

# Copyright Notice

These slides are distributed under the Creative Commons License.

DeepLearning.AI makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite DeepLearning.AI as the source of the slides.

For the rest of the details of the license, see

<https://creativecommons.org/licenses/by-sa/2.0/legalcode>

# Feature Engineering, Transformation and Selection



DeepLearning.AI

---

# Welcome

# Feature Engineering

---



DeepLearning.AI

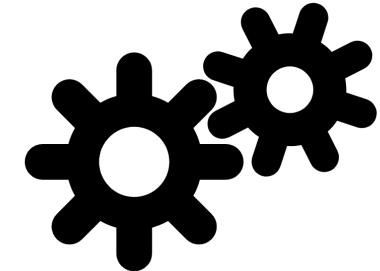
# Introduction to Preprocessing

*“Coming up with features is difficult,  
time-consuming, and requires expert knowledge.  
Applied machine learning often requires careful  
engineering of the features and dataset.”*

— Andrew Ng

# Outline

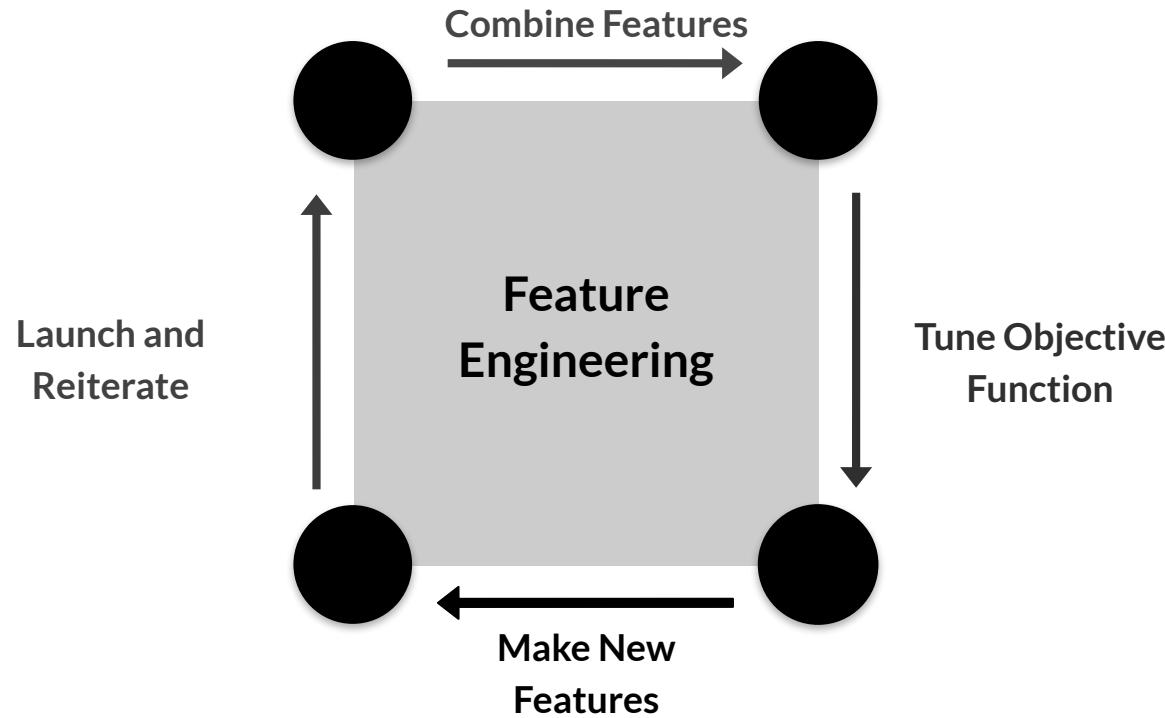
- Squeezing the most out of data
- The art of feature engineering
- Feature engineering process
- How feature engineering is done in a typical ML pipeline



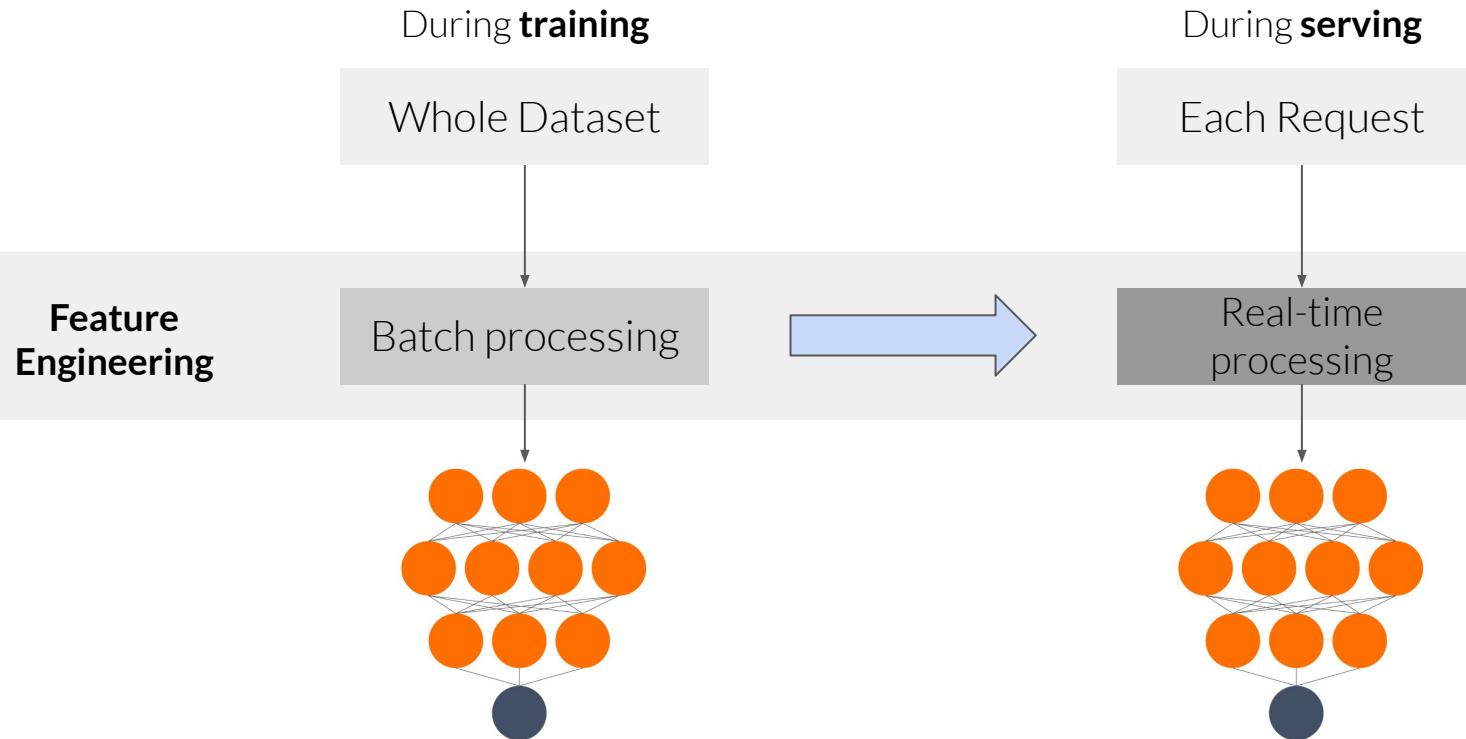
# Squeezing the most out of data

- Making data useful before training a model
- Representing data in forms that help models learn
- Increasing predictive quality
- Reducing dimensionality with feature engineering

# Art of feature engineering



# Typical ML pipeline



# Key points

- Feature engineering can be difficult and time consuming, but also very important to success
- Squeezing the most out of data through feature engineering enables models to learn better
- Concentrating predictive information in fewer features enables more efficient use of compute resources
- Feature engineering during training must also be applied correctly during serving

# Feature Engineering

---

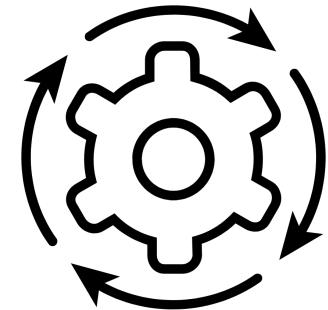


DeepLearning.AI

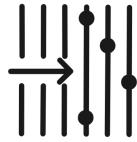
# Preprocessing Operations

# Outline

- Main preprocessing operations
- Mapping raw data into features
- Mapping numeric values
- Mapping categorical values
- Empirical knowledge of data



# Main preprocessing operations



Data cleansing



Feature tuning



Representation  
transformation

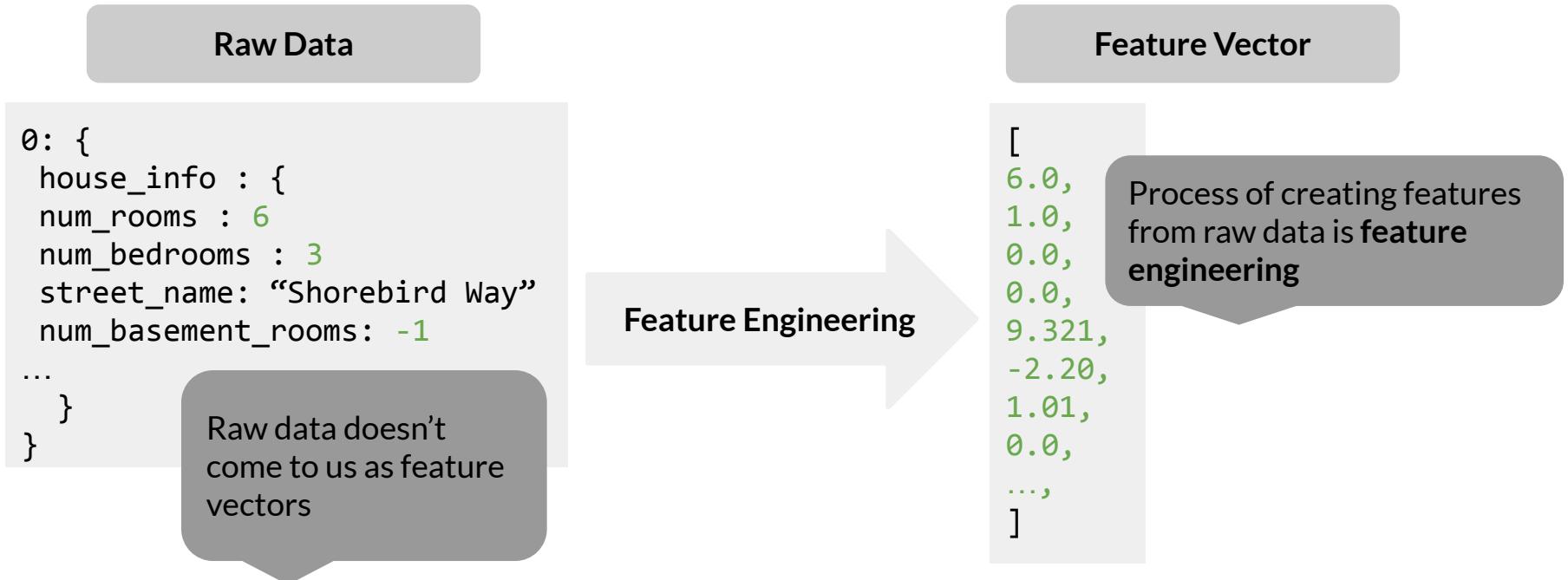


Feature  
extraction



Feature  
construction

# Mapping raw data into features



# Mapping categorical values

## Street names

{'Charleston Road', 'North Shoreline Boulevard', 'Shorebird Way', 'Rengstorff Avenue'}

Raw Data

```
0: {  
    house_info : {  
        num_rooms : 6  
        num_bedrooms : 3  
        street_name: "Shorebird Way"  
        num_basement_rooms: -1  
    ...  
    }  
}
```

String Features can be handled with one-hot encoding

Feature Engineering

Feature Vector

One-hot encoding  
This has a 1 for “Shorebird way” and 0 for all others

street\_name feature=  
[0,0, ..., 0, 1, 0, ..., 0]

# Categorical Vocabulary

```
# From a vocabulary list

vocabulary_feature_column = tf.feature_column.categorical_column_with_vocabulary_list(
    key=feature_name,
    vocabulary_list=["kitchenware", "electronics", "sports"])

# From a vocabulary file

vocabulary_feature_column = tf.feature_column.categorical_column_with_vocabulary_file(
    key=feature_name,
    vocabulary_file="product_class.txt",
    vocabulary_size=3)
```

# Empirical knowledge of data



**Text** - stemming, lemmatization, TF-IDF, n-grams, embedding lookup



**Images** - clipping, resizing, cropping, blur, Canny filters, Sobel filters, photometric distortions

# Key points

- Data preprocessing: transforms raw data into a clean and training-ready dataset
- Feature engineering maps:
  - Raw data into feature vectors
  - Integer values to floating-point values
  - Normalizes numerical values
  - Strings and categorical values to vectors of numeric values
  - Data from one space into a different space



DeepLearning.AI

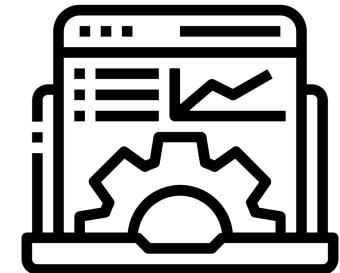
# Feature Engineering

---

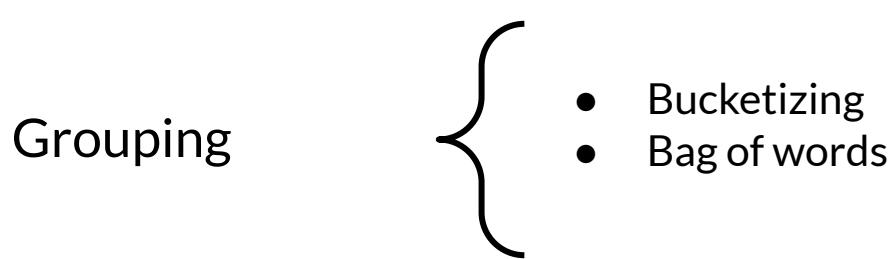
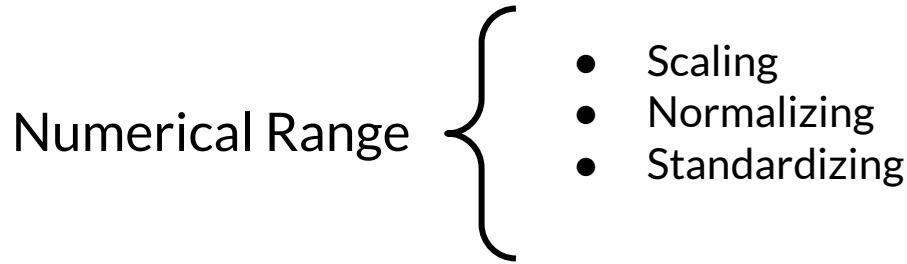
## Feature Engineering Techniques

# Outline

- Feature Scaling
- Normalization and Standardization
- Bucketizing / Binning
- Other techniques



# Feature engineering techniques



# Scaling

- Converts values from their natural range into a prescribed range
  - E.g. Grayscale image pixel intensity scale is [0,255] usually rescaled to [-1,1]

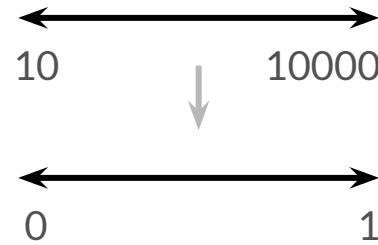
```
image = (image - 127.5) / 127.5
```

- Benefits
  - Helps neural nets converge faster
  - Do away with NaN errors during training
  - For each feature, the model learns the right weights

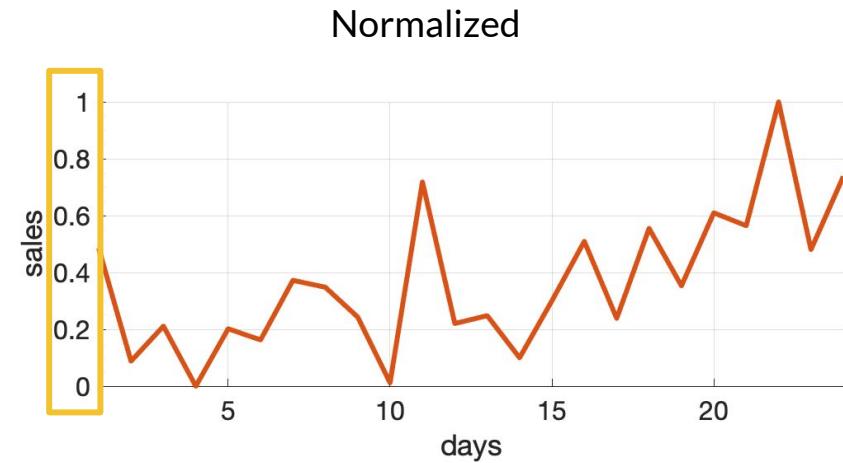
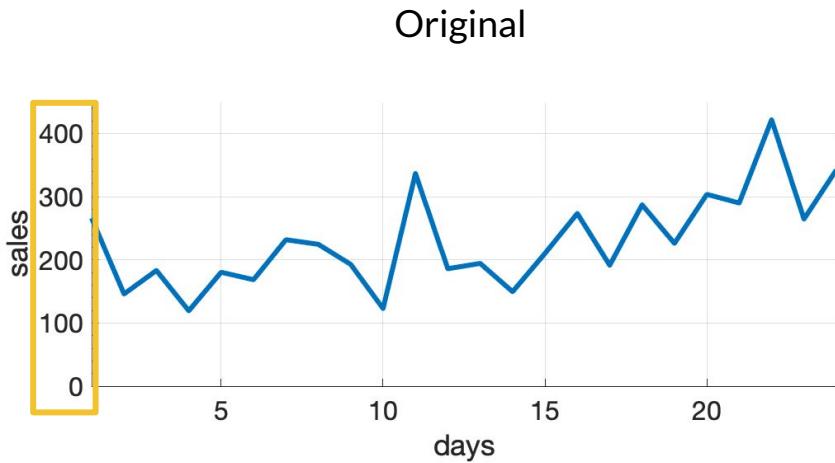
# Normalization

$$X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

$$X_{\text{norm}} \in [0, 1]$$



# Normalization

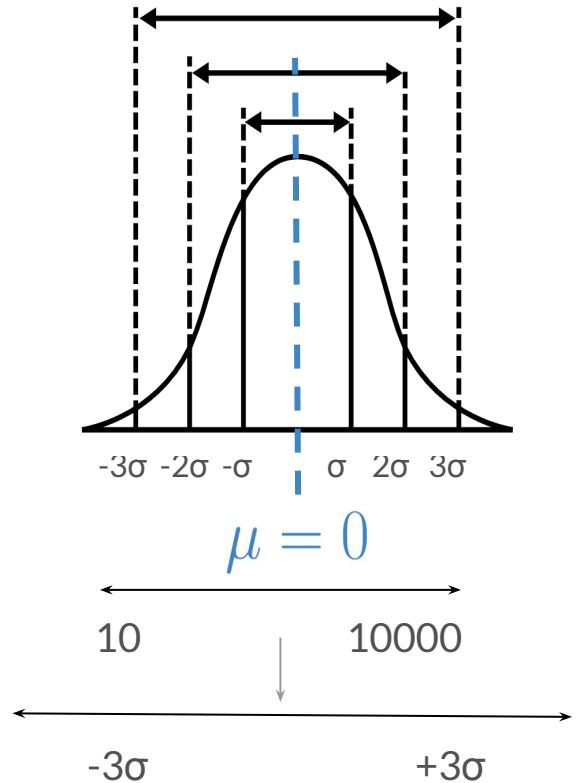


# Standardization (z-score)

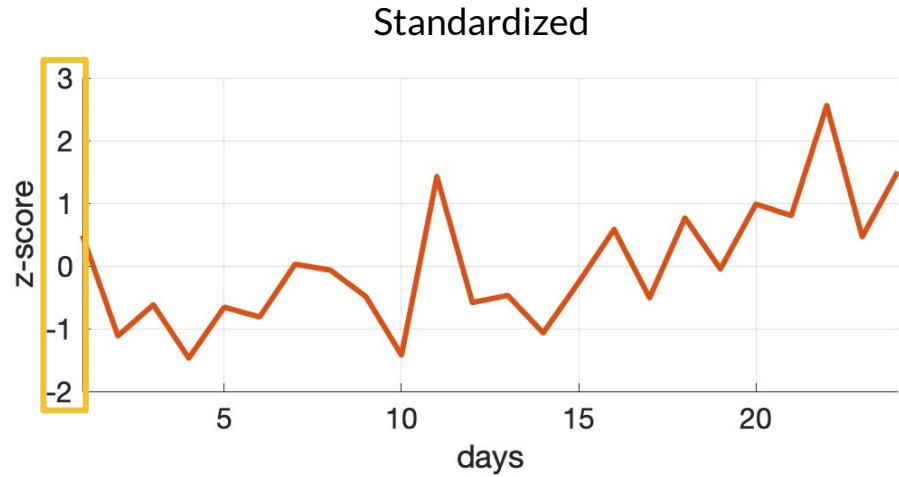
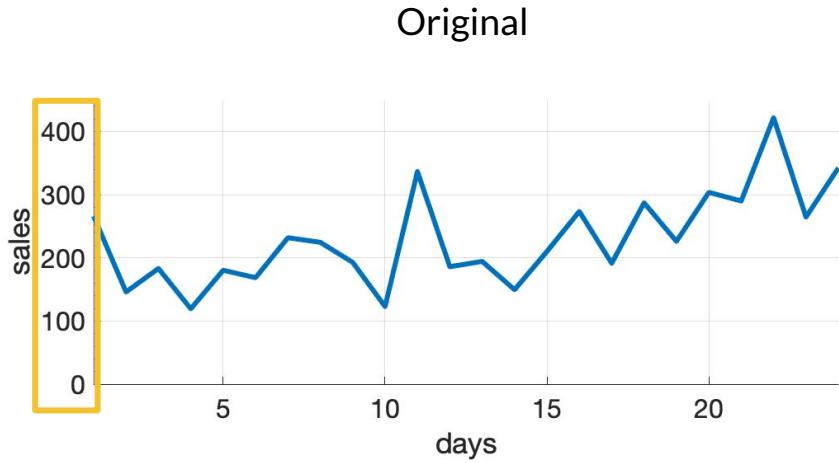
- Z-score relates the number of standard deviations away from the mean
- Example:

$$X_{\text{std}} = \frac{X - \mu}{\sigma} \quad (\text{z-score})$$

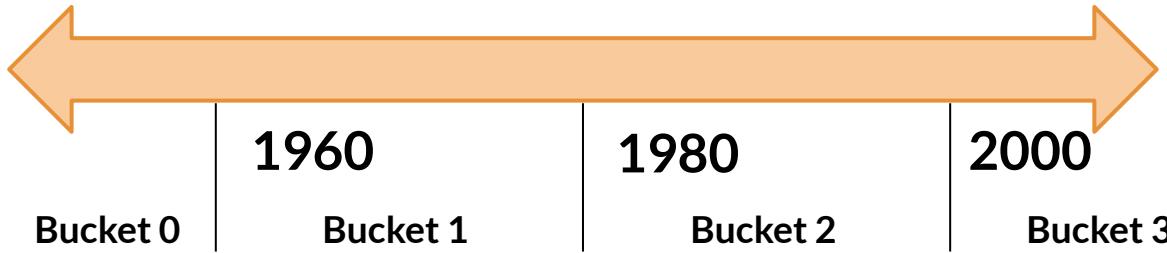
$$X_{\text{std}} \sim \mathcal{N}(0, \sigma)$$



# Standardization (z-score)

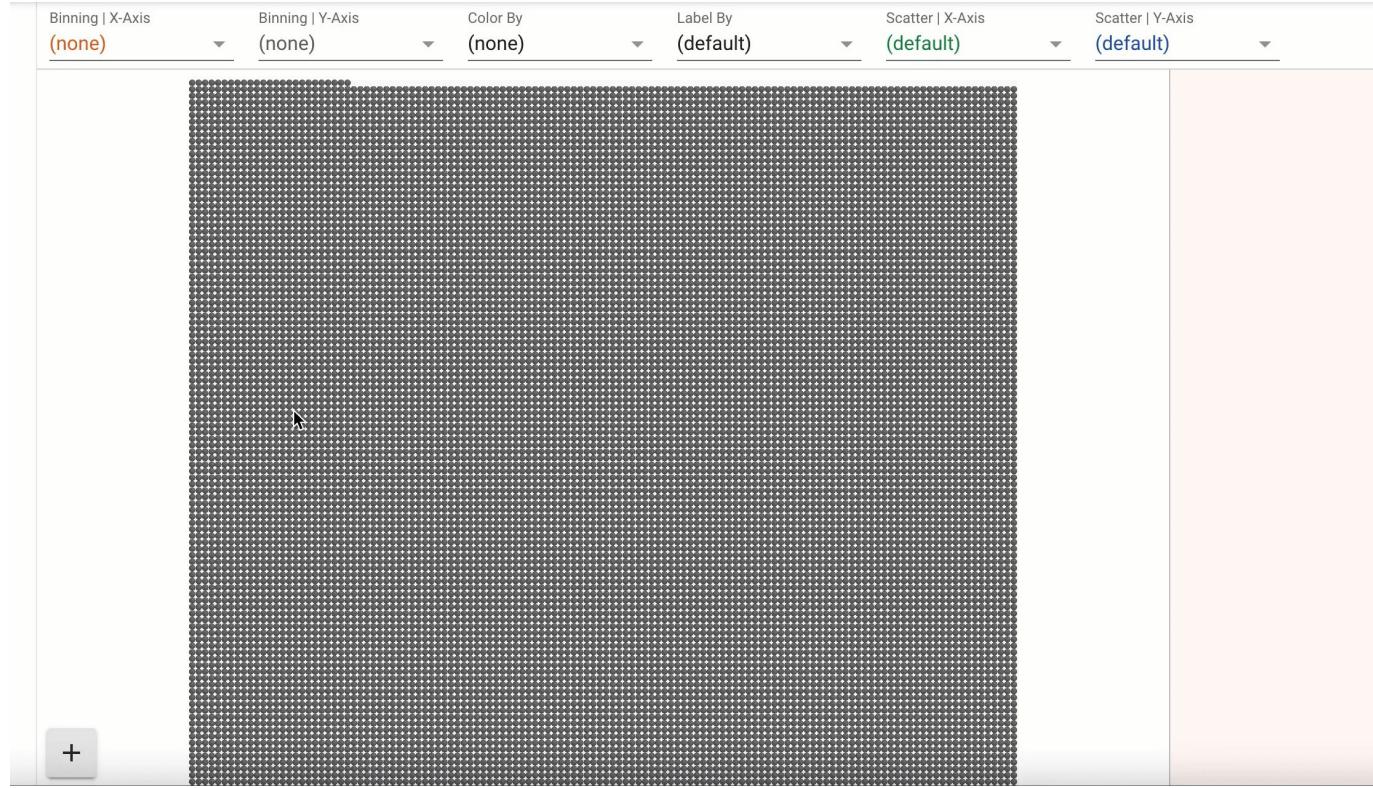


# Bucketizing / Binning



Date Range	Represented as...
< 1960	[1, 0, 0, 0]
$\geq 1960$ but $< 1980$	[0, 1, 0, 0]
$\geq 1980$ but $< 2000$	[0, 0, 1, 0]
$\geq 2000$	[0, 0, 0, 1]

# Binning with Facets



# Other techniques

Dimensionality reduction in embeddings

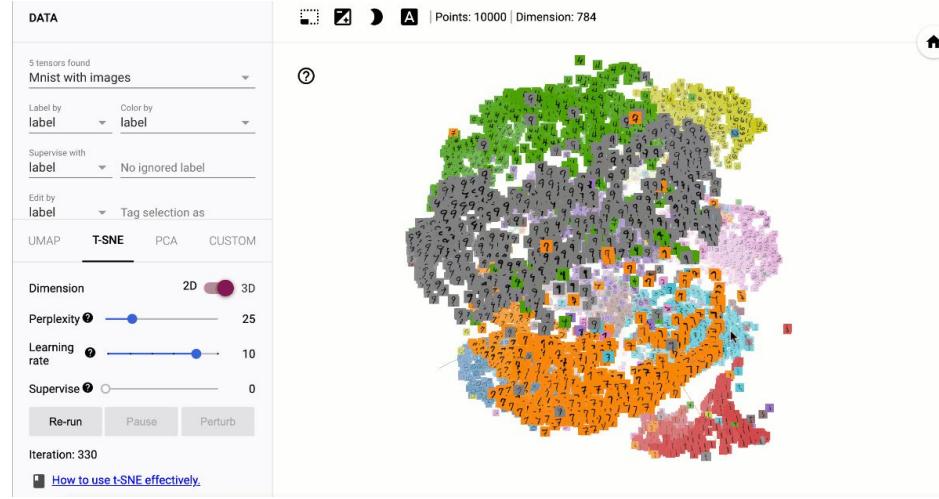
- 
- Principal component analysis (PCA)
  - t-Distributed stochastic neighbor embedding (t-SNE)
  - Uniform manifold approximation and projection (UMAP)

Feature crossing

# TensorFlow embedding projector

- Intuitive exploration of high-dimensional data
- Visualize & analyze
- Techniques
  - PCA
  - t-SNE
  - UMAP
  - Custom linear projections
- Ready to play

@projector.tensorflow.org



# Key points

- Feature engineering:
  - Prepares, tunes, transforms, extracts and constructs features.
- Feature engineering is key for model refinement
- Feature engineering helps with ML analysis

# Feature Engineering

---

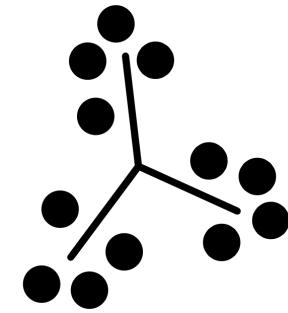


DeepLearning.AI

# Feature Crosses

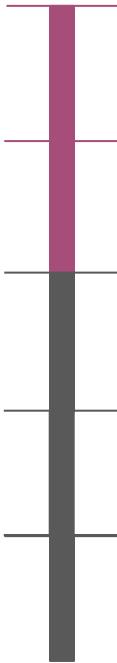
# Outline

- Feature crosses
- Encoding features



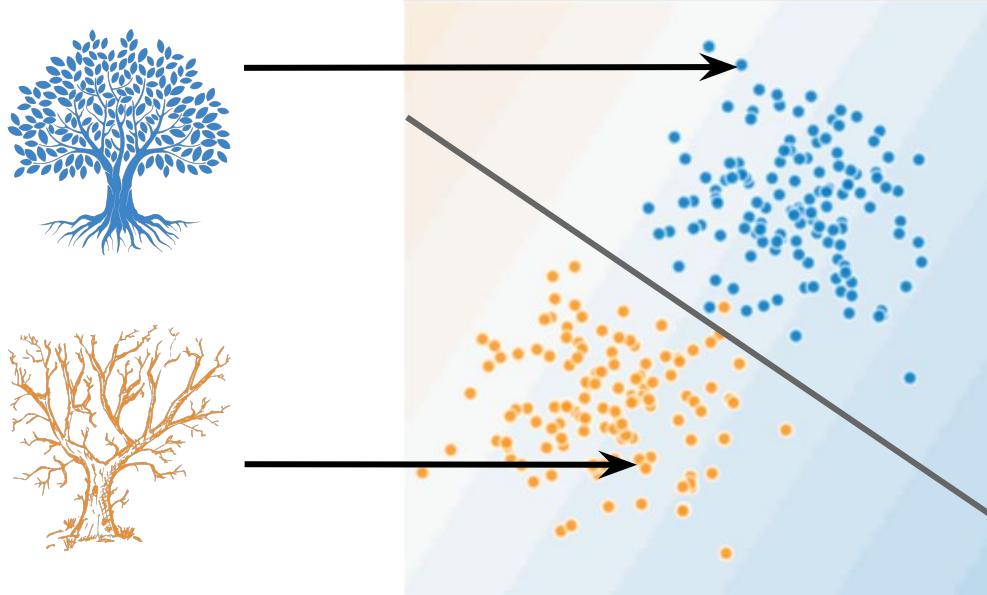
# Feature crosses

We can create many different kinds of feature crosses



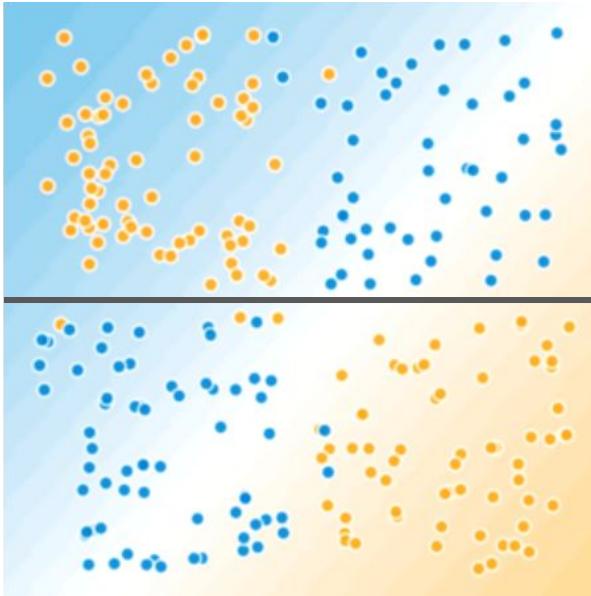
- Combines multiple features together into a new feature
- Encodes nonlinearity in the feature space, or encodes the same information in fewer features
- $[A \times B]$ : multiplying the values of two features
- $[A \times B \times C \times D \times E]$ : multiplying the values of 5 features
- [Day of week, Hour] => [Hour of week]

# Encoding features



- healthy trees
  - sick trees
- Classification boundary

# Need for encoding non-linearity



- healthy trees
  - sick trees
- Classification boundary

# Census dataset



# Key points

- Feature crossing: synthetic feature encoding nonlinearity in feature space.
- Feature coding: transforming categorical to a continuous variable.



DeepLearning.AI

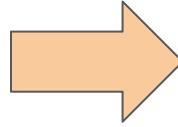
# Feature Transformation At Scale

---

## Preprocessing Data At Scale

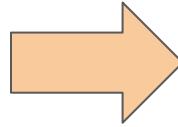
# Probably not ideal

The screenshot shows a Google Colab notebook interface. The title bar says 'beginner.ipynb'. The main content area displays the 'TensorFlow 2 quickstart for beginners' tutorial. It includes sections on Keras, instructions for running the notebook in Colab, and code snippets for importing TensorFlow and preparing the MNIST dataset.



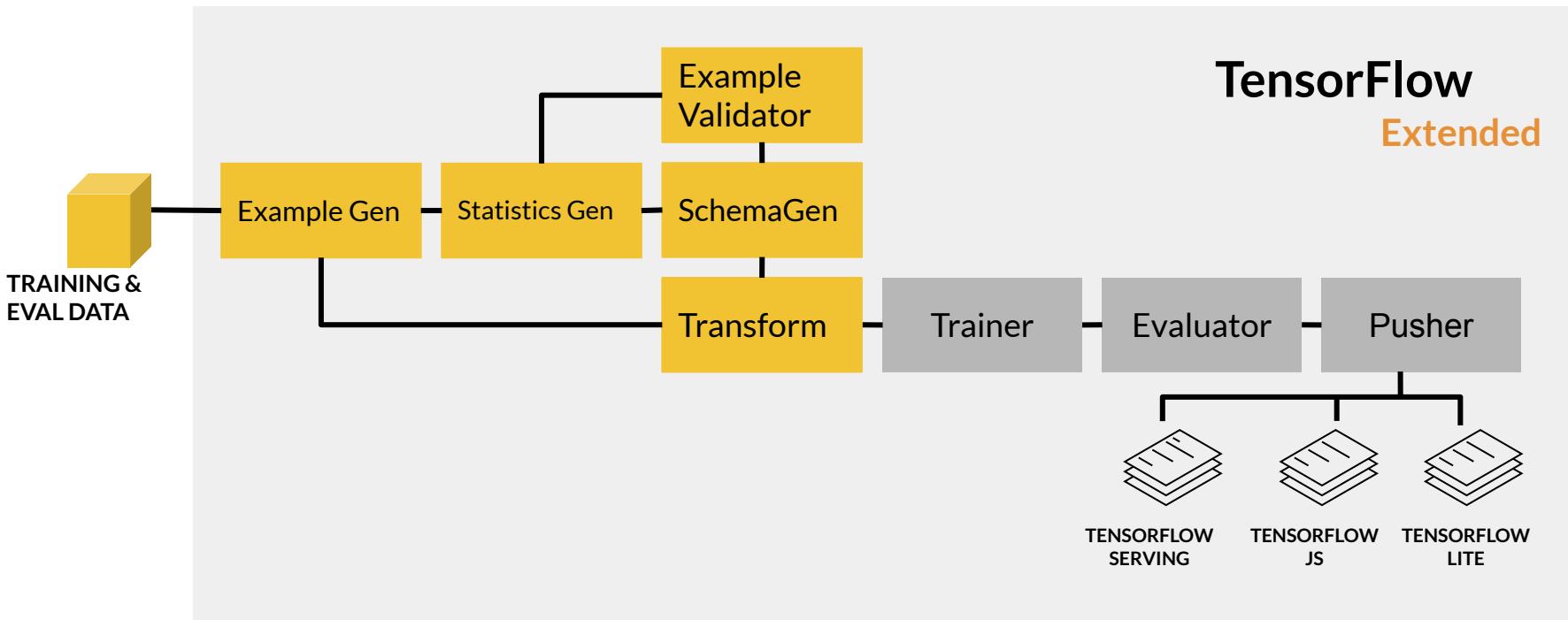
APACHE  
STORM™

Python



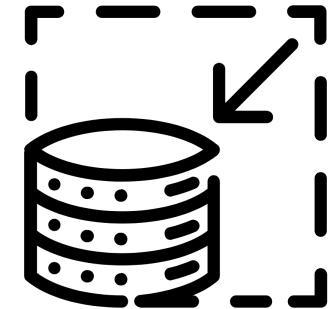
Java

# ML Pipeline



# Outline

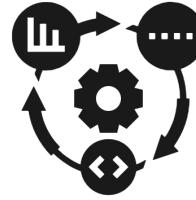
- Inconsistencies in feature engineering
- Preprocessing granularity
- Pre-processing training dataset
- Optimizing instance-level transformations
- Summarizing the challenges



# Preprocessing data at scale



Real-world models:  
terabytes of data



Large-scale data  
processing frameworks

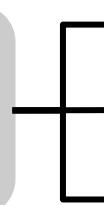


Consistent transforms  
between training &  
serving

# Inconsistencies in feature engineering

Training & serving code paths are different

Diverse deployments scenarios



Mobile (TensorFlow Lite)

Server (TensorFlow Serving)

Web (TensorFlow JS)

Risks of introducing training-serving skews

Skews will lower the performance of your serving model

# Preprocessing granularity

Transformations	
Instance-level	Full-pass
Clipping	Minimax
Multiplying	Standard scaling
Expanding features	Bucketizing
etc.	etc.

# When do you transform?

Pre-processing training dataset

Pros	Cons
Run-once	Transformations reproduced at serving
Compute on entire dataset	Slower iterations

# How about ‘within’ a model?

Transforming within the model

Pros	Cons
Easy iterations	Expensive transforms
Transformation guarantees	Long model latency
	Transformations per batch: skew

# Why transform per batch?

- For example, normalizing features by their average
- Access to a single batch of data, not the full dataset
- Ways to normalize per batch
  - Normalize by average within a batch
  - Precompute average and reuse it during normalization

# Optimizing instance-level transformations

- Indirectly affect training efficiency
- Typically accelerators sit idle while the CPUs transform
- Solution:
  - Prefetching transforms for better accelerator efficiency

# Summarizing the challenges

- Balancing predictive performance
- Full-pass transformations on training data
- Optimizing instance-level transformations for better training efficiency  
(GPUs, TPUs, ...)

# Key points

- Inconsistent data affects the accuracy of the results
- Need for scaled data processing frameworks to process large datasets in an efficient and distributed manner

# Preprocessing Data At Scale

---

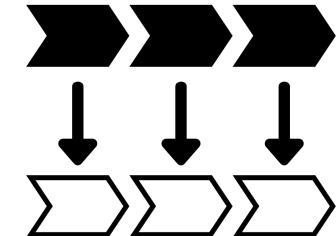


DeepLearning.AI

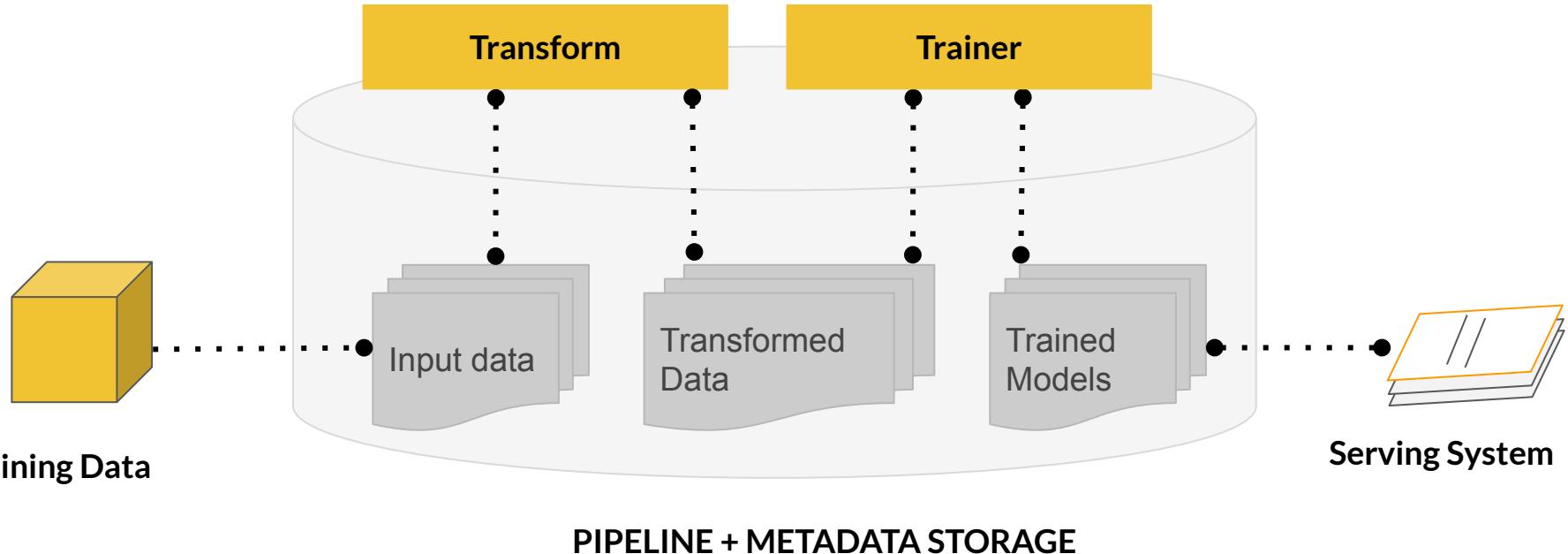
# TensorFlow Transform

# Outline

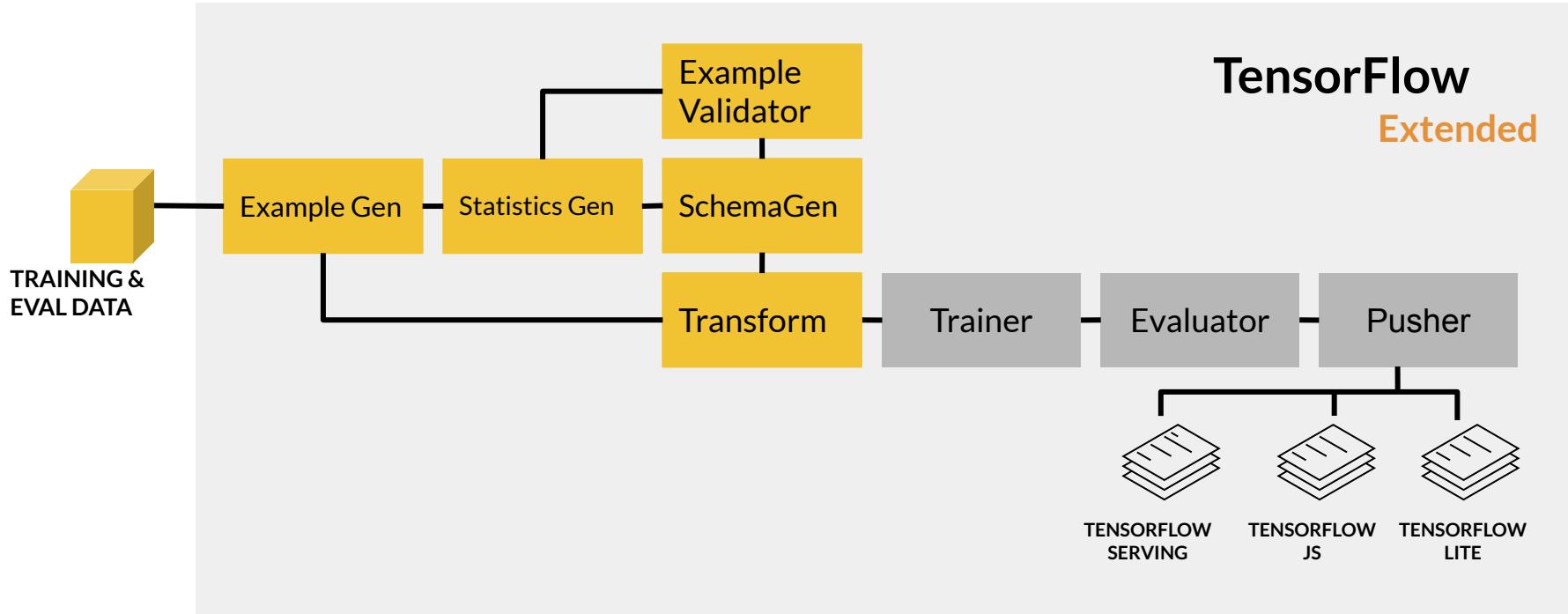
- Going deeper
- Benefits of using TensorFlow Transform
- Applies feature transformations
- tf.Transform Analyzers



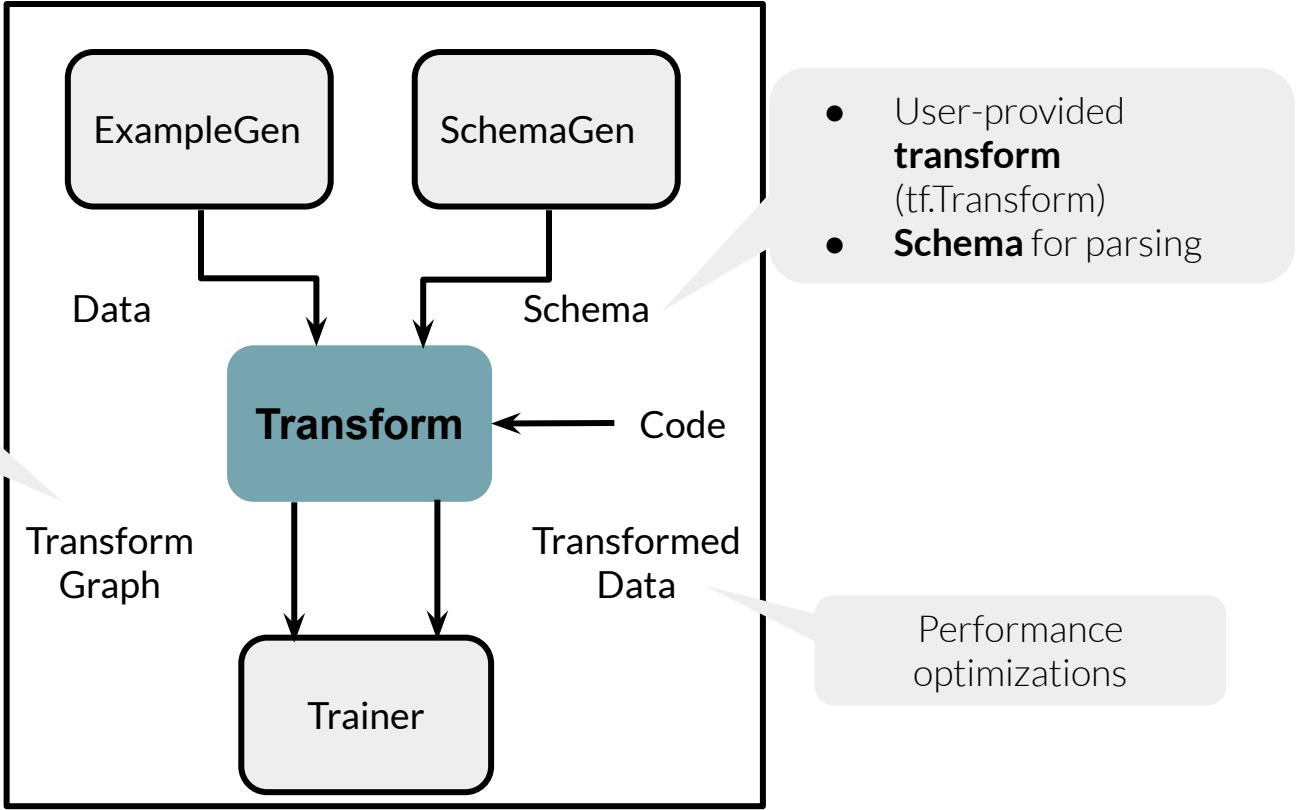
# Enter tf.Transform



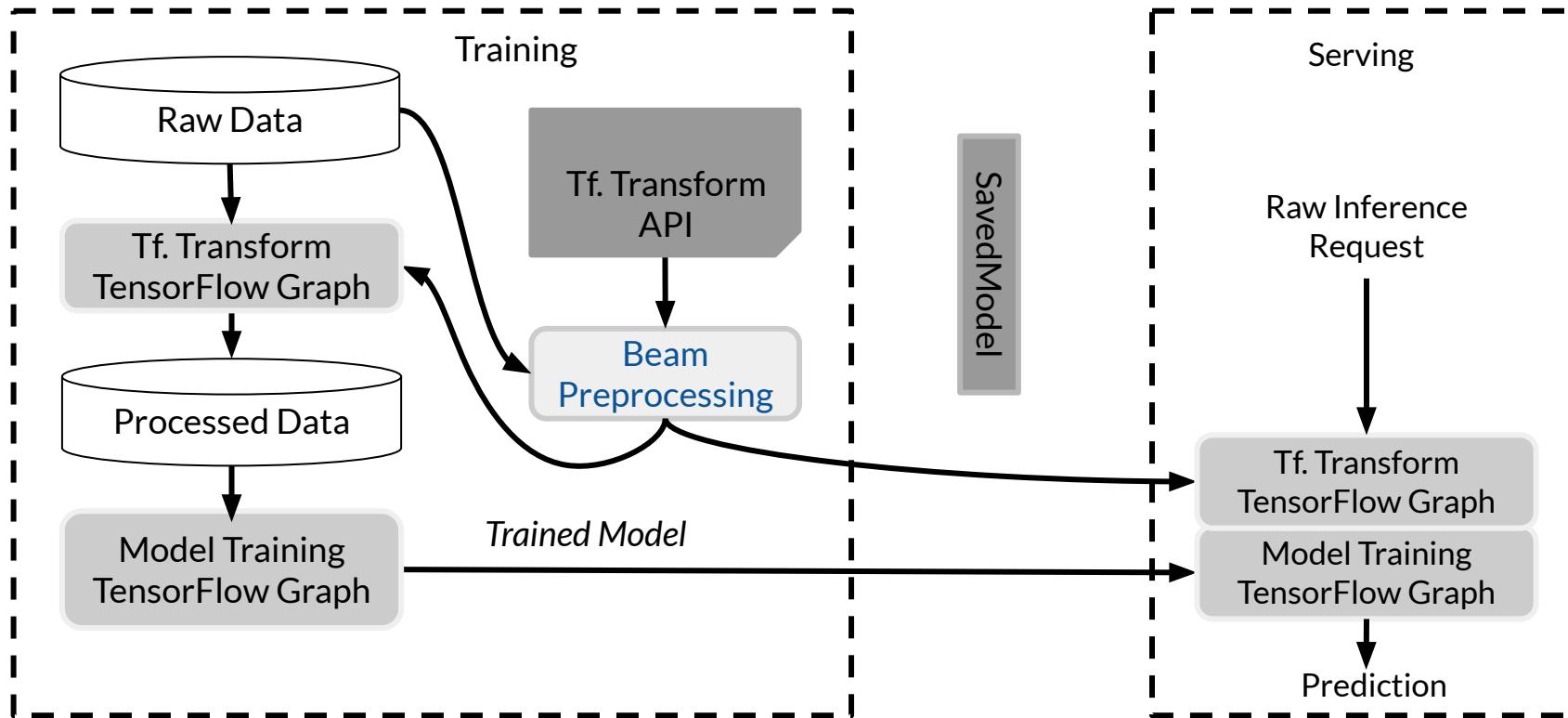
# Inside TensorFlow Extended



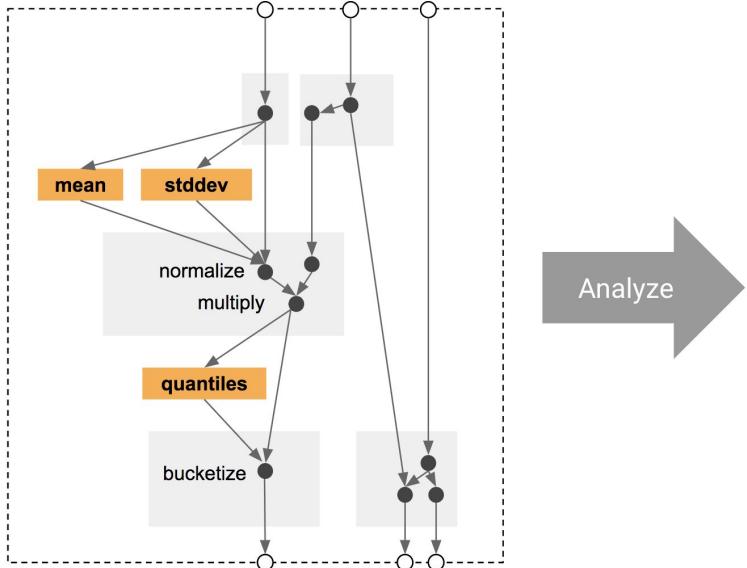
# tf.Transform layout



# tf.Transform: Going deeper



# tf.Transform Analyzers



They behave like TensorFlow Ops, but run only once during training

For example:

*tft.min computes the minimum of a tensor over the training dataset*

# How Transform applies feature transformations

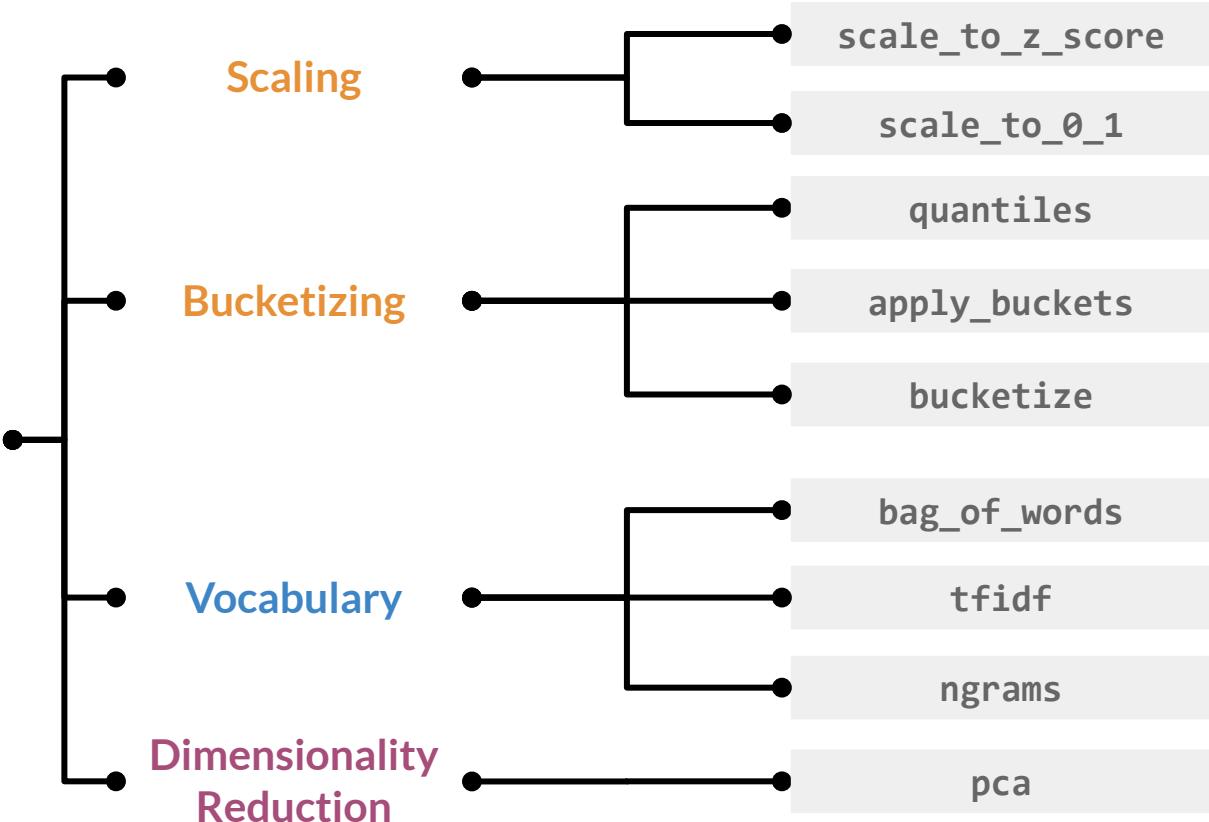


# Benefits of using tf.Transform

- Emitted tf.Graph holds all necessary constants and transformations
- Focus on data preprocessing only at training time
- Works in-line during both training and serving
- No need for preprocessing code at serving time
- Consistently applied transformations irrespective of deployment platform

# Analyzers framework

## tf.Transform Analyzers



# tf.Transform preprocessing\_fn

```
def preprocessing_fn(inputs):
    ...
    for key in DENSE_FLOAT_FEATURE_KEYS:
        outputs[key] = tft.scale_to_z_score(inputs[key])
    for key in VOCAB_FEATURE_KEYS:
        outputs[key] = tft.vocabulary(inputs[key], vocab_filename=key)
    for key in BUCKET_FEATURE_KEYS:
        outputs[key] = tft.bucketize(inputs[key], FEATURE_BUCKET_COUNT)
```

# Commonly used imports

```
import tensorflow as tf  
import apache_beam as beam  
import apache_beam.io.iobase  
  
import tensorflow_transform as tft  
import tensorflow_transform.beam as tft_beam
```

# Feature Transformation At Scale

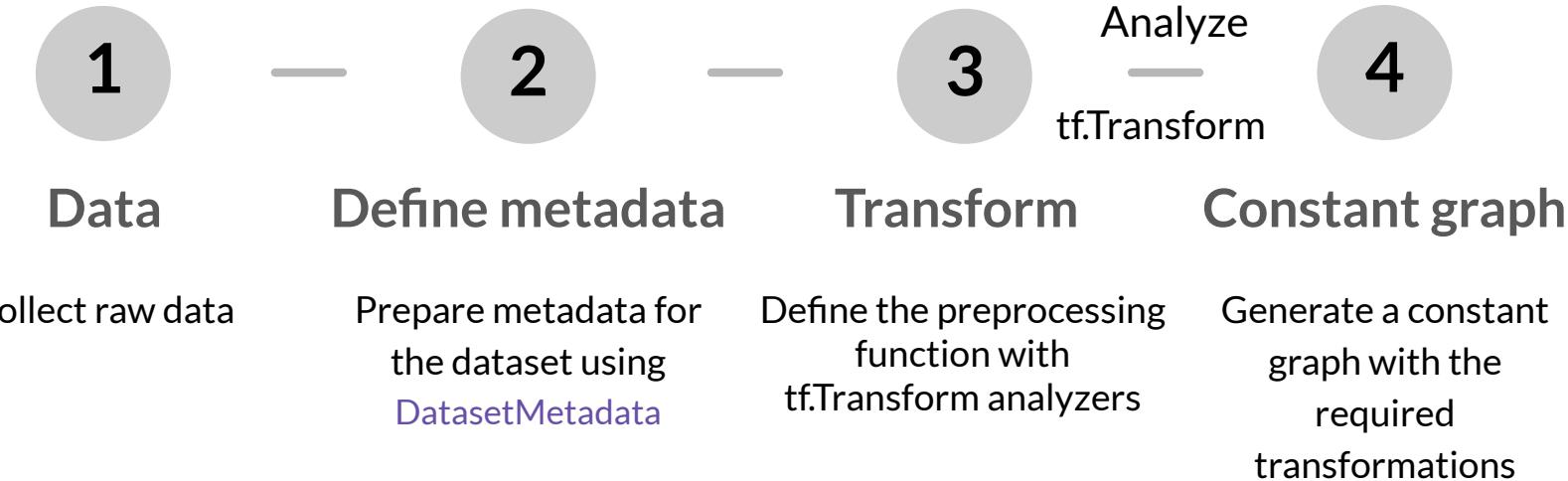
---



DeepLearning.AI

Hello World  
with `tf.Transform`

# Hello world with tf.Transform



# Collect raw samples (Data)

```
[  
  {'x': 1, 'y': 1, 's': 'hello'},  
  {'x': 2, 'y': 2, 's': 'world'},  
  {'x': 3, 'y': 3, 's': 'hello'}]  
]
```

# Inspect data and prepare metadata (Data)

```
from tensorflow_transform.tf_metadata import (
    dataset_metadata, dataset_schema)

raw_data_metadata = dataset_metadata.DatasetMetadata(
    dataset_schema.from_feature_spec({
        'y': tf.io.FixedLenFeature([], tf.float32),
        'x': tf.io.FixedLenFeature([], tf.float32),
        's': tf.io.FixedLenFeature([], tf.string)
    }))
```

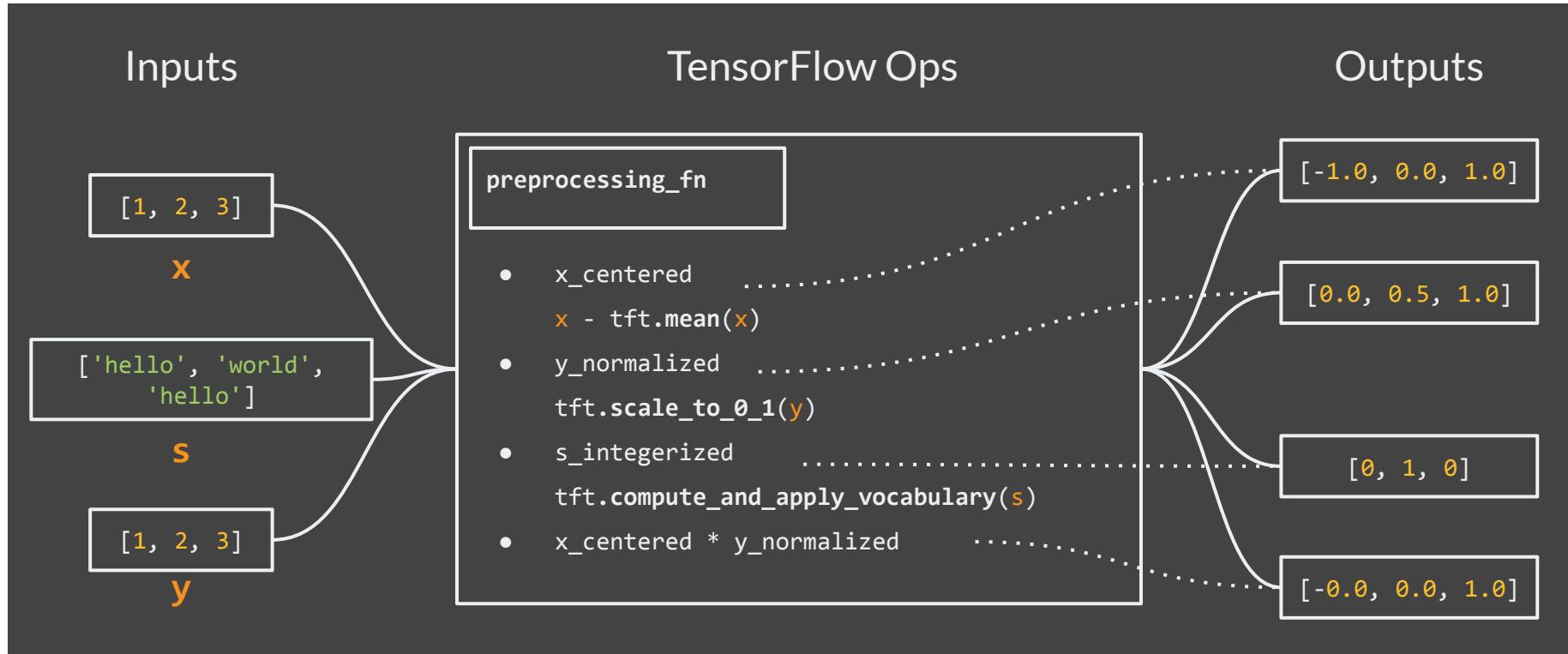
# Preprocessing data (Transform)

```
def preprocessing_fn(inputs):
    """Preprocess input columns into transformed columns."""
    x, y, s = inputs['x'], inputs['y'], inputs['s']
    x_centered = x - tft.mean(x)
    y_normalized = tft.scale_to_0_1(y)
    s_integerized = tft.compute_and_apply_vocabulary(s)
    x_centered_times_y_normalized = (x_centered * y_normalized)
```

# Preprocessing data (Transform)

```
return {  
    'x_centered': x_centered,  
    'y_normalized': y_normalized,  
    's_integerized': s_integerized,  
    'x_centered_times_y_normalized': x_centered_times_y_normalized,  
}
```

# Tensors in... tensors out



# Running the pipeline

```
def main():
    with tft_beam.Context(temp_dir=tempfile.mkdtemp()):
        transformed_dataset, transform_fn = (
            (raw_data, raw_data_metadata) | tft_beam.AnalyzeAndTransformDataset(
                preprocessing_fn))
```

# Running the pipeline

```
transformed_data, transformed_metadata = transformed_dataset

print('\nRaw data:\n{}'.format(pprint.pformat(raw_data)))
print('Transformed data:\n{}'.format(pprint.pformat(transformed_data)))

if __name__ == '__main__':
    main()
```

# Before transforming with tf.Transform

```
# Raw data:  
[{'s': 'hello', 'x': 1, 'y': 1},  
 {'s': 'world', 'x': 2, 'y': 2},  
 {'s': 'hello', 'x': 3, 'y': 3}]
```

# After transforming with tf.Transform

```
# After transform
[{'s_integerized': 0,
 'x_centered': -1.0,
 'x_centered_times_y_normalized': -0.0,
 'y_normalized': 0.0},
 {'s_integerized': 1,
 'x_centered': 0.0,
 'x_centered_times_y_normalized': 0.0,
 'y_normalized': 0.5},
 {'s_integerized': 0,
 'x_centered': 1.0,
 'x_centered_times_y_normalized': 1.0,
 'y_normalized': 1.0}]
```

# Key points

- `tf.Transform` allows the pre-processing of input data and creating features
- `tf.Transform` allows defining pre-processing pipelines and their execution using large-scale data processing frameworks
- In a TFX pipeline, the Transform component implements feature engineering using TensorFlow Transform



DeepLearning.AI

# Feature Selection

---

# Feature Spaces

# Outline

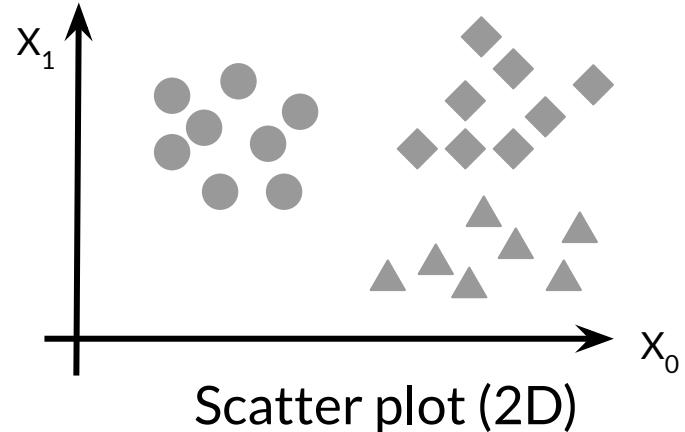
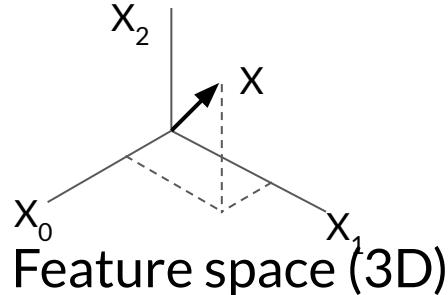
- Introduction to Feature Spaces
- Introduction to Feature Selection
- Filter Methods
- Wrapper Methods
- Embedded Methods

# Feature space

- N dimensional space defined by your N features
- Not including the target label

$$\mathbf{X} = \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_d \end{bmatrix}$$

Feature vector



# Feature space

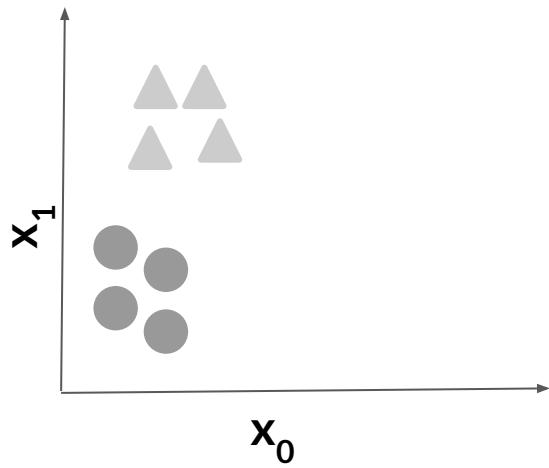


No. of Rooms $X_0$	Area $X_1$	Locality $X_2$	Price $Y$
5	1200 sq. ft	New York	\$40,000
6	1800 sq. ft	Texas	\$30,000

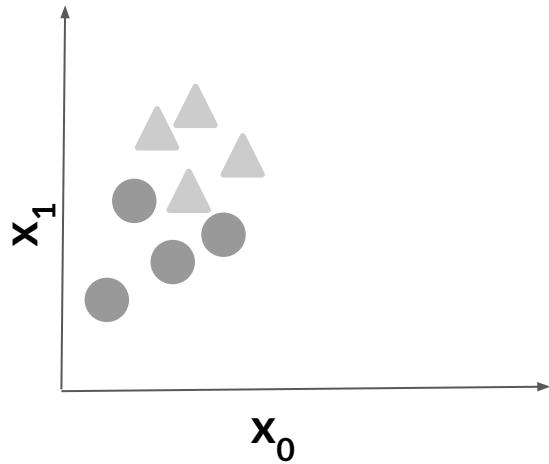
$$Y = f(X_0, X_1, X_2)$$

$f$  is your ML model acting on feature space  $X_0, X_1, X_2$

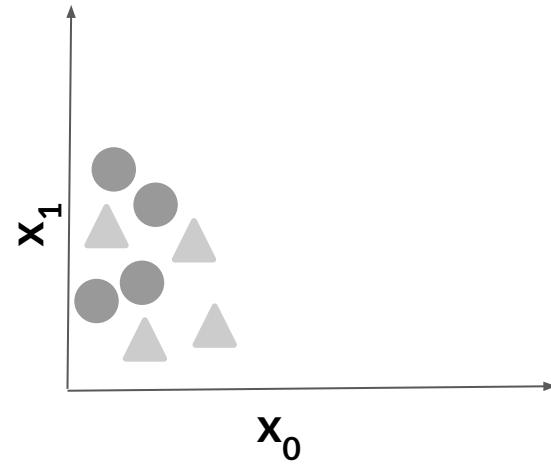
# 2D Feature space - Classification



Ideal

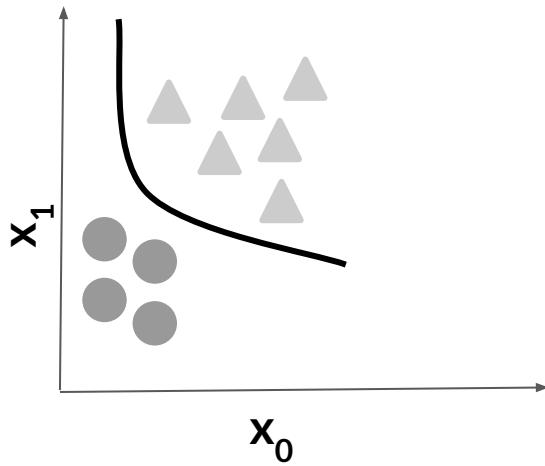


Realistic



Poor

# Drawing decision boundary



Model learns decision boundary

Boundary used to classify data points

# Feature space coverage

- Train/Eval datasets representative of the serving dataset
  - Same numerical ranges
  - Same classes
  - Similar characteristics for image data
  - Similar vocabulary, syntax, and semantics for NLP data

# Ensure feature space coverage

- Data affected by: seasonality, trend, drift.
- Serving data: new values in features and labels.
- Continuous monitoring, key for success!



DeepLearning.AI

# Feature Selection

---

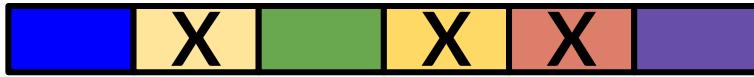
# Feature Selection

# Feature selection

All Features



Feature selection

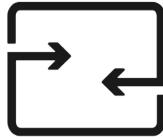


Useful features

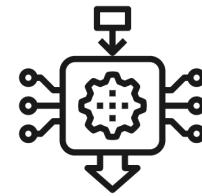


- Identify features that best represent the relationship
- Remove features that don't influence the outcome
- Reduce the size of the feature space
- Reduce the resource requirements and model complexity

# Why is feature selection needed?

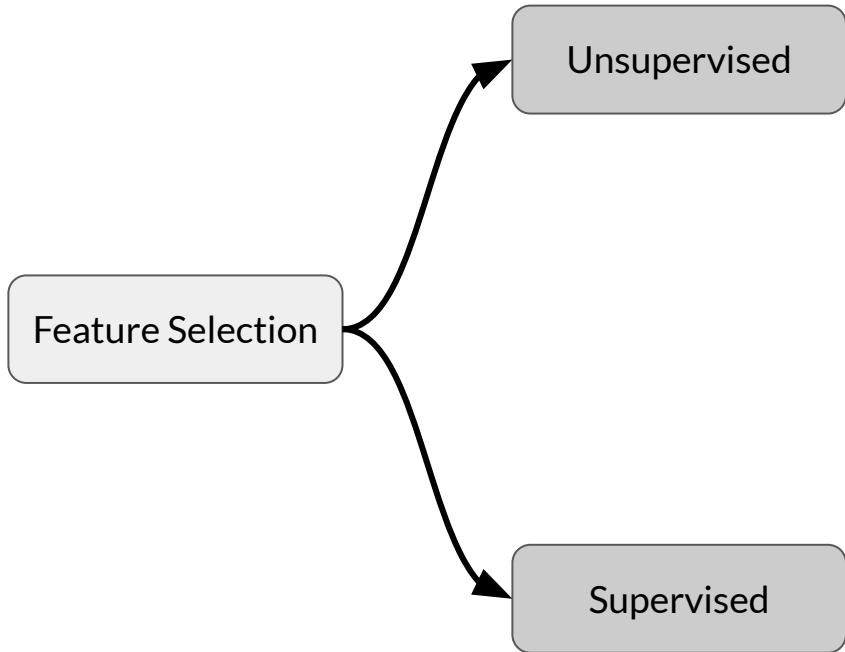


Reduce storage and I/O requirements



Minimize training and inference costs

# Feature selection methods



# Unsupervised feature selection

## 1. Unsupervised

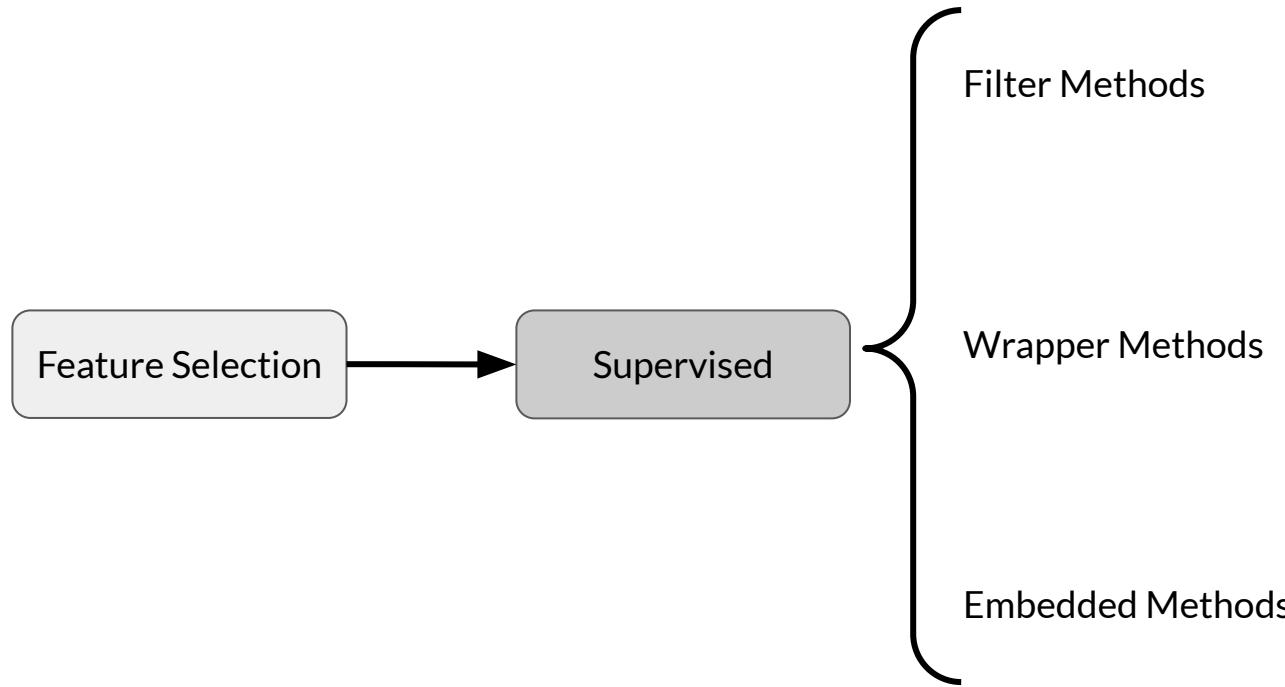
- Features-target variable relationship not considered
- Removes redundant features (correlation)

# Supervised feature selection

## 2. Supervised

- Uses features-target variable relationship
- Selects those contributing the most

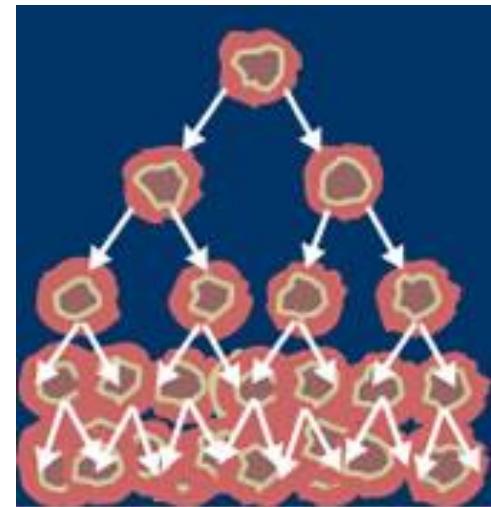
# Supervised methods



# Practical example

Feature selection techniques on Breast Cancer Dataset (Diagnostic)

Predicting whether tumour is benign or malignant.



# Feature list

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
842302	M	17.99	10.38	122.8	1001.0	0.1184	0.2776
concavity_mean	concavepoints_mean	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se
0.3001	0.1471	0.2419	0.07871	1.095	0.9053	8.589	153.4
smoothness_se	compactness_se	concavity_se	concavepoints_se	symmetry_se	fractal_dimension_se	radius-worst	texture_worst
0.0064	0.049	0.054	0.016	0.03	0.006	25.38	17.33
perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concavepoints_worst	symmetry_worst	fractal_dimension_worst
184.6	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.1189
							Unnamed:32
							NaN

# Performance evaluation

We train a **RandomForestClassifier** model in `sklearn.ensemble` on selected features

**Metrics (`sklearn.metrics`):**

Method	Feature Count	Accuracy	AUROC	Precision	Recall	F1 Score
All Features	30	0.967262	0.964912	0.931818	0.97619	0.953488

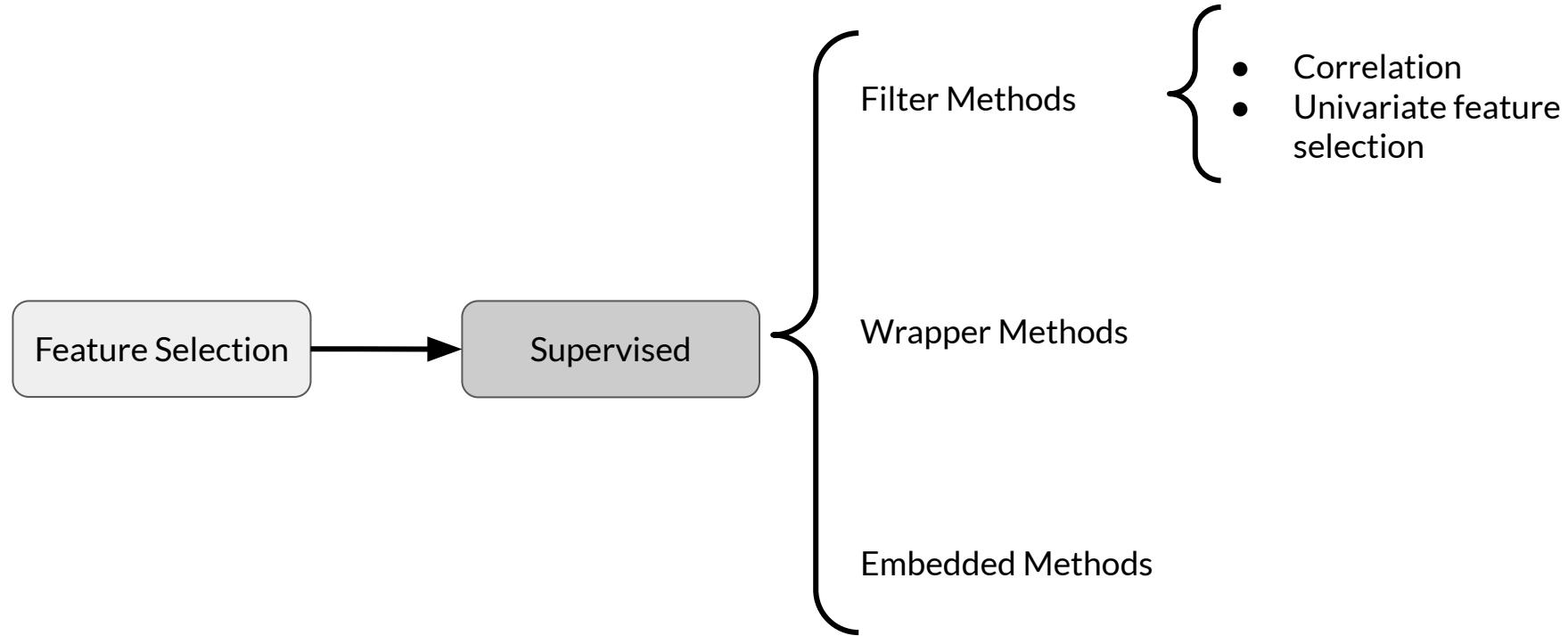
# Feature Selection



DeepLearning.AI

## Filter Methods

# Filter methods



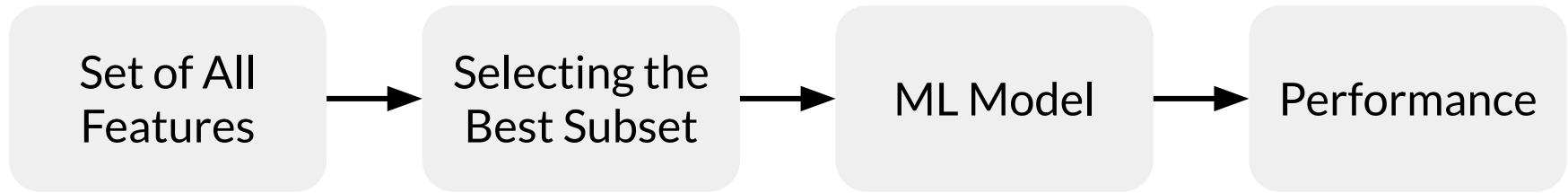
# Filter methods

- Correlated features are usually redundant
  - Remove them!

Popular filter methods:

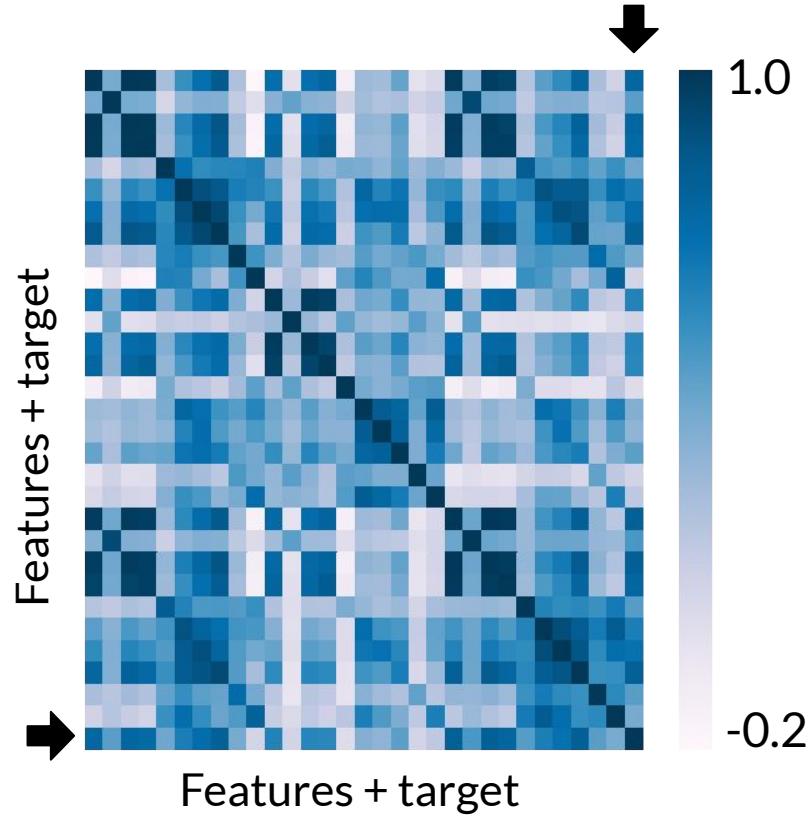
- Pearson Correlation
  - Between features, and between the features and the label
- Univariate Feature Selection

# Filter methods



# Correlation matrix

- Shows how features are related:
  - To each other (Bad)
  - And with target variable (Good)
- Falls in the range  $[-1, 1]$ 
  - 1 High positive correlation
  - -1 High negative correlation



# Feature comparison statistical tests

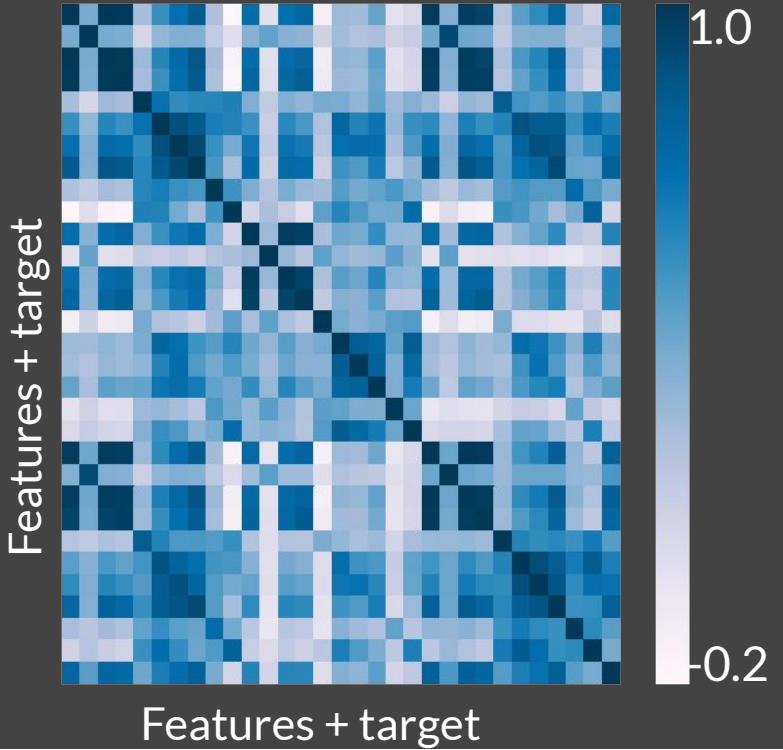
- Pearson's correlation: Linear relationships
- Kendall Tau Rank Correlation Coefficient: Monotonic relationships & small sample size
- Spearman's Rank Correlation Coefficient: Monotonic relationships

Other methods:

- Mutual information
- F-Test
- Chi-Squared test

# Determine correlation

```
# Pearson's correlation by default  
cor = df.corr()  
  
plt.figure(figsize=(20,20))  
# Seaborn  
sns.heatmap(cor, annot=True, cmap=plt.cm.PuBu)  
plt.show()
```



# Selecting features

```
cor_target = abs(cor["diagnosis_int"])

# Selecting highly correlated features as potential features to eliminate
relevant_features = cor_target[cor_target>0.2]
```

# Performance table

Method	Feature Count	Accuracy	AUROC	Precision	Recall	F1 Score
All Features	30	0.967262	0.964912	0.931818	0.97619	0.953488
Correlation	21	0.974206	0.973684	0.953488	0.97619	0.964706

**Best Result**

# Univariate feature selection in SKLearn

SKLearn Univariate feature selection routines:

1. **SelectKBest**
2. SelectPercentile
3. GenericUnivariateSelect

Statistical tests available:

- Regression: f\_regression, mutual\_info\_regression
- Classification: chi2, f\_classif, mutual\_info\_classif

# SelectKBest implementation

```
def univariate_selection():

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
                                                       test_size = 0.2,stratify=Y, random_state = 123)

    X_train_scaled = StandardScaler().fit_transform(X_train)
    X_test_scaled = StandardScaler().fit_transform(X_test)

    min_max_scaler = MinMaxScaler()
    Scaled_X = min_max_scaler.fit_transform(X_train_scaled)

    selector = SelectKBest(chi2, k=20) # Use Chi-Squared test
    X_new = selector.fit_transform(Scaled_X, Y_train)
    feature_idx = selector.get_support()
    feature_names = df.drop("diagnosis_int",axis = 1 ).columns[feature_idx]

    return feature_names
```

# Performance table

Method	Feature Count	Accuracy	AUROC	Precision	Recall	F1 Score
All Features	30	0.967262	0.964912	0.931818	0.97619	0.953488
Correlation	21	0.974206	0.973684	0.953488	0.97619	0.964706
Univariate ( $\text{Chi}^2$ )	20	0.960317	0.95614	0.91111	0.97619	0.94252

**Best Result**

# Feature Selection

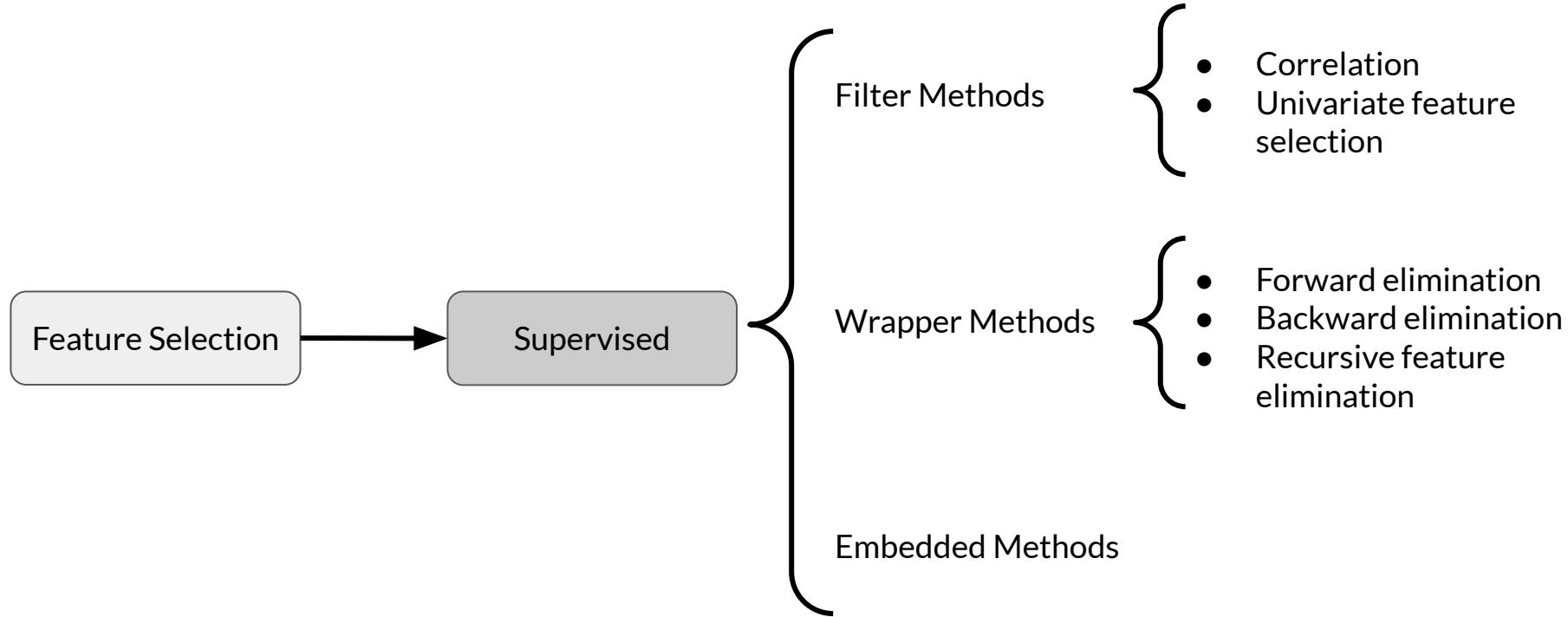
---



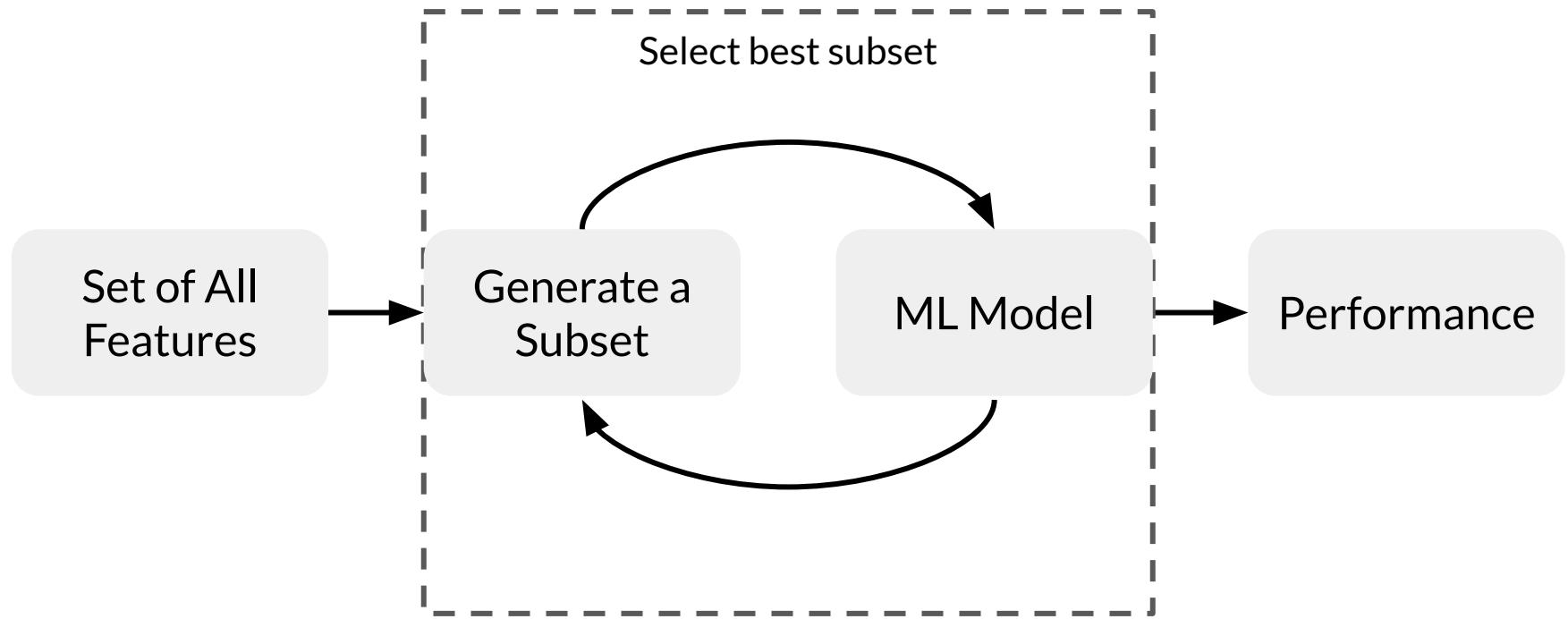
DeepLearning.AI

# Wrapper Methods

# Wrapper methods



# Wrapper methods



# Wrapper methods

Popular wrapper methods

1. Forward Selection
2. Backward Selection
3. Recursive Feature Elimination

# Forward selection

1. Iterative, greedy method
2. Starts with 1 feature
3. Evaluate model performance when **adding** each of the additional features, one at a time
4. Add next feature that gives the best performance
5. Repeat until there is no improvement

# Backward elimination

1. Start with all features
2. Evaluate model performance when **removing** each of the included features, one at a time
3. Remove next feature that gives the best performance
4. Repeat until there is no improvement

# Recursive feature elimination (RFE)

1. Select a model to use for evaluating feature importance
2. Select the desired number of features
3. Fit the model
4. Rank features by importance
5. Discard least important features
6. Repeat until the desired number of features remains

# Recursive feature elimination

```
def run_rfe():

    X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.2, random_state = 0)

    X_train_scaled = StandardScaler().fit_transform(X_train)
    X_test_scaled = StandardScaler().fit_transform(X_test)

    model = RandomForestClassifier(criterion='entropy', random_state=47)
    rfe = RFE(model, 20)
    rfe = rfe.fit(X_train_scaled, y_train)
    feature_names = df.drop("diagnosis_int",axis = 1 ).columns[rfe.get_support()]
    return feature_names

rfe_feature_names = run_rfe()

rfe_eval_df = evaluate_model_on_features(df[rfe_feature_names], Y)
rfe_eval_df.head()
```

# Performance table

Method	Feature Count	Accuracy	AUROC	Precision	Recall	F1 Score
All Features	30	0.96726	0.96491	0.931818	0.97619	0.953488
Correlation	21	0.97420	0.97368	0.9534883	0.97619	0.964705
Univariate ( $\text{Chi}^2$ )	20	0.96031	0.95614	0.91111	0.97619	0.94252
Recursive Feature Elimination	20	0.97420	0.97368	0.953488	0.97619	0.964706

**Best Result**

# Feature Selection

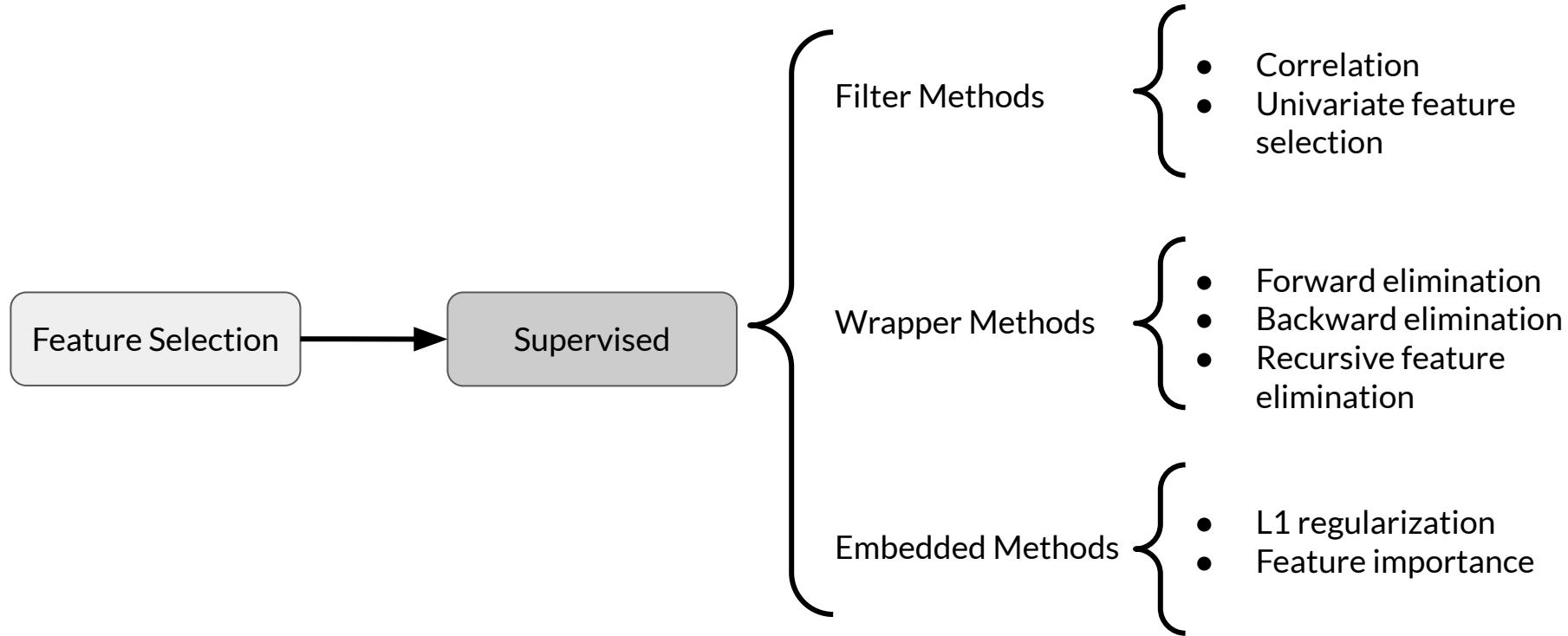
---



DeepLearning.AI

# Embedded Methods

# Embedded methods



# Feature importance

- Assigns scores for each feature in data
- Discard features scored lower by feature importance

# Feature importance with SKLearn

- Feature Importance class is in-built in Tree Based Models (eg., `RandomForestClassifier`)
- Feature importance is available as a property `feature_importances_`
- *We can then use `SelectFromModel` to select features from the trained model based on assigned feature importances.*

# Extracting feature importance

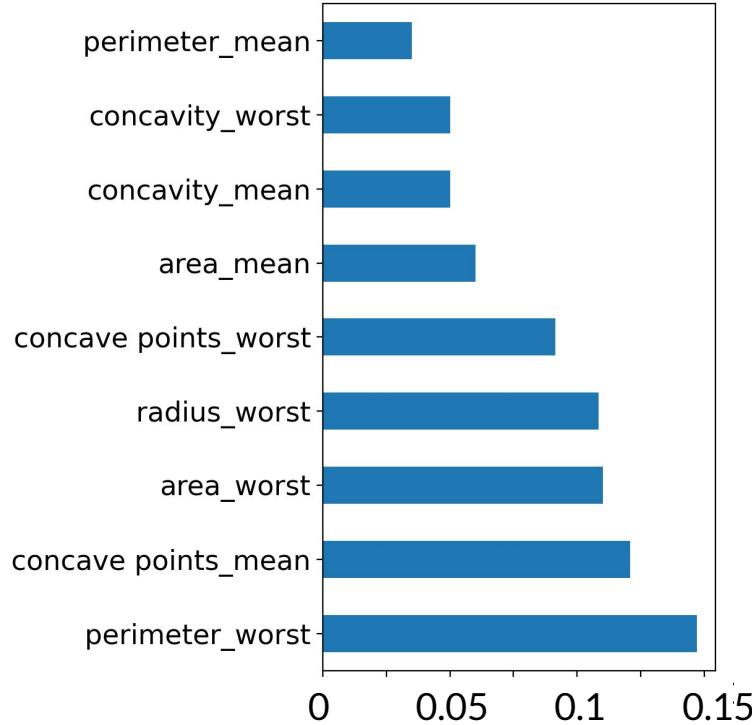
```
def feature_importances_from_tree_based_model_():

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
                                                       stratify=Y, random_state = 123)
    model = RandomForestClassifier()
    model = model.fit(X_train,Y_train)

    feat_importances = pd.Series(model.feature_importances_, index=X.columns)
    feat_importances.nlargest(10).plot(kind='barh')
    plt.show()

    return model
```

# Feature importance plot



# Select features based on importance

```
def select_features_from_model(model):  
  
    model = SelectFromModel(model, prefit=True, threshold=0.012)  
  
    feature_idx = model.get_support()  
    feature_names = df.drop("diagnosis_int", 1).columns[feature_idx]  
    return feature_names
```

# Tying together and evaluation

```
# Calculate and plot feature importances
model = feature_importances_from_tree_based_model_()

# Select features based on feature importances
feature_imp_feature_names = select_features_from_model(model)
```

# Performance table

Method	Feature Count	Accuracy	ROC	Precision	Recall	F1 Score
All Features	30	0.96726	0.964912	0.931818	0.9761900	0.953488
Correlation	21	0.97420	0.973684	0.953488	0.9761904	0.964705
Univariate Feature Selection	20	0.96031	0.95614	0.91111	0.97619	0.94252
Recursive Feature Elimination	20	0.9742	0.973684	0.953488	0.97619	0.964706
Feature Importance	14	0.96726	0.96491	0.931818	0.97619	0.953488

**Best Result**

# Review

- Intro to Preprocessing
- Feature Engineering
- Preprocessing Data at Scale
  - TensorFlow Transform
- Feature Spaces
- Feature Selection
  - Filter Methods
  - Wrapper Methods
  - Embedded Methods