DIPARTIMENTO DI MATEMATICA E
INFORMATICA

House Price Competition

Machine Learning – Project Report

GIUSEPPINA PIA VARANO
LARA SCENI

Accademic Year 2022/23

Summary

Business Understanding

What is the House Price Competition? What is the goal?

Data Understanding

- Presentation of the dataset
- Correlations between features
- Nulls check

Data Preparation

- Encoding of categorical variables
- Selection of the features

Modelling

- Scaling
- Decision Trees
- Naive Bayes
- Neural Network

Evaluation

- Methods (Confusion Matrix, ROC and learning curves
- Simple Split
- min_sample_split (DTs parameter)
- K-folds train-validation split

Business Undestanding

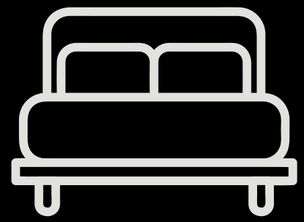
The "House Price Competition" is a competition hosted on the Kaggle platform, which aims to develop machine learning models to predict house prices. The competition is based on a dataset that contains various characteristics (features) of the homes, such as the number of rooms, the size of the land, the geographical location and other related information.

The goal of the competition is to create a model that can predict the price of a home based on these characteristics. The categories are:

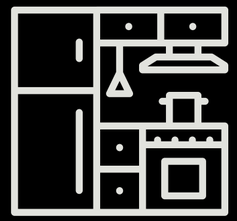
– **Low** (Saleprice <= 150000)
– **Medium** (150000 < Saleprice < 300000)
– **High** (Saleprice => 300000)

Kaggle also provides a separate training set and test set for evaluating model performance.

What are the kitchen conditions?



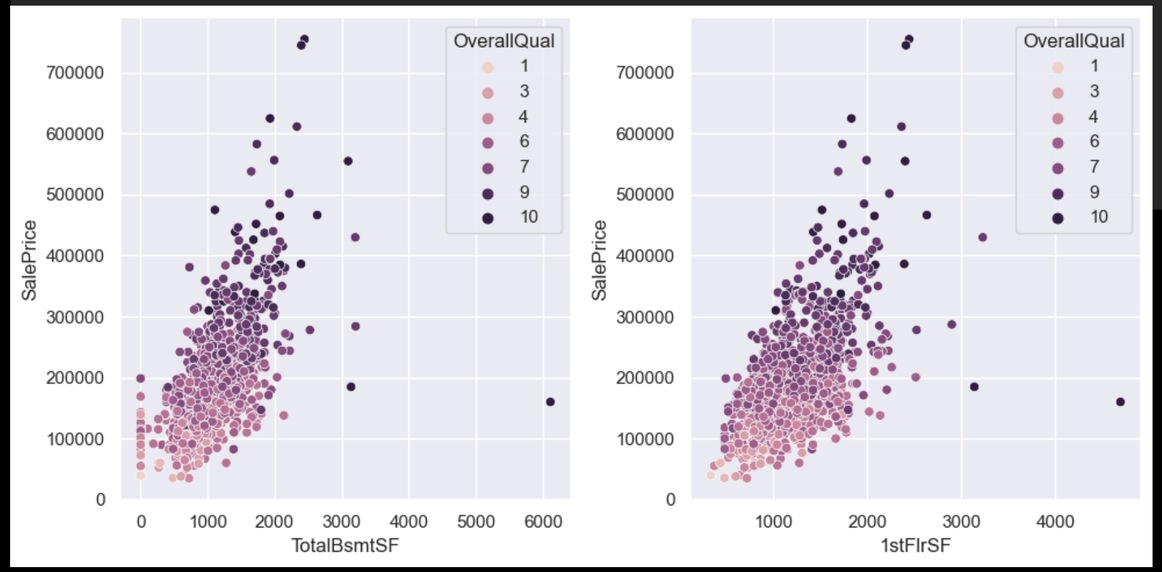
How many bedrooms?

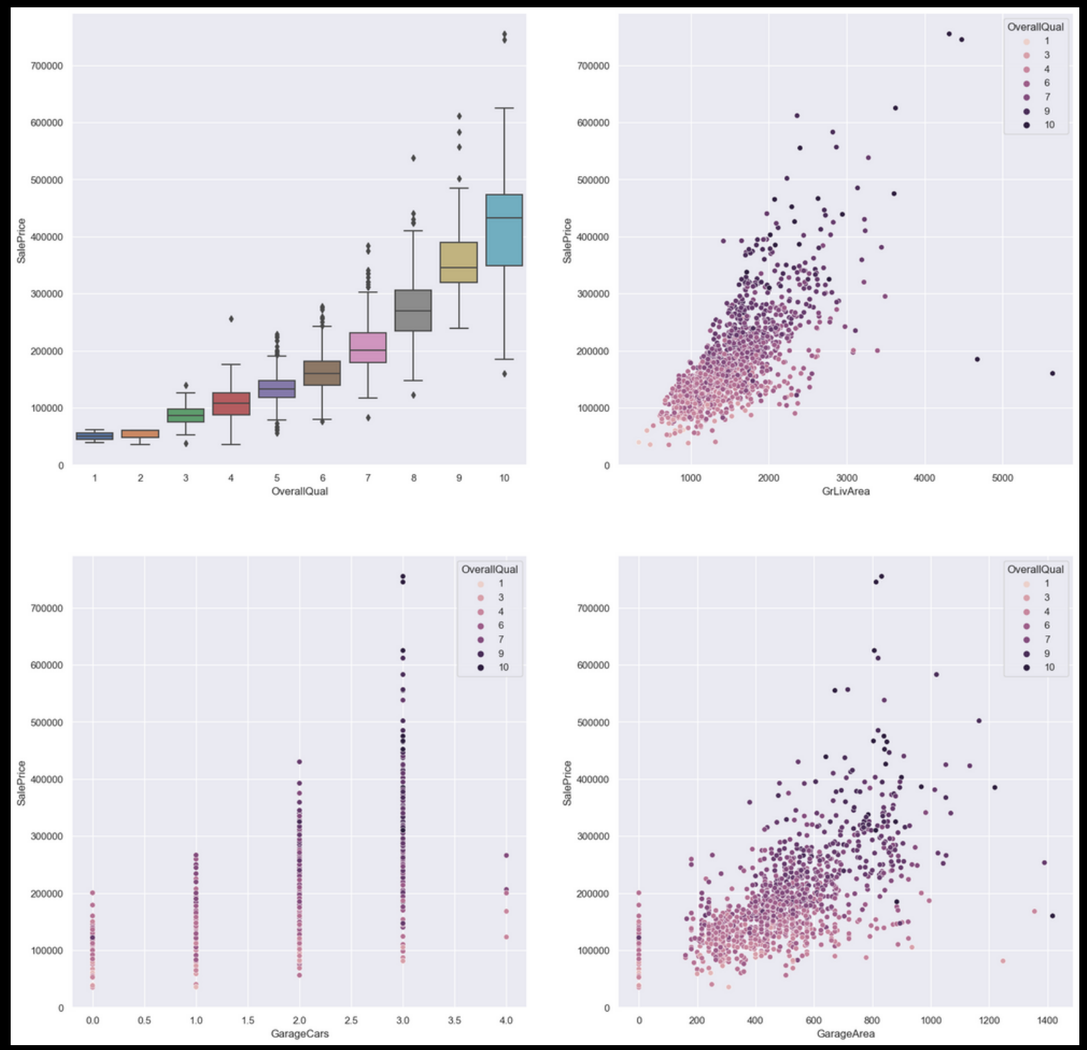


And what about the porch?

Data Understanding

What about the dataset?





Correlations

One way to analyze the numerical features of the dataframe is to understand how correlated they are with the selling price of a house. It is indeed logical to assume that certain characteristics such as the house's area, garage size, and the number of bathrooms can significantly impact the valuation of a property.

	types	counts	uniques	nulls	miss_ratio	skew	curtosi	corr SalePrice
SalePrice	int64	1460	[208500, 181500, 223500, 140000, 250000, 14300...	0	0.000000	1.882876	6.536282	1.000000
OverallQual	int64	1460	[7, 6, 8, 5, 9, 4, 10, 3, 1, 2]	0	0.000000	0.216944	0.096293	0.790982
GrLivArea	int64	1460	[1710, 1262, 1786, 1717, 2198, 1362, 1694, 209...	0	0.000000	1.366560	4.895121	0.708624
GarageCars	int64	1460	[2, 3, 1, 0, 4]	0	0.000000	-0.342549	0.220998	0.640409
GarageArea	int64	1460	[548, 460, 608, 642, 836, 480, 636, 484, 468	0	0.000000	0.179981	0.917067	0.623431

Evaluation of Pearson coefficient of correlation between each feature and SalePrice

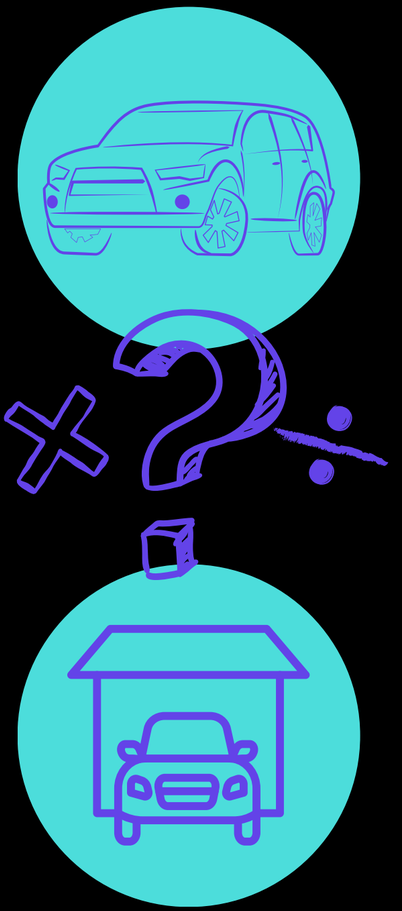
Definition of new features more correlated with the selling price

Remotion of outliers to improve correlation values of the features

Example: where are my cars?

When dealing with garage-related features, for example, we can ask “how many cars can fit in a garage”? So, it seems like a good idea to combine the features related to the number of cars with the total garage area. We conducted the following experiments

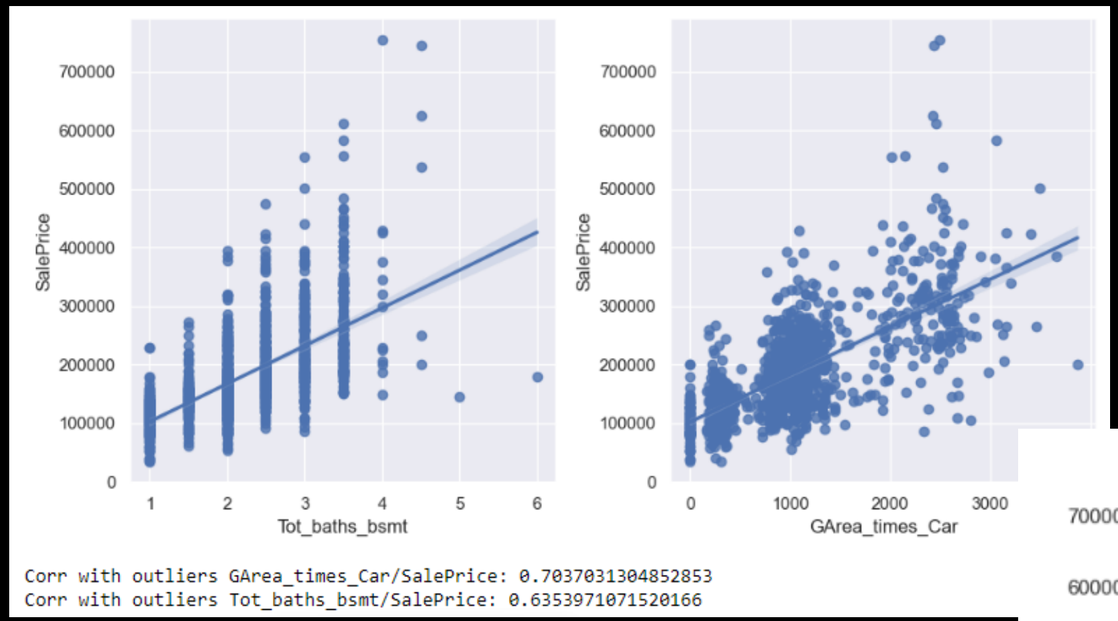
```
house_garage = house['SalePrice', 'GarageCars' , 'GarageArea']]
house_garage['GArea_by_Car'] = house.GarageArea/house.GarageCars
house_garage['GArea_times_Car'] = house.GarageArea * house.GarageCars
details = rstr (house_garage, 'SalePrice')
display(details.sort_values(by = 'corr SalePrice', ascending = False))
```



If we repeat the same reasoning with the number of baths and the basement area occupied by each of them, we should obtain:

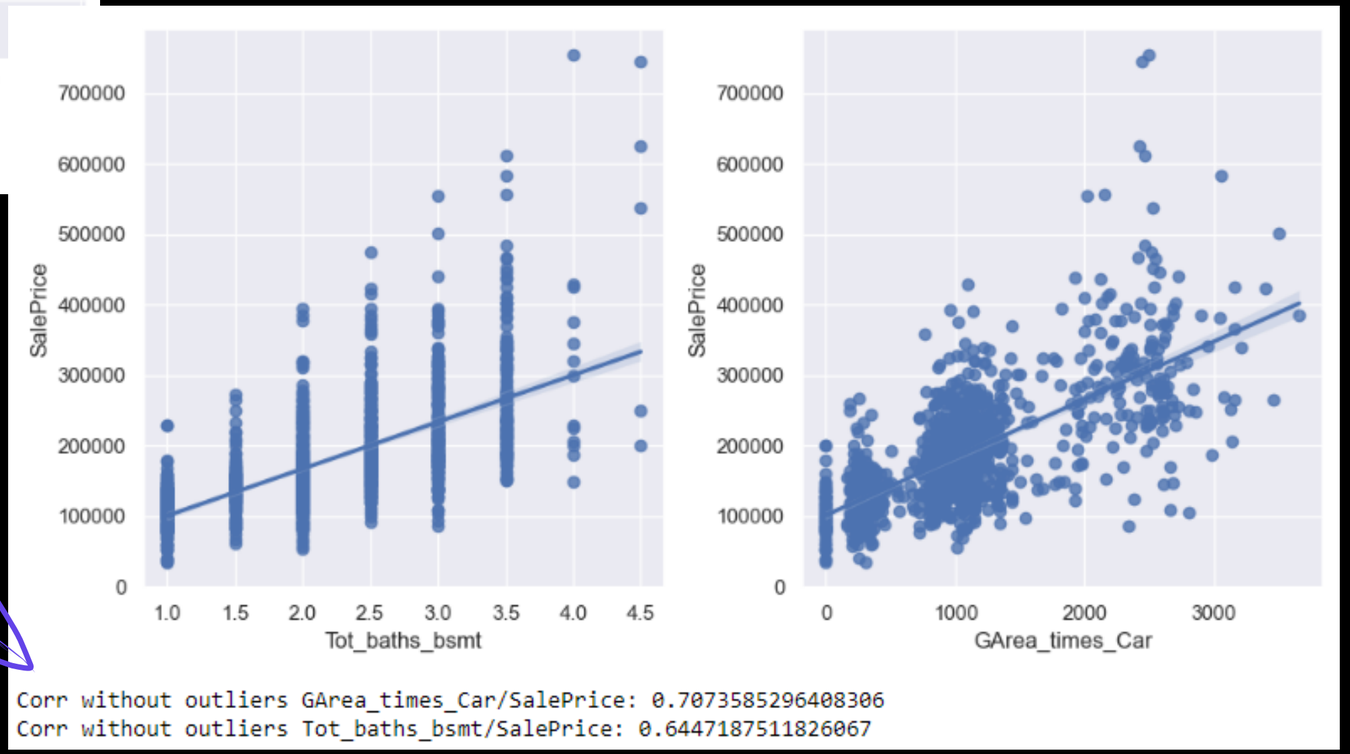
	types	counts	uniques	nulls	miss_ratio	skew	curtosi	corr SalePrice
SalePrice	int64	1455	[208500, 181500, 223500, 140000, 250000, 14300...	0	0.00000	1.883885	6.530542	1.000000
GArea_times_Car	int64	1455	[1096, 920, 1216, 1926, 2508, 960, 1272, 968, ...	0	0.00000	1.077869	1.189616	0.703703
GarageCars	int64	1455	[2, 3, 1, 0, 4]	0	0.00000	-0.359031	0.207334	0.645516
GarageArea	int64	1455	[548, 460, 608, 642, 836, 480, 636, 484, 468, ...	0	0.00000	0.018023	0.411623	0.641664
GArea_by_Car	float64	1374	[274.0, 230.0, 304.0, 214.0, 278.6666666666667...	81	5.56701	2.828027	18.488690	-0.012187

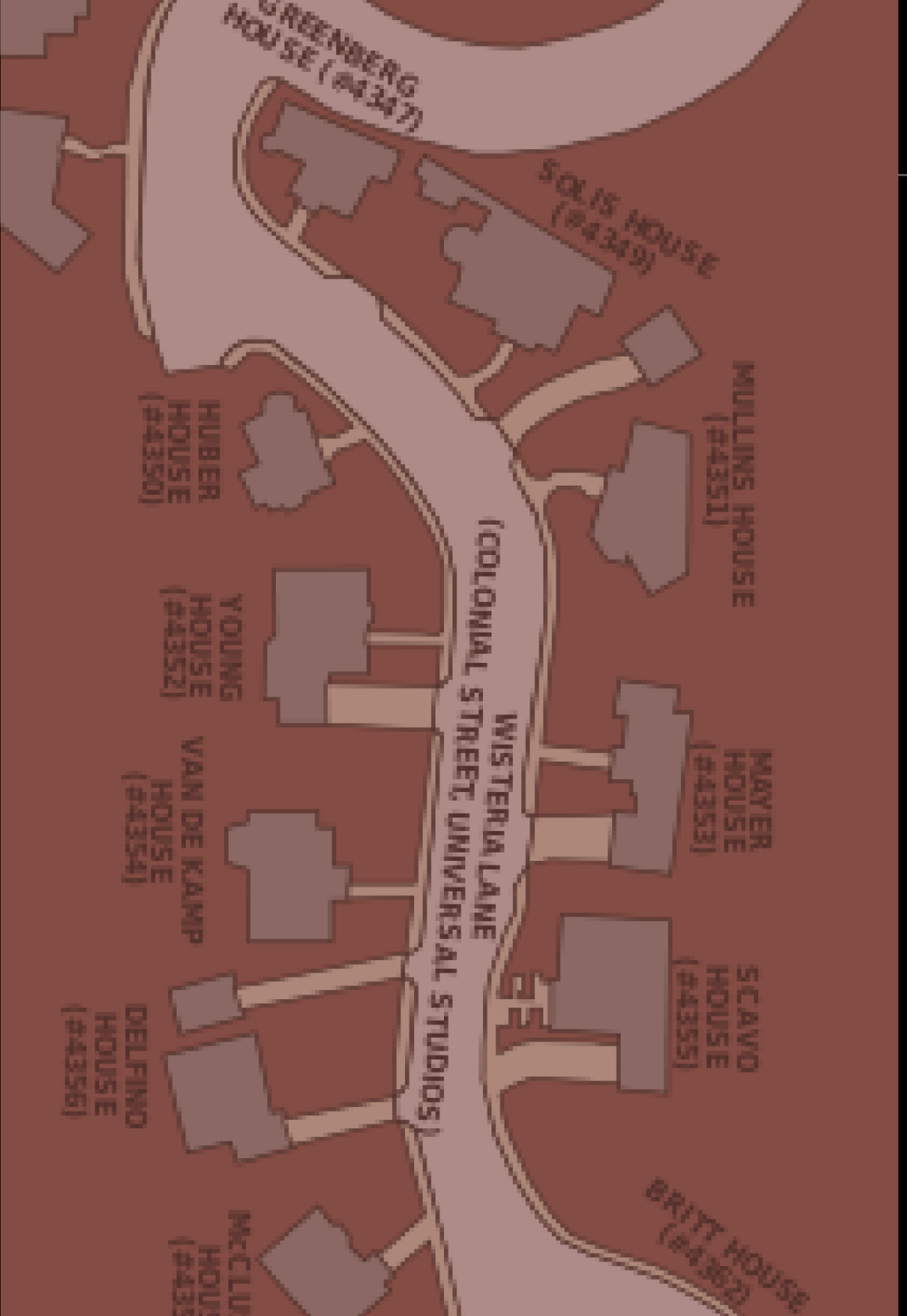
	types	counts	uniques	nulls	miss_ratio	skew	curtosi	corr SalePrice
SalePrice	int64	1455	[208500, 181500, 223500, 140000, 250000, 14300...	0	0.0	1.883885	6.530542	1.000000
Tot_baths_bsmt	float64	1455	[3.5, 2.5, 2.0, 3.0, 4.0, 1.0, 1.5, 6.0, 4.5, ...	0	0.0	0.246647	-0.167873	0.635397
Total-baths	float64	1455	[2.5, 2.0, 1.0, 1.5, 3.0, 3.5, 0.5, 0.0]	0	0.0	0.123868	-0.881474	0.599406
FullBath	int64	1455	[2, 1, 3, 0]	0	0.0	0.029872	-0.868248	0.561666
HalfBath	int64	1455	[1, 0, 2]	0	0.0	0.679947	-1.069607	0.283578
BsmtFullBath	int64	1455	[1, 0, 2, 3]	0	0.0	0.590046	-0.863557	0.228782
Total_bsmts	float64	1455	[1.0, 0.5, 0.0, 2.0, 1.5, 3.0]	0	0.0	0.513184	-0.843951	0.226562
BsmtHalfBath	int64	1455	[0, 1, 2]	0	0.0	4.095176	16.324530	-0.016981



Features related to porch, slope of estates, year of construction of garages have been treated in the same way.

At this level, it's possible to remove outliers and observe how this can alter the correlation with SalePrice



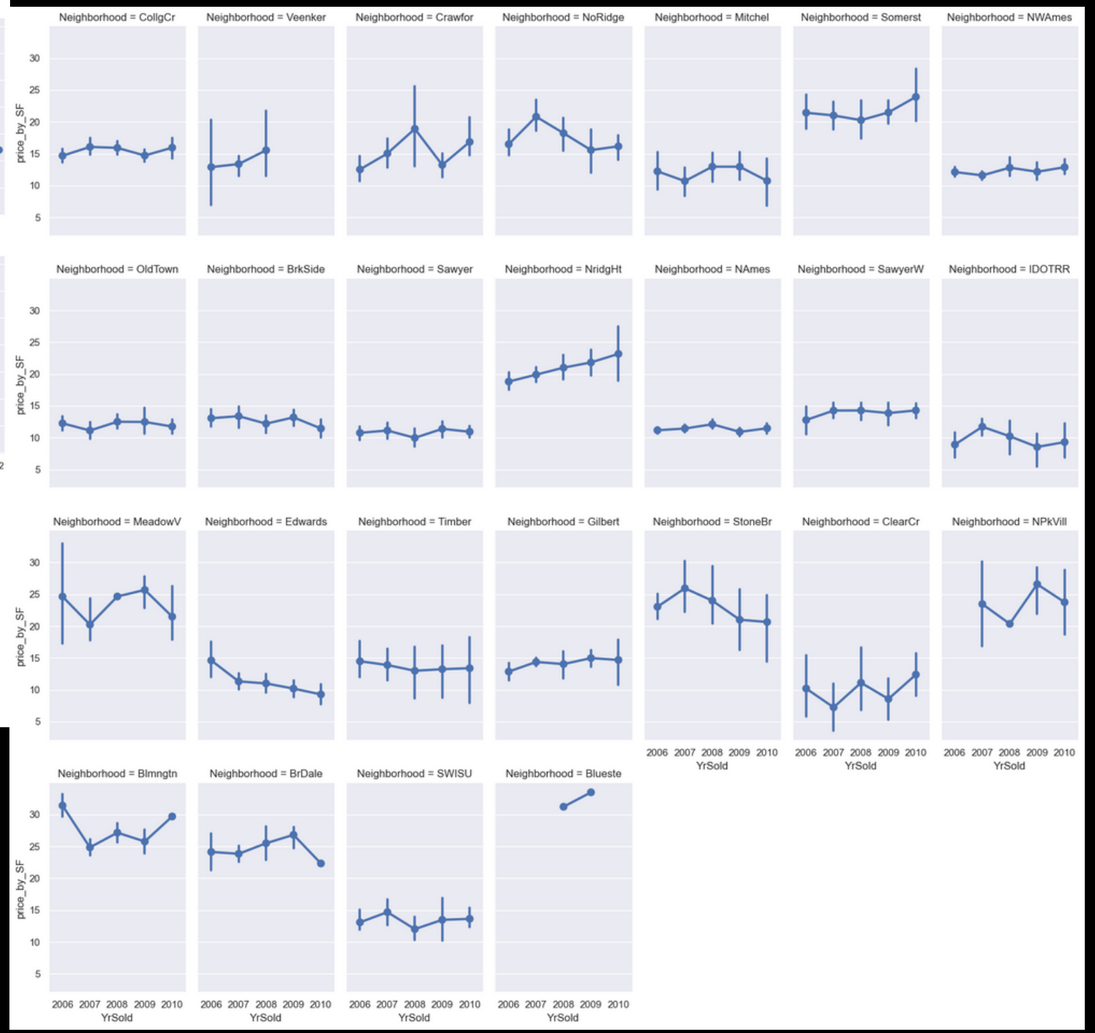


Purchasing advice: when and where buying a new estate?

According to ATTOM, even seasonality influences the selling price of an estate. We can try to relate the season with Neighborhood and the total area. We can firstly define 2 new dataframes with the total areas of estates as follows: myneig=month+neighbours and myneig=year+neighbours

It is evident that there are missing data in both dataframes. Using Python's append() function, we filled in the gaps by adding either the minimum/maximum value of the series or the mean, depending on whether the curve trend is decreasing/increasing or stable. Among all the neighborhood-related data, we created a dictionary such that neighborhoods with similar data and complete datasets belong to the same category.

Purchasing anvice: when and where buying a new estate?



Nulls Check

The standard approach for handling missing values in most modeling approaches is complete-case analysis, where only those observations without any missing values are used in the training process.

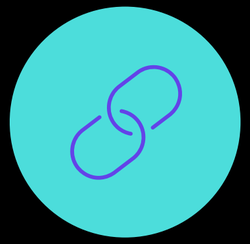
We can think that we could eliminate these missing values from the training set

The problem is even a small number of missing values can cause enormous loss of data in high dimensions.

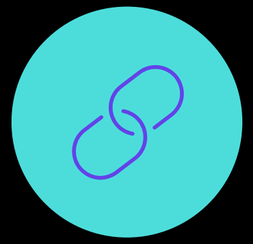
Imputation is a common process to avoid missing values in observations used during training, validation, test and deployment processes.

Nulls Check: coming back to Garage!

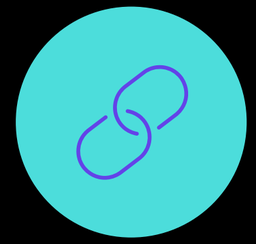
We firstly grouped by Garage Type, then:



We looked for missing values



How many times Garage Area is equal to zero?



How many times Garage Cars is equal to zero?

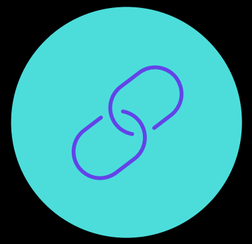
```
# Group by GarageType and fill missing value with median where GarageType=='Detchd' and 0 for the others
cmedian = all_data[all_data.GarageType=='Detchd'].GarageArea.median()
print("GarageArea median of Type Detchd:", cmedian)
all_data.loc[all_data.GarageType=='Detchd', 'GarageArea'] = all_data.loc[all_data.GarageType=='Detchd',
                                                            'GarageArea'].fillna(cmedian)
all_data.GarageArea = all_data.GarageArea.fillna(0)

cmedian = all_data[all_data.GarageType=='Detchd'].GarageCars.median()
print("GarageCars median of Type Detchd:", cmedian)
all_data.loc[all_data.GarageType=='Detchd', 'GarageCars'] = all_data.loc[all_data.GarageType=='Detchd',
                                                            'GarageCars'].fillna(cmedian)
all_data.GarageCars = all_data.GarageCars.fillna(0)
```

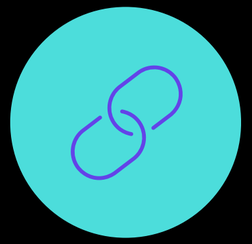
We filled the missing values, when ‘GarageType == Detch’, with the median or the mode of each feature

Nulls Check: and now the Basement!

We firstly grouped by Basement, then:



We looked for missing values



When integer features where equal to zero

We checked which was the most common attribute after the missing value

```
# No Basement è il comune BsmtExposure
display(all_data.BsmtExposure.value_counts())

# Sostituiamo i nulls Exposure con Av dove TotalBsmntSF è maggiore di zero
all_data.loc[(~all_data.TotalBsmtSF.isnull()) & (all_data.BsmtExposure.isnull()) & (all_
```

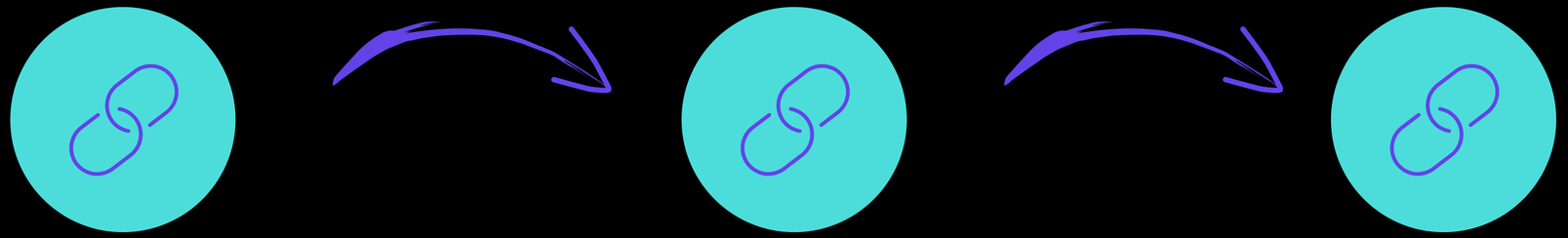
Data Preparation

Encoding

Features selection

Encoding

The **OneHotEncode** transforms certain features into a series of columns that complement the X matrix:



It counts all the categories present in the features

The analyzed features are replaced with as many columns as there are categories

All the values in the new columns will be either 0 or 1

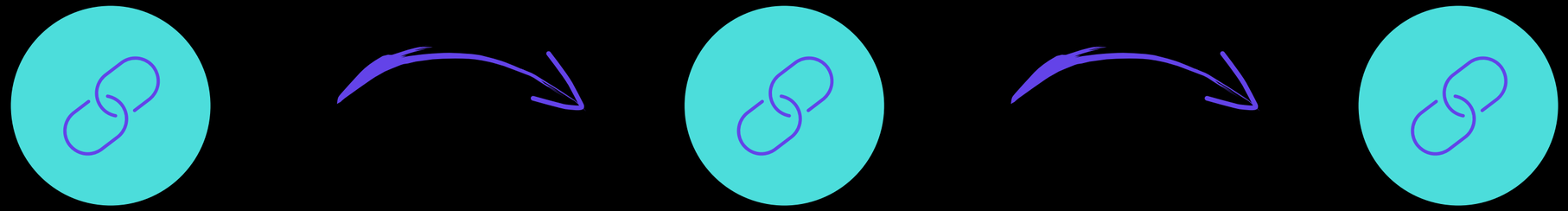
A little example:

the case of a matrix related to real estate sales, we might find repeated values such as ”studio apartment,” ”one-bedroom”, ”two-bedroom.”

In our case, we have three categories, so three columns will be added to the X matrix in place of the processed column. Each column corresponds to a category;

For example, if the columns represent, in order, ”studio apartment,” ”one-bedroom,” and ”two-bedroom,” and a row’s original value was ”studio apartment,” a 1 will be placed in the first column, and 0s will be placed in the others, and so on

Features Selection



Backward Elimination

116 features from 256

The Lasso Function

89 features

Modelling

The Classification algorithm

Decision Trees

- criterion “entropy”
- criterion “gini”

Naive Bayes

Neural Network

```
limiti_fasce = [0, 150000, 300000, np.inf]
etichette_fasce = ['Low', 'Medium', 'High']
df_y_train['Price_tag']  = pd.cut(df_y_train['SalePrice'],
                  bins = limiti_fasce, labels = etichette_fasce, right = False)
```

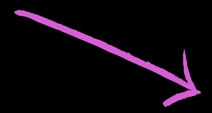
What is the classification algorithm?

Based on training data, the Classification algorithm is a Supervised Learning technique used to categorize new observations. In classification, a program uses the dataset or observations provided to learn how to categorize new observations into various classes or groups.

The Classification algorithm uses labeled input data because it is a supervised learning technique and comprises input and output information. A discrete output function (y) is transferred to an input variable in the classification process (x)

```
x = np.array(RFEcv_df.values)
y = np.array(df_y_train2.values)
seed = 101
test_size = .3
val_size = .2
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=test_size, random_state=seed)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=val_size, random_state=seed)
```

dataframe con le features selezionate dalla Lasso



```
array(['Medium', 'Low', 'Medium', 'Medium', 'Low', 'Medium', 'Medium',
       'Low', 'Low', 'Low', 'Low', 'Low', 'Low', 'Low', 'Medium', ...
```

Decision Trees

- **criterion "entropy"**
- **criterion "gini"**

The strengths of decision trees are that they are very easy to implement, and they are very intuitive. In fact, the results of the model are quite easy to explain to non-technical personnel. Decision trees can be fit very quickly and can score new costumers very easily. As a drawback, decision trees are very unstable models, which means that any minor changes in the training data set can cause substantial changes in the structure of the tree.

Classification metrics:

	precision	recall	f1-score	support
High	0.78	0.50	0.61	14
Low	0.87	0.76	0.81	86
Medium	0.77	0.88	0.82	104
accuracy			0.80	204
macro avg	0.80	0.71	0.75	204
weighted avg	0.81	0.80	0.80	204

Classification metrics for Decision Tree Classifier, criterion "entropy"

Classification metrics:

	precision	recall	f1-score	support
High	0.80	0.57	0.67	14
Low	0.84	0.86	0.85	86
Medium	0.83	0.85	0.84	104
accuracy			0.83	204
macro avg	0.82	0.76	0.79	204
weighted avg	0.83	0.83	0.83	204

Classification metrics for Decision Tree Classifier, criterion "gini"

Naive Bayes

Naive Bayes is very simple. Despite being a simple and dated algorithm, it still solves some classification problems with reasonable efficiency. It requires knowledge of all the data of the problem, in particular simple and conditional probabilities, information which is often difficult to obtain. It provides a naive approximation of the problem because it doesn't consider the correlation between the characteristics of the istance.

Classification metrics:

	precision	recall	f1-score	support
High	0.00	0.00	0.00	14
Low	0.00	0.00	0.00	86
Medium	0.51	0.97	0.67	104
accuracy			0.50	204
macro avg	0.17	0.32	0.22	204
weighted avg	0.26	0.50	0.34	204

Classification metrics for Gaussian Naive Bayes

Neural Network

Neural networks are one of the fastest scoring nonlinear models as they can efficiently score large volumes of data. For example, neural networks are used by banks to assess all their credit card transactions for potential fraud patterns. Although neural networks are one of the fastest scoring nonlinear models, they take a lot of time when fitting, or training, the model. This is especially true as the number of hidden units and layers increase. Although neural networks have unlimited flexibility, they sometimes do not perform as well as simpler models such as linear

Classification metrics:

	precision	recall	f1-score	support
High	0.30	1.00	0.47	14
Low	0.91	0.86	0.89	86
Medium	0.87	0.64	0.74	104
accuracy			0.76	204
macro avg	0.70	0.83	0.70	204
weighted avg	0.85	0.76	0.78	204

Classification metrics for Neural Network

Evaluation

Table of confusion

Roc Curves

Learning Curves

Evaluation of models based on classic train-test split

Modification of min_sample_split in DT models and evaluation

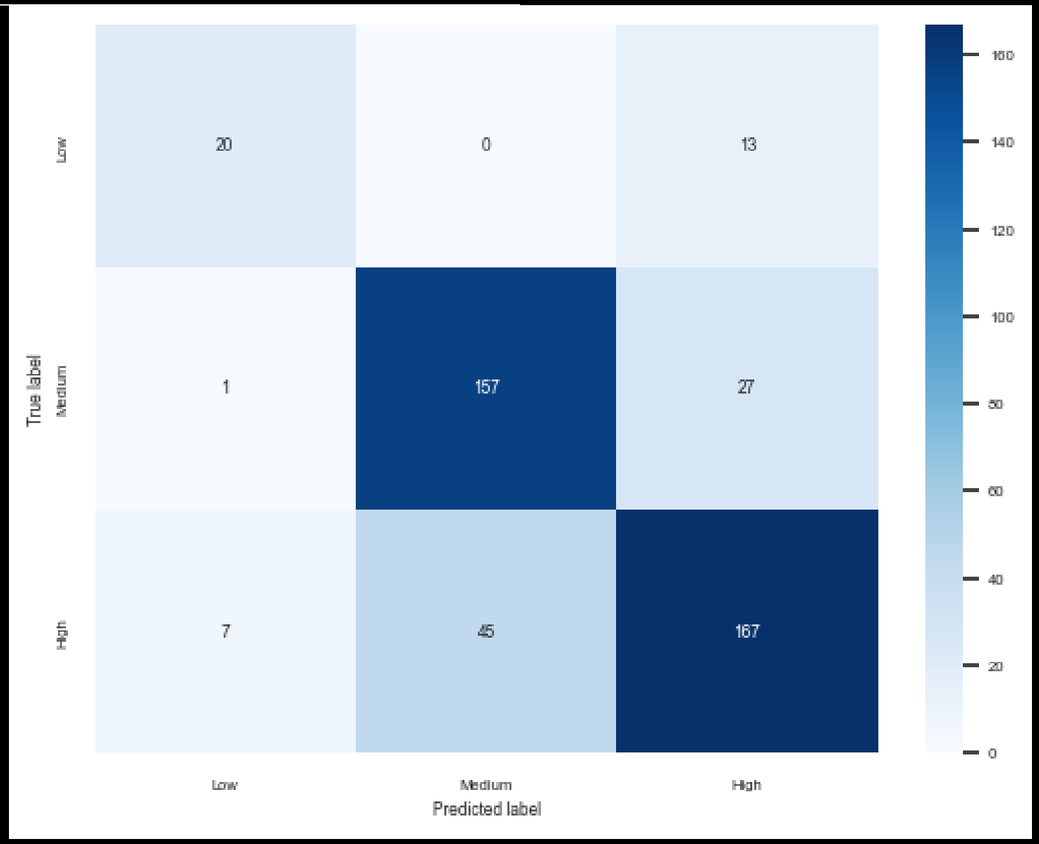
Evaluation of models based k-Fold train-validation-test split

Evaluation of models based on classic train-test split

In our case, one of the two decision trees was the model with the highest F1 score in training phase, so it has been tested on test set.

Test of confusion

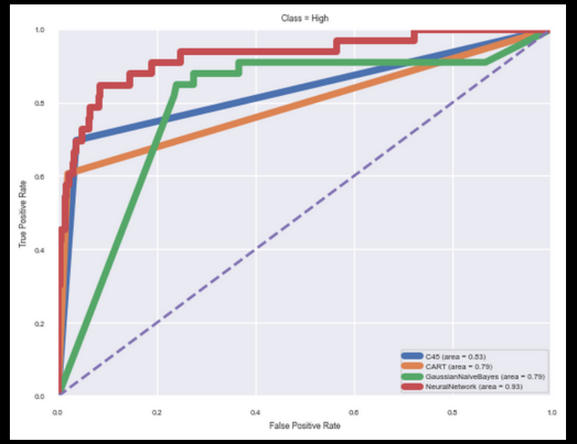
- **True positives**: are when you predict an observation belongs to a class and it actually does belong to that class
- **True negatives**: are when you predict an observation does not belong to a class and it actually does not belong to that class
- **False positives**: occur when you predict an observation belongs to a class when in reality it does not
- **False negatives**: occur when you predict an observation does not belong to a class when in fact it does

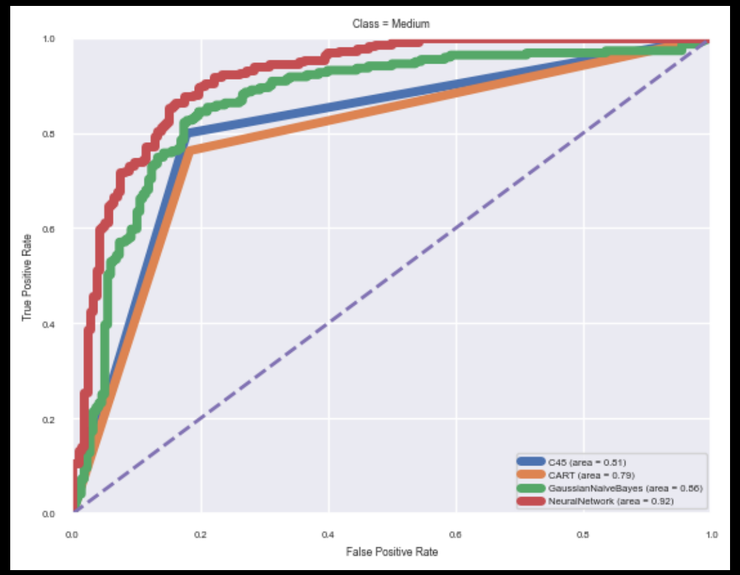


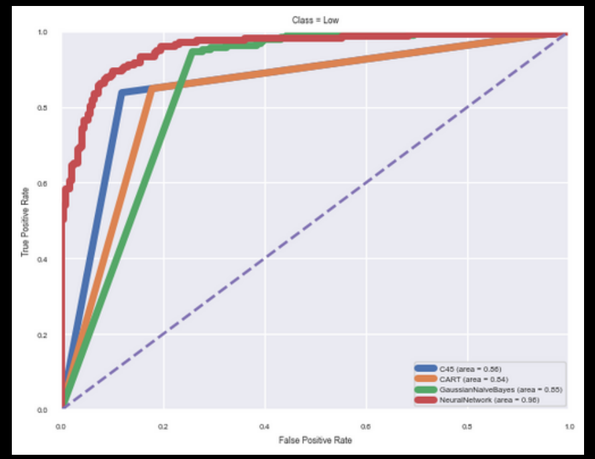
Evaluation of models based on classic train-test split

- The ROC curve is a graphical display that gives a measure of the predictive accuracy of a supervised classification model with binary response.
- AUC is a scalar value that summarizes the performance of a binary classifier across various thresholds.
- We can observe that, despite the two decision trees being the top-performing models during training, the ROC curves of the neural network are much closer to the ideal case compared to the others, indicating a high True Positive Rate (TPR) and a low False Positive Rate (FPR).

ROC curves



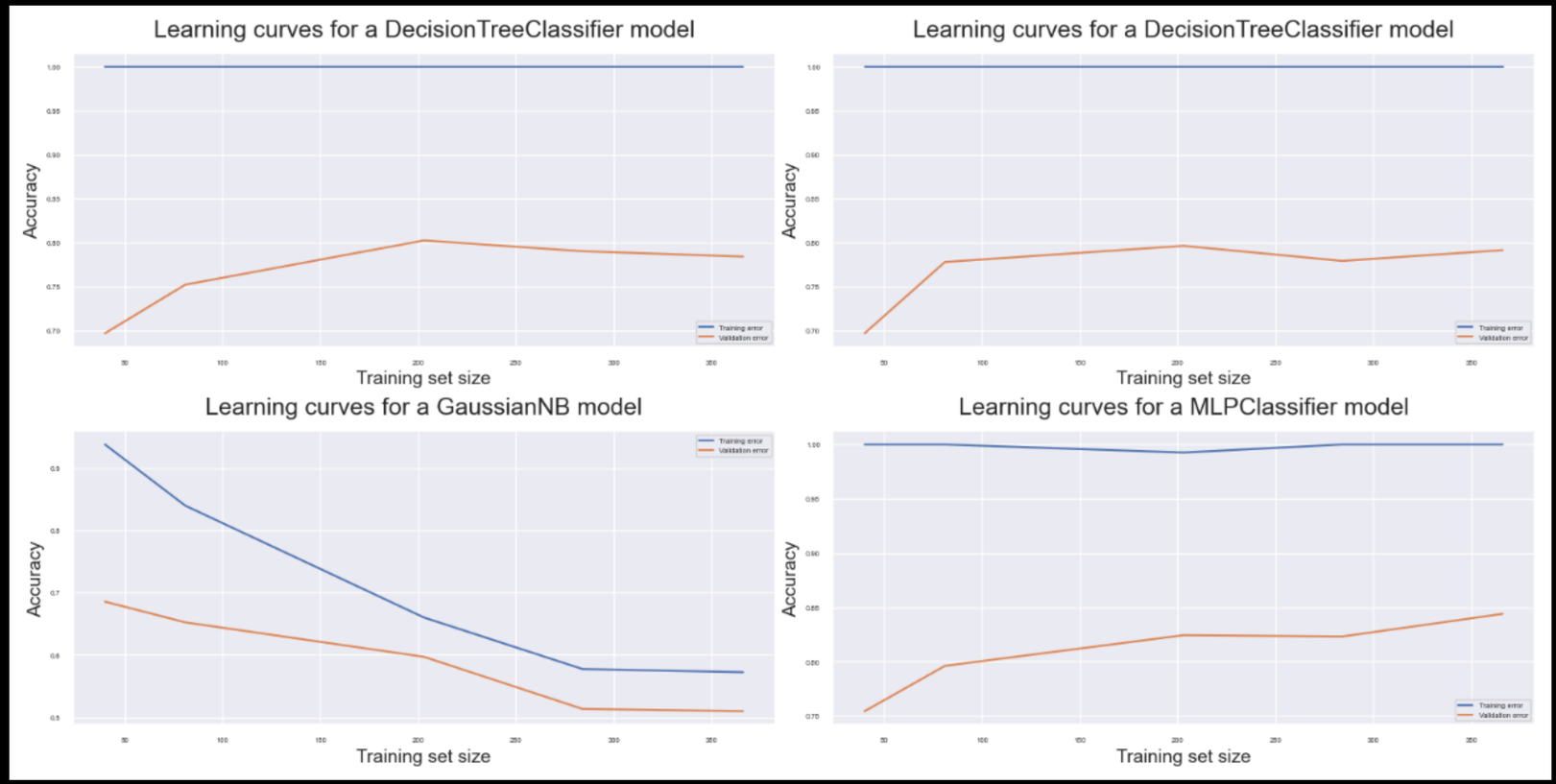




Evaluation of models based on classic train-test split

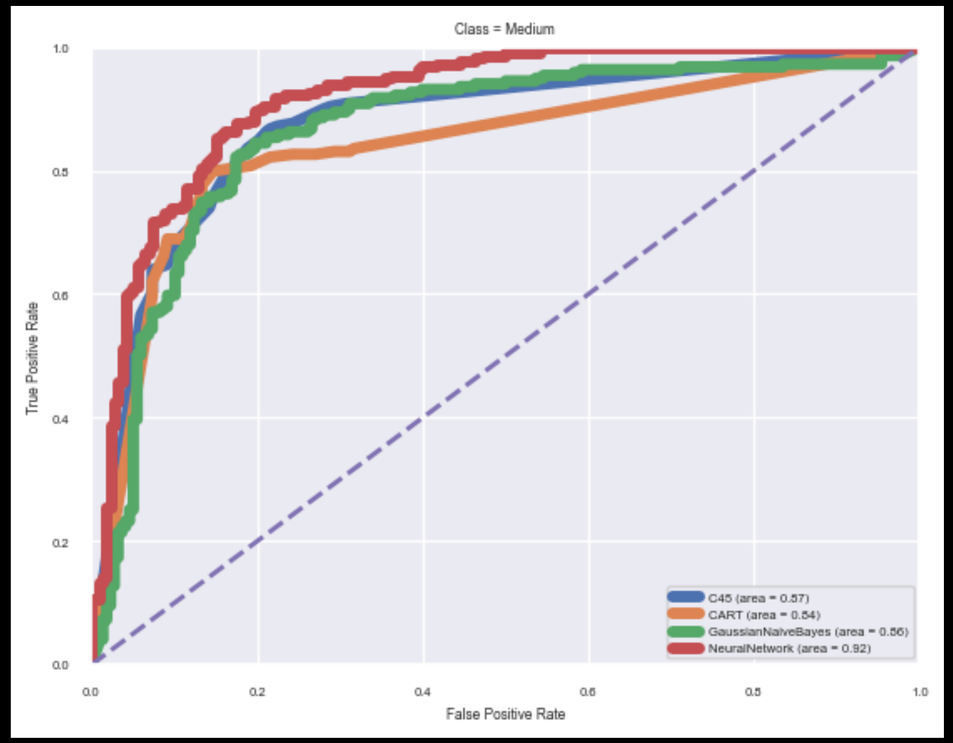
Learning curves

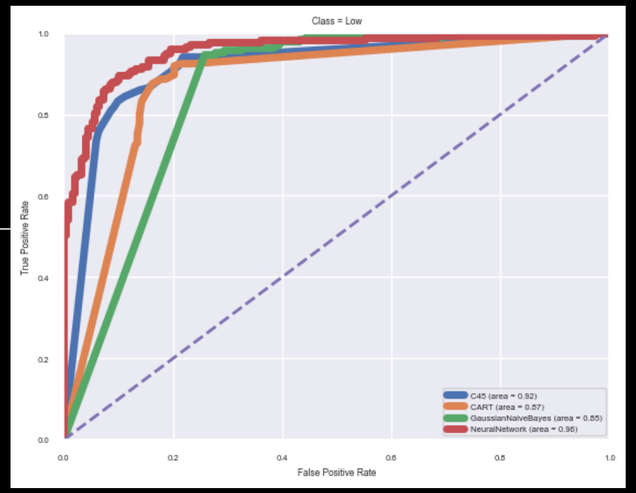
- Learning curves are an important tool in the field of machine learning and model training. They provide a visual representation of how a model's performance evolves as the amount of data used for training varies.
- It immediately catches the eye how the training curve of the two decision trees is entirely flat, indicating that the model is overfitting the data: the test curves, on the other hand, as the sampled data increases, learn up to a few portions of the training set and then reach a plateau.

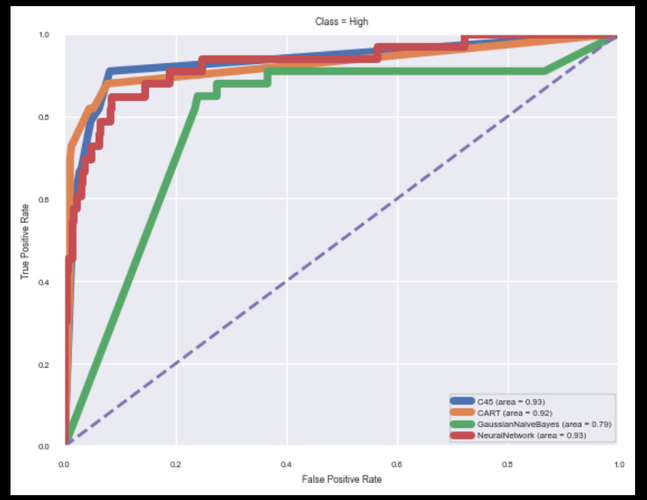


Modification of min_sample_split in DT models and evaluation

- Below, we present the ROC curves for the DecisionTree model trained on the test set. Subsequently, learning curves were plotted.
- Increasing the number of min_samples_split values, even the ROC curves of the decision trees during the evaluation phase follow the trend of the neural network, consequently increasing the area under the curve.

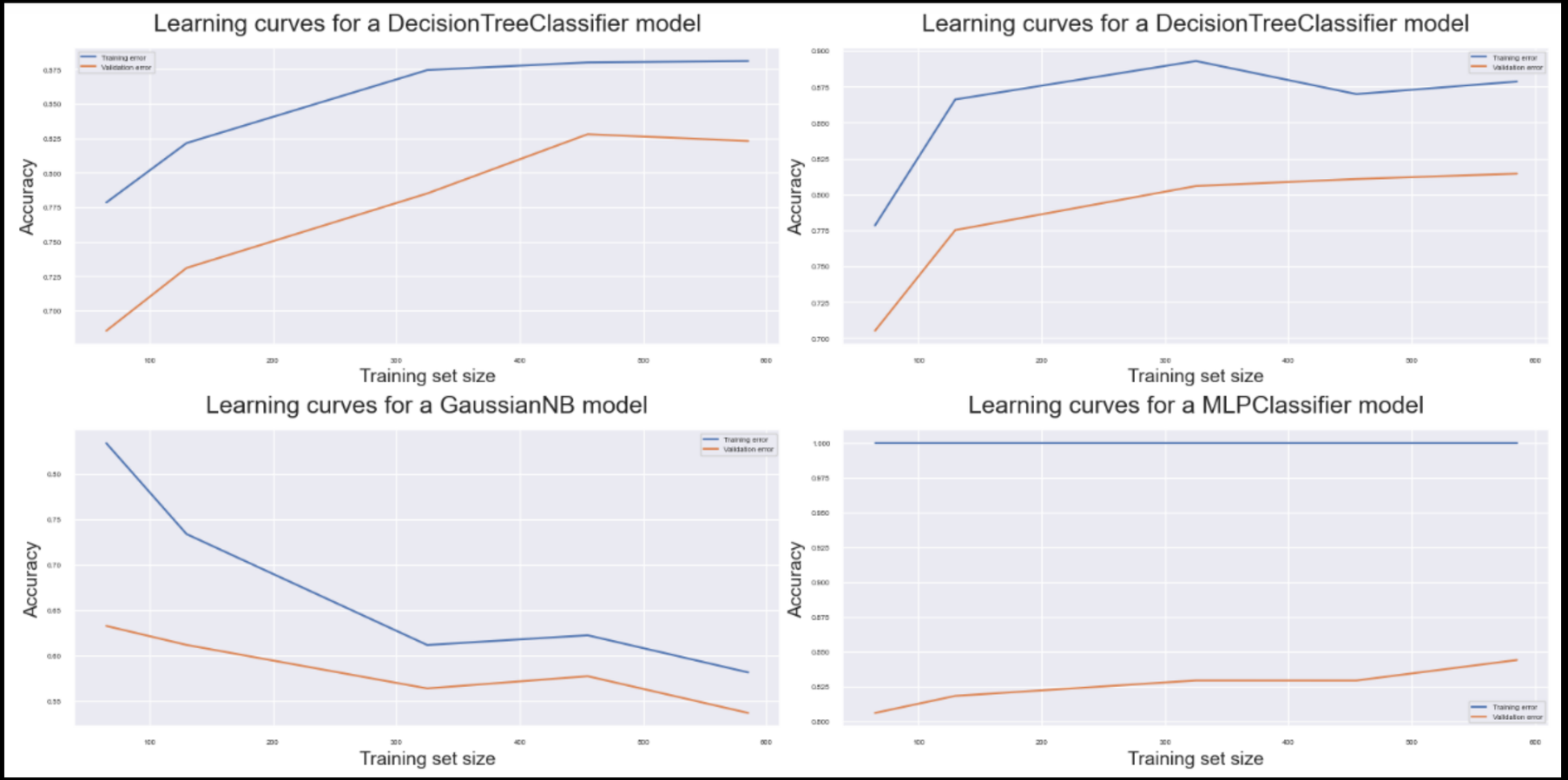






Modification of min_sample_split in DT models and evaluation

We can therefore assert that these models are now less prone to creating classification leaves based solely on a few data points. It is also reasonable to expect that the training curves are not flat anymore, but the gap between them and the test curves is smaller than before.



Evaluation of models based k-Fold train-validation-test split

K-Fold

K-fold cross-validation shuffles the original data set into training and validation.

```
test_size = .3
val_size = .2
x_train, x_test, y_train, y_test =
    train_test_split(x, y, test_size=test_size,
        random_state=seed)

kf = KFold(n_splits=5, shuffle=True, random_state=seed)
for train_index, test_index in kf.split(x_train):
    x_train_fold, x_val_fold =
        x_train[train_index], x_train[test_index]
    y_train_fold, y_val_fold =
        y_train[train_index], y_train[test_index]

    scaler.fit(x_train_fold)
    x_train_scaled = scaler.transform(x_train_fold)
    x_val_scaled = scaler.transform(x_val_fold)
```

Subsequently, the four models were tested on all the folds scaled with the standard scaler. The idea is to average the F1 values across all the folds, in order to determine the best model to apply to the test set.

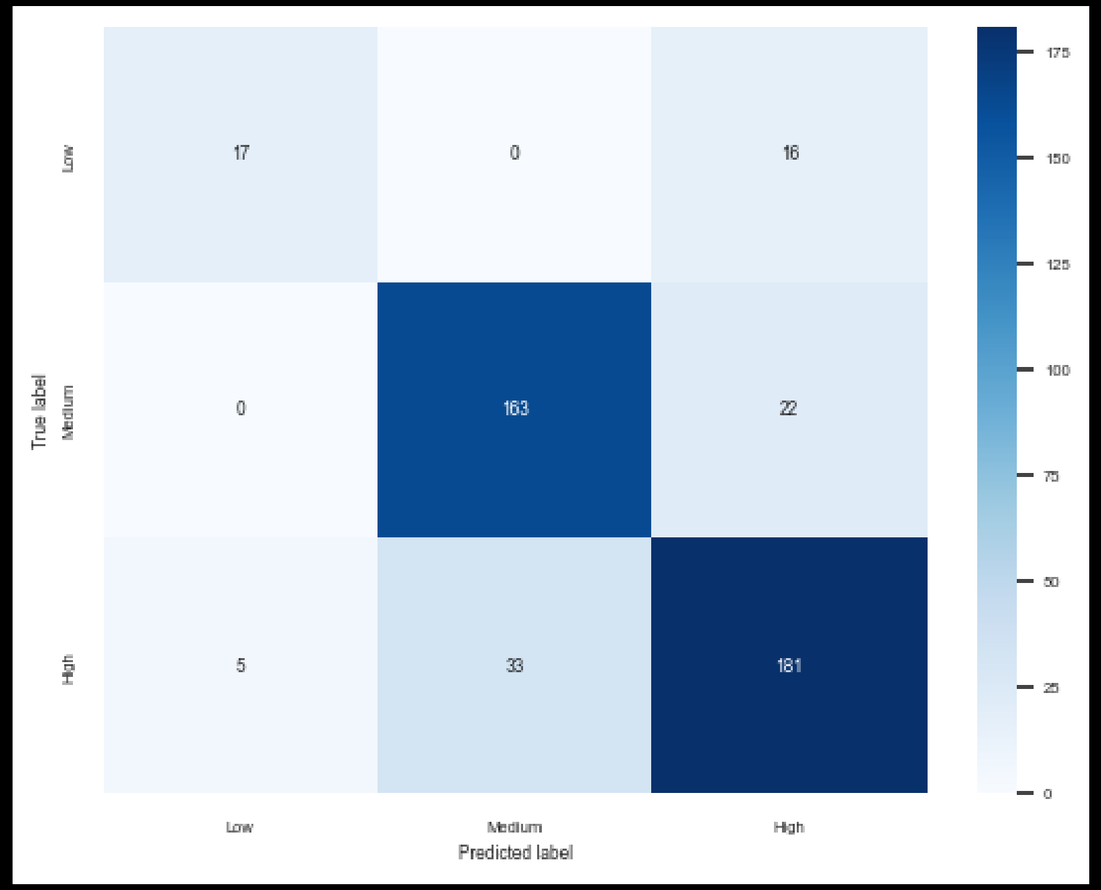
Evaluation of models based k-Fold train-validation-test split

Unlike what we obtained with the simple train-test split presented earlier, this time the Multilayer Perceptron (MLP) neural networks perform better than the Decision Trees, C45 and CART, with the following F1 scores, respectively: 0.87, 0.808, and 0.812. Now we are going to train the MLP model on the test set and proceed similarly to the previous steps. The following results have been obtained:

Classification metrics for MLP model on the test set:

	precision	recall	f1-score	support
High	0.77	0.52	0.62	33
Low	0.83	0.88	0.86	185
Medium	0.83	0.83	0.83	219
accuracy			0.83	437
macro avg	0.81	0.74	0.77	437
weighted avg	0.82	0.83	0.82	437

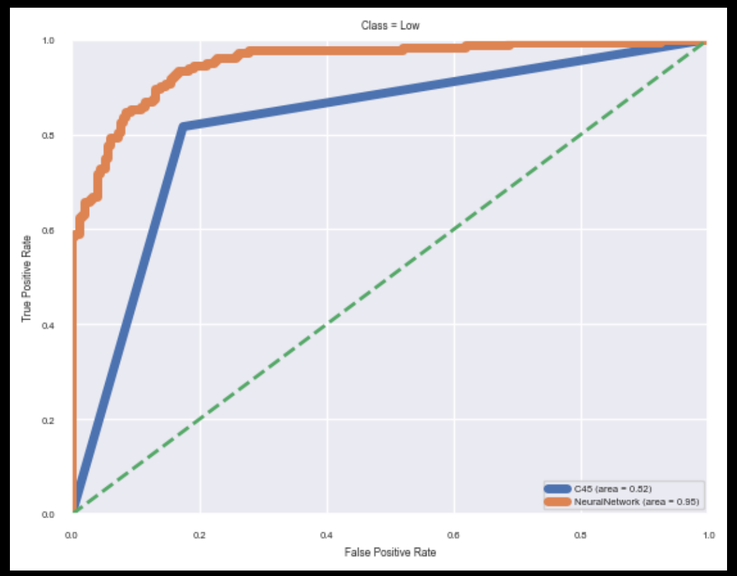
Accuracy: 0.826087

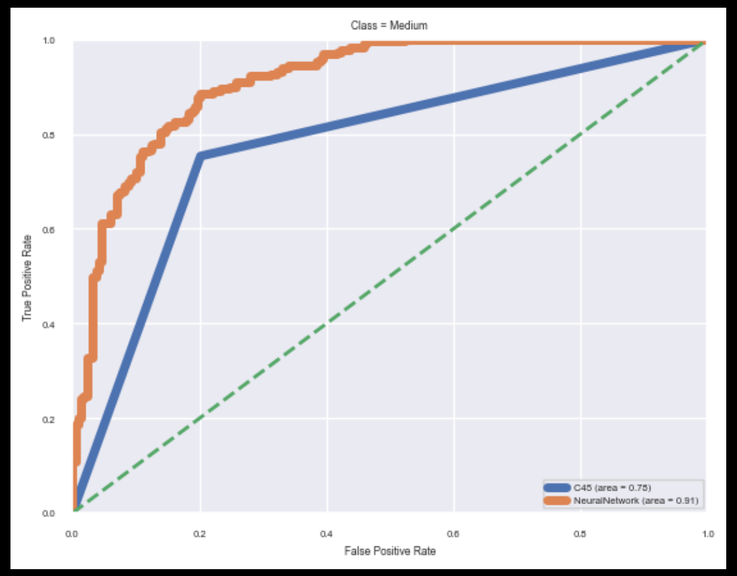


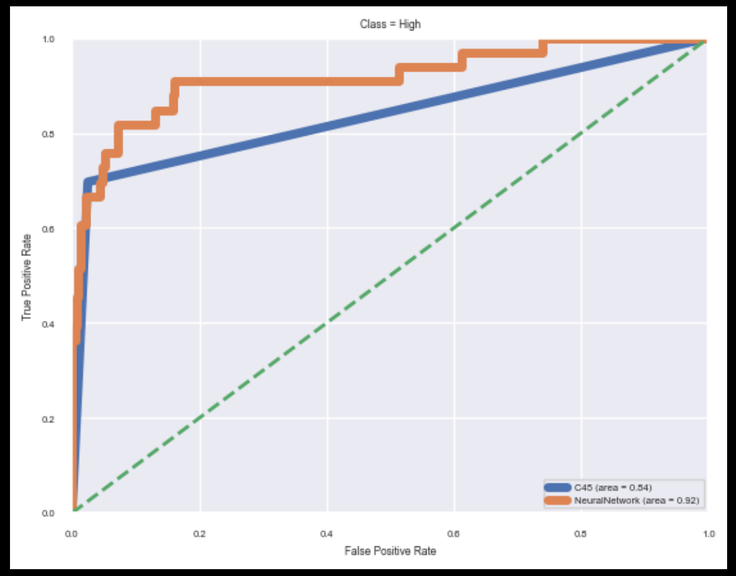
ROC curves

At this point, the next step is to evaluate the ROC curves of the two most extensively studied models so far, namely the Decision Tree and MLP (Multilayer Perceptron).

Evaluation of models based k-Fold train-validation-test split







House Price Competition

Conclusions

It is noteworthy that the K-fold split of the original training set has confirmed that MLP performs better and is more in line with the ideal model: the area under the curves for the three categories averages around 0.9. The result is similar to the previously studied cases: despite simplifying the decision trees (min_sample_split set to 15 instead of 2) and reducing their overfitting, MLP continues to yield the best results without any modification to its hyperparameters. One way to further advance this work could be to fine-tune the hyperparameters of the, albeit simple, neural network used.