

ΠΕΡΙΕΧΟΜΕΝΑ

1. Berlin Preprocess.....	2
2. BERLIN CLASSIFY.....	8
3. BERLIN CLASSIFY 3.....	24
4. BERLIN REGRESSION.....	43

1. Berlin Preprocess:

```
# -*- coding: utf-8 -*-
```

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
import os
```

```
os.chdir("C:/users/dimitris/desktop")
```

```
file = "listings_summary.csv"
```

```
df_listing = pd.read_csv(file)
```

```
## PRERPROCESS DATA
```

```
columns_to_keep = ["price", "bedrooms", "bed_type", "amenities",
```

```
                    "security_deposit", "cleaning_fee", "guests_included",
```

```
                    "extra_people",    "minimum_nights",    "instant_bookable", "latitude",
```

```
                    "longitude"]
```

```
df_raw = df_listing[columns_to_keep]

df_raw.head()

#make one column for each bed type (couch, futon, sofa, real bed)

bed_type_dummy = pd.get_dummies(df_raw["bed_type"], drop_first=True)

df_raw = pd.concat([df_raw, bed_type_dummy], axis=1)

df_raw.head()

# Wifi in amenities - 1 if there is, 0 if not

def wifi(value):

    if "wifi" in value.lower():

        return 1

    else:

        return 0

df_raw["Wifi"] = df_raw["amenities"].apply(wifi)

# TV in amenities - 1 if there is, 0 if not

def tv(value):

    if "tv" in value.lower():

        return 1

    else:

        return 0
```

```

df_raw["Tv"] = df_raw["amenities"].apply(tv)

# security_deposit, cleaning_fee - NaNs replaced with 0

df_raw["security_deposit"].fillna("$0.00", inplace=True)

df_raw["cleaning_fee"].fillna("$0.00", inplace=True)

# Removing $ sign and "," from price

def remove_dollar(str_price):

    if "," in str_price:

        str_price = str_price.replace(",", "")

    else:

        pass

    return str_price[1:]

for dollar_column in ["price", "security_deposit", "cleaning_fee", "extra_people"]:

    price_no_dollar = df_raw[dollar_column].apply(remove_dollar)

    df_raw[dollar_column] = price_no_dollar

def bookable(value):

    if value == "t":

        return 1

    else:

        return 0

```

```
df_raw["instant_bookable"] = df_raw["instant_bookable"].apply(bookable)
```

```
df_raw.drop(["bed_type", "amenities"], axis=1, inplace=True)
```

```
# Converting columns to "float64" dtype
```

```
price_num = pd.to_numeric(df_raw["price"])
```

```
security_deposit_num = pd.to_numeric(df_raw["security_deposit"])
```

```
cleaning_fee_num = pd.to_numeric(df_raw["cleaning_fee"])
```

```
extra_people_num = pd.to_numeric(df_raw["extra_people"])
```

```
# Removing the old columns
```

```
df_raw["price"] = price_num
```

```
df_raw["security_deposit"] = security_deposit_num
```

```
df_raw["cleaning_fee"] = cleaning_fee_num
```

```
df_raw["extra_people"] = extra_people_num
```

```
#if na in bedrooms, set to 1
```

```
df_raw["bedrooms"].fillna(1.0, inplace=True)
```

```

df_raw.head()

from geopy.distance import great_circle

def distance_from_centre(lat, lon):

    berlin_centre = (52.520008, 13.404954)

    apartment_spot = (lat, lon)

    return round(great_circle(berlin_centre, apartment_spot).km, 1)

df_raw["Distance"] = df_raw.apply(lambda x: distance_from_centre(x.latitude,
x.longitude), axis=1)

df_raw.drop(["latitude", "longitude"], axis=1, inplace=True)

print("Our final working dataset has {} rows and {}
columns.".format(*df_raw.shape))

f,ax = plt.subplots(figsize=(9,9))

sns.heatmap(df_raw.corr(),annot=True,linewidth=.5,fmt ='.2f',ax = ax)

# to labels einai false stis fthines times kai true stis akrives

#afto to apofasizoume me vasi tin mesi timi

mean_price = np.mean(df_raw['price'])

labels = df_raw['price'] > mean_price

```

```
price = df_raw['price']
```

```
df_raw.drop(["price"], axis=1, inplace=True)
```

```
np.save("berlin_data.npy",df_raw)
```

```
np.save("berlin_labels.npy",labels)
```

```
np.save("berlin_price.npy",price)
```

```
labels3 = np.zeros(len(price))
```

```
x = np.where((price>0.8*mean_price) & (price<=1.2*mean_price))
```

```
labels3[x] = 1
```

```
x = np.where(price>1.2*mean_price)
```

```
labels3[x] = 2
```

```
np.save("berlin_labels3.npy",labels3)
```

BERLIN CLASSIFY:

```
# -*- coding: utf-8 -*-
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import StratifiedKFold
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn import svm
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

```
from sklearn.metrics import confusion_matrix
```

```
from random import sample
```

```
import os
```

```
os.chdir("C:/users/dimitris/desktop")
```

```
df_raw = np.load("berlin_data.npy")
```

```
labels = np.load("berlin_labels.npy")
```

```
colnames = ['bedrooms', 'security_deposit', 'cleaning_fee',
            'guests_included', 'extra_people', 'minimum_nights', 'instant_bookable',
            'Couch', 'Futon', 'Pull-out Sofa', 'Real Bed', 'Wifi', 'Tv',
            'Distance']
```

```
#%% prepare data
```

```
N = df_raw.shape
```

```
n1 = np.sum(labels==True) #6167
```

```
n2 = np.sum(labels==False) #16385
```

```
SAMPLING = True
```

```
if SAMPLING:
```

```
    x = np.where(labels==False)[0]
```

```
    s2 = sample(list(x), n1) #select 6167 from 16385
```

```
    s2 = np.array(s2)
```

```
    s1 = np.where(labels)[0]
```

```
    s = np.concatenate((s1,s2))
```

```
    df_raw = df_raw[s,:]
```

```
labels = labels[s]

N = df_raw.shape

K = 10

kf = StratifiedKFold(n_splits=K, shuffle=True, random_state=101)

#%% KNN

pred_knn = np.zeros([N[0]])

acc_knn_all = np.zeros(20)

for nn in range(1,21,2):

    i = 0

    for tr, ts in kf.split(df_raw,labels):

        print(i)

        x_train = df_raw[tr,:]

        y_train = labels[tr]

        y_train2 = labels[tr]

        x_test = df_raw[ts,:]

        knn = KNeighborsClassifier(n_neighbors=nn)

        knn.fit(x_train, y_train2)

        pred_knn[ts] = knn.predict(x_test)
```

```
i = i + 1

#end_for

pred_knn = pred_knn.astype('int64')

acc_knn_all[nn] = np.sum(pred_knn==labels)/len(labels)

print("Acc_knn = ", acc_knn_all[nn])


x = np.arange(1,21,2)

plt.plot(x,acc_knn_all[x])

plt.xlabel('K')

plt.ylabel('Accuracy')

plt.axis([0, 20, 0.75, 0.81])

plt.show()


print(np.max(acc_knn_all))

print(np.argmax(acc_knn_all))


#run for best k

nn = np.argmax(acc_knn_all)
```

```
i = 0

for tr, ts in kf.split(df_raw, labels):

    print(i)

    x_train = df_raw[tr,:]

    y_train = labels[tr]

    y_train2 = labels[tr]

    x_test = df_raw[ts,:]


    knn = KNeighborsClassifier(n_neighbors=nn)

    knn.fit(x_train, y_train2)

    pred_knn[ts] = knn.predict(x_test)


    i = i + 1

    #end_for


pred_knn = pred_knn.astype('int64')

acc_knn = np.sum(pred_knn == labels) / len(labels)

conf_knn = confusion_matrix(labels, pred_knn)
```

```

print("KNN")

print("Accuracy  ", acc_knn)

tn, fp, fn, tp = conf_knn.ravel()

sen = tp / (tp + fn)

spe = tn / (tn + fp)

print("Sensitivity ", sen)

print("Specifivity ", spe)

print("Confusion Matrix")

print(conf_knn[0][0], conf_knn[0][1])

print(conf_knn[1][0], conf_knn[1][1])

#%% SVM Linear

#pred_svm_l = np.zeros([N[0]])

#

#i = 0

#for tr, ts in kf.split(df_raw, labels):

#    print(i)

```

```

# x_train = df_raw[tr,:]

# y_train = labels[tr]

# x_test = df_raw[ts,:]

#

# svm_model = svm.SVC(kernel='linear') #rbf kernel, auto-search for best gamma

# svm_model.fit(x_train, y_train)

# pred_svm_l[ts] = svm_model.predict(x_test)

#

# i = i + 1

#

##end_for

#pred_svm_l = pred_svm_l.astype('int64')

#acc_svm_l = np.sum(pred_svm_l==labels)/len(labels)

#conf_svm_l = confusion_matrix(labels, pred_svm_l)

#print("Acc_svm_linear = ", acc_svm_l)


### SVM rbf

```

```
pred_svm = np.zeros([N[0]])

i = 0

for tr, ts in kf.split(df_raw, labels):

    print(i)

    x_train = df_raw[tr,:]

    y_train = labels[tr]

    x_test = df_raw[ts,:]

    svm_model = svm.SVC() #rbf kernel, auto-search for best gamma

    svm_model.fit(x_train, y_train)

    pred_svm[ts] = svm_model.predict(x_test)

    i = i + 1

#end_for

pred_svm = pred_svm.astype('int64')
```

```

acc_svm = np.sum(pred_svm==labels)/len(labels)

conf_svm = confusion_matrix(labels, pred_svm)

print("SVM")

print("Accuracy  ", acc_svm)


tn, fp, fn, tp = conf_svm.ravel()

sen = tp / (tp + fn)

spe = tn / (tn + fp)


print("Sensitivity ", sen)

print("Specifivity ", spe)

print("Confusion Matrix")

print(conf_svm[0][0], conf_svm[0][1])

print(conf_svm[1][0], conf_svm[1][1])


### RF

np.random.seed(1) #set random seed, so we always get same random numbers


#mtry = np.round(0.5*np.sqrt(N[1])).astype('int64')

```

```
mtry = np.round(np.sqrt(N[1])).astype('int64')

#mtry = np.round(2*np.sqrt(N[1])).astype('int64')


pred_rf = np.zeros([N[0]])

imp_rf = np.zeros((K, N[1]))

i = 0

for tr, ts in kf.split(df_raw, labels):

    print(i)

    x_train = df_raw[tr,:]

    y_train = labels[tr]

    x_test = df_raw[ts,:]


    rf = RandomForestClassifier(n_estimators=100, max_features = mtry)

    rf.fit(x_train, y_train)

    pred_rf[ts] = rf.predict(x_test)

    imp_rf[i,:] = rf.feature_importances_


    i = i + 1
```

```
#end_for

pred_rf = pred_rf.astype('int64')

acc_rf = np.sum(pred_rf==labels)/len(labels)

conf_rf = confusion_matrix(labels, pred_rf)

print("RF")

print("Accuracy  ", acc_rf)


tn, fp, fn, tp = conf_rf.ravel()

sen = tp / (tp + fn)

spe = tn / (tn + fp)


print("Sensitivity ", sen)

print("Specifivity ", spe)

print("Confusion Matrix")

print(conf_rf[0][0], conf_rf[0][1])

print(conf_rf[1][0], conf_rf[1][1])
```

```
### Naive Bayes
```

```
pred_nb = np.zeros([N[0]])
```

```
i = 0
```

```
for tr, ts in kf.split(df_raw, labels):
```

```
    print(i)
```

```
    x_train = df_raw[tr,:]
```

```
    y_train = labels[tr]
```

```
    x_test = df_raw[ts,:]
```

```
    gnb_model = GaussianNB()
```

```
    gnb_model.fit(x_train, y_train)
```

```
    pred_nb[ts] = gnb_model.predict(x_test)
```

```
    i = i + 1
```

```
#end_for

pred_nb = pred_nb.astype('int64')

acc_nb = np.sum(pred_nb==labels)/len(labels)

conf_nb = confusion_matrix(labels, pred_nb)

print("NB")

print("Accuracy  ", acc_nb)


tn, fp, fn, tp = conf_nb.ravel()

sen = tp / (tp + fn)

spe = tn / (tn + fp)


print("Sensitivity ", sen)

print("Specifivity ", spe)

print("Confusion Matrix")

print(conf_nb[0][0], conf_nb[0][1])

print(conf_nb[1][0], conf_nb[1][1])
```

```
#%% Decision tree
```

```
pred_dt = np.zeros([N[0]])
```

```
i = 0
```

```
for tr, ts in kf.split(df_raw, labels):
```

```
    print(i)
```

```
    x_train = df_raw[tr,:]
```

```
    y_train = labels[tr]
```

```
    x_test = df_raw[ts,:]
```

```
    dt_model = DecisionTreeClassifier(random_state=0)
```

```
    dt_model.fit(x_train, y_train)
```

```
    pred_dt[ts] = dt_model.predict(x_test)
```

```
    i = i + 1
```

```
#end_for
```

```
pred_dt = pred_dt.astype('int64')

acc_dt = np.sum(pred_dt==labels)/len(labels)

conf_dt= confusion_matrix(labels, pred_dt)

print("DT")

print("Accuracy  ", acc_dt)


tn, fp, fn, tp = conf_dt.ravel()

sen = tp / (tp + fn)

spe = tn / (tn + fp)


print("Sensitivity ", sen)

print("Specifivity ", spe)

print("Confusion Matrix")

print(conf_dt[0][0], conf_dt[0][1])

print(conf_dt[1][0], conf_dt[1][1])


fig = plt.figure(figsize=(8,6), dpi=200)

plot_tree(dt_model, max_depth=2,feature_names=colnames,fontsize=8)
```

```
fig.savefig('tree.png')
```

```
### plots
```

```
accs = [acc_knn, acc_svm, acc_rf]
```

```
plt.bar([1,2,3],accs,tick_label=['KNN','SVM','RF'])
```

```
plt.ylabel('Accuracy')
```

```
plt.show()
```

```
imp_s = np.sum(imp_rf, axis=0)
```

```
s_idx = np.argsort(imp_s)
```

BERLIN CLASSIFY 3:

```
# -*- coding: utf-8 -*-
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import StratifiedKFold
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn import svm
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

```
from sklearn.metrics import confusion_matrix
```

```
from random import sample
```

```
import os
```

```
os.chdir("C:/users/dimitris/desktop")
```

```
df_raw = np.load("berlin_data.npy")

labels = np.load("berlin_labels3.npy")


colnames = ['bedrooms', 'security_deposit', 'cleaning_fee',

            'guests_included', 'extra_people', 'minimum_nights', 'instant_bookable',

            'Couch', 'Futon', 'Pull-out Sofa', 'Real Bed', 'Wifi', 'Tv',

            'Distance']


### subsampling


N = df_raw.shape


n1 = np.sum(labels==0) #13549

n2 = np.sum(labels==1) #5064

n3 = np.sum(labels==2) #3939


SAMPLING = True


if SAMPLING:

    x = np.where(labels==0)[0]
```

```
s1 = sample(list(x), n3) #select 3939 from 13549

s1 = np.array(s1)

x = np.where(labels==1)[0]

s2 = sample(list(x), n3) #select 3939 from 5064

s2 = np.array(s2)

s3 = np.where(labels==2)[0]

s = np.concatenate((s1,s2,s3))

df_raw = df_raw[s,:]

labels = labels[s]

N = df_raw.shape

K = 10

kf = StratifiedKFold(n_splits=K, shuffle=True, random_state=101)

#%% KNN

pred_knn = np.zeros([N[0]])

acc_knn_all = np.zeros(20)

for nn in range(1,21,2):

    i = 0

    for tr, ts in kf.split(df_raw,labels):

        print(i)
```

```
x_train = df_raw[tr,:]

y_train = labels[tr]

y_train2 = labels[tr]

x_test = df_raw[ts,:]


knn = KNeighborsClassifier(n_neighbors=nn)

knn.fit(x_train, y_train2)

pred_knn[ts] = knn.predict(x_test)


i = i + 1


#end_for


pred_knn = pred_knn.astype('int64')

acc_knn_all[nn] = np.sum(pred_knn==labels)/len(labels)

print("Acc_knn = ", acc_knn_all[nn])
```

```
x = np.arange(1,21,2)

plt.plot(x,acc_knn_all[x])

plt.xlabel('K')

plt.ylabel('Accuracy')

plt.show()


print(np.max(acc_knn_all))

print(np.argmax(acc_knn_all))


#run for best k

nn = np.argmax(acc_knn_all)

i = 0

for tr, ts in kf.split(df_raw,labels):

    print(i)

    x_train = df_raw[tr,:]

    y_train = labels[tr]

    y_train2 = labels[tr]
```

```
x_test = df_raw[ts,:]
```

```
knn = KNeighborsClassifier(n_neighbors=nn)
```

```
knn.fit(x_train, y_train2)
```

```
pred_knn[ts] = knn.predict(x_test)
```

```
i = i + 1
```

```
#end_for
```

```
pred_knn = pred_knn.astype('int64')
```

```
acc_knn = np.sum(pred_knn == labels) / len(labels)
```

```
conf_knn = confusion_matrix(labels, pred_knn)
```

```
print("KNN")
```

```
print("Accuracy  ", acc_knn)
```

```
conf_knn = np.array(conf_knn)
```

```
sen = np.zeros(3)

spe = np.zeros(3)

for i in range(3):

    sen[i] = conf_knn[i,i] / np.sum(conf_knn[i,:])

    other_lines = list(range(3))

    del other_lines[i]

    fp = np.sum(conf_knn[other_lines,i])

    tmp = conf_knn[other_lines,:]

    tmp = tmp[:,other_lines]

    tn = np.sum(tmp)

    spe[i] = tn / (tn + fp)


print("Sensitivity ", np.mean(sen))

print(sen)

print("Specifivity ", np.mean(spe))

print(spe)

print("Confusion Matrix")

print(conf_knn)
```

```
### SVM Linear

#pred_svm_l = np.zeros([N[0]])

#

#i = 0

#for tr, ts in kf.split(df_raw,labels):

#    print(i)

#    x_train = df_raw[tr,:]

#    y_train = labels[tr]

#    x_test = df_raw[ts,:]

#

#    svm_model = svm.SVC(kernel='linear') #rbf kernel, auto-search for best gamma

#    svm_model.fit(x_train, y_train)

#    pred_svm_l[ts] = svm_model.predict(x_test)

#

#    i = i + 1

#

##end_for

#pred_svm_l = pred_svm_l.astype('int64')
```

```
#acc_svm_l = np.sum(pred_svm_l==labels)/len(labels)

#conf_svm_l = confusion_matrix(labels, pred_svm_l)

#print("Acc_svm_linear = ", acc_svm_l)


### SVM rbf

pred_svm = np.zeros([N[0]])

i = 0

for tr, ts in kf.split(df_raw, labels):

    print(i)

    x_train = df_raw[tr,:]

    y_train = labels[tr]

    x_test = df_raw[ts,:]

    svm_model = svm.SVC() #rbf kernel, auto-search for best gamma

    svm_model.fit(x_train, y_train)

    pred_svm[ts] = svm_model.predict(x_test)
```

```

    i = i + 1

#end_for

pred_svm = pred_svm.astype('int64')

acc_svm = np.sum(pred_svm==labels)/len(labels)

conf_svm = confusion_matrix(labels, pred_svm)

print("SVM")

print("Accuracy  ", acc_svm)


conf_svm = np.array(conf_svm)

sen = np.zeros(3)

spe = np.zeros(3)

for i in range(3):

    sen[i] = conf_svm[i,i] / np.sum(conf_svm[i,:])

    other_lines = list(range(3))

    del other_lines[i]

```

```

fp = np.sum(conf_svm[other_lines,i])

tmp = conf_svm[other_lines,:]

tmp = tmp[:,other_lines]

tn = np.sum(tmp)

spe[i] = tn / (tn + fp)


print("Sensitivity ", np.mean(sen))

print(sen)

print("Specifivity ", np.mean(spe))

print(spe)

print("Confusion Matrix")

print(conf_svm)


### RF

np.random.seed(1) #set random seed, so we always get same random numbers


#mtry = np.round(0.5*np.sqrt(N[1])).astype('int64')

mtry = np.round(np.sqrt(N[1])).astype('int64')

```

```
#mtry = np.round(2*np.sqrt(N[1])).astype('int64')

pred_rf = np.zeros([N[0]])

imp_rf = np.zeros((K, N[1]))

i = 0

for tr, ts in kf.split(df_raw, labels):

    print(i)

    x_train = df_raw[tr,:]

    y_train = labels[tr]

    x_test = df_raw[ts,:]

    rf = RandomForestClassifier(n_estimators=100, max_features = mtry)

    rf.fit(x_train, y_train)

    pred_rf[ts] = rf.predict(x_test)

    imp_rf[i,:] = rf.feature_importances_

    i = i + 1

#end_for
```

```

pred_rf = pred_rf.astype('int64')

acc_rf = np.sum(pred_rf==labels)/len(labels)

conf_rf = confusion_matrix(labels, pred_rf)

print("RF")

print("Accuracy   ", acc_rf)


conf_rf = np.array(conf_rf)

sen = np.zeros(3)

spe = np.zeros(3)

for i in range(3):

    sen[i] = conf_rf[i,i] / np.sum(conf_rf[i,:])

    other_lines = list(range(3))

    del other_lines[i]

    fp = np.sum(conf_rf[other_lines,i])

    tmp = conf_rf[other_lines,:]

    tmp = tmp[:,other_lines]

    tn = np.sum(tmp)

```

```
spe[i] = tn / (tn + fp)
```

```
print("Sensitivity ", np.mean(sen))
```

```
print(sen)
```

```
print("Specifivity ", np.mean(spe))
```

```
print(spe)
```

```
print("Confusion Matrix")
```

```
print(conf_rf)
```

```
### Naive Bayes
```

```
pred_nb = np.zeros([N[0]])
```

```
i = 0
```

```
for tr, ts in kf.split(df_raw, labels):
```

```
    print(i)
```

```
    x_train = df_raw[tr,:]
```

```
    y_train = labels[tr]
```

```
    x_test = df_raw[ts,:]
```

```
gnb_model = GaussianNB()

gnb_model.fit(x_train, y_train)

pred_nb[ts] = gnb_model.predict(x_test)


i = i + 1


#end_for


pred_nb = pred_nb.astype('int64')

acc_nb = np.sum(pred_nb==labels)/len(labels)

conf_nb = confusion_matrix(labels, pred_nb)

print("NB")

print("Accuracy   ", acc_nb)


sen = np.zeros(3)

spe = np.zeros(3)

for i in range(3):
```

```
sen[i] = conf_nb[i,i] / np.sum(conf_nb[i,:])

other_lines = list(range(3))

del other_lines[i]

fp = np.sum(conf_nb[other_lines,i])

tmp = conf_nb[other_lines,:]

tmp = tmp[:,other_lines]

tn = np.sum(tmp)

spe[i] = tn / (tn + fp)


print("Sensitivity ", np.mean(sen))

print(sen)

print("Specifivity ", np.mean(spe))

print(spe)

print("Confusion Matrix")

print(conf_nb)


### Decision tree

pred_dt = np.zeros([N[0]])
```

```
i = 0

for tr, ts in kf.split(df_raw, labels):

    print(i)

    x_train = df_raw[tr,:]

    y_train = labels[tr]

    x_test = df_raw[ts,:]

    dt_model = DecisionTreeClassifier(random_state=0)

    dt_model.fit(x_train, y_train)

    pred_dt[ts] = dt_model.predict(x_test)

    i = i + 1

#end_for

pred_dt = pred_dt.astype('int64')

acc_dt = np.sum(pred_dt==labels)/len(labels)
```

```
conf_dt= confusion_matrix(labels, pred_dt)
```

```
print("DT")
```

```
print("Accuracy  ", acc_dt)
```

```
sen = np.zeros(3)
```

```
spe = np.zeros(3)
```

```
for i in range(3):
```

```
    sen[i] = conf_dt[i,i] / np.sum(conf_dt[i,:])
```

```
    other_lines = list(range(3))
```

```
    del other_lines[i]
```

```
    fp = np.sum(conf_dt[other_lines,i])
```

```
    tmp = conf_dt[other_lines,:]
```

```
    tmp = tmp[:,other_lines]
```

```
    tn = np.sum(tmp)
```

```
    spe[i] = tn / (tn + fp)
```

```
print("Sensitivity ", np.mean(sen))
```

```
print(sen)
```

```
print("Specifivity ", np.mean(spe))
```

```
print(spe)
```

```
print("Confusion Matrix")
```

```
print(conf_dt)
```

```
fig = plt.figure(figsize=(8,6), dpi=200)
```

```
plot_tree(dt_model, max_depth=2,feature_names=colnames,fontsize=8)
```

```
fig.savefig('tree3.png')
```

```
### plots
```

```
accs = [acc_knn, acc_svm, acc_rf]
```

```
plt.bar([1,2,3],accs,tick_label=['KNN','SVM','RF'])
```

```
plt.ylabel('Accuracy')
```

```
plt.show()
```

```
imp_s = np.sum(imp_rf, axis=0)
```

```
s_idx = np.argsort(imp_s)
```

BERLIN REGRESSION:

```
# -*- coding: utf-8 -*-
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import KFold
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.linear_model import LinearRegression
```

```
import os
```

```
os.chdir("C:/users/dimitris/desktop")
```

```
df_raw = np.load("berlin_data.npy")
```

```
price = np.load("berlin_price.npy")
```

```
plt.plot(np.sort(price))
```

```
N = df_raw.shape
```

```
tmp = price < 500
```

```
price = price[tmp]
```

```
df_raw = df_raw[tmp,:]
```

```
### LR
```

```
K = 10
```

```
kf = KFold(n_splits=K, shuffle=True, random_state=101)
```

```
pred_lr = np.zeros([N[0]])
```

```
mse = np.zeros(10)
```

```
rsq = np.zeros(10)
```

```
i = 0
```

```
for tr, ts in kf.split(df_raw):
```

```
    print(i)
```

```
    x_train = df_raw[tr,:]
```

```
    y_train = price[tr]
```

```
    y_train2 = price[tr]
```

```
    x_test = df_raw[ts,:]
```

```
    y_test = price[ts]
```

```
lr_model = LinearRegression()

lr_model.fit(x_train, y_train2)


pred_lr[ts] = lr_model.predict(x_test)

mse[i] = np.sum((pred_lr[ts] - y_test)**2)

rsq[i] = lr_model.score(x_test, y_test)


i = i + 1


#end_for


print("LR")

print("MSE ", np.mean(mse))

print("RSQ ", np.mean(rsq))
```

```
### RF

np.random.seed(1) #set random seed, so we always get same random numbers

#mtry = np.round(0.5*np.sqrt(N[1])).astype('int64')

mtry = np.round(np.sqrt(N[1])).astype('int64')

#mtry = np.round(2*np.sqrt(N[1])).astype('int64')


pred_rf = np.zeros([N[0]])


mse = np.zeros(10)

rsq = np.zeros(10)

i = 0

for tr, ts in kf.split(df_raw):

    print(i)

    x_train = df_raw[tr,:]

    y_train = price[tr]

    y_train2 = price[tr]

    x_test = df_raw[ts,:]

    y_test = price[ts]
```

```
rf = RandomForestRegressor(n_estimators=100, max_features = mtry)

rf.fit(x_train, y_train)


pred_lr[ts] = rf.predict(x_test)

mse[i] = np.sum((pred_lr[ts] - y_test)**2)

rsq[i] = rf.score(x_test, y_test)


i = i + 1


#end_for


print("RF")

print("MSE ", np.mean(mse))

print("RSQ ", np.mean(rsq))
```