Module 4 Assignment – Real Estate Investment

ALY 6020 – Predictive Analytics

Tyler Mitchell

August 3rd, 2025

Pre-Processing

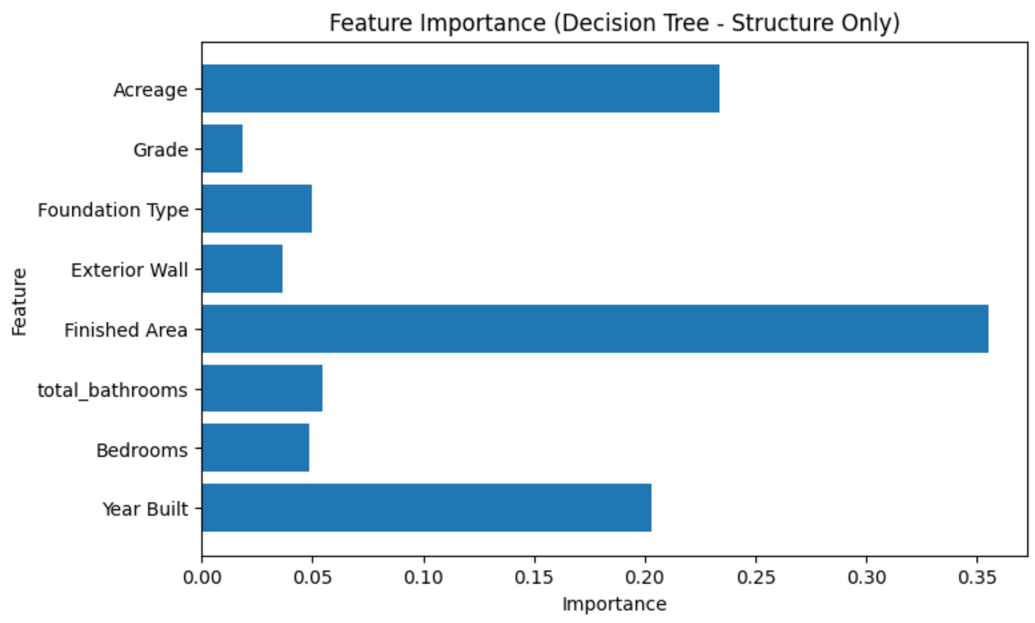
This report outlines a machine learning approach to modeling real estate data in the Nashville area to assist real estate investment firms in locating properties with the strongest value opportunity. The target variable of the report is "Sale Price Compared to Value," which classifies each transaction into either over or under the property's assessed market value. The goal is to predict this classification using a combination of property characteristics, geographic variables, and timing variables.

Prior to developing the model, the dataset underwent pre-processing steps to adhere to appropriate data quality and utility. First, non-useful columns and low information columns such as "Suite/Condo #" were omitted as these either had thousands of missing values or were not useful. Fields such as "Parcel ID" and the full address were also deemed nonpredictive identifiers and excluded from the feature set to reduce noise in the model. The columns of interest were examined for presence of missing values and examined for key missing fields. Rows that were missing the number of bedrooms, bathrooms, finished area, foundation type were eliminated to ensure that the input to our model is consistent when created rows. Full bathrooms were combined with half baths to create the feature, "total_bathrooms", reduce collinearity, and ensure the model's consideration of the interior features that are relevant. Categorical variables, such as "Grade", "Foundation Type", and "Exterior Wall" were label encoded to produce a variable that could be accepted by a tree-based classifier. Lastly, the target variable was encoded into numeric labels to identify a property sale as an over value or under value, rather than its original categorical values. The pre-processing steps were applied uniformly as to maintain consistency across three different models. Other categorical variables that were similarly encoded include: "City", "Neighborhood", and "Sold as Vacant". Columns that were not useful for prediction such as "Parcel ID" and "Legal Reference were determined to not be useful for predictor variables and excluded from certain feature sets. After preprocessing the data, the cleaned dataset was used to create three different classification models using decision tree-based classifiers.

Model 1

The first predictive model was developed using a decision tree classifier that was trained only on structural property characteristics to classify if a home sold above or below its assessed market value. The features included the year built, bedrooms, total bathrooms, finished area, acreage, grade, foundation type, and material of exterior walls. After dropping missing values in each of these fields, the categorical variables were label-encoded, and the data was split into a training-test formulation. The final model showed an overall accuracy of roughly 64%, had solid performance on the dominant class of over-valued properties with an F1-score of 0.76, but had weak performance on selling under-value with precision and recall scores of 0.26. This suggests that there is inherent bias towards the dominant class, and overall limited usefulness in detecting under-value properties. In the evaluation of feature importances, finished area, acreage, and year built were determined to be the most important predictors of the selling price, shedding significant light on commonly understood drivers of property valuation in real estate (1). Overall, the advantage of the model being interpretable and only relying on underlying structural information available at the property, means that while this model would not be regarded as a practical stand-alone decision maker for valuable investment decisions, it would still be useful as a point of reference, or baseline tool.

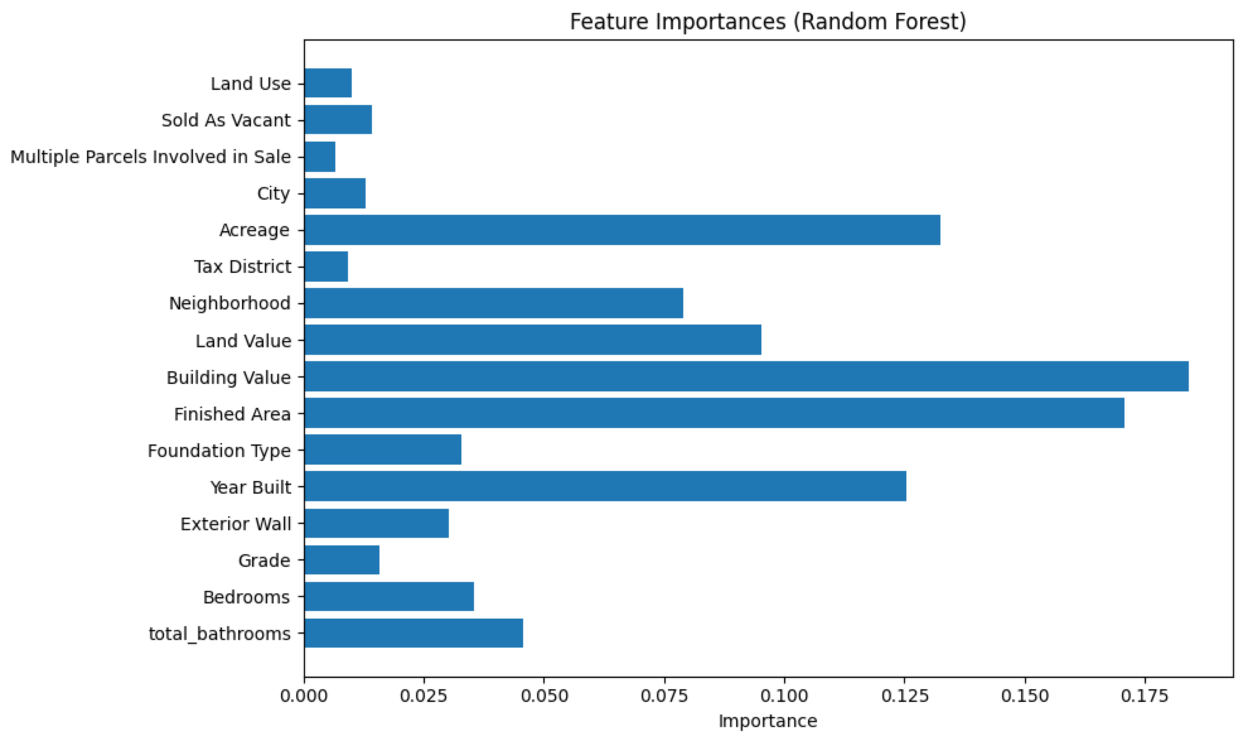
1.



Model 2

The second model was a random forest classifier using the full set of features, including structural, categorical and location variables. The random forest classifier included a complete set of variables to utilize the full predictive power of the features available, including land value, building value, neighborhood, land use, and other categorical features like grade, exterior wall, foundation type, and city. To prepare the dataset, all non-predictive variables like parcel identifiers, sale dates, and legal descriptors were removed. The remaining categorical variables were all label-encoded, and missing data were dropped, so a consistent input space was used. When the random forest classifier was tested with the training and test sets, the classifier had an overall accuracy of roughly 73%, which was an improvement over the simple decision tree generated in Model 1. The model indicated particularly favorable forecasting where over-value sales were concerned, generating an F1-score of 0.83. However, there was particularly poor predictive power for under-value sales, with a recall of just 0.16 and an F1-score of 0.23, meaning there was difficulty with identifying and appropriately flagging under-valued opportunities. An examination on feature importance indicated building value, finished area, acreage, and land value were the most important variables (2). It appears that when trying to classify sale prices, size and valuation of assets plays a considerable role. Overall, while the model produces good forecasting ability, it does rely on an ensemble approach for its prediction, thus limiting its interpretability.

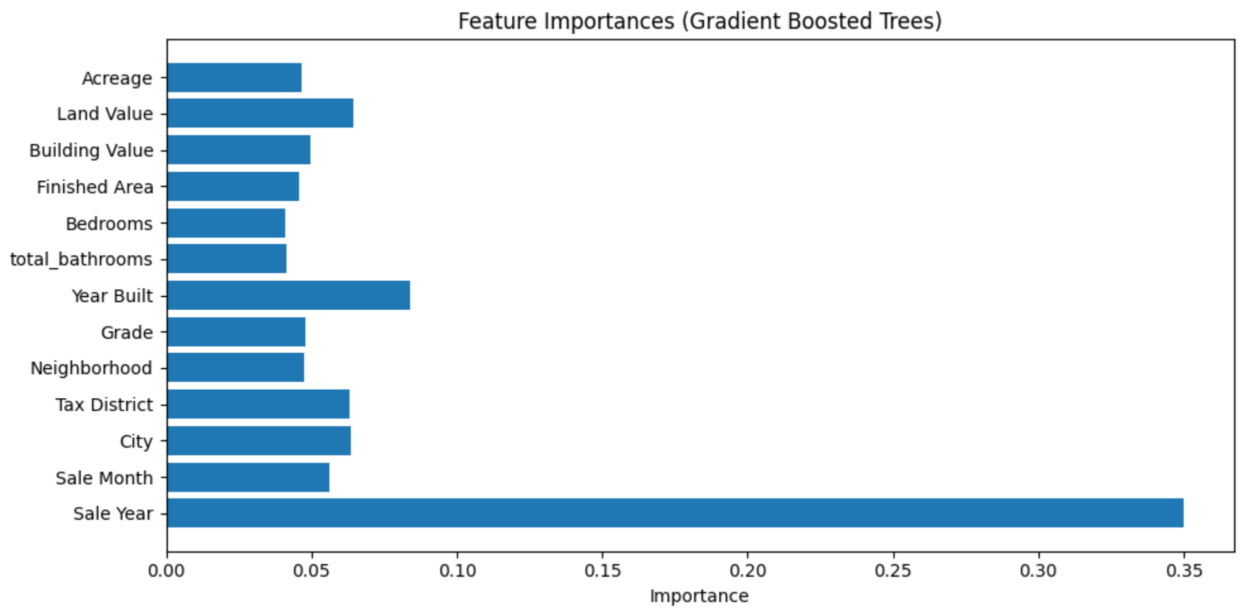
2.



Model 3

The third model implemented a gradient boosted tree classifier using the XGBoost library. The model utilized a selection of curated structural, location, and market-timing features such as acreage, building and land value, finished area, number of bedrooms, total bathrooms, year built, grade, neighborhood, tax district, city, and the month and year of sale. The categorical variables, including the target variable, were all label encoded. The data split was completed using an 80/20 ratio for the training and testing data. After model training, the predictions were evaluated by the standard classification metrics to assess the model performance. The model reached an accuracy of roughly 79%. The precision and recall for the “Over” class were 0.81 and 0.93 respectively, while the recall for the “Under” class was noticeably lower at 0.33 and its precision was 0.62. The resulting F1-scores were 0.87 for “Over” and 0.43 for “Under.” The unbalanced results were troubling, but its weighted average F1-score was 0.761. Its top model features were "Sale Year", "Year built" and location features such as "Neighborhood" and "City" (3).

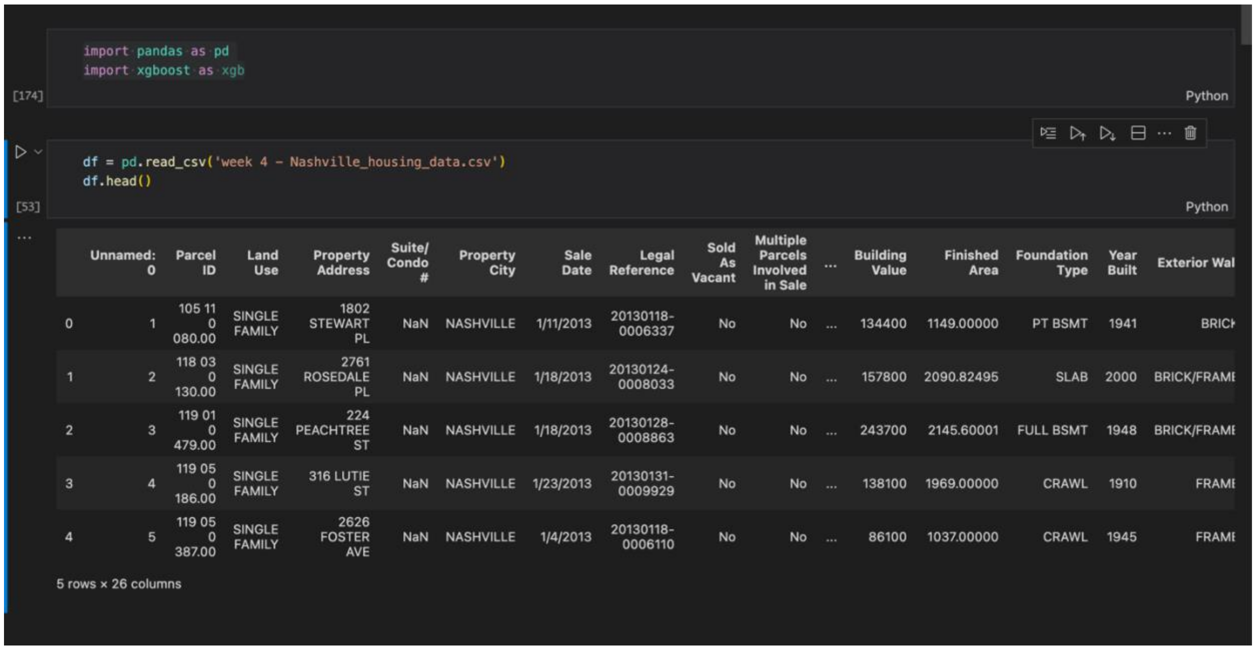
3.



Conclusion/Recommendation

The gradient-boosted tree classifier, or Model 3, was the best option based on the performance metrics and qualitative traits. With a weighted F1-score of 0.76 and the highest overall accuracy of 79%, it demonstrated consistent predictive power across the "Over" and "Under" classes. The model's capacity to accurately detect overpriced properties is especially useful in high-stakes real estate situations, even though recall for the "Under" category was lower. Because feature importances can still be extracted and shared with stakeholders, this model provides a meaningful balance between interpretability and complexity when compared to the more straightforward logistic regression and decision tree models. Even though it was the study's best-performing model, its 79% accuracy and inconsistent class performance suggest that it might not yet be adequate for institutional decision-making when it comes to million-dollar deals. Higher reliability standards, likely involving model calibration, additional tuning, and integration with domain expertise, would be necessary for such use. Although Model 3 is not a fully autonomous system for high-value transactions, it does show promise as a screening tool or decisional aid.

Code

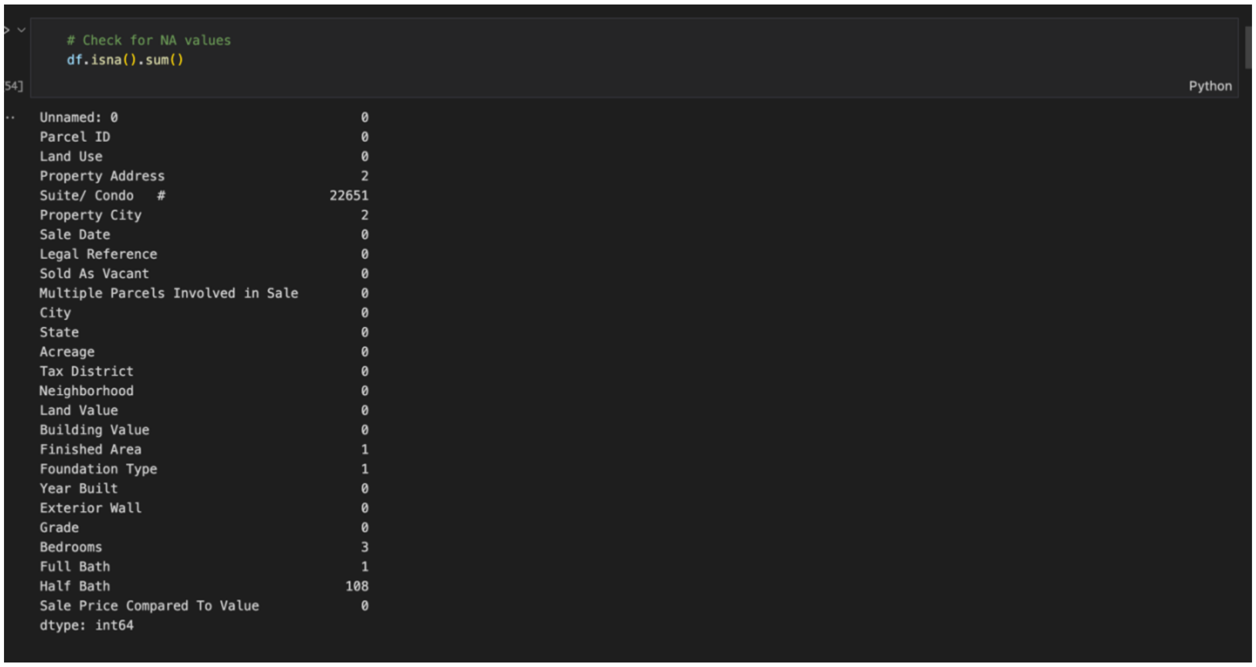


```
import pandas as pd
import xgboost as xgb
```
[174] Python

```
df = pd.read_csv('week 4 - Nashville_housing_data.csv')
df.head()
```
[53] Python

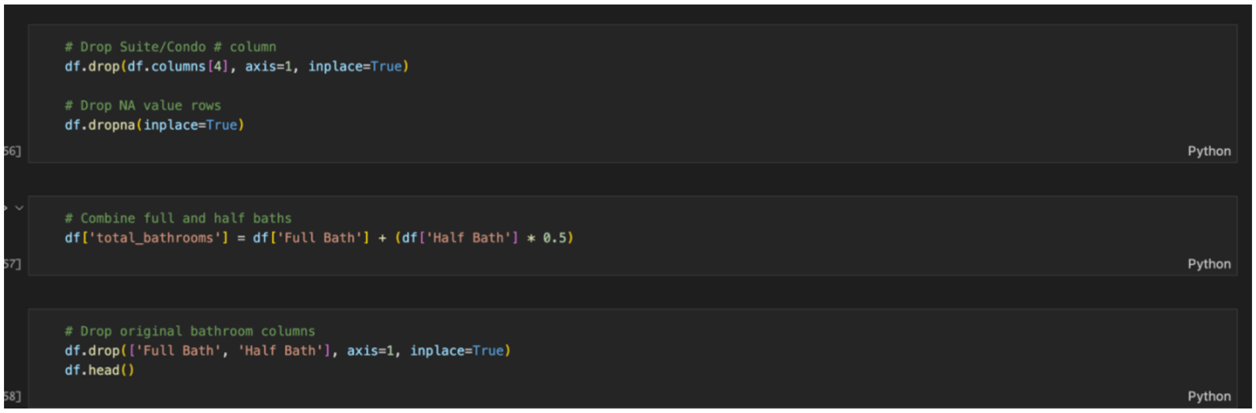
	Unnamed: 0	Parcel ID	Land Use	Property Address	Suite/ Condo #	Property City	Sale Date	Legal Reference	Sold As Vacant	Multiple Parcels Involved in Sale	...	Building Value	Finished Area	Foundation Type	Year Built	Exterior Wal
0	1	105 11 0 080.00	SINGLE FAMILY	1802 STEWART PL	NaN	NASHVILLE	1/11/2013	20130118-0006337	No	No	...	134400	1149.00000	PT BSMT	1941	BRICK
1	2	118 03 0 130.00	SINGLE FAMILY	2761 ROSEDALE PL	NaN	NASHVILLE	1/18/2013	20130124-0008033	No	No	...	157800	2090.82495	SLAB	2000	BRICK/FRAME
2	3	119 01 0 479.00	SINGLE FAMILY	224 PEACHTREE ST	NaN	NASHVILLE	1/18/2013	20130128-0008863	No	No	...	243700	2145.60001	FULL BSMT	1948	BRICK/FRAME
3	4	119 05 0 186.00	SINGLE FAMILY	316 LUTIE ST	NaN	NASHVILLE	1/23/2013	20130131-0009929	No	No	...	138100	1969.00000	CRAWL	1910	FRAME
4	5	119 05 0 387.00	SINGLE FAMILY	2626 FOSTER AVE	NaN	NASHVILLE	1/4/2013	20130118-0006110	No	No	...	86100	1037.00000	CRAWL	1945	FRAME

5 rows × 26 columns



```
# Check for NA values
df.isna().sum()
```
54] Python

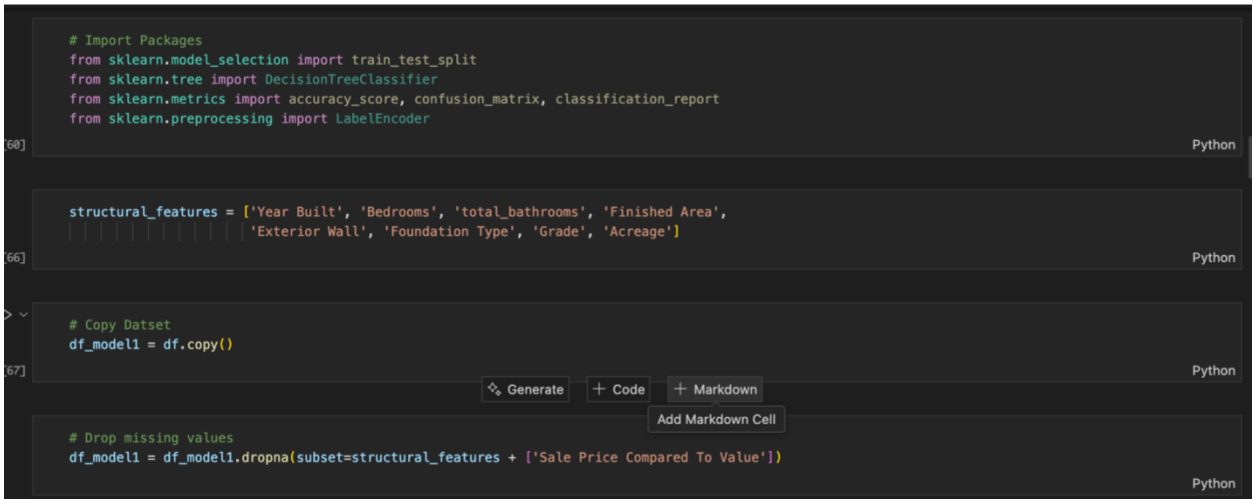
```
Unnamed: 0                            0
Parcel ID                             0
Land Use                              0
Property Address                      2
Suite/ Condo   #                  22651
Property City                         2
Sale Date                             0
Legal Reference                       0
Sold As Vacant                        0
Multiple Parcels Involved in Sale     0
City                                  0
State                                 0
Acreage                               0
Tax District                          0
Neighborhood                          0
Land Value                            0
Building Value                        0
Finished Area                         1
Foundation Type                       1
Year Built                            0
Exterior Wall                         0
Grade                                 0
Bedrooms                              3
Full Bath                             1
Half Bath                           108
Sale Price Compared To Value          0
dtype: int64
```



```python
# Drop Suite/Condo # column
df.drop(df.columns[4], axis=1, inplace=True)

# Drop NA value rows
df.dropna(inplace=True)
```

```python
# Combine full and half baths
df['total_bathrooms'] = df['Full Bath'] + (df['Half Bath'] * 0.5)
```

```python
# Drop original bathroom columns
df.drop(['Full Bath', 'Half Bath'], axis=1, inplace=True)
df.head()
```

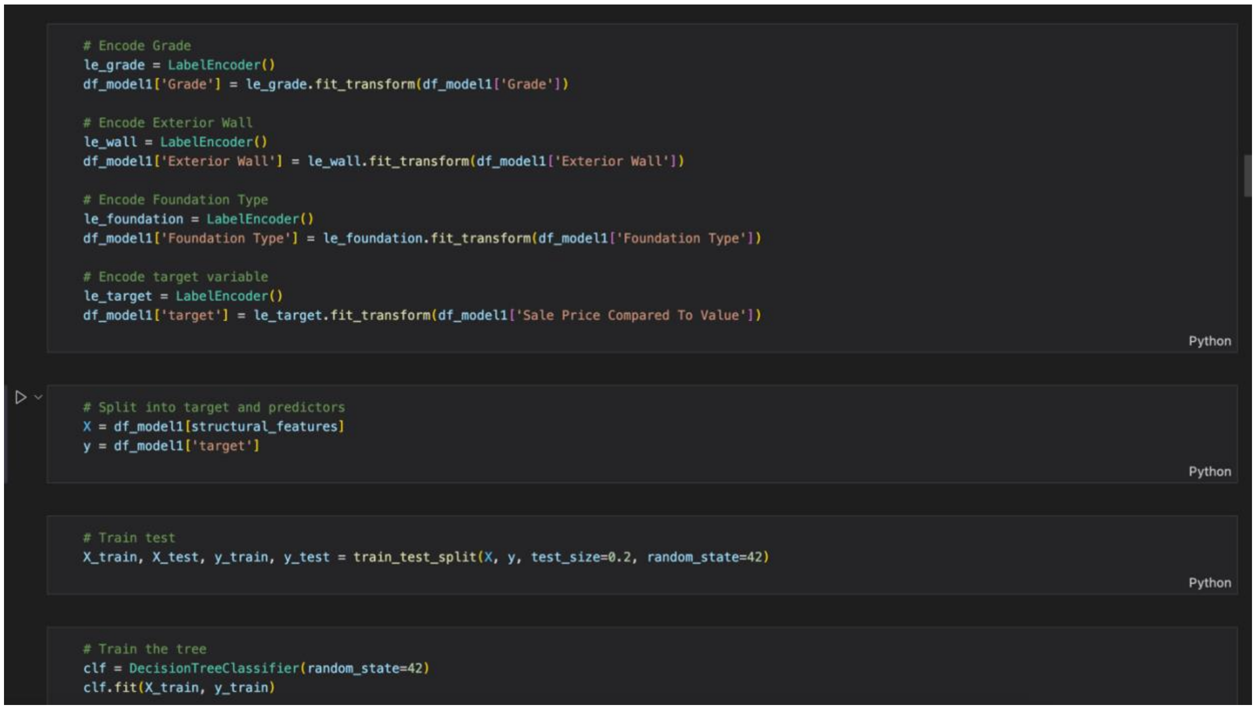


```python
# Import Packages
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder
```

```python
structural_features = ['Year Built', 'Bedrooms', 'total_bathrooms', 'Finished Area',
                       'Exterior Wall', 'Foundation Type', 'Grade', 'Acreage']
```

```python
# Copy Datset
df_model1 = df.copy()
```

```python
# Drop missing values
df_model1 = df_model1.dropna(subset=structural_features + ['Sale Price Compared To Value'])
```

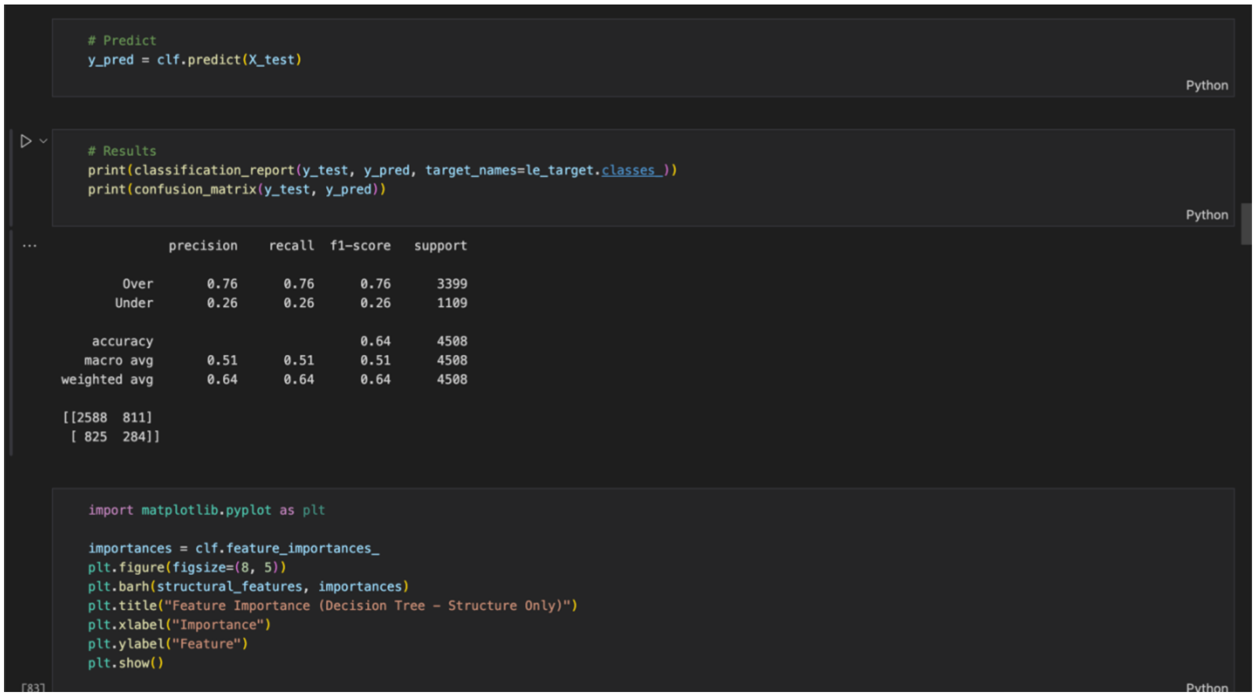


```python
# Encode Grade
le_grade = LabelEncoder()
df_model1['Grade'] = le_grade.fit_transform(df_model1['Grade'])

# Encode Exterior Wall
le_wall = LabelEncoder()
df_model1['Exterior Wall'] = le_wall.fit_transform(df_model1['Exterior Wall'])

# Encode Foundation Type
le_foundation = LabelEncoder()
df_model1['Foundation Type'] = le_foundation.fit_transform(df_model1['Foundation Type'])

# Encode target variable
le_target = LabelEncoder()
df_model1['target'] = le_target.fit_transform(df_model1['Sale Price Compared To Value'])
```

```python
# Split into target and predictors
X = df_model1[structural_features]
y = df_model1['target']
```

```python
# Train test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Train the tree
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
```



```
# Predict
y_pred = clf.predict(X_test)
```

Python

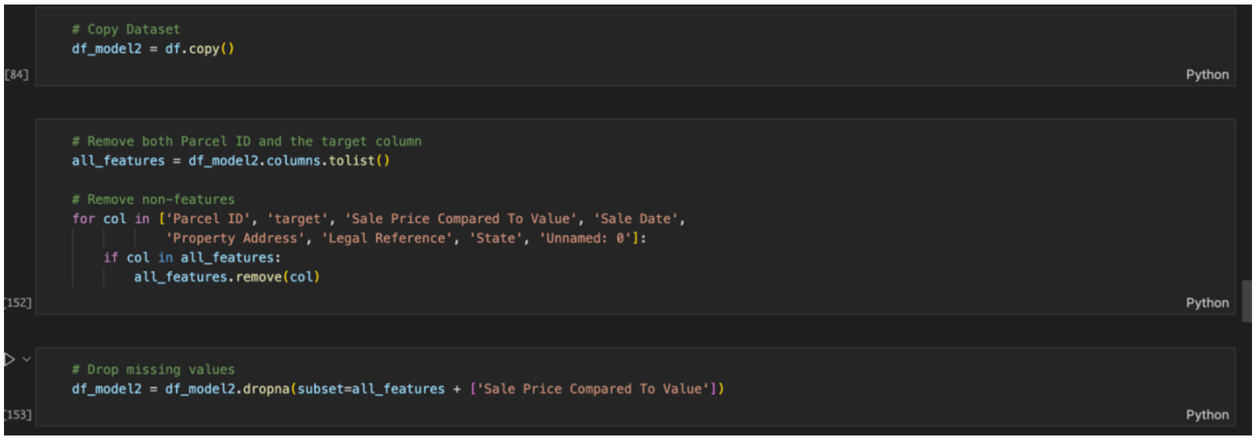
```
# Results
print(classification_report(y_test, y_pred, target_names=le_target.classes_))
print(confusion_matrix(y_test, y_pred))
```

Python

```
              precision    recall  f1-score   support

        Over       0.76      0.76      0.76      3399
       Under       0.26      0.26      0.26      1109

    accuracy                           0.64      4508
   macro avg       0.51      0.51      0.51      4508
weighted avg       0.64      0.64      0.64      4508

[[2588  811]
 [ 825  284]]
```

```
import matplotlib.pyplot as plt

importances = clf.feature_importances_
plt.figure(figsize=(8, 5))
plt.barh(structural_features, importances)
plt.title("Feature Importance (Decision Tree - Structure Only)")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```

[83] Python



```
# Copy Dataset
df_model2 = df.copy()
```

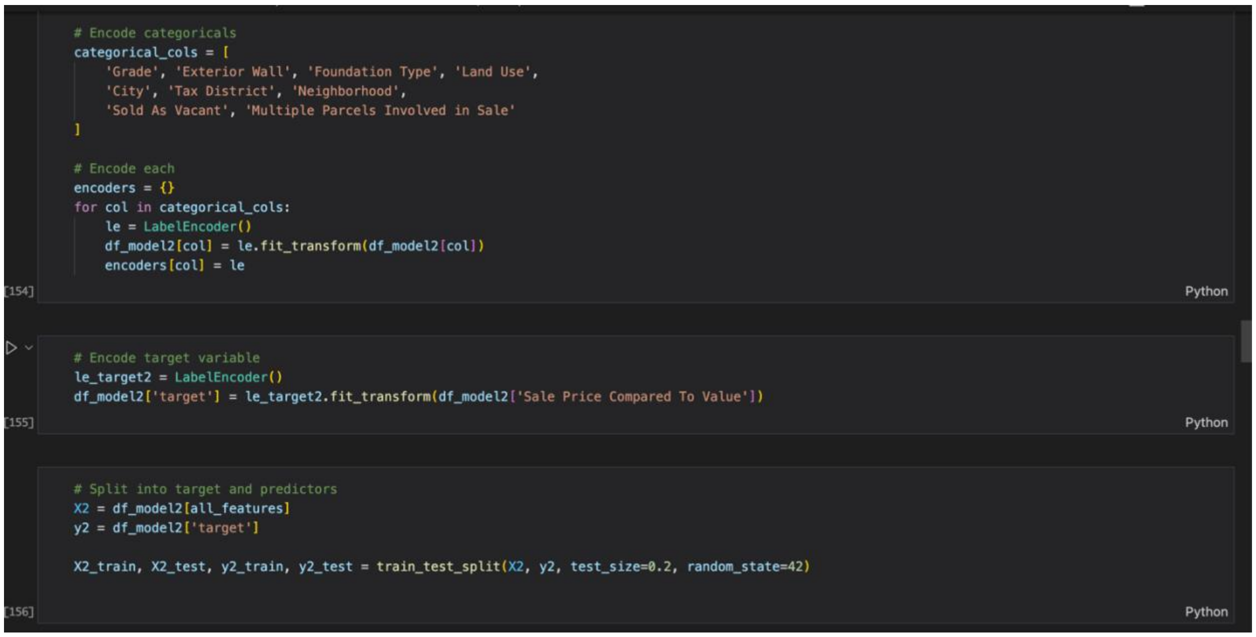
[84] Python

```
# Remove both Parcel ID and the target column
all_features = df_model2.columns.tolist()

# Remove non-features
for col in ['Parcel ID', 'target', 'Sale Price Compared To Value', 'Sale Date',
            'Property Address', 'Legal Reference', 'State', 'Unnamed: 0']:
    if col in all_features:
        all_features.remove(col)
```

[152] Python

```
# Drop missing values
df_model2 = df_model2.dropna(subset=all_features + ['Sale Price Compared To Value'])
```

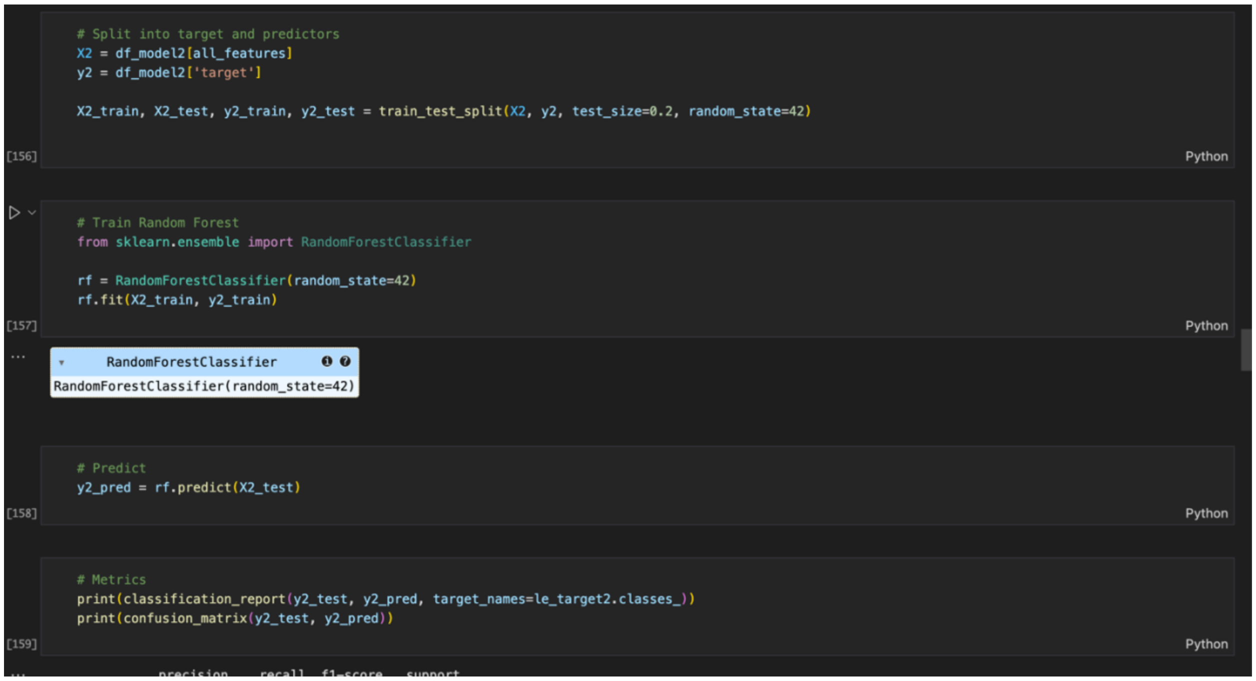
[153] Python



```python
# Encode categoricals
categorical_cols = [
    'Grade', 'Exterior Wall', 'Foundation Type', 'Land Use',
    'City', 'Tax District', 'Neighborhood',
    'Sold As Vacant', 'Multiple Parcels Involved in Sale'
]

# Encode each
encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_model2[col] = le.fit_transform(df_model2[col])
    encoders[col] = le
```
[154] Python

```python
# Encode target variable
le_target2 = LabelEncoder()
df_model2['target'] = le_target2.fit_transform(df_model2['Sale Price Compared To Value'])
```
[155] Python

```python
# Split into target and predictors
X2 = df_model2[all_features]
y2 = df_model2['target']

X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2, random_state=42)
```
[156] Python



```python
# Split into target and predictors
X2 = df_model2[all_features]
y2 = df_model2['target']

X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2, random_state=42)
```
[156] Python

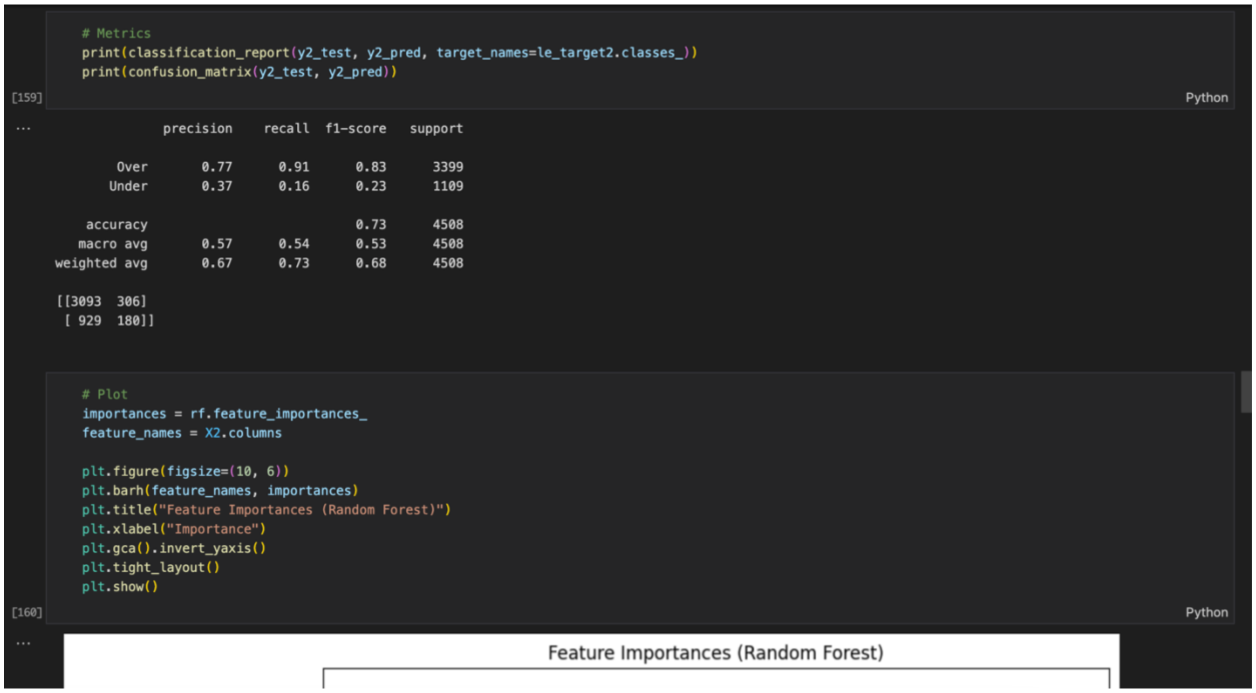
```python
# Train Random Forest
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)
rf.fit(X2_train, y2_train)
```
[157] Python

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```python
# Predict
y2_pred = rf.predict(X2_test)
```
[158] Python

```python
# Metrics
print(classification_report(y2_test, y2_pred, target_names=le_target2.classes_))
print(confusion_matrix(y2_test, y2_pred))
```
[159] Python

```
              precision    recall  f1-score   support
```

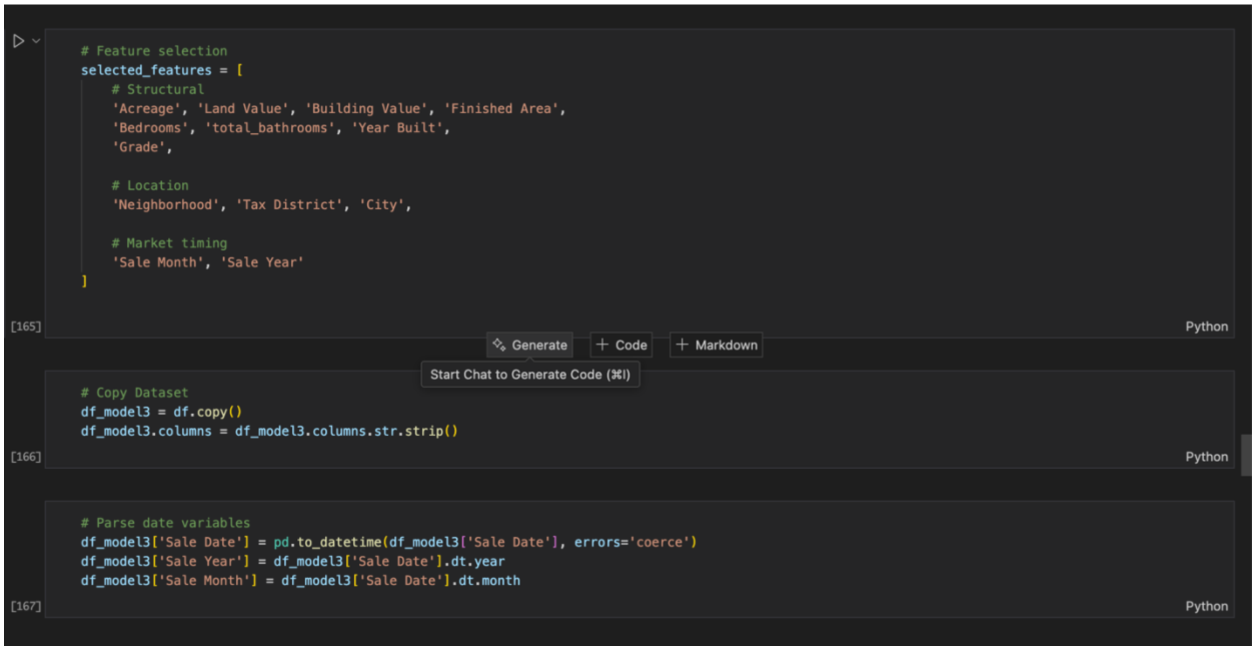


```
# Metrics
print(classification_report(y2_test, y2_pred, target_names=le_target2.classes_))
print(confusion_matrix(y2_test, y2_pred))
```
[159] Python

```
              precision    recall  f1-score   support

        Over       0.77      0.91      0.83      3399
       Under       0.37      0.16      0.23      1109

    accuracy                           0.73      4508
   macro avg       0.57      0.54      0.53      4508
weighted avg       0.67      0.73      0.68      4508

[[3093  306]
 [ 929  180]]
```

```
# Plot
importances = rf.feature_importances_
feature_names = X2.columns

plt.figure(figsize=(10, 6))
plt.barh(feature_names, importances)
plt.title("Feature Importances (Random Forest)")
plt.xlabel("Importance")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```
[160] Python

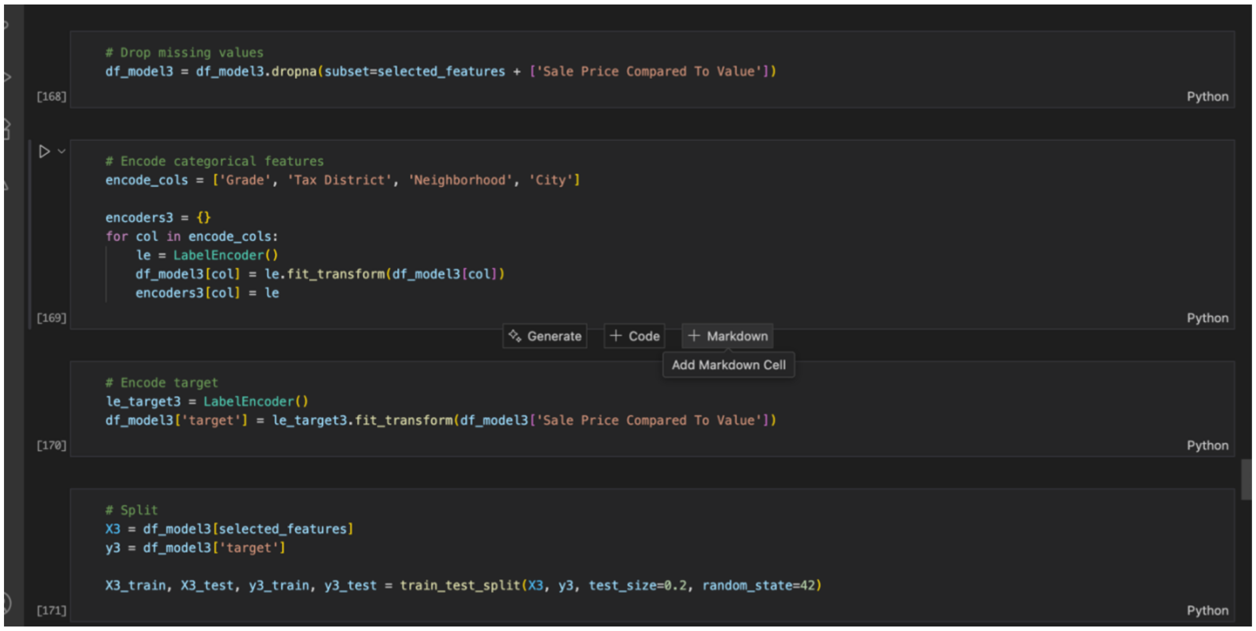
Feature Importances (Random Forest)



```
# Feature selection
selected_features = [
    # Structural
    'Acreage', 'Land Value', 'Building Value', 'Finished Area',
    'Bedrooms', 'total_bathrooms', 'Year Built',
    'Grade',

    # Location
    'Neighborhood', 'Tax District', 'City',

    # Market timing
    'Sale Month', 'Sale Year'
]
```
[165] Python

```
# Copy Dataset
df_model3 = df.copy()
df_model3.columns = df_model3.columns.str.strip()
```
[166] Python

```
# Parse date variables
df_model3['Sale Date'] = pd.to_datetime(df_model3['Sale Date'], errors='coerce')
df_model3['Sale Year'] = df_model3['Sale Date'].dt.year
df_model3['Sale Month'] = df_model3['Sale Date'].dt.month
```
[167] Python

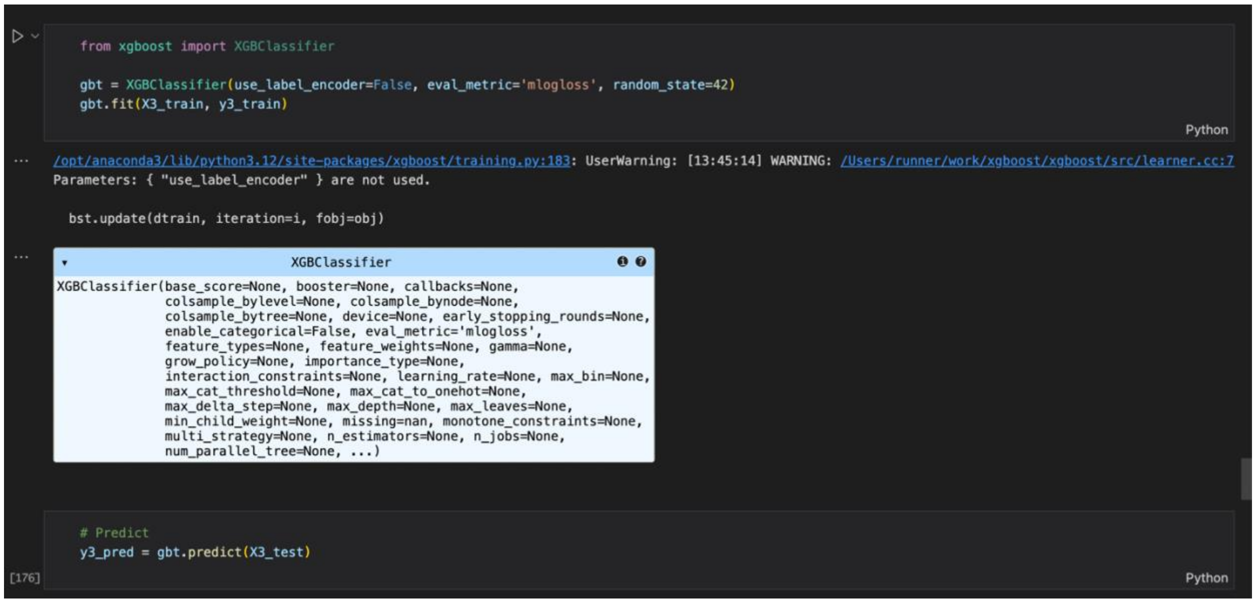


```python
# Drop missing values
df_model3 = df_model3.dropna(subset=selected_features + ['Sale Price Compared To Value'])
```
[168]

```python
# Encode categorical features
encode_cols = ['Grade', 'Tax District', 'Neighborhood', 'City']

encoders3 = {}
for col in encode_cols:
    le = LabelEncoder()
    df_model3[col] = le.fit_transform(df_model3[col])
    encoders3[col] = le
```
[169]

```python
# Encode target
le_target3 = LabelEncoder()
df_model3['target'] = le_target3.fit_transform(df_model3['Sale Price Compared To Value'])
```
[170]

```python
# Split
X3 = df_model3[selected_features]
y3 = df_model3['target']

X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_size=0.2, random_state=42)
```
[171]

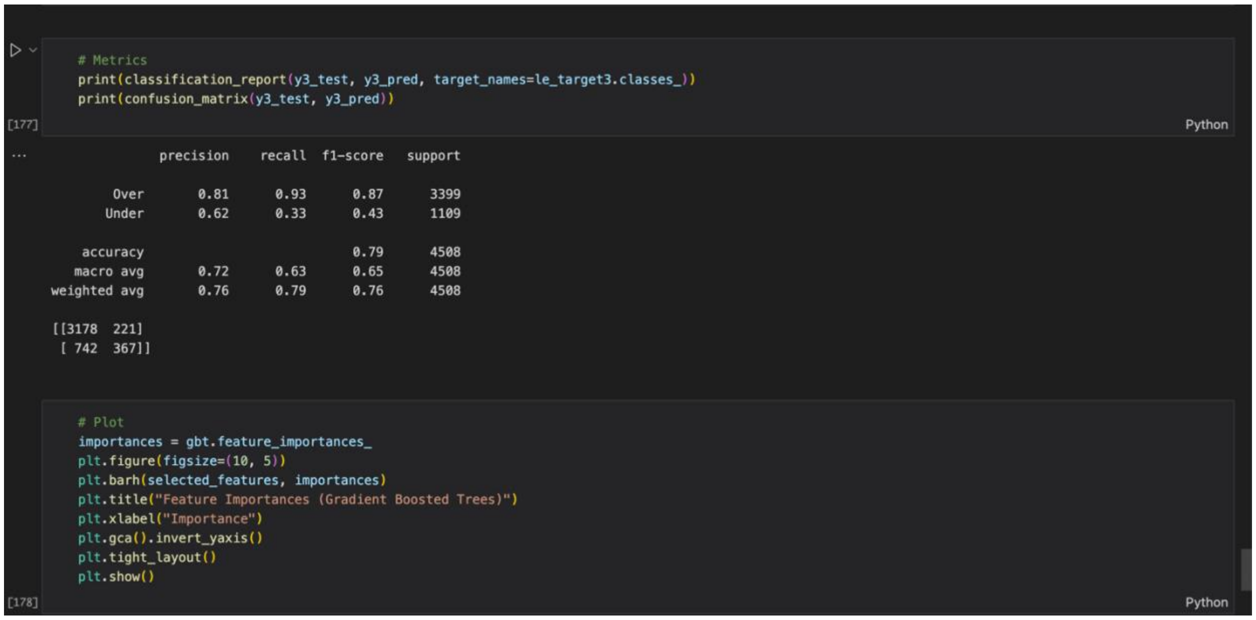


```python
from xgboost import XGBClassifier

gbt = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42)
gbt.fit(X3_train, y3_train)
```

```
/opt/anaconda3/lib/python3.12/site-packages/xgboost/training.py:183: UserWarning: [13:45:14] WARNING: /Users/runner/work/xgboost/xgboost/src/learner.cc:7
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='mlogloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, ...)
```

```python
# Predict
y3_pred = gbt.predict(X3_test)
```
[176]



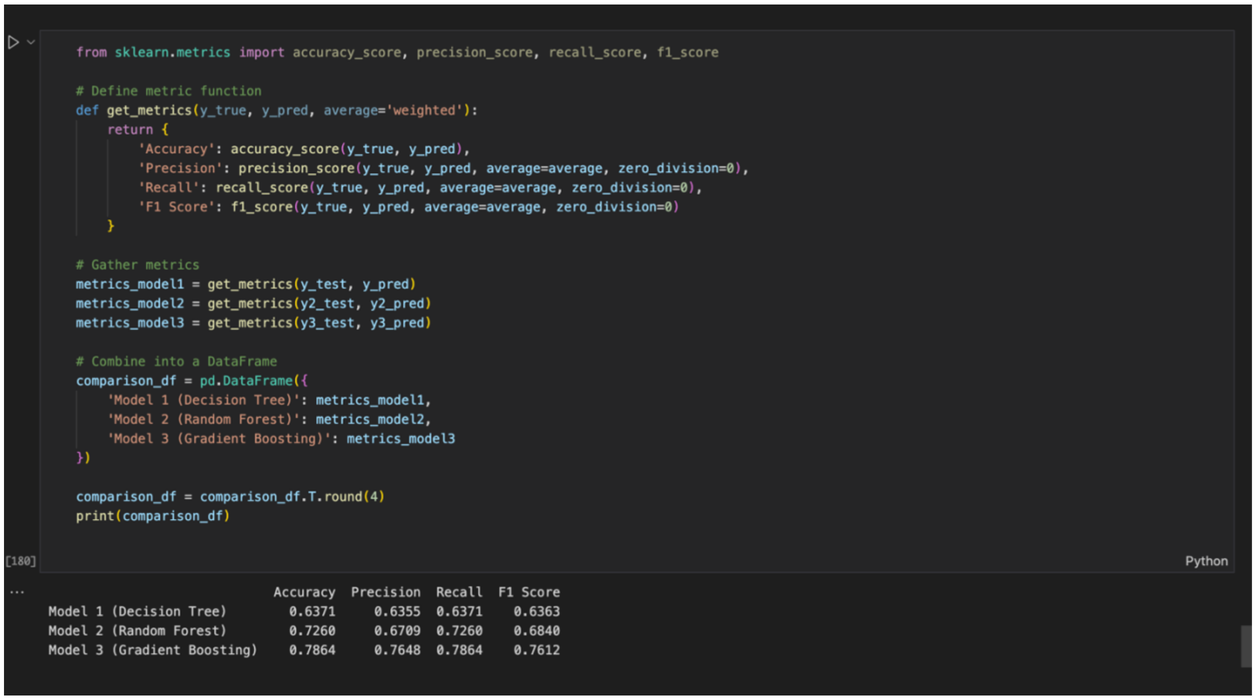
```
# Metrics
print(classification_report(y3_test, y3_pred, target_names=le_target3.classes_))
print(confusion_matrix(y3_test, y3_pred))
```

[177] Python

```
              precision    recall  f1-score   support

        Over       0.81      0.93      0.87      3399
       Under       0.62      0.33      0.43      1109

    accuracy                           0.79      4508
   macro avg       0.72      0.63      0.65      4508
weighted avg       0.76      0.79      0.76      4508

[[3178  221]
 [ 742  367]]
```

```
# Plot
importances = gbt.feature_importances_
plt.figure(figsize=(10, 5))
plt.barh(selected_features, importances)
plt.title("Feature Importances (Gradient Boosted Trees)")
plt.xlabel("Importance")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

[178] Python



```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Define metric function
def get_metrics(y_true, y_pred, average='weighted'):
    return {
        'Accuracy': accuracy_score(y_true, y_pred),
        'Precision': precision_score(y_true, y_pred, average=average, zero_division=0),
        'Recall': recall_score(y_true, y_pred, average=average, zero_division=0),
        'F1 Score': f1_score(y_true, y_pred, average=average, zero_division=0)
    }

# Gather metrics
metrics_model1 = get_metrics(y_test, y_pred)
metrics_model2 = get_metrics(y2_test, y2_pred)
metrics_model3 = get_metrics(y3_test, y3_pred)

# Combine into a DataFrame
comparison_df = pd.DataFrame({
    'Model 1 (Decision Tree)': metrics_model1,
    'Model 2 (Random Forest)': metrics_model2,
    'Model 3 (Gradient Boosting)': metrics_model3
})

comparison_df = comparison_df.T.round(4)
print(comparison_df)
```

[180] Python

```
                             Accuracy  Precision  Recall  F1 Score
Model 1 (Decision Tree)        0.6371     0.6355  0.6371    0.6363
Model 2 (Random Forest)        0.7260     0.6709  0.7260    0.6840
Model 3 (Gradient Boosting)    0.7864     0.7648  0.7864    0.7612
```