SmartPrice: Automated Housing Price Prediction System

Ben Thiele Jiaxuan Chen Nicole Young Qinglin Cui

December 12, 2024

Meet the Team

Qinglin Cui

Jiaxuan Chen

Ben Thiele

Nicole Young

Agenda

1. Business Problem
2. Exploratory Data Analysis
3. Model Pipeline
4. Monitoring
5. Conclusion

Business Problem

Automated Housing Price Prediction Model

- Predicts property prices using key attributes like size, rooms, and location.
- Fully automated: Updates and retrains with new data uploads.

Benefits for Real Estate Businesses:

- **Optimize Pricing:** Accurate, data-driven price recommendations for sellers.
- **Support Buyers:** Provide tailored property value estimates.
- **Market Insights:** Stay updated on trends and key pricing factors.

Dynamic, always up-to-date predictions ensure businesses stay competitive and responsive to market changes.

EDA - Data Overview

Dataset Summary:

- **Total Rows:** 21,613 (Each row represents a house sold in Washington State)
- **Missing Values:** None (0 missing values)

Features Used (11 out of 21 columns):

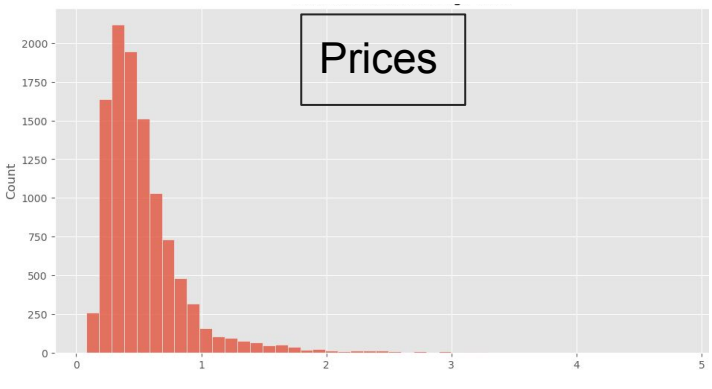
- **price** (Target variable)
- **bedrooms** (Number of bedrooms)
- **bathrooms** (Number of bathrooms)
- **sqft_living** (Square footage of the living area)
- **sqft_lot** (Square footage of the lot)
- **floors** (Number of floors)
- **condition** (Condition of the house)
- **grade** (Overall grade given to the house)
- **sqft_above** (Square footage of the house above ground level)
- **sqft_basement** (Square footage of the basement)
- **yr_built** (Year the house was built)

Preprocessing Steps:

- Selected **11 key columns** based on relevance to house pricing analysis.
- Ensured **data quality** with no missing values.
- Applied **data type conversions** to ensure consistency and transfer to feature store.

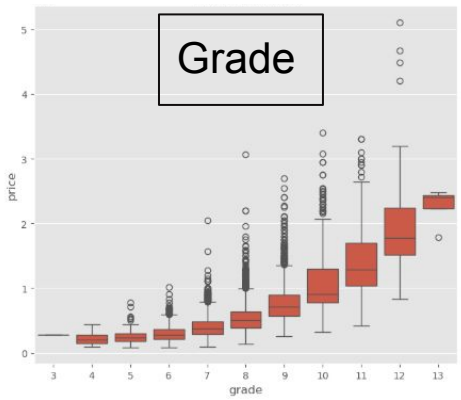
Exploratory Data Analysis: Key Distributions & Relationships

Right-skewed Prices & Attributes

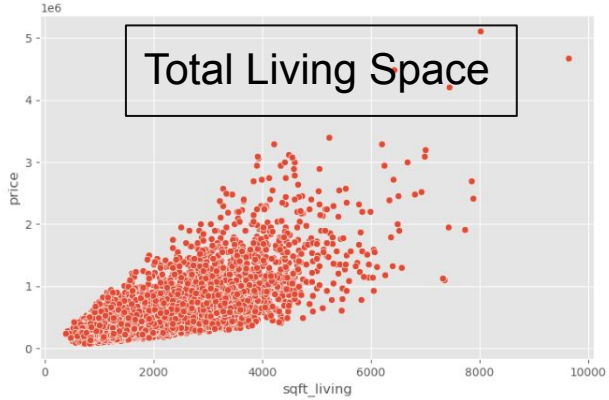


Most Correlated Features

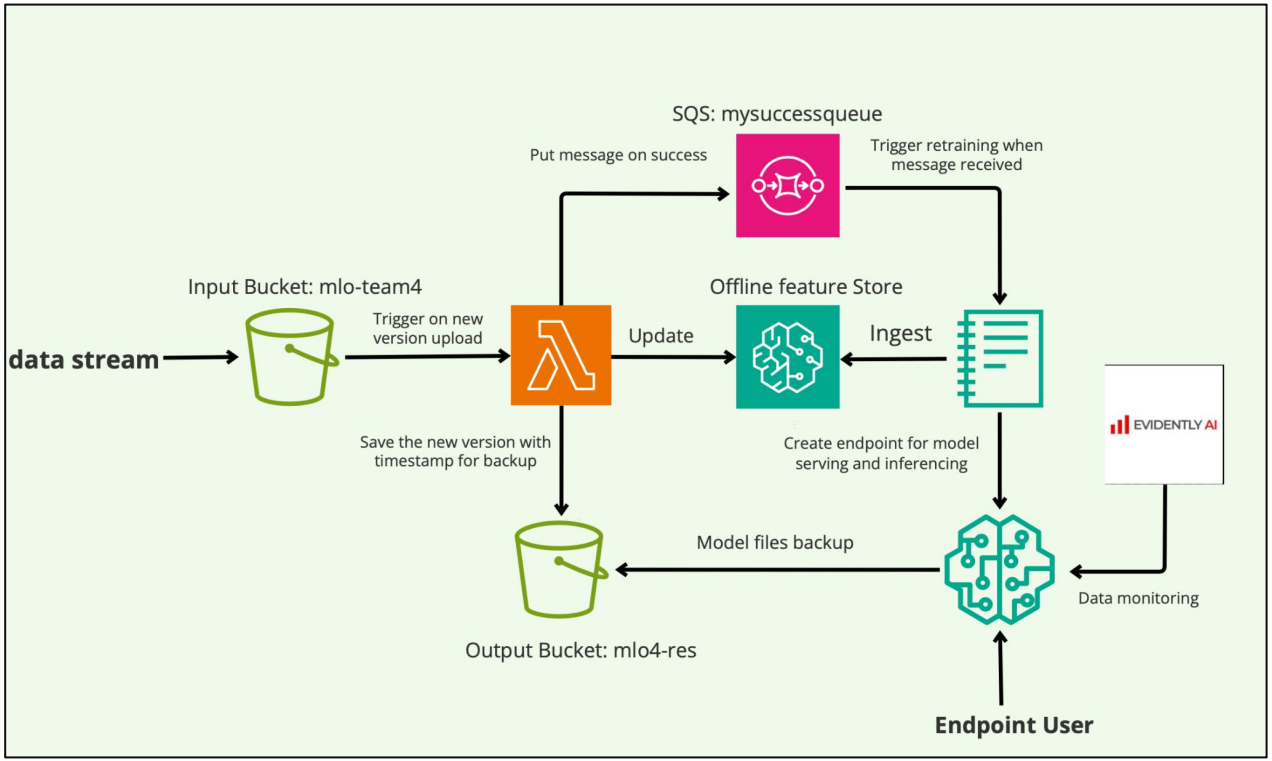
Pearson Correlation = 0.68



Pearson Correlation = 0.70

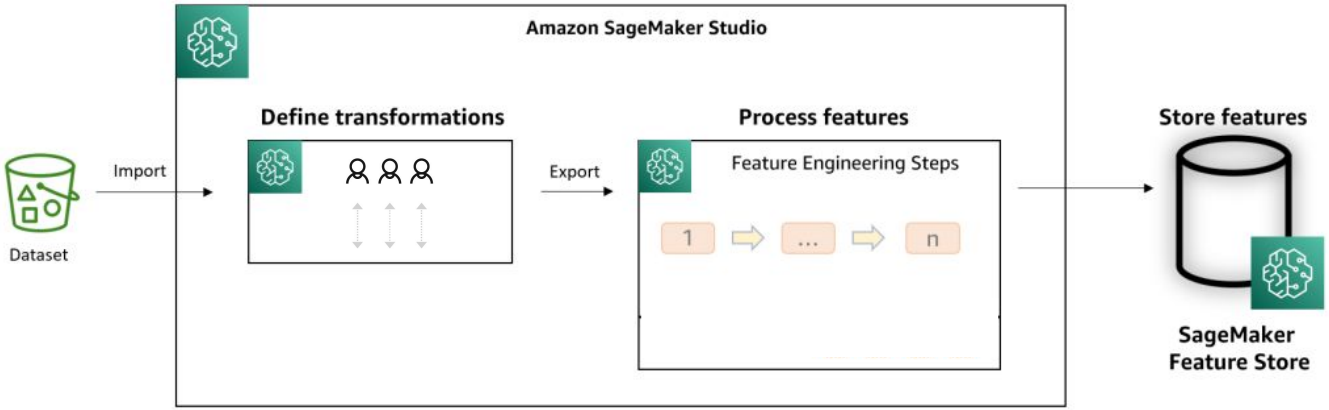


Pipeline


SQS: mysuccessqueue
Put message on success
Trigger retraining when message received
Input Bucket: mlo-team4
Offline feature Store
data stream
Trigger on new version upload
Update
Ingest
EVIDENTLY AI
Save the new version with timestamp for backup
Create endpoint for model serving and inferencing
Model files backup
Data monitoring
Output Bucket: mlo4-res
Endpoint User

Feature Store in AWS

A Feature Store centralizes and manages data features, making them consistent, reusable, and easy to access for both training and real-time predictions.



Feature Store in AWS:

Define the Training Feature Group

1. Create a feature group that will hold the training features.

```
# Define training feature group
feature_group_name_simulation = 'housing-feature-group-simulation'


# Define feature groups
feature_definitions_v1 = FeatureGroup(
    name=feature_group_name_1,
    sagemaker_session=sagemaker_session,
    feature_definitions=[
        FeatureDefinition(feature_name="number", feature_type=FeatureTypeEnum.STRING),
        FeatureDefinition(feature_name="event_time", feature_type=FeatureTypeEnum.STRING),
        FeatureDefinition(feature_name="date", feature_type=FeatureTypeEnum.FRACTIONAL),
        FeatureDefinition(feature_name="price", feature_type=FeatureTypeEnum.FRACTIONAL),
        FeatureDefinition(feature_name="bedrooms", feature_type=FeatureTypeEnum.FRACTIONAL),
        FeatureDefinition(feature_name="sqft_living", feature_type=FeatureTypeEnum.FRACTIONAL),
        FeatureDefinition(feature_name="sqft_lot", feature_type=FeatureTypeEnum.FRACTIONAL),
        FeatureDefinition(feature_name="floors", feature_type=FeatureTypeEnum.FRACTIONAL),
        FeatureDefinition(feature_name="condition", feature_type=FeatureTypeEnum.FRACTIONAL),
        FeatureDefinition(feature_name="grade", feature_type=FeatureTypeEnum.FRACTIONAL),
        FeatureDefinition(feature_name="sqft_above", feature_type=FeatureTypeEnum.FRACTIONAL),
        FeatureDefinition(feature_name="sqft_basement", feature_type=FeatureTypeEnum.FRACTIONAL),
        FeatureDefinition(feature_name="yr_built", feature_type=FeatureTypeEnum.FRACTIONAL)
    ]
)
```

```
feature_definitions_v1.create(
    record_identifier_name="number",
    event_time_feature_name="event_time",
    s3_uri=f's3://mlo-team4/features/',
    role_arn=role,
    enable_online_store=True
)
```

```
{'FeatureGroupArn': 'arn:aws:sagemaker:us-east-2:637423203755:feature-group/r',
 'ResponseMetadata': {'RequestId': '6a9e8cc2-3138-49fc-bd20-bf7238f9972b',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '6a9e8cc2-3138-49fc-bd20-bf7238f9972b',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '78',
   'date': 'Tue, 10 Dec 2024 21:20:36 GMT'},
  'RetryAttempts': 0}}
```

Feature Store in AWS:

Prepare Data for Feature Store Ingestion

2. Format and clean the data to match the feature group schema.

```
# Prepare data for feature store ingestion
features_required_new['number'] = features_required_new.index.astype('string')
features_required_new['event_time'] = pd.Timestamp.now().strftime('%Y-%m-%dT%H:%M:%SZ')

features_required_new['event_time'] = features_required_new['event_time'].astype('string')
```

```
# Select features for ingestion
features_required_new = features_required_new[['number', 'event_time', 'date', 'price', 'bedrooms',
                                               'bathrooms', 'sqft_living', 'sqft_lot', 'floors',
                                               'condition', 'grade', 'sqft_above', 'sqft_basement',
                                               'yr_built']]
```

Feature Store in AWS:

Check Feature Group Status

3. Verify that the feature group status is "Created."

```
feature_group_name ='housing-feature-group-simulation'
feature_group = FeatureGroup(name=feature_group_name, sagemaker_session=sagemaker_session)

status = feature_group.describe().get("FeatureGroupStatus")
print("Feature Group Status:", status)
```

```
Feature Group Status: Created
```

Feature Store in AWS:

Ingest Data

4. Load the prepared data into the feature store for future use.

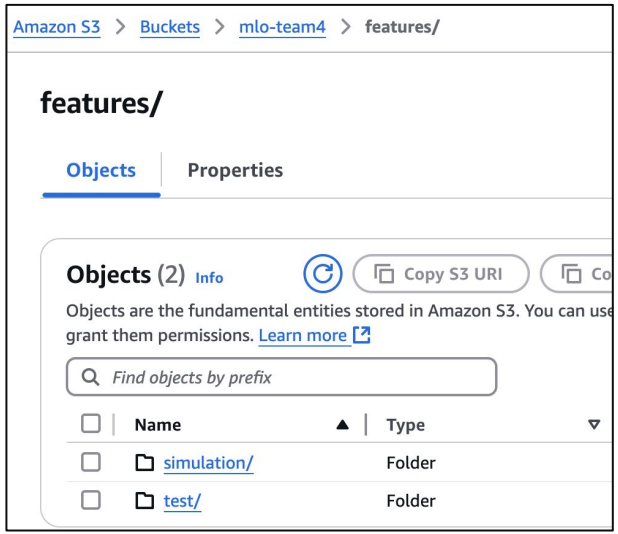
```
# Ingest data
feature_definitions_v1.ingest(data_frame=features_required_new, max_workers=3, wait=True)
```

```
IngestionManagerPandas(feature_group_name='t', feature_definitions={'number_id': {'FeatureName': 'number_id', 'FeatureTyp
e': 'String'}, 'event_time': {'FeatureName': 'event_time', 'FeatureType': 'String'}, 'date': {'FeatureName': 'date', 'Fea
tureType': 'Fractional'}, 'price': {'FeatureName': 'price', 'FeatureType': 'Fractional'}, 'bedrooms': {'FeatureName': 'be
drooms', 'FeatureType': 'Fractional'}, 'bathrooms': {'FeatureName': 'bathrooms', 'FeatureType': 'Fractional'}, 'sqft_livi
ng': {'FeatureName': 'sqft_living', 'FeatureType': 'Fractional'}, 'sqft_lot': {'FeatureName': 'sqft_lot', 'FeatureType':
'Fractional'}, 'floors': {'FeatureName': 'floors', 'FeatureType': 'Fractional'}, 'condition': {'FeatureName': 'conditio
n', 'FeatureType': 'Fractional'}, 'grade': {'FeatureName': 'grade', 'FeatureType': 'Fractional'}, 'sqft_above': {'Feature
Name': 'sqft_above', 'FeatureType': 'Fractional'}, 'sqft_basement': {'FeatureName': 'sqft_basement', 'FeatureType': 'Frac
tional'}, 'yr_built': {'FeatureName': 'yr_built', 'FeatureType': 'Fractional'}}, sagemaker_fs_runtime_client_config=<boto
core.config.Config object at 0x7fdff0474a90>, sagemaker_session=<sagemaker.session.Session object at 0x7fdff10857e0>, max
_workers=3, max_processes=1, profile_name=None, _async_result=<multiprocess.pool.MapResult object at 0x7fdff1244eb0>, _pr
ocessing_pool=<pool ProcessPool(ncpus=1)>, _failed_indices=[])
```

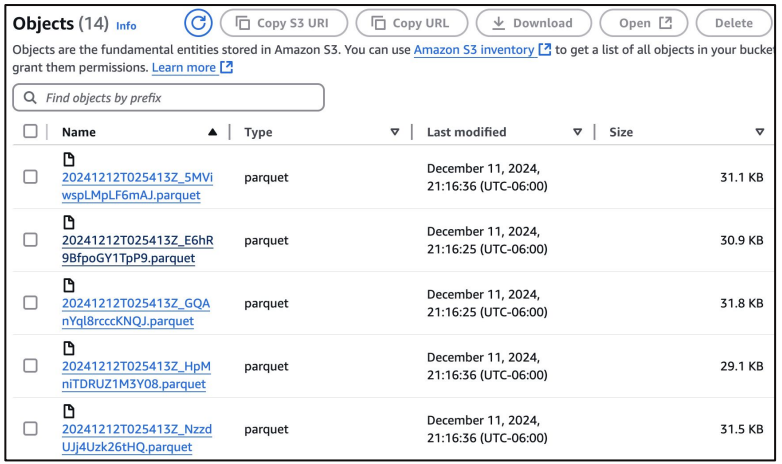
Feature Store in AWS:

Offline Feature Store

Data in the offline feature store is stored in Amazon S3 as Parquet files, offering efficient storage, compression, and fast access for large datasets.



2 feature stores



S3 PUT event works as a trigger for lambda to update the feature store

For lambda function:

- Trigger by creation event in /house_streaming_data subfolder (let's say a new version of data)
- Load the new data from S3 and insert into the feature store
- Save the version of the new data with timestamp and store into the result bucket for backup

S3 PUT event works as a trigger for lambda to update the feature store

```
# Fetch file content from S3
response = s3_client.get_object(Bucket=bucket_name, Key=object_key)
file_content = response['Body'].read().decode('utf-8')

# Add file to the processed set
processed_files.add(object_key)

# Convert CSV content to DataFrame
df = pd.read_csv(StringIO(file_content))
```

```
# Save the processed file to another S3 bucket
s3_client.put_object(
    Bucket='mlo4-res',  # Replace with your destination bucket
    Key=new_file_name,
    Body=file_content
)
print(f"File saved to mlo4-res bucket with name: {new_file_name}")
```

```
# Prepare data for ingestion
feature_group_name = 'housing-feature-group-simulation'  # Replace with your feature group
records = df.to_dict(orient='records')

# Ingest data into the feature store
for record in records:
    response = featurestore_client.put_record(
        FeatureGroupName=feature_group_name,
        Record=[{'FeatureName': k, 'ValueAsString': str(v)} for k, v in record.items()]
    )
    print(f"Record ingested: {response}")

print(f"Successfully ingested {len(df)} rows to the feature store.")
```

Sagemaker notebook polls message from the queue to retrain the model

```python
# Prepare data for ingestion
feature_group_name = 'housing-feature-group-simulation'  # Replace with your feature group
records = df.to_dict(orient='records')

# Ingest data into the feature store
for record in records:
    response = featurestore_client.put_record(
        FeatureGroupName=feature_group_name,
        Record=[{'FeatureName': k, 'ValueAsString': str(v)} for k, v in record.items()]
    )
    print(f"Record ingested: {response}")

print(f"Successfully ingested {len(df)} rows to the feature store.")
```

```python
sqs.delete_message(
    QueueUrl=queue_url,
    ReceiptHandle=message['ReceiptHandle']
)
```

Data Preparation:

- Convert the DataFrame to a dictionary for ingestion into the feature store.
- Data Ingestion:
- Use the featurestore_client.put_record() method to upload data to the specified feature group.
- Log success for each ingested record.

Message Queue Management:

- After successful ingestion, delete processed messages from the queue using sqs.delete_message().

Sagemaker notebook polls message from the queue to retrain the model

For SQS queue:

- Set Delivery delay = 5 min to accommodate delays on offline feature stores update
- Set invisibility period = 15 min to make sure the retrain attempts have enough time to complete

For Sagemaker Notebook (ML pipelines with H2O):

- Set message waiting = 20 sec for maximum message long-polling
- Retrain pipeline every time there is a message from the queue
- Ingest the feature from the feature store (with appended new data) to retrain

Predictive Model: Why H2O?

Best Performance in Under 30 Minutes

AutoML Tool	Best Model RMSE
H2O	192,453
AWS Autopilot	203,476

- Performs well in short time frames
- Easy integration with Evidently AI
- Granular control of selecting algorithms and tuning hyperparameters

Model Comparison and Best Model Update

Condition: The current model is compared to the existing best model using Root Mean Squared Error (RMSE).

- Update Trigger: The current model's RMSE must be lower than the existing best model's RMSE OR we manual update 0.

Steps for Updating Best Model:

- **Create a New Best Model Record:**
  - Store details such as model_id, rmse, and training_time_ms in a DataFrame.
  - Save this information to a CSV file: best_model_info.csv.
  - Upload the CSV to an S3 bucket.
- **Save the Model:**
  - The best model is saved locally using the h2o.save_model() method.
  - Specify the file path for saving.
- **Upload Model to S3:**
  - Upload the best model file (.zip) to an S3 bucket with a unique key.
- **Fallback Case:**
  - If the current model is not better, print: "Current model is not better than the existing best model."

Predictive Model: Best Model

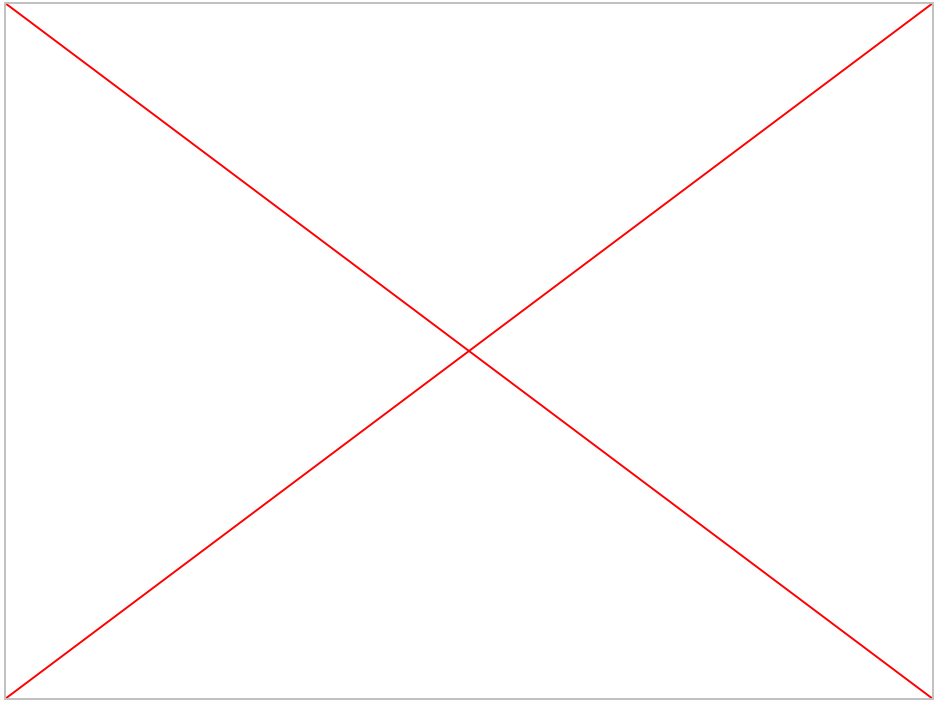
AutoML Leaderboard

Model Architecture	RMSE
Ensemble - Best of Family	192,453
Ensemble - All Models	192,728
GBM - Grid	194,536
GBM	199,513
XGBoost	200,288

- Stacked ensemble models, which have two levels (base and meta), top the leaderboard
- Stacked ensembles tend to reduce overfitting and improve generalization
- RMSE uses same units as target variable and penalizes large errors

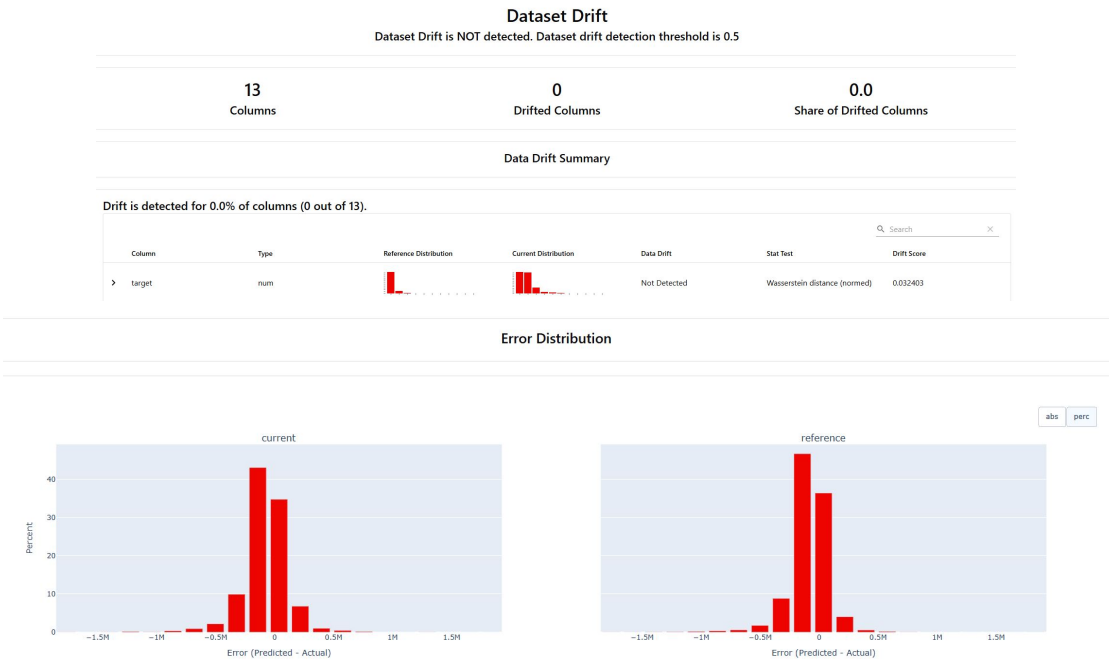
Monitoring for Data Quality, Drift, and Model Performance

EVIDENTLY AI



Monitoring for Data Quality, Drift, and Model Performance

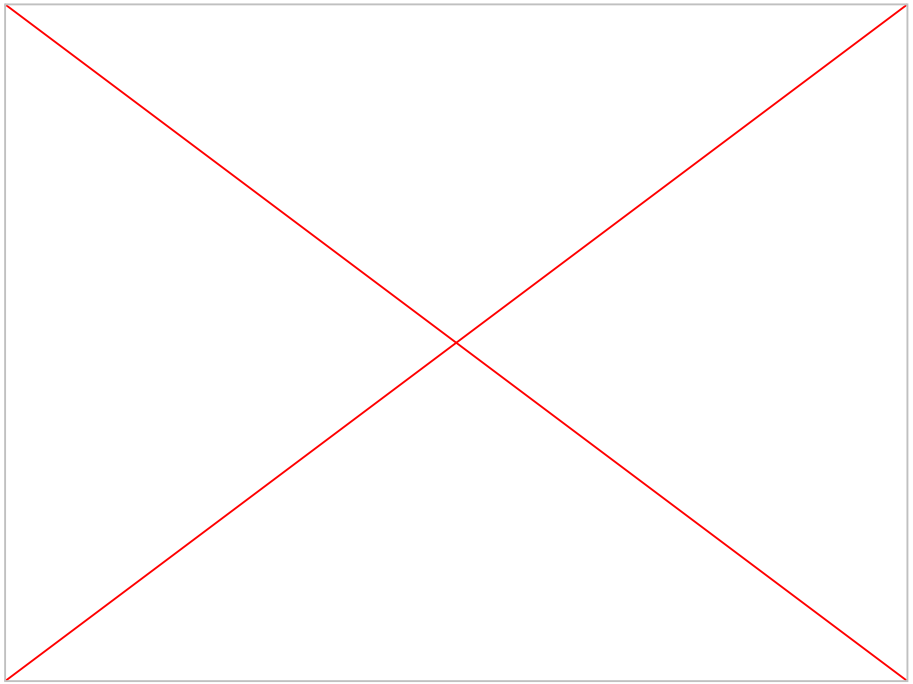
EVIDENTLY AI



- Faster, simpler setup than AWS monitoring
- Pre-built dashboards and visualizations that meet all our needs
- Flexibility to work with virtually all ML tools and frameworks

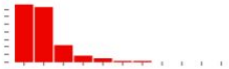
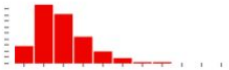
Monitoring for Data Poisoning (Video)

EVIDENTLY AI



Monitoring for Data Poisoning

EVIDENTLY AI

Column	Reference Distribution	Current Distribution	Data Drift	Drift Score
target			Not Detected	0.029904
prediction			Not Detected	0.037556
sqft_living			Detected	0.129463
sqft_lot			Not Detected	0.062713

Method of Poisoning: Swap values from Living Space Sq. Ft. & Lot Size Sq. Ft.

Expectation: Drift should be detected in both columns and predictions should be systematically biased toward overestimation.

Monitoring Results: Surprisingly only Living Space Sq. Ft. drift detected

Conclusion

What we did now?

- **Automatic Trigger:** The pipeline runs automatically when the CSV data is updated.
- **Feature Store:** All processed features are saved for easy reuse.
- **AutoML Model Comparison:** Automatically trains and selects the best model.
- **Monitoring:** Tracks model performance and sends alerts for retraining if needed.

What would we improve our model?

- **Advanced AutoML Comparison:** Compare models based on additional metrics like training time, MAE, and RMSE for better selection.
- **Online Feature Store:** Upgrade from offline to an online feature store for real-time data access and faster inference.
- **Improved Trigger System:** Add triggers from tools like Evidently.ai for automatic retraining, report generation, and model runs.
- **ML Endpoint Setup:** Deploy models to ML endpoints for real-time predictions and seamless integration with applications.

Thank you! Questions?

Appendix

Team Member Contributions

- Ben Thiele: Monitoring, Exploration of no-code AWS solutions
- Jiaxuan Chen: Pipeline including lambda + sqs as well as updating feature store
- Nicole Young: Initial Feature Engineering, AWS Feature Store
- Qinglin Cui: AutoML + deployment of the model to the AWS sagemaker model registry

Github: https://github.com/jchenhsch/seattle_house_predict

First output - train data 10,000 rows

model_id	rmse	mse	mae	rmsle	mean_residual_deviance
StackedEnsemble_AllModels_1_AutoML_2_20241212_50502	209910	4.40622e+10	132270	0.3123	4.40622e+10
StackedEnsemble_BestOfFamily_1_AutoML_2_20241212_50502	210767	4.44226e+10	132733	0.314223	4.44226e+10
DeepLearning_grid_2_AutoML_2_20241212_50502_model_1	215650	4.65051e+10	138419	0.327862	4.65051e+10
GBM_2_AutoML_2_20241212_50502	216362	4.68127e+10	133623	0.316034	4.68127e+10
DeepLearning_grid_3_AutoML_2_20241212_50502_model_1	216530	4.68853e+10	137927	0.324232	4.68853e+10
GBM_3_AutoML_2_20241212_50502	216969	4.70754e+10	133527	0.316078	4.70754e+10
GBM_4_AutoML_2_20241212_50502	217150	4.7154e+10	133895	0.316387	4.7154e+10
GBM_grid_1_AutoML_2_20241212_50502_model_2	219141	4.80227e+10	134531	0.316951	4.80227e+10

Second output - train data 14,000 rows

AutoML progress: | (done)| 100%%

model_id	rmse	mse	mae	rmsle	mean_residual_deviance
StackedEnsemble_AllModels_1_AutoML_21_20241212_90049	210161	4.41675e+10	131908	0.311866	4.41675e+10
StackedEnsemble_BestOfFamily_1_AutoML_21_20241212_90049	211124	4.45735e+10	132541	0.313891	4.45735e+10
DeepLearning_grid_1_AutoML_21_20241212_90049_model_1	215133	4.6282e+10	138632	0.332072	4.6282e+10
DeepLearning_grid_2_AutoML_21_20241212_90049_model_1	215993	4.66531e+10	137501	0.325041	4.66531e+10
GBM_2_AutoML_21_20241212_90049	216362	4.68127e+10	133623	0.316034	4.68127e+10
GBM_3_AutoML_21_20241212_90049	216969	4.70754e+10	133527	0.316078	4.70754e+10
GBM_4_AutoML_21_20241212_90049	217150	4.7154e+10	133895	0.316387	4.7154e+10
DeepLearning_grid_3_AutoML_21_20241212_90049_model_1	217971	4.75112e+10	138696	0.325545	4.75112e+10
GBM_grid_1_AutoML_21_20241212_90049_model_2	219141	4.80227e+10	134531	0.316951	4.80227e+10
DRF_1_AutoML_21_20241212_90049	220347	4.85527e+10	135366	0.319627	4.85527e+10

[22 rows x 6 columns]

Third output - train data 18,000 rows

AutoML progress: |  (done)| 100%

model_id	rmse	mse	mae	rmsle	mean_residual_deviance
StackedEnsemble_BestOfFamily_1_AutoML_24_20241212_183249	192453	3.7038e+10	120634	0.290552	3.7038e+10
StackedEnsemble_AllModels_1_AutoML_24_20241212_183249	192728	3.7144e+10	120484	0.290356	3.7144e+10
DRF_1_AutoML_24_20241212_183249	194536	3.78443e+10	119672	0.291258	3.78443e+10
GBM_4_AutoML_24_20241212_183249	199091	3.96371e+10	123478	0.298373	3.96371e+10
GBM_5_AutoML_24_20241212_183249	199513	3.98056e+10	125322	0.302547	3.98056e+10
GBM_3_AutoML_24_20241212_183249	200288	4.01153e+10	124701	0.300153	4.01153e+10
GBM_grid_1_AutoML_24_20241212_183249_model_2	200911	4.03652e+10	124099	0.298257	4.03652e+10
XRT_1_AutoML_24_20241212_183249	201018	4.04081e+10	123909	0.300156	4.04081e+10
GBM_2_AutoML_24_20241212_183249	201465	4.05883e+10	125690	0.302392	4.05883e+10
GBM_grid_1_AutoML_24_20241212_183249_model_1	203326	4.13415e+10	127577	0.305582	4.13415e+10

[22 rows x 6 columns]

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	129352.5615	41724331266.3280	203643.1126	0.6652	0.3114	0.2673	0.1100
lightgbm	Light Gradient Boosting Machine	127494.4554	42451049123.2469	204979.0231	0.6621	0.3054	0.2588	0.2160
rf	Random Forest Regressor	129293.0623	43340889068.0986	207531.4285	0.6525	0.3123	0.2638	0.3440
et	Extra Trees Regressor	132648.8291	44355985446.9110	210112.7689	0.6429	0.3208	0.2710	0.2240
xgboost	Extreme Gradient Boosting	131800.5672	44950232268.8000	211130.8219	0.6396	0.3161	0.2667	0.0410
lasso	Lasso Regression	141204.2700	47607891955.9353	217555.8083	0.6188	0.4006	0.2947	0.4430
ridge	Ridge Regression	141202.8627	47607802888.3907	217555.6170	0.6188	0.3957	0.2947	0.0070
br	Bayesian Ridge	141176.0193	47607228704.2572	217554.6187	0.6188	0.3904	0.2946	0.0090
lr	Linear Regression	141204.3696	47607892406.4992	217555.8081	0.6188	0.3986	0.2947	0.6720
llar	Lasso Least Angle Regression	141842.3935	48034826833.9866	218385.8559	0.6163	0.3804	0.2961	0.0090
lar	Least Angle Regression	147401.0761	49300783571.1038	221479.6390	0.6046	0.4096	0.3094	0.0090
en	Elastic Net	146596.2230	50735354482.8920	224678.7612	0.5936	0.3636	0.3053	0.0210

Compare code

```
if current_best_rmse < existing_best_model_df['rmse'].iloc[0] or existing_best_model_df['rmse'].iloc[0] == 0:

    new_best_model_df = pd.DataFrame({
        'model_id': [current_best_model_id],
        'rmse': [current_best_rmse],
        'training_time_ms': [current_best_training_time]
    })
-----
    new_best_model_df.to_csv('best_model_info.csv', index=False)
-----
    s3.upload_file('best_model_info.csv', bucket_name, s3_key)
-----
    best_model = aml.leader
    model_path = h2o.save_model(model=best_model, path="h2o_models/", force=True)
    print(f"Model saved to: {model_path}")
-----
    s3_model_key = "housing_automl/h2o_best_model.zip"
    s3.upload_file(Filename=model_path, Bucket=bucket_name, Key=s3_model_key)
    print(f"Model uploaded to s3://{bucket_name}/{s3_model_key}")
else:
    print("Current model is not better than the existing best model.")
```