Project 1: Ames Housing

Elvessa Tatum
UHID: 2064084

Classification

Our Purpose

- Our goal here is to make a model that predicts if a house is expensive or not, based on the median value. This is important as people should know the kind of features that lead a house to be expensive or not. That way, they can see if a house they wish to buy has a good price based on the features it has. Or, they can put a house for sale at a good price based on these features.

Background info

- In this project, we're using Logistic Regression. This is different from linear regression in that it's used for classification problems. These classification problems help predict a probability or class label. We'll also be using decision trees. Decision trees can help with determining the decision process that someone takes when deciding to put a home for sale or buy a home themselves.

- We should also define the difference between predictors and response variables. Predictors are input variables used to make predictions. They can be thought of as independent variables, or features of the dataset. Response variables are the output variable that we're trying to predict. They can be thought of as a target or dependent variable

- We preprocessed the raw data to make it cleaner for analysis. It's necessary to make sure the data is clean to avoid errors later, get rid of missing values, splitting the data for training and testing, etc..

- Overfitting is when a model learns the data too well. It typically leads to a high testing error, due to the results creating poor generalizations of the data. On the other hand, underfitting is when the model doesn't learn the data enough. It leads to the model having a high training and testing error, thus producing bad results.

Data Information

This dataset contains various features and attributes of residential homes in Ames, Iowa, USA. It holds a multitude of variables, but after processing, we decided to use the following:

'Overall Qual': Rates the overall material and finish of the house (Ordinal)

'Full Bath': Full bathrooms above grade (Discrete)

'Year Built': The year the house was built (Discrete)

'Garage Cars': Size of garage in car capacity (Discrete)

'Gr Liv Area': Above grade (ground) living area square feet (Continuous)

'Foundation': Type of foundation (nominal)

'Exter Qual': Evaluates the quality of the material on the exterior (Ordinal)

'Year Remod/Add': Remodel date (same as construction date if no remodeling or additions) (Discrete)

'Garage Yr Blt': Year the garage was built (Discrete)

'Garage Area': Size of garage in square feet (Continuous)

'Kitchen Qual': Kitchen Quality (Ordinal)

'Bsmt Qual': Evaluates the height of the basement (Ordinal)

'Total Bsmt SF': Total square feet of basement area (Continuous)

'Fireplaces': Number of fireplaces (Discrete)

'1st Flr SF': First Floor Square Feet (Continuous)

'Garage Type': Garage location (Nominal)

'BsmtFin Type 1': Rating of basement finished area (Ordinal)

'Exterior 1st': Exterior covering on house (Nominal)

'TotRms AbvGrd': Total rooms above grade (does not include bathrooms) (Discrete)

Data Info Continued

- The continuous data can be summed up as: Overall Qual, Full Bath, Year Built, Garage Cars, Gr Liv Area, Year Remod/Add, Garage Yr Blt, Garage Area, Total Bsmt SF, Fireplaces, 1st Flr SF, and TotRms AbvGrd.

- The categorical variables can be summed up as: Foundation, Exter Qual, Kitchen Qual, Bsmt Qual, Garage Type, BsmtFin Type 1, and Exterior 1st.

- Our response variable here is "AboveMedian," which indicates if a house is above or below the median value.

Data Exploration

- Let's look at the first 5 rows, and check for missing values

```
[4]: housing_data.head()
```

	Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	...	Pool Area	Pool QC
0	1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	...	0	NaN
1	2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	...	0	NaN
2	3	526351010	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	...	0	NaN
3	4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	...	0	NaN
4	5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	...	0	NaN

5 rows × 82 columns

Fence	Misc Feature	Misc Val	Mo Sold	Yr Sold	Sale Type	Sale Condition	SalePrice
NaN	NaN	0	5	2010	WD	Normal	215000
MnPrv	NaN	0	6	2010	WD	Normal	105000
NaN	Gar2	12500	6	2010	WD	Normal	172000
NaN	NaN	0	4	2010	WD	Normal	244000
MnPrv	NaN	0	3	2010	WD	Normal	189900

Data exploration continued

[5]:
```python
# checking missing values
missing = housing_data.isnull().sum()
missing = missing[missing > 0].sort_values(ascending=False)
print(missing)
```

```
Pool QC           2917
Misc Feature      2824
Alley             2732
Fence             2358
Mas Vnr Type      1775
Fireplace Qu      1422
Lot Frontage       490
Garage Cond        159
Garage Qual        159
Garage Finish      159
Garage Yr Blt      159
Garage Type        157
Bsmt Exposure       83
BsmtFin Type 2      81
Bsmt Cond           80
Bsmt Qual           80
BsmtFin Type 1      80
Mas Vnr Area        23
Bsmt Half Bath       2
Bsmt Full Bath       2
BsmtFin SF 1         1
Garage Cars          1
Garage Area          1
Total Bsmt SF        1
Bsmt Unf SF          1
BsmtFin SF 2         1
Electrical           1
dtype: int64
```

Preprocessing

- Let's deal with the missing values by filling them in or getting rid of the variables attached to them if that variable has too many missing values.

```
[6]: # drop columns with too many missing values
     cols_to_drop = ['Pool QC', 'Misc Feature', 'Alley', 'Fence', 'Mas Vnr Type', 'Fireplace Qu']
     cols_to_drop = [col for col in cols_to_drop if col in housing_data.columns]
     housing_data = housing_data.drop(columns=cols_to_drop)

     # separate remaining columns with missing values
     cat_cols = housing_data.select_dtypes(include='object').columns
     num_cols = housing_data.select_dtypes(include=['int64', 'float64']).columns

     # fill missing categorical with "Missing"
     for col in housing_data.columns:
         if col in cat_cols and housing_data[col].isnull().sum() > 0:
             housing_data[col] = housing_data[col].fillna("Missing")

     # fill missing numerical with median
     for col in housing_data.columns:
         if col in num_cols and housing_data[col].isnull().sum() > 0:
             housing_data[col] = housing_data[col].fillna(housing_data[col].median())
```

```
[7]: # checking missing values again
     missing = housing_data.isnull().sum()
     missing = missing[missing > 0].sort_values(ascending=False)
     print(missing)

     Series([], dtype: int64)
```

Preprocessing and data exploration continued

- Let’s add the target variable

```
[8]: # adding target variable
housing_data['AboveMedian'] = (housing_data['SalePrice'] > housing_data['SalePrice'].median()).astype(int)
```

- Then, let’s see how the data correlates with the target variable

```
[9]: # taking a look at what numerical values correlate best with the target.
correlation_with_target = housing_data.select_dtypes(include='number').corr()['AboveMedian'].sort_values(ascending=False)
print(correlation_with_target)
```

Preprocessing and data exploration continued

- Clearly, the top 3 variables that allign best with the target are SalePrice, Overall Quality, and Full bath.

- SalePrice was the original target in the dataset, so it should be removed from future use

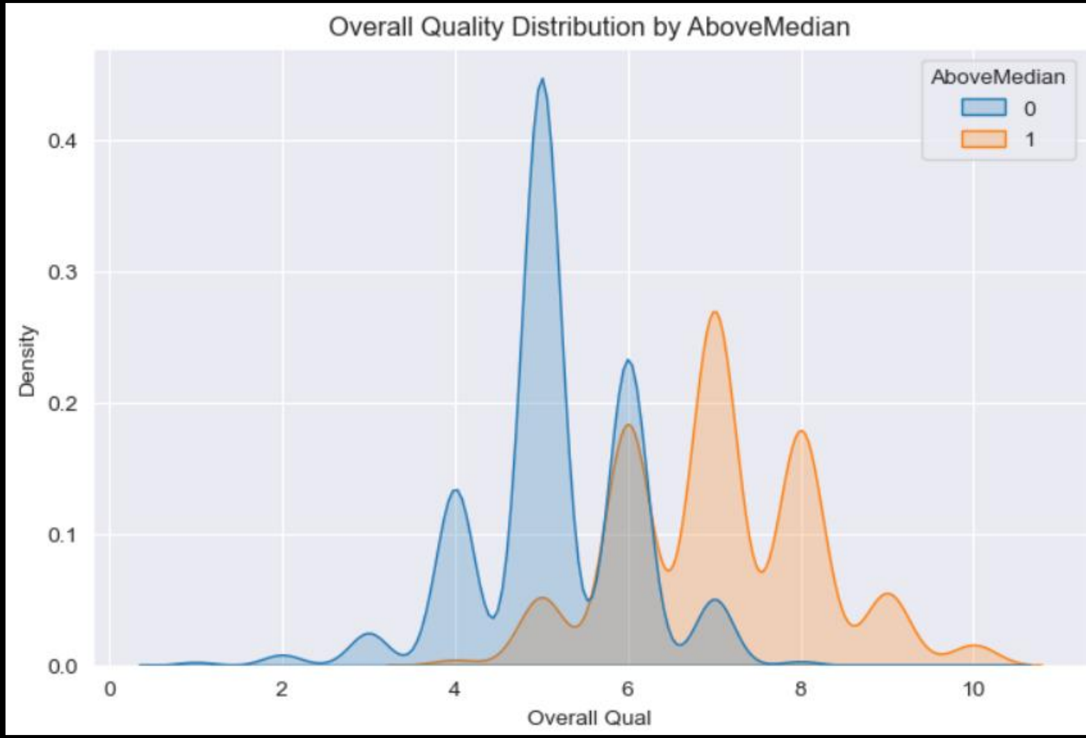
AboveMedian	1.000000
SalePrice	0.702148
Overall Qual	0.674081
Full Bath	0.611337
Year Built	0.591459
Garage Cars	0.579992
Gr Liv Area	0.565414
Year Remod/Add	0.545115
Garage Yr Blt	0.528351
Garage Area	0.510413
Total Bsmt SF	0.448391
Fireplaces	0.437493
1st Flr SF	0.433844
TotRms AbvGrd	0.385368
Mas Vnr Area	0.309760
Half Bath	0.309360
Open Porch SF	0.294787
2nd Flr SF	0.275688
Wood Deck SF	0.271617
Lot Frontage	0.241983
BsmtFin SF 1	0.240897
Bsmt Unf SF	0.202650
Lot Area	0.192664
Bsmt Full Bath	0.153988
Bedroom AbvGr	0.113994
Screen Porch	0.082175
Pool Area	0.036094
3Ssn Porch	0.031803
Mo Sold	0.029409
Misc Val	0.011308
BsmtFin SF 2	-0.007060
Yr Sold	-0.013699
Bsmt Half Bath	-0.023329

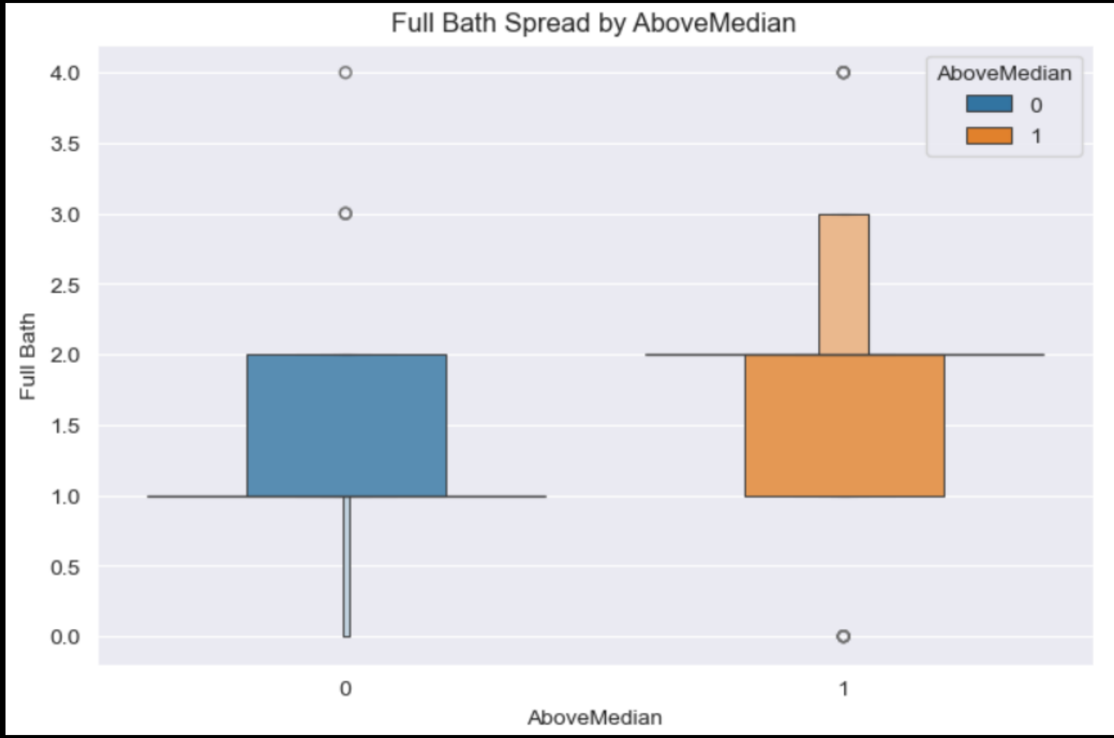
MS SubClass	-0.028785
Order	-0.039034
Low Qual Fin SF	-0.049698
Enclosed Porch	-0.132350
Kitchen AbvGr	-0.143227
Overall Cond	-0.154088
PID	-0.210707

Name: AboveMedian, dtype: float64

Data exploration continued

- Let’s look at some plots based on what correlates with the data. Let’s look at the distribution of overall quality and full bath
- From here, we can see that most houses that are below the median have 2 bathrooms and have a lesser quality. Meanwhile, houses with a higher quality and not many bathrooms are more expensive, being higher than the median.





Data exploration continued

- Let's encode the data to include categorical values in our exploration, and see what are the most correlated features. After this, it is clear what columns should or shouldn't be kept. Anything above a correlation of 0.35 should be kept.

```
[13]: # Encoding to check categorical values allign best with the target

from sklearn.preprocessing import OneHotEncoder

# Separate features and target
X = housing_data.drop(columns=['AboveMedian', 'SalePrice'])
y = housing_data['AboveMedian']

# One-hot encode categorical variables
X_encoded = pd.get_dummies(X, drop_first=True)  # drop_first to avoid multicollinearity

X_encoded['AboveMedian'] = y

# Correlation with target
corr = X_encoded.corr()['AboveMedian'].sort_values(ascending=False)

# Top positively correlated features
print(corr.head(20))
```

```
AboveMedian              1.000000
Overall Qual             0.674081
Full Bath                0.611337
Year Built               0.591459
Garage Cars              0.579992
Gr Liv Area              0.565414
Foundation_PConc         0.559996
Exter Qual_Gd            0.550230
Year Remod/Add           0.545115
Garage Yr Blt            0.528351
Garage Area              0.510413
Kitchen Qual_Gd          0.460314
Bsmt Qual_Gd             0.449152
Total Bsmt SF            0.448391
Fireplaces               0.437493
1st Flr SF               0.433844
Garage Type_Attchd       0.413261
BsmtFin Type 1_GLQ       0.406491
Exterior 1st_VinylSd     0.385927
TotRms AbvGrd            0.385368
Name: AboveMedian, dtype: float64
```

Data preprocessing continued

- Let's get the selected features, encode them, and scale and normalize them

```
[15]: # Let's separate the dataset so we can do classification with meaningful data

selected_features = [
    'Overall Qual', 'Full Bath', 'Year Built', 'Garage Cars', 'Gr Liv Area',
    'Foundation', 'Exter Qual', 'Year Remod/Add', 'Garage Yr Blt',
    'Garage Area', 'Kitchen Qual', 'Bsmt Qual', 'Total Bsmt SF',
    'Fireplaces', '1st Flr SF', 'Garage Type', 'BsmtFin Type 1',
    'Exterior 1st', 'TotRms AbvGrd'
]

categorical_cols = [col for col in selected_features if housing_data[col].dtype == 'object']
numeric_cols = [col for col in selected_features if col not in categorical_cols]

X = housing_data[selected_features]
y = housing_data['AboveMedian']

# one-hot encode categoricals
X_encoded = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

# scale the numerical columns
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_encoded[numeric_cols] = scaler.fit_transform(X_encoded[numeric_cols])
```

Creating the models

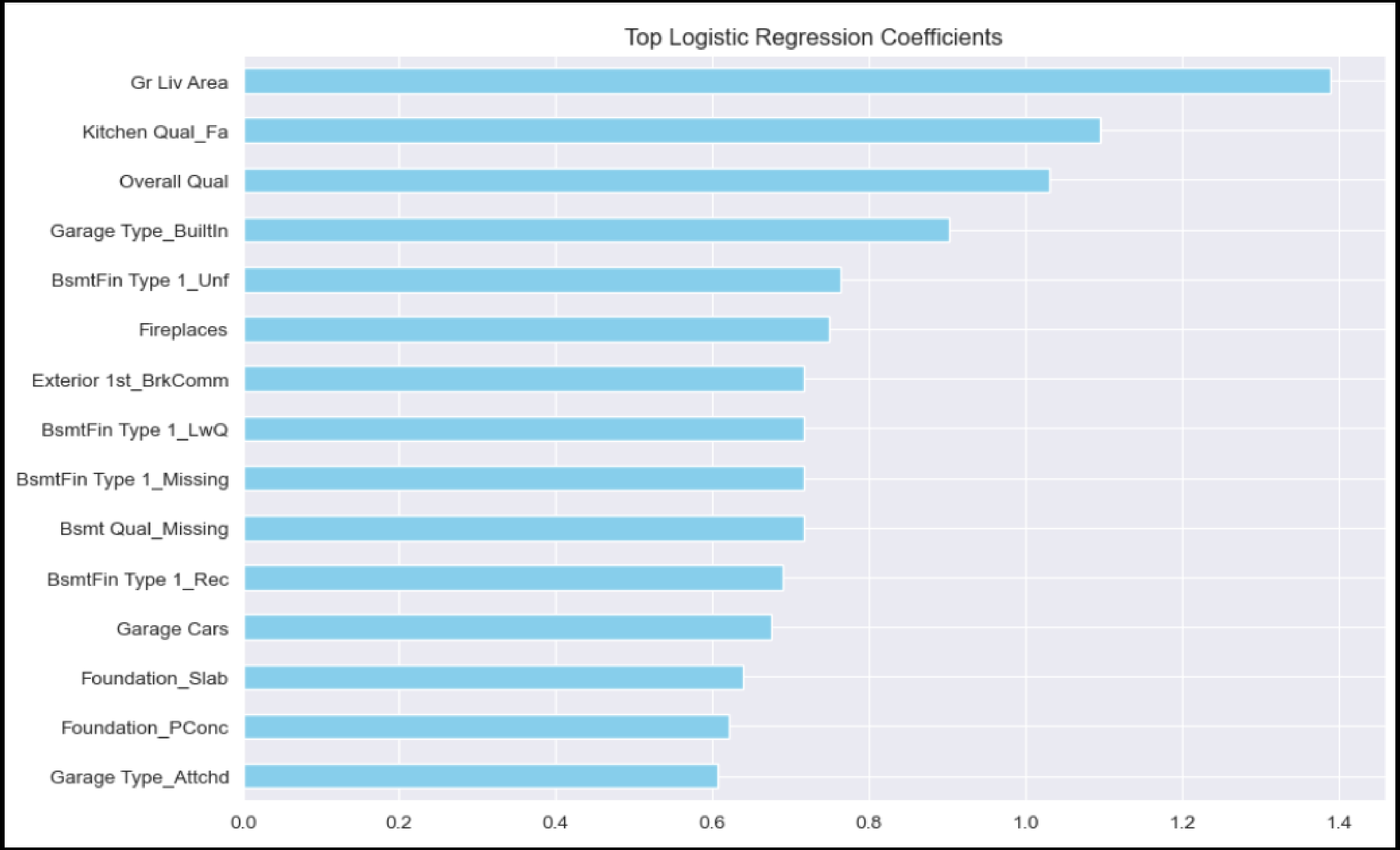
- Let's use logistic regression and decision trees to run our data.

[17]:

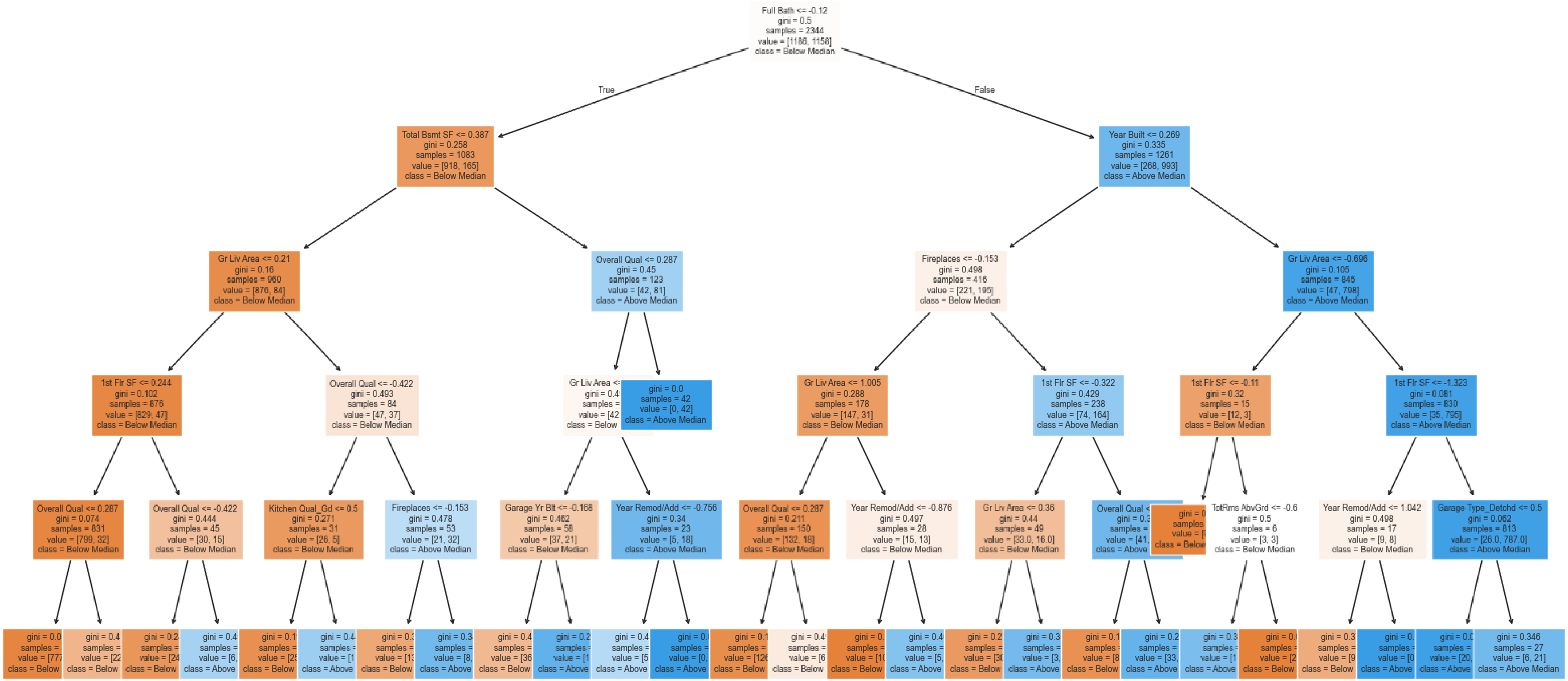
```python
# let's run the classification models Logistic regression and decision trees

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

model_log = LogisticRegression(max_iter=1000)
model_log.fit(X_train, y_train)

y_pred_log = model_log.predict(X_test)
print("Logistic regression classifier")
print(classification_report(y_test, y_pred_log))
print("Accuracy:", accuracy_score(y_test, y_pred_log))
log_cv_scores = cross_val_score(model_log, X_encoded, y, cv=5)
print("CV Accuracy (LogReg):", np.mean(log_cv_scores))


model_tree = DecisionTreeClassifier(max_depth=5)
model_tree.fit(X_train, y_train)
y_pred_tree = model_tree.predict(X_test)
print("Decision tree classifier")
print(classification_report(y_test, y_pred_tree))
print("Accuracy:", accuracy_score(y_test, y_pred_tree))
tree_cv_scores = cross_val_score(model_tree, X_encoded, y, cv=5)
print("CV Accuracy (Tree):", np.mean(tree_cv_scores))
```

Creating the models continued

- Let’s visualize our models, starting with logistic regression, then decision tree.

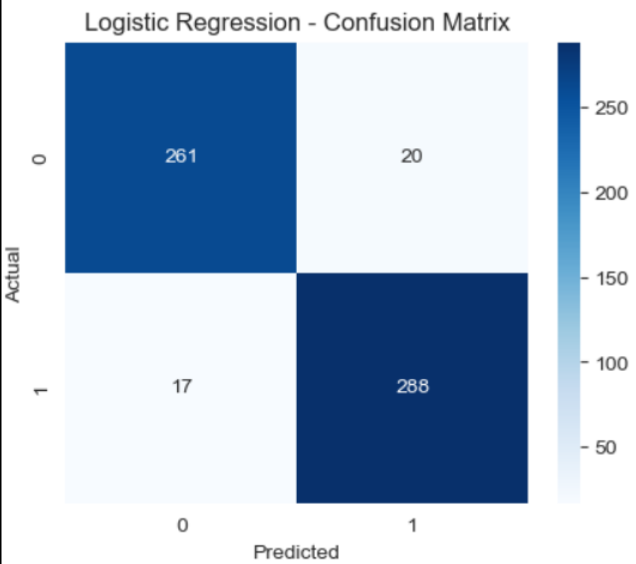


Decision Tree (Depth=5)



Evaluating the models

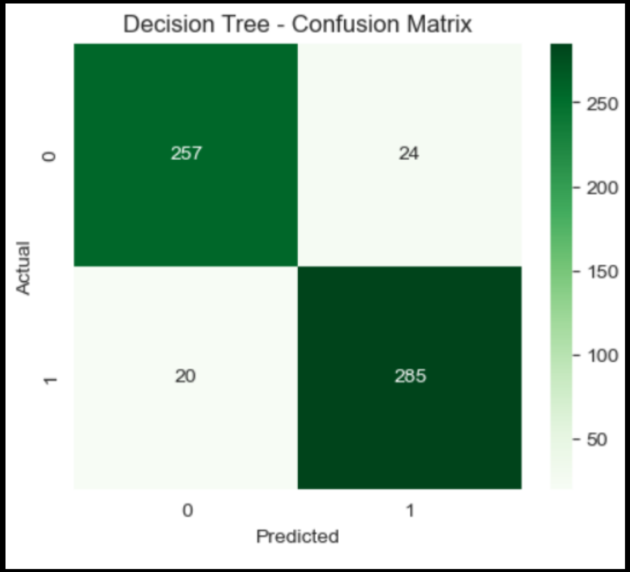
- Let’s look at the classification report and confusion matrix for our models



Logistic regression classifier

	precision	recall	f1-score	support
0	0.94	0.93	0.93	281
1	0.94	0.94	0.94	305
accuracy			0.94	586
macro avg	0.94	0.94	0.94	586
weighted avg	0.94	0.94	0.94	586

Accuracy: 0.9368600682593856
CV Accuracy (LogReg): 0.9098976109215018



Decision tree classifier

	precision	recall	f1-score	support
0	0.92	0.91	0.92	281
1	0.92	0.93	0.93	305
accuracy			0.92	586
macro avg	0.92	0.92	0.92	586
weighted avg	0.92	0.92	0.92	586

Accuracy: 0.9232081911262798
CV Accuracy (Tree): 0.897269624573379

Conclusion

- The logistic regression model shows that the ground living area and the kitchen quality are actually more important than the overall quality of the house. This is important as it differed from what we originally saw. Originally, we saw the overall quality of the house being the most important factor, but we now know that isn't the case. Rather, the kitchen and ground living area parts of the house are the most important

- Meanwhile, the decision tree model showed that the root node was split on full bath, followed by other features like Overall Quality, Year Built, Fireplaces, and ground living area. This builds on what we saw before, which is that the ground living area is one of, if not the most important feature in whether a house is above or below the median value.

- With these results, people are able to understand what exactly leads a house to be above or below median value. With this information, people can determine how much to sell or buy a house for, based on the important features such as the overall quality of the house, the number of bathrooms, and the ground living area.

- Both models did extremely well, with an accuracy of 0.94 for the logistic regression model and an accuracy of 0.92 for the decision tree model. Even with cross validation, both models had an accuracy of about 0.9. This means that the model is accurate even on unseen data.

- I would expand upon this by using different models (maybe random forest instead of decision trees) or changing how much data is used for training vs testing.