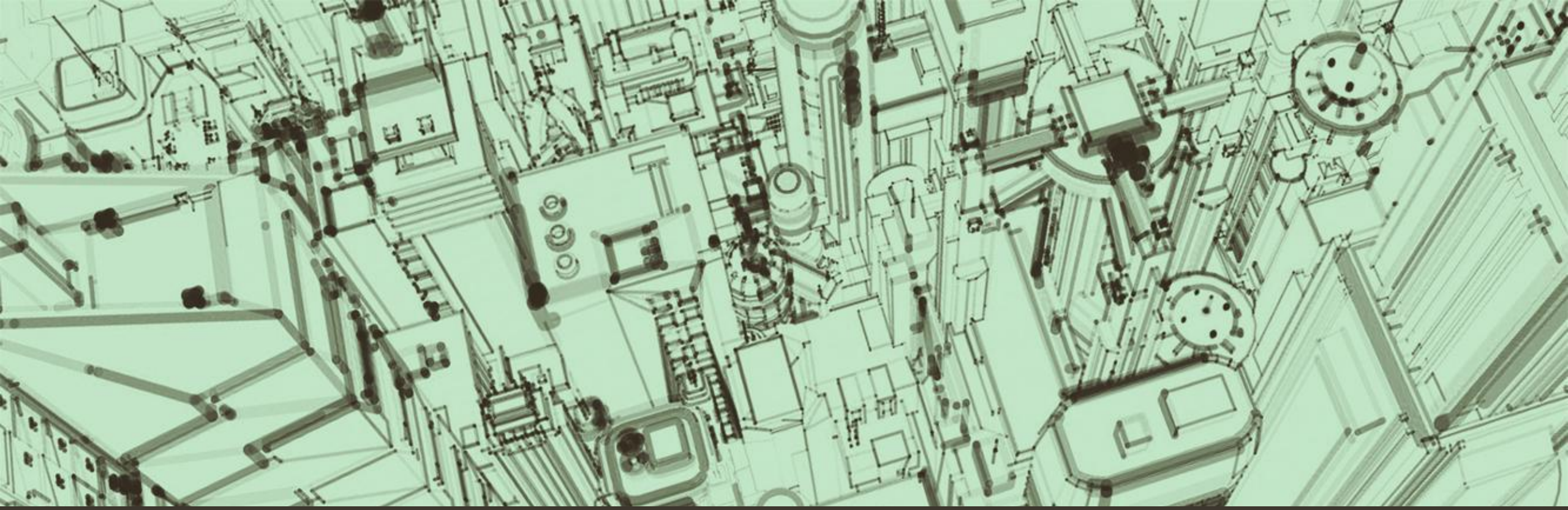


Triangle Real Estate

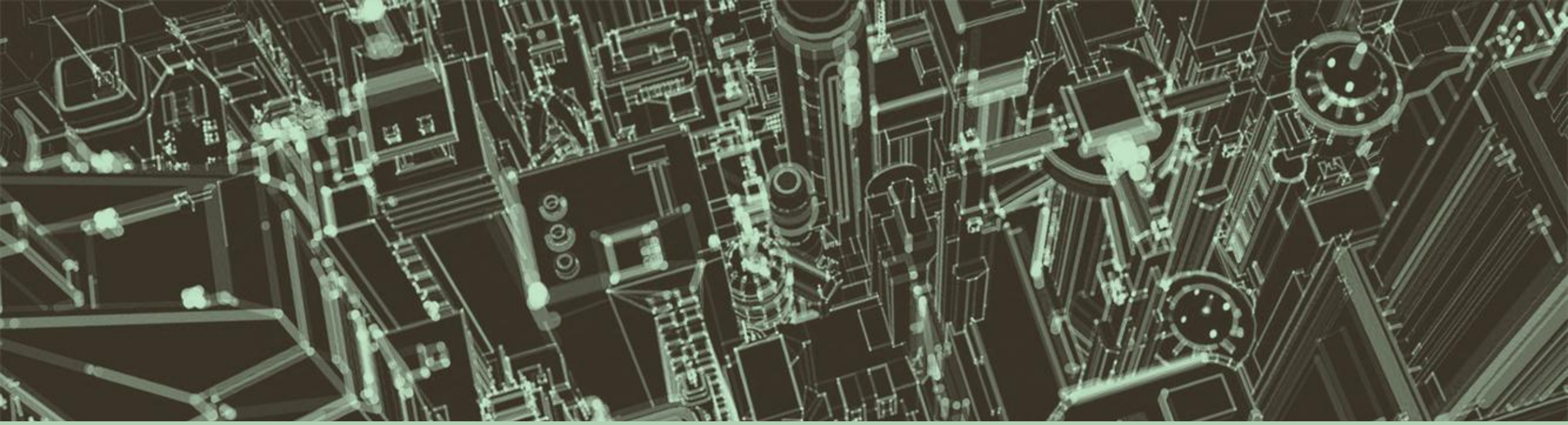
Amelia, Brandi, Catina, David, Kurt

Executive Summary

We will continue working with real estate data from the Multiple Listing Service (MLS), focusing on building a predictive model to accurately estimate home sale prices. We will leverage Gradient Boosting Machines (GBM) and Natural Language Processing (NLP) techniques to enhance the model's performance.

Project Approach

- Our team's goal was to produce two deliverables:
  - A regression model for predicting the sale price of a home
  - An NLP model which assesses mortgage rate sentiment from news articles
- For the regression model:
  - 4 years of housing data were collected from the MLS
  - Data was cleaned and engineered to produce necessary features
  - Multiple regression models were researched and tested in order to determine the best model for price prediction. These included Random Forest, XGBoost, CatBoost, and LightGBM
- For the NLP model:
  - BERT (Bidirectional Encoder Representations from Transformers) and VADER (Valence Aware Dictionary for Sentiment Reasoning) models utilized
  - Total Articles Analyzed: 10,000+
  - Time Range:
    - January 2019 to December 2019, General News Articles
      - Sources: 20 different news outlets
      - Categories: Various topics including politics, crime, business, and technology
    - January 2022 to July 2024, New York Times Articles API based on the following keywords: Real Estate, Federal Reserve, Interest Rates, Home Selling
  - Initial goal to utilize only articles selected based on keywords, encountered API limits requiring the use of general news articles available for public use.



Regression: Data Engineering

Data Engineering

- Four years worth of real estate data were collected from the MLS, including features such as Address, City, Zip, Beds, Baths, Sqft, Acres, etc.
- Only single-family homes were utilized, and only cities which contained 1000 or more data points
- Primarily null columns were removed (subdivision, neighborhood), along with any rows containing features whose values fell outside 3 standard deviations of the mean
- Using a python library, addresses were converted to lat and lon coordinates
- Data was scaled to make it suitable for the models
- Cities were one-hot encoded to handle categorical aspects

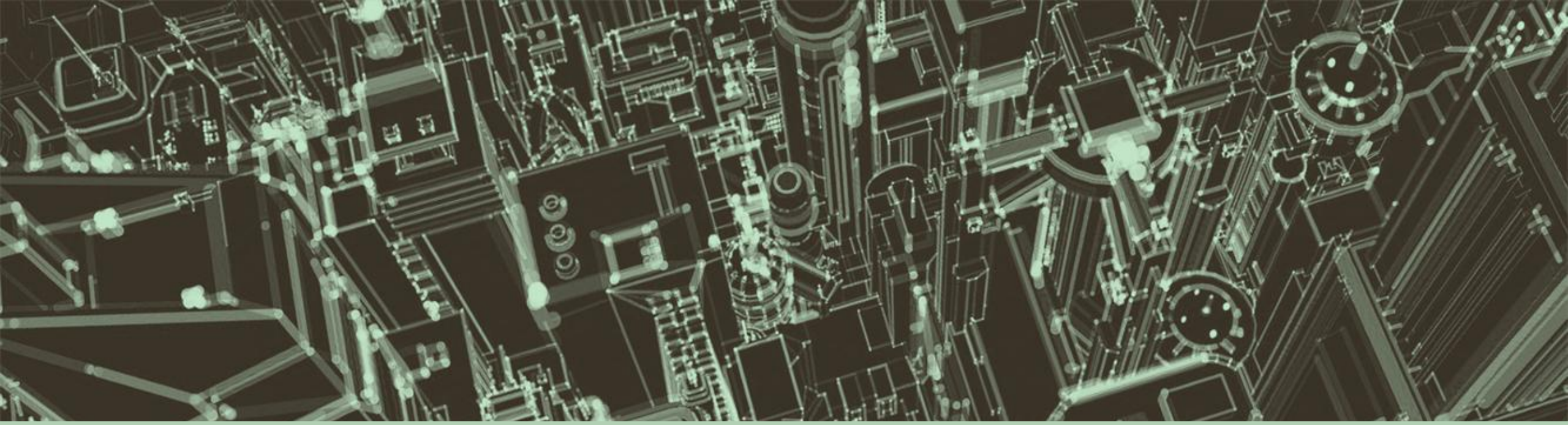
Data Engineering

- Because neither latitude nor longitude by themselves have a direct effect on the sale price of the home, we needed a way to combine these two features
- Multiple methods for combining lat and lon data were researched, such as euclidean scalars, geohashing, and embedding
- It was decided to create a deep learning neural network, from which spatial features could be extracted from the dense layers, and plugged back into the regression models using a hybrid approach
- 32 spatial features were generated from the NN and plugged back into the data to be used for the regressor models

Data Engineering

- Interestingly, the best results from the Random Forest model were obtained when utilizing both the Lat/Lon and embedded spatial data. Choosing only one of the two options resulted in lower scores.
- By using this embedded approach alongside the Lat and Lon, we were able to significantly improve on the results from our last project
- Because tree-based models like Random Forest and XGBoost handle redundant data well, leaving both sets of features in the data proved to be beneficial

```
Random Forest Regressor - Train MAE: 16504.783259767235
Random Forest Regressor - Test MAE: 44391.89741333775
Random Forest Regressor - Train MSE: 616148879.9088664
Random Forest Regressor - Test MSE: 4451159286.066654
Random Forest Regressor - Train R2: 0.9819719841521849
Random Forest Regressor - Test R2: 0.868768533909393 5
```



Regression:
CatBoost Modules

CatBoost: 1st Iteration

Using 3 x Hyperparameters

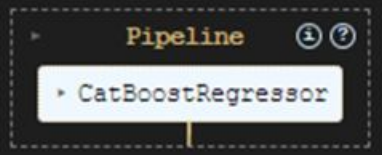
```
# Define the search space for CatBoost hyperparameters
search_space = {
    'clf__iterations': Integer(5, 10),
    'clf__learning_rate': Real(0.001, 0.1, prior='log-uniform'),
    'clf__depth': Integer(4, 8),
    #'clf__l2_leaf_reg': Real(20, 30, prior='log-uniform'),
    #'clf__bagging_temperature': Real(0, 1),
    #'clf__random_strength': Real(0, 10),
    #'clf__one_hot_max_size': Integer(2, 10),
    #'clf__rsm': Real(0.5, 1.0),
    #'clf__boosting_type': Categorical(['Ordered', 'Plain']),
    #'clf__od_type': Categorical(['IncToDec', 'Iter']),
    #'clf__od_wait': Integer(10, 20)
}
```

opt.score and opt.best_score

```
opt.score(X_test, y_test)
```
✓ 0.0s

```
0.6311455735256641
```

```
#Evaluate the model and make predictions
opt.best_estimator_
```
✓ 0.0s




```
opt.best_score_
```
✓ 0.0s

```
0.627493023811888
```

Score model using MSE and MAE

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

#make predictions with training data
y_pred_train = opt.predict(X_train)

#score model using MSE and MAE
mse_train = mean_squared_error(y_train, y_pred_train)
mae_train = mean_absolute_error(y_train, y_pred_train)
print(f"Mean Squared Error (MSE) for trained data: {mse_train}")
print(f"Mean Absolute Error (MAE) for trained data: {mae_train}")
```
✓ 0.0s

```
Mean Squared Error (MSE) for trained data: 12582155346.005207
Mean Absolute Error (MAE) for trained data: 84268.43306949637
```

```
# Make predictions with testing data
y_pred = opt.predict(X_test)

#score model using MSE and MAE
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
```
✓ 0.0s

```
Mean Squared Error (MSE): 12587194777.104313
Mean Absolute Error (MAE): 84295.8105864517
```

CatBoost: 2nd Iteration

Using 3 x Hyperparameters

```
# Define the search space for CatBoost hyperparameters
search_space = {
    'clf__iterations': Integer(10, 20),
    'clf__learning_rate': Real(0.001, 0.1, prior='log-uniform'),
    'clf__depth': Integer(8, 12),
    #'clf__l2_leaf_reg': Real(20, 30, prior='log-uniform'),
    #'clf__bagging_temperature': Real(0, 1),
    #'clf__random_strength': Real(0, 10),
    #'clf__one_hot_max_size': Integer(2, 10),
    #'clf__rsm': Real(0.5, 1.0),
    #'clf__boosting_type': Categorical(['Ordered', 'Plain']),
    #'clf__od_type': Categorical(['IncToDec', 'Iter']),
    #'clf__od_wait': Integer(10, 20)
}
```

opt.score and opt.best_score

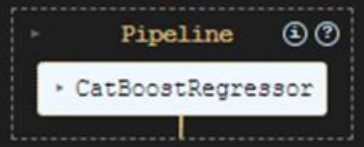
```
opt.score(X_test, y_test)
```

✓ 0.0s

```
0.7906015630921458
```

```
#Evaluate the model and make predictions
opt.best_estimator_
```

✓ 0.0s




```
opt.best_score_
```

✓ 0.0s

```
0.7879264481006916
```

Score model using MSE and MAE

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

#make predictions with training data
y_pred_train = opt.predict(X_train)

#score model using MSE and MAE
mse_train = mean_squared_error(y_train, y_pred_train)
mae_train = mean_absolute_error(y_train, y_pred_train)
print(f"Mean Squared Error (MSE) for trained data: {mse_train}")
print(f"Mean Absolute Error (MAE) for trained data: {mae_train}")
```

✓ 0.0s

```
Mean Squared Error (MSE) for trained data: 7056042209.196917
Mean Absolute Error (MAE) for trained data: 60636.80515128903
```

```
# Make predictions with testing data
y_pred = opt.predict(X_test)

#score model using MSE and MAE
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
```

✓ 0.0s

```
Mean Squared Error (MSE): 7145742933.259169
Mean Absolute Error (MAE): 61000.023295875515
```

CatBoost: 3rd Iteration

Using 3 x Hyperparameters

```
# Define the search space for CatBoost hyperparameters
search_space = {
    'clf__iterations': Integer(30, 50),
    'clf__learning_rate': Real(0.001, 0.1, prior='log-uniform'),
    'clf__depth': Integer(10, 16),
    #'clf__l2_leaf_reg': Real(20, 30, prior='log-uniform'),
    #'clf__bagging_temperature': Real(0, 1),
    #'clf__random_strength': Real(0, 10),
    #'clf__one_hot_max_size': Integer(2, 10),
    #'clf__rsm': Real(0.5, 1.0),
    #'clf__boosting_type': Categorical(['Ordered', 'Plain']),
    #'clf__od_type': Categorical(['IncToDec', 'Iter']),
    #'clf__od_wait': Integer(10, 20)
}
```

opt.score and opt.best_score

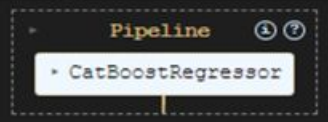
```
opt.score(X_test, y_test)
```
✓ 0.0s

```
0.8624663022162016
```

```
# Add opt.score training data **
opt.score(X_train, y_train)
```
✓ 0.0s

```
0.8860522862795976
```

```
#Evaluate the model and make predictions
opt.best_estimator_
```
✓ 0.0s




```
opt.best_score_
```
✓ 0.0s

```
0.8578964109847129
```

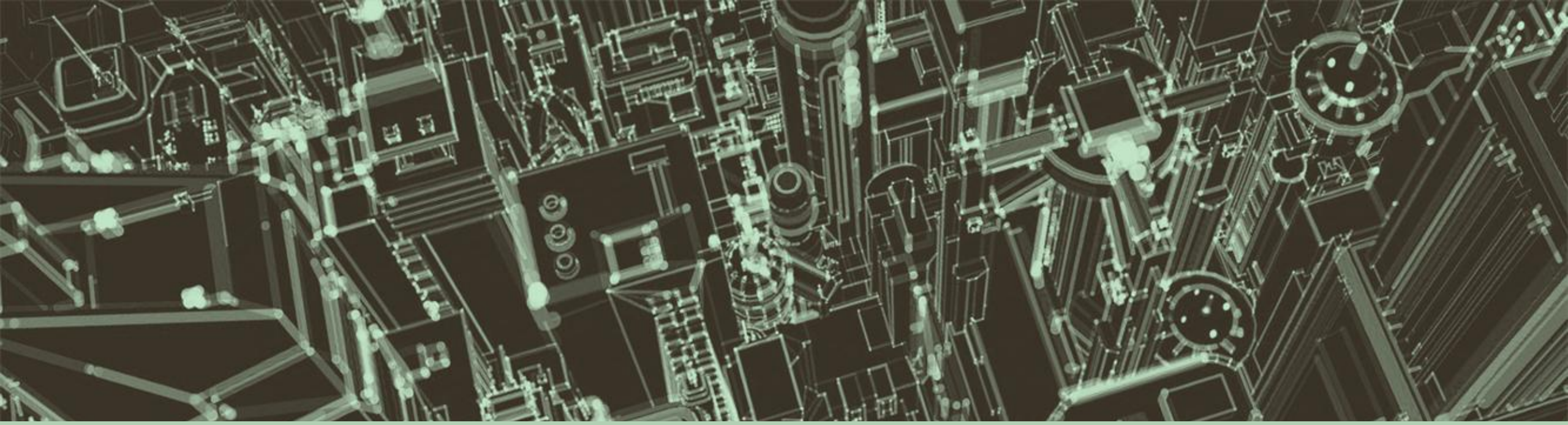
Score model using MSE and MAE

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

#make predictions with training data
y_pred_train = opt.predict(X_train)

#score model using MSE and MAE
mse_train = mean_squared_error(y_train, y_pred_train)
mae_train = mean_absolute_error(y_train, y_pred_train)
print(f"Mean Squared Error (MSE) for trained data: {mse_train}")
print(f"Mean Absolute Error (MAE) for trained data: {mae_train}")
```
✓ 0.0s

```
Mean Squared Error (MSE) for trained data: 3888506018.3560705
Mean Absolute Error (MAE) for trained data: 44004.627689416615
```

```
# Make predictions with testing data
y_pred = opt.predict(X_test)

#score model using MSE and MAE
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
```
✓ 0.0s

```
Mean Squared Error (MSE): 4693351409.571658
Mean Absolute Error (MAE): 47602.25807342462
```

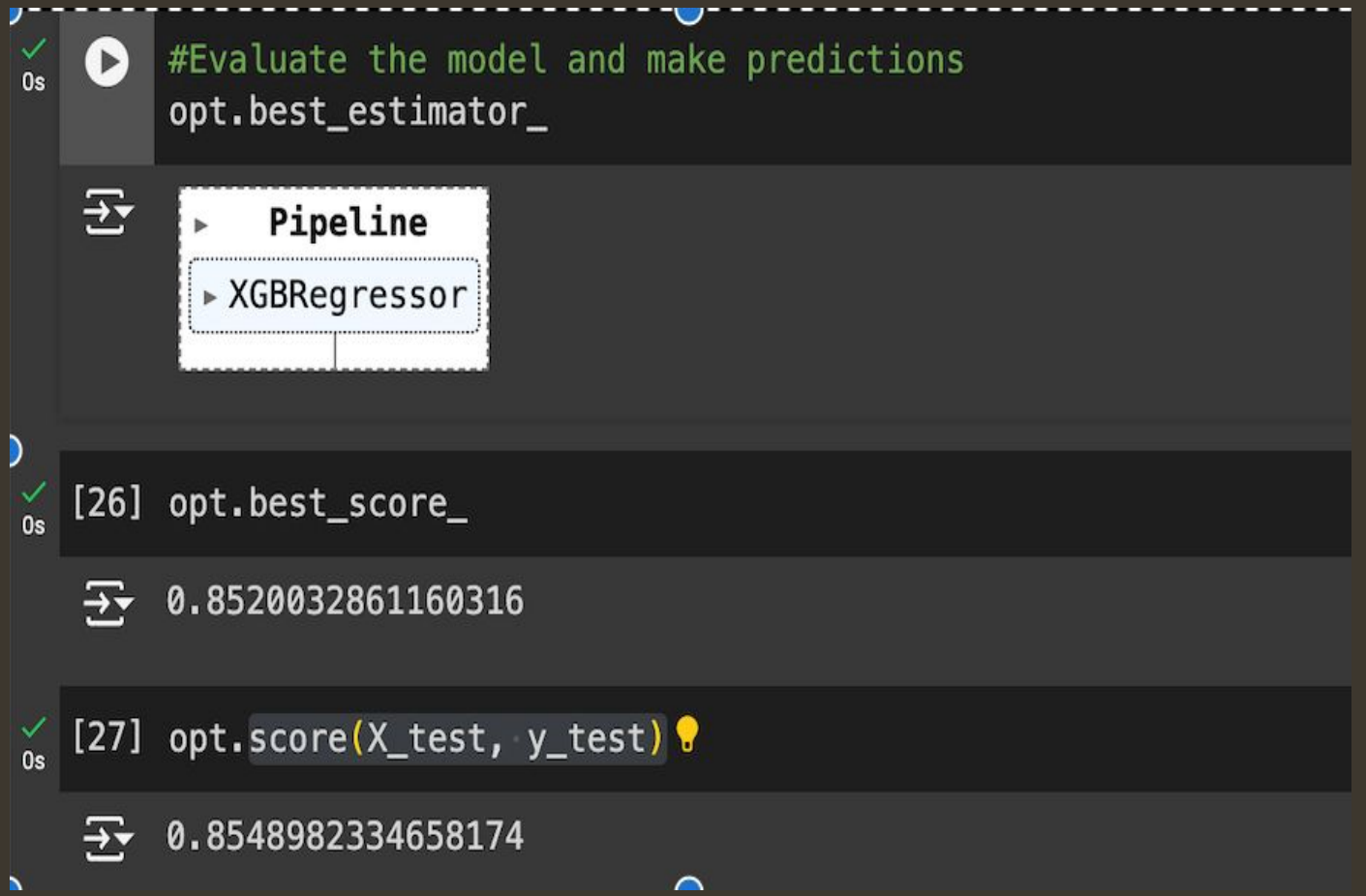


Regression:
XG Boost Modules

XGBoost 1st round:

- Bayes Search CV(cv=8, n_iter=20, scoring=neg_mean_squared_error)
- Nine parameters are used here.
- No predictions were made yet.

```
max_depth': Integer(2,8),
learning_rate': Real(0.001, 1.0, prior='log-uniform')
subsample': Real(0.5, 1.0),
colsample_bytree': Real(0.5, 1.0),
colsample_bylevel': Real(0.5, 1.0),
colsample_bynode' : Real(0.5, 1.0),
reg_alpha': Real(0.0, 10.0),
reg_lambda': Real(0.0, 10.0),
gamma': Real(0.0, 10.0)
```



XGBoost 4th round:

- Added: ‘n_estimators:Integer (50, 500)’, ‘min_child_weight:Integer (1, 10)’
- Adjusted: ‘reg_alpha (0.0, 2.0)’, ‘gamma (0.0, 5.0)’
- Adjusted: CV=8, n_iter=40
- scoring=r2, added MSE & MAE for testing data

```
#Set up hyperparameter tuning
from skopt import BayesSearchCV
from skopt.space import Real, Integer

search_space = {
    'clf__max_depth': Integer(3,10),
    'clf__learning_rate': Real(0.01, 0.3, prior='log-uniform'),
    'clf__subsample': Real(0.6, 1.0),
    'clf__colsample_bytree': Real(0.6, 1.0),
    'clf__colsample_bylevel': Real(0.6, 1.0),
    'clf__colsample_bynode' : Real(0.6, 1.0),
    'clf__reg_alpha': Real(0.0, 2.0),
    'clf__reg_lambda': Real(0.0, 5.0),
    'clf__gamma': Real(0.0, 5.0),
    'clf__min_child_weight': Integer(1, 10),
    'clf__n_estimators': Integer(50, 500)
}

# Perform cross-validation with early stopping
bayes_search = BayesSearchCV(pipe, search_space, cv=8, n_iter=40, scoring='r2', random_state=8)
```

```
[14] #Evaluate the model and make predictions
     best_model = bayes_search.best_estimator_

     bayes_search.best_score_

     0.8588677352270248

     + Code

[16] bayes_search.score(X_test, y_test)

     0.861725126053729

[17] #make predictions with testing data
     y_pred = bayes_search.predict(X_test)

[18] #ensure all data in y_pred are integers
     y_pred = y_pred.round().astype(int)

[19] from sklearn.metrics import mean_squared_error, mean_absolute_error
     #score model using MSE and MAE
     mse = mean_squared_error(y_test, y_pred)
     mae = mean_absolute_error(y_test, y_pred)

     print(f"Mean Squared Error (MSE): {mse}")
     print(f"Mean Absolute Error (MAE): {mae}")

     Mean Squared Error (MSE): 4426753958.21613
     Mean Absolute Error (MAE): 44980.30990058239
```

XGBoost 9th round:

- Used updated data.
- Adjusted 4 of the hyperparameters to accommodate data size & prevent overfitting.
- n_iter=50
- added MSE & MAE for training data

```
#Set up hyperparameter tuning
from skopt import BayesSearchCV
from skopt.space import Real, Integer

search_space = {
    'clf__max_depth': Integer(3, 12),
    'clf__learning_rate': Real(0.01, 0.3, prior='log-uniform'),
    'clf__subsample': Real(0.6, 1.0),
    'clf__colsample_bytree': Real(0.6, 1.0),
    'clf__colsample_bylevel': Real(0.6, 1.0),
    'clf__colsample_bynode' : Real(0.6, 1.0),
    'clf__reg_alpha': Real(0.0, 3.0),
    'clf__reg_lambda': Real(0.0, 5.0),
    'clf__gamma': Real(0.0, 5.0),
    'clf__min_child_weight': Integer(2, 10),
    'clf__n_estimators': Integer(50, 600)
}
# Perform cross-validation with early stopping
bayes_search = BayesSearchCV(pipe, search_space, cv=8, n_iter=50, scoring='r2', random_state=8)
```

```
[13] #Evaluate the model and make predictions
     best_model = bayes_search.best_estimator_
```

```
[14] bayes_search.best_score_
```

```
0.8943478956915709
```

```
[15] bayes_search.score(X_test, y_test)
```

```
0.8974277332683237
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
y_pred_train = bayes_search.predict(X_train)

mse_train = mean_squared_error(y_train, y_pred_train)
mae_train = mean_absolute_error(y_train, y_pred_train)

print(f"Mean Squared Error (MSE) for trained data: {mse_train}")
print(f"Mean Absolute Error (MAE) for trained data: {mae_train}")
```

```
Mean Squared Error (MSE) for trained data: 1171949762.6859877
Mean Absolute Error (MAE) for trained data: 24463.335774052095
```

```
[17] #make predictions with testing data
     y_pred = bayes_search.predict(X_test)
```

```
[18] #ensure all data in y_pred are integers
     y_pred = y_pred.round().astype(int)
```

```
[19] #score model using MSE and MAE
     mse = mean_squared_error(y_test, y_pred)
     mae = mean_absolute_error(y_test, y_pred)

     print(f"Mean Squared Error (MSE): {mse}")
     print(f"Mean Absolute Error (MAE): {mae}")
```

```
Mean Squared Error (MSE): 3500288309.0589004
Mean Absolute Error (MAE): 40049.72967422757
```

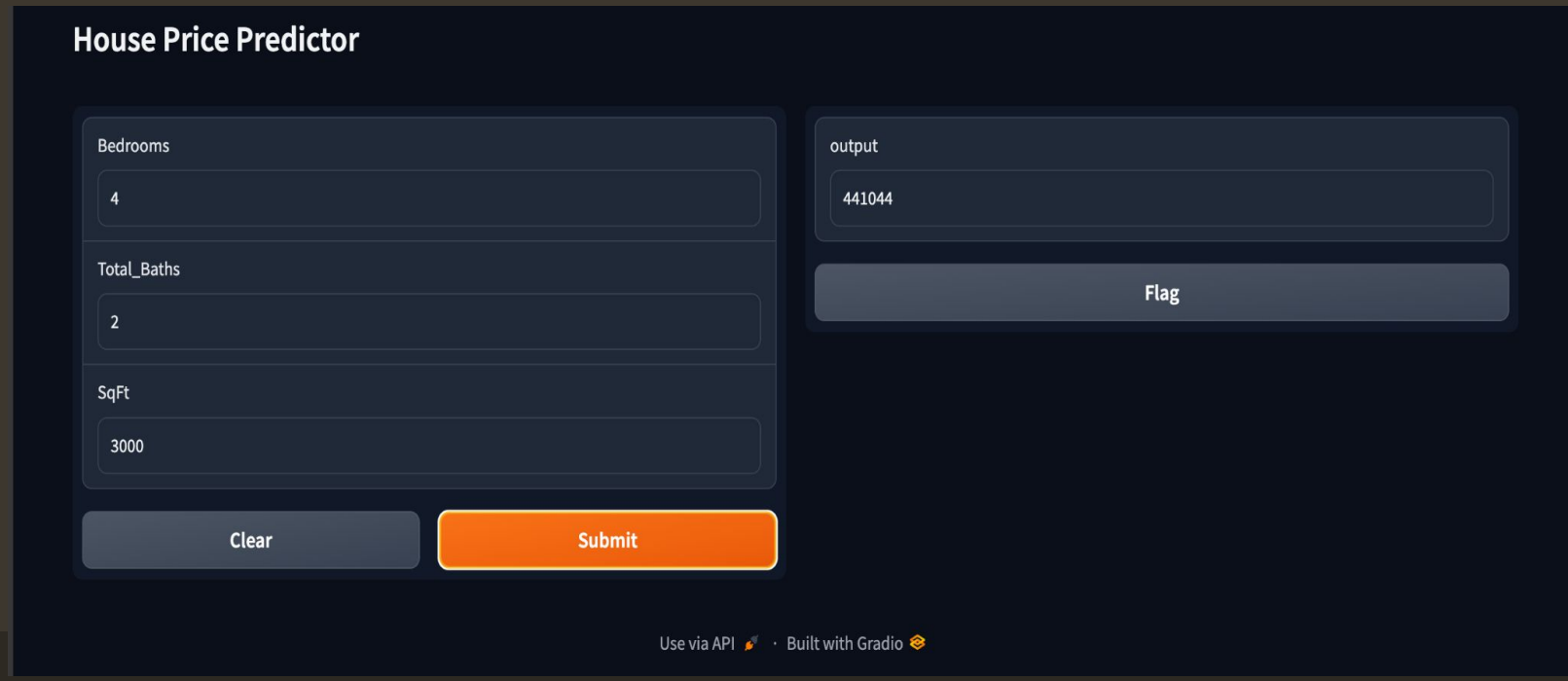
```
sfr_gradio = sfr_df[["Bedrooms", "Total Baths", "SqFt", "Sold Price"]]
sfr_gradio.head()
```

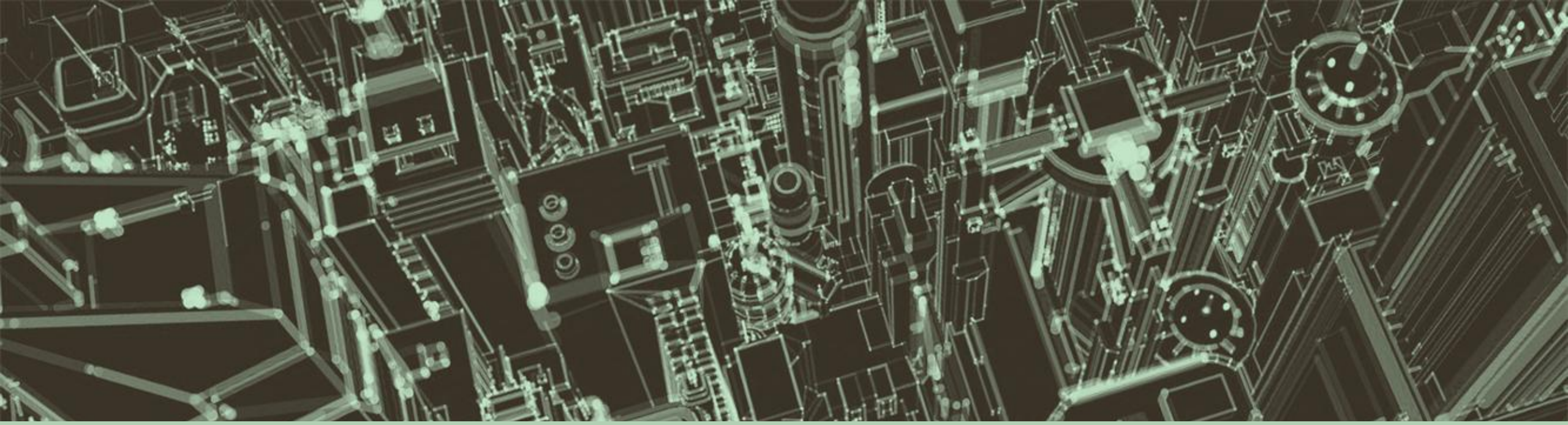
	Bedrooms	Total Baths	SqFt	Sold Price
0	3	1	-1.569749	54000
1	3	2	-1.096891	57500
2	2	1	-1.626796	60000
3	3	1	-1.482276	62000
4	3	1	0.109973	63000

XGBoost Mock Gradio:

- Created a new dataframe with only 4 of the features to prepare for a mock gradio interface.
- Built a function for gradio using Bedrooms, Total Baths, and Sqft to predict the home price.

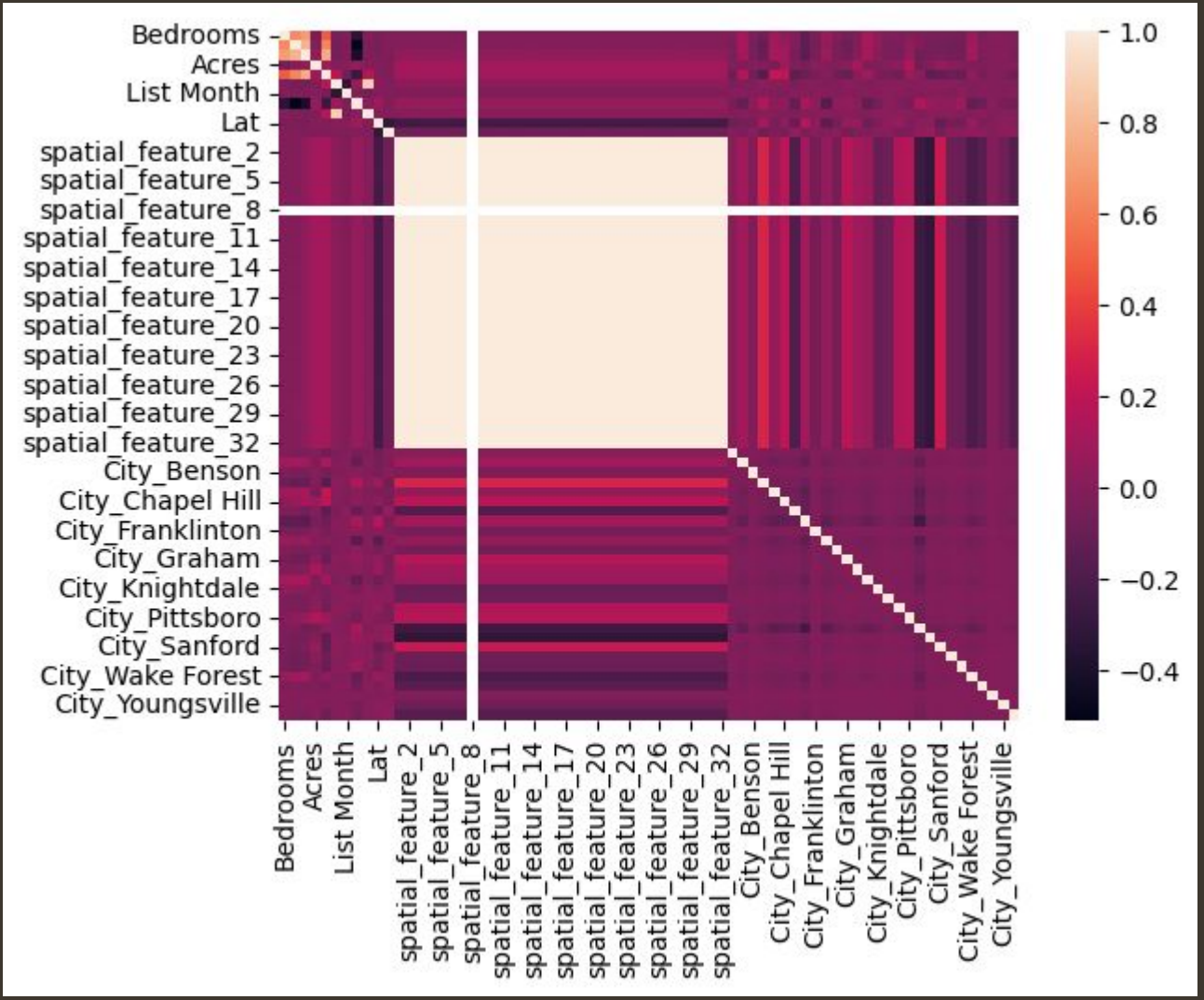
```
import gradio as gr
from sklearn.preprocessing import StandardScaler

def predict_price(Bedrooms, Total_Baths, SqFt):
    try:
        # Convert input features to a DataFrame
        input_df = pd.DataFrame([[Bedrooms, Total_Baths, SqFt]],
                                columns=['Bedrooms', 'Total Baths', 'SqFt'])
        scaler = StandardScaler()
        input_df['SqFt'] = scaler.fit_transform(input_df[['SqFt']])

        # Make prediction using the trained model
        prediction = bayes_search.predict(input_df)[0]

        print(f"Prediction: {prediction}")  # Log the prediction value
        return round(prediction)

    except Exception as e:
        print(f"Error during prediction: {e}")  # Log the error
        return "Error: Could not calculate prediction"  # Return a user-friendly message

# Create the Gradio interface
iface = gr.Interface(
    fn=predict_price,
    inputs=["number", "number", "number"],
    outputs="number",
    title="House Price Predictor"
)

iface.launch()
```





Regression: LightGBM Modules

Heatmap


Bedrooms
Acres
List Month
Lat
spatial_feature_2
spatial_feature_5
spatial_feature_8
spatial_feature_11
spatial_feature_14
spatial_feature_17
spatial_feature_20
spatial_feature_23
spatial_feature_26
spatial_feature_29
spatial_feature_32
City_Benson
City_Chapel Hill
City_Franklinton
City_Graham
City_Knightdale
City_Pittsboro
City_Sanford
City_Wake Forest
City_Youngsville
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

Hyperparameters

```
from skopt.space import Real, Integer, Categorical
# defining parameters
params = {

    'num_leaves': Integer(20, 50),
    'learning_rate': Real(0.01, 0.1, 'log-uniform'),
    'n_estimators': Integer(50, 500),
    'max_depth': Integer(3, 10),
    'metric': Categorical(['l1', 'l2', 'rmse']),
    'min_child_samples': Integer(10, 200),

}
```

```
# Define LightGBM model
import lightgbm as lgb
lgbm =
lgb.LGBMRegressor(objective='regres
sion')
```

```
X = sfr_df.drop(columns='Sold Price')
y = sfr_df['Sold Price']
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=8)


from skopt.space import Real,
Integer, Categorical
# defining parameters
params = {

    'num_leaves': Integer(20, 50),
    'learning_rate': Real(0.01,
0.1, 'log-uniform'),
    'n_estimators': Integer(50,
500),
    'max_depth': Integer(3, 10),
    'metric': Categorical(['l1',
'l2', 'rmse']),
    'min_child_samples':
Integer(10, 200),

}
```

```
#Train the LightGBM model
bayes_search.fit(X_train, y_train)
```

```
#Evaluate the model
best_model = bayes_search.best_estimator_
best_model
```

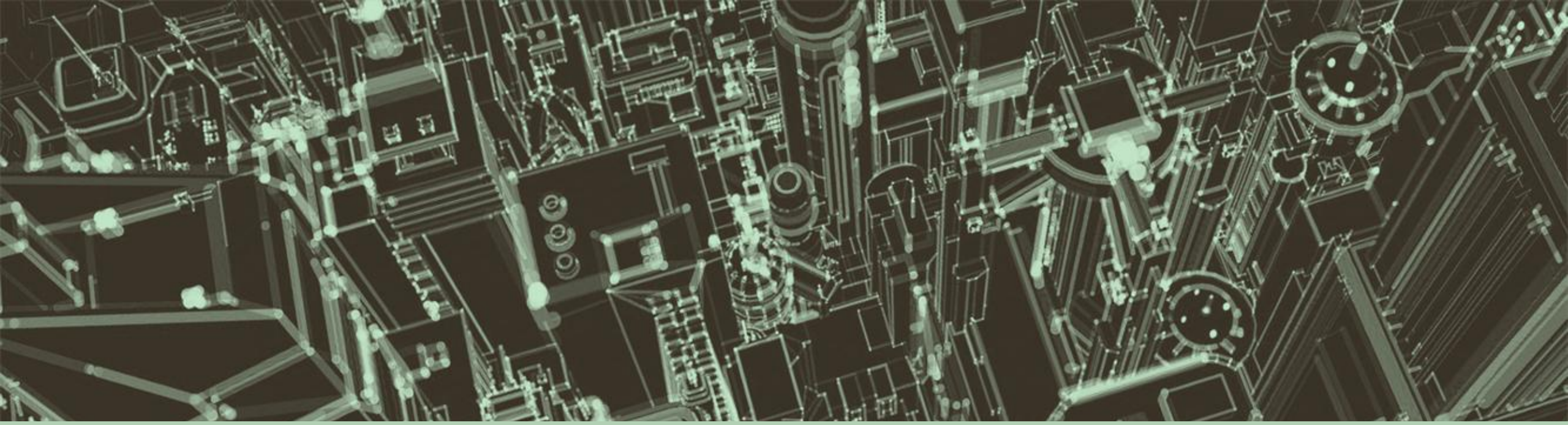
```
bayes_search.score(X_test, y_test)
```

```
-3571206253.8192706
```

```
# Make predictions with testing data
y_pred = best_model.predict(X_test)
```

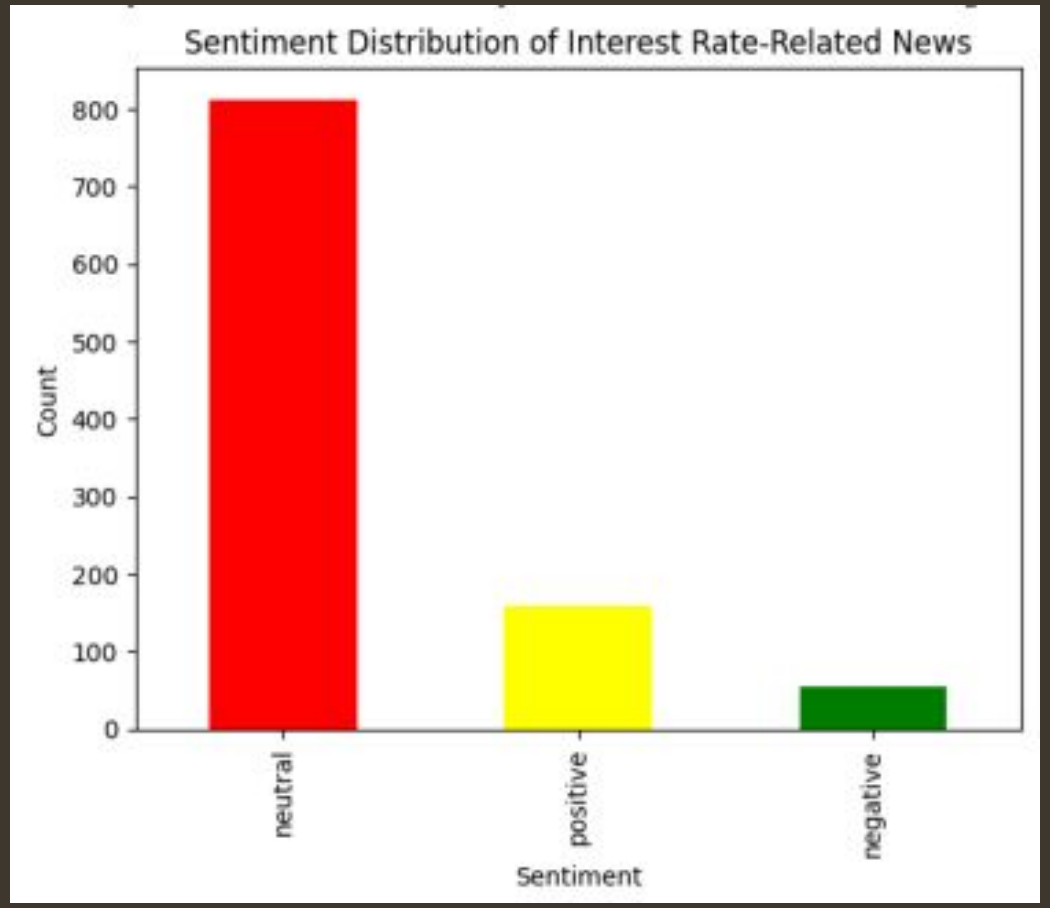
```
# Calculate MSE and RMSE, and MAE
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = mse**(0.5)
print("MSE: %.2f" % mse)
print("RMSE: %.2f" % rmse)
print("MAE: %.2f" % mae)
```

```
MSE: 3571206253.82
RMSE: 59759.57
MAE: 40862.68
```



NLP (BERT / VADER)

BERT / Pytorch

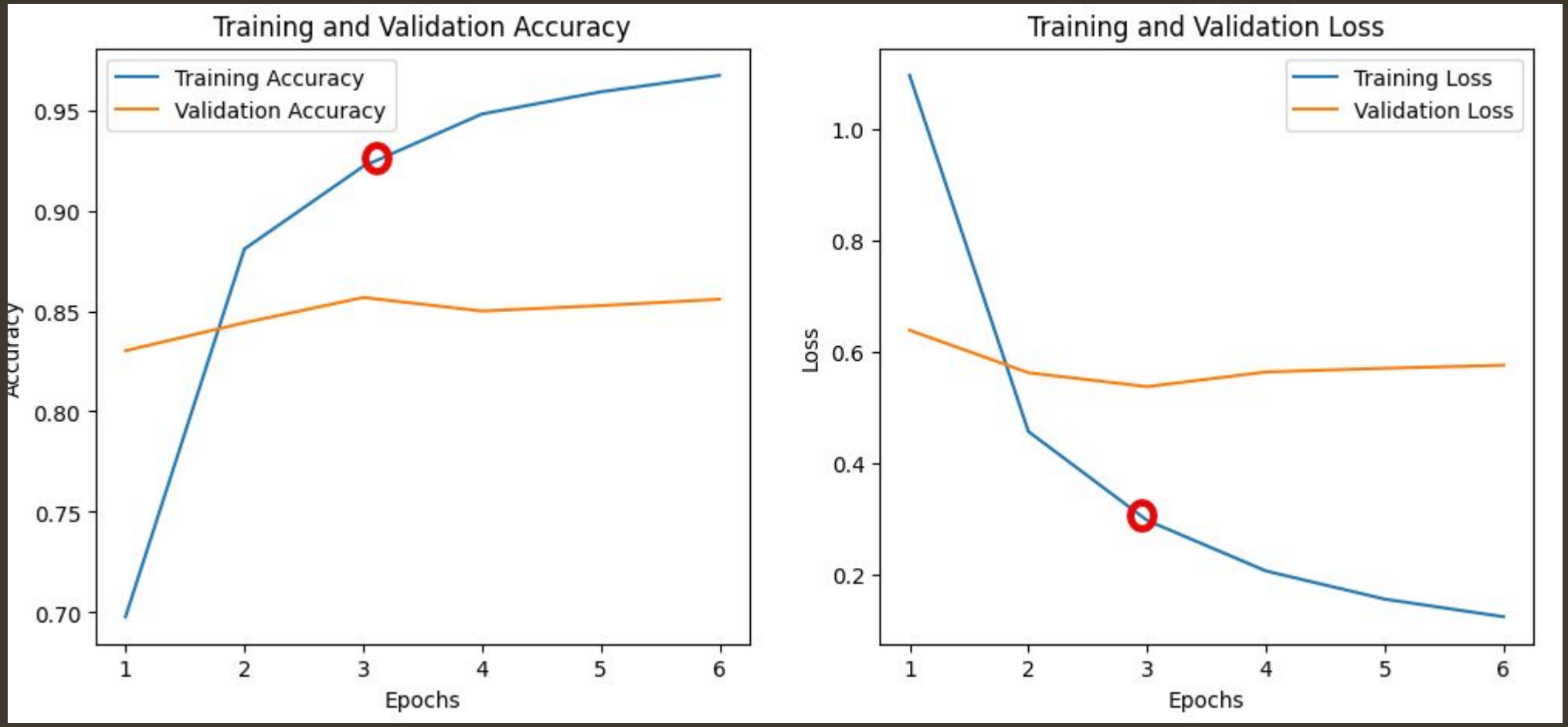


BERT / Pytorch

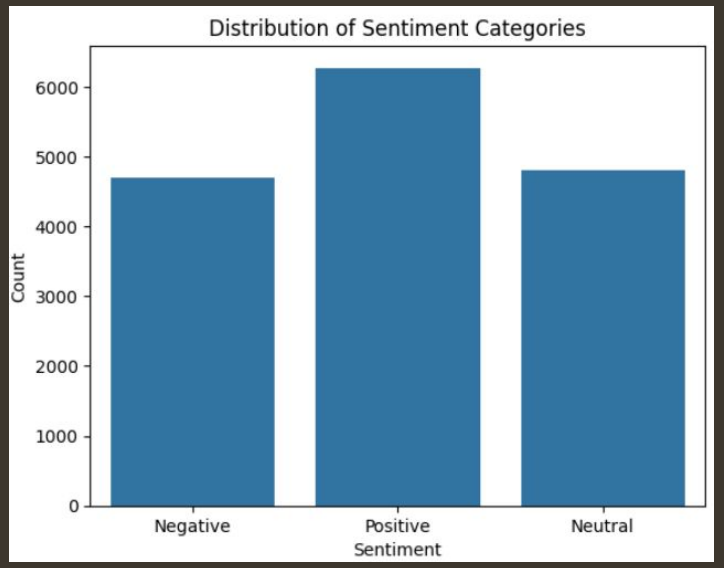
3 Epochs utilized to improve model accuracy, over fitting noted after Epoch 3.

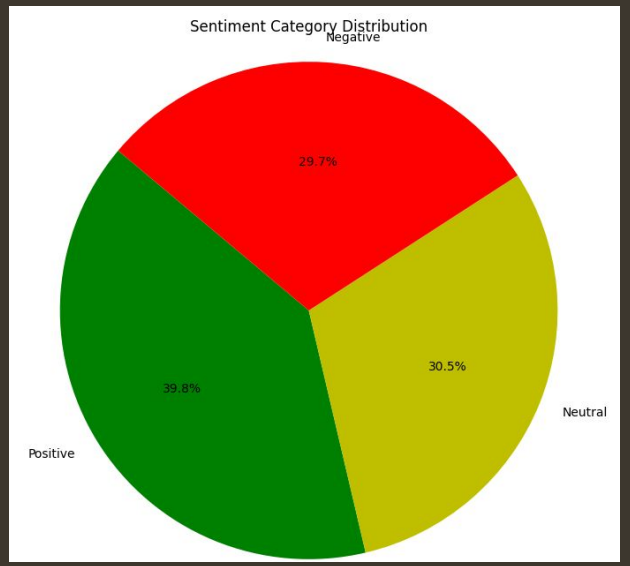
```
Epoch 1/3
Training: 100%|██████████| 557/557 [03:07<00:00,  2.96it/s]
Validation: 100%|██████████| 140/140 [00:16<00:00,  8.64it/s]
Train loss: 1.1368510144189188, Train accuracy: 0.6953932584269663
Validation loss: 0.6351048793643713, Validation accuracy: 0.8265947888589399
Epoch 2/3
Training: 100%|██████████| 557/557 [03:09<00:00,  2.94it/s]
Validation: 100%|██████████| 140/140 [00:16<00:00,  8.56it/s]
Train loss: 0.4487701477131788, Train accuracy: 0.8837078651685394
Validation loss: 0.5335162905177899, Validation accuracy: 0.8481581311769991
Epoch 3/3
Training: 100%|██████████| 557/557 [03:08<00:00,  2.95it/s]
Validation: 100%|██████████| 140/140 [00:16<00:00,  8.67it/s]
Train loss: 0.30344499430253, Train accuracy: 0.9256179775280899
Validation loss: 0.5258766201191715, Validation accuracy: 0.8562443845462714
```

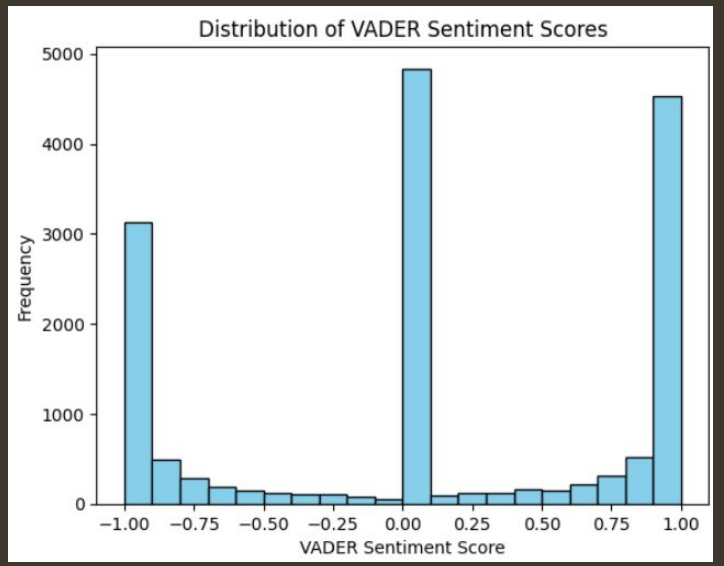
BERT / Pytorch



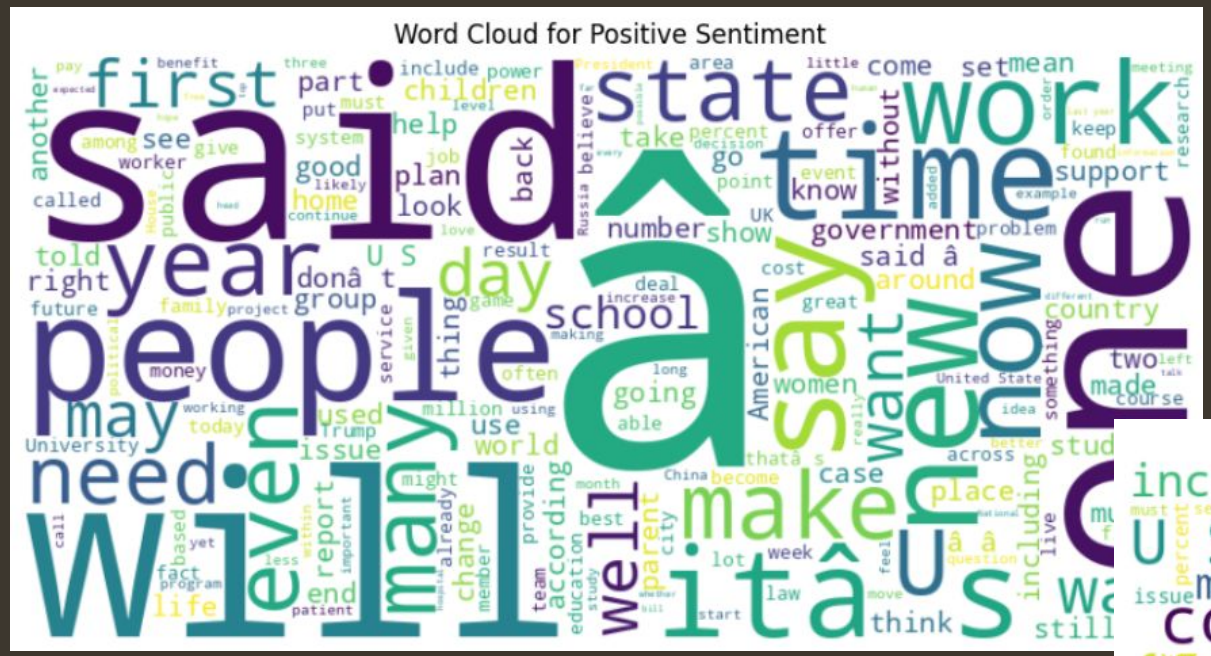
VADER - Sentiment Analysis







VADER - Word Cloud



Word clouds provide a broad overview of the most common terms associated with positive and negative sentiments, they also highlight the complexities and nuances of language that can cause certain words to appear in both categories, example, “said”, “people”, “state”.

Conclusions

	R2 Train	R2 Test	MSE Train	MSE Test	MAE Train	MAE Test
Random Forest	0.982	0.869	6.16E8	4.45E9	16505	44392
XGBoost		0.897	1.17E9	3.50E9	24463	40050
CatBoost	0.886	0.857	3.89E9	4.69E9	44004	47602
LightGBM		59760		3.57E9		40863

- XGBoost tends to perform best with the test data and overfits less than the Random Forest model, but still overfits significantly
- CatBoost seems to overfit the least, yet still produce a reasonable mean absolute error

Conclusions

- Sale prices for dataset range from 50k to 1.13 million
- Median home price for the data set is roughly 460k
- Based on these numbers, we have roughly an 8.6% error with XGBoost and 10.3% for CatBoost
- Most sites like Zillow and Redfin have an error of ~8%

```
df.describe()
```

✓ 0.8s

	Bedrooms	Total Baths	SqFt	Acres	Sold Price
count	53564.000000	53564.000000	5.356400e+04	5.356400e+04	5.356400e+04
mean	3.507001	2.837746	-7.853060e-17	5.478570e-17	4.635964e+05
std	0.770229	0.826589	1.000009e+00	1.000009e+00	1.847327e+05
min	2.000000	1.000000	-2.666322e+00	-1.609030e-01	5.000000e+04
25%	3.000000	2.000000	-7.888368e-01	-1.609030e-01	3.310000e+05
50%	3.000000	3.000000	-1.030035e-01	-1.609030e-01	4.295000e+05
75%	4.000000	3.000000	6.626959e-01	-1.609030e-01	5.700000e+05
max	5.000000	5.000000	3.359124e+00	1.710457e+01	1.131500e+06

NLP Conclusions

- BERT (Bidirectional Encoder Representations from Transformers) and VADER (Valence Aware Dictionary for Sentiment Reasoning) models indicate positive article sentiment.
- BERT model provide more accurate and nuanced results due to the complexity of financial language and the need to understand subtle sentiment variations whereas VADER works well on simple, short texts where sentiment is straightforward.
- BERT overfitting noted after Epoch 3, training accuracy of 92%
- Larger data set specific to keywords needed.

Improvements / Changes

- Include other spatial features like scalars and geohashes alongside existing features
- Find a way to improve overfitting on tree models
- Create pipeline to take in any given address via Gradio, scale data, and perform Lat/Lon conversion to predict home value
- Incorporate more features alongside NLP sentiment to predict future mortgage rates
- NLP sentiment on database
- Create UI, package these items together, alongside others, to sell to real estate investors as investment tool

Questions?

Thank You!