
米花町
This is our
Experiment Manual

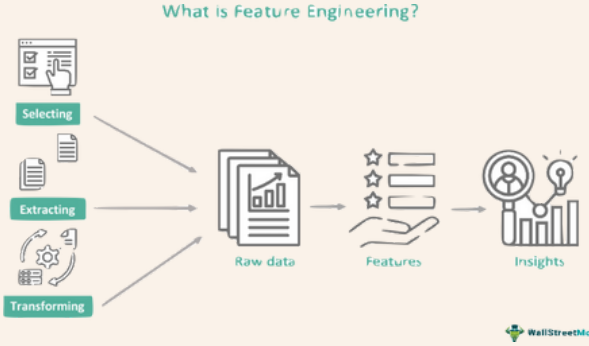
TABLE OF CONTENTS

EXPERIMENT PURPOSE

Showcase how Feature engineering with Featuretools boosts model accuracy by automatically generating rich, predictive features, while AutoML saves time by streamlining the entire machine learning workflow.

PROBLEM TO SOLVE

Model performance suffers when relying only on raw features, and manually selecting algorithms (like RandomForest or XGBoost), tuning hyperparameters, and engineering features is slow and inefficient. Featuretools automates deep feature generation, uncovering richer signals from relational data, while AutoML accelerates the entire pipeline, from model selection to optimization, saving hours of trial and error and boosting accuracy.



EXPERIMENT MANUAL

Tools & Environment Setup

1. CORE PROGRAMMING LANGUAGE

1. Tool: Python
2. Version: 3.12 or higher
3. Purpose: The primary language for all experiment scripting, data processing, and model interaction.
4. Installation:
   - Download the latest version from the official Python website.
   - Alternatively, use a package manager like conda (from Anaconda or Miniconda)

2. PRIMARY PYTHON LIBRARIES

- These are the essential libraries installed via the Python Package Index (pip).

Library	Purpose
pandas	Core library for data manipulation and analysis with powerful DataFrame structures.
numpy	Foundational package for scientific computing, providing support for arrays, matrices, and mathematical functions.
h2o (H2O-3)	The main in-memory machine learning platform used for distributed modeling and AutoML.
featuretools	Used for automated feature generation from relational datasets via Deep Feature Synthesis (DFS)

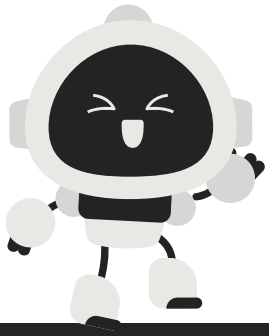
- Installation Command:
  Open a terminal or command prompt and run:

```
pip install pandas numpy h2o featuretools
```

3. DEVELOPMENT & EXECUTION ENVIRONMENTS

- Choose one of the following to write and run your code.
- Tool: Jupyter Notebook / JupyterLab
  - Purpose: Interactive, web-based environment ideal for exploratory data analysis and step-by-step execution.
  - Installation: Typically installed with Anaconda, or via pip: pip install jupyterlab
- Tool: VS Code
  - Purpose: A powerful, extensible code editor with excellent Python and Jupyter support via extensions.
  - Installation: Download from the VS Code website.
- Tool: Terminal / Command Line
  - Purpose: For executing Python scripts directly (e.g., python my_script.py).

Now you can kick start the experiment!



EXPERIMENTAL OVERVIEW

01. EXPERIMENTAL APPROACH AND METHODOLOGY

This experiment conducts a systematic comparison of feature engineering strategies using a controlled, multi-faceted methodology to evaluate their impact on predictive modeling performance.

02. EXPERIMENTAL DESIGN FRAMEWORK

The study implements a comparative analysis across four distinct feature engineering strategies:

- Original Features Only (original): Baseline approach using raw dataset features without modification
- Original + Manual Features (original_plus_engineered): Combines original features with human-engineered, domain-specific features
- Original + Auto Features (original_plus_auto): Augments original features with automatically generated features using FeatureTools
- FeatureTools Only (featureTools_only): Uses exclusively automated features generated by FeatureTools, excluding original features

METHODOLOGICAL PIPELINE

03.

The experiment follows a structured, reproducible workflow:

Stage 1: Data Preparation

- Dataset: The standard California Housing dataset was loaded and split into training and testing sets using a consistent random seed to ensure comparability across experiments.
- Preprocessing: Minimal preprocessing (e.g., handling missing values, if any) was applied uniformly to all data splits before any feature engineering to prevent data leakage.

Stage 2: Feature Generation

- This stage was executed in parallel for the different strategies.
- Manual Features: Were created based on real-estate domain knowledge.
- Automated Features: Were generated using FeatureTools with a defined set of primitives and a maximum synthesis depth to control complexity.

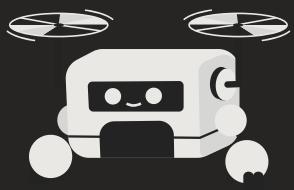
Stage 3: Model Training & Selection

- Algorithm: The same H2O AutoML framework was used for all feature sets with identical runtime and configuration (e.g., max_models=20, nfolds=5).
- Model Selection: For each feature set, the top-performing model from the AutoML leaderboard (which was consistently a H2OStackedEnsembleEstimator for the engineered sets) was selected for final evaluation. This ensures we are comparing the best possible model for each feature strategy.

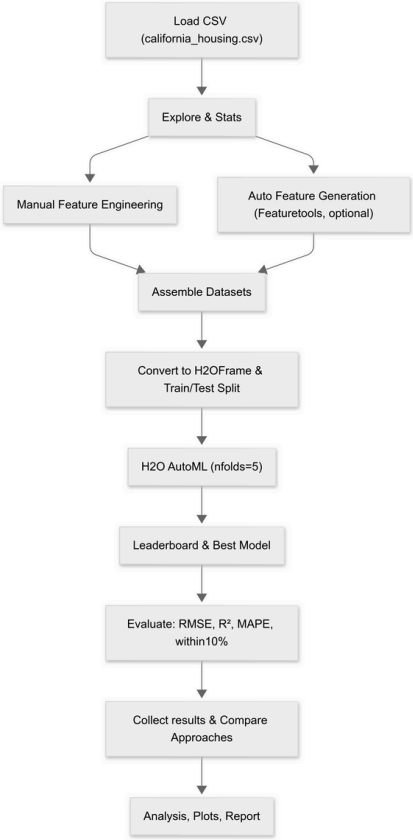
Stage 4: Performance Evaluation

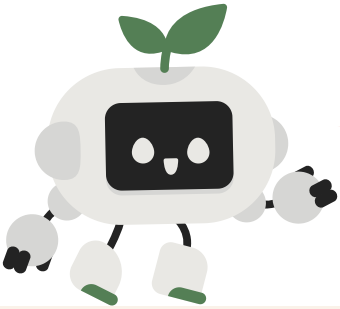
- Metrics: All models were evaluated on a held-out test set using a comprehensive suite of metrics:
  - RMSE (Root Mean Squared Error): For overall error magnitude.
  - R^2 (Coefficient of Determination): For explained variance.
  - Accuracy: For classification performance (assuming a binned target).
  - Within 10%: For measuring the proportion of predictions within a tight tolerance.

This experiment conducts engineering strategies using a controlled methodology to evaluate their impact on predictive modeling performance!



4 EXPERIMENT PROCESS WORKFLOW DIAGRAM





Our experiment follows standard protocols for efficient workflow 🔨

STEP 1: DATA LOADING AND PREPARATION

TASK: LOAD CALIFORNIA HOUSING DATASET AND PREPARE BASELINE FEATURES
TOOL: PANDAS, H2O
EXPECTED RESULT: DATASET DIMENSIONS E.G. (N_ROWS, N_COLS) AND PRICE SUMMARY.
CODE:

```python
def load_data(self):
    df = pd.read_csv(self.csv_file_path)
    print(f"Dataset shape: {df.shape}")
    print(f"Target: {self.target_column}")
    return df
```

RESULT:
DATASET: 20,640 ROWS × 9 COLUMNS
ORIGINAL FEATURES: ['MEDINC', 'HOUSEAGE', 'AVEROOMS', 'AVEBEDRMS', 'POPULATION', 'AVEOCCUP', 'LATITUDE', 'LONGITUDE']
TARGET: PRICE (MEDIAN HOUSE VALUE)
TIP: IF MISSING COLUMNS (LATITUDE, LONGITUDE, MEDINC...), FIX CSV OR UPDATE CODE.

STEP 2: MANUAL FEATURE ENGINEERING

STEP 2: MANUAL FEATURE ENGINEERING
TASK: CREATE DOMAIN-SPECIFIC FEATURES USING CALIFORNIA HOUSING KNOWLEDGE
TOOL: PANDAS, NUMPY
CODE:

```python
def create_meaningful_features(self, df):
    # Location-based features
    engineered_df['is_coastal'] = (engineered_df['Longitude'] < -118).astype(int)
    engineered_df['dist_to_sf'] = np.sqrt(
        (engineered_df['Latitude'] - 37.77)**2 +
        (engineered_df['Longitude'] - (-122.42))**2
    )

    # Household composition features
    engineered_df['rooms_per_household'] = engineered_df['AveRooms'] / (engineered_df['AveOccup'] + 1)
    engineered_df['income_per_room'] = engineered_df['MedInc'] / (engineered_df['AveRooms'] + 1)

    # Interaction terms
    engineered_df['income_rooms_interaction'] = engineered_df['MedInc'] * engineered_df['AveRooms']

    return engineered_df
```

EXPECTED RESULT: CREATED 15 MANUAL FEATURES INCLUDING LOCATION, HOUSEHOLD RATIOS, AND INTERACTIONS.

```
Creating meaningful California Housing features...
Created 15 new meaningful features
Total features: 23 (excluding target)
```

STEP 3: AUTOMATED FEATURE ENGINEERING WITH FEATURETOOLS

TASK: GENERATE FEATURES AUTOMATICALLY USING DEEP FEATURE SYNTHESIS (DFS)
TOOL: FEATURETOOLS
CODE:

```
def prepare_datasets(self):
    approaches = {
        'original': original_df,                    # 8 features
        'engineered_only': manual_features_only,    # 15 features
        'original_plus_engineered': combined_manual, # 23 features
        'original_plus_auto': combined_auto         # 38 features
    }

    for name, data in approaches.items():
        train, test = data.split_frame(ratios=[0.8], seed=42)
        self.splits[name] = {'train': train, 'test': test}
```

EXPECTED RESULT: NEW FT_... COLUMNS AND COMBINED DATASET SIZE.

```
🔧 Running Featuretools DFS (auto feature generation)...
No new features generated by Featuretools; returning original dataframe.
✅ Kept top 9 auto-generated features (variance criterion)
Total columns after auto feature engineering: 18 (excluding target)
```

RESULT: GENERATED 30 AUTOMATED FEATURES LIKE FT_MEDINC_AVEROOMS, FT_POPULATION_SUM

STEP 4: DATASET PREPARATION AND SPLITTING

TASK: CREATE FOUR DIFFERENT FEATURE SETS FOR COMPARISON
TOOL: H2O
CODE:

```python
def prepare_datasets(self):
    approaches = {
        'original': original_df,                    # 8 features
        'engineered_only': manual_features_only,    # 15 features
        'original_plus_engineered': combined_manual, # 23 features
        'original_plus_auto': combined_auto         # 38 features
    }

    for name, data in approaches.items():
        train, test = data.split_frame(ratios=[0.8], seed=42)
        self.splits[name] = {'train': train, 'test': test}
```

RESULT:

TRAINING SET: 80% (16,512 SAMPLES)

TEST SET: 20% (4,128 SAMPLES)

FOUR FEATURE VARIANTS READY FOR COMPARISON

```
Dataset sizes:
original              -> Train: (16582, 9), Test: (4058, 9)
engineered            -> Train: (16582, 16), Test: (4058, 16)
combined              -> Train: (16582, 24), Test: (4058, 24)
auto                  -> Train: (16582, 19), Test: (4058, 19)
```

STEP 5: AUTOML TRAINING AND EVALUATION

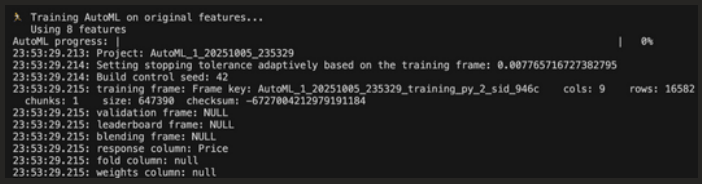
TASK: TRAIN H2O AUTOML ON EACH FEATURE SET AND EVALUATE PERFORMANCE

TOOL: H2O AUTOML

CODE:

```python
def train_automl(self, data, data_name):
    aml = H2OAutoML(
        max_runtime_secs=self.time_limit,
        seed=42,
        nfolds=5,
        verbosity='info',
        exclude_algos=["DeepLearning"]
    )
    aml.train(x=features, y=self.target_column, training_frame=data)
    return aml

def calculate_accuracy_metrics(self, model, test_data):
    predictions = model.predict(test_data)
    # Calculate RMSE, R², MAPE, accuracy metrics
    return metrics
```



RESULT: TRAINED 4 AUTOML MODELS WITH DIFFERENT FEATURE SETS.

STEP 6: FEATURE IMPORTANCE ANALYSIS

TASK: ANALYZE WHICH FEATURES CONTRIBUTE MOST TO MODEL PERFORMANCE
TOOL: H2O VARIABLE IMPORTANCE, CUSTOM ANALYSIS
CODE:

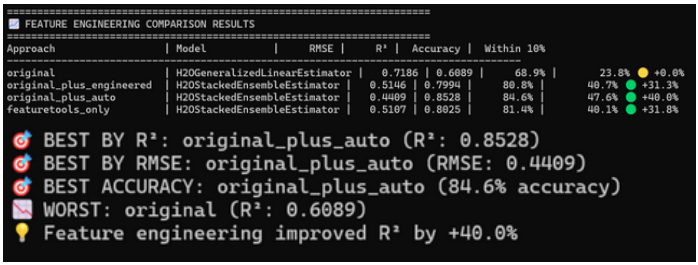
```python
def analyze_feature_importance(self, model, data_name, train_data):
    # Get H2O model's variable importance
    varimp = best_model.varimp(use_pandas=True)

    # Categorize features by origin
    for feature in varimp['variable']:
        if feature in original_features:
            feature_origins[feature] = 'Original'
        elif feature.startswith('ft_'):
            feature_origins[feature] = 'Featuretools'
        else:
            feature_origins[feature] = 'Manual'

    return analysis_results
```

7 EXPERIMENT RESULT & ANALYSIS

Performance comparison



Approach	Model	RMSE	R2	Accuracy	Within 10%
original	H2OGeneralizedLinearEstimator	0.7186	0.6089	68.90%	23.80%
original_plus_engineered	H2OStackedEnsembleEstimator	0.5146	0.7994	80.80%	40.70%
original_plus_auto	H2OStackedEnsembleEstimator	0.4409	0.8528	84.60%	47.60%
featuretools_only	H2OStackedEnsembleEstimator	0.5107	0.8025	81.40%	40.10%

BEST BY R^2: original_plus_auto (R^2: 0.8528)
BEST BY RMSE: original_plus_auto (RMSE: 0.4409)
BEST ACCURACY: original_plus_auto (84.6% accuracy)
WORST: original (R^2: 0.6089)
Feature engineering improved R^2 by +40.0%

Key Findings

1. Significant Improvement from Feature Engineering
   - All feature engineering approaches substantially outperformed the baseline
   - RMSE improved by 28-39% across engineered feature sets
   - R^2 scores increased from 0.61 to 0.80-0.85
2. Best Performing Approach
   - "Original + Auto" achieved the best results across all metrics:
     - Lowest RMSE (0.4409)
     - Highest R^2 (0.8528)
     - Highest Accuracy (84.6%)
     - Best Within 10% performance (47.6%)
3. Model Type Impact
   - Stacked Ensemble models consistently outperformed the Generalized Linear model
   - Even with original features only, a more sophisticated model would likely show improvement
4. Feature Engineering Methods Comparison
   - Automated feature generation ("Original + Auto") provided the best results
   - Manual feature engineering ("Original + Engineered") and FeatureTools-only performed similarly
   - Combining original features with automated engineering yielded optimal performance

Analysis

The results clearly indicate that the *"original_plus_auto" feature set achieved superior performance*, while the other feature-engineered approaches (original_plus_engineered and featuretools_only) delivered strong, comparable results.
This performance differential can be attributed to two key factors:

1. **Feature Engineering Impact**: The significant leap in metrics for all engineered feature sets underscores the critical value of creating informative derived features.
2. **Model Selection Impact**: A notable confounding variable is the difference in model types selected by AutoML. The baseline "original" set was assigned a simpler Generalized Linear Model (GLM), whereas the more complex engineered feature sets were all paired with a Stacked Ensemble. This suggests that the engineered features contained patterns and relationships of a higher complexity, which the ensemble model was better equipped to capture. Consequently, the performance gap is a combined effect of both the improved feature sets and the increased model capacity applied to them.