Data Mining project for 3rd year CSED department

By:

يوسف محمد رشاد Sec 7

محمد حازم محمد Sec 5

مصطفى إبراهيم عبد الحميد Sec 7

محمد علاء محمد سليمان Sec 6

ندى حسام الدين علي Sec 7

محمد عماد العوض Sec 6

نيره أشرف حمدي Sec 7

هدى عطيه أحمد Sec 7

Housing Prices Classification

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

Data Preparation

Loading the dataset

The dataset file 'Ames_Housing_Sales.csv' is read using `pandas.csv_reader`, and stored in a `pandas.DataFrame`

```
df = pd.read_csv('Ames_Housing_Sales.csv', na_values='None')
```

The `DataFrame.head()` function displays the first 5 examples from the dataset.

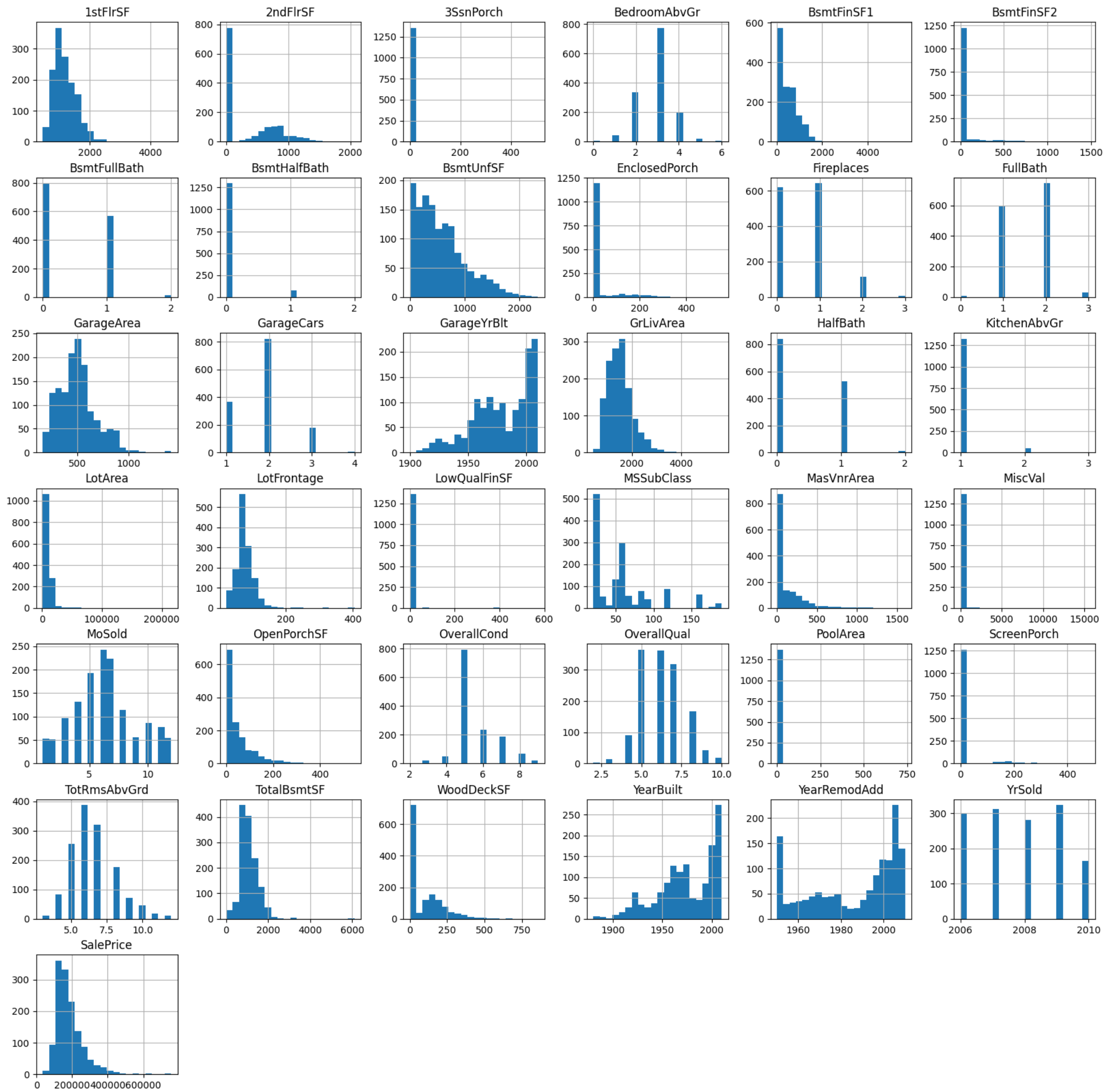
```
# viewing the dataset in table format
df.head()
```

	1stFlrSF	2ndFlrSF	3SsnPorch	Alley	BedroomAbvGr	BldgType	BsmtCond	BsmtExposure	BsmtFi
0	856.0	854.0	0.0	NaN	3	1Fam	TA	No	[illegible]
1	1262.0	0.0	0.0	NaN	3	1Fam	TA	Gd	[illegible]
2	920.0	866.0	0.0	NaN	3	1Fam	TA	Mn	[illegible]
3	961.0	756.0	0.0	NaN	3	1Fam	Gd	No	[illegible]
4	1145.0	1053.0	0.0	NaN	4	1Fam	TA	Av	[illegible]

5 rows × 80 columns

Numerical features will be visualized using histograms with 20 bins using `DataFrame.hist()` and `pyplot.show()` from matplotlib.

```
df.hist(bins = 20, figsize= (20,20))
plt.show()
```



`DataFrame.info()` is used to provide information about each data feature (`column`), namely, the name, the number of non-null entries, and the datatype(`Dtype`).

```
#displaying information for each feature (column)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1379 entries, 0 to 1378
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   1stFlrSF       1379 non-null   float64
 1   2ndFlrSF       1379 non-null   float64
 2   3SsnPorch      1379 non-null   float64
 3   Alley          82 non-null     object 
 4   BedroomAbvGr   1379 non-null   int64  
 5   BldgType       1379 non-null   object 
 6   BsmtCond       953 non-null    object 
 7   BsmtExposure   953 non-null    object 
 8   BsmtFinSF1     1379 non-null   float64
 9   BsmtFinSF2     1379 non-null   float64
 10  BsmtFinType1   953 non-null    object 
 11  BsmtFinType2   952 non-null    object 
 12  BsmtFullBath   1379 non-null   int64  
 13  BsmtHalfBath   1379 non-null   int64  
 14  BsmtQual       953 non-null    object 
 15  BsmtUnfSF      1379 non-null   float64
 16  CentralAir     1379 non-null   object 
 17  Condition1     1379 non-null   object 
 18  Condition2     1379 non-null   object 
 19  Electrical     1379 non-null   object 
 20  EnclosedPorch  1379 non-null   float64
 21  ExterCond      1379 non-null   object 
 22  ExterQual      1379 non-null   object 
 23  Exterior1st    1379 non-null   object 
 24  Exterior2nd    1379 non-null   object 
 25  Fence          265 non-null    object 
 26  FireplaceQu    761 non-null    object 
 27  Fireplaces     1379 non-null   int64  
 28  Foundation     1379 non-null   object 
 29  FullBath       1379 non-null   int64  
 30  Functional     1379 non-null   object 
 31  GarageArea     1379 non-null   float64
 32  GarageCars     1379 non-null   int64  
 33  GarageCond     1379 non-null   object 
 34  GarageFinish   1379 non-null   object 
 35  GarageQual     1379 non-null   object 
 36  GarageType     1379 non-null   object 
 37  GarageYrBlt    1379 non-null   float64
 38  GrLivArea      1379 non-null   float64
 39  HalfBath       1379 non-null   int64  
 40  Heating        1379 non-null   object 
 41  HeatingQC      1379 non-null   object 
 42  HouseStyle     1379 non-null   object 
 43  KitchenAbvGr   1379 non-null   int64  
 44  KitchenQual    1379 non-null   object 
 45  LandContour    1379 non-null   object 
 46  LandSlope      1379 non-null   object 
 47  LotArea        1379 non-null   float64
 48  LotConfig      1379 non-null   object 
 49  LotFrontage    1379 non-null   float64
 50  LotShape       1379 non-null   object 
```

```
 51  LowQualFinSF   1379 non-null   float64
 52  MSSubClass     1379 non-null   int64
 53  MSZoning       1379 non-null   object
 54  MasVnrArea     1379 non-null   float64
 55  MasVnrType     582 non-null    object
 56  MiscFeature    51 non-null     object
 57  MiscVal        1379 non-null   float64
 58  MoSold         1379 non-null   int64
 59  Neighborhood   1379 non-null   object
 60  OpenPorchSF    1379 non-null   float64
 61  OverallCond    1379 non-null   int64
 62  OverallQual    1379 non-null   int64
 63  PavedDrive     1379 non-null   object
 64  PoolArea       1379 non-null   float64
 65  PoolQC         7 non-null      object
 66  RoofMatl       1379 non-null   object
 67  RoofStyle      1379 non-null   object
 68  SaleCondition  1379 non-null   object
 69  SaleType       1379 non-null   object
 70  ScreenPorch    1379 non-null   float64
 71  Street         1379 non-null   object
 72  TotRmsAbvGrd   1379 non-null   int64
 73  TotalBsmtSF    1379 non-null   float64
 74  Utilities      1379 non-null   object
 75  WoodDeckSF     1379 non-null   float64
 76  YearBuilt      1379 non-null   int64
 77  YearRemodAdd   1379 non-null   int64
 78  YrSold         1379 non-null   int64
 79  SalePrice      1379 non-null   float64
dtypes: float64(21), int64(16), object(43)
memory usage: 862.0+ KB
```

Data Cleaning

Check Null Values

the `DataFrame.isna()` function of the DataFrame is used to return the subset of the data containing `null` values and storing the `sum()` into `na_vals`. The subset of columns containing null values is displayed along with the number of null values for each column.

```python
# Check missing values
na_vals = df.isna().sum()

#filter out the columns that have 0 null values
na_cols = na_vals[na_vals > 0]

print(f" == {len(na_cols)} Columns ==")
print(na_cols)
```

```
 == 11 Columns ==
Alley           1297
BsmtCond         426
BsmtExposure     426
BsmtFinType1     426
BsmtFinType2     427
BsmtQual         426
Fence           1114
FireplaceQu      618
MasVnrType       797
MiscFeature     1328
PoolQC          1372
dtype: int64
```

As can be seen, the number of missing values for each column is very high. This indicates that the features entailed are not reliable for data analysis. The columns will be dropped to avoid losing the bulk of data from the dataset. The `DataFrame.dropna()` function is utilized along `axis = 1`, indicating the `columns` instead of rows.

```
df.dropna(axis=1, inplace=True)
```

Check Duplicates

The dataset is checked for duplicated values using `DataFrame.duplicated()`. `sum()` yields the total number of duplicates.

```
df.duplicated().sum()
```

```
0
```

No duplicates found. No further actions are needed.

Get numerical & categorical features

The dataset is analyzed against the presence of categorical features. This is in order to later encode them to be ready for classification. `DataFrame.select_dtypes` returns the subset of the data of a specific datatype or superset of datatypes. A `pandas.Index` is created from the subtraction of the set of numerical features from the total set of features.

```
#return columns of `np.number` datatype
num_cols = df.select_dtypes(np.number).columns

#substract numerical columns from the total set of columns to get categorical features
cat_cols = pd.Index(list(set(df.columns) - set(num_cols)))
```

The `cat_cols` index is used to index `df` to obtain the number of unique values for each feature. This is accomplished using `DataFrame.nunique()`

```
# Check unique values for each feature
df[cat_cols].nunique()
```

```
HeatingQC         5
Foundation        6
GarageFinish      3
RoofMatl          8
Functional        7
SaleType          9
GarageQual        5
Electrical        5
ExterCond         4
Neighborhood     25
Exterior1st      14
Condition1        9
MSZoning          5
Exterior2nd      16
Utilities         2
CentralAir        2
RoofStyle         6
HouseStyle        8
KitchenQual       4
LotConfig         5
Street            2
GarageCond        5
Condition2        8
GarageType        6
SaleCondition     6
Heating           6
BldgType          5
LotShape          4
ExterQual         4
PavedDrive        3
LandContour       4
LandSlope         3
dtype: int64
```

Encode categorical features

The `LabelEncoder` class from scikit-learn is imported to assign a unique integer value for each unique category. The `LabelEncoder.fit_transform()` function firstly computes the unique integers for each feature and then transforms the original dataset `df`.

```
#Import LabelEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

#Iterate over the categorical features of df and transform them
for cat_col in cat_cols:
    df[cat_col] = le.fit_transform(df[cat_col])
```

```
df.head()
```

	1stFlrSF	2ndFlrSF	3SsnPorch	BedroomAbvGr	BldgType	BsmtFinSF1	BsmtFinSF2	BsmtFullBath
0	856.0	854.0	0.0	3	0	706.0	0.0	1
1	1262.0	0.0	0.0	3	0	978.0	0.0	0
2	920.0	866.0	0.0	3	0	486.0	0.0	1
3	961.0	756.0	0.0	3	0	216.0	0.0	1
4	1145.0	1053.0	0.0	4	0	655.0	0.0	1

5 rows × 69 columns

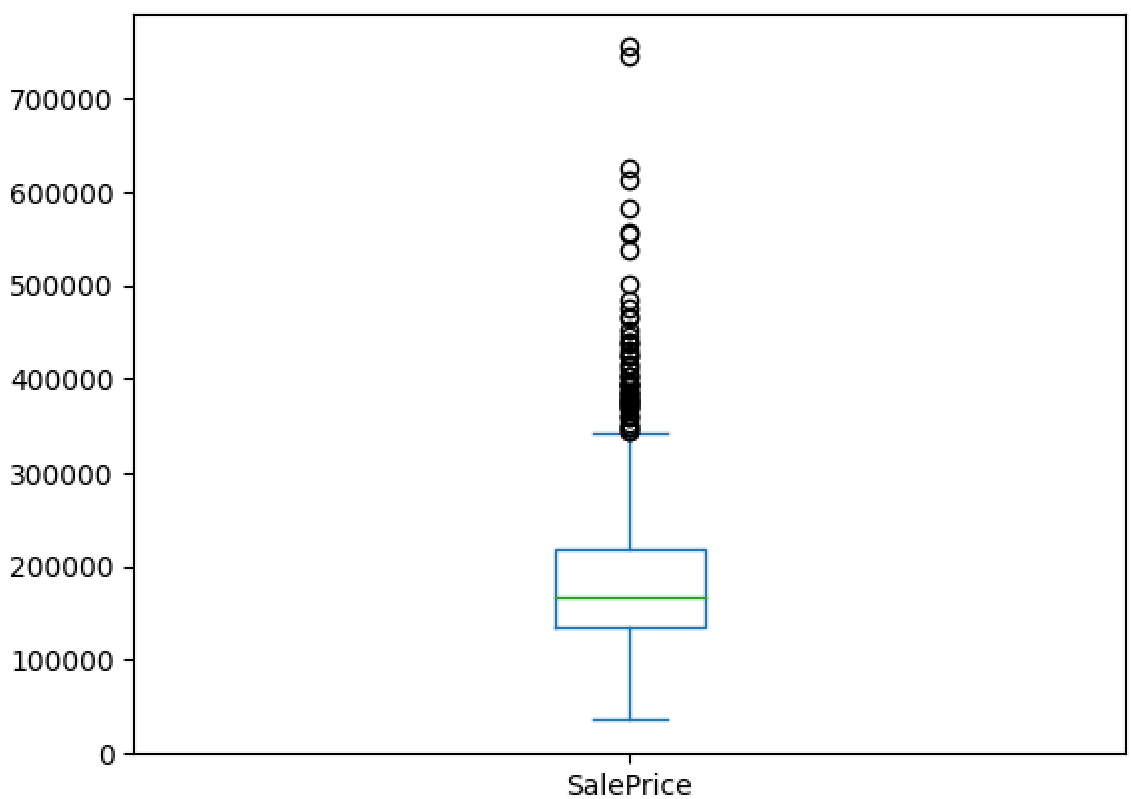
As can be observed, the data is now purely numerical.

Outlier Removal

In this section, the dataset will be analyzed against the presence of outliers, and if detected, will be removed. The target variable "SalePrice" is stored in variable `y`. It is, then, visualized using a Box Plot.

```
y = df["SalePrice"]
y.plot(kind="box")
```

```
<Axes: >
```



As can be observed, a number of outliers is are located outside the third quartile. Below, it is seen that these data points are only 76. They will be removed, for 76 is a small number compared to the size of the dataset.

```
# Check number of sample above certain price
max_price = 325000
print(len(y[y > max_price]))
```

76

`df` is reassigned to the subset of data below the `max_price`. The dataset is then split into features `X` and target `y` i.e. the "SalePrice" feature.

```
# Samples above that number are considered as outliers
df = df[y < max_price]

# Separate data into features and target
x = df.drop("SalePrice", axis=1)
y = df["SalePrice"]
```

Data Scaling

The `StandardScaler` class is imported from scikit-learn to standardize features stored in `x` by removing the mean and scaling to unit variance. `fit_transform` is explained previously.

```
#Import StandardScaler
from sklearn.preprocessing import StandardScaler

#Scale X
x = StandardScaler().fit_transform(x)

pd.DataFrame(x).head()
```

	0	1	2	3	4	5	6	7	8	
0	-0.799945	1.239028	-0.117513	0.187396	-0.411888	0.664251	-0.298908	1.158936	-0.251106	-0
1	0.338556	-0.802365	-0.117513	0.187396	-0.411888	1.301372	-0.298908	-0.806414	3.877845	-0
2	-0.620477	1.267713	-0.117513	0.187396	-0.411888	0.148933	-0.298908	1.158936	-0.251106	-0
3	-0.505505	1.004770	-0.117513	0.187396	-0.411888	-0.483502	-0.298908	1.158936	-0.251106	-0
4	0.010466	1.714716	-0.117513	1.482225	-0.411888	0.544791	-0.298908	1.158936	-0.251106	-0

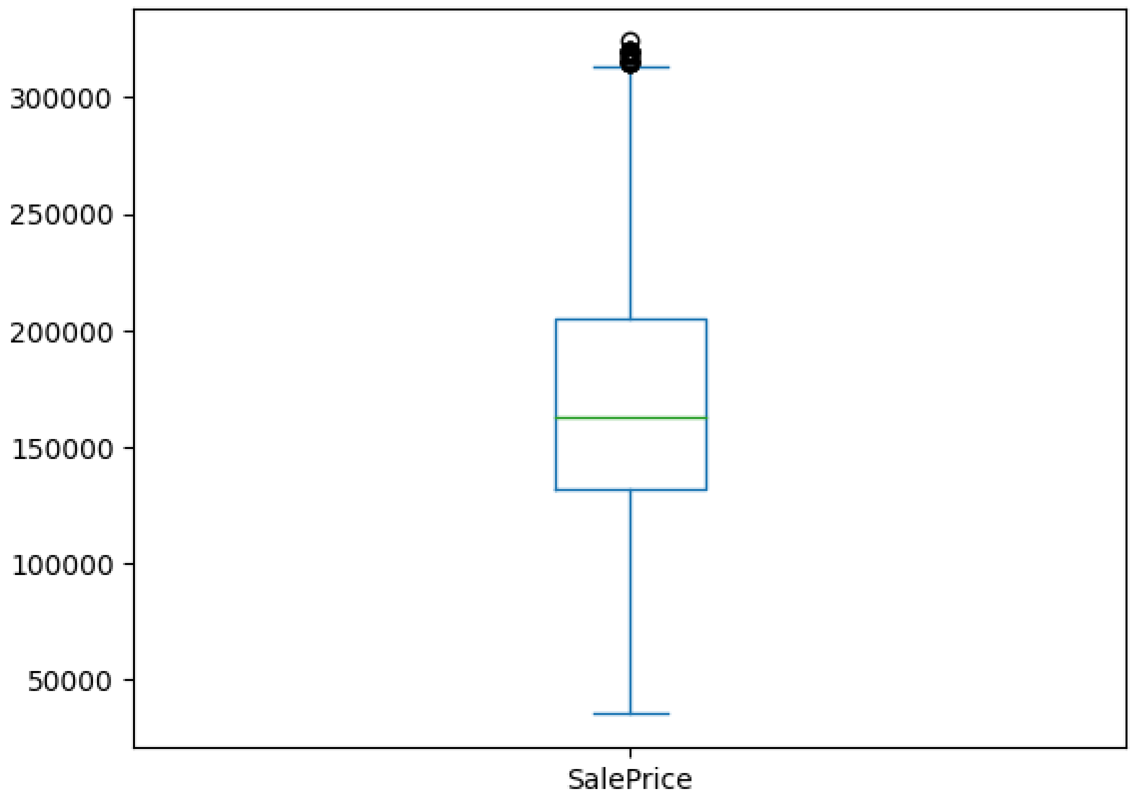
5 rows × 68 columns

Target Grouping

In order to classify housing prices into a specified set of classes, the target variable "SalePrice" needs to be divided into classes. This is accomplished using clustering techniques, as the classes are not defined a priori.

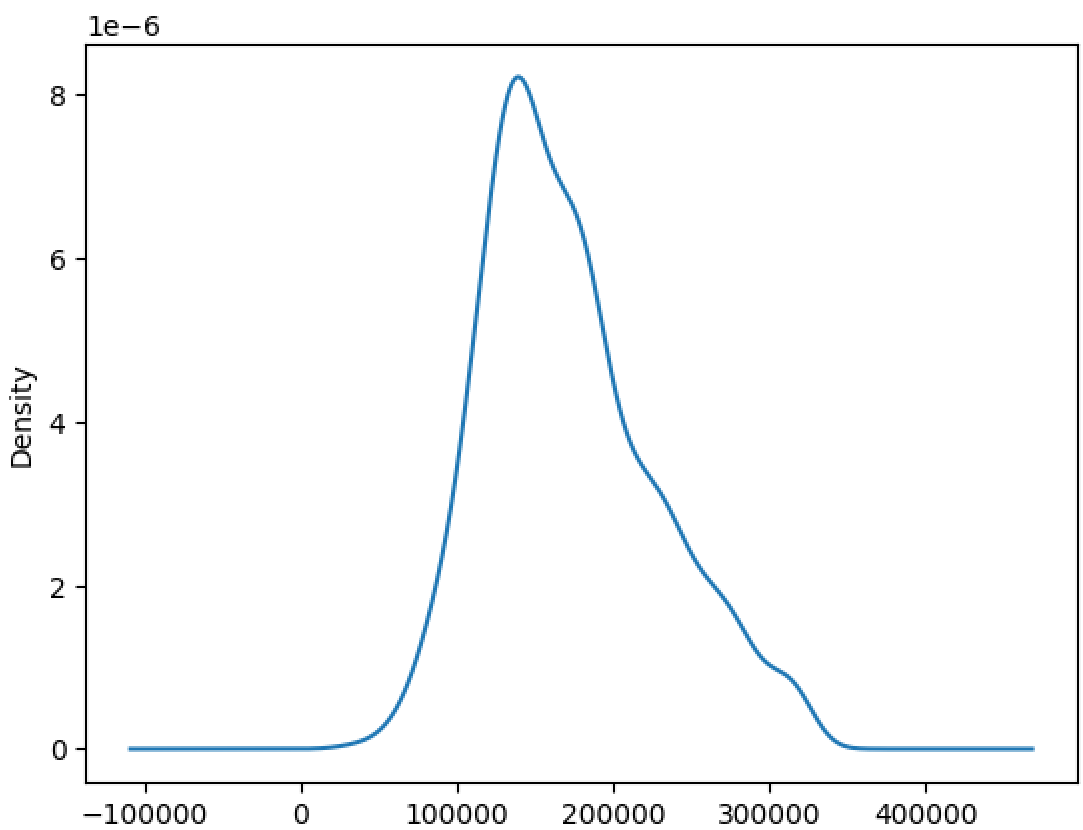
```
# Show the box plot of the target variable `y`
y.plot(kind="box")
```

<Axes: >




```
# Show the KDE plot of the target variable `y`
y.plot(kind = "kde")
```

<Axes: ylabel='Density'>



From the plot above, it can be concluded that the `y` is not skewed, i.e. distributed somewhat equally around the mean.

The target `y` will be clustered into distinct groups using the scikit-learn class KMeans, which is an implementation of the k-means clustering algorithm. The number of clusters `k` is set to 4. `random_state` is used to produce repeatable results. `n_init` is the number of times the k-means algorithm is run with different centroid seeds. it is set to 'auto'. As required by the KMeans class, `y` is reshaped into a 2D array using `numpy.reshape()`.

`KMeans.fit_predict()` returns an array of cluster values, each for each data point. `KMeans.cluster_centers_` stores the center of cluster for each data point

```
from sklearn.cluster import KMeans

#initialize kmeans clustering algorithm
k = 4
kmeans = KMeans(n_clusters=k, random_state=10, n_init='auto')

#predict cluster for each data point, store them in y_clustered
y_clustered = pd.Series(kmeans.fit_predict(np.array(y).reshape(-1, 1)))

#store cluster center for each data point
y_centers = pd.Series(kmeans.cluster_centers_[i] for i in y_clustered).astype("float64")

#view y_clustered
y_clustered.head()
```

```
0    3
1    3
2    3
3    1
4    0
dtype: int32
```

```
#view the centers and number of data points assigned to each
y_centers.value_counts()
```

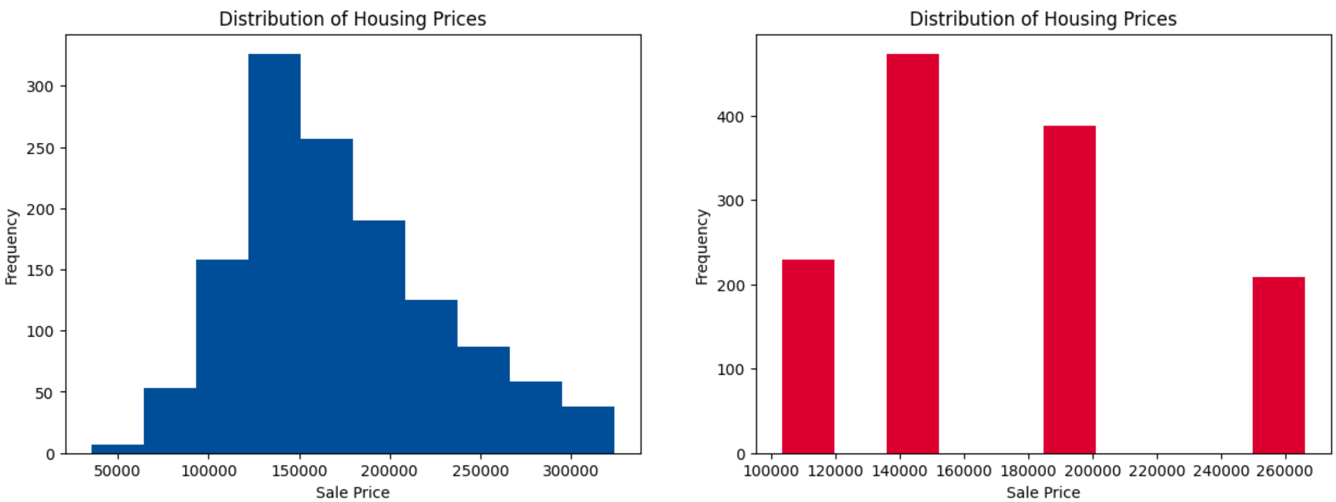
```
144590.733615    473
193875.531579    388
103281.471616    229
266245.267281    209
Name: count, dtype: int64
```

`y` and `y_centers` are plotted using histograms to show the distribution of data before clustering and its distribution in the individual clusters.

```
fig, ax = plt.subplots(figsize=(15, 5), ncols=2)
b = 10

y.plot(color="#004D98", kind="hist", bins=b, ax=ax[0])
ax[0].set_title('Distribution of Housing Prices')
ax[0].set_xlabel('Sale Price')
```

```
ax[0].set_ylabel('Frequency')


y_centers.plot(color="#DB0030", kind="hist", bins=b, ax=ax[1])
ax[1].set_title('Distribution of Housing Prices')
ax[1].set_xlabel('Sale Price')
ax[1].set_ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



Classification

Feature Selection

After the preliminary analysis and exploration of the dataset, it is evident that the dataset contains a lot of unwanted or unhelpful features. These features can worsen the model or cause it to over-fit to the training data. To avoid this, the use of feature selection is needed.

Two feature selection algorithms from scikit-learn will be used:

- `SelectPercentile` which selects the top percentile of the features after applying the scoring function.
- `SelectFromModel` which selects the features that best fit a certain Machine Learning model. Two models are used: `ExtraTreesClassifier` and `RandomForestClassifier`, both of which are classification algorithms that are based on Decision Trees.

The following statement gets a `pandas.Index` of feature names, excluding the target variable "SalePrice"

```
# Get Index of feature names
features = df.drop("SalePrice", axis=1).columns
features
```

```
Index(['1stFlrSF', '2ndFlrSF', '3SsnPorch', 'BedroomAbvGr', 'BldgType',
       'BsmtFinSF1', 'BsmtFinSF2', 'BsmtFullBath', 'BsmtHalfBath', 'BsmtUnfSF',
       'CentralAir', 'Condition1', 'Condition2', 'Electrical', 'EnclosedPorch',
       'ExterCond', 'ExterQual', 'Exterior1st', 'Exterior2nd', 'Fireplaces',
       'Foundation', 'FullBath', 'Functional', 'GarageArea', 'GarageCars',
       'GarageCond', 'GarageFinish', 'GarageQual', 'GarageType', 'GarageYrBlt',
       'GrLivArea', 'HalfBath', 'Heating', 'HeatingQC', 'HouseStyle',
       'KitchenAbvGr', 'KitchenQual', 'LandContour', 'LandSlope', 'LotArea',
       'LotConfig', 'LotFrontage', 'LotShape', 'LowQualFinSF', 'MSSubClass',
       'MSZoning', 'MasVnrArea', 'MiscVal', 'MoSold', 'Neighborhood',
       'OpenPorchSF', 'OverallCond', 'OverallQual', 'PavedDrive', 'PoolArea',
       'RoofMatl', 'RoofStyle', 'SaleCondition', 'SaleType', 'ScreenPorch',
       'Street', 'TotRmsAbvGrd', 'TotalBsmtSF', 'Utilities', 'WoodDeckSF',
       'YearBuilt', 'YearRemodAdd', 'YrSold'],
      dtype='object')
```

Feature Selection by percentile

In the following cell, features will be selected based on a percentile of features ranked by their scores. This will utilize the `SelectPercentile` class. Scores will be determined using the `f_classif` function which implements the ANOVA(Analysis Of Variance) F-value method.

The `f_sel` object of `SelectPercentile` returns `x_fcif` which is the subset of columns it has chosen via its `fit_transform()` function. The features and target are inputs to the function. A percentile of 50% of the features will be chosen.

```python
# Import feature selection algorithms
from sklearn.feature_selection import SelectFromModel, SelectPercentile
# Import scoring function
from sklearn.feature_selection import f_classif

# f_classification
f_sel = SelectPercentile(score_func=f_classif, percentile=50)
x_fcif = f_sel.fit_transform(x, y)
x_fcif.shape
```

```
(1299, 34)
```

Feature Selection for an Extra Trees model

In this algorithm, features will be chosen for an Extra Trees classification model. The model is firstly imported, instantiated as `etc`, initialized with 50 estimators(trees), and, then, fit to the data.

The selector is instantiated as `etc_sel`, initialized with the model, and it is specified that the model has been fit before feature selection using `prefit = True`. The model is pre-fit to improve accuracy of feature selection.

`x_etc` is, similarly, the subset of chosen columns, however, it is obtained from the `transform()` function instead of `fit_transform()`, because the model was pre-fit to the data.

```python
# Import ExtraTrees classifier
from sklearn.ensemble import ExtraTreesClassifier
```

```python
# Initialize model with 50 estimators and fit to data
etc = ExtraTreesClassifier(n_estimators=50).fit(x, y)

# Select from model
etc_sel = SelectFromModel(etc, prefit=True)
x_etc = etc_sel.transform(x)
x_etc.shape
```

```
(1299, 30)
```

Feature Selection for a Random Forest model

This cell is similar to the previous one, only different in the type of model, which, in this case, is the `RandomForestClassifier`.

```python
# Import RandomForest classifier
from sklearn.ensemble import RandomForestClassifier

# Initialize model with 50 estimators and fit to data
rfc = RandomForestClassifier(n_estimators=50).fit(x, y)

# Select from model
rfc_sel = SelectFromModel(rfc, prefit=True)
x_rfc = rfc_sel.transform(x)
x_rfc.shape
```

```
(1299, 25)
```

Analyzing Selections

Selecting features that are common to multiple feature selection methods is a useful approach to identify the most informative features for a machine learning model. This approach can reduce the risk of selecting features that are only informative for a specific method or selecting redundant features. By selecting the features that are agreed upon by multiple methods, we can potentially improve the generalization performance of our model and make it more robust to variations in the data.

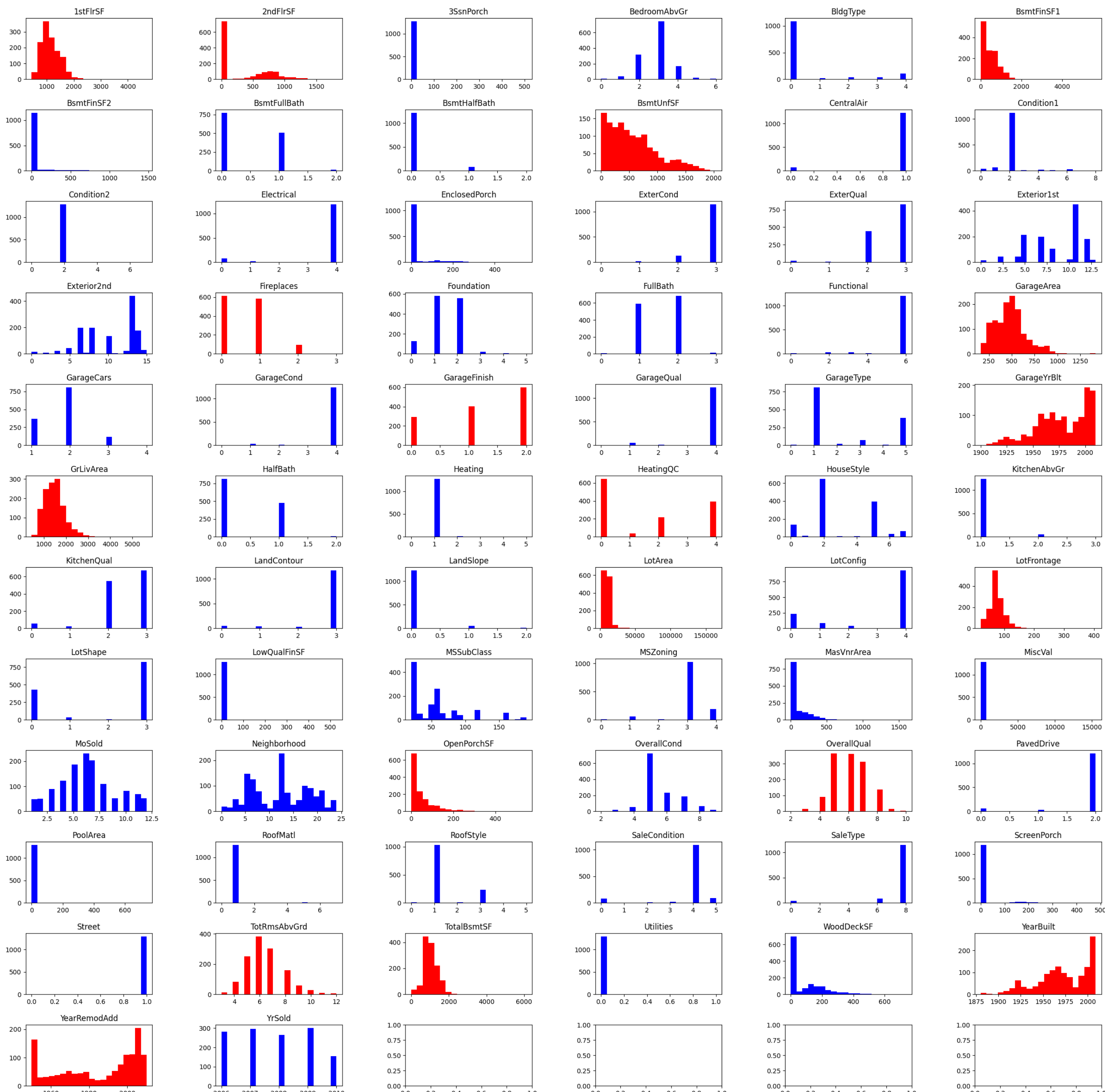
A boolean mask, i.e. an array to represent the selected features with a `True` boolean value and the deselcted with `False`, is obtained from each selection via the `get_support` method. The mask are logicalled anded to produce a mask of the agreed upon features.

The feature dataset `x` will be indexed with the produced mask, which will yield the selected data.

```python
mask1 = f_sel.get_support()
mask2 = etc_sel.get_support()
mask3 = rfc_sel.get_support()

mask  = np.logical_and(mask1, mask2, mask3)

x_selected = x[:,mask]
```

Using `pyplot.subplots`, a histogram of every features will be plotted, with the selected features highlighted in red. A `for` loop will iterate over the columns of the dataset, and according to the value of the mask at each feature's iteration, the color of the histogram will be chosen, blue if `mask[j] == False` i.e. deselcted, and red if `mask[j] == True` i.e. selected.

```
fig, axs = plt.subplots(nrows=12, ncols=6, figsize=(30, 30))
fig.subplots_adjust(hspace=0.5, wspace=0.5)
j = 0
for i, ax in zip(df.drop("SalePrice", axis=1).columns, axs.flat):
    if (mask[j] == True):
        ax.hist(df[i], bins=20, color='red')
    else:
        ax.hist(df[i], bins=20, color='blue')
    ax.set_title(i)
    j += 1
plt.show()
```



train_evaluate() function

This function trains and evaluates a machine learning model on a given dataset. The function takes the following arguments:

- `model` : The machine learning model to train and evaluate.
- `x` : The feature data.
- `y` : The target data.
- `grid_params` : A dictionary of hyperparameters to tune using grid search. If None, grid search will not be used.
- `cv` : The number of folds to use for cross-validation.

The function first splits the data into train and test sets. If grid search is being used, the function then performs grid search to find the best hyperparameters for the model. The function then trains the model on the train set and evaluates the model on the test set. The function prints the classification report and confusion matrix for the model.

The `train_evaluate()` function can be used to train and evaluate any machine learning model. It is a useful function for comparing different models and for finding the best hyperparameters for a given model.

```
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split, GridSearchCV

def train_evaluate(model, x, y, grid_params=None, cv=5):
    # split data into train & test sets
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=10)

    # check for applying grid search
    if grid_params is None:
        model.fit(x_train, y_train)
    else:
        model_gs = GridSearchCV(model, grid_params, scoring='balanced_accuracy',
                                cv=cv, return_train_score=True)
        model_gs.fit(x_train, y_train)
        print(model_gs.best_params_)
        model = model_gs.best_estimator_

    # predict test results
    y_pred = model.predict(x_test)

    # show results
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("Confusion_Matrix:")
    ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred)).plot()

    return model
```

Classification Algorithms

Using the `train_evaluate()` function, several different classification algorithms will be trained, tested, and evaluated.

KNN

The first algorithm to be tested is the K-Nearest Neighbors algorithm. It is implemented in scikit-learn by the `KKNeighborsClassifier` class.

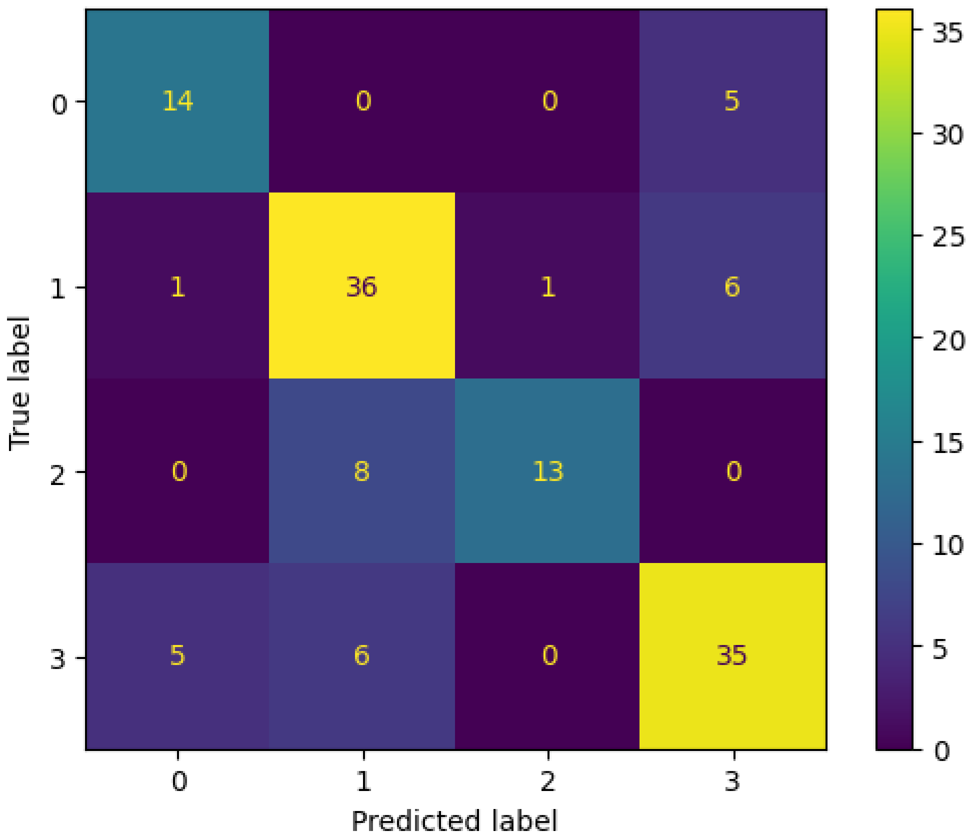
The hyperparametres to be tested for are the number of neighbors to calculate the distance to `n_neighbors`, and the type of `weights` to be assigned to each neighbor.

- `uniform` : uniform weights. All points in each neighborhood are weighted equally.
- `distance` : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

The hyperparametres are stored in a dictionary `grid_params` and passed to the `train_evaluate()` function along with the model `KNeighborsClassifier()`, and features `x_selected` and target data `y_clustered`. The function will return the best model, and display the confusion matrix and evaluation metrics.

```
# Import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier

# Choose the grid search parametres
grid_params = {"n_neighbors": [5, 7, 9], "weights": ["uniform", "distance"]}

#Train and evaluate classifier
knn = train_evaluate(KNeighborsClassifier(), x_selected, y_clustered, grid_params)
```

```
{'n_neighbors': 9, 'weights': 'distance'}
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.74      0.72        19
           1       0.72      0.82      0.77        44
           2       0.93      0.62      0.74        21
           3       0.76      0.76      0.76        46

    accuracy                           0.75       130
   macro avg       0.78      0.73      0.75       130
weighted avg       0.77      0.75      0.75       130

Confusion_Matrix:
```



As can be observed from the confusion matrix and the evaluation metrics, the model has an accuracy of 75% and an average f1-score over classes of 75%.

Naive Bayes

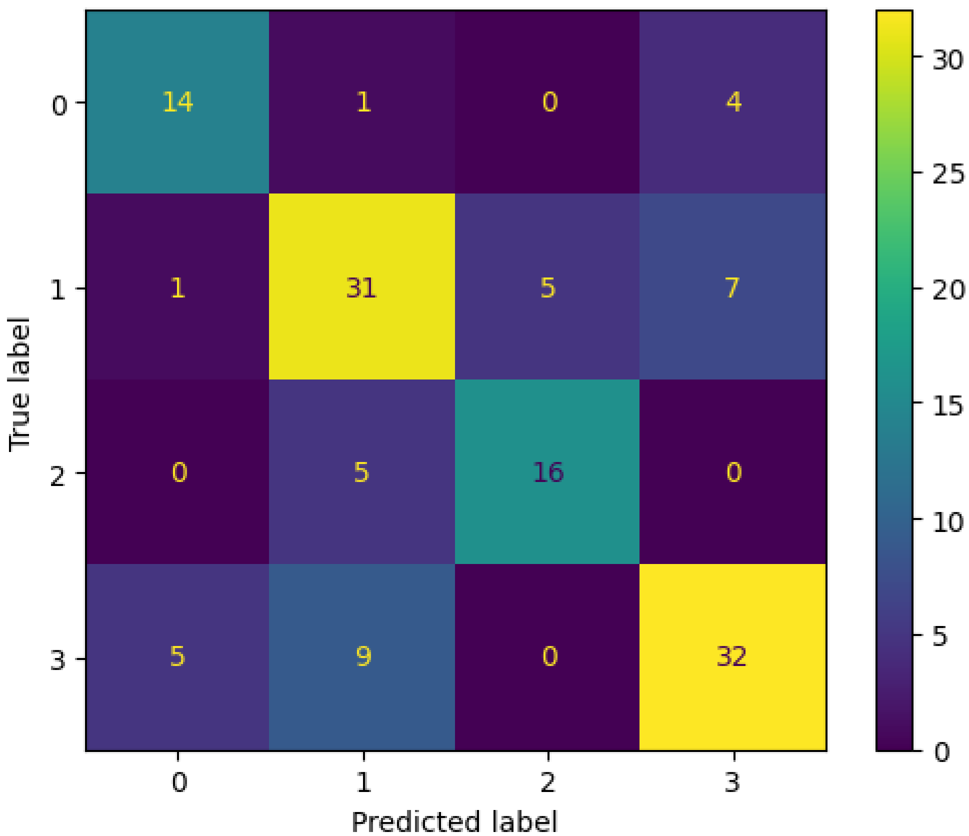
The second algorithm is the Gaussian Naïve Bayes classifier. It is implemented in scikit-learn by the `GaussianNB` class. The Gaussian Naïve Bayes classifier assumes the data to be distributed continuously over a normal(Guassian) distribution.

There are no hyperparametres considred for this model, so `grid_params` will be `None`.

The model is trained and evaluated similarly to the previous model.

```
# Import GaussianNB
from sklearn.naive_bayes import GaussianNB

# Train and evaluate classifier
gnb = train_evaluate(GaussianNB(), x_selected, y_clustered)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.74      0.72        19
           1       0.67      0.70      0.69        44
           2       0.76      0.76      0.76        21
           3       0.74      0.70      0.72        46

    accuracy                           0.72       130
   macro avg       0.72      0.72      0.72       130
weighted avg       0.72      0.72      0.72       130

Confusion_Matrix:
```



The model has an accuracy of 72% and an average f1-score over classes of, also, 72%.

It can be observed that the KNN classifier performs better on the test set than the GNB classifier.

Combining Classifiers

The two models can, however, be combined in a `StackingClassifier`.

```
from sklearn.ensemble import StackingClassifier

stacked = StackingClassifier(estimators=[('knn', knn), ('gnb', gnb)], final_estimator=knn)

train_evaluate(stacked, x_selected, y_clustered)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.68      0.68        19
           1       0.65      0.70      0.67        44
           2       0.88      0.67      0.76        21
           3       0.68      0.70      0.69        46

    accuracy                           0.69       130
   macro avg       0.72      0.69      0.70       130
weighted avg       0.70      0.69      0.69       130
```

Confusion_Matrix: