Hyper parameter tunning

- video
- github
- Note Random Search CV -> chose randomly from all the parameter , instent of trying all the possible combination like gride search cv
- value of scoring
  - For classification metrics
    - Accuracy: scoring='accuracy'
    - Precision: scoring='precision'
    - Recall: scoring='recall'
    - F1 Score: scoring='f1'
  - For Regression metrics
    - Mean Absolute Error (MAE): scoring='neg_mean_absolute_error'
    - Mean Squared Error (MSE): scoring='neg_mean_squared_error'
    - R^2 Score: scoring='r2'
    - Root Mean Squared Error (RMSE): Need to create custom scoring function for it
    - Adjusted R^2 Score: Need to create custom scoring function for it
    - code for custom Root Mean Squared Error (RMSE) and Adjusted R^2 Score:

```
from sklearn.metrics import make_scorer, r2_score, mean_squared_error
import numpy as np

# Custom RMSE scorer
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

# Custom Adjusted R^2 scorer
def adjusted_r2(y_true, y_pred, n, p):
    r2 = r2_score(y_true, y_pred)
    adj_r2 = 1 - (1 - r2) * ((n - 1) / (n - p - 1))
    return adj_r2

# Create custom scorers
rmse_scorer = make_scorer(rmse, greater_is_better=Fasle)  # We want to minimize rmse, hence greater_is_better=False
adj_r2_scorer = make_scorer(adjusted_r2, greater_is_better=True)  # We want to maximize adjusted r2, hence greater_is_better=True
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression

# Define model
model = LinearRegression()

# Parameter grid
param_grid = {
    'fit_intercept': [True, False],
    'normalize': [True, False]
}

# GridSearchCV with custom scoring
grid_search_rmse = GridSearchCV(model, param_grid, cv=5, scoring=rmse_scorer)
grid_search_adj_r2 = GridSearchCV(model, param_grid, cv=5, scoring=adj_r2_scorer)

# Fit GridSearchCV
grid_search_rmse.fit(X_train, y_train)
grid_search_adj_r2.fit(X_train, y_train)

# Best RMSE
best_rmse = -grid_search_rmse.best_score_  # negate to get positive value
print(f"Best RMSE: {best_rmse:.4f}")

# Best Adjusted R^2
best_adj_r2 = grid_search_adj_r2.best_score_
print(f"Best Adjusted R^2: {best_adj_r2:.4f}")
```

- code
  - GridSearchCV

```
# Number of trees in random forest
n_estimators = [20,60,100,120]
```

```
# Number of features to consider at every split
max_features = [0.2,0.6,1.0]
```

```
# Maximum number of levels in tree
max_depth = [2,8,None]
```

```
# Number of samples
max_samples = [0.5,0.75,1.0]
```

```
param_grid = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
             'max_samples':max_samples
             }
```

```
rf = RandomForestClassifier()
```

```
from sklearn.model_selection import GridSearchCV

rf_grid = GridSearchCV(estimator = rf,
                       param_grid = param_grid,
                       cv = 5,
                       scoring='accuracy'
                       verbose=2,
                       n_jobs = -1)
```

```
rf_grid.fit(X_train,y_train)
```

```
rf_grid.best_params_
```

```
rf_grid.best_score_
```

  - RandomSearchCV

```
# Number of trees in random forest
n_estimators = [20,60,100,120]
```

```
# Number of features to consider at every split
max_features = [0.2,0.6,1.0]
```

```
# Maximum number of levels in tree
max_depth = [2,8,None]
```

```
# Number of samples
max_samples = [0.5,0.75,1.0]
```

```
# Bootstrap samples
bootstrap = [True,False]
```

```
# Minimum number of samples required to split a node
min_samples_split = [2, 5]
```

```
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2]
```

```
param_grid = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
             'max_samples':max_samples,
             'bootstrap':bootstrap,
             'min_samples_split':min_samples_split,
             'min_samples_leaf':min_samples_leaf
             }
```

```
from sklearn.model_selection import RandomizedSearchCV

rf_grid = RandomizedSearchCV(estimator = rf,
                       param_distributions = param_grid,
                       cv = 5,
                       scoring='accuracy'
                       verbose=2,
                       n_jobs = -1)
```

```
rf_grid.fit(X_train,y_train)
```

```
rf_grid.best_params_
```

```
rf_grid.best_score_
```