Stock Price Prediction (NIFTY 50)

This project focuses on predicting **NIFTY 50 stock prices** using both **Machine Learning (ML)** and **Deep Learning (DL)** approaches.
The workflow involves preparing time-series data, training multiple models, and comparing their performance.

Pipeline Overview

1. **Data Loading**
   - Load stock price data (`data.csv`).
   - Features: `Open`, `Close`, `High`, `Low`.
2. **Data Preparation**
   - Create supervised learning datasets using sliding windows (30–250 days).
   - Generate `(X, y)` pairs for each feature.
3. **Modeling**
   - **Machine Learning Models**
     - Linear: `LinearRegression`, `Ridge`, `Lasso`
     - Tree-based: `RandomForest`, `GradientBoosting`, `XGBoost`, `LightGBM`
     - Others: `SVR`, `KNN`
   - **Deep Learning Models**
     - RNN, LSTM, GRU, Bidirectional LSTM (Keras Sequential API)
4. **Training**
   - Train models on rolling window datasets.
   - Evaluate using **MAE** and **RMSE**.
5. **Evaluation & Comparison**
   - Store results for all models.
   - Compare ML vs DL models for different input window sizes.

Key Highlights

- Hybrid pipeline combining **classical ML** and **neural networks**.
- Uses **multiple time horizons (30–250 days)** for robust prediction.
- Tracks **training and testing errors** to evaluate generalization.

Assignment Has been Submitted at the end of the Notebook

Assignment:

#Based on the analysis performed in this notebook, the assignment is to focus on building and evaluating models for predicting the **High** price of the NIFTY 50 index.

1. **Import Libraries**
   - `numpy`, `pandas`: data handling
   - `tqdm`: progress bars
   - `sklearn`: machine learning models & metrics
   - `xgboost`, `lightgbm`: gradient boosting models
   - `warnings`: ignore warnings
2. **Load Dataset**
   - `df = pd.read_csv('data.csv')` → load data from CSV
   - `df.head()` → preview first 5 rows
3. **Models Imported**
   - Linear: `LinearRegression`, `Ridge`, `Lasso`
   - Tree-based: `RandomForestRegressor`, `GradientBoostingRegressor`
   - Others: `SVR`, `KNeighborsRegressor`, `XGBRegressor`, `LGBMRegressor`
4. **Metrics Imported**
   - `mean_absolute_error`, `mean_squared_error` → to evaluate model performance

```
import numpy as np
import pandas as pd
from tqdm.auto import tqdm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from copy import deepcopy
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
import warnings
warnings.filterwarnings("ignore")
df = pd.read_csv('data.csv')
display(df.head())
```


```
{"summary":"{\n  \"name\": \"display(df\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"Date\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n          \"2000-
01-04\",\n          \"2000-01-07\",\n          \"2000-01-05\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Open\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
59.658266401228865,\n        \"min\": 1482.15,\n        \"max\":
1634.55,\n        \"num_unique_values\": 5,\n        \"samples\": [\n
1594.4,\n          1616.6,\n          1634.55\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"High\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 20.017323747194546,\n
\"min\": 1592.9,\n        \"max\": 1641.95,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          1641.95,\
n          1628.25,\n          1635.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Low\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 49.53012467175908,\n
\"min\": 1482.15,\n        \"max\": 1597.2,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          1594.4,\n
1597.2,\n          1555.05\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Close\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 18.703395413667547,\n        \"min\":
1592.2,\n        \"max\": 1638.7,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          1638.7,\n          1613.3,\n
1595.8\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


2. Data Preparation

1. **Train-Test Split**
   - `train_test_split()` → split data into training & testing sets.
2. **Models List**
   - A collection of regressors:
     - Linear: `LinearRegression`, `Ridge`, `Lasso`
     - Tree-based: `RandomForestRegressor`, `GradientBoostingRegressor`
     - Others: `SVR`, `KNeighborsRegressor`, `XGBRegressor`, `LGBMRegressor`

3. **Training Loop**
   - For each model:
     - `fit()` → train on training data
     - `predict()` → generate predictions on test data
4. **Evaluation**
   - Metrics used:
     - `mean_absolute_error`
     - `mean_squared_error`
   - Store results for comparison of all models.

```
def return_pairs(column, days):
    pricess = list(column)
    X = []
    y = []
    for i in range(len(pricess) - days):
        X.append(pricess[i:i+days])
        y.append(pricess[i+days])
    return np.array(X), np.array(y)

target_columns =  ['Open', 'Close', 'High', 'Low']
day_chunks =  [30, 45, 60, 90, 120, 150 ,200, 250]

chunked_data = {}

for col in target_columns:
    for days in day_chunks:
        key_X = f"X_{col}_{days}"
        key_y = f"y_{col}_{days}"
        X, y = return_pairs(df[col], days)
        chunked_data[key_X] = X
        chunked_data[key_y] = y


chunk_pairs = []

for key in chunked_data.keys():
    if key.startswith("X_"):
        y_key = key.replace("X_", "y_")
        if y_key in chunked_data:
            chunk_pairs.append([key, y_key])
```

3. Define Neural Network Models

1. **Imports**
   - `Sequential` → build models layer-by-layer

- Layers: `Dense`, `SimpleRNN`, `LSTM`, `GRU`, `Bidirectional`

2. **Model Builder Functions**
   - `build_rnn(input_shape)`
     - Simple RNN with 50 units → `Dense(1)` output
   - `build_lstm(input_shape)`
     - LSTM with 50 units → `Dense(1)` output
   - `build_gru(input_shape)`
     - GRU with 50 units → `Dense(1)` output
   - `build_bilstm(input_shape)`
     - Bidirectional LSTM with 50 units → `Dense(1)` output
3. **Compilation**
   - Optimizer: `adam`
   - Loss: `mse` (Mean Squared Error)
4. **Purpose**
   - All models → designed for **regression tasks on sequential data**.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, LSTM, GRU,
Bidirectional


def build_rnn(input_shape):
    model = Sequential([
        SimpleRNN(50, activation='tanh', input_shape=input_shape),
        Dense(1)   # regression output
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

def build_lstm(input_shape):
    model = Sequential([
        LSTM(50, activation='tanh', input_shape=input_shape),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

def build_gru(input_shape):
    model = Sequential([
        GRU(50, activation='tanh', input_shape=input_shape),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model
```

```
def build_bilstm(input_shape):
    model = Sequential([
        Bidirectional(LSTM(50, activation='tanh'),
input_shape=input_shape),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model
```

4. Define ML Models

1. **ml_models (Traditional ML)**
   - A list of tuples: (name, model instance)
   - Includes:
     - Linear models → `LinearRegression`, `Ridge`, `Lasso`
     - Tree-based → `RandomForest`, `GradientBoosting`
     - Others → `SVR`, `KNN`, `XGBoost`, `LightGBM`
2. **dl_models (Deep Learning)**
   - A dictionary: {name: builder function}
   - Includes:
     - `"RNN"` → `build_rnn`
     - `"LSTM"` → `build_lstm`
     - `"GRU"` → `build_gru`
     - `"Bidirectional_LSTM"` → `build_bilstm`
3. **Purpose**
   - `ml_models`: ready-to-train classical ML regressors
   - `dl_models`: functions that return compiled neural nets (when given `input_shape`)

```
ml_models = [
    ("LinearRegression", LinearRegression()),
    ("Ridge", Ridge()),
    ("Lasso", Lasso()),
    ("RandomForest", RandomForestRegressor()),
    ("GradientBoosting", GradientBoostingRegressor()),
    ("SVR", SVR()),
    ("KNN", KNeighborsRegressor()),
    ("XGBoost", XGBRegressor(verbosity=0)),
```

```
    ("LightGBM", LGBMRegressor(verbosity=0))
]

dl_models = {
    "RNN": build_rnn,
    "LSTM": build_lstm,
    "GRU": build_gru,
    "Bidirectional_LSTM": build_bilstm
}
```

5. Model Training

1. **Initialize**
   - `trained_models = {}` → store results of all models
2. **Iterate over Data Pairs**
   - For each `(X, y)` in `chunk_pairs`
   - Extract features `X_data` and target `y_data` from `chunked_data`
   - Split → `train_test_split` (90% train, 10% test)
3. **Train ML Models**
   - Loop through `ml_models`
   - Use `deepcopy` to avoid reusing fitted models
   - `fit()` on training data
   - Predict on train & test sets
   - Save model + metrics:
     - `train_mae`, `train_rmse`
     - `test_mae`, `test_rmse`
4. **Prepare Data for DL**
   - Expand dims → shape becomes `(samples, timesteps, features)`
5. **Train DL Models**
   - Loop through `dl_models`
   - Build model with correct input shape
   - Train for 10 epochs, batch size = 8
   - Predict on train & test
   - Save model + metrics (same as ML)
6. **Final Output**

– `trained_models` → dictionary with all trained models & evaluation scores

```
trained_models = {}

for X, y in tqdm(chunk_pairs):
    X_data = chunked_data[X]
    y_data = chunked_data[y]

    X_train, X_test, y_train, y_test = train_test_split(
        X_data, y_data, test_size=0.1, random_state=42
    )

    # ML models
    for model_name, model in tqdm(ml_models):
        key = model_name + '_' + X[2:]
        model_copy = deepcopy(model)
        model_copy.fit(X_train, y_train)

        y_train_pred = model_copy.predict(X_train)
        y_test_pred = model_copy.predict(X_test)

        trained_models[key] = {
            'model': model_copy,
            'train_mae': mean_absolute_error(y_train, y_train_pred),
            'train_rmse': np.sqrt(mean_squared_error(y_train,
y_train_pred)),
            'test_mae': mean_absolute_error(y_test, y_test_pred),
            'test_rmse': np.sqrt(mean_squared_error(y_test,
y_test_pred))
        }

    # DL models
    X_train_rnn = np.expand_dims(X_train, -1)
    X_test_rnn = np.expand_dims(X_test, -1)

    for model_name, builder in tqdm(dl_models.items()):
        key = model_name + '_' + X[2:]
        model_dl = builder((X_train.shape[1], 1))

        model_dl.fit(X_train_rnn, y_train, epochs=10, batch_size=8,
verbose=0)

        y_train_pred = model_dl.predict(X_train_rnn).flatten()
        y_test_pred = model_dl.predict(X_test_rnn).flatten()

        trained_models[key] = {
            'model': model_dl,
            'train_mae': mean_absolute_error(y_train, y_train_pred),
            'train_rmse': np.sqrt(mean_squared_error(y_train,
y_train_pred)),
            'test_mae': mean_absolute_error(y_test, y_test_pred),
```

```
            'test_rmse': np.sqrt(mean_squared_error(y_test,
y_test_pred))
        }
```

{"model_id":"3f92839657664c55b670dad6e4f5f505","version_major":2,"vers
ion_minor":0}

{"model_id":"98f2f7f5660d4cc3975e95d622634d51","version_major":2,"vers
ion_minor":0}

{"model_id":"653bc927c84a4c7f8589dfc87e188b77","version_major":2,"vers
ion_minor":0}

```
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

{"model_id":"5bbd6f7dd2ca404896fddf26de6a04ae","version_major":2,"vers
ion_minor":0}

{"model_id":"5fe833312d8c459481d64ecfda42bff3","version_major":2,"vers
ion_minor":0}

```
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

{"model_id":"f5babc245514484e836401b33284336d","version_major":2,"vers
ion_minor":0}

{"model_id":"bdf9c4aae37b4c238efe45dff2b57c19","version_major":2,"vers
ion_minor":0}

```
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

{"model_id":"1e866c430bc84dd0ab8fb069cb2bcebe","version_major":2,"version_minor":0}

{"model_id":"514e271bfb1c4ed9bd67474163ca2e5a","version_major":2,"version_minor":0}

```
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

{"model_id":"59932b021ab64872be77f23dfeecd355","version_major":2,"version_minor":0}

{"model_id":"6958c0696b634a4e934a714b5e448693","version_major":2,"version_minor":0}

```
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

{"model_id":"5515fd0a38fe4f48a5aaa5b12be994a6","version_major":2,"version_minor":0}

{"model_id":"a65eba0b04e54229bd16c546a6080dd4","version_major":2,"version_minor":0}

```
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

{"model_id":"ce2d14a6c5904329be96ea2a88ce2cc9","version_major":2,"version_minor":0}

{"model_id":"95a8d5228f5c47eb9b7653510eee811b","version_major":2,"version_minor":0}

```
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```


```
{"model_id":"99c9cb3edece44ba89aa289a6c9d7406","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"972c5ca6209f434b87734182023edda5","version_major":2,"vers
ion_minor":0}
```


```
171/171 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```


```
{"model_id":"5e20ac49e68044349c91e28e469175a3","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"2d814e2b063b417ab21fbf2ed1e99c38","version_major":2,"vers
ion_minor":0}
```


```
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```


```
{"model_id":"cc65a44e99a24c2da0adfdd1be45bdb4","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"d1260f3813da40fc8113629056bb0816","version_major":2,"vers
ion_minor":0}
```


```
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

```
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```


```
{"model_id":"5e1fa859dc9b4d66b55a60d850f51bd2","version_major":2,"vers
ion_minor":0}
```



```
{"model_id":"1739090bd85249f69669d9f244c25f2e","version_major":2,"vers
ion_minor":0}
```


```
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```


```
{"model_id":"e1bf0a3d51604443bb3ef8a542c83319","version_major":2,"vers
ion_minor":0}
```



```
{"model_id":"5b1ba10dc56d498fadfcbdefa8fccb13","version_major":2,"vers
ion_minor":0}
```


```
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```


```
{"model_id":"f7943c4bf49d4fc985405d33c03069fb","version_major":2,"vers
ion_minor":0}
```



```
{"model_id":"707bb1d70c014ff79369e4dfc3c0632f","version_major":2,"vers
ion_minor":0}
```


```
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```


```
{"model_id":"dfb432fdfda74cf980f2e7b908256586","version_major":2,"vers
ion_minor":0}
```

{"model_id":"675e683dada64789bcb62b3c9b772437","version_major":2,"version_minor":0}

```
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

{"model_id":"276f940527c84b3585fa5db067cbaeb7","version_major":2,"version_minor":0}

{"model_id":"8abcca30cbc64327aba72c2f8999ebd9","version_major":2,"version_minor":0}

```
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
```

{"model_id":"5bc2911eac044114be0a46c07f10258c","version_major":2,"version_minor":0}

{"model_id":"f847c58e58c54442825eab18f2ca4e0b","version_major":2,"version_minor":0}

```
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```

{"model_id":"a120eec6b8e94ee3a0221f9a72d350a0","version_major":2,"version_minor":0}

{"model_id":"5a8f723d1eb940e1a398964858415bd1","version_major":2,"version_minor":0}

```
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step

{"model_id":"a3ec7ec1236d448ca04f851280cd54aa","version_major":2,"vers
ion_minor":0}

{"model_id":"efaaa3c847e648a79716b7625952e265","version_major":2,"vers
ion_minor":0}

177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

{"model_id":"c5d0ea3dacbe4a1bb58453956c7153e7","version_major":2,"vers
ion_minor":0}

{"model_id":"b7b651d478e04a8c86142d288f0092c0","version_major":2,"vers
ion_minor":0}

176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step

{"model_id":"6716bb7bf7d4442e9c887e0b14c67ba4","version_major":2,"vers
ion_minor":0}

{"model_id":"fcba44619db04b1a8a177209ea73a861","version_major":2,"vers
ion_minor":0}

176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

{"model_id":"d4b730d2805f45118e41c2ae24cad210","version_major":2,"version_minor":0}

{"model_id":"90ca5fa974d54a5ab761c4ba0e4daf2d","version_major":2,"version_minor":0}

```
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

{"model_id":"7a878665afd64874af4bfdc75b340a10","version_major":2,"version_minor":0}

{"model_id":"70052dff8cd746a0aee5d59c9c958831","version_major":2,"version_minor":0}

```
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```

{"model_id":"cf77ba346f3a4ccebabb520ba9c1a20c","version_major":2,"version_minor":0}

{"model_id":"bb42ae60929945018d1569f3957d659e","version_major":2,"version_minor":0}

```
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
```

{"model_id":"8a551231a7e34c8d81b872cdf7eb02eb","version_major":2,"version_minor":0}

{"model_id":"ff12cfb57d894b9e9f2eb9523d2ebd89","version_major":2,"version_minor":0}

```
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
```


```
{"model_id":"07ecd6a2cd9d4bf591d34861fb69ce08","version_major":2,"vers
ion_minor":0}
```



```
{"model_id":"6422575a1f064012b85859315b5c02cc","version_major":2,"vers
ion_minor":0}
```


```
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```


```
{"model_id":"5a0cab4378ac48fa909a4e1cb1b94915","version_major":2,"vers
ion_minor":0}
```



```
{"model_id":"ea25fc3e00464252b71f08fa43206ecb","version_major":2,"vers
ion_minor":0}
```


```
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```


```
{"model_id":"ca5c8e88f0674848952ceaed43e4fbd7","version_major":2,"vers
ion_minor":0}
```



```
{"model_id":"4cbc64cf095e4555ad10395445140b58","version_major":2,"vers
ion_minor":0}
```


```
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

```
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

{"model_id":"704ba02837614c88a8793f886d665151","version_major":2,"vers
ion_minor":0}

{"model_id":"887f926bf70947898f914975fab49075","version_major":2,"vers
ion_minor":0}

176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step

{"model_id":"ba45b1c296114f0b8e3e50c7fbca324c","version_major":2,"vers
ion_minor":0}

{"model_id":"6208438b96344a0ab4e764462fc4f99b","version_major":2,"vers
ion_minor":0}

175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
175/175 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step

{"model_id":"868f4827b64c417db5b4245d262a79cf","version_major":2,"vers
ion_minor":0}

{"model_id":"4c23d53a7e404ea58fe207cd7fecbe7c","version_major":2,"vers
ion_minor":0}

174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
174/174 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step

{"model_id":"89d8ff6511a3441ea63162dfc4c9ff2d","version_major":2,"vers
ion_minor":0}
```

{"model_id":"35e61e3071124234b5978fa4d3fe33f0","version_major":2,"version_minor":0}

```
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
172/172 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```

{"model_id":"887156502e2c4bbd8792e0a73b857c50","version_major":2,"version_minor":0}

{"model_id":"558ae33917484c36ad7270fff5a397ed","version_major":2,"version_minor":0}

```
171/171 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
171/171 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
```

6. Saving Model Statistics

1. **Collect Results**
   - Convert `trained_models` dict → list of dicts
   - Each row = {"Model": model_name, metrics...}
2. **Create DataFrame**
   - `results_df = pd.DataFrame([...])`
   - Columns: `Model`, `train_mae`, `train_rmse`, `test_mae`, `test_rmse`
3. **Sort Results**
   - Sort by `test_mae` (ascending → best first)
4. **Display**
   - Show top 50 models with lowest test MAE

```
results_df = pd.DataFrame([
    {"Model": name, **metrics}
    for name, metrics in trained_models.items()])

results_df.sort_values(by = 'test_mae', ascending = True).head(50)
```

```
{"summary":"{\n  \"name\": \"results_df\",\n  \"rows\": 50,\n
\"fields\": [\n    {\n      \"column\": \"Model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 50,\n        \"samples\": [\n
\"LinearRegression_High_120\",\n          \"Ridge_Low_30\",\n
\"RandomForest_High_45\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"model\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 50,\n        \"samples\":
[\n          \"LinearRegression()\",\n          \"Ridge()\",\n
\"RandomForestRegressor()\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"train_mae\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 11.491333575950371,\n
\"min\": 20.61307497757841,\n        \"max\": 61.06409332036474,\n
\"num_unique_values\": 50,\n        \"samples\": [\n
51.78261611784351,\n          59.031255574363186,\n
20.671060694666014\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"train_rmse\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 18.830094704096688,\n        \"min\":
33.32159731930624,\n        \"max\": 101.18701848252172,\n
\"num_unique_values\": 50,\n        \"samples\": [\n
83.4835122199342,\n          101.18701848252171,\n
34.35734835773132\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"test_mae\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3.3189325819970805,\n        \"min\":
46.97174365244455,\n        \"max\": 58.057636422865386,\n
\"num_unique_values\": 50,\n        \"samples\": [\n
51.553715691288666,\n          57.19382346505478,\n
56.04569617224889\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"test_rmse\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 7.143919924931163,\n        \"min\":
73.10791845748425,\n        \"max\": 106.93276916780667,\n
\"num_unique_values\": 50,\n        \"samples\": [\n
81.08695112697632,\n          97.28603584072259,\n
88.73787971748045\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


7. Top 50 Models

1. **Select Top 50**
   - Sort `results_df` by `test_mae`
   - Keep best 50 models
2. **Create Figure**
   - `plt.figure(figsize=(25, 8))` → wide chart for readability

3. **Plot Lines**
   - Plot `train_mae` with markers
   - Plot `test_mae` with markers
4. **Customize**
   - Rotate x-axis labels (75°) for clarity
   - Add labels (x, y), title, legend, and grid
   - `tight_layout()` → avoid overlap
5. **Show Chart**
   - `plt.show()` → display line chart comparing Train vs Test MAE

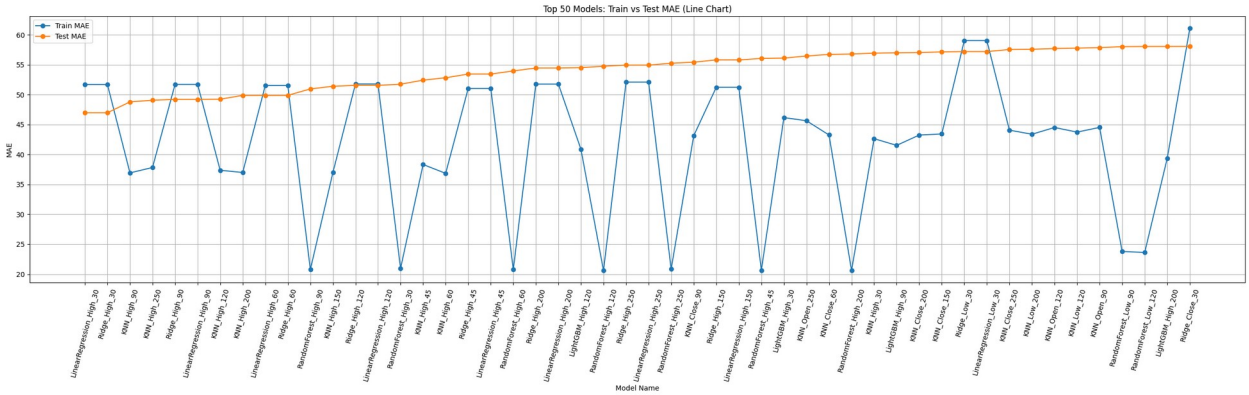
```
import matplotlib.pyplot as plt


top_50 = results_df.sort_values(by='test_mae',
ascending=True).head(50)

plt.figure(figsize=(25, 8))
plt.plot(top_50['Model'], top_50['train_mae'], marker='o',
label='Train MAE')

plt.plot(top_50['Model'], top_50['test_mae'], marker='o', label='Test
MAE')

plt.xticks(rotation=75)
plt.xlabel('Model Name')
plt.ylabel('MAE')
plt.title('Top 50 Models: Train vs Test MAE (Line Chart)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



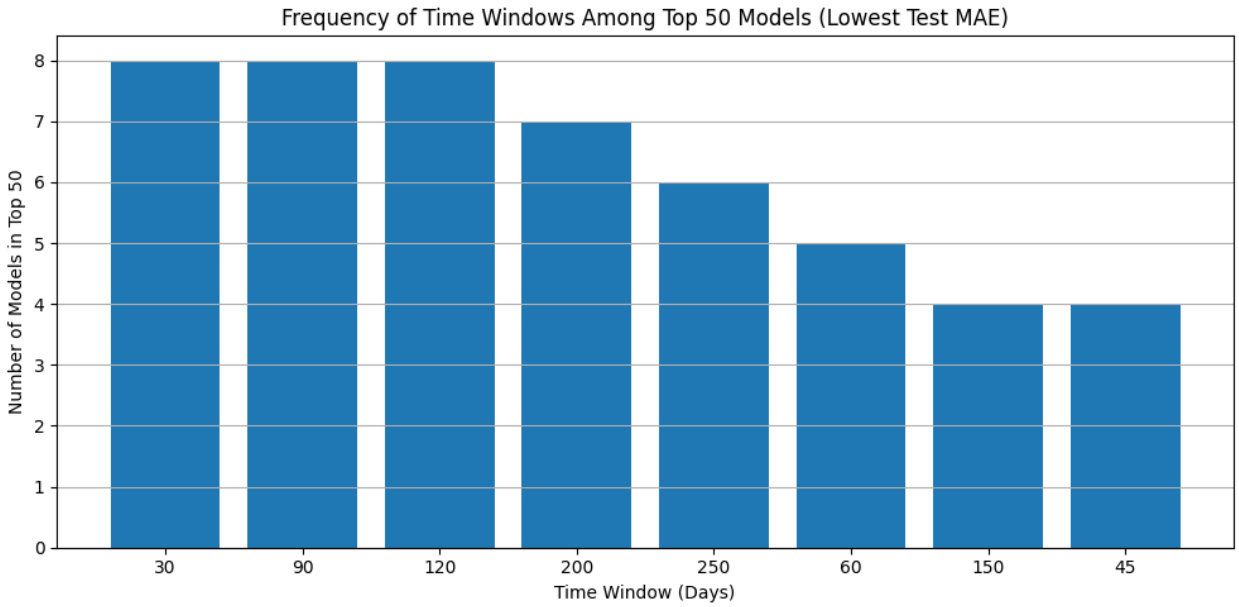
8. Relation btw. No of Input Days and Model Performance

1. **Extract Time Windows**
   - From each model name in `top_50`
   - Split string by `_` → take last part (time window)
2. **Count Frequencies**
   - `value_counts()` → count models per time window
   - Sort by count (descending)
3. **Plot Bar Chart**
   - X-axis: time windows
   - Y-axis: number of models in Top 50
4. **Customize**
   - Add labels (x, y), title
   - Grid only on Y-axis for readability
   - `tight_layout()` → clean layout
5. **Show Chart**
   - `plt.show()` → display bar chart of time-window frequencies

```
top_50 = results_df.sort_values(by='test_mae',
ascending=True).head(50)
time_windows = pd.Series([i.split('_')[-1] for i in top_50['Model']])
time_counts = time_windows.value_counts().sort_values(ascending=False)
# Sort by count

# Plotting
plt.figure(figsize=(10, 5))
plt.bar(time_counts.index, time_counts.values)

# Labels and aesthetics
plt.xlabel('Time Window (Days)')
plt.ylabel('Number of Models in Top 50')
plt.title('Frequency of Time Windows Among Top 50 Models (Lowest Test
MAE)')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



9. Which column(HIGH/LOW/CLOSE/OPEN) should be taken into considration for model building?

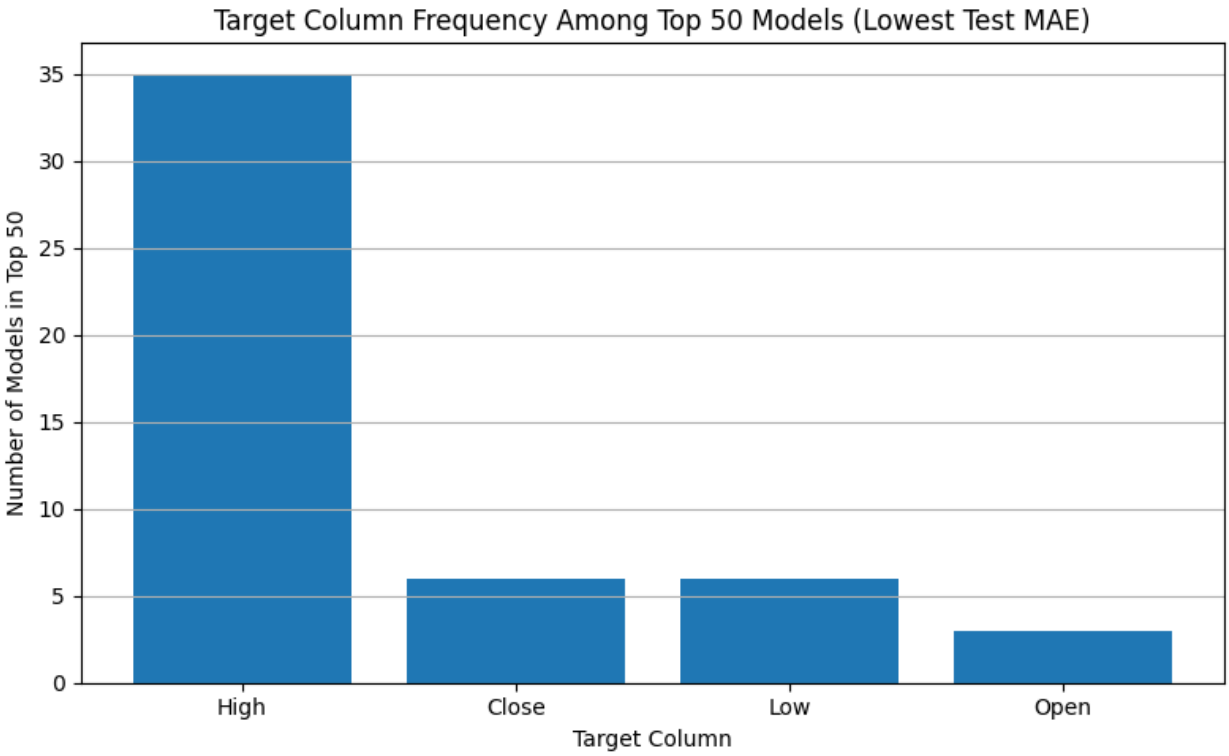
1. **Select Top 50 Models**
   - Sort by `test_mae` → best 50 models
2. **Extract Target Column**
   - From model names → split by `_`
   - Take second last part as target column name
3. **Count Frequencies**
   - `value_counts()` → count occurrences of each target
   - Sort by frequency (descending)
4. **Plot Bar Chart**
   - X-axis: target column names
   - Y-axis: number of models in Top 50
5. **Customize**
   - Add labels, title
   - Grid on Y-axis for readability
   - Use `tight_layout()` to prevent label overlap
6. **Show Chart**
   - `plt.show()` → display bar chart of target column frequencies

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Extract target columns from top 50 models
top_50 = results_df.sort_values(by='test_mae',
ascending=True).head(50)
target_columns = pd.Series([i.split('_')[-2] for i in
top_50['Model']])
target_counts =
target_columns.value_counts().sort_values(ascending=False)  # Sort by
count

# Plotting
plt.figure(figsize=(8, 5))
plt.bar(target_counts.index, target_counts.values)

# Labels and aesthetics
plt.xlabel('Target Column')
plt.ylabel('Number of Models in Top 50')
plt.title('Target Column Frequency Among Top 50 Models (Lowest Test
MAE)')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



10. Which model works in general better on this task?

1. **Select Top 50 Models**

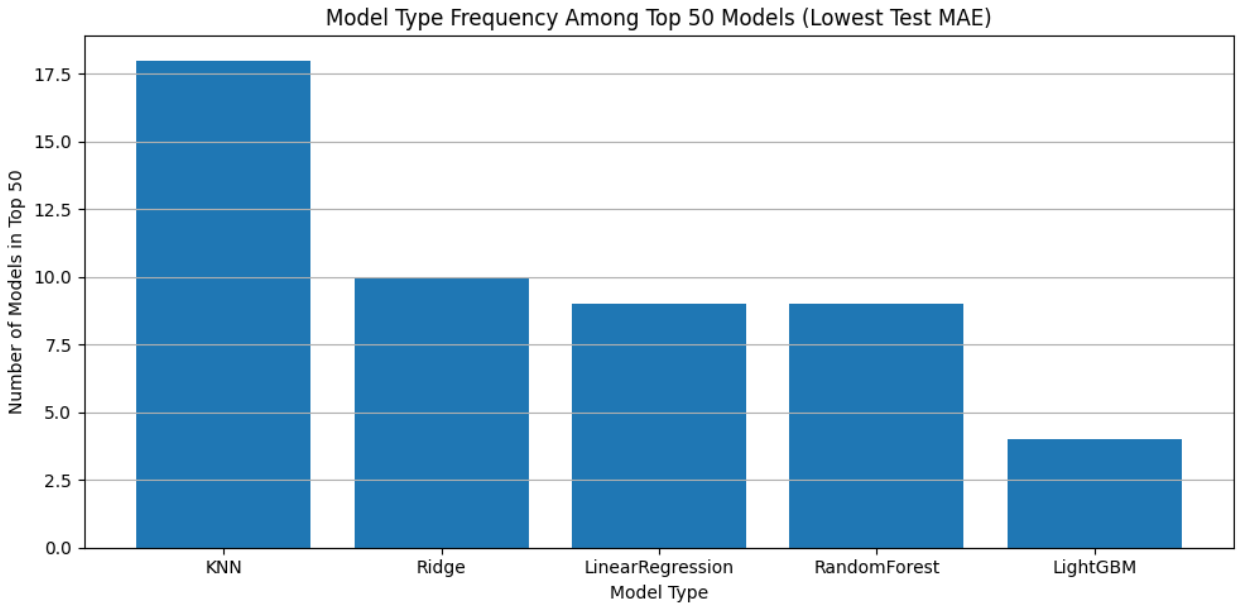
- Sort `results_df` by `test_mae`
- Keep best 50 models

2. **Extract Model Types**
   - From model names → split by `_`
   - Take the first part as model type (e.g., LinearRegression, LSTM)
3. **Count Frequencies**
   - `value_counts()` → count occurrences of each model type
   - Sort counts in descending order
4. **Plot Bar Chart**
   - X-axis: model types
   - Y-axis: number of models in Top 50
5. **Customize**
   - Add axis labels, title
   - Grid only on Y-axis
   - `tight_layout()` for spacing
6. **Show Chart**
   - `plt.show()` → display bar chart of model type distribution

```
top_50 = results_df.sort_values(by='test_mae',
ascending=True).head(50)
model_types = pd.Series([i.split('_')[0] for i in top_50['Model']])
model_counts = model_types.value_counts().sort_values(ascending=False)

# Plotting
plt.figure(figsize=(10, 5))
plt.bar(model_counts.index, model_counts.values)

# Labels and aesthetics
plt.xlabel('Model Type')
plt.ylabel('Number of Models in Top 50')
plt.title('Model Type Frequency Among Top 50 Models (Lowest Test
MAE)')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



11. Saving Models

1. **Save Results Table**
   - `results_df.to_csv('models.csv')` → save metrics as CSV
2. **Save Trained Models**
   - `joblib.dump(trained_models, 'trained_models.joblib')`
   - Stores all fitted ML + DL models + their metrics
3. **Load Models**
   - `loaded_models = joblib.load('trained_models.joblib')`
   - Reload models/metrics into memory for reuse

```
import joblib

results_df.to_csv('models.csv')
joblib.dump(trained_models, 'trained_models.joblib')

loaded_models = joblib.load('trained_models.joblib')
```

12. Loading Saved Models

1. **Access Specific Model**
   - `loaded_models['KNN_High_90']`
   - Retrieves dictionary with:
     - Trained model object
     - Train/Test MAE & RMSE metrics
2. **Extract Model**

- `model = loaded_models['KNN_High_90']['model']`
- Assigns the trained KNN regressor to `model` variable
- Now can be used for `.predict()` on new data

```
loaded_models['KNN_High_90']
model = loaded_models['KNN_High_90']['model']

{'model': KNeighborsRegressor(),
 'train_mae': 36.929139593002496,
 'train_rmse': np.float64(58.769966157340505),
 'test_mae': 48.79014446227931,
 'test_rmse': np.float64(74.20484820380837)}
```

13. Model Inference

1. **Inspect Input Sample**
   - `print(chunked_data['X_Open_90'][5])`
   - Displays the 6th sample from feature set `X_Open_90`
2. **Make Prediction**
   - `model.predict([chunked_data['X_Open_90'][5]])`
   - Wrap sample in a list → ensures 2D shape `(1, n_features)`
   - Outputs predicted value for that input using trained KNN model

```
print(chunked_data['X_Open_90'][5])
print(model.predict([chunked_data['X_Open_90'][5]]))
```

- High
- KNN,RNN,GRU,LSTM,Bidirectional (50 Epochs)
- 30,60,90

Assignment Documentation

Based on the analysis performed in this notebook, the assignment is to focus on building and evaluating models for predicting the **High** price of the NIFTY 50 index.

Specifically, you should concentrate on the following models and time windows:

- **Models:**
  - KNN (K-Nearest Neighbors Regressor)
  - RNN (Simple Recurrent Neural Network)
  - GRU (Gated Recurrent Unit)
  - LSTM (Long Short-Term Memory)
  - Bidirectional LSTM
- **Time Windows (Input Days):**

- 30 days
- 60 days
- 90 days

For the Deep Learning models (RNN, GRU, LSTM, Bidirectional LSTM), train them for **50 epochs**.

The goal is to train these specific models for the 'High' column using the specified time windows and evaluate their performance using MAE and RMSE, comparing the results.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np
import pandas as pd
from tqdm.auto import tqdm
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from copy import deepcopy


from sklearn.metrics import mean_absolute_error, mean_squared_error

import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv('/content/drive/MyDrive/21 DAYS 21
PROJECTS/data.csv')
df.head()
```

```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 6315,\n  \"fields\":
[\n    {\n      \"column\": \"Date\",\n      \"properties\": {\n
\"dtype\": \"object\",\n        \"num_unique_values\": 6315,\n
\"samples\": [\n          \"2004-11-17\",\n          \"2015-07-22\",\n
\"2019-08-07\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Open\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 6248.404052765841,\n        \"min\": 853.0,\n        \"max\":
26248.25,\n        \"num_unique_values\": 6209,\n        \"samples\":
[\n          7897.4,\n          957.55,\n          10738.45\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"High\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 6270.434419505976,\n        \"min\": 877.0,\n        \"max\":
26277.35,\n        \"num_unique_values\": 6226,\n        \"samples\":
[\n          1349.25,\n          1610.6,\n          2842.9\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Low\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 6216.988573625996,\n        \"min\": 849.95,\n
\"max\": 26151.4,\n        \"num_unique_values\": 6207,\n
\"samples\": [\n          8408.3,\n          8517.2,\n
4392.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Close\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 6244.654056542602,\n        \"min\": 854.2,\n        \"max\":
26216.05,\n        \"num_unique_values\": 6127,\n        \"samples\":
[\n          2419.0,\n          19389.0,\n          10480.6\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


Data Loading and Filtering

```python
def return_pairs(column, days):
    prices = list(column)
    X = []
    y = []
    for i in range(len(prices) - days):
        X.append(prices[i:i+days])
        y.append(prices[i+days])
    return np.array(X), np.array(y)

target_columns = ['Open', 'Close', 'High', 'Low']
day_chunks = [30, 45, 60, 90, 120, 150, 200, 250]
chunked_data = {}

for col in target_columns:
    for days in day_chunks:
        key_X = f"X_{col}_{days}"
        key_y = f"y_{col}_{days}"
        X, y = return_pairs(df[col], days)
        chunked_data[key_X] = X
        chunked_data[key_y] = y

all_chunk_pairs = []
for key in chunked_data.keys():
    if key.startswith("X_"):
        y_key = key.replace("X_", "y_")
        if y_key in chunked_data:
            all_chunk_pairs.append([key, y_key])
```

```
# filter down to only the data asked
assignment_target = 'High'
assignment_windows = [30, 60, 90]
assignment_chunk_pairs = []

for X_key, y_key in all_chunk_pairs:
    _, target, window = X_key.split('_')
    if target == assignment_target and int(window) in
assignment_windows:
        assignment_chunk_pairs.append([X_key, y_key])

print("Filtered Data Pairs for the Assignment:")
print(assignment_chunk_pairs)
```

```
Filtered Data Pairs for the Assignment:
[['X_High_30', 'y_High_30'], ['X_High_60', 'y_High_60'], ['X_High_90',
'y_High_90']]
```

Defining the Models

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, LSTM, GRU,
Bidirectional
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.neighbors import KNeighborsRegressor

def build_rnn(input_shape):
    model = Sequential([
        SimpleRNN(50, activation='tanh', input_shape=input_shape),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

def build_lstm(input_shape):
    model = Sequential([
        LSTM(50, activation='tanh', input_shape=input_shape),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

def build_gru(input_shape):
    model = Sequential([
        GRU(50, activation='tanh', input_shape=input_shape),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model
```

```
def build_bilstm(input_shape):
    model = Sequential([
        Bidirectional(LSTM(50, activation='tanh'),
input_shape=input_shape),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

# ML model
assignment_ml_models = [
    ("KNN", KNeighborsRegressor())
]

# DL models
assignment_dl_models = {
    "RNN": build_rnn,
    "LSTM": build_lstm,
    "GRU": build_gru,
    "Bidirectional_LSTM": build_bilstm
}

print("ML Models for Assignment:")
print([name for name, model in assignment_ml_models])
print("\nDL Models for Assignment:")
print(list(assignment_dl_models.keys()))
```

```
ML Models for Assignment:
['KNN']

DL Models for Assignment:
['RNN', 'LSTM', 'GRU', 'Bidirectional_LSTM']
```

Train the Models

```
results = {}

for X_key, y_key in tqdm(assignment_chunk_pairs, desc="Processing Time
Windows"):

    # Get the data for the current pair
    X_data = chunked_data[X_key]
    y_data = chunked_data[y_key]

    X_train, X_test, y_train, y_test = train_test_split(
        X_data, y_data, test_size=0.1, random_state=42
    )

    # KNN
```

```
    for model_name, model in tqdm(assignment_ml_models, desc=f"ML
Models for {X_key}"):
        key = f"{model_name}_{X_key[2:]}"
        model_copy = deepcopy(model)
        model_copy.fit(X_train, y_train)

        y_train_pred = model_copy.predict(X_train)
        y_test_pred = model_copy.predict(X_test)

        results[key] = {
            'model_type': 'ML',
            'train_mae': mean_absolute_error(y_train, y_train_pred),
            'train_rmse': np.sqrt(mean_squared_error(y_train,
y_train_pred)),
            'test_mae': mean_absolute_error(y_test, y_test_pred),
            'test_rmse': np.sqrt(mean_squared_error(y_test,
y_test_pred))
        }

    # DL Models
    X_train_dl = np.expand_dims(X_train, -1)
    X_test_dl = np.expand_dims(X_test, -1)

    for model_name, builder in tqdm(assignment_dl_models.items(),
desc=f"DL Models for {X_key}"):
        key = f"{model_name}_{X_key[2:]}"
        model_dl = builder((X_train_dl.shape[1], 1))

        # Train for 50 epochs
        model_dl.fit(X_train_dl, y_train, epochs=50, batch_size=8,
verbose=0)

        y_train_pred_dl = model_dl.predict(X_train_dl).flatten()
        y_test_pred_dl = model_dl.predict(X_test_dl).flatten()

        results[key] = {
            'model_type': 'DL',
            'train_mae': mean_absolute_error(y_train,
y_train_pred_dl),
            'train_rmse': np.sqrt(mean_squared_error(y_train,
y_train_pred_dl)),
            'test_mae': mean_absolute_error(y_test, y_test_pred_dl),
            'test_rmse': np.sqrt(mean_squared_error(y_test,
y_test_pred_dl))
        }

print("\nTraining Complete!")
```

```
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step
```

```
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
177/177 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step

176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step

Training Complete!
```

Analyze and Visualize Results

```
results_df = pd.DataFrame.from_dict(results, orient='index')

results_df['window'] = [int(i.split('_')[-1]) for i in
results_df.index]
results_df['model_name'] = [i.split('_')[0] for i in results_df.index]

print("Model Performance Results")
display(results_df.sort_values(by='test_mae', ascending=True))
```

```
Model Performance Results
```

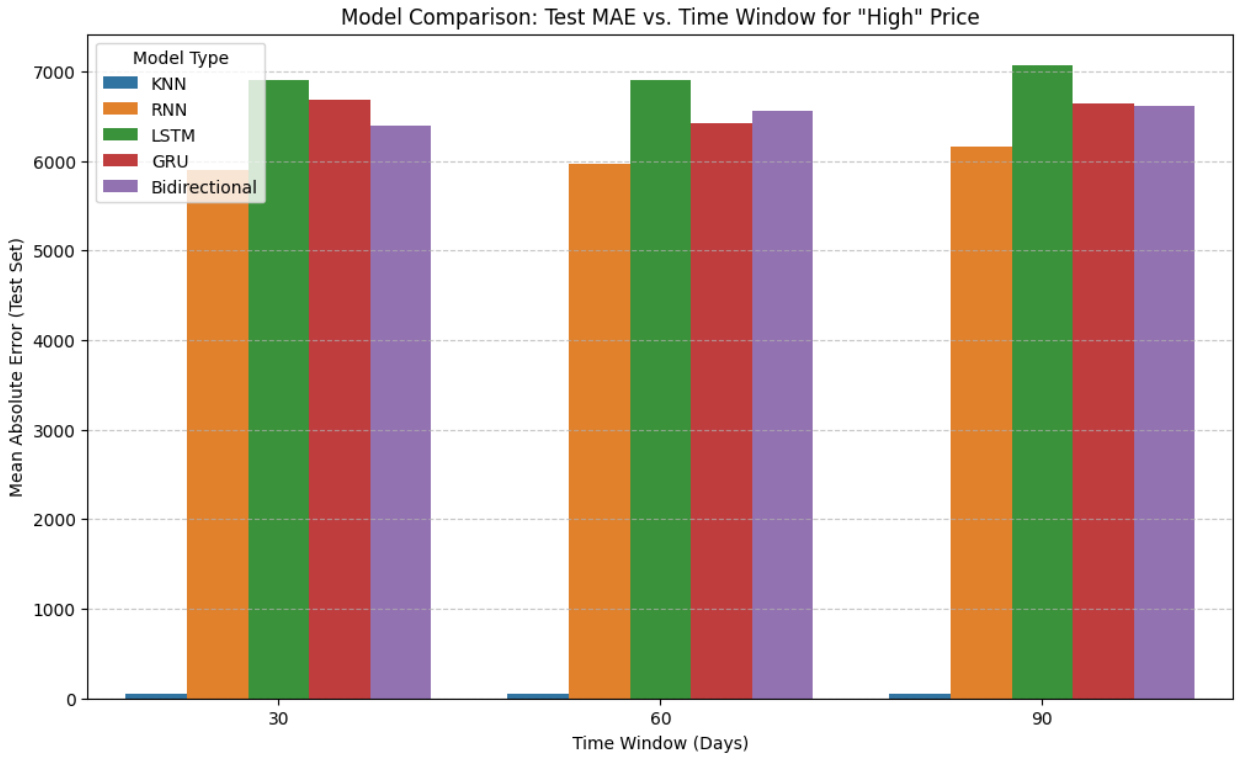

```
{"summary":"{\n  \"name\": \"display(results_df\",\n  \"rows\": 15,\n
\"fields\": [\n    {\n      \"column\": \"model_type\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"DL\",\n
\"ML\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"train_mae\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2877.5868298635396,\n        \"min\":
36.85089891632618,\n        \"max\": 7423.226930213075,\n
\"num_unique_values\": 15,\n        \"samples\": [\n
```

```
6905.548471824375,\n          7196.38195950995\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"train_rmse\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3850.594157846032,\n        \"min\": 58.3335888587827,\n
\"max\": 9719.05700158275,\n        \"num_unique_values\": 15,\n
\"samples\": [\n          9328.34237894512,\n
9546.920144723865\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"test_mae\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 2695.8338317171524,\n        \"min\":
48.79014446227931,\n        \"max\": 7061.515797446361,\n
\"num_unique_values\": 15,\n        \"samples\": [\n
6607.562093033645,\n          6677.196778178253\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"test_rmse\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3676.02079877209,\n        \"min\": 74.20484820380837,\n
\"max\": 9382.311672985554,\n        \"num_unique_values\": 15,\n
\"samples\": [\n          9038.844981305918,\n
9126.201907795172\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"window\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 25,\n        \"min\": 30,\n        \"max\": 90,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          90,\n
60\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"model_name\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          \"RNN\",\n          \"LSTM\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 7))
sns.barplot(x='window', y='test_mae', hue='model_name',
data=results_df)

plt.title('Model Comparison: Test MAE vs. Time Window for "High"
Price')
plt.xlabel('Time Window (Days)')
plt.ylabel('Mean Absolute Error (Test Set)')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend(title='Model Type')
plt.show()
```



Conclusion on the Model Comparison:

- **Best Overall Model**: The K-Nearest Neighbors (KNN) model is by far the most effective for this prediction task. Its Mean Absolute Error (MAE) is significantly lower than all the deep learning models across every time window, to the point that its bars are barely visible on the chart.
- **Deep Learning Model Comparison**: Among the deep learning models, there is a clear performance ranking:
  - The standard **RNN** model consistently achieved the lowest MAE, making it the best performer in this category. Its best result was with a 60-day time window.
  - **GRU** and **Bidirectional LSTM** models had similar, moderate performance, generally performing better than the LSTM model but worse than the RNN.
  - The **LSTM** model was the worst-performing model across all three time windows.
- **Effect of Time Window**:
  - The performance of the KNN model was consistently excellent across all tested time windows (30, 60, and 90 days).
  - For the deep learning models, the choice of time window had a noticeable but not drastic effect. The RNN model, for instance, saw its best performance with a 60-day window.

In summary, the key takeaway is that the simpler KNN algorithm dramatically outperformed the more complex deep learning architectures for predicting the "High" price on this dataset.