# * MLOps *

## * Experiment Tracking in MLOps

→ Process of recording all details of machine learning experiments. This includes configurations, code, versions datasets, metrics & results.

metrics → Performance [Accuracy, F1Score, R2 Score]
Loss → Error. [MAE, RMSE]

## * Why MLFlow for MLOps

1. Reproducibility
- Ensures experiments can be repeated with same settings, helping to verify results.

2. Comparison
- Make it easy to compare different models & experiments to find best & performing one.

3. Collaboration
- Allows team members to share and review each other's work, enhancing teamwork.

4. Efficiency
- Saves times by avoiding repeated work and helps in quickly finding the best model settings.

5. Auditability
- keeps a history of all experiments, useful for tracking progress and compliance purposes.

## * Where Does MLflow Fit...?

1. Experimentation:
   - Tracking: MLflow help log parameters, metrics and artifacts of each experiment. This ensures that all details are recorded and can be compared later.

2. Model Development:
   - Projects: Standardizes the way to package and share ML code. MLflow Projects can be used to run experiments in a consistent environment.

3. Model Validation
   - Tracking: Continues to log validation metrics and results, making it easier to evaluate model performance.

4. Deployment:
   - Models: Allows you to register, version and deploy models with ease. Models can be served directly via API or integrated into existing systems.

5. Monitoring:
   - Tracking: Helps monitor deployed models by logging predictions and performance metrics, ensuring the model remains effective over time.

6. Life Cycle Management:
   - Registry: Manages the full lifecycle of ML models from development to deployment to retirement.

## * MLFlow Unique Features

1. End-to-End ML Lifecycle Management.
2. Modular Design:
   - Tracking
   - Project
   - Models
   - Registry
3. Flexible Deployment Options.
4. Interoperability
5. Open Source and Extensible
6. Comprehensive model Registry.
7. Built in Experiment Comparison.
8. Seamless Integration.
9. User-friendly interface.
10. Scalability.

* Feature Engineering ⟶ Experiment

1 unique Combination ⟶ Experiment.

If Data changed ⟶ Feature Engineering changes ⟶ model ⟶ Diff Hyper Parameters.
model ⟶ metrics (R2, MSE, MAE, MAPE) ⟶ Diff

* Experiments ⟶ * RUNS
  - Data
  - Model
  - Workflow
  - New exp.

  Hyperparameter Combination.
  - Parent Run ⟶ [HP tuning]
  - Child Run

* A Single experiment Can have multiple runs.

* What is logged?

→

1) Parameters log
2) Metrics log
3) Artifacts log → [Plot, Code, Data]
4) Model log → [After it will get register]

* Key Modules → 1) mlflow
2) mlflow.models → [model agnostic]
3) mlflow.sklearn

* In mlflow, mlflow module is high level api provides a wrapper

mlflow.log_figure() → for plotly, matplotlib graphs
mlflow.log_img() → for storing .png, .jpeg images

* Use docker for packaging instead of mlflow

* preprocessor.get_params()
↓
gives default as well as tuned parameter values of model in dictionary form.
(Get parameter for this estimator.)

* Set_params (**kwargs)
Set the parameters for this estimator.

* How to get params for given estimator?
→ e.g
RandomForestClassifier().get_params()
↳ gives dictionary.

* model_pipe.get_params() (parameters)

* Start MLflow Server in Terminal
↳ mlflow server --port 8000

* When Data, Workflow & model is change new experiment Can be create.

Check if uri is set → mlflow.is_tracking_uri_set()

** Run all the mlflow related tracking code in a single cell **

set uri → mlflow.set_tracking_uri("uri with port")

Set experiment → mlflow.set_experiment("Name")

mlflow logging →
with mlflow.start_run(run_name='first run'):

log parameters →
mlflow.log_params(model_pipe.get_param())

log metrics → mlflow.log_metrics(metrics)

log model → mlflow.sklearn.log_model(sk_model=model_pipe, artifact_path='models')

log Confusion matrix ⟶
mlflow.log_figure(figure = cm_figure, artifact_file = "Confusion matrix.png")

track input Signature ⟶
Signature = mlflow.models.infer_signature(model_input = X_train, model_output = model_pipe.predict(X_train))

~~log data ⟶~~
~~mlflow.log_input(X_train, "train.csv", context = 'training')~~
~~mlflow.log_table() ⟶ Saves in csv file~~

mlflow.log_input(X_train, context = "training")
↳ Saves in form of numpy array

~~mlflow.log_table(data = X_train, artifact_file = 'train.csv')~~

log data ⟶

data = mlflow.data.from_pandas(df = X_train, name = "training")
↳ Create mlflow data

mlflow.log_input(dataset = data, context = 'training')
↳ Log the input.

HP → Hyper Parameters

## * Hyperparameter Tuning & Model Registration

1) GRID Search
2) Random Search
3) Baysean Optimization Search

} HP Tuning Techniques.

Parameters → learned by model through data.

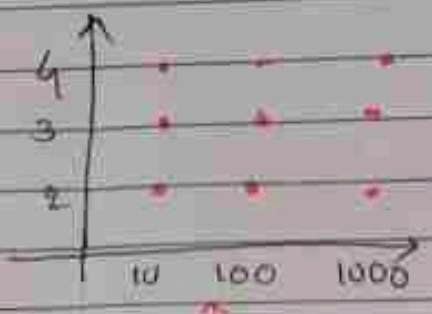
Hyperparameters → does not learn by model.
→ Provided by user
→ Can change the behaviour of model
→ Tuning knobes for model.

Baseline model → Default HP values use in model.

In random forest
Parameters → Rows, Col, Split
Hyper Parameters → n_estimators, max_depth, n_leaves.

### * Grid Search:

↓
Behind the back forms Grid of HP's
→ It is exhaustive Search approach




→ Always return best Combination never skips
→ easy to implement
→ Computationaly expensive

for 3 HP

* Random Search:

↳ Also create a Grid.
→ & Searches for Randomly Combination.

* When you apply Random search CV in diff. 60 random combinations results can be in top 5%

60 random Combination → top 5% result.

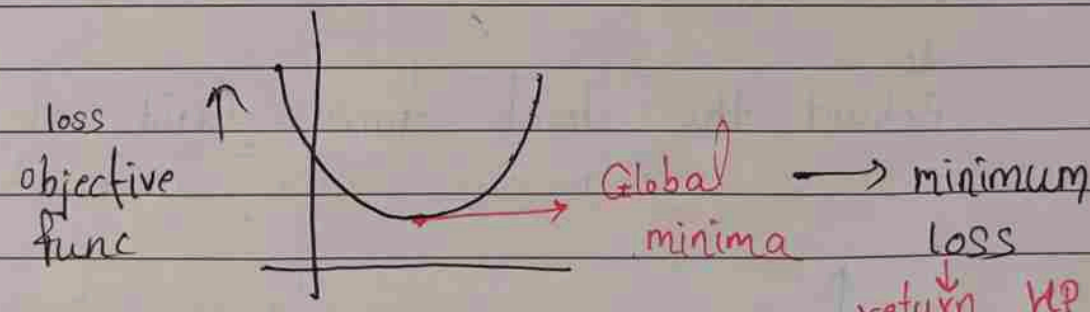
* BaySean Search (HyperOpt) [It is Smart Approach than Random & Grid]

1) Hyperopt (Tree algo. under the hood)
2) Optuna

Requirements:

1) HyperparameterSpace → grid
2) Objective function (Loss function)

3)




* less time Large Combination.

fmin (objective_func, mse, space, algo, 100, Trials).
↳ minimum function
Or epochs / where to store combination

objective (parameter) ⟷ Through MLflow we logged this
*SE

→ ~~finds for obje~~ rejects objective function with higher loss and above combinations

* mlflow.sklearn.autolog() → logs parameters to mlflow.

* log automatic initializes when models fit.

mlflow.log_params(grid_search.best_params_)
↳ logging best parameters.

* We Can Create a nested runs by

* Paraller Co-ordinate Plot → for Hyperparameters and accuracy.