Lyons Housing Data Set

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

```
#load the data, change the file address

infile= "lyon_housing.csv"
lyon=pd.read_csv(infile)
```

```
lyon.head()
```

transaction	type_achat	type_bien	nombre_pieces	surface_logement	surface_carrez_log
2019-10-31	ancien	maison	5	100.0	
2018-11-26	ancien	maison	2	52.0	
2016-08-04	ancien	appartement	1	28.0	
2016-11-18	ancien	appartement	3	67.0	
2016-12-16	ancien	appartement	1	28.0	

This data is from https://www.kaggle.com/benoitfavier/lyon-housing

type_achat- ancien means existing house, VEFA= sale prior to completion

type bien- house or apartment

nombre_pieces- probably number of rooms, use this

Surface_legement-interior space in square meeters

surface_carrez_logment-area with roof height under 1.8 m (drop this variable)

surface_terrain- drop this

nombre_parkings- parking spots

prix- selling price, predict this

anciennete- age of the property in years

Convert the dates into Pandas datetime variables, so we can extract the year of the build and the year of the sale

It is also possible to extract quarter of the year from the datetime variables, or even months, we could look for seasonality in prices if so inclined

Anyway, extract the year of the sale, that is a categorical variable we will want

```
lyon['date_transaction']=pd.to_datetime(lyon['date_transaction'])
```

```
lyon['date_transaction'].head()
```

```
0   2019-10-31
1   2018-11-26
2   2016-08-04
3   2016-11-18
4   2016-12-16
Name: date_transaction, dtype: datetime64[ns]
```

```
lyon['year_transaction']=lyon['date_transaction'].dt.year
```

```
lyon['date_construction']=pd.to_datetime(lyon['date_construction'])
```

```
lyon['year_construction']=lyon['date_construction'].dt.year
```

```
lyon.head()
```

	date_transaction	type_achat	type_bien	nombre_pieces	surface_logement	surface_c
0	2019-10-31	ancien	maison	5	100.0	
1	2018-11-26	ancien	maison	2	52.0	
2	2016-08-04	ancien	appartement	1	28.0	
3	2016-11-18	ancien	appartement	3	67.0	
4	2016-12-16	ancien	appartement	1	28.0	

```
# how about the age of the property?
lyon['anciennete'].describe()
```

```
count    40516.000000
mean        21.246938
std          9.397379
min         -3.853563
25%         15.064690
50%         26.571388
75%         28.775403
max         31.494144
Name: anciennete, dtype: float64
```

Convert from a continuous variable into categorical, using Pandas cut

There is too much detail in the age of properties, use the cut function in pandas to convert this to a limited number of categories

```
temp=pd.cut(lyon.anciennete,bins=[-5,0,5,10,20,30,40],labels=['UnderConstruction','0-5','5-10','10-20','20-30','30+'])
```

```
lyon['age']=temp
```

```
lyon.head(3)
```

	date_transaction	type_achat	type_bien	nombre_pieces	surface_logement	surface_c
0	2019-10-31	ancien	maison	5	100.0	
1	2018-11-26	ancien	maison	2	52.0	
2	2016-08-04	ancien	appartement	1	28.0	

Okay, that's as far as I will go in addressing a couple of issues there, your turn.

Build your predictor of housing prices in Lyons

Predictors- use at least these variables type_achat, type_bien, nombre_pieces, surface_logement, nombre_parkings, commune(?), year_transaction (as a category,not an integer, age (category)

-one hot encode the categories

-standard scale the other data

-combine the standard-scaled and the onehot data into a pd Dataframe

-Build a neural net regressor, a nearest neighbhor and a linear

-use some metrics, what is the MSE?, the R2, the mean absolute value error?

-use cross validation to figure out which model seems to be best

-use EPI5 to understand what the most important predictors are

Initial drops suggested by you

```
lyon = lyon.drop(labels="surface_terrain", axis=1)
```

```
lyon.head(3)
```

	date_transaction	type_achat	type_bien	nombre_pieces	surface_logement	surface_c
0	2019-10-31	ancien	maison	5	100.0	
1	2018-11-26	ancien	maison	2	52.0	
2	2016-08-04	ancien	appartement	1	28.0	

```
lyon = lyon.drop(labels="surface_carrez_logement", axis=1)
```

```
lyon
```

	date_transaction	type_achat	type_bien	nombre_pieces	surface_logement	nombr
0	2019-10-31	ancien	maison	5	100.0	
1	2018-11-26	ancien	maison	2	52.0	
2	2016-08-04	ancien	appartement	1	28.0	
3	2016-11-18	ancien	appartement	3	67.0	
4	2016-12-16	ancien	appartement	1	28.0	
...	...	...	...	...	...	
40511	2020-01-28	ancien	appartement	2	34.0	
40512	2020-04-17	ancien	appartement	2	33.0	
40513	2020-04-22	ancien	appartement	2	23.0	
40514	2020-04-22	ancien	appartement	2	34.0	
40515	2020-05-25	ancien	appartement	2	30.0	

40516 rows × 16 columns

Lets get all the unique variables from the given catagorical variables. type_achat, type_bien, nombre_pieces, surface_logement, nombre_parkings, commune, year_transaction, age

```
lyon["type_achat"].unique()
```

```
array(['ancien', 'VEFA'], dtype=object)
```

```
lyon["type_bien"].unique()
```

```
array(['maison', 'appartement'], dtype=object)
```

```
lyon["nombre_pieces"].unique()
```

```
array([5, 2, 1, 3, 4, 6])
```

```
lyon["surface_logement"].unique()
#This one is continuous
```

```
array([100.,  52.,  28.,  67.,  42.,  84.,  70.,  63.,  77.,  51.,  99.,
        65.,  58.,  25.,  83.,  78.,  45.,  68.,  69.,  33.,  27.,  32.,
        72.,  88.,  74.,  55.,  75.,  76.,  30.,  40.,  54.,  98.,  86.,
        57.,  81.,  60.,  21.,  23.,  20.,  29.,  89.,  66.,  79.,  49.,
        37.,  31.,  34.,  71., 111.,  64.,  48.,  82.,  39.,  87.,  41.,
       200.,  44.,  47.,  90.,  97., 102., 123., 127.,  91.,  80.,  59.,
        46.,  22.,  43.,  62.,  61., 126., 101.,  56.,  50.,  36.,  38.,
        73., 121., 106., 105., 130.,  96.,  35., 108., 141., 110.,  92.,
        85., 117.,  24., 120., 113., 116., 103.,  53., 139., 118., 163.,
       205., 150., 125., 124.,  94., 168.,  93., 144., 152.,  95., 155.,
       114., 158., 174., 286., 115., 128., 162.,  26., 107., 122., 109.,
       148., 149., 112., 145., 147., 164., 140., 134., 104., 210., 142.,
       185., 161., 137., 157., 135., 192., 129., 119., 160., 159., 136.,
       170., 138., 133., 184., 166., 176., 132., 220., 154., 197., 178.,
```

```
           143., 213., 182., 175., 151., 180., 146., 187., 189., 165., 169.,
           203., 156., 186., 173., 177., 300., 232., 190., 191., 153., 194.,
           172., 207., 167., 193., 235., 219., 257., 196., 201., 237., 206.,
           268., 223., 217., 198., 211., 270., 188., 230., 179., 216., 195.,
           131., 208., 183., 280., 221., 229., 284., 240., 265., 250., 228.,
           251., 225., 227., 171., 181., 267., 199., 236., 204., 285., 224.])
```

```
lyon["nombre_parkings"].unique()
```

```
array([0, 1, 2, 3])
```

```
lyon["commune"].unique()
```

```
array(['Villeurbanne', 'Lyon 1er Arrondissement',
       'Lyon 2e Arrondissement', 'Lyon 3e Arrondissement',
       'Lyon 4e Arrondissement', 'Lyon 5e Arrondissement',
       'Lyon 6e Arrondissement', 'Lyon 7e Arrondissement',
       'Lyon 8e Arrondissement', 'Lyon 9e Arrondissement'], dtype=object)
```

```
lyon["year_transaction"].unique()
```

```
array([2019, 2018, 2016, 2017, 2020, 2021])
```

```
lyon["age"].unique()
```

```
['10-20', '5-10', 'UnderConstruction', '0-5', '20-30', '30+']
Categories (6, object): ['UnderConstruction' < '0-5' < '5-10' < '10-20' < '20-30' < '30+']
```

```
#going to to impute all strings with the most frequent string
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

#Impute catagorical data

# I should keep commune out perhaps as they are all commune, however I will keep it in there because you wanted it
# There is also no nan values in that column but I just need it in the data frame

imp = SimpleImputer(strategy="most_frequent")
StringImpuing= pd.DataFrame(imp.fit_transform(lyon[["type_achat","type_bien","commune","age"]]), columns = ["type_achat","type_bien","commu

#Impute Continuous Data with the median value

impCont = SimpleImputer(strategy="median")
continuousImputing= pd.DataFrame(impCont.fit_transform(lyon[["nombre_pieces","surface_logement","nombre_parkings"]]), columns = ["nombre_pi
```

```
StringImpuing
```

	type_achat	type_bien	commune	age
0	ancien	maison	Villeurbanne	10-20
1	ancien	maison	Villeurbanne	10-20
2	ancien	appartement	Villeurbanne	10-20
3	ancien	appartement	Villeurbanne	10-20
4	ancien	appartement	Villeurbanne	10-20
...	...	...	...	...
40511	ancien	appartement	Lyon 9e Arrondissement	5-10
40512	ancien	appartement	Lyon 9e Arrondissement	5-10
40513	ancien	appartement	Lyon 9e Arrondissement	5-10
40514	ancien	appartement	Lyon 9e Arrondissement	5-10
40515	ancien	appartement	Lyon 9e Arrondissement	5-10

40516 rows × 4 columns

```
continuousImputing
```

	nombre_pieces	surface_logement	nombre_parkings
0	5.0	100.0	0.0
1	2.0	52.0	0.0
2	1.0	28.0	1.0
3	3.0	67.0	1.0
4	1.0	28.0	1.0
...	...	...	...
40511	2.0	34.0	1.0
40512	2.0	33.0	0.0
40513	2.0	23.0	0.0
40514	2.0	34.0	0.0
40515	2.0	30.0	0.0

40516 rows × 3 columns

```
# Combine string imouting with year transaaction

CatagoricalVariable = pd.concat([StringImpuing,lyon["year_transaction"]],axis = 1)
CatagoricalVariable["year_transaction"]
```

```
0        2019
1        2018
2        2016
3        2016
4        2016
         ...
40511    2020
40512    2020
40513    2020
40514    2020
40515    2020
Name: year_transaction, Length: 40516, dtype: int64
```

one hot encode the categories

```
from sklearn.preprocessing import OneHotEncoder

encode_lyon=OneHotEncoder()

encode_lyon_fit1= encode_lyon.fit_transform(lyon[CatagoricalVariable.columns[0]].to_numpy().reshape(-1,1))
df_type_achat=pd.DataFrame(encode_lyon_fit1.toarray(),columns=encode_lyon.categories_[0][:])

encode_lyon_fit2= encode_lyon.fit_transform(lyon[CatagoricalVariable.columns[1]].to_numpy().reshape(-1,1))
df_type_2=pd.DataFrame(encode_lyon_fit2.toarray(),columns=encode_lyon.categories_[0][:])

encode_lyon_fit3= encode_lyon.fit_transform(lyon[CatagoricalVariable.columns[2]].to_numpy().reshape(-1,1))
df_type_3=pd.DataFrame(encode_lyon_fit3.toarray(),columns=encode_lyon.categories_[0][:])

encode_lyon_fit4= encode_lyon.fit_transform(lyon[CatagoricalVariable.columns[3]].to_numpy().reshape(-1,1))
df_type_4=pd.DataFrame(encode_lyon_fit4.toarray(),columns=encode_lyon.categories_[0][:])



encode_lyon_fit5= encode_lyon.fit_transform(lyon[CatagoricalVariable.columns[4]].to_numpy().reshape(-1,1))
stringCon= encode_lyon.categories_[0][:].astype(str)
df_type_5=pd.DataFrame(encode_lyon_fit5.toarray(),columns=encode_lyon.categories_[0][:])

df_type_5.columns=stringCon




#df_type_5=pd.DataFrame(encode_lyon_fit5.toarray(),columns=df5Names)
```

```
type(encode_lyon.categories_[0][1])
```

```
numpy.int64
```

```
stringCon
```

```
array(['2016', '2017', '2018', '2019', '2020', '2021'], dtype='<U21')
```

```
df_type_achat
```

	VEFA	ancien
0	0.0	1.0
1	0.0	1.0
2	0.0	1.0
3	0.0	1.0
4	0.0	1.0
...	...	...
40511	0.0	1.0
40512	0.0	1.0
40513	0.0	1.0
40514	0.0	1.0
40515	0.0	1.0

40516 rows × 2 columns

```
df_type_2
```

	appartement	maison
0	0.0	1.0
1	0.0	1.0
2	1.0	0.0
3	1.0	0.0
4	1.0	0.0
...	...	...
40511	1.0	0.0
40512	1.0	0.0
40513	1.0	0.0
40514	1.0	0.0
40515	1.0	0.0

40516 rows × 2 columns

```
df_type_3
```

	Lyon 1er Arrondissement	Lyon 2e Arrondissement	Lyon 3e Arrondissement	Lyon 4e Arrondissement	Lyon 5e Arrondissement
0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...
40511	0.0	0.0	0.0	0.0	0.0
40512	0.0	0.0	0.0	0.0	0.0
40513	0.0	0.0	0.0	0.0	0.0
40514	0.0	0.0	0.0	0.0	0.0
40515	0.0	0.0	0.0	0.0	0.0

40516 rows × 10 columns

```
df_type_4
```

	0-5	10-20	20-30	30+	5-10	UnderConstruction
0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	1.0	0.0	0.0	0.0	0.0
4	0.0	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...
40511	0.0	0.0	0.0	0.0	1.0	0.0
40512	0.0	0.0	0.0	0.0	1.0	0.0
40513	0.0	0.0	0.0	0.0	1.0	0.0
40514	0.0	0.0	0.0	0.0	1.0	0.0
40515	0.0	0.0	0.0	0.0	1.0	0.0

40516 rows × 6 columns

```
df_type_5
```

	2016	2017	2018	2019	2020	2021
0	0.0	0.0	0.0	1.0	0.0	0.0
1	0.0	0.0	1.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0	0.0	0.0
4	1.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...
40511	0.0	0.0	0.0	0.0	1.0	0.0
40512	0.0	0.0	0.0	0.0	1.0	0.0
40513	0.0	0.0	0.0	0.0	1.0	0.0
40514	0.0	0.0	0.0	0.0	1.0	0.0
40515	0.0	0.0	0.0	0.0	1.0	0.0

40516 rows × 6 columns

```
# concatenate these one-hot-encoded versions of the categorical variables

df_cats_lyon=pd.concat([df_type_achat,df_type_2,df_type_3,df_type_4,df_type_5], axis=1)
df_cats_lyon.head()
```

	VEFA	ancien	appartement	maison	Lyon 1er Arrondissement	Lyon 2e Arrondissement	Lyon 3e Arrondissement	Ar
0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	
1	0.0	1.0	0.0	1.0	0.0	0.0	0.0	
2	0.0	1.0	1.0	0.0	0.0	0.0	0.0	
3	0.0	1.0	1.0	0.0	0.0	0.0	0.0	
4	0.0	1.0	1.0	0.0	0.0	0.0	0.0	

5 rows × 26 columns

Standardization

standard scale the other data

Not going to create a pipeline as I already imputed everything.

```
continuousImputing
```

	nombre_pieces	surface_logement	nombre_parkings
0	5.0	100.0	0.0
1	2.0	52.0	0.0
2	1.0	28.0	1.0
3	3.0	67.0	1.0
4	1.0	28.0	1.0
...	...	...	...
40511	2.0	34.0	1.0
40512	2.0	33.0	0.0
40513	2.0	23.0	0.0
40514	2.0	34.0	0.0
40515	2.0	30.0	0.0

40516 rows × 3 columns

```
continuousImputing.columns
```

```
Index(['nombre_pieces', 'surface_logement', 'nombre_parkings'], dtype='object')
```

```
scaler = StandardScaler()
```

```
lyons_continuous = scaler.fit_transform(continuousImputing)
```

```
lyons_continuous=pd.DataFrame(lyons_continuous,columns=continuousImputing.columns)
lyons_continuous
```

	nombre_pieces	surface_logement	nombre_parkings
0	1.870396	1.231567	-0.996559
1	-0.671310	-0.469113	-0.996559
2	-1.518546	-1.319453	0.666260
3	0.175925	0.062349	0.666260
4	-1.518546	-1.319453	0.666260
...	...	...	...
40511	-0.671310	-1.106868	0.666260
40512	-0.671310	-1.142299	-0.996559
40513	-0.671310	-1.496607	-0.996559
40514	-0.671310	-1.106868	-0.996559
40515	-0.671310	-1.248592	-0.996559

40516 rows × 3 columns

```
lyon_housing_final = pd.concat([df_cats_lyon,lyons_continuous],axis=1)
```

```
lyon_housing_final
```

	VEFA	ancien	appartement	maison	Lyon 1er Arrondissement	Lyon 2e Arrondissement	Lyon 3e Arrondissement	Ar
)	0.0	1.0	0.0	1.0	0.0	0.0	0.0	
l	0.0	1.0	0.0	1.0	0.0	0.0	0.0	
!	0.0	1.0	1.0	0.0	0.0	0.0	0.0	
}	0.0	1.0	1.0	0.0	0.0	0.0	0.0	
l	0.0	1.0	1.0	0.0	0.0	0.0	0.0	
..	...	...	...	...	...	...	...	
i11	0.0	1.0	1.0	0.0	0.0	0.0	0.0	
i12	0.0	1.0	1.0	0.0	0.0	0.0	0.0	
i13	0.0	1.0	1.0	0.0	0.0	0.0	0.0	
i14	0.0	1.0	1.0	0.0	0.0	0.0	0.0	
i15	0.0	1.0	1.0	0.0	0.0	0.0	0.0	

16 rows × 29 columns

```
y=lyon.loc[:,'prix']
type(y)
```

```
pandas.core.series.Series
```

```
y.head()
```

```
0    530000.0
1    328550.0
2     42500.0
3    180900.0
4     97000.0
Name: prix, dtype: float64
```

```
y.isnull().values.any()
```

```
False
```

Build a neural net regressor, a nearest neighbhor and a linear

use some metrics, what is the MSE?, the R2, the mean absolute value error?

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(lyon_housing_final, y,train_size=0.8,random_state=1)
```

Nural Net

```
from sklearn.neural_network import MLPRegressor

regr = MLPRegressor(hidden_layer_sizes=(6,3,),random_state=1, max_iter=50000, verbose=False)
regr.fit(X_train, y_train)
y_pred=regr.predict(X_train)
```

```
##Explained variance or R^2
## not sure what the diffrence btween this and r2_score is the numbers are diffrent at like the 10 millionths place
from sklearn.metrics import explained_variance_score

explained_variance_score(y_train,y_pred)
```

    0.7443733685835602

```
from sklearn.metrics import mean_squared_error

mean_squared_error(y_train, y_pred)

# Mean of the squared errors.
```

    6109777768.7059145

```
from sklearn.metrics import r2_score
r2_score(y_train, y_pred)
```

    0.744373342572046

```
from sklearn.metrics import mean_absolute_error

mean_absolute_error(y_train, y_pred)

## 48658.99353165955
```

    48658.99353165955

Nearest Neighbour

```
from sklearn.neighbors import KNeighborsRegressor

neigh = KNeighborsRegressor(n_neighbors=8)
neigh.fit(X_train, y_train)

y_pred_nn=neigh.predict(X_train)

explained_variance_score(y_train,y_pred_nn)
```

    0.7796836835835779

```
r2_score(y_train,y_pred_nn)
```

    0.7791875639067137

```
mean_absolute_error(y_train, y_pred_nn)
```

    44746.359509016715

```
mean_squared_error(y_train, y_pred_nn)
```

    5277676931.940449

Linear Regression

```
from sklearn.linear_model import LinearRegression

reg = LinearRegression().fit(X_train,y_train)
y_pred_lin=reg.predict(X_train)
```

```
X_train
```

	VEFA	ancien	appartement	maison	Lyon 1er Arrondissement	Lyon 2e Arrondissement	Lyon 3e Arrondissement
789	0.0	1.0	1.0	0.0	0.0	0.0	0.0
34722	0.0	1.0	1.0	0.0	0.0	0.0	0.0
32810	0.0	1.0	1.0	0.0	0.0	0.0	0.0
29287	0.0	1.0	1.0	0.0	0.0	0.0	0.0
16454	0.0	1.0	1.0	0.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...
7813	0.0	1.0	1.0	0.0	0.0	0.0	0.0
32511	1.0	0.0	1.0	0.0	0.0	0.0	0.0
5192	0.0	1.0	1.0	0.0	0.0	0.0	0.0
12172	0.0	1.0	1.0	0.0	0.0	1.0	0.0
33003	0.0	1.0	1.0	0.0	0.0	0.0	0.0

32412 rows × 29 columns

```
y_train
```

```
789      207400.0
34722    246300.0
32810    183000.0
29287    282500.0
16454    272400.0
           ...
7813      85000.0
32511    106100.0
5192     173000.0
12172    495000.0
33003    154500.0
Name: prix, Length: 32412, dtype: float64
```

```
# Squared of the errors mean
mean_squared_error(y_train, y_pred_lin)
```

```
6784509900.029959
```

```
#Absolute Value of the errors mean
mean_absolute_error(y_train, y_pred_lin)
```

```
53135.58291496977
```

```
# Explained variance or R^2

r2_score(y_train,y_pred_lin)
```

```
0.7161432618851447
```

use cross validation to figure out which model seems to be best

Nearest Neighbour

```
from sklearn.model_selection import cross_val_score

neigh2 = KNeighborsRegressor(n_neighbors=8)
scores = cross_val_score(neigh2, X_train, y_train, cv=5,scoring='r2')
```

```
np.mean(scores)
```

```
0.7068407274531442
```

```
scoresTest = cross_val_score(neigh2, X_test, y_test, cv=5,scoring='r2')
```

```
np.mean(scoresTest)
```

    0.6763891592524371

```
reg2 = LinearRegression()
scores2 = cross_val_score(reg2, X_train, y_train, cv=10,scoring='r2')
```

```
np.mean(scores2)
```

    0.7146099874752011

```
scores2Test = cross_val_score(reg2, X_test, y_test, cv=10,scoring='r2')
```

```
np.mean(scores2Test)
```

    0.7341273663024189

Looks underfitted becuase the Test score returns a higher r2 on average

Nural Net

```
regr2 = MLPRegressor(hidden_layer_sizes=(6,3,),random_state=1, max_iter=50000, verbose=False)
Score3Input = regr2.fit(X_train, y_train)
scores3 = cross_val_score(Score3Input, X_train, y_train, cv=4,scoring='r2')
```

```
np.mean(scores3)
```

    0.7380983019890226

```
scores3Test = cross_val_score(Score3Input, X_test, y_test, cv=4,scoring='r2')
```

```
np.mean(scores3Test)
```

    0.7520919304772871

I just wanna see

```
np.mean(scores3[1:])
```

    0.7526459886318794

```
np.mean(scores3Test)
```

    0.7520919304772871

```
scores3Test
```

    array([0.75638718, 0.75047416, 0.76733262, 0.73417375])

```
scores2Test
```

    array([0.69089575, 0.7762123 , 0.72312221, 0.76375536, 0.76570073,
           0.77558544, 0.73543911, 0.69777858, 0.706851 , 0.70593318])

```
scores2
```

```
array([0.71969299, 0.630208  , 0.70695979, 0.71505285, 0.74834515,
       0.72832398, 0.71646985, 0.70414288, 0.73809456, 0.73880983])
```

```
scores2Test
```

```
array([0.69089575, 0.7762123 , 0.72312221, 0.76375536, 0.76570073,
       0.77558544, 0.73543911, 0.69777858, 0.706851  , 0.70593318])
```

```
scores
```

```
array([0.66647924, 0.71110073, 0.71976349, 0.71661357, 0.7202466 ])
```

```
scoresTest
```

```
array([0.63921822, 0.7200684 , 0.7191613 , 0.65176442, 0.65173346])
```

The nural-net seem to be the best as it returns the best training resutls and returns a even better testing result.

use EPI5 to understand what the most important predictors are

```
!pip install eli5
```

```
Collecting eli5
  Downloading eli5-0.13.0.tar.gz (216 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 216.2/216.2 kB 5.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: attrs>17.1.0 in /usr/local/lib/python3.10/dist-packages (from eli5) (23.2.0)
Requirement already satisfied: jinja2>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from eli5) (3.1.3)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from eli5) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from eli5) (1.11.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from eli5) (1.16.0)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.10/dist-packages (from eli5) (1.2.2)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from eli5) (0.20.1)
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.10/dist-packages (from eli5) (0.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2>=3.0.0->eli5) (2.1.5)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20->eli5) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20->eli5) (3.2.0)
Building wheels for collected packages: eli5
  Building wheel for eli5 (setup.py) ... done
  Created wheel for eli5: filename=eli5-0.13.0-py2.py3-none-any.whl size=107717 sha256=53e5a489c0a09377fcb4a0f4e0a92f6c8e3db57e73a11c6a
  Stored in directory: /root/.cache/pip/wheels/b8/58/ef/2cf4c306898c2338d51540e0922c8e0d6028e07007085c0004
Successfully built eli5
Installing collected packages: eli5
Successfully installed eli5-0.13.0
```

```
import eli5
from eli5.sklearn import PermutationImportance
```

```
perm = PermutationImportance(regr, random_state=1).fit(X_train, y_train)
```

```
X_train
```

Lyon 3e ndissement	Lyon 4e Arrondissement	Lyon 5e Arrondissement	Lyon 6e Arrondissement	...	UnderConstruction	2016
0.0	0.0	0.0	0.0	...	0.0	0.0
0.0	0.0	0.0	0.0	...	0.0	0.0
0.0	0.0	0.0	0.0	...	0.0	1.0
0.0	0.0	0.0	0.0	...	0.0	0.0
1.0	0.0	0.0	0.0	...	0.0	0.0
...	...	...	...	...	...	...