In [97]:
```
import pandas as pd
import datetime

use_average = True
use_predict = True

output_name = 'average.csv' if use_average else 'normal.csv'

output_name2020 = '2020average.csv' if use_average else '2020normal.csv'
```

In [98]:
```
def read_csv(file):
    df = pd.read_csv(file)
    del df['Station Name']
    del df['Climate ID']
    del df['Longitude (x)']
    del df['Latitude (y)']
    del df['Year']
    del df['Month']
    del df['Day']
    del df['Time']


    df.columns = ['date', 'temp', 'dew', 'rel', 'wdir', 'wspeed', 'stn']
    df['date'] = pd.to_datetime(df['date'])
    return df
```

In [99]:
```
if use_predict:
    df1 = read_csv('HALIFAX DOCKYARD.csv')
    df2 = read_csv('HALIFAX KOOTENAY.csv')
else:
    df1 = read_csv('2020-HALIFAX DOCKYARD.csv')
    df2 = read_csv('2020-HALIFAX KOOTENAY.csv')

if use_average:
    data = {'date': df1['date'],
            'temp': (df1['temp'] + df2['temp'])/2.0,
            'dew': (df1['dew'] + df2['dew'])/2.0,
            'rel': (df1['rel'] + df2['rel'])/2.0,
            'wdir': (df1['wdir'] + df2['wdir'])/2.0,
            'wspeed': (df1['wspeed'] + df2['wspeed'])/2.0,
            'stn': (df1['stn'] + df2['stn'])/2.0,
            }
else:
    data = {'date': df1['date'],
            's1_temp': df1['temp'],
            's2_temp': df2['temp'],
            's1_dew': df1['dew'],
            's2_dew': df2['dew'],
            's1_rel': df1['rel'],
            's2_rel': df2['rel'],
            's1_wdir': df1['wdir'],
            's2_wdir': df2['wdir'],
            's1_wspeed': df1['wspeed'],
            's2_wspeed': df2['wspeed'],
            's1_stn': df1['stn'],
            's2_stn': df2['stn'],
            }
```

In [100]:
```
df = pd.DataFrame(data)

def str2datetime(string):
    return datetime.datetime.strptime(string, '%b %d, %Y %I%p')

if use_predict:
    headache = [
    [str2datetime('Sep 5, 2019 7AM'), str2datetime('Sep 5, 2019 10AM'), 8],
    [str2datetime('Sep 15, 2019 9AM'), str2datetime('Sep 15, 2019 3PM'), 7],
    [str2datetime('Sep 27, 2019 3PM'), str2datetime('Sep 27, 2019 9PM'), 8],
    [str2datetime('Oct 15, 2019 2PM'), str2datetime('OCT 15, 2019 6PM'), 7],
    [str2datetime('Oct 16, 2019 4AM'), str2datetime('OCt 16, 2019 6PM'), 10],
    [str2datetime('Oct 28, 2019 3PM'), str2datetime('OCT 28, 2019 9PM'), 6],
    [str2datetime('Nov 28, 2019 5PM'), str2datetime('Nov 28, 2019 6PM'), 6],
    [str2datetime('Dec 10, 2019 5AM'), str2datetime('Dec 10, 2019 10AM'), 4],
    [str2datetime('Dec 15, 2019 2AM'), str2datetime('Dec 15, 2019 10AM'), 7],
    [str2datetime('Dec 20, 2019 1PM'), str2datetime('Dec 20, 2019 6PM'), 9],
    ]

workout = [
    [str2datetime('SEP 2, 2019 6PM'), str2datetime('SEP 2, 2019 7PM')],
    [str2datetime('SEP 4, 2019 6PM'), str2datetime('SEP 4, 2019 7PM')],
    [str2datetime('SEP 6, 2019 6PM'), str2datetime('SEP 6, 2019 7PM')],
    [str2datetime('SEP 27, 2019 6PM'), str2datetime('SEP 27, 2019 7PM')],
    [str2datetime('OCT 8, 2019 6PM'), str2datetime('OCT 8, 2019 7PM')],
    [str2datetime('OCT 20, 2019 8PM'), str2datetime('OCT 20, 2019 9PM')],
    [str2datetime('OCT 27, 2019 6AM'), str2datetime('OCT 27, 2019 7AM')],
    [str2datetime('OCT 28, 2019 6PM'), str2datetime('OCT 28, 2019 7PM')],
    [str2datetime('NOV 2, 2019 6PM'), str2datetime('NOV 2, 2019 7PM')],
    [str2datetime('NOV 7, 2019 6PM'), str2datetime('NOV 7, 2019 7PM')],
    [str2datetime('NOV 9, 2019 6PM'), str2datetime('NOV 9, 2019 7PM')],
    [str2datetime('NOV 12, 2019 6PM'), str2datetime('NOV 12, 2019 7PM')],
    [str2datetime('NOV 20, 2019 6PM'), str2datetime('NOV 20, 2019 7PM')],
    [str2datetime('DEC 15, 2019 6PM'), str2datetime('DEC 15, 2019 7PM')],
    [str2datetime('DEC 16, 2019 6PM'), str2datetime('DEC 16, 2019 7PM')],
    [str2datetime('SEP 2, 2020 6PM'), str2datetime('SEP 2, 2020 7PM')],
    [str2datetime('SEP 4, 2020 6PM'), str2datetime('SEP 4, 2020 7PM')],
    [str2datetime('SEP 6, 2020 6PM'), str2datetime('SEP 6, 2020 7PM')],
]
```

In [101]:
```
df['hour'] = pd.Series([0]* df['date'].size)
df['dayweek'] = pd.Series([0]* df['date'].size)
for i in range(df['date'].size):
    df['hour'][i] = df['date'][i].hour
    df['dayweek'][i] = df['date'][i].weekday()

df['workout'] = pd.Series([0]* df['date'].size)
workout_index = []
for w in workout:
    for i in range(df['date'].size):
        if w[0] <= df['date'][i] < w[1]:
            workout_index.append(i)
df['workout'][workout_index] = 1

df['work'] = pd.Series([0]* df['date'].size)
for i in range(df['date'].size):
    if df['date'][i].weekday() == 6 or df['date'][i].weekday() == 0:
        continue
    if df['date'][i].hour < 9 or df['date'][i].hour > 17:
        continue
    if str2datetime('OCT 27, 2019 1AM') < df['date'][i] < str2datetime('NOV 3, 2019 11PM'):
        continue
    df['work'][i] = 1
df['work_hour'] = pd.Series([0] * df['date'].size)
work_sum = 0
for i in range(df['date'].size):
    if df['work'][i] == 1:
        work_sum += 1
        df['work_hour'][i] = work_sum
    else:
        work_sum = 0

if use_predict:
    df['headache'] = pd.Series([0]* df['date'].size)
    for w in headache:
        for i in range(df['date'].size):
            if df['date'][i] >= w[0] and df['date'][i] < w[1]:
                df['headache'][i] = w[2]


# del df['date']
if use_predict:
    df.to_csv(output_name)
else:
    df.to_csv(output_name2020)
```

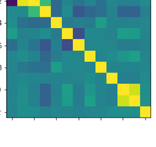
```
<ipython-input-101-79c08b35a8fb>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['hour'][i] = df['date'][i].hour
<ipython-input-101-79c08b35a8fb>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['dayweek'][i] = df['date'][i].weekday()
<ipython-input-101-79c08b35a8fb>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['workout'][workout_index] = 1
<ipython-input-101-79c08b35a8fb>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['work'][i] = 1
<ipython-input-101-79c08b35a8fb>:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['work_hour'][i] = work_sum
<ipython-input-101-79c08b35a8fb>:38: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['headache'][i] = w[2]
```

In [102]:
```
if use_predict and use_average:
    data = pd.read_csv('average.csv')
elif use_predict and not use_average:
    data = pd.read_csv('normal.csv')
elif not use_predict and use_average:
    data = pd.read_csv('2020average.csv')
else:
    data = pd.read_csv('2020normal.csv')

df = data.dropna(axis = 0 , how = 'any')

checkFor_nan = df.isnull().values.any()
print (checkFor_nan)
```

```
False
```

In [103]:
```
if use_predict and use_average:
    df.to_csv('DataSetAverage.csv')
elif use_predict and not use_average:
    df.to_csv('DataSetNormal.csv')
elif not use_predict and use_average:
    df.to_csv('PredictionDataAverage.csv')
else:
    df.to_csv('PredictionDataNormal.csv')
```

In [104]:
```
import matplotlib.pyplot as plt

plt.matshow(df.corr())
plt.show()
```



Load Libririies + Training Data, We used skLearn library for training.

In [102]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.utils.class_weight import compute_class_weight


data = pd.read_csv('DataSetAverage.csv')
```

In [103]:
```python
data
```

Out[103]:

	date	temp	dew	rel	wdir	wspeed	stn	hour	dayweek	workout	work	work_hour	headache
0	2019-09-01 0:00	16.85	11.40	70.0	31.0	12.0	101.760	0	6	0	0	0	0
1	2019-09-01 1:00	16.25	11.55	73.5	30.5	10.5	101.830	1	6	0	0	0	0
2	2019-09-01 2:00	16.10	11.30	73.5	31.0	10.0	101.890	2	6	0	0	0	0
3	2019-09-01 3:00	16.15	11.85	75.5	33.0	13.5	101.935	3	6	0	0	0	0
4	2019-09-01 4:00	15.60	11.25	75.0	33.5	15.5	101.965	4	6	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2896	2019-12-31 19:00	2.60	2.00	96.0	10.5	50.0	99.675	19	1	0	0	0	0
2897	2019-12-31 20:00	3.15	2.50	95.5	11.0	42.5	99.520	20	1	0	0	0	0
2898	2019-12-31 21:00	4.20	3.55	96.0	12.0	22.0	99.495	21	1	0	0	0	0
2899	2019-12-31 22:00	5.15	4.60	96.5	11.0	14.5	99.335	22	1	0	0	0	0
2900	2019-12-31 23:00	5.20	4.70	96.5	8.5	15.0	99.185	23	1	0	0	0	0

2901 rows × 13 columns

In [104]:
```python
#Finding Data Correlation
alpha = data.columns
data.corr()
```

Out[104]:

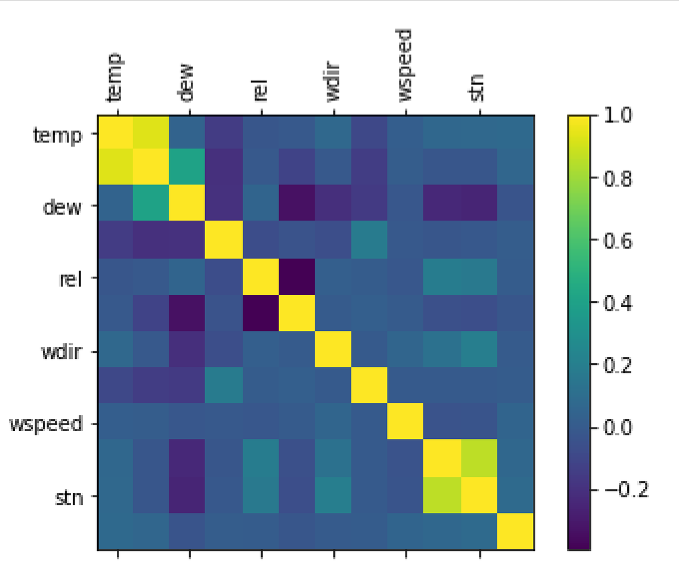
	temp	dew	rel	wdir	wspeed	stn	hour	dayweek	workout	work	work_hour	heada
temp	1.000000	0.931206	0.049493	-0.147794	-0.025635	-0.004353	0.076709	-0.093552	0.022741	0.068829	0.075488	0.082
dew	0.931206	1.000000	0.407451	-0.203143	-0.005487	-0.116894	-0.004788	-0.143230	0.014283	-0.025341	-0.025818	0.063
rel	0.049493	0.407451	1.000000	-0.198324	0.054884	-0.331662	-0.210424	-0.162037	-0.020008	-0.238168	-0.251976	-0.031
wdir	-0.147794	-0.203143	-0.198324	1.000000	-0.065561	-0.039296	-0.063278	0.183133	-0.007188	-0.025122	-0.012792	0.014
wspeed	-0.025635	-0.005487	0.054884	-0.065561	1.000000	-0.396950	0.023671	0.011215	-0.018984	0.187953	0.167101	0.010
stn	-0.004353	-0.116894	-0.331662	-0.039296	-0.396950	1.000000	0.005153	0.027562	0.005779	-0.055323	-0.060976	-0.025
hour	0.076709	-0.004788	-0.210424	-0.063278	0.023671	0.005153	1.000000	0.000840	0.060581	0.123158	0.201790	0.004
dayweek	-0.093552	-0.143230	-0.162037	0.183133	0.011215	0.027562	0.000840	1.000000	0.000443	-0.003540	-0.003040	0.010
workout	0.022741	0.014283	-0.020008	-0.007188	-0.018984	0.005779	0.060581	0.000443	1.000000	-0.041690	-0.035789	0.051
work	0.068829	-0.025341	-0.238168	-0.025122	0.187953	-0.055323	0.123158	-0.003540	-0.041690	1.000000	0.858458	0.068
work_hour	0.075488	-0.025818	-0.251976	-0.012792	0.167101	-0.060976	0.201790	-0.003040	-0.035789	0.858458	1.000000	0.085
headache	0.082381	0.063103	-0.031769	0.014816	0.010035	-0.025244	0.004529	0.010346	0.051382	0.068859	0.085044	1.000

In [105]:
```python
#Date is deleted since it is not usefull for decision Tree
df = data.drop(['date'], 1)
df.columns
```

Out[105]:
```
Index(['temp', 'dew', 'rel', 'wdir', 'wspeed', 'stn', 'hour', 'dayweek',
       'workout', 'work', 'work_hour', 'headache'],
      dtype='object')
```

We checked for correlation between features to avoid redundunt feature. As you may see in correlation hitmap temp and dew temprature are highly correlated.

In [106]:
```python
alpha = data.columns
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(data.corr())
fig.colorbar(cax)

ax.set_xticklabels(['']+alpha, rotation='vertical',verticalalignment='bottom')
ax.set_yticklabels(['']+alpha)

plt.show()
plt.close()
```



In [121]:
```python
#Data with high correlation is deleted
df = data.drop(['dew', 'work','dayweek'], 1)
df.columns
```

Out[121]:
```
Index(['date', 'temp', 'rel', 'wdir', 'wspeed', 'stn', 'hour', 'workout',
       'work_hour', 'headache'],
      dtype='object')
```

In [122]:
```python
#check and remove null record data by row.
Rnan = []
for index, row in df.iterrows():
    isnanValue = row.isnull()
    if isnanValue.any():
        Rnan.append(index)

print(Rnan)
df = df.dropna(axis=0, how='any')
df.isnull().values.any()
```

```
[]
```

Out[122]:
```
False
```

In [123]:
```python
from collections import Counter
```

In [124]:
```python
X_train = df[['temp', 'rel', 'wdir', 'wspeed', 'stn', 'hour', 'workout', 'work_hour']]

y_train = df['headache']


factor = compute_class_weight('balanced', list(Counter(y_train).keys()), y_train)
class_weights = {k:v for k, v in zip(Counter(y_train).keys(), factor)}
```

```
C:\Users\reza\anaconda3\lib\site-packages\sklearn\utils\validation.py:68: FutureWarning: Pass classes
=[0, 8, 7, 10, 6, 4, 9], y=0       0
1       0
2       0
3       0
4       0
       ..
2896    0
2897    0
2898    0
2899    0
2900    0
Name: headache, Length: 2901, dtype: int64 as keyword args. From version 0.25 passing these as positi
onal arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

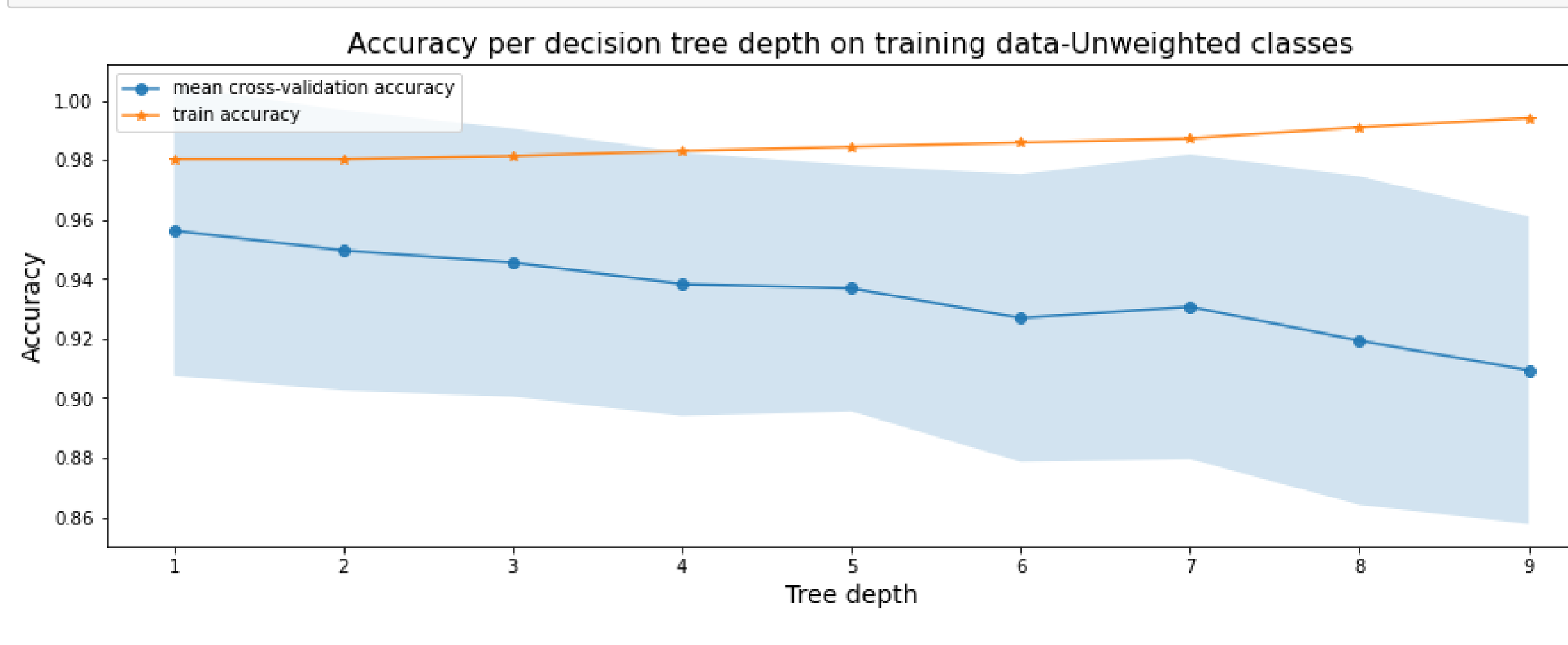
In [125]:
```python
class_weights
```

Out[125]:
```
{0: 0.14572031344183242,
 8: 46.04761904761905,
 7: 23.023809523809526,
 10: 29.602040816326532,
 6: 69.07142857142857,
 4: 82.88571428571429,
 9: 82.88571428571429}
```

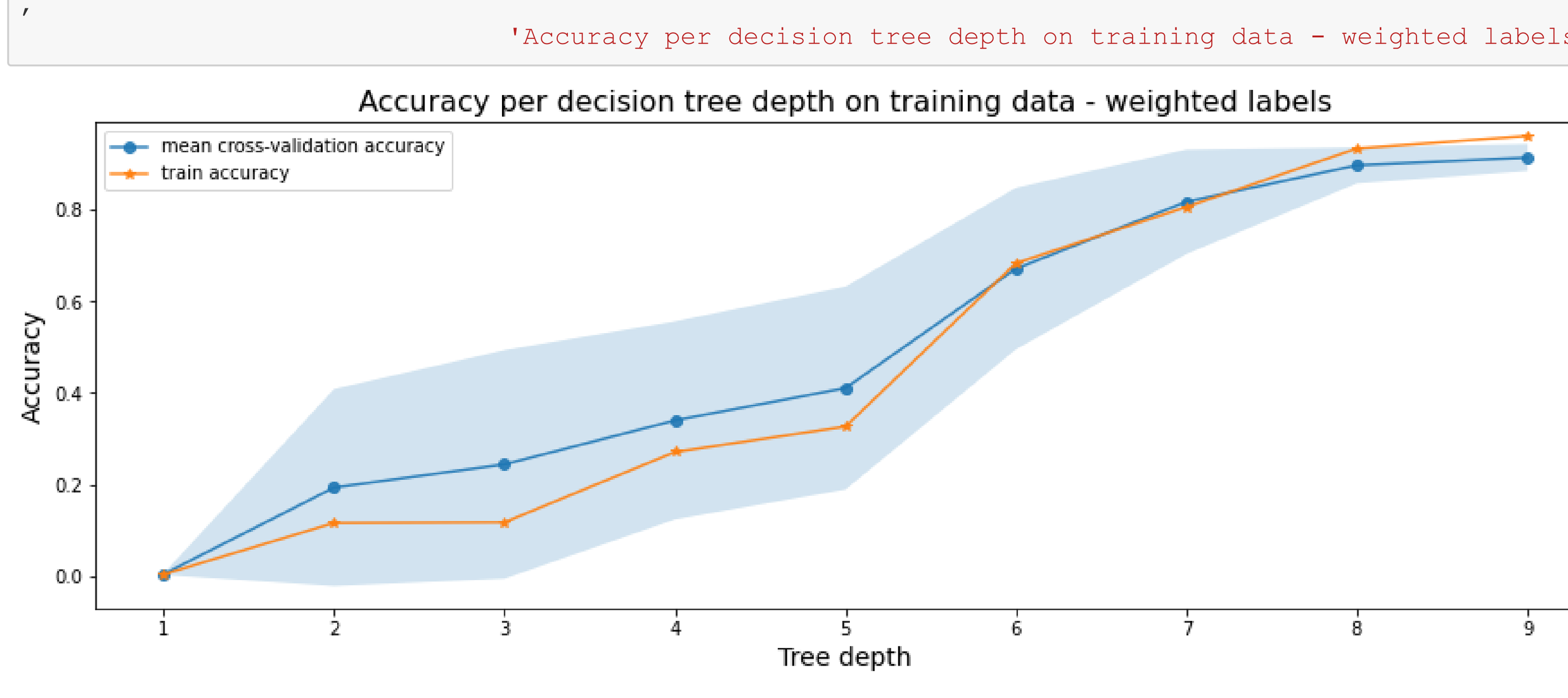
since data are unbalanced we need to pay attention more to the data that are in minority

In [158]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
import numpy as np
import matplotlib.pyplot as plt
from sklearn import tree
def run_cross_validation_on_trees(X, y, tree_depths, weights=None, cv=5, scoring='accuracy'):
    Slist = []
    Mean = []
    InputCV = []
    accuracy = []
    for depth in tree_depths:
        tree_model = DecisionTreeClassifier(max_depth=depth, class_weight=weights)
        SC = cross_val_score(tree_model, X, y, cv=cv, scoring=scoring)
        Mean.append(SC.mean())
        InputCV.append(SC.std())
        Slist.append(SC)
        accuracy.append(tree_model.fit(X, y).score(X, y))
    Mean = np.array(Mean)
    InputCV = np.array(InputCV)
    accuracy = np.array(accuracy)
    return Mean, InputCV, accuracy

def plot_cross_validation_on_trees(depths, cv_scores_mean, cv_scores_std, accuracy_scores, title):
    fig, ax = plt.subplots(1,1, figsize=(15,5))
    ax.plot(depths, cv_scores_mean, '-o', label='mean cross-validation accuracy', alpha=0.9)
    ax.fill_between(depths, cv_scores_mean-cv_scores_std, cv_scores_mean+cv_scores_std, alpha=0.2)
    ylim = plt.ylim()
    ax.plot(depths, accuracy_scores, '-*', label='train accuracy', alpha=0.9)
    ax.set_title(title, fontsize=16)
    ax.set_xlabel('Tree depth', fontsize=14)
    ax.set_ylabel('Accuracy', fontsize=14)
    ax.set_ylim(ylim)
    ax.set_xticks(depths)
    ax.legend()
```

In [159]:
```python
# fitting trees of depth 1 to 10 - unbalanced data
sm_tree_depths = range(1,10)
sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores = run_cross_validation_on_trees(X_train, y_tra
in, sm_tree_depths)

# plotting accuracy
plot_cross_validation_on_trees(sm_tree_depths, sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores
,
                               'Accuracy per decision tree depth on training data-Unweighted classes')
```



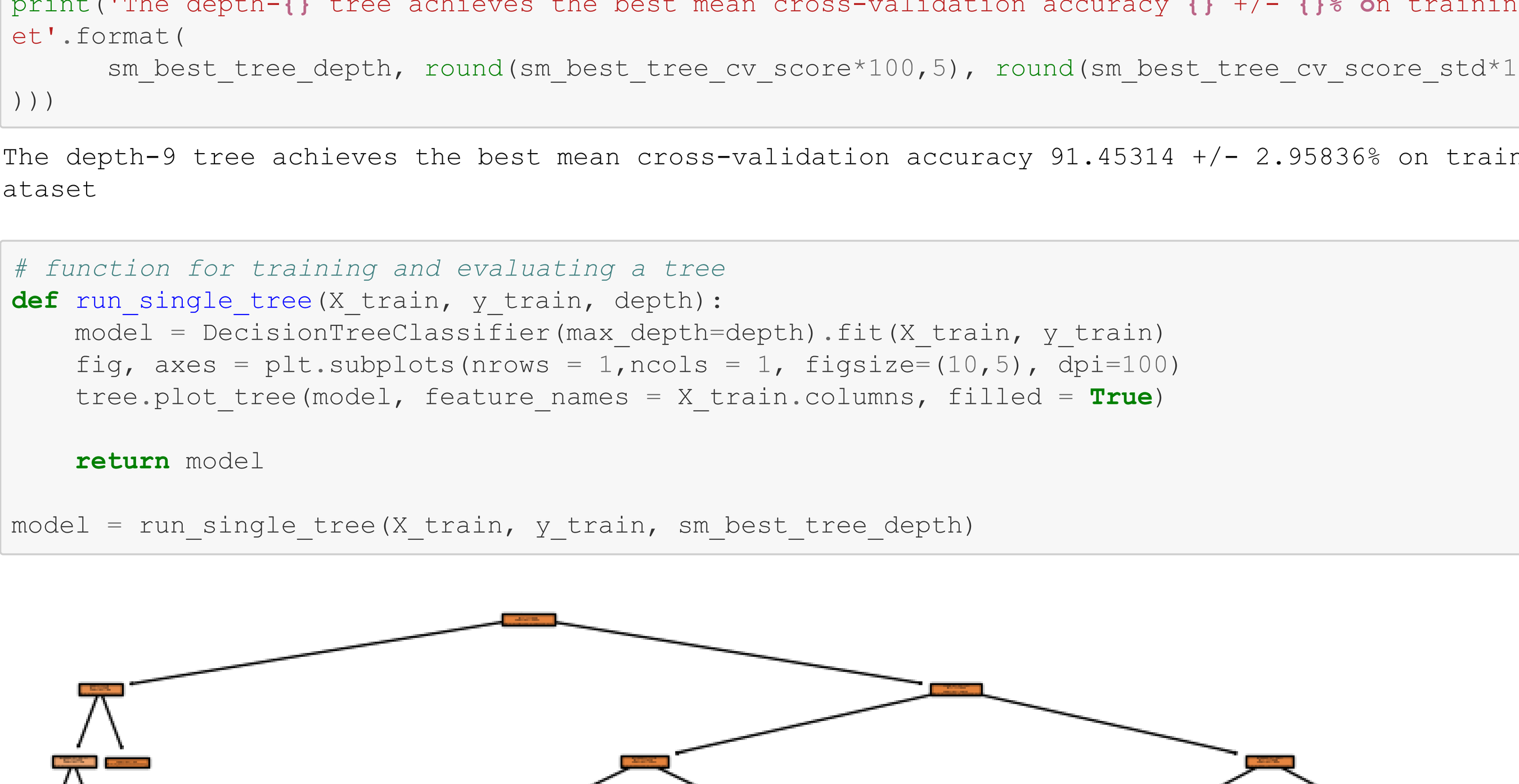
In [128]:
```python
# fitting trees of depth 1 to 10 - weighted data
sm_tree_depths = range(1,10)
sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores = run_cross_validation_on_trees(X_train, y_tra
in, sm_tree_depths, class_weights)

# plotting accuracy
plot_cross_validation_on_trees(sm_tree_depths, sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores
,
                               'Accuracy per decision tree depth on training data - weighted labels')
```



In [129]:
```python
idx_max = sm_cv_scores_mean.argmax()
sm_best_tree_depth = sm_tree_depths[idx_max]
sm_best_tree_cv_score = sm_cv_scores_mean[idx_max]
sm_best_tree_cv_score_std = sm_cv_scores_std[idx_max]
print('The depth-{} tree achieves the best mean cross-validation accuracy {} +/- {}% on training datas
et'.format(
      sm_best_tree_depth, round(sm_best_tree_cv_score*100,5), round(sm_best_tree_cv_score_std*100, 5
)))
```

```
The depth-9 tree achieves the best mean cross-validation accuracy 91.45314 +/- 2.95836% on training d
ataset
```

In [130]:
```python
# function for training and evaluating a tree
def run_single_tree(X_train, y_train, depth):
    model = DecisionTreeClassifier(max_depth=depth).fit(X_train, y_train)
    fig, axes = plt.subplots(nrows = 1,ncols = 1, figsize=(10,5), dpi=100)
    tree.plot_tree(model, feature_names = X_train.columns, filled = True)

    return model

model = run_single_tree(X_train, y_train, sm_best_tree_depth)
```



In [131]:
```python
model.feature_importances_
```

Out[131]:
```
array([0.1521591 , 0.28918785, 0.16800672, 0.14953643, 0.17422121,
       0.02334318, 0.00587173, 0.03767377])
```

In [133]:
```python
X_train.columns
```

Out[133]:
```
Index(['temp', 'rel', 'wdir', 'wspeed', 'stn', 'hour', 'workout', 'work_hour'], dtype='object')
```

In [134]:
```python
import numpy as np

X_train.columns[np.argsort(model.feature_importances_)[::-1]][:3]
```

Out[134]:
```
Index(['rel', 'stn', 'wdir'], dtype='object')
```

In [135]:
```python
test_data = pd.read_csv('2020average.csv')
test_data
```

Out[135]:

	date	temp	dew	rel	wdir	wspeed	stn	hour	dayweek	workout	work	work_hour
0	2020-09-01 0:00	12.70	7.85	72.5	30.5	8.5	101.785	0	1	0	0	0
1	2020-09-01 1:00	12.45	7.85	73.5	30.0	10.5	101.800	1	1	0	0	0
2	2020-09-01 2:00	12.35	7.85	74.0	30.0	13.5	101.830	2	1	0	0	0
3	2020-09-01 3:00	11.85	7.85	76.5	31.0	10.5	101.885	3	1	0	0	0
4	2020-09-01 4:00	11.75	8.55	81.0	32.5	8.0	101.895	4	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
739	2020-10-01 19:00	15.40	15.05	98.0	16.5	10.0	101.210	19	3	0	0	0
740	2020-10-01 20:00	15.30	14.95	98.0	16.5	5.0	101.240	20	3	0	0	0
741	2020-10-01 21:00	15.20	14.85	98.0	17.5	4.0	101.250	21	3	0	0	0
742	2020-10-01 22:00	15.20	14.75	97.0	8.5	2.0	101.280	22	3	0	0	0
743	2020-10-01 23:00	15.00	14.50	97.0	NaN	1.0	101.320	23	3	0	0	0

744 rows × 12 columns

In [136]:
```python
test_df = test_data.drop(['date', 'dew', 'work','dayweek'], 1)
test_df.columns
```

Out[136]:
```
Index(['temp', 'rel', 'wdir', 'wspeed', 'stn', 'hour', 'workout', 'work_hour'], dtype='object')
```

In [137]:
```python
rows_with_nan = []
for index, row in test_df.iterrows():
    is_nan_series = row.isnull()
    if is_nan_series.any():
        rows_with_nan.append(index)

print(rows_with_nan)
test_df = test_df.dropna(axis=0, how='any')
```

```
[6, 9, 11, 17, 18, 26, 89, 91, 92, 93, 94, 95, 96, 97, 98, 107, 116, 117, 118, 119, 123, 124, 125, 12
6, 127, 128, 129, 130, 131, 132, 133, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 15
2, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 165, 166, 168, 170, 171, 172, 173, 174, 175, 17
6, 177, 178, 179, 180, 181, 183, 184, 185, 186, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 19
8, 199, 200, 201, 202, 203, 204, 205, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 22
1, 222, 223, 224, 225, 226, 227, 228, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 26
0, 261, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 285, 286, 287, 288, 289, 290, 291, 292, 29
3, 294, 295, 296, 324, 326, 327, 328, 329, 330, 331, 365, 405, 409, 410, 538, 688, 690, 699, 700, 73
4, 735, 736, 743]
```

In [138]:
```python
test_df.isnull().values.any()
```

Out[138]:
```
False
```

In [182]:
```python
sum(model.predict_proba(test_df)[0][1:])
```

Out[182]:
```
0.022058823529411766
```

In [183]:
```python
a = model.predict_proba(test_df)
count=0
for x in a:
    count+=1
    if sum(x[1:])> .5 :
        print(sum(x[1:]))
```

```
1.0
0.75
0.75
0.75
0.75
1.0
0.75
0.75
0.75
0.75
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
1.0
```