Featuretools

AUTOMATED FEATURE ENGINEERING (FEATURETOOLS) AND AUTOML

古莲玉 1820232022, 黄兰珊 1820232052, 李玫珊 1820232030, 林芷颖 1820232074, 赖整义 1820232024

小龙 1820232046, 黄璐瑶1120233470, 穆子雯 1120230627, 魏晓彤 1820232029, 李恩心 1820232039

WHAT IS AUTOML?

AutoML automates the process of **building** and **tuning** machine learning models. It’s like the autopilot of data science, saving time and effort

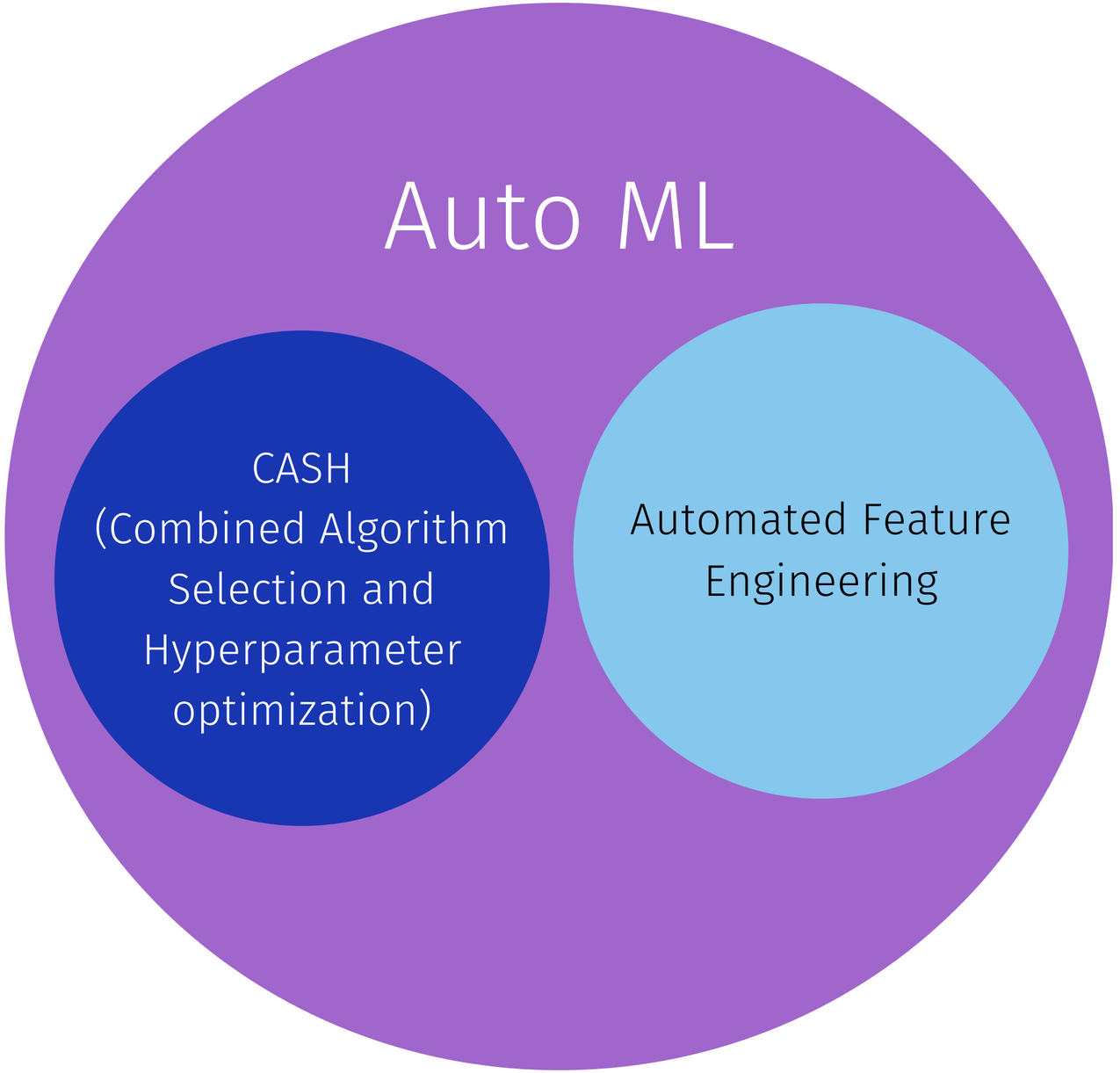
AutoML Concept

- **CASH: Formal optimization problem**

  Goal: choose the best combination of algorithm + hyperparameters

- **AutoML: CASH + everything a good analyst would provide**

  Goal: preprocessing, **automated feature engineering**, model selection & hyperparameter tuning(CASH), evaluation, interpretability, and deployment

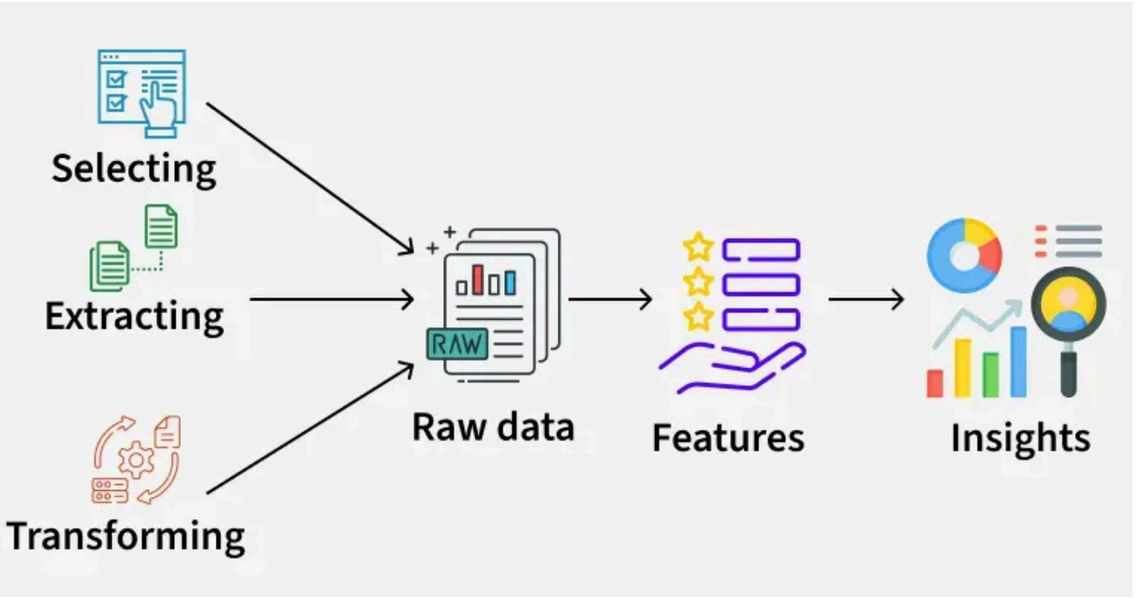


What is AFE?

(Automated Feature Engineering)

Feature Engineering: creates and transforms input variable that helps the model to learn better

- “Date of Birth” → “Age”
- “Height” & “Weight” → “BMI”



Automated Feature Engineering (AFE) uses algorithms to do this automatically, discovering new, smarter features that boost model performance.

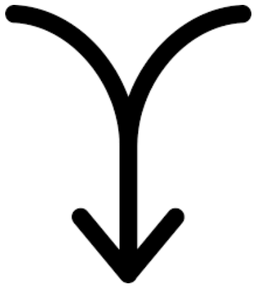
The Role of Automated Feature Engineering (AFE)

Customers

customer_ID	region

Orders

customer_ID	order_amount	date



AFE can automatically create features like:

Avg_cust_order	Total_ord_month	Time_sice_last_order

Tools

Automated Feature Engineering

AutoML

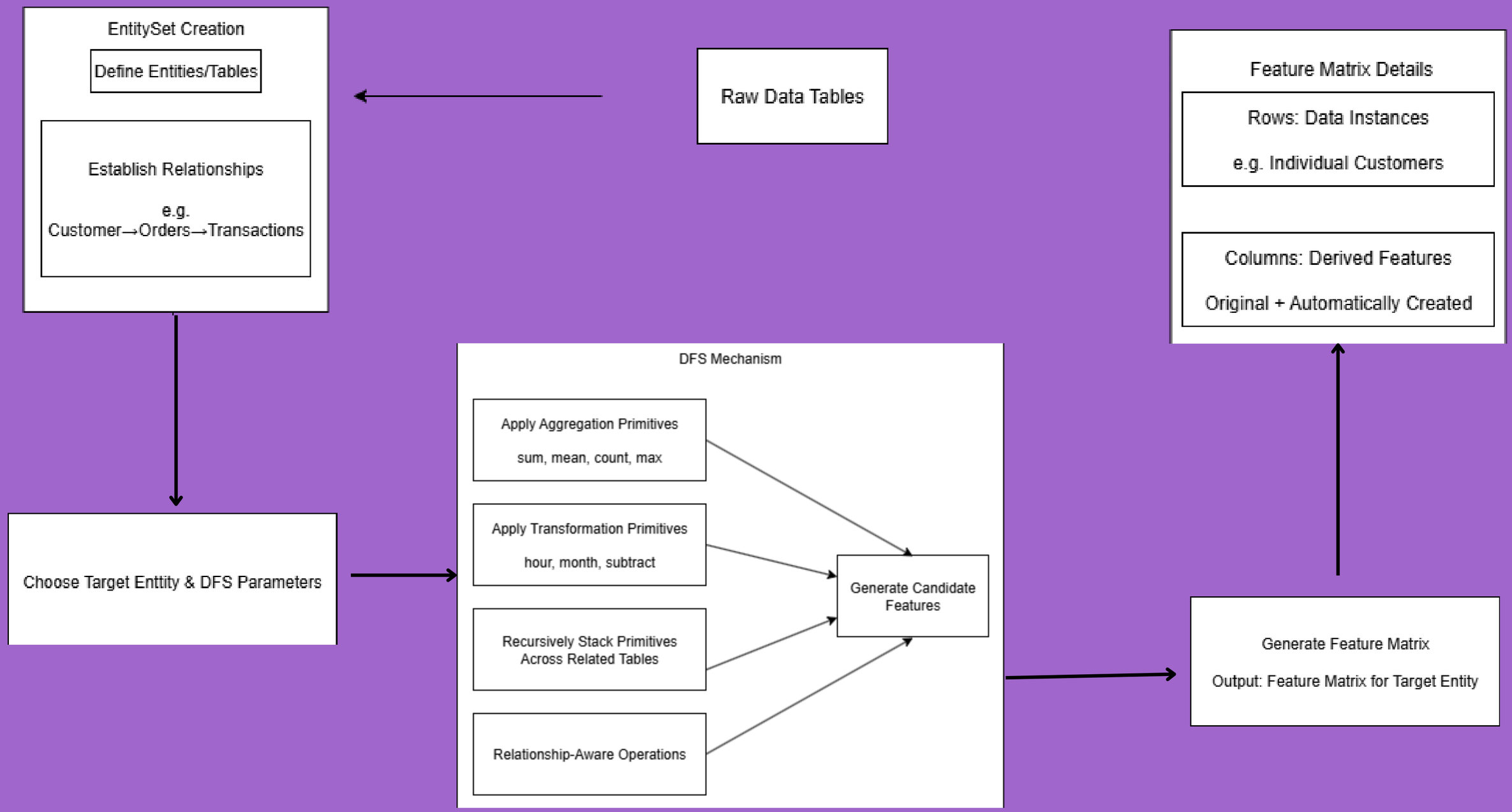
AutoML vs Feature Tools

	AutoML	Feature Tools
Purpose	Automates entire ML pipeline from preprocessing to tuning	Focuses on AFE, turns raw data into features
How it works	Uses optimization and ensembling to tune models	Uses DFS for aggregation and transformation
Input	Works on entire dataset to pick models and pipelines	Generates new features from tables and relations
Output	Trained ML model	Feature matrix with auto-derived feature
Strengths	Reduces the need for human model tuning	Creates complex relational features
Limitations	could miss domain-specific ideas	Create less interpretable features

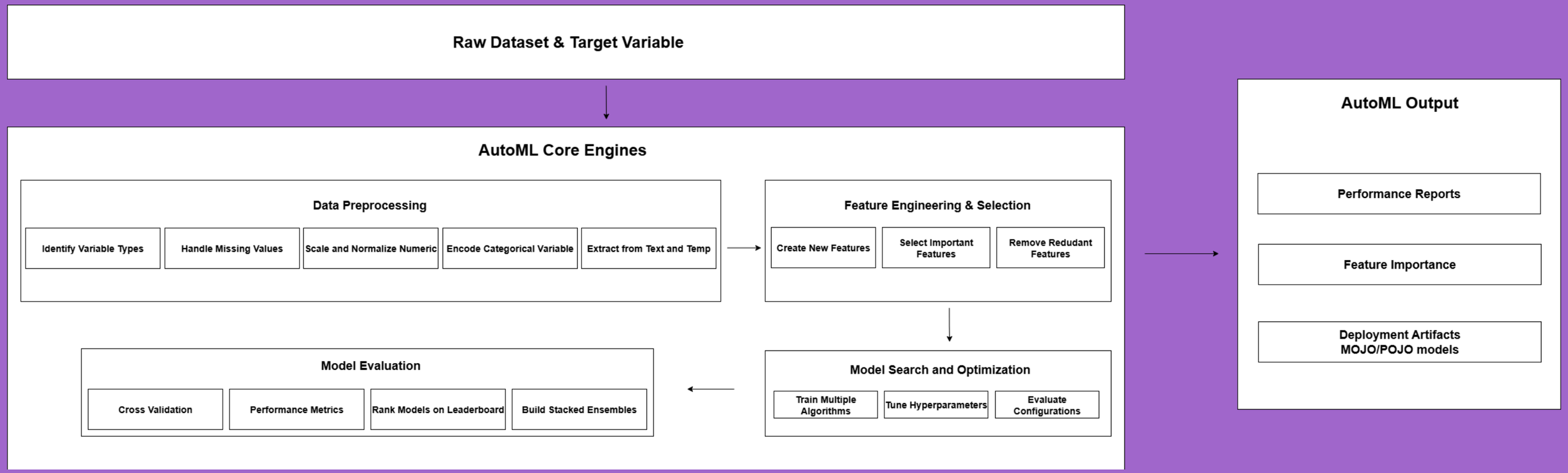
Traditional ML Workflow vs AutoML (H2O and Featuretools)

	Traditional ML	AutoML (e.g. H2O AutoML + Featuretools)
Problem Framing	Define task/target	
Data Ingestion & Preprocessing	Manual type detection, imputation, scaling, encoding	Automated type inference, imputation, scaling/encoding; robust handling of missing values
Feature Engineering	Manual domain crafting & interaction terms	Automated FE (e.g., Featuretools DFS) + optional manual tweaks
Feature Selection	Manual correlation checks/ model-based filtering	Automated filtering/embedded selection; redundancy pruning
Model Selection	Try a few families based on intuition	CASH search across many algorithms (GLM, RF, GBM, XGBoost, DL, etc.)
Hyperparameter Tuning	Manual grid/random; limited budget	Automated, guided search (CASH) with budget control
Evaluation	Manual CV/holdout; custom metrics	Automated CV/holdout; leaderboard with standardized metrics
Ensembling	Often manual or skipped	Automatic stacked ensembles of top models
Explainability	Ad-hoc plots/ feature importance	Automated feature importance & per-model reports
Deployment	Custom export/ packaging	Export deployment artifacts (e.g., MOJO/POJO)

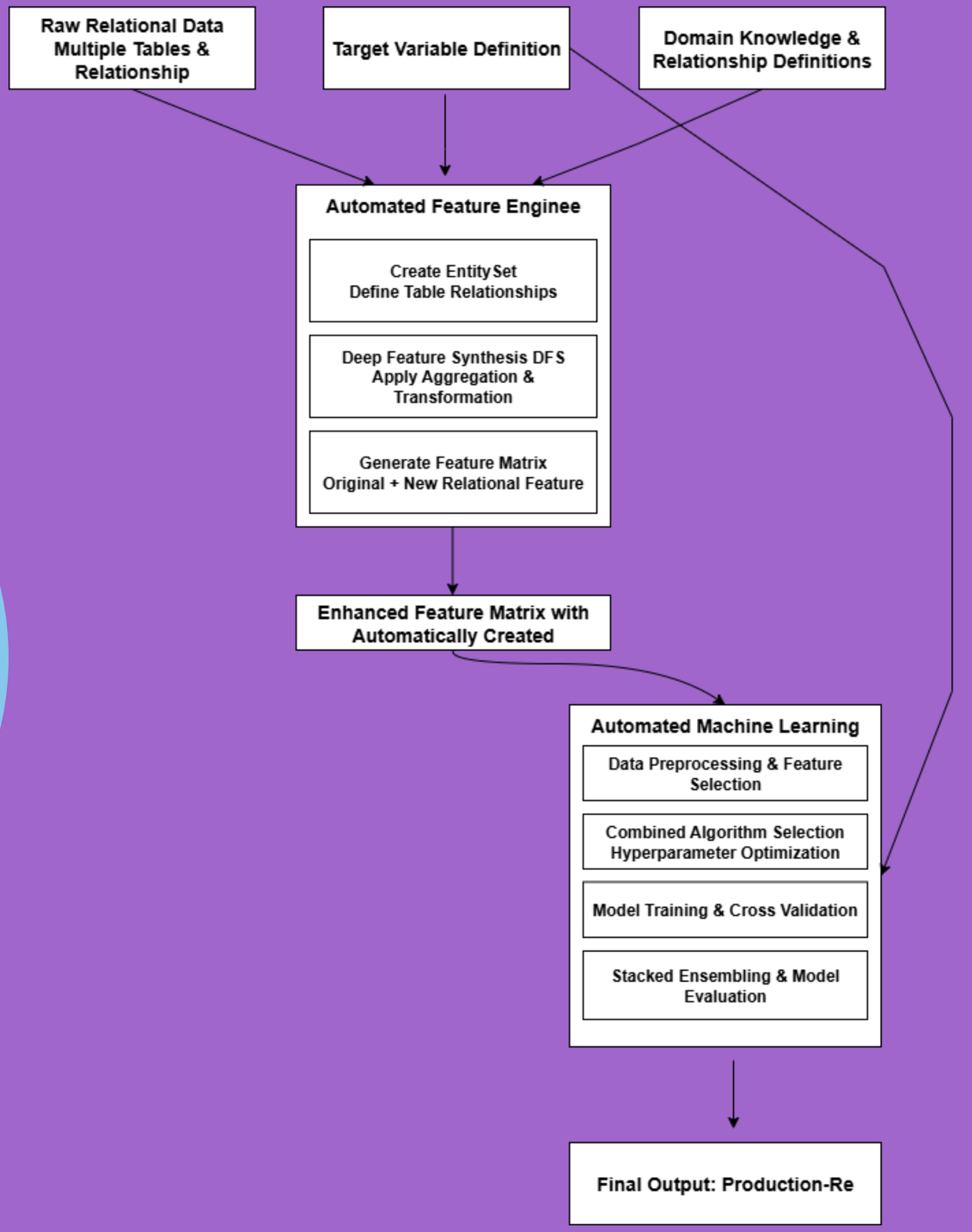
How does Feature Tools Works?



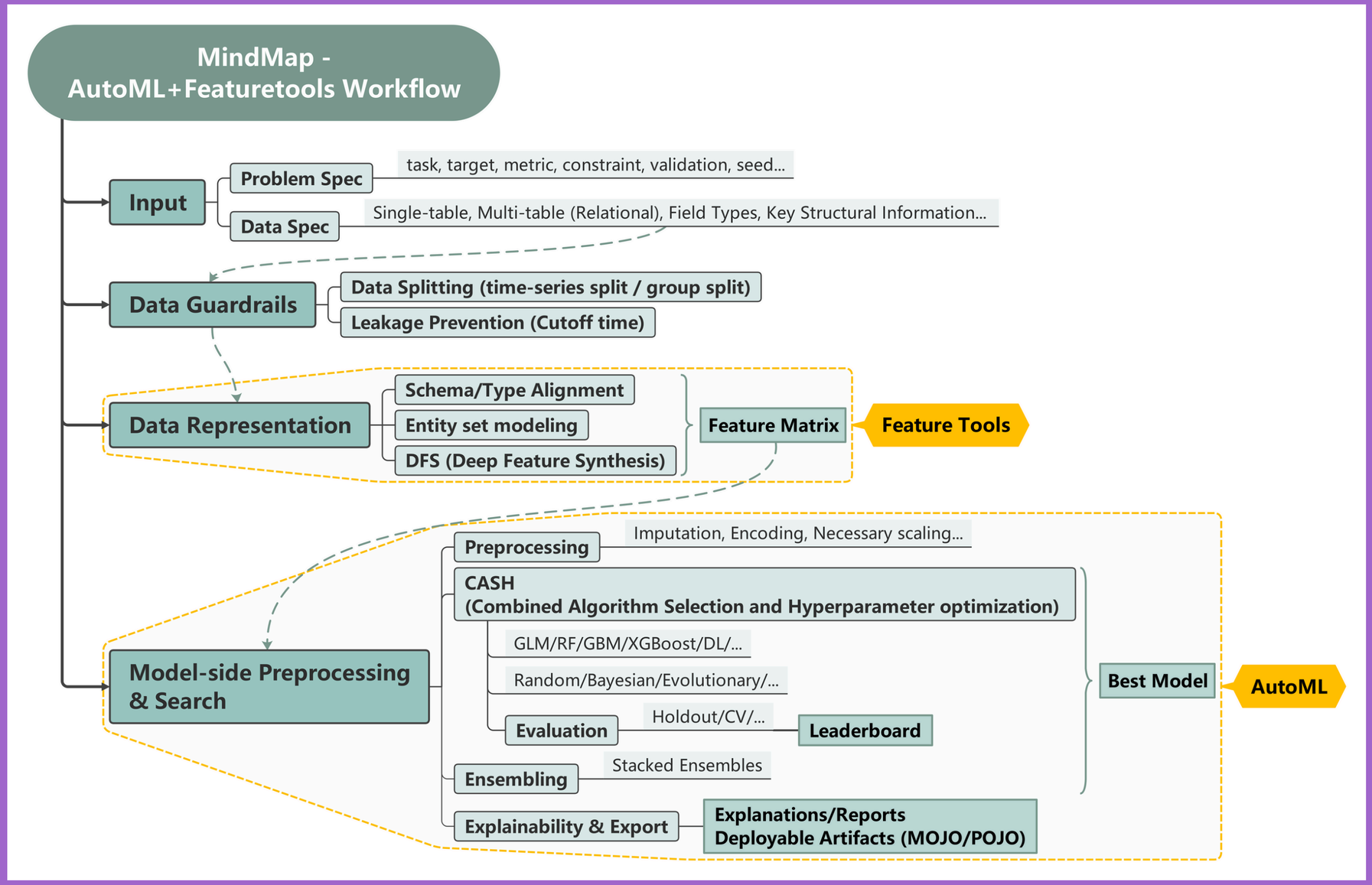
How does AutoML Works?



How does AutoML and featuretools work together?



Mind Map


MindMap -
AutoML+Featuretools Workflow
Input
Problem Spec
task, target, metric, constraint, validation, seed...
Data Spec
Single-table, Multi-table (Relational), Field Types, Key Structural Information...
Data Guardrails
Data Splitting (time-series split / group split)
Leakage Prevention (Cutoff time)
Data Representation
Schema/Type Alignment
Entity set modeling
DFS (Deep Feature Synthesis)
Feature Matrix
Feature Tools
Model-side Preprocessing & Search
Preprocessing
Imputation, Encoding, Necessary scaling...
CASH
(Combined Algorithm Selection and Hyperparameter optimization)
GLM/RF/GBM/XGBoost/DL/...
Random/Bayesian/Evolutionary/...
Evaluation
Holdout/CV/...
Leaderboard
Ensembling
Stacked Ensembles
Explainability & Export
Explanations/Reports
Deployable Artifacts (MOJO/POJO)
Best Model
AutoML

Case Scenario

House Price Prediction

Problem:
A real estate company wants to predict the market price of a house

Data Set

- MedInc
- HouseAge
- Longitude
- Latitude
- AveRooms
- AveBedrms
- Population
- AveOccup

Featuretools

- **Input:** California housing dataset
- **Process:** Apply Deep Feature Synthesis (DFS) to generate new features
- **Output:** A feature matrix with original + synthesized features.

AutoML

- **Input:** Feature matrix from Featuretools
- **Process:** Preprocessing, Feature selection, CASH optimization, Performance estimation
- **Output:** Best predictive model for house price in California

Experiment

Problem to Solve

Manual feature engineering and tuning are slow—Featuretools and AutoML automate them to save time and boost accuracy.

Experiment Purpose

Demonstrate how Featuretools enhances model accuracy through automated feature creation, while AutoML speeds up the entire ML workflow.

Experiment Hypothesis and Success Criteria

Hypothesis:

Automated + original features will slightly improve performance over original only; manual-only engineered features may or may not help.

Success criteria:

A measurable improvement in regression metrics (R^2 increase and RMSE decrease) that is consistent across multiple runs / seeds.

Experiment Manual

- **Core Programming Language**

  Python Version 3.12 or higher

- **Primary Python Libraries**

  pandas, numpy, h2o (H2O-3), featuretools

- **Development & Execution Environments**

  Jupyter Notebook / JupyterLab or VS Code and Terminal / Command Line

Experiment Overview

Approach and Methodology

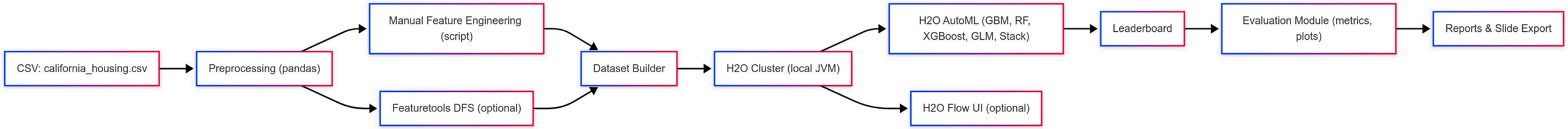
- Evaluate raw vs. engineered features under identical conditions to measure performance gains.
- Controlled, multi-step workflow to ensure fair, reliable results.

Design Framework

- Original : raw dataset features only
- Original + Manual : Add human-engineered, domain specific features
- Original + Auto : Add automatically generated features
- FeatureTools Only : Use only auto-geerated feature

Experiment Overview

Methodological Pipeline



1 2 3 4

Step 1- Data Preparation

Minimal preprocessing to prevent data leakage

Step 2- Feature Generation

- Manual : Based on domain knowledge
- Automated : Generated via FeatureTools

Step 3- Model Training and Selection

- Framework : H2O AutoML
- Best Model chosen from leaderboard

Step 4 - Performance Evaluation

RMSE : Error magnitude
R^2: Explained variance
Accuracy : Binned classification
Within 10% : Prediction tolerance

Experiment Application System Architecture

CSV data	raw input saved in repo
Preprocessing	remove missing values or ensure types correct
ManualFE	script create_meaningful_features() that adds domain-specific columns
AutoFE	Featuretools DFS output; pruned by correlation to price
DatasetBuilder	constructs the four experimental datasets and converts to H2OFrame

Experiment Application System Architecture

H2O cluster	local JVM process started by h2o.init()
AutoML	runs per-dataset with H2OAutoML(max_runtime_secs=..., nfolds=5, exclude_algos=...)
Leaderboard	top models H2O found
Evaluation	script collecting RMSE/R^2/MAPE/%within10 and compiling final table

Experiment Steps

Step 1 : Data Loading and Preparation

Task: Load California housing dataset and prepare baseline features

```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']

Experiment Steps

Step 2 : Manual Feature Engineering

Task: Create domain-specific features using California housing knowledge

Result :
Created 15 manual features including location, household ratios, and interactions.

```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
```

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

Experiment Steps

Task: Create domain-specific features using California housing knowledge

Result :
Created 15 manual features including location, household ratios, and interactions.

Step 2 : Manual Feature Engineering

- is_northern_ca
- is_coastal
- dist_to_sf
- dist_to_la
- rooms_per_household
- bedrooms_per_room
- people_per_room
- income_per_room
- income_per_person
- medinc_squared
- medinc_log
- rooms_log
- income_rooms_interaction
- income_age_interaction
- coastal_income_interaction

Experiment Steps

Step 3 : Automated Feature Engineering with Featuretools

Task: Generate features automatically using Deep Feature Synthesis (DFS)

Result :
Generated 100+ automated features like ft_MedInc_AveRooms, ft_Population_sum

```
# Create EntitySet
es = ft.EntitySet(id="housing_analysis")

# Define data types efficiently
logical_types = {
    'MedInc': Double, 'HouseAge': Double, 'AveRooms': Double, 'AveBedrms': Double,
    'Population': Double, 'AveOccup': Double, 'Latitude': Double, 'Longitude': Double,
    'Price': Double, 'rooms_per_bedroom': Double, 'people_per_household': Double,
    'bedroom_ratio': Double, 'income_level': Categorical, 'age_group': Categorical
}

# Add data to EntitySet
es = es.add_dataframe(
    dataframe=df_features,
    dataframe_name="houses",
    index="house_id",
    logical_types=logical_types
)

# Generate features with optimized parameters
print("   Running Deep Feature Synthesis (optimized)...")
feature_matrix, feature_definitions = ft.dfs(
    entityset=es,
    target_dataframe_name="houses",
    agg_primitives=["mean", "std"],  # Reduced primitives for speed
    trans_primitives=["add_numeric", "multiply_numeric"],  # Essential transformations only
    max_depth=1,  # Reduced depth for speed
    verbose=False
)
```

```
Price,income_level,age_group,rooms_per_bedroom,people_per_household,bedroom_ratio,AveBedrms + AveOccup,AveBedrms + AveRoc
4.526,High,Old,6.821705426356592,126.0,0.146590909090909,3.579365079365079,8.007936507936508,42.023809523809526,38.903809
3.585,High,Medium,6.418625678119349,1138.0,0.1557965910691646,3.081722319859402,7.210017574692443,21.971880492091387,38.8
3.521,High,Old,7.721052631578947,177.0,0.1295160190865712,3.8757062146892656,9.361581920903957,53.07344632768361,38.92344
3.413,Medium,Old,5.421276595744681,219.0,0.184458398744113,3.6210045662100456,6.89041095890411,53.0730593607306,38.923059
3.422,Medium,Old,5.810714285714285,259.0,0.1720958819913952,3.2625482625482625,7.362934362934363,53.08108108108108,38.931
```

Experiment Steps

Step 4 : Dataset Preparation and Splitting

Task: Create four different feature sets for comparison

Result :
Training set: 80% (16,512 samples)
Test set: 20% (4,128 samples)

```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}
```

```
Dataset sizes:
original              -> Train: (16582, 9), Test: (4058, 9)
combined              -> Train: (16582, 24), Test: (4058, 24)
auto                  -> Train: (16582, 39), Test: (4058, 39)
ft_only               -> Train: (16582, 31), Test: (4058, 31)
```

Experiment Steps

Step 5: AutoML Training and Evaluation

Task: Train H2O AutoML on each feature set and evaluate performance

Result :
Trained 4 AutoML models with different feature sets.

```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
```

```
🏃 Training AutoML on original features...
   Using 8 features
AutoML progress: |                                                                                    |   0%
23:53:29.213: Project: AutoML_1_20251005_235329
23:53:29.214: Setting stopping tolerance adaptively based on the training frame: 0.007765716727382795
23:53:29.214: Build control seed: 42
23:53:29.215: training frame: Frame key: AutoML_1_20251005_235329_training_py_2_sid_946c    cols: 9    rows: 16582
  chunks: 1    size: 647390  checksum: -6727004212979191184
23:53:29.215: validation frame: NULL
23:53:29.215: leaderboard frame: NULL
23:53:29.215: blending frame: NULL
23:53:29.215: response column: Price
23:53:29.215: fold column: null
23:53:29.215: weights column: null
```

Experiment Steps

Step 6: Feature Importance Analysis

Task: Analyze which features contribute most to model performance

```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
```

Experiment Results

Main comparison table

```
================================================================================
📈 FEATURE ENGINEERING COMPARISON RESULTS
================================================================================
Approach                  | Model                          |     RMSE |     R² |  Accuracy |  Within 10%
--------------------------------------------------------------------------------
original                  | H2OGeneralizedLinearEstimator  |   0.7186 | 0.6089 |     68.9% |       23.8% 🟡 +0.0%
original_plus_engineered  | H2OStackedEnsembleEstimator    |   0.5146 | 0.7994 |     80.8% |       40.7% 🟢 +31.3%
original_plus_auto        | H2OStackedEnsembleEstimator    |   0.4409 | 0.8528 |     84.6% |       47.6% 🟢 +40.0%
featuretools_only         | H2OStackedEnsembleEstimator    |   0.5107 | 0.8025 |     81.4% |       40.1% 🟢 +31.8%
```

Key Findings:

- All methods beat the baseline
- RMSE dropped by 28–39%
- R^2 jumped from 0.61 to 0.80–0.85

- "Original + Auto" achieved the best results across all metrics:
- Lowest RMSE (0.4409)
- R^2 (0.8528)
- Highest Accuracy (84.6%)
- Best Within 10% performance (47.6%)

Experiment Results

Top Performing Results

Rank	Feature	Origin	Max Imp	Avg Imp	Appears In
1.	Latitude	Original	1.000	0.118	original, origina...
2.	income_per_person	Manual	1.000	1.000	original_plus_eng...
3.	ft_bedroom_ratio * MedInc	Featuretools	1.000	1.000	original_plus_aut...
4.	Longitude	Original	0.972	0.111	original, origina...
5.	MedInc	Original	0.957	0.071	original, origina...
6.	income_per_room	Manual	0.922	0.922	original_plus_eng...

Feature Origin Summary

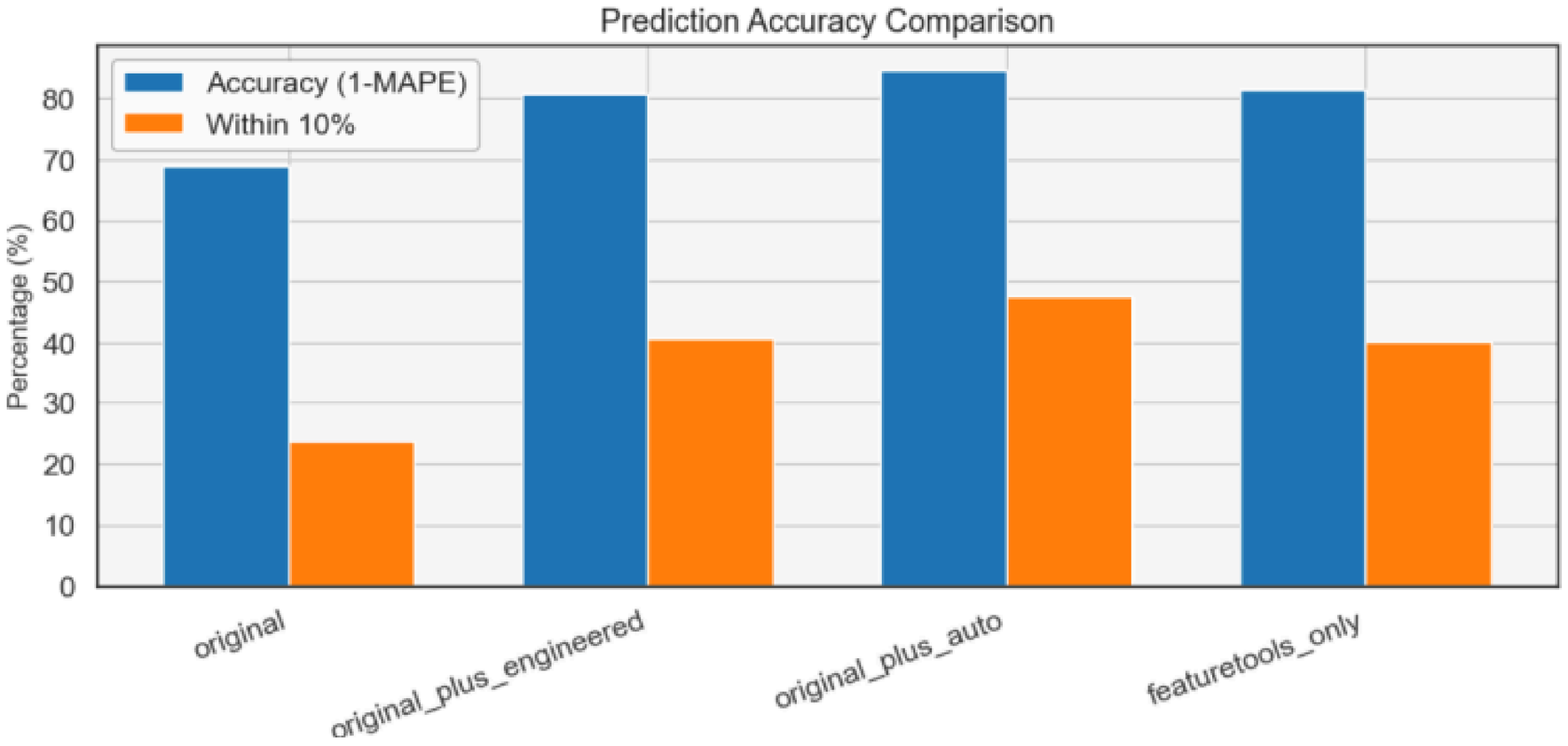
- Original | Avg Importance: 1.578 | Avg Count: 8.0 | In 3 approaches
- Manual | Avg Importance: 2.511 | Avg Count: 15.0 | In 1 approaches
- Featuretools | Avg Importance: 3.047 | Avg Count: 30.0 | In 2 approaches

Experimental Result

Conclusion :
The combined feature engineering of AutoML and FeatureTools significantly enhances the overall model fitting capability.

RMSE: How large the model's average prediction error is per prediction.
R^2: Measures the model's ability to capture the underlying patterns in the data.

Experimental Result

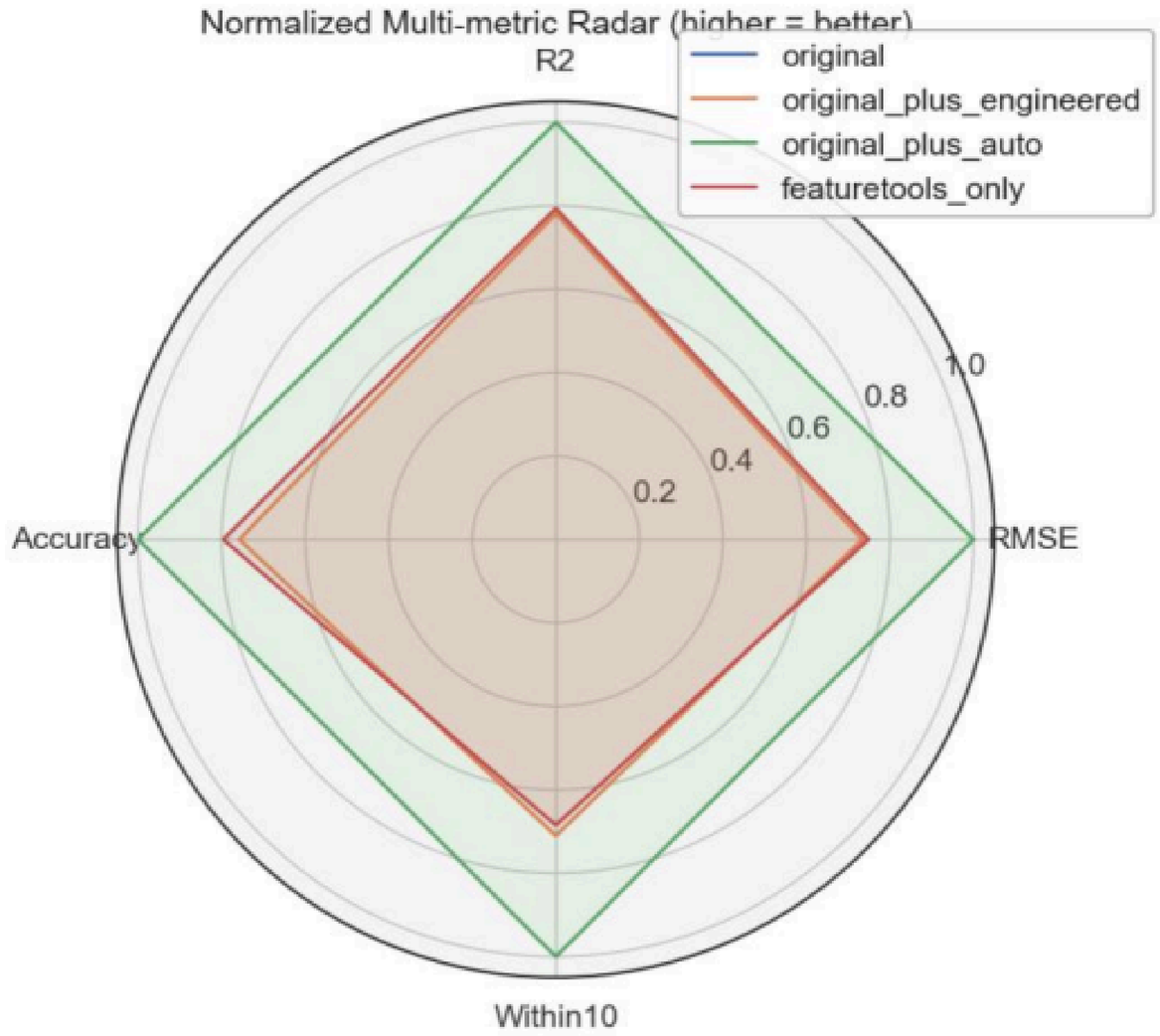


Conclusion :

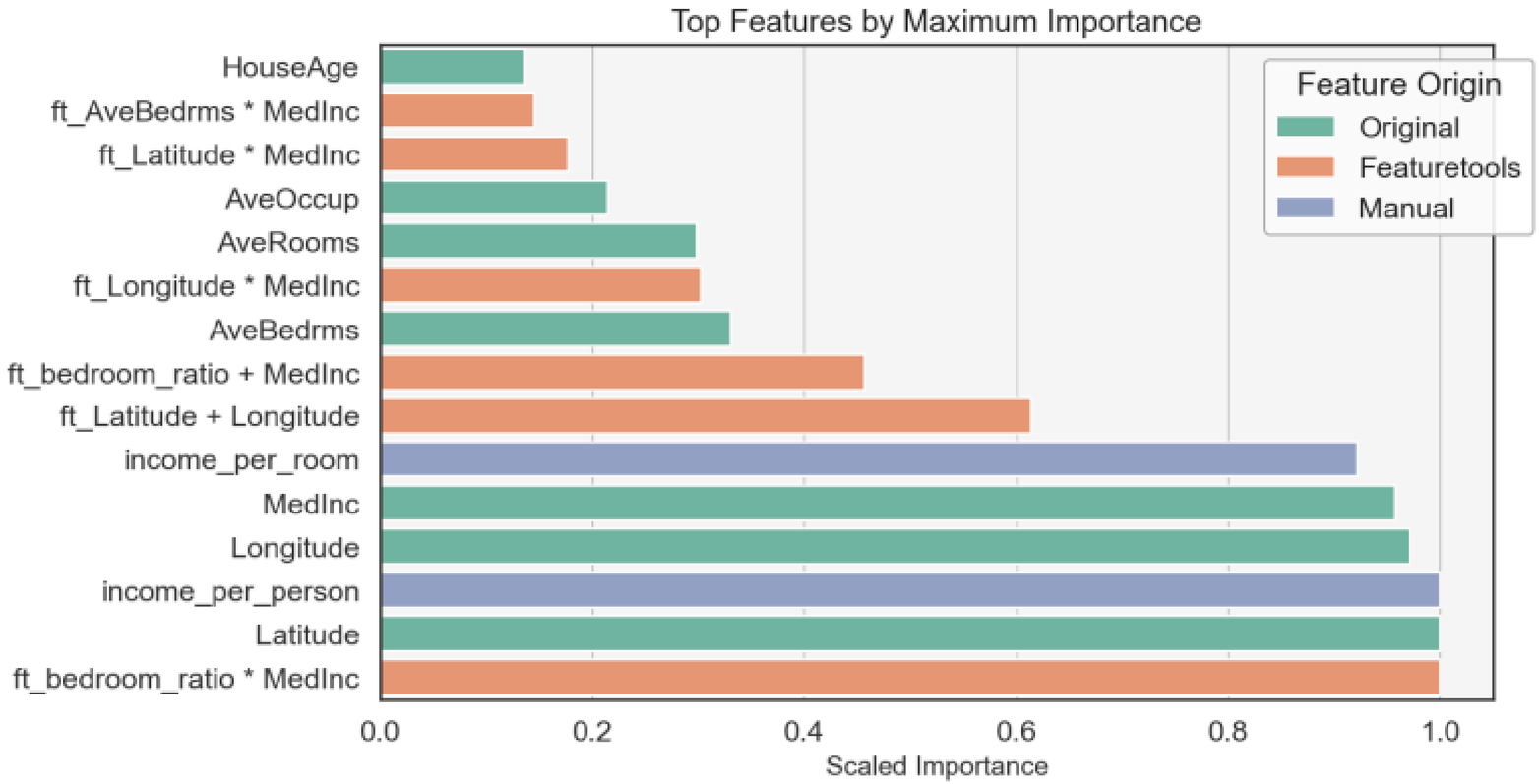
- Automated features boost accuracy and reduce errors
- FeatureTools_only still performs well without original features
- Other approaches - moderate

Experimental Result

This chart shows that Auto + FeatureTools is the most comprehensive and robust.



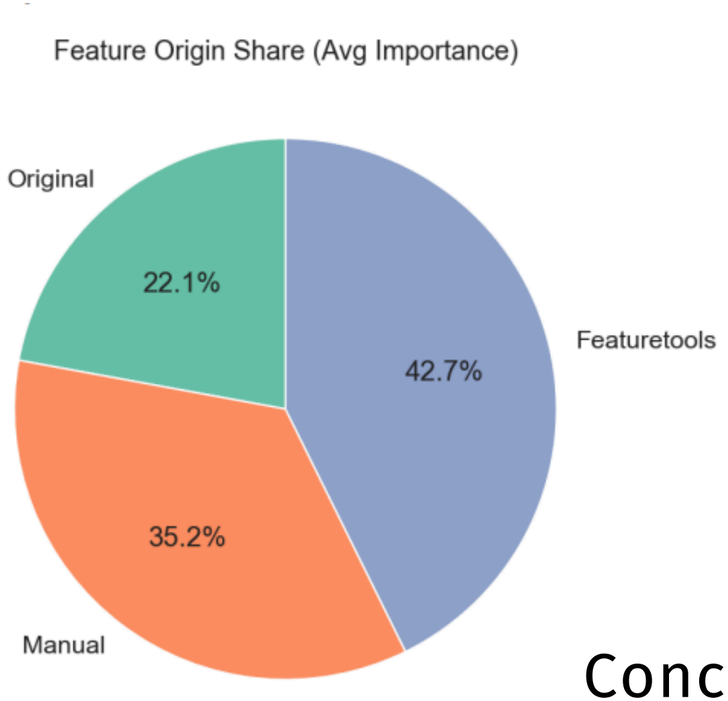
Experimental Result

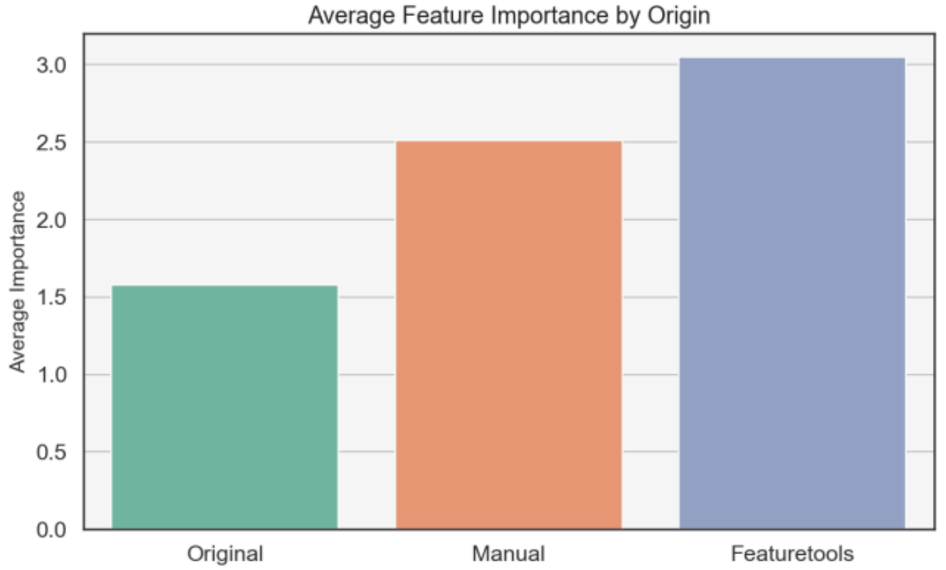


Conclusion :

- Automated features find complex patterns
- Manual features offer domain interpetability
- Original features provide basic signal source

Experimental Result





- Pie chart area represents the relative contribution
- Bar chart reinforces quantitative differences

Conclusion:
Automated feature engineering (FeatureTools) greatly increases model information and performance

Experiment Analysis

Feature Engineering Impact

Engineered features greatly improved metrics, proving the importance of informative derived features

Model Selection Impact

AutoML picked GLM (Generalized Linear Model) for raw data, but a Stacked Ensemble for engineered data. This concludes that proving richer features led to more complex, better-performing models.

Human vs AI in the loop

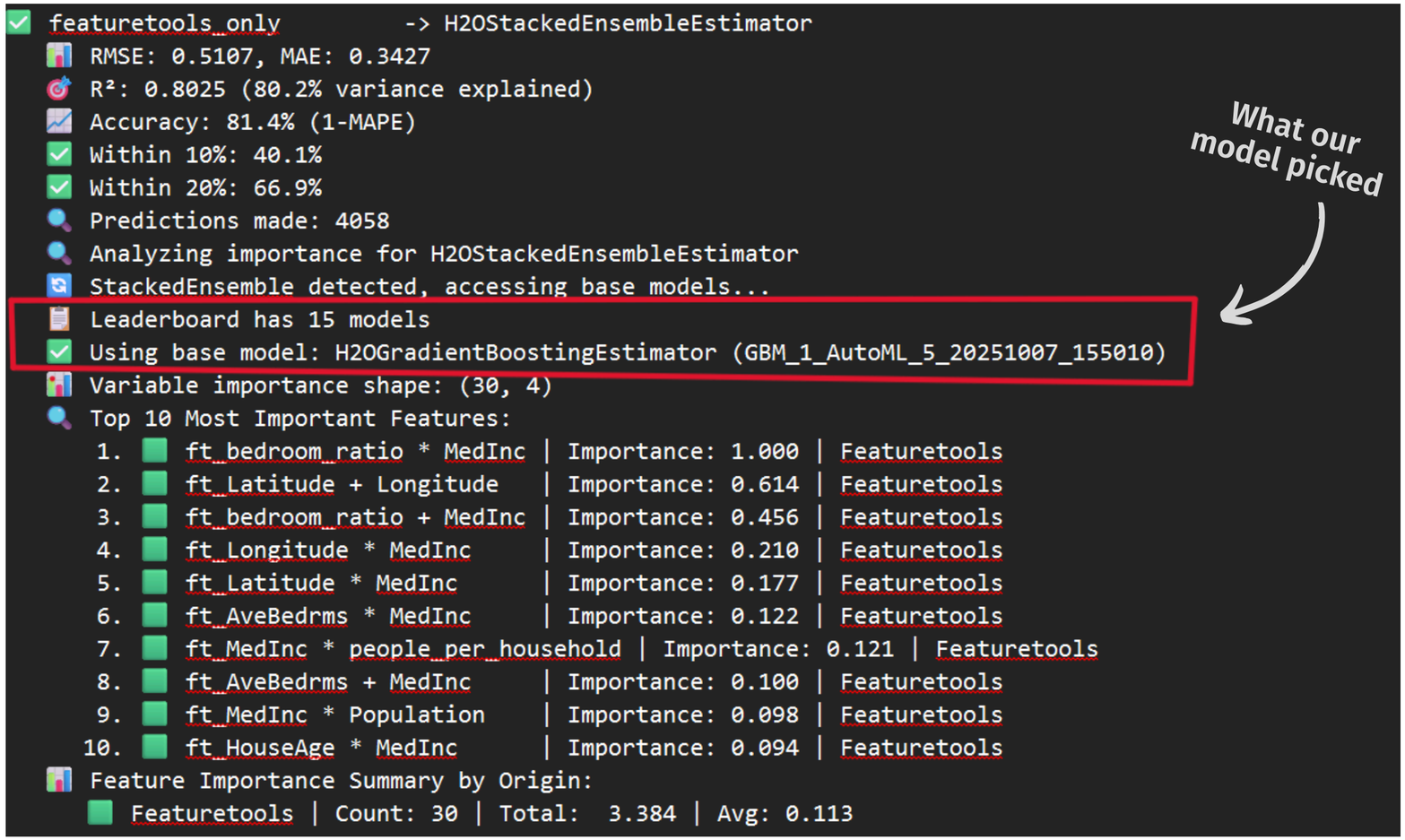
AutoML

Human in the Loop	AI in the Loop
Problem Definition	Basic data preprocessing
Data Understanding	Model training
Workflow Design	Hyperparameter training
Advanced data preprocessing	Stacking (a ensemble learning technique)
Feature engineering	Creating model results tables
Model deployment	

Stacking (a ensemble learning technique)

Stacking in H2O AutoML

- Combines predictions from multiple models using a meta-model
- Learns the best way to blend base models (e.g. GBM, RF, XGBoost)
- Boosts accuracy by leveraging diverse model strengths



How AI helps the experiment

Helps via FeatureTools

(auto feature engineering) + H2O AutoML

Eliminates manual work

No need to design features manually or tune models

Accessible to all

can be made without any prior knowledge of coding or even ML skills

Conclusion

1

Feature engineering is the backbone of model performance, even in AutoML.

2

Automating it unlocks scale, speed, and consistency across diverse datasets.

Thank you!

Feel free to approach us if you have any questions.