```
'''
Purpose of the model building phase:
- Implementing the design decisions from the model planning phase
- Training the models across the defined hyperparameter configurations
- Evaluating each model using the chosen metrics (RMSE, MAE, MSE)
- Logging parameters, metrics, and model artifacts via MLflow for comparison and reuse
'''
```

```
'\nPurpose of the model building phase:\n- Implementing the design decisions from the model planning phase\n- Training the models across the defined hyperparameter configurations\n- Evaluating each model using the chosen metrics (RMSE, MAE, MSE)\n- Logging parameters, metrics, and model artifacts via MLflow for comparison and reuse\n'
```

```
'''
Model Building Summary/ Results of model building:
In the model building phase, the preprocessing and prediction steps were combined into a single,
reproducible scikit-learn Pipeline consisting of a StandardScaler and a RandomForestRegressor.
The data was split chronologically into train, validation and test sets (80% / 10% / 10%),
as shuffling is not appropriate for time-dependent data.A baseline Random Forest model was trained and evaluated,
followed by a hyperparameter tuning step using RandomizedSearchCV. The search was executed inside an MLflow-tracked training
allowing parameters, metrics and artifacts to be logged and compared across experiments. The best configuration found within
defined parameter ranges was: n_estimators = 200, min_samples_leaf = 3, max_features = "sqrt", max_depth = 10.

The tuned model showed only a small improvement over the baseline, with validation metrics RMSE = 6.20, MAE = 4.49 and MSE =
These values are the best combination sampled by randomized search but do not represent a guaranteed global optimum.
The final best_estimator_ was logged to MLflow and will be used as the model selected for deployment.

'''
```

```
'\nModel Building Summary/ Results of model building:\nIn the model building phase, the preprocessing and prediction steps were combined into a single,\nreproducible scikit-learn Pipeline consisting of a StandardScaler and a RandomForestRegressor. \nThe data was split chronologically into train, validation and test sets (80% / 10% / 10%),\nas shuffling is not appropriate for time-dependent data.A baseline Random Forest model was trained and evaluated,\nfollowed by a hyperparameter tuning step using RandomizedSearchCV. The search was executed inside an MLflow-tracked training run,\nallowing parameters, metrics and artifacts to be logged and compared across experiments. The best configuration found within the \ndefined parameter ranges was: n_estimators = 200, min_samples_leaf = 3, max_features = "sqrt", max_depth = 10.\n\nThe tuned model showed only a small improvement over the baseline, with validation metrics RMSE = 6.20, MAE = 4.49 and MSE = 38.49.\nThese values are the best combination sampled by randomized search but do not represent a guaranteed global optimum.\nThe final best_estimator_ was logged to MLflow and will be used as the model selected for deployment.\n\n'
```

```
import pandas as pd

!pip install scikit-learn
!pip install mlflow
!pip install cloudpickle
import mlflow
import numpy as np

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
Collecting scikit-learn
  Downloading scikit_learn-1.7.2-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (11 kB)
Requirement already satisfied: numpy>=1.22.0 in /opt/conda/lib/python3.12/site-packages (from scikit-learn) (2.3.5)
Collecting scipy>=1.8.0 (from scikit-learn)
  Downloading scipy-1.16.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (62 kB)
Collecting joblib>=1.2.0 (from scikit-learn)
  Downloading joblib-1.5.2-py3-none-any.whl.metadata (5.6 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Downloading scikit_learn-1.7.2-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (9.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.5/9.5 MB 18.5 MB/s  0:00:00m0:00:0100:01
Downloading joblib-1.5.2-py3-none-any.whl (308 kB)
Downloading scipy-1.16.3-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (35.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 35.7/35.7 MB 19.7 MB/s  0:00:01m0:00:0100:02
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4/4 [scikit-learn][0m [scikit-learn]
Successfully installed joblib-1.5.2 scikit-learn-1.7.2 scipy-1.16.3 threadpoolctl-3.6.0
Collecting mlflow
  Downloading mlflow-3.6.0-py3-none-any.whl.metadata (31 kB)
Collecting mlflow-skinny==3.6.0 (from mlflow)
  Downloading mlflow_skinny-3.6.0-py3-none-any.whl.metadata (31 kB)
Collecting mlflow-tracing==3.6.0 (from mlflow)
  Downloading mlflow_tracing-3.6.0-py3-none-any.whl.metadata (19 kB)
Collecting Flask-CORS<7 (from mlflow)
  Downloading flask_cors-6.0.1-py3-none-any.whl.metadata (5.3 kB)
Collecting Flask<4 (from mlflow)
```

```
  Downloading flask-3.1.2-py3-none-any.whl.metadata (3.2 kB)
Requirement already satisfied: alembic!=1.10.0,<2 in /opt/conda/lib/python3.12/site-packages (from mlflow) (1.17.2)
Requirement already satisfied: cryptography<47,>=43.0.0 in /opt/conda/lib/python3.12/site-packages (from mlflow) (46.0.3)
Collecting docker<8,>=4.0.0 (from mlflow)
  Downloading docker-7.1.0-py3-none-any.whl.metadata (3.8 kB)
Collecting graphene<4 (from mlflow)
  Downloading graphene-3.4.3-py2.py3-none-any.whl.metadata (6.9 kB)
Collecting gunicorn<24 (from mlflow)
  Downloading gunicorn-23.0.0-py3-none-any.whl.metadata (4.4 kB)
Collecting huey<3,>=2.5.0 (from mlflow)
  Downloading huey-2.5.4-py3-none-any.whl.metadata (4.6 kB)
Requirement already satisfied: matplotlib<4 in /opt/conda/lib/python3.12/site-packages (from mlflow) (3.10.8)
Requirement already satisfied: numpy<3 in /opt/conda/lib/python3.12/site-packages (from mlflow) (2.3.5)
Requirement already satisfied: pandas<3 in /opt/conda/lib/python3.12/site-packages (from mlflow) (2.3.3)
Requirement already satisfied: pyarrow<23,>=4.0.0 in /opt/conda/lib/python3.12/site-packages (from mlflow) (21.0.0)
Requirement already satisfied: scikit-learn<2 in /opt/conda/lib/python3.12/site-packages (from mlflow) (1.7.2)
Requirement already satisfied: scipy<2 in /opt/conda/lib/python3.12/site-packages (from mlflow) (1.16.3)
Requirement already satisfied: sqlalchemy<3,>=1.4.0 in /opt/conda/lib/python3.12/site-packages (from mlflow) (2.0.44)
Requirement already satisfied: cachetools<7,>=5.0.0 in /opt/conda/lib/python3.12/site-packages (from mlflow-skinny==3.6.0-
Requirement already satisfied: click<9,>=7.0 in /opt/conda/lib/python3.12/site-packages (from mlflow-skinny==3.6.0->mlflow
Requirement already satisfied: cloudpickle<4 in /opt/conda/lib/python3.12/site-packages (from mlflow-skinny==3.6.0->mlflow
Collecting databricks-sdk<1,>=0.20.0 (from mlflow-skinny==3.6.0->mlflow)
  Downloading databricks_sdk-0.73.0-py3-none-any.whl.metadata (40 kB)
Collecting fastapi<1 (from mlflow-skinny==3.6.0->mlflow)
  Downloading fastapi-0.123.8-py3-none-any.whl.metadata (30 kB)
Requirement already satisfied: gitpython<4,>=3.1.9 in /opt/conda/lib/python3.12/site-packages (from mlflow-skinny==3.6.0->
Requirement already satisfied: importlib_metadata!=4.7.0,<9,>=3.7.0 in /opt/conda/lib/python3.12/site-packages (from mlflo
Collecting opentelemetry-api<3,>=1.9.0 (from mlflow-skinny==3.6.0->mlflow)
  Downloading opentelemetry_api-1.39.0-py3-none-any.whl.metadata (1.5 kB)
Collecting opentelemetry-proto<3,>=1.9.0 (from mlflow-skinny==3.6.0->mlflow)
```

```
import os
os.listdir()
```

```
['mlruns',
 '.ipynb_checkpoints',
 'file_modelbuilding.py',
 'file_dis&dataprep.py',
 'Model Building.ipynb',
 'file_modelbuilding_uv.py',
 'Discovery&DataPrep.ipynb',
 'Model_planning.ipynb']
```

```
#loading df_merged and splitting data
df_merged = pd.read_csv("/work/bda2/datasets/df_merged.csv")
#1:1 the same function as in discovery&dataprep notebook, but I have to define it for each notebook
def create_splits(df_merged, seed=42):


    df_merged = df_merged.sort_values(by="target_time").reset_index(drop=True)

    n = len(df_merged)


    split_1 = int(n * 0.80)
    temp_train = df_merged.iloc[:split_1]
    test_df = df_merged.iloc[split_1:]

    # 2️⃣ Train/Validation split (80/20 of temp_train)
    split_2 = int(len(temp_train) * 0.80)
    train_df = temp_train.iloc[:split_2]
    val_df = temp_train.iloc[split_2:]

    return train_df, val_df, test_df

# create the splits
train_df, val_df, test_df = create_splits(df_merged, seed=42)

print(len(train_df), len(val_df), len(test_df))

#appying general function def create_splits to df_merged
train_df, val_df, test_df = create_splits(df_merged)
```

```
169 43 53
```

```
df_merged.head()
```

	target_time	Direction	Lead_hours	Source_time	Speed	Total	dir_deg	dir_rad	u	v
0	2021-12-11 15:00:00+00:00	SSE	1	1639227600	11.17600	27.621048	157.5	2.748894	-10.325278	4.276870e+00
1	2021-12-11 18:00:00+00:00	SSW	1	1639238400	8.04672	21.135542	202.5	3.534292	-7.434200	-3.079346e+00
2	2021-12-11 21:00:00+00:00	WSW	1	1639249200	11.17600	20.616209	247.5	4.319690	-4.276870	-1.032528e+01
	2021-12-12									

```
#defining features and label (supervised learning)
numeric_features = ["Speed", "u", "v"] #nneded for preprocessing, i.e., the preprocessor knows which features to scale
feature_cols = ["Speed", "u", "v"]
keep_cols = ["Speed", "u", "v", "Total"]
#only keeping the nessecar columns. I also decided to not keep target_time, since not time series analysis
train_df = train_df[keep_cols]
val_df   = val_df[keep_cols]
test_df  = test_df[keep_cols]
X_train = train_df[feature_cols]
y_train = train_df["Total"]

X_val = val_df[feature_cols]
y_val = val_df["Total"]
```

```
#defining evaluation function
def evaluate_model(model, X_val, y_val, name="model"):
    preds = model.predict(X_val)

    rmse = np.sqrt(mean_squared_error(y_val, preds))
    mae  = mean_absolute_error(y_val, preds)
    mse  = mean_squared_error(y_val, preds)

    print(f"\nResults for {name}:")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAE : {mae:.4f}")
    print(f"MSE : {mse:.4f}")

    return rmse, mae, mse
```

```
# preprocessing and pipeline generic / modular approach
# preprocessing like in discovery & data prep
preprocessor = ColumnTransformer(
    [("num", StandardScaler(), numeric_features)],
    remainder="passthrough"
)

from sklearn.pipeline import Pipeline

# generic pipeline builder
def make_pipeline(model):
    return Pipeline([
        ("preprocess", preprocessor),
        ("model", model)
    ])

'''
Note:  Although splitting and preprocessing were already tested in the Data Preparation notebook,# they must be implementec
here in the Model Building phase. The previous steps were exploratory, this notebook performs the final, reproducibletrain,
test splits and constructs the full preprocessing+model pipeline that will be  trained, evaluated, and logged with MLflow.
'''
```

```
'\nNote:  Although splitting and preprocessing were already tested in the Data Preparation notebook,# they must be
implemented again\nhere in the Model Building phase. The previous steps were exploratory, this notebook performs the final,
reproducibletrain/validation/\ntest splits and constructs the full preprocessing+model pipeline that will be  trained,
evaluated, and logged with MLflow.\n'
```

```
#pipeline specif part for random forest, will be used for tain/val/ test seperately to prevent data leakage
rfr_baseline = RandomForestRegressor(
    n_estimators=300,
    max_depth=10,
    min_samples_leaf=3,
    max_features="sqrt",
    random_state=42
)
```

```
pipeline_rfr_baseline = make_pipeline(rfr_baseline)
```

```
#training the model
pipeline_rfr_baseline.fit(X_train, y_train)
```

```
Pipeline
  preprocess: ColumnTransformer
    num                 remainder
    StandardScaler      passthrough
  RandomForestRegressor
```

```
rmse_rfr, mae_rfr, mse_rfr = evaluate_model(
    pipeline_rfr_baseline, X_val, y_val, "RandomForest"
)
```

```
Results for RandomForest:
RMSE: 6.2040
MAE : 4.4881
MSE : 38.4901
```

```
# Collect results for a clean model comparison
results = {}

results["RandomForest"] = (rmse_rfr, mae_rfr, mse_rfr)
```

```
#for having context to evaluate results
df_merged["Total"].describe()
```

```
count    265.000000
mean      21.449641
std       10.795674
min        1.096348
25%       11.408295
50%       26.096300
75%       30.348545
max       35.492622
Name: Total, dtype: float64
```

```
#The baseline Random Forest achieves an MAE of 4.49 MW (≈21% of mean production) and an RMSE of 6.20 MW,
#indicating that the model captures the general wind-power relationship but still shows occasional larger deviations due to

#need to optimize the model in another run. I use randomized search over grid search, since this hyperparameter tuning appr
#is supposed to be better for small datasets.
```

```
import mlflow
import mlflow.sklearn
from sklearn.model_selection import RandomizedSearchCV

# Hyperparameter search space
param_grid = {
    "model__n_estimators": [200, 300, 500],
    "model__max_depth": [8, 10, 12, None],
    "model__min_samples_leaf": [1, 2, 3, 5],
    "model__max_features": ["sqrt", "auto"],
}

# RandomizedSearchCV setup
search = RandomizedSearchCV(
    estimator=pipeline_rfr_baseline,
    param_distributions=param_grid,
    n_iter=12,
    scoring="neg_mean_absolute_error",
    cv=3,
    n_jobs=-1,
    random_state=42
)
```

```
# --- MLflow Run ---
with mlflow.start_run(run_name="RandomForest_Finetuning"):

    search.fit(X_train, y_train)
    best_model = search.best_estimator_

    val_preds = best_model.predict(X_val)

    rmse_val = np.sqrt(mean_squared_error(y_val, val_preds))
    mae_val  = mean_absolute_error(y_val, val_preds)
    mse_val  = mean_squared_error(y_val, val_preds)

    # Log hyperparameters
    for param_name, param_value in search.best_params_.items():
        mlflow.log_param(param_name, param_value)

    # Log metrics
    mlflow.log_metric("rmse_val", rmse_val)
    mlflow.log_metric("mae_val", mae_val)
    mlflow.log_metric("mse_val", mse_val)

    # Log model
    mlflow.sklearn.log_model(best_model, "model")

    print("Best hyperparameters:", search.best_params_)
    print("MAE:", mae_val)
    print("RMSE:", rmse_val)
    print("MSE:", mse_val)
```

```
/opt/conda/lib/python3.12/site-packages/mlflow/tracking/_tracking_service/utils.py:140: FutureWarning: Filesystem tracking b
  return FileStore(store_uri, store_uri)
/opt/conda/lib/python3.12/site-packages/sklearn/model_selection/_validation.py:516: FitFailedWarning:
12 fits failed out of a total of 36.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
7 fits failed with the following error:
Traceback (most recent call last):
  File "/opt/conda/lib/python3.12/site-packages/sklearn/model_selection/_validation.py", line 859, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/conda/lib/python3.12/site-packages/sklearn/base.py", line 1365, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/opt/conda/lib/python3.12/site-packages/sklearn/pipeline.py", line 663, in fit
    self._final_estimator.fit(Xt, y, **last_step_params["fit"])
  File "/opt/conda/lib/python3.12/site-packages/sklearn/base.py", line 1358, in wrapper
    estimator._validate_params()
  File "/opt/conda/lib/python3.12/site-packages/sklearn/base.py", line 471, in _validate_params
    validate_parameter_constraints(
  File "/opt/conda/lib/python3.12/site-packages/sklearn/utils/_param_validation.py", line 98, in validate_parameter_constrai
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of RandomForestRegressor must be an int

--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/opt/conda/lib/python3.12/site-packages/sklearn/model_selection/_validation.py", line 859, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/opt/conda/lib/python3.12/site-packages/sklearn/base.py", line 1365, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/opt/conda/lib/python3.12/site-packages/sklearn/pipeline.py", line 663, in fit
    self._final_estimator.fit(Xt, y, **last_step_params["fit"])
  File "/opt/conda/lib/python3.12/site-packages/sklearn/base.py", line 1358, in wrapper
    estimator._validate_params()
  File "/opt/conda/lib/python3.12/site-packages/sklearn/base.py", line 471, in _validate_params
    validate_parameter_constraints(
  File "/opt/conda/lib/python3.12/site-packages/sklearn/utils/_param_validation.py", line 98, in validate_parameter_constrai
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of RandomForestRegressor must be an int

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/opt/conda/lib/python3.12/site-packages/sklearn/model_selection/_search.py:1135: UserWarning: One or more of the test scores
         nan         nan -4.87945125 -4.82538025         nan -4.65675384]
  warnings.warn(
2025/12/04 18:58:21 WARNING mlflow.models.model: `artifact_path` is deprecated. Please use `name` instead.
2025/12/04 18:58:30 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_examp
Best hyperparameters: {'model__n_estimators': 200, 'model__min_samples_leaf': 3, 'model__max_features': 'sqrt', 'model__max_
MAE: 4.456784194494066
RMSE: 6.169130309172878
MSE: 38.05816877155546
```

```
 # Save tuned Random Forest results in dictionary
results["RandomForest_Tuned"] = (rmse_val, mae_val, mse_val)
```

```
'''
evaluation metrics of run1 for comparison: RMSE (Root Mean Squared Error): 6.2040,MAE  (Mean Absolute Error):4.4881,
MSE  (Mean Squared Error): 38.4901

--> "only" a very small improvement
interpretation of best hyperparameters according to randomized search.for the given range defined in the code in the
function parameter_grid, this values for the hyperparameters are the best combination:
'model__n_estimators': 200, 'model__min_samples_leaf': 3, 'model__max_features': 'sqrt', 'model__max_depth': 10

It is worth mentioning, that it is not by default the best combination possible. It is the best combination this hyperparar

'''
```

```
'\nevaluation metrics of run1 for comparison: RMSE (Root Mean Squared Error): 6.2040,MAE  (Mean Absolute
Error):4.4881,\nMSE  (Mean Squared Error): 38.4901\n\n--> "only" a very small improvement\ninterpretation of best
hyperparameters according to randomized search.for the given range defined in the code in the \nfunction parameter_grid,
this values for the hyperparameters are the best combination: \n\'model__n_estimators\': 200, \'model__min_samples_leaf\':
3, \'model__max_features\': \'sqrt\', \'model__max_depth\': 10\n\nIt is worth mentioning, that it is not by default the
best combination possible. It is the best combination this hyperparameter optimization algorithm has found. Other
hyperparameters might have come to other, "better" performing hyperparameter combinations.\n\n'
```

```
#using now gradient descent regressor to be able to compare the model performance
```

```
#using same generic data pipeline and preprocessing
#defing specific pipeline part for gradient descent regressor
from sklearn.ensemble import GradientBoostingRegressor

gbr_model = GradientBoostingRegressor(
    n_estimators=300,
    learning_rate=0.1,
    max_depth=3,
    min_samples_split=2,
    loss="squared_error",
    random_state=42
)
```

```
#applying pipeline
pipeline_gbr = make_pipeline(gbr_model)
#training model
pipeline_gbr.fit(X_train, y_train)
```

Pipeline

preprocess: ColumnTransformer

num | **remainder**

StandardScaler | passthrough

GradientBoostingRegressor

```
#evaluating the model with generic function defined above
rmse_gbr, mae_gbr, mse_gbr = evaluate_model(
    pipeline_gbr, X_val, y_val, "GradientBoosting"
)
```

```
Results for GradientBoosting:
RMSE: 7.7969
MAE : 5.1331
MSE : 60.7921
```

```
results["GradientBoosting"] = (rmse_gbr, mae_gbr, mse_gbr)
print(results)
```

```
{'RandomForest': (np.float64(6.20403528759331), 4.488051380047191, 38.49005384970301), 'RandomForest_Tuned': (np.float64(6.1
```

```
#in baseline comparison random forest clearly better
```

```
#optimizing gradientboosting
#same method as in random forest,i.e., randomized search space
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import GradientBoostingRegressor
import mlflow
import mlflow.sklearn

# Gradient Boosting Hyperparameter Grid
param_grid_gbr = {
    "model__n_estimators": [100, 300, 500],
    "model__learning_rate": [0.1, 0.05, 0.01],
    "model__max_depth": [2, 3, 4],
    "model__min_samples_split": [2, 5, 10],
    "model__loss": ["squared_error"]
}
```

```
search_gbr = RandomizedSearchCV(
    estimator=pipeline_gbr,
    param_distributions=param_grid_gbr,
    n_iter=12,
    scoring="neg_mean_absolute_error",
    cv=3,
    n_jobs=-1,
    random_state=42
)
```

```
with mlflow.start_run(run_name="GradientBoosting_Finetuning"):

    search_gbr.fit(X_train, y_train)
    best_gbr = search_gbr.best_estimator_

    rmse_val, mae_val, mse_val = evaluate_model(
        best_gbr, X_val, y_val, name="GradientBoosting_Tuned"
    )

    # Log hyperparameters
    for param, value in search_gbr.best_params_.items():
        mlflow.log_param(param, value)

    # Log metrics
    mlflow.log_metric("rmse_val", rmse_val)
    mlflow.log_metric("mae_val", mae_val)
    mlflow.log_metric("mse_val", mse_val)

    # Log fitted model as MLflow artifact
    mlflow.sklearn.log_model(best_gbr, "model")

    print("Best hyperparameters:", search_gbr.best_params_)
    print("MAE:", mae_val)
    print("RMSE:", rmse_val)
    print("MSE:", mse_val)
```

```
2025/12/04 18:58:31 WARNING mlflow.models.model: `artifact_path` is deprecated. Please use `name` instead.

Results for GradientBoosting_Tuned:
RMSE: 6.8582
MAE : 4.9408
MSE : 47.0356
2025/12/04 18:58:37 WARNING mlflow.models.model: Model logged without a signature and input example. Please set `input_examp
Best hyperparameters: {'model__n_estimators': 100, 'model__min_samples_split': 5, 'model__max_depth': 2, 'model__loss': 'squ
MAE: 4.940792109355536
RMSE: 6.858247887874707
MSE: 47.03556409153787
```

```
results["GradientBoosting_Tuned"] = (rmse_val, mae_val, mse_val)
results
```

```
{'RandomForest': (np.float64(6.20403528759331),
  4.488051380047191,
  38.49005384970301),
 'RandomForest_Tuned': (np.float64(6.169130309172878),
  4.456784194494066,
  38.05816877155546),
 'GradientBoosting': (np.float64(7.7969309773830915),
  5.133091161779405,
  60.79213266607606),
```

```
 'GradientBoosting_Tuned': (np.float64(6.858247887874707),
  4.940792109355536,
  47.03556409153787)}
```

```
#randomforest_tuned is the best model
```

```
#showing me the run id for the best model, so I can deploy it with mlflow
with mlflow.start_run(run_name="RandomForest_Tuned") as run:
    run_id = run.info.run_id
    print("Run ID:", run_id)
```

```
Run ID: ddb8e93ba4924decbcea9d58108a5241
```

```
#now applying the best model to future_df
# --- Prepare df_future the same way as df_merged ---

# Convert direction to degrees
df_future  = pd.read_csv("../datasets/future.csv")
dir_map = {
    "N": 0, "NNE": 22.5, "NE": 45, "ENE": 67.5,
    "E": 90, "ESE": 112.5, "SE": 135, "SSE": 157.5,
    "S": 180, "SSW": 202.5, "SW": 225, "WSW": 247.5,
    "W": 270, "WNW": 292.5, "NW": 315, "NNW": 337.5
}

df_future["dir_deg"] = df_future["Direction"].map(dir_map)
df_future["dir_rad"] = np.deg2rad(df_future["dir_deg"])

# calculate u and v
df_future["u"] = df_future["Speed"] * np.cos(df_future["dir_rad"])
df_future["v"] = df_future["Speed"] * np.sin(df_future["dir_rad"])
```

```
#select features
X_future = df_future[["Speed", "u", "v"]]
```

```
future_predictions = best_model.predict(X_future)

df_future["Predicted_Power"] = future_predictions

df_future[["time", "Predicted_Power"]].head()
```

	time	Predicted_Power
0	2022-03-11 15:00:00+00:00	30.620526
1	2022-03-11 18:00:00+00:00	30.794534
2	2022-03-11 21:00:00+00:00	30.725303
3	2022-03-12 00:00:00+00:00	31.407954
4	2022-03-12 03:00:00+00:00	30.637139

```
#saving it
'''
df_future.to_csv("../datasets/future_predictions.csv", index=False)
'''
```

```
'\ndf_future.to_csv("../datasets/future_predictions.csv", index=False)\n'
```

```
#visualising the results

# Load predictions
df_future = pd.read_csv("../datasets/future_predictions.csv")

# Ensure time column is datetime
df_future["time"] = pd.to_datetime(df_future["time"])

# Sort by time (just to be safe)
df_future = df_future.sort_values("time")

# Plot
plt.figure(figsize=(12,6))
```

```
plt.plot(df_future["time"], df_future["Predicted_Power"], marker="o", linestyle="-")

plt.title("Predicted Future Power Output (3-hour intervals)")
plt.xlabel("Time")
plt.ylabel("Predicted Power (MW)")

# Format date labels
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d\n%H:%M'))
plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator())

plt.grid(True)
plt.tight_layout()
plt.show()
```