data_m.index) #importing test dataset into dataframe
```

In [373]:
```
data
```

Out[373]:

	retail_and_recreation_percent_change_from_baseline	parks_percent_change_from_baseline	residential_percent_change_from_baseline	go_out
0	RR-Positive	PP-Positive	R-Negative	NO
1	RR-Negative	PP-Negative	R-Positive	NO
2	RR-Negative	PP-Negative	R-Positive	YES
3	RR-Negative	PP-Negative	R-Positive	YES
4	RR-Negative	PP-Positive	R-Positive	NO
5	RR-Negative	PP-Positive	R-Negative	YES
6	RR-Negative	PP-Positive	R-Negative	YES
7	RR-Negative	PP-Positive	R-Positive	NO
8	RR-Negative	PP-Positive	R-Positive	NO
9	RR-Negative	PP-Negative	R-Positive	YES
10	RR-Negative	PP-Negative	R-Positive	NO

In [374]:
```
train_data_m
```

Out[374]:

	retail_and_recreation_percent_change_from_baseline	parks_percent_change_from_baseline	residential_percent_change_from_baseline	go_out
0	RR-Positive	PP-Positive	R-Negative	NO
5	RR-Negative	PP-Positive	R-Negative	YES
7	RR-Negative	PP-Positive	R-Positive	NO
9	RR-Negative	PP-Negative	R-Positive	YES
3	RR-Negative	PP-Negative	R-Positive	YES
8	RR-Negative	PP-Positive	R-Positive	NO

In [375]:
```
test_data_m
```

Out[375]:

	retail_and_recreation_percent_change_from_baseline	parks_percent_change_from_baseline	residential_percent_change_from_baseline	go_out
1	RR-Negative	PP-Negative	R-Positive	NO
2	RR-Negative	PP-Negative	R-Positive	YES
4	RR-Negative	PP-Positive	R-Positive	NO
6	RR-Negative	PP-Positive	R-Negative	YES
10	RR-Negative	PP-Negative	R-Positive	NO

In [392]:
```
#total entropy
def calc_total_entropy(train_data, label, class_list):
    total_row = train_data.shape[0] #the total size of the dataset
    total_entr = 0

    for c in class_list: #for each class in the label
        total_class_count = train_data[train_data[label] == c].shape[0] #number of the class
        total_class_entr = - (total_class_count/total_row)*np.log2(total_class_count/total_row) #entropy of the class
        total_entr += total_class_entr #adding the class entropy to the total entropy of the dataset

    return total_entr
```

In [393]:
```
#entropy for each attribute
def calc_entropy(feature_value_data, label, class_list):
    class_count = feature_value_data.shape[0]
    entropy = 0

    for c in class_list:
        label_class_count = feature_value_data[feature_value_data[label] == c].shape[0] #row count of class c
        entropy_class = 0
        if label_class_count != 0:
            probability_class = label_class_count/class_count #probability of the class
            entropy_class = - probability_class * np.log2(probability_class)  #entropy
        entropy += entropy_class
    return entropy
```

In [394]:
```
#information gain for each attribute
def calc_info_gain(feature_name, train_data, label, class_list):
    feature_value_list = train_data[feature_name].unique() #unqiue values of the feature
    total_row = train_data.shape[0]
    feature_info = 0.0

    for feature_value in feature_value_list:
        feature_value_data = train_data[train_data[feature_name] == feature_value] #filtering rows with that feature_value
        feature_value_count = feature_value_data.shape[0]
        feature_value_entropy = calc_entropy(feature_value_data, label, class_list) #calculcating entropy for the feature value
        feature_value_probability = feature_value_count/total_row
        feature_info += feature_value_probability * feature_value_entropy #calculating information of the feature value

    return calc_total_entropy(train_data, label, class_list) - feature_info #calculating information gain by subtracting
```

In [395]:
```
#find most informative attribute
def find_most_informative_feature(train_data, label, class_list):
    feature_list = train_data.columns.drop(label) #finding the feature names in the dataset
                                          #N.B. label is not a feature, so dropping it
    max_info_gain = -1
    max_info_feature = None

    for feature in feature_list:  #for each feature in the dataset
        feature_info_gain = calc_info_gain(feature, train_data, label, class_list)
        if max_info_gain < feature_info_gain: #selecting feature name with highest information gain
            max_info_gain = feature_info_gain
            max_info_feature = feature

    return max_info_feature
```

In [396]:
```
#generate sub tree
def generate_sub_tree(feature_name, train_data, label, class_list):
    #dictionary of the count of unqiue feature value
    feature_value_count_dict = train_data[feature_name].value_counts(sort=False)

    tree = {} #sub tree or node

    for feature_value, count in feature_value_count_dict.iteritems():
        #dataset with only feature_name = feature_value
        feature_value_data = train_data[train_data[feature_name] == feature_value]

        assigned_to_node = False #flag for tracking feature_value is pure class or not
        for c in class_list: #for each class
            class_count = feature_value_data[feature_value_data[label] == c].shape[0] #count of class c

            if class_count == count: #count of feature_value = count of class (pure class)
                tree[feature_value] = c #adding node to the tree
                train_data = train_data[train_data[feature_name] != feature_value] #removing rows with feature_value
                assigned_to_node = True
        if not assigned_to_node: #not pure class
            tree[feature_value] = "?" #should extend the node, so the branch is marked with ?

    return tree, train_data
```

In [397]:
```
#recursive method that creates tree
def make_tree(root, prev_feature_value, train_data, label, class_list):
    if train_data.shape[0] != 0: #if dataset becomes enpty after updating
        max_info_feature = find_most_informative_feature(train_data, label, class_list) #most informative feature
        tree, train_data = generate_sub_tree(max_info_feature, train_data, label, class_list)
        #getting tree node and updated dataset
        next_root = None

        if prev_feature_value != None: #add to intermediate node of the tree
            root[prev_feature_value] = dict()
            root[prev_feature_value][max_info_feature] = tree
            next_root = root[prev_feature_value][max_info_feature]
        else: #add to root of the tree
            root[max_info_feature] = tree
            next_root = root[max_info_feature]

        for node, branch in list(next_root.items()): #iterating the tree node
            if branch == "?": #if it is expandable
                feature_value_data = train_data[train_data[max_info_feature] == node] #using the updated dataset
                make_tree(next_root, node, feature_value_data, label, class_list) #recursive call with updated dataset
```

In [398]:
```
#main function
def id3(train_data_m, label):
    train_data = train_data_m.copy() #getting a copy of the dataset
    tree = {} #tree which will be updated
    class_list = train_data[label].unique() #getting unqiue classes of the label
    make_tree(tree, None, train_data_m, label, class_list) #start calling recursion
    return tree
```

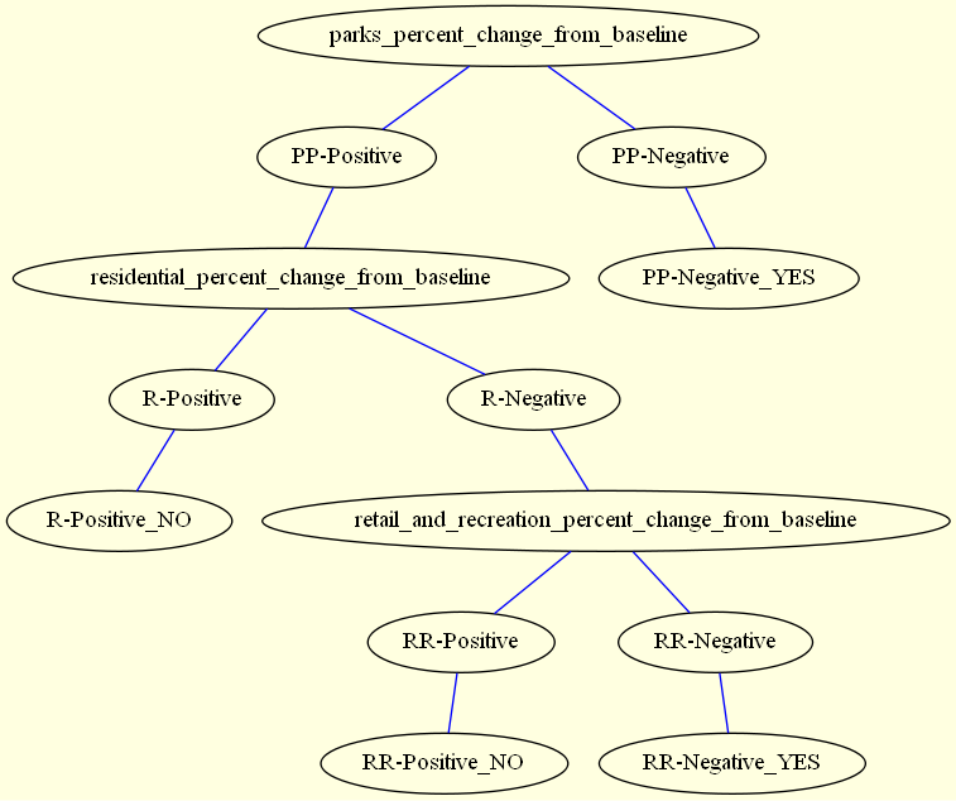
In [399]:
```
tree = id3(train_data_m, 'go_out')
```

In [400]:
```
tree
```

Out[400]:
```
{'parks_percent_change_from_baseline': {'PP-Positive': {'residential_percent_change_from_baseline': {'R-Positive': 'NO',
    'R-Negative': {'retail_and_recreation_percent_change_from_baseline': {'RR-Positive': 'NO',
      'RR-Negative': 'YES'}}}},
  'PP-Negative': 'YES'}}
```

In [401]:
```
#visualization tree
import pydot
from IPython.display import Image as img


def draw(parent_name, child_name):
    edge = pydot.Edge(parent_name, child_name, color="blue")
    graph.add_edge(edge)

def visit(node, parent=None):
    for k,v in node.items():
        if isinstance(v, dict):
            # We start with the root node whose parent is None
            # we don't want to graph the None node
            if parent:
                draw(parent, k)
            visit(v, k)
        else:
            draw(parent, k)
            # drawing the label using a distinct name
            draw(k, k+'_'+v)

graph = pydot.Dot(graph_type='graph', bgcolor="lightyellow")
visit(tree)
graph.write_png('example1_graph.png')
img("./example1_graph.png")
```

Out[401]:



In [402]:
```
def predict(tree, instance):
    if not isinstance(tree, dict): #if it is leaf node
        return tree #return the value
    else:
        root_node = next(iter(tree)) #getting first key/feature name of the dictionary
        feature_value = instance[root_node] #value of the feature
        if feature_value in tree[root_node]: #checking the feature value in current tree node
            return predict(tree[root_node][feature_value], instance) #goto next feature
        else:
            return None
```

In [403]:
```
def evaluate(tree, test_data_m, label):
    correct_preditct = 0
    wrong_preditct = 0
    index = 0
    for row in test_data_m.iterrows(): #for each row in the dataset
        result = predict(tree, test_data_m.iloc[index]) #predict the row
        if result == test_data_m[label].iloc[index]: #predicted value and expected value is same or not
            correct_preditct += 1 #increase correct count
        else:
            wrong_preditct += 1 #increase incorrect count
        index = index + 1
    accuracy = correct_preditct / (correct_preditct + wrong_preditct) #calculating accuracy
    return accuracy
```

In [404]:
```
accuracy = evaluate(tree, test_data_m, 'go_out') #evaluating the test dataset
```

In [407]:
```
accuracy
```

Out[407]:
```
0.6
```

KAYNAKÇA

https://medium.com/geekculture/step-by-step-decision-tree-id3-algorithm-from-scratch-in-python-no-fancy-library-4822bbfdd88f (https://medium.com/geekculture/step-by-step-decision-tree-id3-algorithm-from-scratch-in-python-no-fancy-library-4822bbfdd88f)

https://www.geeksforgeeks.org/bar-plot-in-matplotlib/ (https://www.geeksforgeeks.org/bar-plot-in-matplotlib/)

https://github.com/pydot/pydot (https://github.com/pydot/pydot)

https://stackoverflow.com/questions/32370281/how-to-embed-image-or-picture-in-jupyter-notebook-either-from-a-local-machine-o (https://stackoverflow.com/questions/32370281/how-to-embed-image-or-picture-in-jupyter-notebook-either-from-a-local-machine-o)

https://stackoverflow.com/questions/13688410/dictionary-object-to-decision-tree-in-pydot (https://stackoverflow.com/questions/13688410/dictionary-object-to-decision-tree-in-pydot)

In [ ]:
```
```