

# Analyze



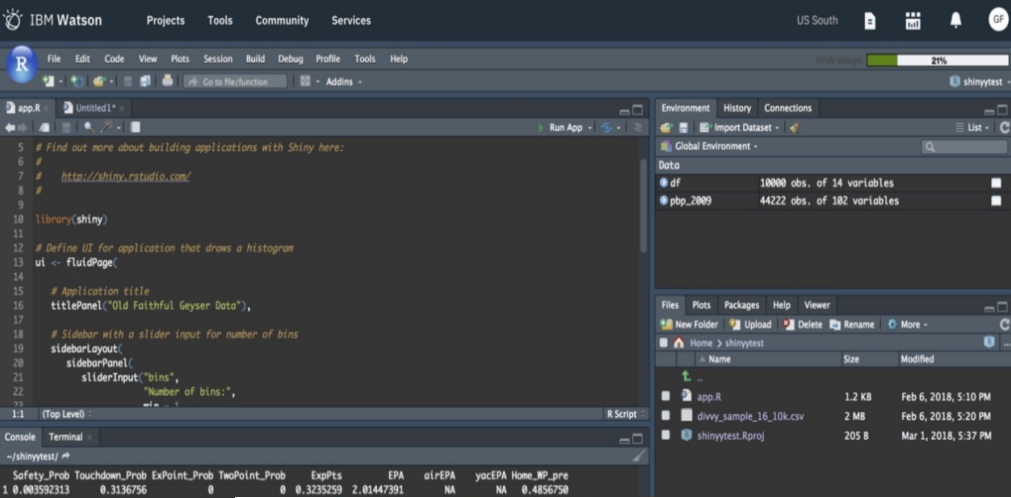
# Agenda

- Identify how you can analyze data in IBM Cloud Pak for Data
- Analyze data using notebooks  
Identify other tools that you can use to analyze data
- Automatically analyze your data using the AutoAI tool
- Deploy machine learning (ML) models



# Programmatic Manipulation

- Jupyter Notebooks and RStudio
  - **Python**, Scala and R
- Use / Install open source libraries
  - Data structure/manipulation: **Pandas**, Numpy
  - Visualizations: **Matplotlib** (most popular), **Brunel**, **seaborn**, **bokeh**. **Plotly**, **ggplot**, **pygal**, **PixieDust**, etc
  - Algorithmic: SciPy (integrals, calcs), Scikit-learn, Statsmodels, etc



The screenshot shows the IBM Watson Studio interface. The main editor displays R code for a Shiny application. The code includes comments about finding more about building applications with Shiny, a link to the Shiny website, and the installation of the 'shiny' library. It defines a UI for an application titled 'Old Faithful Geyser Data', featuring a sidebar with a slider input for the number of bins. The console output shows a dataset with columns: Safety\_Prob, Touchdown\_Prob, ExPoint\_Prob, ExpPts, EPA, alrEPA, yocEPA, Home\_MP\_pre, Away\_MP\_pre, Home\_MP\_post, Away\_MP\_post, and Win. The dataset has 6 rows of data.

```
# Find out more about building applications with Shiny here:
# http://shiny.rstudio.com/
library(shiny)

# Define UI for application that draws a histogram
ui <- fluidPage(

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of bins
  sidebarPanel(
    sliderInput("bins",
      "Number of bins:",
      min = 1,
      max = 10,
      value = 5)
  )
)
```

Console Output:

```
--shinytest1--
Safety_Prob Touchdown_Prob ExPoint_Prob ExpPts EPA alrEPA yocEPA Home_MP_pre
1 0.003592313 0.3136796 0 0.3235259 2.01447391 NA NA 0.4856750
2 0.003637510 0.4238014 0 0.4214784 0.4899122 0.546
3 0.003826829 0.4214784 0 0.4892869 0.551
4 0.004776485 0.3186208 0 0.5387829 0.510
5 0.006404003 0.2081109 0 0.4418706 0.461
6 0.003899273 0.2163574 0 0.4215468 0.441
Away_MP_pre Home_MP_post Away_MP_post Win
1 0.5143250 0.5464328 0.4535672 0.485
2 0.4535672 0.5510878 0.4499122 0.546
3 0.4489122 0.5107931 0.4892869 0.551
4 0.4892869 0.4612121 0.5387829 0.510
5 0.5387829 0.5589294 0.4418706 0.461
6 0.4418706 0.5784532 0.4215468 0.441
```

Encode categorical columns

```
# Discrete value integer encoder
label_encoder = preprocessing.LabelEncoder()

# State, international plans and voice mail plan are strings and we want them as integers
df['state'] = label_encoder.fit_transform(df['state'])
df['international plan'] = label_encoder.fit_transform(df['international plan'])
df['voice mail plan'] = label_encoder.fit_transform(df['voice mail plan'])

print(df.dtypes)
```

state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge	customer service calls	churn
int64	int64	int64	object	int64	int64	int64	float64	int64	float64	float64	int64	float64	float64	int64	float64	float64	int64	float64	int64	bool

dtype: object


# SPSS Modeler

**Cleansing:** fix or remove data that is incorrect, incomplete, improperly formatted, or duplicated


**Shaping:** customize data by filtering, sorting, combining, or removing columns, and performing operations

**Visualization, statistics and processing**


**Original data set is not modified, changes pushed to output file or data source**




Chart














Spreadsheet

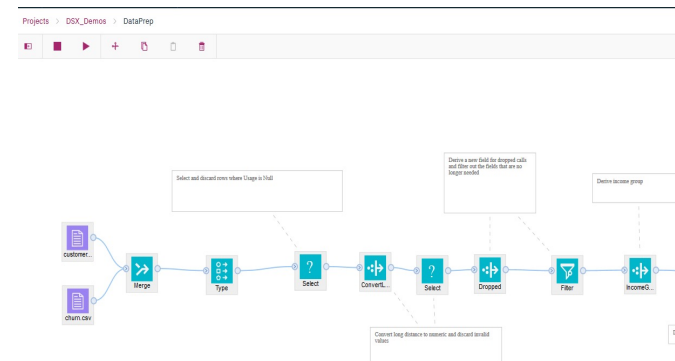


Data Audit



Preferences

	Audit	Quality	Statistics	Pearson Correlations		
	FIELD	GRAPH	MEASUREMENT	MIN	MAX	SUM
1	ID		continuous	1	3,709	3,682,8
2	Gender		discrete	--	--	--
3	Status		discrete	--	--	--
4	Children		continuous	0	2	2,295
5	Est Income		continuous	96.33	120,000	102,881
6	Car Owner		discrete	--	--	--
7	Age		continuous	12.327	77	85,539
8	LongDistance		discrete	--	--	--
9	International		continuous	0	9.7	2,417.12
10	Local		continuous	0.68	332.46	118,827
11	DroppedPeak		continuous	0	8	2,221



## Auto Data Prep node

Preparing data for analysis is one of the most important steps in analyzing your data and identifying fixes, screening out fields that are not useful, and using intelligent screening techniques. You can use the algorithm in the Auto Data Prep node to screen out fields before they are made and accept or reject them as you want.

## Type node

Field properties can be specified in a source node or in a separate node.

## Filter node

You can rename or exclude fields at any point in a stream. For example, you can filter out the **K** (potassium) field regardless of which node it is accessed from.

## Derive node

One of the most powerful features in Watson Studio is the ability to create new fields from existing fields. Several derivations, such as extracting a customer ID from a string, can be performed, using a variety of field operations.

## Filler node

Filler nodes are used to replace field values and change storage type. You can choose to replace all blanks or null values with a specific value.

## Reclassify node

The Reclassify node enables the transformation from one set of values to another.

## Binning node

The Binning node enables you to automatically create new nodes from a continuous income field into a new categorical field containing discrete values in order to preserve the strength of the original association between the two fields.

## RFM Analysis node

You can use the Recency, Frequency, Monetary (RFM) Analysis node to analyze your data by recency, how often they purchased (frequency), and monetary value.

## Ensemble node

The Ensemble node combines two or more model nuggets to create a single model. Limitations in individual models may be avoided, resulting in a more accurate model.

## Partition node

Partition nodes are used to generate a partition field that splits the data into two groups: one for generating the model and a separate sample to test it, you can use the Set to Flag node.

## Set to Flag node

Use the Set to Flag node to derive flag fields based on the categorical values of a field.

## Restructure node

The Restructure node can be used to generate multiple fields from a single field. The functionality of this node is similar to that of the Transpose node. You can then perform aggregation on the new fields. (You can also use the Restructure node to create flag fields.)

## Transpose node

By default, columns are fields and rows are records or observations. The Transpose node swaps columns and rows so that rows become fields.

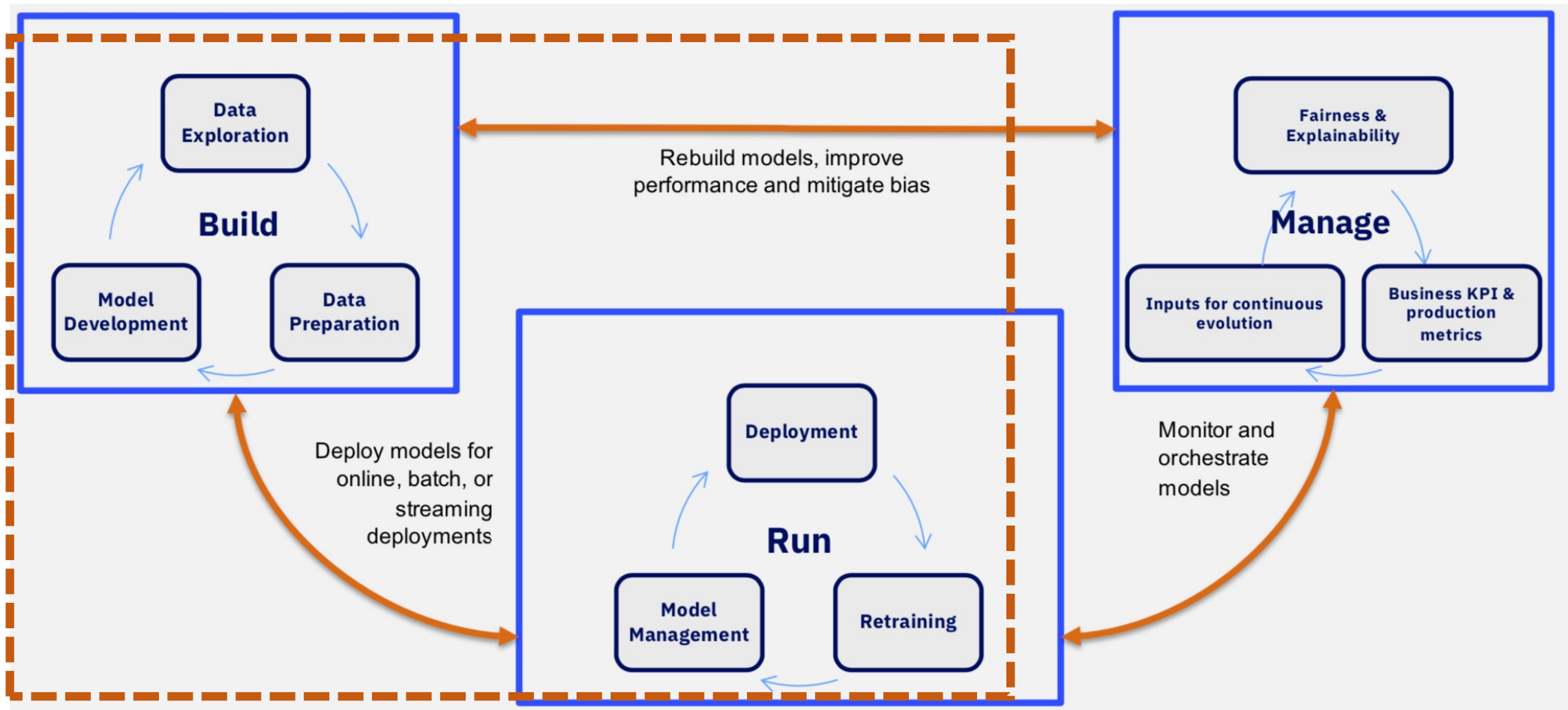
## Field Reorder node

With the Field Reorder node, you can define the natural order of the fields in the data stream.

## History node

History nodes are most often used for sequential data, such as time series data.

# Data & AI Workflow



# Machine Learning (ML)

## Overview



# What is Machine Learning ?

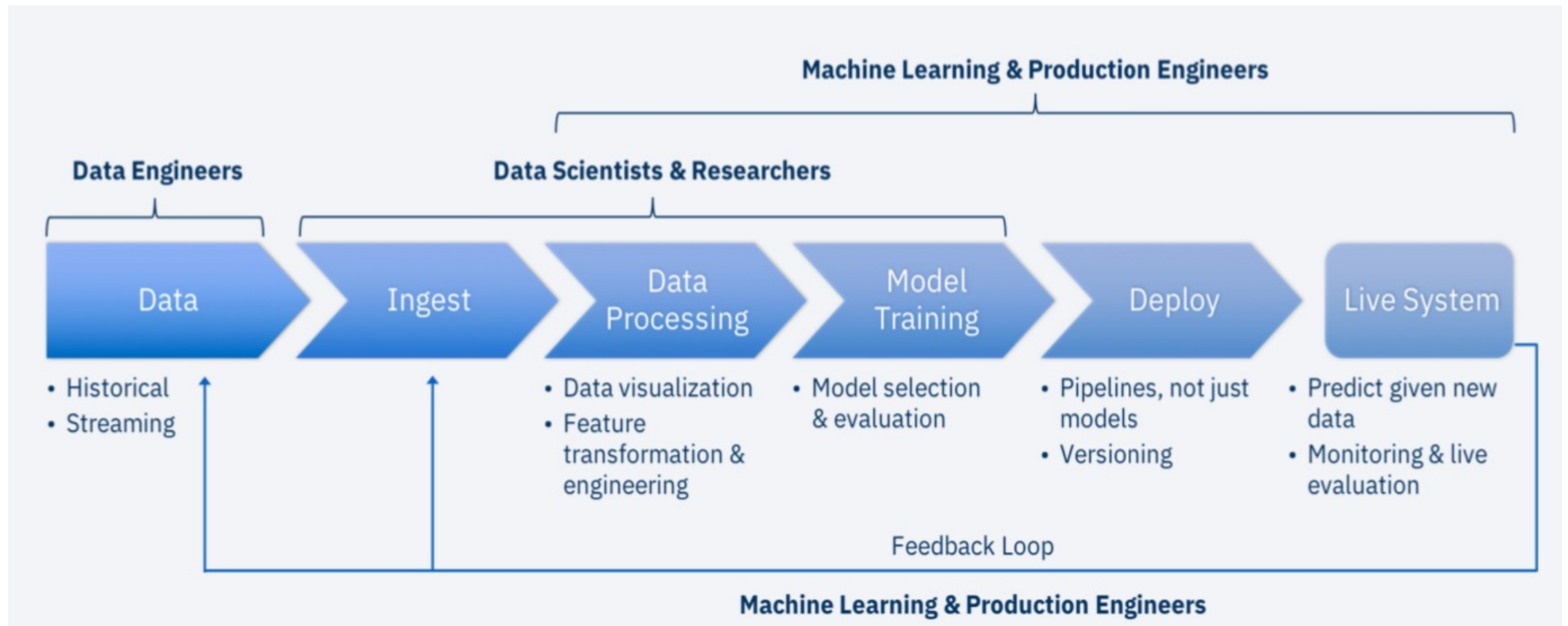
- *“A computer program is said to learn from experience  $E$  with respect to some task  $T$  and some performance measure  $P$ , if its performance on  $T$ , as measured by  $P$ , improves with experience  $E$ .” -- Tom Mitchell, Carnegie Mellon University*
- Learn from data to make predictions
- Computers apply statistical learning techniques to identify patterns in data and make predictions.  
Learn from historical data to make predictions about the future
- Use example data or past experience to solve a given problem.

Applied Machine Learning seeks to learn from historical data to make predictions about the future, in order to make decisions



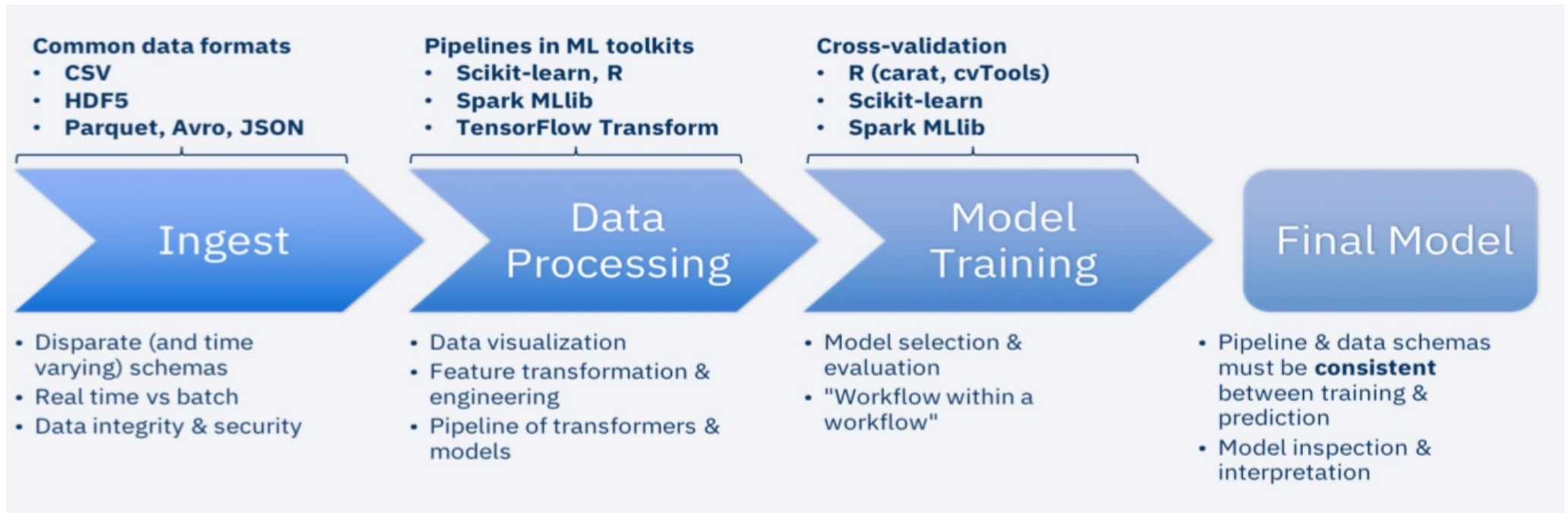
Credit: Nick Pentreath Principal Engineer, IBM Center for Open-Source Data & AI Technologies

# Machine Learning Across Teams

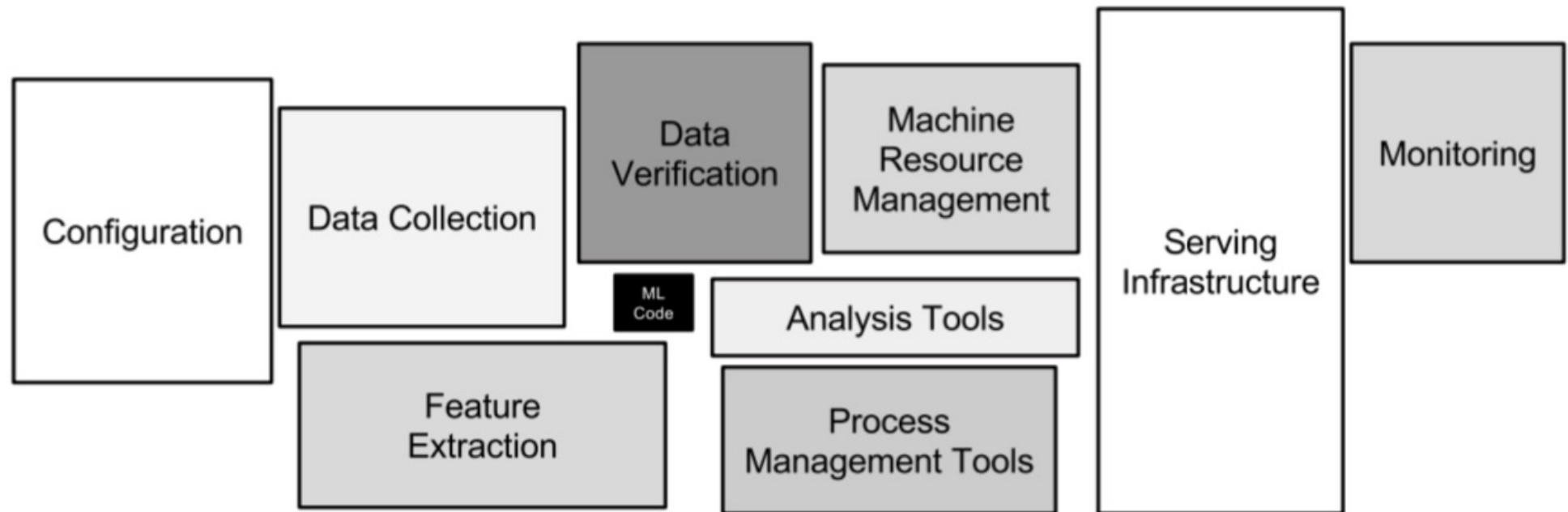




... tools



# Just one piece of the puzzle



[Reference](#)

# Example Linear Regression

```
1 from sklearn.linear_model import LinearRegression
2 from sklearn.datasets import load_boston
3 import matplotlib.pyplot as plt
4 from sklearn.model_selection import train_test_split
5 from sklearn.metrics import mean_squared_error, r2_score
6
7 boston = load_boston()
8 X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target)
9
10 # Create a new Linear Regression Model
11 LR_model = LinearRegression()
12
13 # Train the model
14 LR_model.fit(X_train, y_train)
15
16 # store actual and predicted data to draw chart
17 predicted = LR_model.predict(X_test)
18 actual = y_test
19
20
21 # The coefficients
22 print('Coefficients: \n', LR_model.coef_)
23 # The mean squared error
24 print("Mean squared error: %.2f"
25       % mean_squared_error(actual, predicted))
26 # Explained variance score: 1 is perfect prediction
27 print('Variance score: %.2f' % r2_score(actual, predicted))
```

```
# we will use WML to work with IBM Machine Learning Service
from watson_machine_learning_client import WatsonMachineLearni

# Grab your credentials from the Watson Service section in Wat
wml_credentials = {
}

# Instantiate WatsonMachineLearningAPIClient
from watson_machine_learning_client import WatsonMachineLearni
client = WatsonMachineLearningAPIClient( wml_credentials )

# store the model
published_model = client.repository.store_model(model=LR_model
                                                meta_props={'r
                                                training_data=


import json

# grab the model from IBM Cloud
published_model_uid = client.repository.get_model_uid(published_

# create a new deployment for the model
model_deployed = client.deployments.create(published_model_uid,

# get the scoring endpoint
scoring_endpoint = client.deployments.get_scoring_url(model_depl
print(scoring_endpoint)

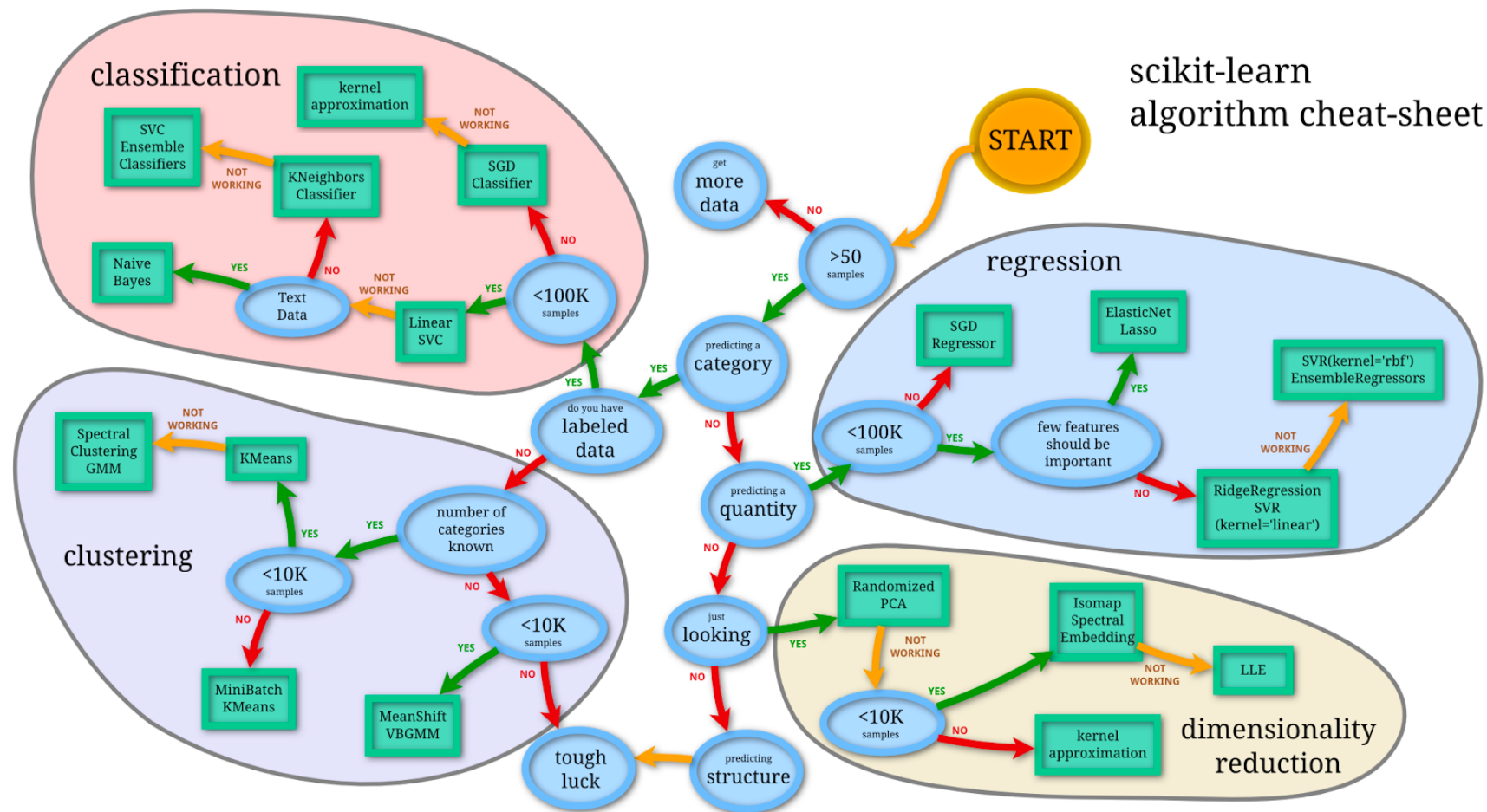
# use the scoring endpoint to predict house median price some tes
scoring_payload = {"values": [list(X_test[0]), list(X_test[1])]}
predictions = client.deployments.score(scoring_endpoint, scoring
print(json.dumps(predictions, indent=2))
```



# ML Model Building and Deploying

IBM Cognitive Applications © 2019 IBM Corporation

# Andreas Mueller – ML workflow graph





# Code / Notebook

- Anaconda based default or customized environments for Python (3.5) and R
  - Use Spark environment for Scala
- Don't forget : File → Stop Kernel

## Build a Machine Learning Model with Spark ML

```
In [4]: from pyspark.ml import Pipeline
        from pyspark.ml.regression import LinearRegression
        from pyspark.ml.feature import VectorAssembler
```

```
In [5]: assembler = VectorAssembler(inputCols=['SquareFeet', 'Bedrooms'], outputCol="features")
        lr = LinearRegression(labelCol='Price', featuresCol='features')
        pipeline = Pipeline(stages=[assembler, lr])
        model = pipeline.fit(df)
```

### TensorFlow

- Version 1.5
- Version 1.11 in an Anaconda 5.2.0 environment

### Spark MLlib

Spark 2.1

### Caffe

Version 1.0

### Predictive Model Markup Language (PMML)

Version 3.0 to 4.3

### XGBoost

- XGBoost 0.6a2 and 0.71 in an Anaconda 4.2.x environment with scikit-learn 0.17 and Python 3.5
- XGBoost 0.80 in an Anaconda 5.0.1 environment with scikit-learn 0.19 and Python 3.5

### scikit-learn

- scikit-learn 0.17 on Anaconda 4.2.x for Python 3.5 Runtime
- scikit-learn 0.19 on Anaconda 5.0.0 for Python 3.5 Runtime

### PyTorch

Versions: 0.3, 0.4.1, 1.0

### Keras

- Version 2.1.3 with Tensorflow version 1.5
- Version 2.2.4 with TensorFlow version 1.11 in an Anaconda 5.2.0 environment

### IBM SPSS Modeler

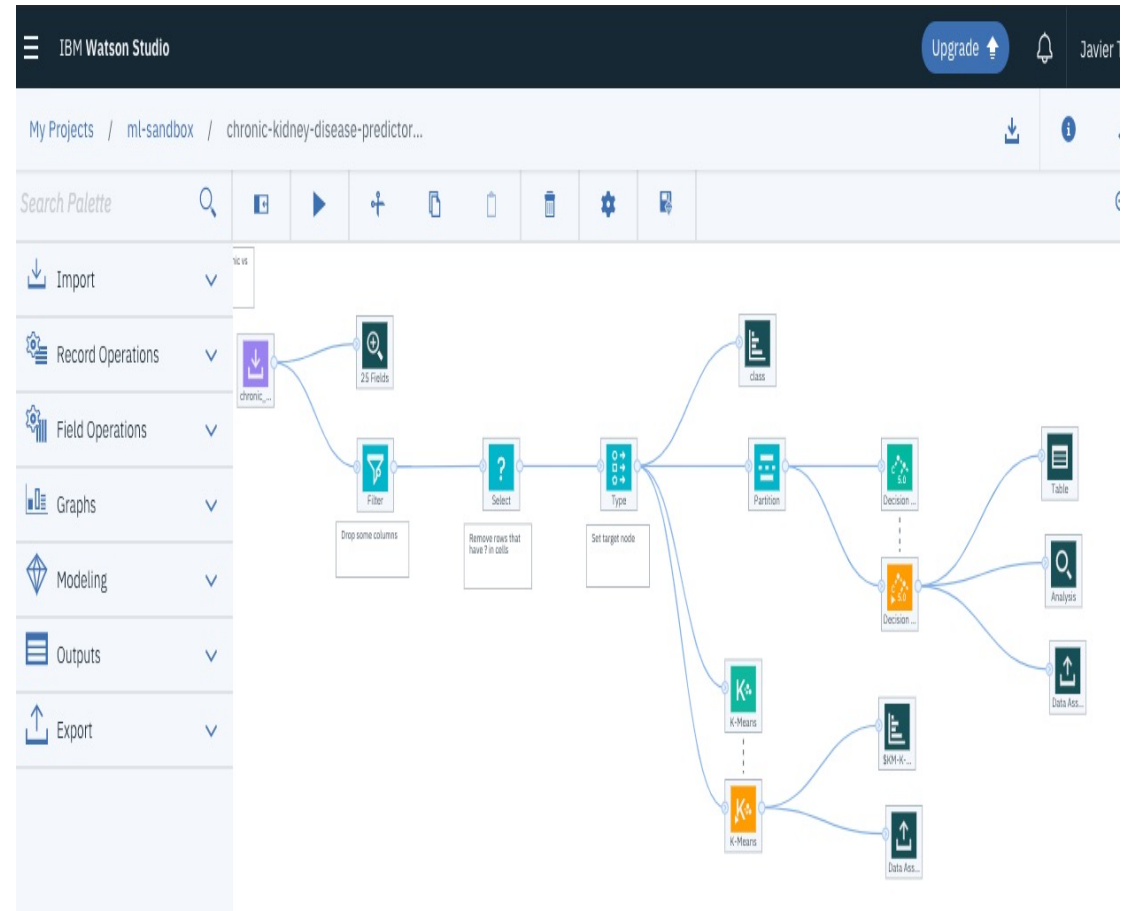
- IBM SPSS Modeler 17.1
- IBM SPSS Modeler 18.0

As of June 2019



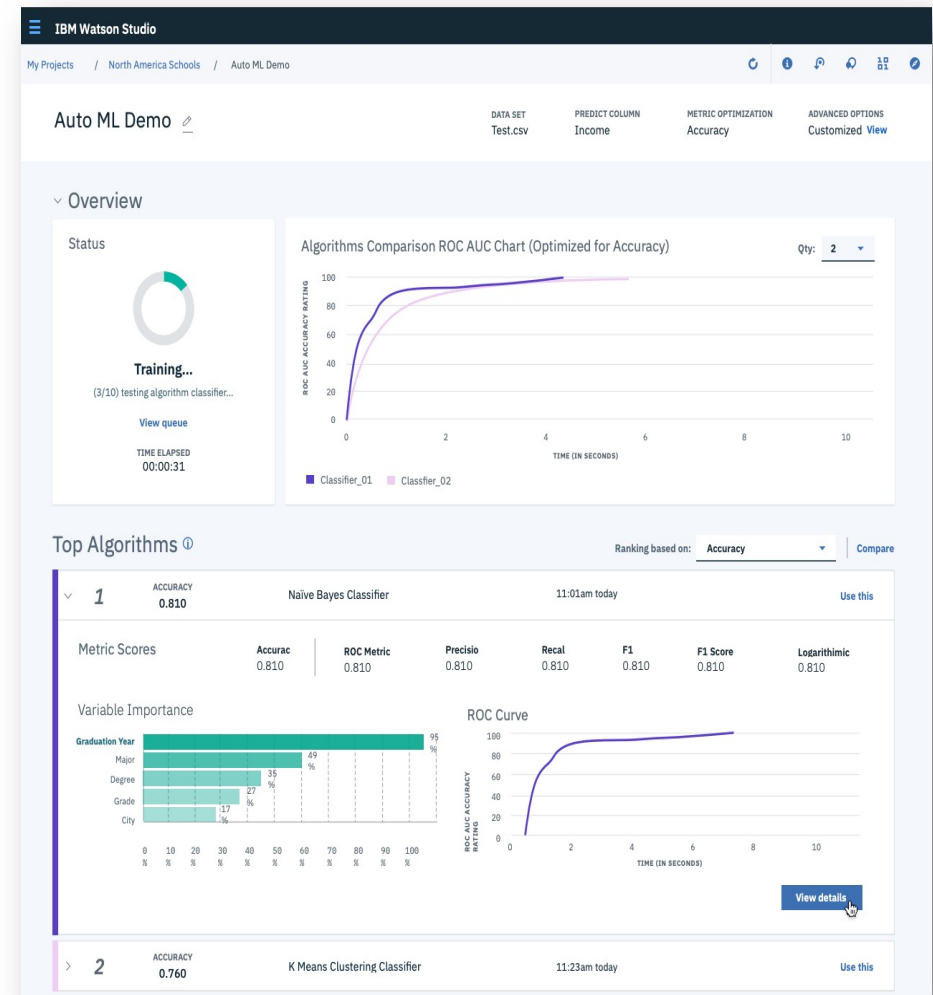
# Visual Assembly

- Drag and Drop visual assembly of pipelines using either SPSS Modeler runtime or Spark.
  - Data asset nodes to bring in data
  - Data preparation nodes to modify data (or use auto data prep node)
  - Modeling nodes to develop models (decision trees, random forest, linear regression ,etc).

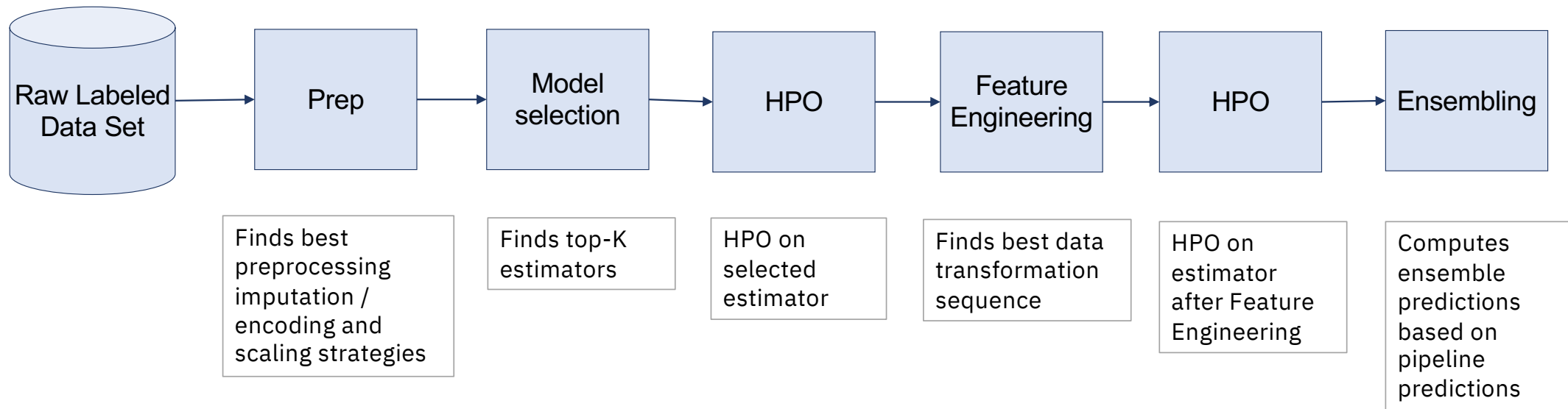


# AutoAI

- Encompasses three approaches to simplify model creation:
  - Transfer knowledge learning in one deep learning system to apply to a different domain (Watson Services)
  - **AutoAI Pipeline Optimization to auto clean data, engineer features, and complete HPO to find the optimal end to end pipeline**
  - Neural Network Search to generate custom deep neural network through searching the best architectures for the input data.
- Training feedback visualizations provide real-time results to see model performance
- One-click deployment to Watson Machine Learning



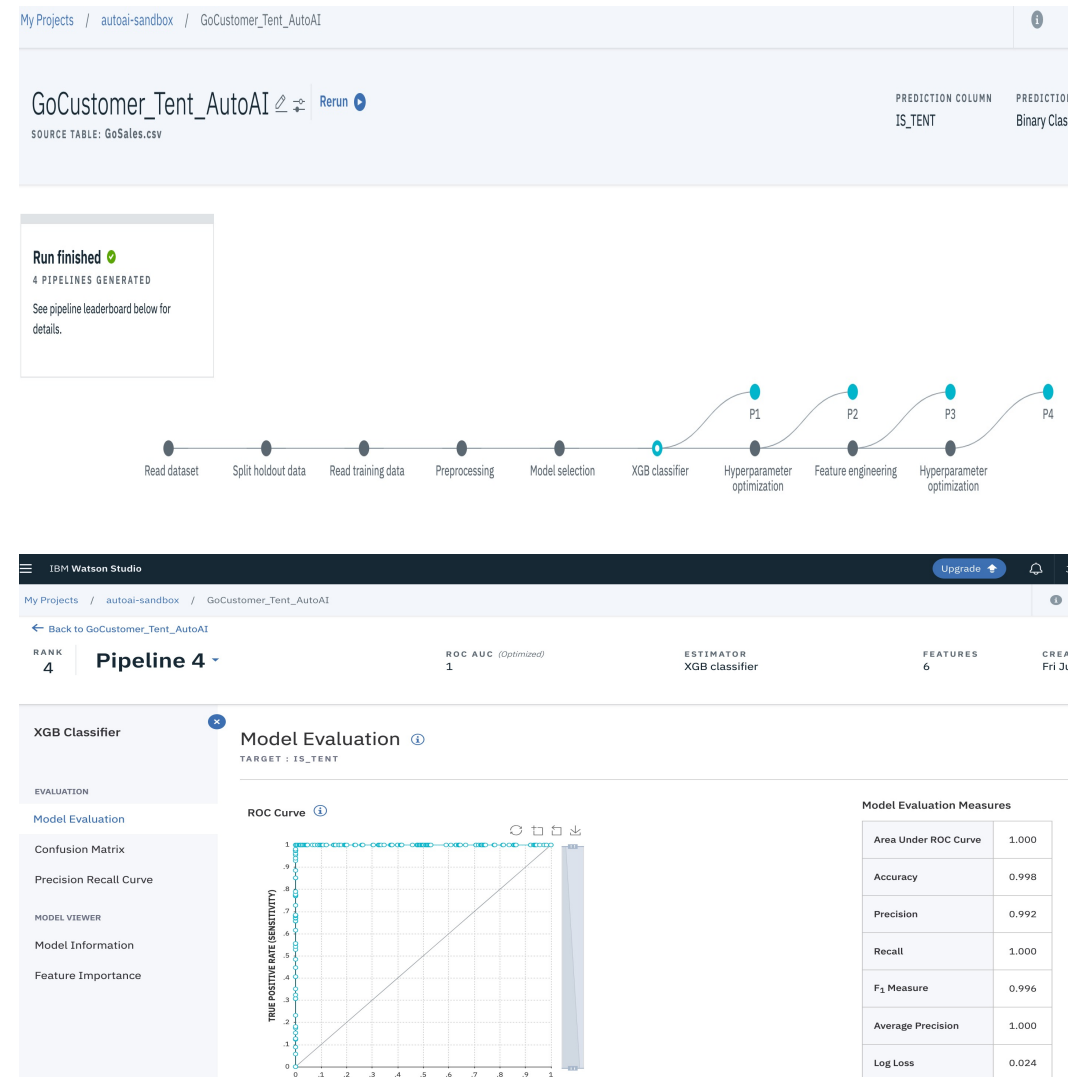
# AutoAI: How does it work ?



# AutoAI: Pipelines

- Builds models using sklearn
- 4 pipelines per estimator
  - Baseline sklearn estimator
  - HPO optimized estimator
  - Feature engineering
  - Feature engineering + HPO

The leaderboard contains model pipelines ranked by cross-validation scores (defaults = AUC score, Accuracy, and RMSE)



# Deploying Models

- Watson Machine Learning can host model (creates scoring endpoint)
- Deployments via Python Client, REST API, Watson Studio GUI
  - Online web service
  - Batch
  - CoreML
- Deploy models directly or Python functions to encapsulate model + logic
  - Preprocessing data, error handling, custom model orchestration, etc.

The screenshot displays the IBM Watson Studio interface. The top navigation bar includes 'My Projects' and 'ml-sandbox'. The main navigation tabs are 'Overview', 'Assets', 'Environments', 'Bookmarks', 'Deployments', 'Access Control', and 'Settings'. The 'Deployments' tab is active, showing a list of deployments with columns for NAME and a selection icon. The list includes: TentPredictionAutoAIXGBoostDeployment, Titanic-Survival-Classifer-WS, JRT Spark German Risk Deployment - Final, IRIS\_Model\_WS, Spark German Risk Deployment - Final (highlighted), Oil Price RNN Deployment, ModelerSparkChurnDT-Deployed, ChurnModel\_WS, and House Prices Deployment. Below the list, the breadcrumb path is 'My Projects / ml-sandbox / JRT Spark German Risk Model - Fi... / JRT Spa'. The detailed view for 'JRT Spark German Risk Deployment - Final' is shown, with tabs for 'Overview', 'Implementation' (selected), and 'Test'. The 'Implementation' tab contains a table with the following details:

Implementation	
Scoring End-point	https://us-south.ml.cloud.ibm.com/v3/wml
Authorization: Bearer <token>	Review the <a href="#">WML authentication</a> documents
ML-Instance-ID	The "ML-Instance-ID" HTTP header must b
Content-type: application/json	Required if the request body is sent in JSO

Below the table, the 'Code Snippets' section is visible, with tabs for 'cURL', 'Java', 'JavaScript', 'Python', and 'Scala'. The 'cURL' tab is selected, showing a code snippet:

```
# TODO: manually define and pass values to be scored below
curl -X POST --header 'Content-Type: application/json' --h
```

# Thank You

- Binu Midhun
- Email: [bmidhun1@in.ibm.com](mailto:bmidhun1@in.ibm.com)
- Twitter: @binu\_midhun 