# 04-register-model

June 10, 2024

[ ]:
```
import os
import pickle
import logging
import mlflow
import click
from mlflow.entities import ViewType
from mlflow.tracking import MlflowClient
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

Configure logging

[ ]:
```
logging.basicConfig(filename='logs/model_register.log', level=logging.INFO)
```

[ ]:
```
HPO_EXPERIMENT_NAME = "random-forest-hyperopt"
EXPERIMENT_NAME = "random-forest-best-models"
RF_PARAMS = ['max_depth', 'n_estimators', 'min_samples_split',␣
 ↪'min_samples_leaf', 'random_state']
```

Set MLflow tracking URI and experiment name

[ ]:
```
mlflow.set_tracking_uri("http://127.0.0.1:5000")
mlflow.set_experiment(EXPERIMENT_NAME)
mlflow.sklearn.autolog()
```

[ ]:
```
def load_pickle(fileName: str):
    """
    Load data from a pickle file.
    Args:
    fileName (str): Path to the pickle file.
    Returns:
    object: Data loaded from the pickle file.
    """
    try:
        with open(fileName, 'rb') as f:
            return pickle.load(f)
    except FileNotFoundError:
        logging.error(f"Error: File '{fileName}' not found.")
```

```
        return None
```

[ ]:
```
def train_and_log_model(data_path, params):
    # Load data
    X_train, y_train = load_pickle(os.path.join(data_path, "train.pkl"))
    X_val, y_val = load_pickle(os.path.join(data_path, "val.pkl"))
    X_test, y_test = load_pickle(os.path.join(data_path, "test.pkl"))
    with mlflow.start_run():
        # Convert relevant parameters to int
        for param in RF_PARAMS:
            if param in params:
                params[param] = int(params[param])

        # Initialize and train RandomForestRegressor
        rf = RandomForestRegressor(**params)
        rf.fit(X_train, y_train)

        # Evaluate model on validation and test sets
        val_rmse = mean_squared_error(y_val, rf.predict(X_val), squared=False)
        mlflow.log_metric("val_rmse", val_rmse)
        test_rmse = mean_squared_error(y_test, rf.predict(X_test),␣
 ↪squared=False)
        mlflow.log_metric("test_rmse", test_rmse)
```

[1]:
```
def run_register_model(data_path: str, top_n: int):
    client = MlflowClient()

    # Retrieve the top_n model runs and log the models
    experiment = client.get_experiment_by_name(HPO_EXPERIMENT_NAME)
    if experiment is None:
        logging.error(f"Error: Experiment '{HPO_EXPERIMENT_NAME}' not found.")
        return None
    runs = client.search_runs(
        experiment_ids=experiment.experiment_id,
        run_view_type=ViewType.ACTIVE_ONLY,
        max_results=top_n,
        order_by=["metrics.rmse ASC"]
    )
    for run in runs:
        train_and_log_model(data_path=data_path, params=run.data.params)

    # Select the model with the lowest test RMSE
    experiment = client.get_experiment_by_name(EXPERIMENT_NAME)
    best_run = client.search_runs(
        experiment_ids=experiment.experiment_id,
        run_view_type=ViewType.ACTIVE_ONLY,
        max_results=top_n,
```

```
        order_by=["metrics.test_rmse ASC"]
    )[0]

    # Register the best model
    run_id = best_run.info.run_id
    model_uri = f"runs:/{run_id}/model"
    mlflow.register_model(model_uri, name="rf-best-model")
```

[ ]:
```
if __name__ == '__main__':
    # Set the path to the data directory
    CURRENT_DIRECTORY = os.getcwd()
    DEST_PATH = os.path.join(CURRENT_DIRECTORY, 'DEST_PATH')

    run_register_model(DEST_PATH, top_n=5)
```